同济大学计算机系

计算机组成原理实验报告

学　　号	2151140
姓　　名	王谦
专　　业	信息安全
授课老师	张冬冬

一、实验内容

54 条指令 CPU 设计

实验介绍：
在本次实验中，我们将使用 Verilog HDL 语言实现 54 条 MIPS 指令的 CPU 的设计和仿真，设计的 CPU 可以是单周期的，也可以是多周期。

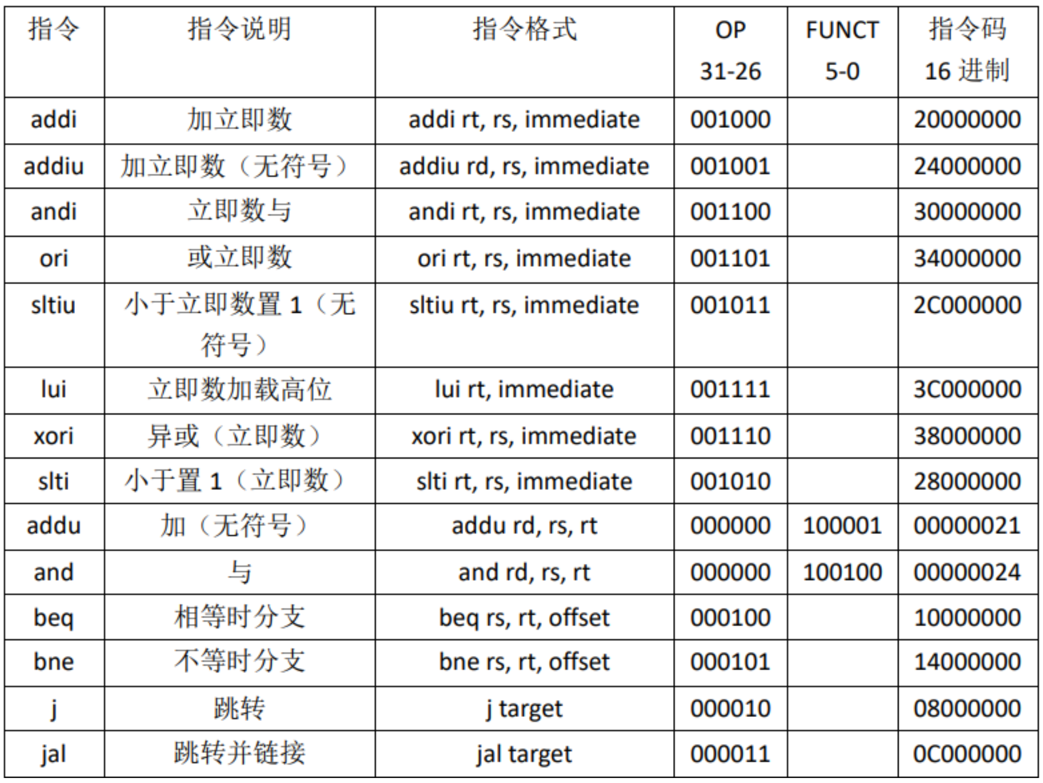
实验目标：
（1） 深入了解 CPU 的原理。
（2） 画出实现 54 条指令的 CPU 的通路图。
（3） 学习使用 Verilog HDL 语言设计实现 54 条指令的 CPU。

实验原理：

（*注意：为了便于使用 Mars 进行测试，CPU 设计做如下规定： Mars 中 CPU 采用冯诺依曼结构，我们使用它的 default 内存设置，故指令存储起始地址为： 0x00400000，数据存储起始地址为 0x10010000，由于我们设计的 CPU 为哈佛结构，指令和 数据的起始地址都为 0，所以最后请在 PC 模块、指令存储模块、数据存储模块进行地址映 射（将逻辑地址转换为物理地址 0））

1） 需要实现的 54 条 MIPS 指令，见图

表 8.6.1 54 条 MIPS 指令集

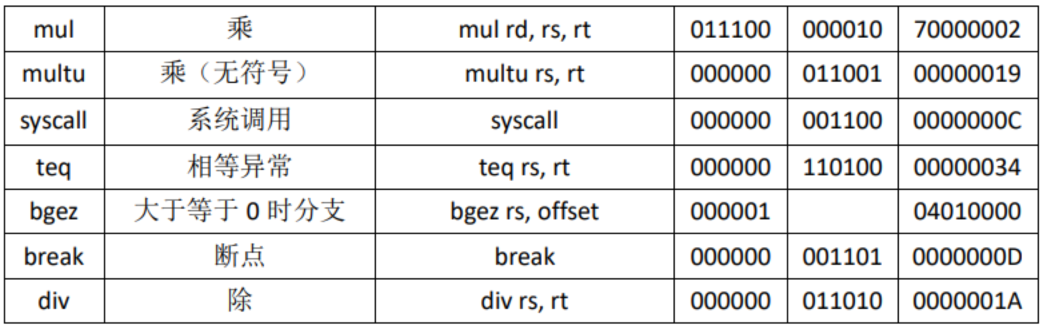


指令	指令说明	指令格式	OP 31-26	FUNCT 5-0	指令码 16 进制
addi	加立即数	addi rt, rs, immediate	001000		20000000
addiu	加立即数（无符号）	addiu rd, rs, immediate	001001		24000000
andi	立即数与	andi rt, rs, immediate	001100		30000000
ori	或立即数	ori rt, rs, immediate	001101		34000000
sltiu	小于立即数置 1（无符号）	sltiu rt, rs, immediate	001011		2C000000
lui	立即数加载高位	lui rt, immediate	001111		3C000000
xori	异或（立即数）	xori rt, rs, immediate	001110		38000000
slti	小于置 1（立即数）	slti rt, rs, immediate	001010		28000000
addu	加（无符号）	addu rd, rs, rt	000000	100001	00000021
and	与	and rd, rs, rt	000000	100100	00000024
beq	相等时分支	beq rs, rt, offset	000100		10000000
bne	不等时分支	bne rs, rt, offset	000101		14000000
j	跳转	j target	000010		08000000
jal	跳转并链接	jal target	000011		0C000000

jr	跳转至寄存器所指地址	jr rs	000000	001000	00000009
lw	取字	lw rt, offset(base)	100011		8C000000
xor	异或	xor rd, rs, rt	000000	100110	00000026
nor	或非	nor rd, rs, rt	000000	100111	00000027
or	或	or rd, rs, rt	000000	100101	00000025

sll	逻辑左移	sll rd, rt, sa	000000	000000	00000000
sllv	逻辑左移（位数可变）	sllv rd, rt, rs	000000	000100	00000004
sltu	小于置 1（无符号）	sltu rd, rs, rt	000000	101011	0000002B
sra	算数右移	sra rd, rt, sa	000000	000011	00000003
srl	逻辑右移	srl rd, rt, sa	000000	000010	00000002
subu	减（无符号）	sub rd, rs, rt	000000	100010	00000022
sw	存字	sw rt, offset(base)	101011		AC000000
add	加	add rd, rs, rt	000000	100000	00000020
sub	减	sub rd, rs, rt	000000	100010	00000022
slt	小于置 1	slt rd, rs, rt	000000	101010	0000002A
srlv	逻辑右移（位数可变）	srlv rd, rt, rs	000000	000110	00000006
srav	算数右移（位数可变）	srav rd, rt, rs	000000	000111	00000007
clz	前导零计数	clz rd, rs	011100	100000	70000020
divu	除（无符号）	divu rs, rt	000000	011011	0000001B
eret	异常返回	eret	010000	011000	42000018
jalr	跳转至寄存器所指地址，返回地址保存在	jalr rs	000000	001001	00000008
lb	取字节	lb rt, offset(base)	100000		80000000
lbu	取字节（无符号）	lbu rt, offset(base)	100100		90000000
lhu	取半字（无符号）	lhu rt, offset(base)	100101		94000000
sb	存字节	sb rt, offset(base)	101000		A0000000
sh	存半字	sh rt, offset(base)	101001		A4000000
lh	取半字	lh rt, offset(base)	100001		84000000

mfc0	读 CP0 寄存器	mfc0 rt, rd	010000	000000	40000000
mfhi	读 Hi 寄存器	mfhi rd	000000	010000	00000010
mflo	读 Lo 寄存器	mflo rd	000000	010010	00000012
mtc0	写 CP0 寄存器	mtc0 rt, rd	010000	000000	40800000
mthi	写 Hi 寄存器	mthi rd	000000	010001	00000011
mtlo	写 Lo 寄存器	mtlo rd	000000	010011	00000013



mul	乘	mul rd, rs, rt	011100	000010	70000002
multu	乘（无符号）	multu rs, rt	000000	011001	00000019
syscall	系统调用	syscall	000000	001100	0000000C
teq	相等异常	teq rs, rt	000000	110100	00000034
bgez	大于等于 0 时分支	bgez rs, offset	000001		04010000
break	断点	break	000000	001101	0000000D
div	除	div rs, rt	000000	011010	0000001A

2） 单周期数据通路设计的一般性方法

（1）　阅读每条指令，对每条指令所需执行的功能与过程都有充分的了解

（2）　确定每条指令在执行过程中所用到的部件

（3）　使用表格列出指令所用部件，并在表格中填入每个部件的数据输入来源

（4）　根据表格所涉及部件和部件的数据输入来源，画出整个数据通路

3） 前仿真

前仿真也称为功能仿真，主旨在于验证电路的功能是否符合设计要求，其特点是不考虑电路门延迟与线延迟，主要是验证电路与理想情况是否一致。可综合 FPGA 代码是用 RTL 级代码语言描述的，其输入为 RTL 级代码与 Testbench.

4） 后仿真

后仿真也称为时序仿真或者布局布线后仿真，是指电路已经映射到特定的工艺环境以后，综合考虑电路的路径延迟与门延迟的影响，验证电路能否在一定时序条件下满足设计构 想的过程，是否存在时序违规。其输入文件为从布局布线结果中抽象出来的门级网表、 Testbench 和扩展名为 SDO 或 SDF 的标准时延文件。SDO 或 SDF 的标准时延文件不仅包 含门延迟，还包括实际布线延迟，能较好地反映芯片的实际工作情况。一般来说后仿真是必选的，检查设计时序与实际的 FPGA 运行情况是否一致，确保设计的可靠性和稳定性。选定了器件分配引脚后在做后仿真。

5） 下板测试

由于我们自行编写的指令 RAM 用来初始化内存的 initial 指令是不可综合的，无法在开发板上运行，所以，我们可以使用 Vivado 提供的 ip 核来替换我们的 ram，其可以使用一个 coe

6） 54 条指令单周期 cpu 相比于 31 条指令单周期 cpu

54 条指令在 31 条的基础上添加了乘除法运算、对 Lo/Hi 寄存器的读写、内存半字和字节的存取操作、CP0 的异常处理指令和 CP0 寄存器的读写，以及一些跳转指令。

乘、除法器和 CP0 模块在前面的部分已经有所介绍，在 CPU 中仅需处理相应指令的控制信号和输入输出引脚，剩余的主要是添加对内存块的读写控制。在 CPU 通路中需要加入乘、除法器模块和 CP0 模块。

指令的测试和 31 条指令 CPU 的测试方法类似，对 CP0 模块的测试需要自行编写测试用例，要对 CP0 寄存器的读写功能、异常发生时的跳转功能和异常返回等环节进行测试，主要验证几个关键寄存器值的正确写入和控制信号的判断处理。实验中异常的入口地址为 0x4，可在入口处添加跳转指令，跳入统一的异常处理程序，再判断异常号然

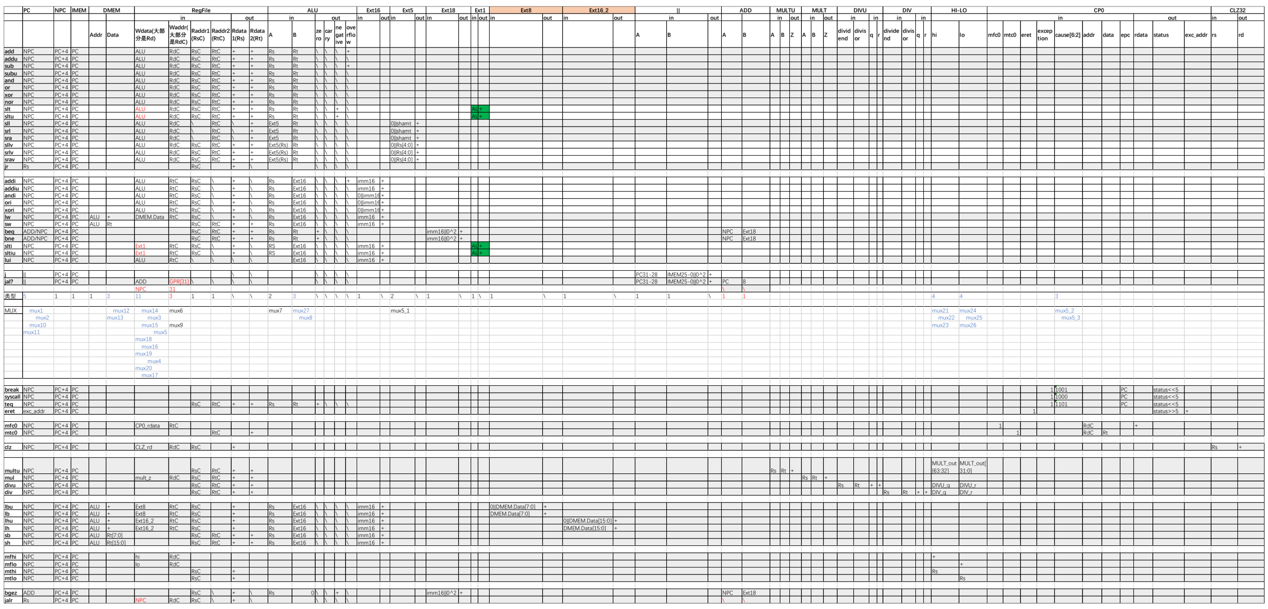
后进入相应的处理入口。

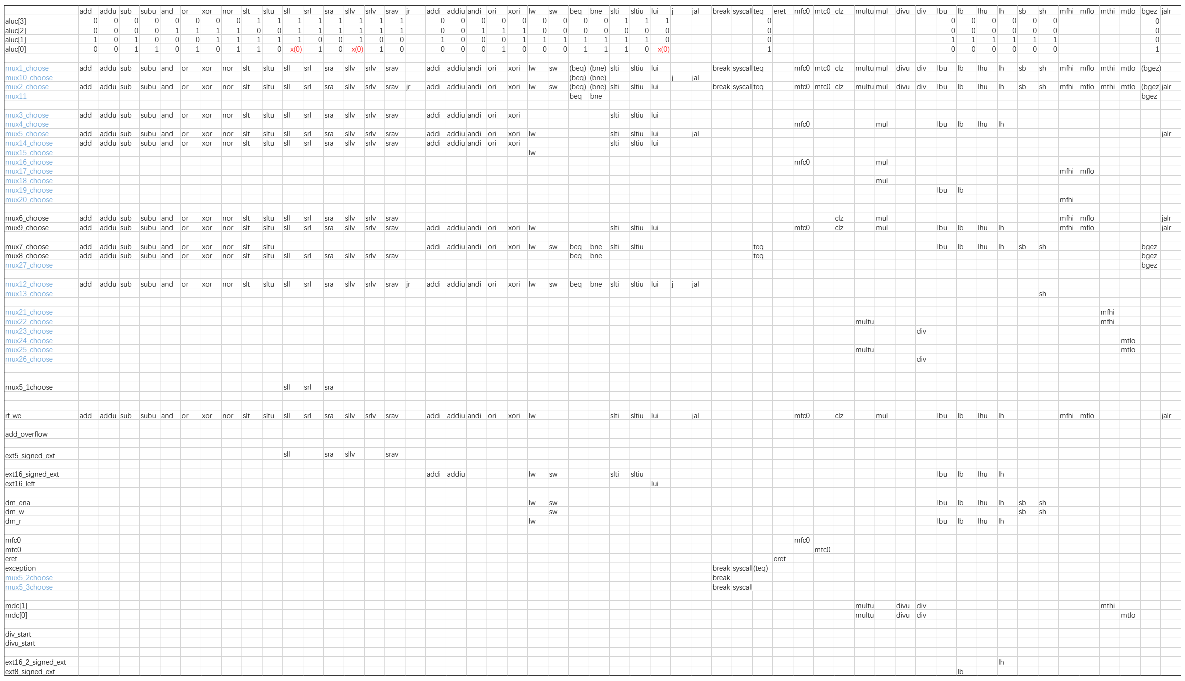
实验步骤：

1.新建 Vivado 工程，在 31 条指令的基础上编写各个模块。
2.用 ModelSim 前仿真逐条测试所有指令。
3.用 ModelSim 进行后仿真测试。
4.配置 XDC 文件，综合下板，并观察实验现象。
5.按照要求书写实验报告

二、硬件逻辑图

缩略图：





	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr	addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu	lui	j	jal	break	syscall	teq	eret	mfc0	mtc0	clz	multu	mul	divu	div	lbu	lb	lhu	lh	sb	sh	mfhi	mflo	mthi	mtlo	bgez	jalr
aluc[3]	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1		0	0	0	0	0	0	0	0	0	1	1	1					0									0	0	0	0	0	0					0	
aluc[2]	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1		0	0	1	1	1	0	0	0	0	0	0	0					0									0	0	0	0	0	0					0	
aluc[1]	1	0	1	0	0	0	1	1	1	1	1	0	0	1	0	0		1	0	0	0	1	1	1	1	1	1	1	0					0									1	1	1	1	1	1					0	
aluc[0]	0	0	1	1	0	1	0	1	1	0	x(0)	1	0	x(0)	1	0		0	0	0	1	0	0	0	1	1	1	0	x(0)					1									0	0	0	0	0	0					1	
mux1_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav		addi	addiu	andi	ori	xori	lw	sw	(beq)	(bne)	slti	sltiu	lui			break	syscall	teq		mfc0	mtc0	clz	multu	mul	divu	div	lbu	lb	lhu	lh	sb	sh	mfhi	mflo	mthi	mtlo	(bgez)	
mux10_choose																									(beq)	(bne)				j	jal																							
mux2_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr	addi	addiu	andi	ori	xori	lw	sw	(beq)	(bne)	slti	sltiu	lui			break	syscall	teq		mfc0	mtc0	clz	multu	mul	divu	div	lbu	lb	lhu	lh	sb	sh	mfhi	mflo	mthi	mtlo	(bgez)	jalr
mux11																									beq	bne																											bgez	
mux3_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav		addi	addiu	andi	ori	xori					slti	sltiu	lui																									
mux4_choose																																				mfc0				mul			lbu	lb	lhu	lh								
mux5_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav		addi	addiu	andi	ori	xori	lw				slti	sltiu	lui		jal																							jalr
mux14_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav		addi	addiu	andi	ori	xori					slti	sltiu	lui																									
mux15_choose																							lw																															
mux16_choose																																				mfc0				mul														
mux17_choose																																																	mfhi	mflo				
mux18_choose																																								mul														
mux19_choose																																											lbu	lb										
mux20_choose																																																	mfhi					
mux6_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav																						clz		mul									mfhi	mflo				jalr
mux9_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav		addi	addiu	andi	ori	xori	lw				slti	sltiu	lui							mfc0		clz		mul			lbu	lb	lhu	lh			mfhi	mflo				jalr
mux7_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu								addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu						teq									lbu	lb	lhu	lh	sb	sh					bgez	
mux8_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav									beq	bne								teq																			bgez	
mux27_choose																																																					bgez	
mux12_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr	addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu	lui	j	jal																							
mux13_choose																																																sh						
mux21_choose																																																			mfhi			
mux22_choose																																							multu												mfhi			
mux23_choose																																										div												
mux24_choose																																																				mtlo		
mux25_choose																																							multu													mtlo		
mux26_choose																																										div												
mux5_1choose											sll	srl	sra																																									
rf_we	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav		addi	addiu	andi	ori	xori	lw				slti	sltiu	lui		jal					mfc0		clz		mul			lbu	lb	lhu	lh			mfhi	mflo				jalr
add_overflow																																																						
ext5_signed_ext											sll		sra	sllv		srav																																						
ext16_signed_ext																		addi	addiu				lw	sw			slti	sltiu															lbu	lb	lhu	lh								
ext16_left																													lui																									
dm_ena																							lw	sw																			lbu	lb	lhu	lh	sb	sh						
dm_w																								sw																							sb	sh						
dm_r																							lw																				lbu	lb	lhu	lh								
mfc0																																				mfc0																		
mtc0																																					mtc0																	
eret																																			eret																			
exception																																break	syscall(teq)																					
mux5_2choose																																break																						
mux5_3choose																																break	syscall																					
mdc[1]																																							multu		divu	div									mthi			
mdc[0]																																							multu		divu	div										mtlo		
div_start																																																						
divu_start																																																						
ext16_2_signed_ext																																														lh								
ext8_signed_ext																																												lb										

展开图：

	PC	NPC	IMEM	DMEM		RegFile						ALU						Ext16		Ext5		Ext18		Ext1	
						in				out		in		out				in	out	in	out	in	out	in	out
				Addr	Data	Wdata(大部分是Rd)	Waddr(大部分是RdC)	Raddr1 (RsC)	Raddr2 (RtC)	Rdata 1(Rs)	Rdata 2(Rt)	A	B	zero	carry	negative	overflow								
add	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	+								
addu	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\								
sub	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	+								
subu	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\								
and	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\								
or	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\								
xor	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\								
nor	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\								
slt	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	+	\							AL	+
sltu	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	+	\							AL	+
sll	NPC	PC+4	PC			ALU	RdC	\	RtC	\	+	Ext5	Rt	\	\	\	\			0||shamt	+				
srl	NPC	PC+4	PC			ALU	RdC	\	RtC	\	+	Ext5	Rt	\	\	\	\			0||shamt	+				
sra	NPC	PC+4	PC			ALU	RdC	\	RtC	\	+	Ext5	Rt	\	\	\	\			0||shamt	+				
sllv	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Ext5(Rs)	Rt	\	\	\	\			0||Rs[4:0]	+				
srlv	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Ext5(Rs)	Rt	\	\	\	\			0||Rs[4:0]	+				
srav	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Ext5(Rs)	Rt	\	\	\	\			0||Rs[4:0]	+				
jr	Rs	PC+4	PC					RsC		+	\			\	\	\	\								
addi	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	+	imm16	+						
addiu	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	imm16	+						
andi	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	0||imm16	+						
ori	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	0||imm16	+						
xori	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	0||imm16	+						
lw	NPC	PC+4	PC	ALU	+	DMEM.Data	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	imm16	+						
sw	NPC	PC+4	PC	ALU	Rt			RsC	RtC	+	+	Rs	Ext16	\	\	\	\	imm16	+						
beq	ADD/NPC	PC+4	PC					RsC	RtC	+	+	Rs	Rt	+	\	\	\					imm16||0^2	+		
bne	ADD/NPC	PC+4	PC					RsC	RtC	+	+	Rs	Rt	+	\	\	\					imm16||0^2	+		
slti	NPC	PC+4	PC			Ext1	RtC	RsC	\	+	\	RS	Ext16	\	\	\	\	imm16	+					AL	+
sltiu	NPC	PC+4	PC			Ext1	RtC	RsC	\	+	\	RS	Ext16	\	\	\	\	imm16	+					AL	+
lui	NPC	PC+4	PC			ALU	RtC	\	\	\	\		Ext16	\	\	\	\	imm16	+						
j	||	PC+4	PC							\	\			\	\	\	\								
jal?	||	PC+4	PC			ADD	GPR[31]	\	\	\	\			\	\	\	\								
						NPC	31																		
类型	5	1	1	1	3	11	3	1	1	\	\	2	3	\	\	\	\	1	\	2	\	1	\	1	\
MUX	mux1				mux12	mux14	mux6					mux7	mux27							mux5_1					
	mux2				mux13	mux3							mux8												
	mux10					mux15	mux9																		
	mux11					mux5																			
						mux18																			
						mux16																			
						mux19																			
						mux4																			
						mux20																			
						mux17																			
break	NPC	PC+4	PC																						
syscall	NPC	PC+4	PC																						
teq	NPC	PC+4	PC					RsC	RtC	+	+	Rs	Rt	+	\	\	\								
eret	exc_addr	PC+4	PC																						
mfc0	NPC	PC+4	PC			CP0_rdata	RtC																		
mtc0	NPC	PC+4	PC						RtC		+														
clz	NPC	PC+4	PC			CLZ_rd	RdC	RsC		+															
multu	NPC	PC+4	PC					RsC	RtC	+	+														
mul	NPC	PC+4	PC			mult_z	RdC	RsC	RtC	+	+														
divu	NPC	PC+4	PC					RsC	RtC	+	+														
div	NPC	PC+4	PC					RsC	RtC	+	+														
lbu	NPC	PC+4	PC	ALU	+	Ext8	RtC	RsC		+		Rs	Ext16	\	\	\	\	imm16	+						
lb	NPC	PC+4	PC	ALU	+	Ext8	RtC	RsC		+		Rs	Ext16	\	\	\	\	imm16	+						
lhu	NPC	PC+4	PC	ALU	+	Ext16_2	RtC	RsC		+		Rs	Ext16	\	\	\	\	imm16	+						
lh	NPC	PC+4	PC	ALU	+	Ext16_2	RtC	RsC		+		Rs	Ext16	\	\	\	\	imm16	+						
sb	NPC	PC+4	PC	ALU	Rt[7:0]			RsC	RtC	+	+	Rs	Ext16	\	\	\	\	imm16	+						
sh	NPC	PC+4	PC	ALU	Rt[15:0]			RsC	RtC	+	+	Rs	Ext16	\	\	\	\	imm16	+						
mfhi	NPC	PC+4	PC			hi	RdC																		
mflo	NPC	PC+4	PC			lo	RdC																		
mthi	NPC	PC+4	PC					RsC		+															
mtlo	NPC	PC+4	PC					RsC		+															
bgez	ADD	PC+4	PC					RsC	\	+	\	Rs	0	\	\	+	\					imm16||0^2	+		
jalr	Rs	PC+4	PC			NPC	RdC	RsC	\	+	\			\	\	\	\								

	Ext8		Ext16_2		||			ADD		MULTU			MULT			DIVU				DIV			
	in	out	in	out	in		out			in		out	in		out	in		in		in		in	
					A	B		A	B	A	B	Z	A	B	Z	dividend	divisor	q	r	dividend	divisor	q	r
add																							
addu																							
sub																							
subu																							
and																							
or																							
xor																							
nor																							
slt																							
sltu																							
sll																							
srl																							
sra																							
sllv																							
srlv																							
srav																							
jr																							
addi																							
addiu																							
andi																							
ori																							
xori																							
lw																							
sw																							
beq								NPC	Ext18														
bne								NPC	Ext18														
slti																							
sltiu																							
lui																							
j					PC31-28	IMEM25-0||0^2	+																
jal?					PC31-28	IMEM25-0||0^2	+	PC	8														
								\	\														
类型	1	\	1	\	1	1	\	1	1														
MUX																							
break																							
syscall																							
teq																							
eret																							
mfc0																							
mtc0																							
clz																							
multu										Rs	Rt	+											
mul													Rs	Rt	+								
divu																Rs	Rt	+	+				
div																				Rs	Rt	+	+
lbu	0||DMEM.Data[7:0]	+																					
lb	DMEM.Data[7:0]	+																					
lhu			0||DMEM.Data[15:0]	+																			
lh			DMEM.Data[15:0]	+																			
sb																							
sh																							
mfhi																							
mflo																							
mthi																							
mtlo																							
bgez								NPC	Ext18														

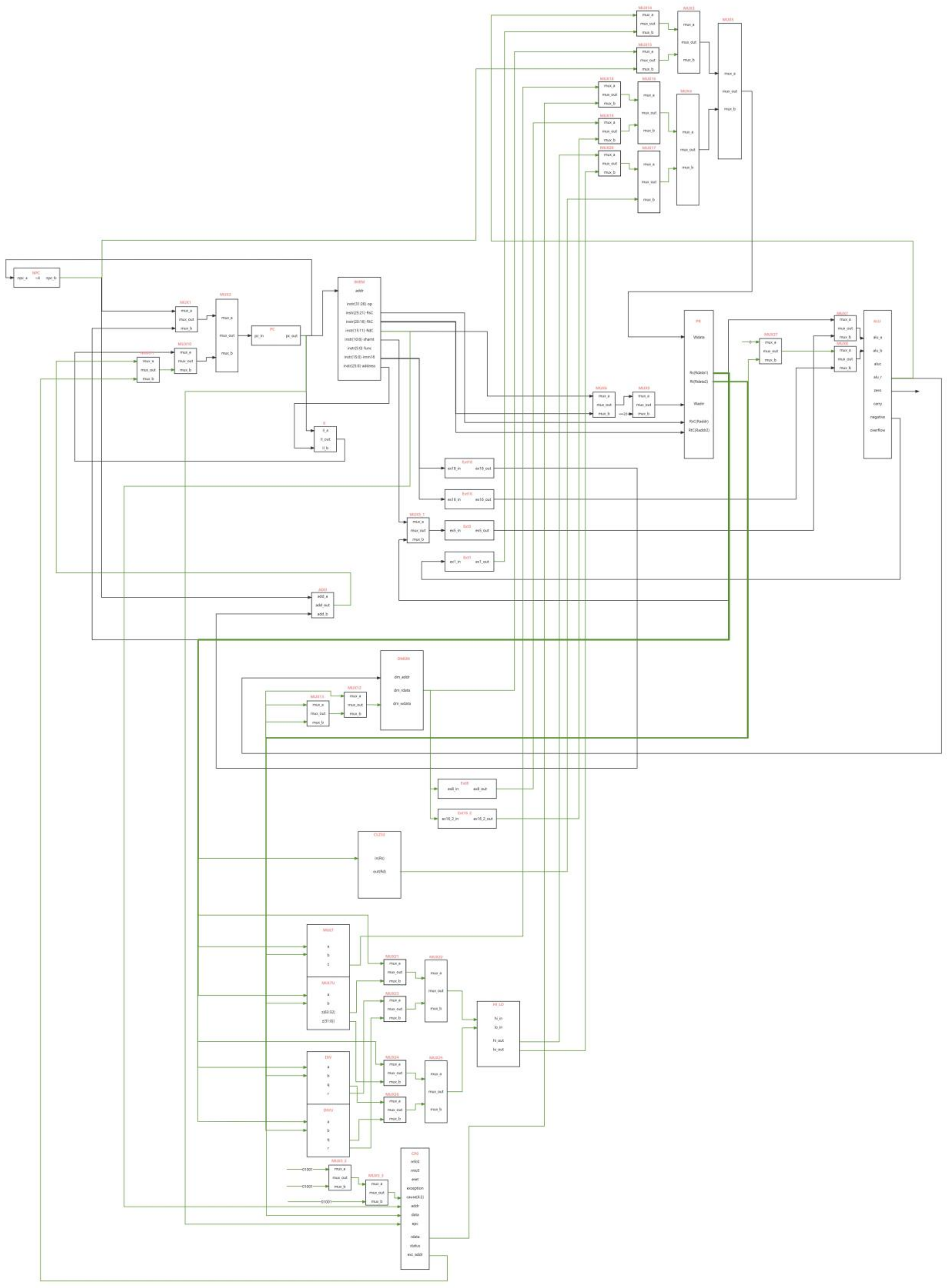
	HI-LO		CP0											CLZ32	
			in								out			in	out
	hi	lo	mfc0	mtc0	eret	exception	cause[6:2]	addr	data	epc	rdata	status	exc_addr	rs	rd
add															
addu															
sub															
subu															
and															
or															
xor															
nor															
slt															
sltu															
sll															
srl															
sra															
sllv															
srlv															
srav															
jr															
addi															
addiu															
andi															
ori															
xori															
lw															
sw															
beq															
bne															
slti															
sltiu															
lui															
j															
jal?															
类型	4	4					3								
MUX	mux21	mux24					mux5_2								
	mux22	mux25					mux5_3								
	mux23	mux26													
break						1	1001			PC		status<<5			
syscall						1	1000			PC		status<<5			
teq						1	1101			PC		status<<5			
eret					1							status>>5	+		
mfc0			1					RdC			+				
mtc0				1				RdC	Rt						
clz														Rs	+
multu	MULT_out [63:32]	MULT_out[31:0]													
mul															
divu	DIVU_q	DIVU_r													
div	DIV_q	DIV_r													
lbu															
lb															
lhu															
lh															
sb															
sh															
mfhi	+														
mflo		+													
mthi	Rs														
mtlo		Rs													
bgez															
jalr															

	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr
aluc[3]	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
aluc[2]	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	
aluc[1]	1	0	1	0	0	0	1	1	1	1	1	0	0	1	0	0	
aluc[0]	0	0	1	1	0	1	0	1	1	0	x(0)	1	0	x(0)	1	0	
mux1_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
mux10_choose																	
mux2_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr
mux11																	
mux3_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
mux4_choose																	
mux5_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
mux14_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
mux15_choose																	
mux16_choose																	
mux17_choose																	
mux18_choose																	
mux19_choose																	
mux20_choose																	
mux6_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
mux9_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
mux7_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu							
mux8_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
mux27_choose																	
mux12_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr
mux13_choose																	
mux21_choose																	
mux22_choose																	
mux23_choose																	
mux24_choose																	
mux25_choose																	
mux26_choose																	
mux5_1choose											sll	srl	sra				
rf_we	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	
add_overflow																	
ext5_signed_ext											sll		sra	sllv		srav	
ext16_signed_ext																	
ext16_left																	
dm_ena																	
dm_w																	
dm_r																	
mfc0																	
mtc0																	
eret																	
exception																	
mux5_2choose																	
mux5_3choose																	
mdc[1]																	
mdc[0]																	
div_start																	
divu_start																	
ext16_2_signed_ext																	
ext8_signed_ext																	

addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu	lui	j	jal	break	syscall	teq	eret	mfc0	mtc0
0	0	0	0	0	0	0	0	0	1	1	1					0			
0	0	1	1	1	0	0	0	0	0	0	0					0			
1	0	0	0	1	1	1	1	1	1	1	0					0			
0	0	0	1	0	0	0	1	1	1	0	x(0)					1			
addi	addiu	andi	ori	xori	lw	sw	(beq)	(bne)	slti	sltiu	lui			break	syscall	teq		mfc0	mtc0
							(beq)	(bne)				j	jal						
addi	addiu	andi	ori	xori	lw	sw	(beq)	(bne)	slti	sltiu	lui			break	syscall	teq		mfc0	mtc0
							beq	bne											
addi	addiu	andi	ori	xori					slti	sltiu	lui								
																		mfc0	
addi	addiu	andi	ori	xori	lw				slti	sltiu	lui		jal						
addi	addiu	andi	ori	xori					slti	sltiu	lui								
					lw														
																		mfc0	
addi	addiu	andi	ori	xori	lw				slti	sltiu	lui							mfc0	
addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu						teq			
							beq	bne								teq			
addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu	lui	j	jal						
addi	addiu	andi	ori	xori	lw				slti	sltiu	lui		jal					mfc0	
addi	addiu				lw	sw			slti	sltiu									
											lui								
					lw	sw													
						sw													
					lw														
																		mfc0	
																			mtc0
																	eret		
														break	syscall	(teq)			
														break					
														break	syscall				

clz	multu	mul	divu	div	lbu	lb	lhu	lh	sb	sh	mfhi	mflo	mthi	mtlo	bgez	jalr
					0	0	0	0	0	0					0	
					0	0	0	0	0	0					0	
					1	1	1	1	1	1					0	
					0	0	0	0	0	0					1	
clz	multu	mul	divu	div	lbu	lb	lhu	lh	sb	sh	mfhi	mflo	mthi	mtlo	(bgez)	
clz	multu	mul	divu	div	lbu	lb	lhu	lh	sb	sh	mfhi	mflo	mthi	mtlo	(bgez)	jalr
															bgez	
		mul			lbu	lb	lhu	lh								
																jalr
		mul														
											mfhi	mflo				
		mul														
					lbu	lb										
											mfhi					
clz		mul									mfhi	mflo				jalr
clz		mul			lbu	lb	lhu	lh			mfhi	mflo				jalr
					lbu	lb	lhu	lh	sb	sh					bgez	
															bgez	
															bgez	
										sh						
													mfhi			
	multu												mfhi			
				div												
														mtlo		
	multu													mtlo		
				div												
clz		mul			lbu	lb	lhu	lh			mfhi	mflo				jalr
					lbu	lb	lhu	lh								
					lbu	lb	lhu	lh	sb	sh						
									sb	sh						
					lbu	lb	lhu	lh								
	multu		divu	div									mthi			
	multu		divu	div										mtlo		
								lh								
						lb										

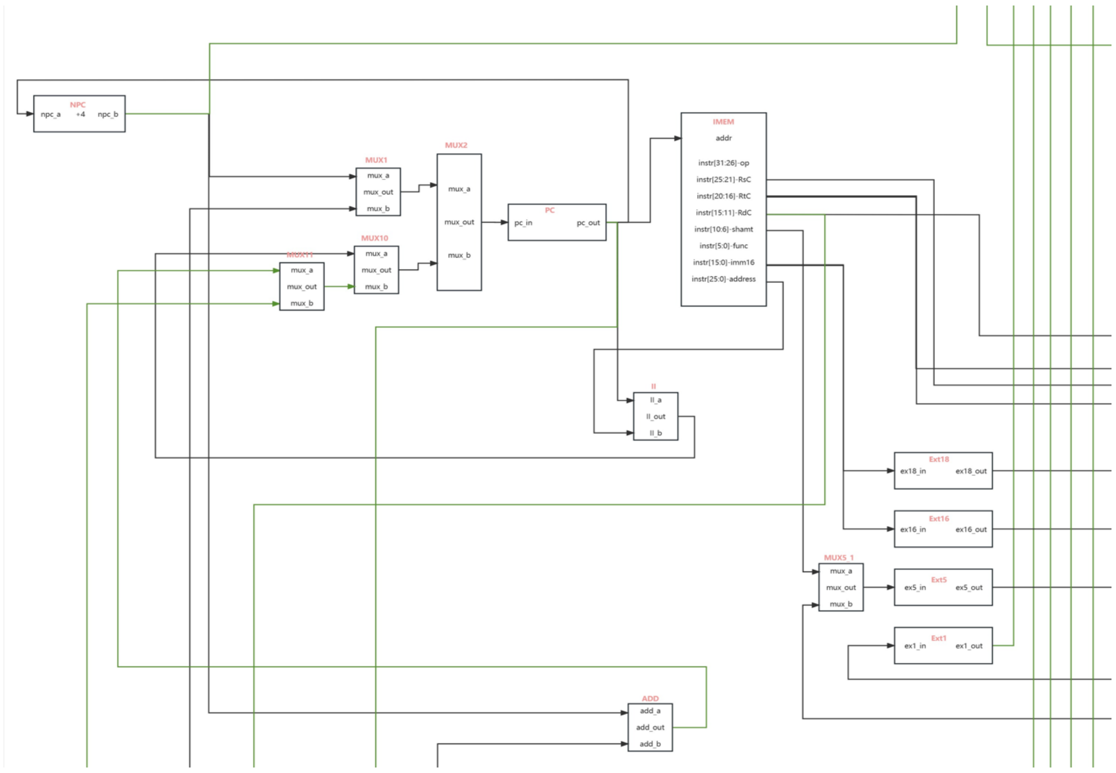
缩略图：



NPC
PC
IMEM
RF
ALU
DMEM
CLZ32
MULT
MULTU
DIV
DIVU
CP0
ADD
II
Ext18
Ext16
Ext5
Ext1
Ext8
Ext16_2

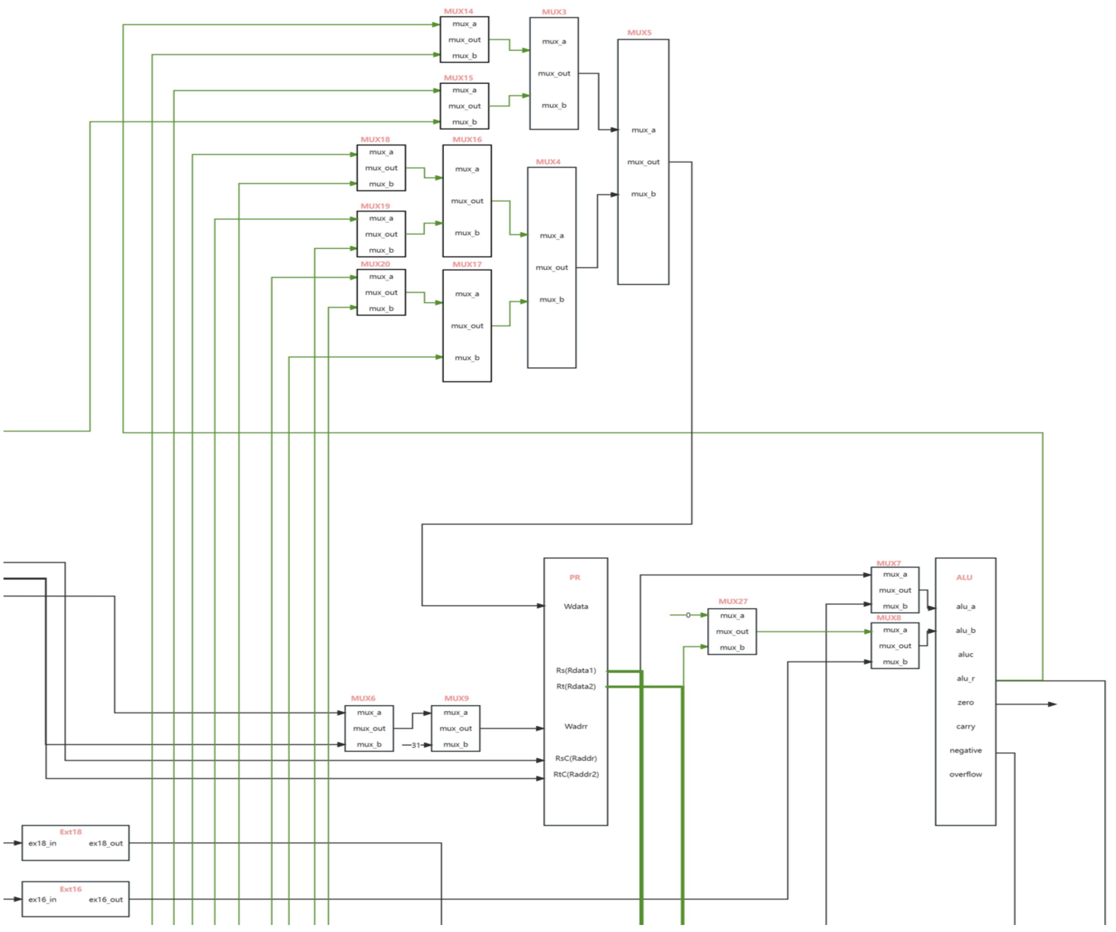
展开图：

左上：



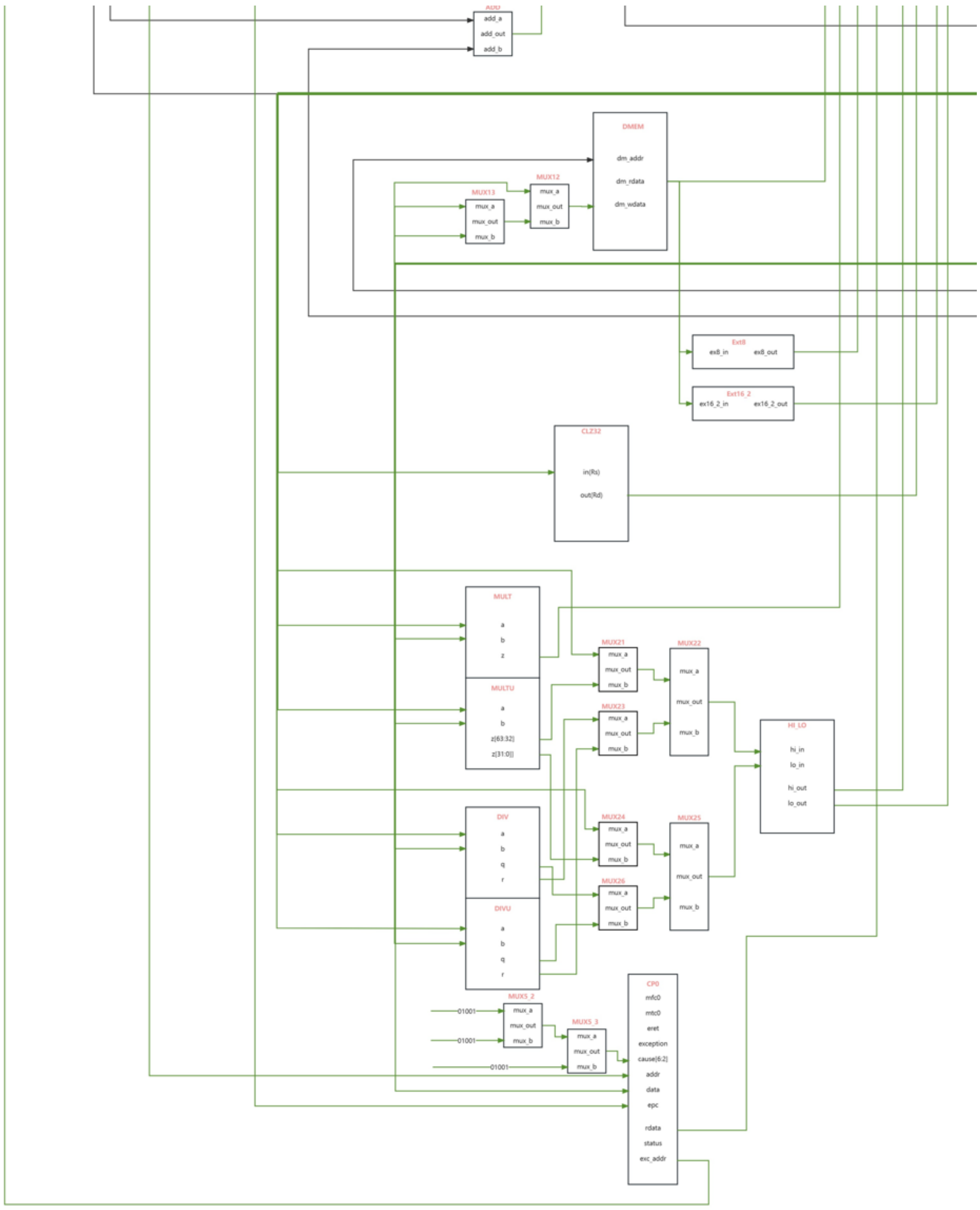
NPC
npc_a
+4
npc_b
MUX1
mux_a
mux_out
mux_b
MUX2
MUX10
MUX11
PC
pc_in
pc_out
IMEM
addr
instr[31:26]-op
instr[25:21]-RsC
instr[20:16]-RtC
instr[15:11]-RdC
instr[10:6]-shamt
instr[5:0]-func
instr[15:0]-imm16
instr[25:0]-address
II
II_a
II_out
II_b
Ext18
ex18_in
ex18_out
Ext16
ex16_in
ex16_out
MUX5_1
Ext5
ex5_in
ex5_out
Ext1
ex1_in
ex1_out
ADD
add_a
add_out
add_b

右上：



MUX14
MUX3
MUX5
MUX15
MUX18
MUX16
MUX4
MUX19
MUX20
MUX17
mux_a
mux_out
mux_b
PR
Wdata
Rs(Rdata1)
Rt(Rdata2)
Wadrr
RsC(Raddr)
RtC(Raddr2)
MUX6
MUX9
31
MUX27
0
MUX7
MUX8
ALU
alu_a
alu_b
aluc
alu_r
zero
carry
negative
overflow
Ext18
ex18_in
ex18_out
Ext16
ex16_in
ex16_out

中下：



三、模块建模

（该部分要求对实验中建模的所有模块进行功能描述，并列出各模块建模的 verilog 代码）

PC：

```
`timescale 1ns / 1ps

module PC_reg(
    input clk,
    input rst,
    input ena,
    input [31:0] PC_in,
    output [31:0] PC_out
    );
    reg [31:0] PC_r;
    always @(negedge clk or posedge rst)
    begin
        if (ena) begin
            if (rst)
            begin
                PC_r <= 32'h0040_0000;//Mars 中的 PC 是从 0x0040_0000 开始
            end
            else
            begin
                PC_r <= PC_in;
            end
        end
    end

    //assign PC_out = (ena && !rst) ? PC_r : 32'hz;
    assign PC_out = (ena) ? PC_r : 32'hz;
endmodule
```

NPC：

```
`timescale 1ns / 1ps

module NPC(
    input [31:0] a,
    input rst,
    output [31:0] b
    );

    assign b = rst ? a : a + 4;
endmodule
```

Regfiles：

```
module Regfiles(clk,rst,we,raddr1,raddr2,waddr,wdata,rdata1,rdata2);
```

```
input clk,rst,we;//we 高电位写
input[4:0]raddr1,raddr2,waddr;//raddr1-RsC raddr2-RtC waddr-RdC
input[31:0]wdata;//wdata-Rd
output [31:0]rdata1,rdata2;//rdata1-Rs rdata2-Rt

reg [31:0] array_reg[31:0];
integer i; // 定义整型变量 i，用于循环迭代

always @(posedge clk or posedge rst) begin
    if (rst) begin  // 复位时清空所有寄存器内容
        for (i = 0; i < 32; i = i + 1) begin
            array_reg[i] <= 32'h00000000;
        end
    end else begin
        if (we && waddr != 5'b0) begin // 写操作
            array_reg[waddr] <= wdata;
        end
    end
end
        // 读操作
assign rdata1 = array_reg[raddr1];
assign rdata2 = array_reg[raddr2];

endmodule
```

Ext:

```
`timescale 1ns / 1ps

module Extend18(
    input [15:0] a,
    //input signed_ext,//不需要无符号扩展
    output [31:0] b
    );

    assign b = {{(14){a[15]}}, a[15:0], 2'b00};
endmodule
```

```
`timescale 1ns / 1ps

module Extend16(
    input [15:0] a,
    input signed_ext,
    //input left,
    output [31:0] b
    );

    assign b = signed_ext ? {{(16){a[15]}}, a[15:0]} : {{(16){1'b0}},
a[15:0]};
endmodule
```

```
`timescale 1ns / 1ps

module Extend8(
    input [15:0] a,
    input signed_ext,
    //input left,
    output [31:0] b
    );

    assign b = signed_ext ? {{(24){a[7]}}, a[7:0]} : {{(24){1'b0}}, a[7:0]};

endmodule
```

```
`timescale 1ns / 1ps

module Extend5(
    input [4:0] a,
    input signed_ext,
    output [31:0] b
    );

    assign b = signed_ext ? {{(27){a[4]}}, a[4:0]} : {{(27){1'b0}}, a[4:0]};
endmodule
```

MUX:

```
`timescale 1ns / 1ps

module Mux(
    input [31:0] a,
    input [31:0] b,
    input choose,
    output [31:0] z
    );
    reg [31:0] r;
    always @(*)
    begin
    case(choose)
```

```
        1'b1:r <= a;
        1'b0:r <= b;
        default:r <= 5'bz;
    endcase
    end

    assign z = r;
endmodule
```

```
`timescale 1ns / 1ps

module Mux_5(
    input [4:0] a,
    input [4:0] b,
    input choose,
    output reg [4:0] z
    );

    always @(*)
    begin
    case(choose)
        1'b1:z <= a;
        1'b0:z <= b;
        default:z <= 5'bz;
    endcase
    end
endmodule
```

ALU:

```
`timescale 1ns / 1ps

module ALU(a,b,aluc,r,zero,carry,negative,overflow);
input [31:0] a,b;
input [3:0] aluc;
output reg [31:0] r;
output reg zero,carry,negative,overflow;

always@(*)
begin
    case(aluc)
        //add
        4'b0010:
            begin
            r=a+b;
            overflow=((a[31]==b[31])&&(~r[31]==a[31]))?1:0;
            zero=(r==0)?1:0;
            carry=0;
            negative=(r<0)?1:0;
            end
        //addu
        4'b0000:
            begin
            {carry,r}=a+b;
            zero=(r==0)?1:0;
            overflow=0;
            negative=(r[31]==1)?1:0;
            end
        //sub
        4'b0011:
            begin
            r=a-b;
            overflow=((a[31]==0 && b[31]==1 && r[31]==1) || (a[31]==1 && b[31]==0 && r[31]==0))?1:0;
            zero=(a==b)?1:0;
            carry=0;
            negative=(r<0)?1:0;
            end
        //subu
        4'b0001:
            begin
            {carry,r}=a-b;
            zero=(r==0)?1:0;
            overflow=0;
            negative=(r[31]==1)?1:0;
            end
        //and
        4'b0100:
            begin
            r=a&b;
            zero=(r==0)?1:0;
            carry=0;
            overflow=0;
            negative=(r[31]==1)?1:0;
            end
        //or
        4'b0101:
            begin
            r=a|b;
            zero=(r==0)?1:0;
            carry=0;
            overflow=0;
            negative=(r[31]==1)?1:0;
            end
        //xor
        4'b0110:
            begin
            r=a^b;
            zero=(r==0)?1:0;
            carry=0;
            overflow=0;
            negative=(r[31]==1)?1:0;
            end
        //nor
        4'b0111:
            begin
            r=~(a|b);
            zero=(r==0)?1:0;
            carry=0;
            overflow=0;
            negative=(r[31]==1)?1:0;
            end
        //slt
        11'b1011:
            begin
            if(a[31]==1 && b[31]==0)
                r=1;
            else if(a[31]==0 && b[31]==1)
                r=0;
            else
                r=(a<b)?1:0;
           overflow=r;
           zero=(r==0)?1:0;
           carry=0;
           negative=(a<b)?1:0;
           end
        //sltu
        4'b1010:
            begin
            r=(a<b)?1:0;
            carry=r;
            zero=(r==0)?1:0;
            overflow=0;
```

```
            negative=(r[31]==1)?1:0;
            end
        //sll/slr
        4'b1110:
            begin
            {carry,r}=b<<a;
            overflow=0;
            zero=(r==0)?1:0;
            negative=(r[31]==1)?1:0;
            end
        //srl
        4'b1101:
            begin
            r=b>>a;
            carry=b[a-1];
            overflow=0;
            zero=(r==0)?1:0;
            negative=(r[31]==1)?1:0;
            end
        //sra
        4'b1100:
            begin
            r=($signed(b))>>>a;
            carry=b[a];
            overflow=0;
            zero=(r==0)?1:0;
            negative=(r[31]==1)?1:0;
            end
        //lui
        4'b1000:
            begin
            r={b[15:0],16'b0};
            carry=0;
            overflow=0;
            zero=(r==0)?1:0;
            negative=(r[31]==1)?1:0;
            end
    endcase
end
endmodule
```

II：

```
`timescale 1ns / 1ps
```

```
module II(
    input [3:0] a,
    input [25:0] b,
    output [31:0] r
    );

    assign r = {a, b, 2'b00};
endmodule
```

ADD:

```
`timescale 1ns / 1ps

module add(
    input [31:0] a,
    input [31:0] b,
    output [31:0] r,
    output overflow
    );

    assign r=a+b;
    assign overflow=((a[31]==b[31])&&(~r[31]==a[31]))?1:0;//照着以前写的ALU 搬过来的
endmodule
```

CLZ:

```
`timescale 1ns / 1ps

module CLZ32 (
  input [31:0] rs,
  output reg [31:0] rd
);

integer i;

always @(*) begin
    /*rd = 32; // 默认情况下，将前导零的数量设置为 32
    for (i = 31; i >= 0; i = i - 1) begin
      if (rs[i] == 1'b1) begin
        rd = 31 - i; // 计算前导零的数量
        i = -1;
      end
    end*/
    if (rs[31]) begin
```

```
        rd = 0;
    end else if (rs[30]) begin
        rd = 1;
    end else if (rs[29]) begin
        rd = 2;
    end else if (rs[28]) begin
        rd = 3;
    end else if (rs[27]) begin
        rd = 4;
    end else if (rs[26]) begin
        rd = 5;
    end else if (rs[25]) begin
        rd = 6;
    end else if (rs[24]) begin
        rd = 7;
    end else if (rs[23]) begin
        rd = 8;
    end else if (rs[22]) begin
        rd = 9;
    end else if (rs[21]) begin
        rd = 10;
    end else if (rs[20]) begin
        rd = 11;
    end else if (rs[19]) begin
        rd = 12;
    end else if (rs[18]) begin
        rd = 13;
    end else if (rs[17]) begin
        rd = 14;
    end else if (rs[16]) begin
        rd = 15;
    end else if (rs[15]) begin
        rd = 16;
    end else if (rs[14]) begin
        rd = 17;
    end else if (rs[13]) begin
        rd = 18;
    end else if (rs[12]) begin
        rd = 19;
    end else if (rs[11]) begin
        rd = 20;
    end else if (rs[10]) begin
        rd = 21;
    end else if (rs[9]) begin
```

```
        rd = 22;
    end else if (rs[8]) begin
        rd = 23;
    end else if (rs[7]) begin
        rd = 24;
    end else if (rs[6]) begin
        rd = 25;
    end else if (rs[5]) begin
        rd = 26;
    end else if (rs[4]) begin
        rd = 27;
    end else if (rs[3]) begin
        rd = 28;
    end else if (rs[2]) begin
        rd = 29;
    end else if (rs[1]) begin
        rd = 30;
    end else if (rs[0]) begin
        rd = 31;
    end else begin
        rd = 32;
    end
end

endmodule
```

CP0：

```
module CP0(
  input clk,              // 时钟信号
  input rst,              // 复位信号
  input mfc0,             // 指令为 mfc0
  input mtc0,             // 指令为 mtc0
  input eret,             // 指令为 eret
  input exception,        // 异常发生信号
  input [4:0] cause,      // 异常原因
  input [4:0] addr,       // cp0 寄存器地址
  input [31:0] data,      // 写入的数据
  input [31:0] pc,        // 程序计数器
  output wire [31:0] rdata,   // Cp0 寄存器读出数据
  output wire [31:0] status,  // 状态
  output wire [31:0] exc_addr // 异常发生地址
);

// 异常入口地址
```

```
parameter ExcInAddr = 32'h4;

// CP0 寄存器地址
parameter STATUSADDR = 5'd12; //01100
parameter EPCADDR = 5'd14; //01110
parameter CAUSEADDR = 5'd13; //01101

reg [31:0] cp0Reg[31:0];

always @(posedge clk or posedge rst) begin
  if (rst) begin
    cp0Reg[STATUSADDR] <= {27'd0, 5'b11111};
  end else begin
    if (mtc0) begin
      cp0Reg[addr] <= data;
    end
    if (exception) begin
      cp0Reg[STATUSADDR] <= {cp0Reg[STATUSADDR][26:0], 5'd0};
      cp0Reg[CAUSEADDR] <= {25'd0, cause, 2'd0};
      cp0Reg[EPCADDR] <= pc;
    end
    if (eret) begin
      cp0Reg[STATUSADDR] <= {5'd0, cp0Reg[STATUSADDR][31:5]};
    end
  end
end

assign exc_addr = cp0Reg[EPCADDR];
assign status = cp0Reg[STATUSADDR];
assign rdata = mfc0 ? cp0Reg[addr] : 32'hzzzzzzzz;

endmodule
```

DIV：

```
`timescale 1ns / 1ps

module DIV(
    input [31:0] dividend,
    input [31:0] divisor,
    input start,    //start=is_div&~busy
    input clock,
    input reset,
    output [31:0]q,   //商
    output [31:0]r,   //余数
    output busy    //除法器忙标志位
    );
    //对于有符号数除法将其化为无符号数后利用无符号除法处理，然后商的符号位由异
或决定，余数的符号与被除数同号
    reg is_busy;
    reg [31:0]pos_divisor;
    reg [31:0]neg_divisor;
    reg [63:0]save;
    reg [31:0]true_dividend;     //用来存储该数绝对值的补码
    reg [31:0]true_divisor;

    assign busy=is_busy;
    assign q=save[31:0];
    assign r=save[63:32];

    integer i;
    always@(*)
    begin
        if(reset)
        begin
            is_busy=1'b0;
        end
       else if(start)
        begin
            is_busy=1'b1;
            true_dividend=(dividend[31]==0)?dividend:~(dividend-1);
            true_divisor=(divisor[31]==0)?divisor:~(divisor-1);
            save={32'b0,true_dividend};
            pos_divisor=true_divisor;
            neg_divisor=~pos_divisor+1;
            is_busy=1'b1;
            if(save!=64'b0)      //对被除数是 0 的情况单独处理
            begin
                for(i=0;i<32;i=i+1)
                begin
                    save={save[62:0],1'b0};
                    if(save[63:32]>=pos_divisor)
                        save={save[63:32]+neg_divisor,save[31:0]+1'b1};
                end
                //该步骤结束 save 中存储绝对值的除法操作后的商和余数
                save[31:0]=(dividend[31]^divisor[31]==1'b1)?~save[31:0]+
1:save[31:0];   //处理商
                save[63:32]=(dividend[31]==1'b1)?~save[63:32]+1:save[63:
32];       //处理余数
```

```
            end
            is_busy=1'b0;
        end
    end
endmodule
```

DIVU:

```
`timescale 1ns / 1ps

module DIVU(
    input [31:0] dividend,
    input [31:0] divisor,
    input start,    //start=is_div&~busy
    input clock,
    input reset,
    output [31:0]q,   //商
    output [31:0]r,   //余数
    output busy    //除法器忙标志位
    );
    reg is_busy;
    reg[31:0]pos_divisor;
    reg[31:0]neg_divisor;
    reg [63:0]save;

    assign busy=is_busy;
    assign q=save[31:0];
    assign r=save[63:32];

    integer i;
    always@(*)
    begin
        if(reset)
            is_busy=1'b0;
       else if(start)
        begin
            is_busy=1'b1;
            save={32'b0,dividend};
            pos_divisor=divisor;
            neg_divisor=~pos_divisor+1;
            if(save==64'b0)   //对被除数是 0 的情况单独处理
                save={32'b0,dividend};
            else
            begin
                for(i=0;i<32;i=i+1)
                begin
                    save={save[62:0],1'b0};
                    if(save[63:32]>=pos_divisor)
                        save={save[63:32]+neg_divisor,save[31:0]+1'b1};
                    else
                    ;
                end
            end
            is_busy=1'b0;
        end
    end
endmodule
```

MULT:

```
`timescale 1ns / 1ps

module MULT(
//ĘĪĎÖÓĐ˘űşĹłË˘¨Ć÷
    input clk,
    input reset,
    input cs,
    input [31:0]a,
    input [31:0]b,
    output [63:0]z
    );
    reg [32:0]reg_a_pos;      //´ć´˘aĺÄ ¸šÂë
    reg [32:0]reg_a_neg;      //´ć´˘[-a]ĺÄ ¸šÂë
    reg [32:0]partial_product;      //´ć´˘ ¸ż˘Öťý
    reg [32:0]product;        //´ć´˘łËĘý
    integer i;
    always@(*)
    begin
        if(reset)
        begin
            reg_a_pos=(a[31]==0)?{1'b0,a}:{1'b1,a};
            reg_a_neg=(a[31]==0)?{1'b1,~a+1'b1}:{1'b0,~(a-1'b1)};
            product={b,1'b0};
            partial_product=33'h00000000;
        end
        else if(cs==1)
        begin
            if(a==0 || b==0)
            begin
                partial_product=33'b0;
                product=33'b0;
            end
            else
            begin
                reg_a_pos=(a[31]==0)?{1'b0,a}:{1'b1,a};
                reg_a_neg=(a[31]==0)?{1'b1,~a+1'b1}:{1'b0,~(a-1'b1)};
                product={b,1'b0};
                partial_product=33'h00000000;
                for(i=0;i<32;i=i+1)
                begin
                    if({product[1],product[0]}==2'b10)
                    begin
                        partial_product=partial_product+reg_a_neg;
                    end
                    else if({product[1],product[0]}==2'b01)
                    begin
                        partial_product=partial_product+reg_a_pos;
                    end
                    product={partial_product[0],product[32:1]};
                    partial_product={partial_product[32],partial_product
[32:1]};              //Ö´ĐĐŇĆÎť¸Ů×÷
                end
            end
        end
    end
    assign z={partial_product[31:0],product[32:1]};
endmodule
```

MULTU:

```
`timescale 1ns / 1ps

module MULTU(
    input clk,
    input reset,      //高电平有效
    input cs,         //乘法运行信号
    input [31:0]a,        //存储被乘数
    input [31:0]b,        //存储乘数
    output [63:0]z        //输出结果
    );
    reg [32:0] partial_product;       //存储部分积(采取 2 位符号位)
    reg [31:0] product_number;         //存储乘数
    integer i;
    always@(*)
    begin
```

```verilog
        if(reset)
        begin
            product_number=b;
            partial_product=33'h00000000;
        end
       else if(cs==1)
        begin
            product_number=b;
            partial_product=33'h00000000;
            for(i=0;i<32;i=i+1)
            begin
                if(product_number[0]==1'b1)
                begin
                    partial_product=partial_product+a;
                end
                product_number={partial_product[0],product_number[31:1]}
;
                partial_product={1'b0,partial_product[32:1]};
  //执行移位操作
            end
        end
    end
    assign z={partial_product[31:0],product_number};
endmodule
```

HILO:

```verilog
`timescale 1ns / 1ps

module HILO(
input clk,
    input reset,
    input [1:0] mdc, // 控制信号
    input [31:0] a,
    input [31:0] b,
    output reg [31:0] hi,
    output reg [31:0] lo
);

    always @(*)
        begin
            if (reset)
            begin
                hi <= 0;
                lo <= 0;
            end
            else
            begin
                hi = mdc[1] ? a : hi;
                lo = mdc[0] ? b : lo;
            end
        end

endmodule
```

CPU + control + connect：

```
`timescale 1ns / 1ps

module cpu_top(
    input clk,
    input ena,
    input rst,

    input [31:0] IM_inst,
    input [31:0] DM_rdata,

    output DM_ena,
    output DM_W,
    output DM_R,
    output [10:0] addr,
    output [31:0] DM_wdata,
    output [31:0] PC_out,
    output [31:0] ALU_out
    );


    //imem
    wire [31:0] imem;
    assign imem = IM_inst;

    //dmem
    wire [31:0] dm_rdata  = DM_rdata;
    wire [31:0] dm_wdata;
    //reg [10:0] dm_addr;
    wire dm_ena;
    wire dm_w, dm_r;


    //assign addr = dm_addr;
    assign DM_ena = dm_ena;
    assign DM_W = dm_w;
    assign DM_R = dm_r;
    assign DM_wdata = dm_wdata;

    wire [31:0] pc_in;
    wire [31:0] pc_out;//
    assign PC_out = pc_out;

    //npc
    wire [32:0] npc_a, npc_b;

    //alu
    wire ZF, CF, NF, OF;
    wire [31:0] alu_a, alu_b, alu_r;
    wire [4:0] aluc;
    assign ALU_out = alu_r;

    //regfiles
    wire rf_clk = ~clk;
    wire rf_we;
    wire [4:0] RsC, RtC, RdC;//raddr1-RsC raddr2-RtC waddr-RdC
    wire [31:0] Rd;//wdata-Rd
    wire [31:0] Rs, Rt;//rdata1-Rs rdata2-Rt

    //mux
    wire [31:0] mux1_a, mux1_b, mux1_z, mux2_a, mux2_b, mux2_z, mux3_a,
mux3_b, mux3_z,
                mux4_a, mux4_b, mux4_z, mux5_a, mux5_b, mux5_z, mux6_a,
mux6_b, mux6_z,
                mux7_a, mux7_b, mux7_z, mux8_a, mux8_b, mux8_z, mux9_a,
mux9_b, mux9_z,
                mux10_a, mux10_b, mux10_z, mux11_a, mux11_b, mux11_z,
mux12_a, mux12_b, mux12_z,
                mux13_a, mux13_b, mux13_z, mux14_a, mux14_b, mux14_z,
mux15_a, mux15_b, mux15_z,
                mux16_a, mux16_b, mux16_z, mux17_a, mux17_b, mux17_z,
mux18_a, mux18_b, mux18_z,
                mux19_a, mux19_b, mux19_z, mux20_a, mux20_b, mux20_z,
mux21_a, mux21_b, mux21_z,
                mux22_a, mux22_b, mux22_z, mux23_a, mux23_b, mux23_z,
mux24_a, mux24_b, mux24_z,
                 mux25_a, mux25_b, mux25_z, mux26_a, mux26_b, mux26_z,
mux27_a, mux27_b, mux27_z;

    wire mux1_choose, mux2_choose, mux3_choose,
          mux4_choose, mux5_choose, mux6_choose,
          mux7_choose, mux8_choose, mux9_choose,
          mux10_choose, mux11_choose, mux12_choose,
          mux13_choose, mux14_choose, mux15_choose,
          mux16_choose, mux17_choose, mux18_choose,
          mux19_choose, mux20_choose, mux21_choose,
          mux22_choose, mux23_choose, mux24_choose,
          mux25_choose, mux26_choose, mux27_choose;

    //mux_5
    wire [4:0] mux5_1a, mux5_1b, mux5_1z, mux5_2a, mux5_2b, mux5_2z, mux5_3a,
mux5_3b, mux5_3z;
    wire mux5_1choose, mux5_2choose, mux5_3choose;
    //assign mux5_1a[4:0] = imem[10:6];

    //Extend1
    wire ext1_a;
    wire [31:0] ext1_b;

    //Extend5
    wire [4:0] ext5_a;
    wire ext5_signed_ext;
    wire [31:0] ext5_b;

    //Extend16
    wire [15:0] ext16_a;
    wire ext16_signed_ext;
    //wire ext16_left;
    wire [31:0] ext16_b;

//Extend16_2
    wire [15:0] ext16_2_a;
    wire ext16_2_signed_ext;
    wire [31:0] ext16_2_b;

//Extend8
    wire [15:0] ext8_a;
    wire ext8_signed_ext;
    wire [31:0] ext8_b;
```

```verilog
    //Extend18
    wire [15:0] ext18_a;
    wire ext18_signed_ext;
    wire [31:0] ext18_b;

    //II
    wire [3:0] ii_a;
    wire [25:0] ii_b;
    wire [31:0] ii_r;

    //add
    wire [31:0] add_a;
    wire [31:0] add_b;
    wire [31:0] add_r;
    wire add_overflow;

    //mul div
    wire [31:0] mult_a, mult_b, div_q, divu_q, div_r, divu_r;
    wire [63:0] mult_z, multu_z;
    wire div_start, div_busy;

    //cp0
    wire mfc0, mtc0, eret, exception;
    wire [4:0] cause, cp0_addr;
    wire [31:0] cp0_data, epc, cp0_rdata, status, exc_addr;

    //clz
    wire [31:0] clz_rs, clz_rd;

    //hilo
    wire [1:0] mdc;
    wire [31:0] hilo_a, hilo_b;
    wire [31:0] hi, lo;

    //connect
    assign PC_out = pc_out;
    assign npc_a = pc_out;

    assign mux1_a = npc_b;
    assign mux2_a = mux1_z;
    assign pc_in = mux2_z;
    assign ii_a = pc_out[31:28];
    assign mux10_a = ii_r;
    assign mux10_b = mux11_z;
    assign mux11_a = add_r;
    assign mux11_b = exc_addr;
    assign mux2_b = mux10_z;
    assign ii_b = imem[25:0];
    assign mux1_b = Rs;

    assign add_a = npc_b;
    assign add_b = ext18_b;

    assign mux5_b = mux4_z;
    assign mux5_a = mux3_z;
    assign Rd = mux5_z;
    assign mux3_a = mux14_z;
    assign mux14_a = alu_r;
    assign mux14_b = ext1_b;
    assign mux3_b = mux15_z;
    assign mux15_a = dm_rdata;
    assign mux15_b = npc_b;
    assign mux4_a = mux16_z;
    assign mux16_a = mux18_z;
    assign mux18_a = mult_z;
    assign mux18_b = cp0_rdata;
    assign mux16_b = mux19_z;
    assign mux19_a = ext8_b;
    assign mux19_b = ext16_2_b;
    assign mux4_b = mux17_z;
    assign mux17_a = mux20_z;
    assign mux20_a = hi;
    assign mux20_b = lo;
    assign mux17_b = clz_rd;

    assign epc = pc_out;
    assign cp0_addr = imem[15:11];
    assign cp0_data = Rt;

    assign divu_a = Rs;
    assign div_a = Rs;
    assign mult_a = Rs;
    assign multu_a = Rs;
    assign clz_rs = Rs;
    assign mux21_a = Rs;
    assign mux24_a = Rs;

    assign divu_b = Rt;
    assign div_b = Rt;
    assign mult_b = Rt;
    assign multu_b = Rt;
    assign mux12_a = Rt;
    assign mux12_b = mux13_z;
    assign mux13_a = Rt[15:0];
    assign mux13_b = Rt[7:0];
    //assign dm_addr = alu_r[10:0];
    assign dm_wdata = mux12_z;
    assign addr = alu_r[10:0];

    assign ext8_a = dm_rdata[7:0];
    assign ext16_2_a = dm_rdata[15:0];

    assign hilo_a = mux22_z;
    assign mux22_a = mux21_z;
    //assign mux21_a = Rs;
    assign mux21_b = multu_z[63:32];
    assign mux22_b = mux23_z;
    assign mux23_a = div_r;
    assign mux23_b = divu_r;

    assign hilo_b = mux25_z;
    assign mux25_a = mux24_z;
    //assign mux24_a = Rs;
    assign mux24_b = multu_z[31:0];
    assign mux25_b = mux26_z;
    assign mux26_a = div_q;
    assign mux26_b = divu_q;

    assign mux6_a = imem[15:11];
    assign mux6_b = imem[20:16];
    assign mux9_a = mux6_z;
    assign mux9_b = 31;
    assign RdC = mux9_z;
    assign RsC = imem[25:21];
    assign RtC = imem[20:16];
    assign ext18_a = imem[15:0];
    assign ext16_a = imem[15:0];
    assign mux5_1a = imem[10:6];
    assign mux5_1b = Rs;
    assign ext5_a = mux5_1z;
    assign ext1a = NF;
    assign mux8_b = ext16_b;
    assign mux7_b = ext5_b;
    assign mux7_a = Rs;
    assign mux8_a = mux27_z;
    assign mux27_a = 32'b0;
    assign mux27_b = Rt;
    assign alu_a = mux7_z;
    assign alu_b = mux8_z;

    assign cause = mux5_3z;
    assign mux5_3a = mux5_2z;
    assign mux5_2a = 5'b01001;
    assign mux5_2b = 5'b01000;
    assign mux5_3b = 5'b01101;


    //31
    wire _add, _addu, _sub, _subu, _and, _or, _xor, _nor;
    wire _slt, _sltu, _sll, _srl, _sra, _sllv, _srlv, _srav, _jr;
    wire  _addi, _addiu, _andi, _ori, _xori, _lw, _sw;
    wire _beq, _bne, _slti, _sltiu, _lui, _j, _jal;
    //23
    wire _break, _syscall, _teq, _eret;
    wire _mfc0, _mtc0;
    wire _clz;
    wire _multu, _mul, _divu, _div;
    wire _lbu, _lb, _lhu, _lh, _sb, _sh;
    wire _mfhi, _mflo, _mthi, _mtlo;
    wire _bgez, _jalr;

    //decode
    //1~17
    assign _add =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100000)?1'b1:1'b0;
    assign _addu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100001)?1'b1:1'b0;
    assign _sub =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100010)?1'b1:1'b0;
    assign _subu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100011)?1'b1:1'b0;
    assign _and =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100100)?1'b1:1'b0;
    assign _or =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100101)?1'b1:1'b0;
```

```verilog
    assign _xor =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100110)?1'b1:1'b0;
    assign _nor =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100111)?1'b1:1'b0;

    assign _slt =
(imem[31:26]==6'b000000&&imem[5:0]==6'b101010)?1'b1:1'b0;
    assign _sltu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b101011)?1'b1:1'b0;
    assign _sll =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000000)?1'b1:1'b0;
    assign _srl =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000010)?1'b1:1'b0;
    assign _sra =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000011)?1'b1:1'b0;
    assign _sllv =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000100)?1'b1:1'b0;
    assign _srlv =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000110)?1'b1:1'b0;
    assign _srav =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000111)?1'b1:1'b0;
    assign _jr =
(imem[31:26]==6'b000000&&imem[5:0]==6'b001000)?1'b1:1'b0;

    //18~29
    assign _addi = (imem[31:26]==6'b001000)?1'b1:1'b0;
    assign _addiu = (imem[31:26]==6'b001001)?1'b1:1'b0;
    assign _andi = (imem[31:26]==6'b001100)?1'b1:1'b0;
    assign _ori = (imem[31:26]==6'b001101)?1'b1:1'b0;
    assign _xori = (imem[31:26]==6'b001110)?1'b1:1'b0;
    assign _lw = (imem[31:26]==6'b100011)?1'b1:1'b0;
    assign _sw = (imem[31:26]==6'b101011)?1'b1:1'b0;
    assign _beq = (imem[31:26]==6'b000100)?1'b1:1'b0;
    assign _bne = (imem[31:26]==6'b000101)?1'b1:1'b0;
    assign _slti = (imem[31:26]==6'b001010)?1'b1:1'b0;
    assign _sltiu = (imem[31:26]==6'b001011)?1'b1:1'b0;
    assign _lui = (imem[31:26]==6'b001111)?1'b1:1'b0;

    //30 31
    assign _j = (imem[31:26]==6'b000010)?1'b1:1'b0;
    assign _jal = (imem[31:26]==6'b000011)?1'b1:1'b0;

    //32~54
    assign _break =
(imem[31:26]==6'b000000&&imem[5:0]==6'b001101)?1'b1:1'b0;
    assign _syscall =
(imem[31:26]==6'b000000&&imem[5:0]==6'b001100)?1'b1:1'b0;
    assign _teq =
(imem[31:26]==6'b000000&&imem[5:0]==6'b110100)?1'b1:1'b0;
    assign _eret =
(imem[31:26]==6'b010000&&imem[5:0]==6'b011000)?1'b1:1'b0;

    assign _mfc0 =
(imem[31:26]==6'b010000&&imem[5:0]==6'b000000&&imem[25:21]==5'b00000)?1
'b1:1'b0;
    assign _mtc0 =
(imem[31:26]==6'b010000&&imem[5:0]==6'b000000&&imem[25:21]==5'b00100)?1
'b1:1'b0;

    assign _clz =
(imem[31:26]==6'b011100&&imem[5:0]==6'b100000)?1'b1:1'b0;

    assign _multu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b011001)?1'b1:1'b0;
    assign _mul =
(imem[31:26]==6'b011100&&imem[5:0]==6'b000010)?1'b1:1'b0;
    assign _divu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b011011)?1'b1:1'b0;
    assign _div =
(imem[31:26]==6'b000000&&imem[5:0]==6'b011010)?1'b1:1'b0;

    assign _lbu = (imem[31:26]==6'b100100)?1'b1:1'b0;
    assign _lb = (imem[31:26]==6'b100000)?1'b1:1'b0;
    assign _lhu = (imem[31:26]==6'b100101)?1'b1:1'b0;
    assign _lh = (imem[31:26]==6'b100001)?1'b1:1'b0;
    assign _sb = (imem[31:26]==6'b101000)?1'b1:1'b0;
    assign _sh = (imem[31:26]==6'b101001)?1'b1:1'b0;

    assign _mfhi =
(imem[31:26]==6'b000000&&imem[5:0]==6'b010000)?1'b1:1'b0;
    assign _mflo =
(imem[31:26]==6'b000000&&imem[5:0]==6'b010010)?1'b1:1'b0;
    assign _mthi =
(imem[31:26]==6'b000000&&imem[5:0]==6'b010001)?1'b1:1'b0;
    assign _mtlo =
(imem[31:26]==6'b000000&&imem[5:0]==6'b010011)?1'b1:1'b0;
```

```
    assign _bgez = (imem[31:26]==6'b000001)?1'b1:1'b0;
    assign _jalr =
(imem[31:26]==6'b000000&&imem[5:0]==6'b001001)?1'b1:1'b0;

    //control
    assign aluc[3] = _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv
|| _srav || _slti || _sltiu || _lui;
    assign aluc[2] = _and || _or || _xor || _nor || _sll || _srl || _sra ||
_sllv || _srlv || _srav || _andi || _ori || _xori;
    assign aluc[1] = _add || _sub || _xor || _nor || _slt || _sltu || _sll
|| _sllv || _addi || _xori || _lw || _sw || _beq || _bne || _slti || _sltiu
|| _lbu || _lb || _lhu || _lh || _sb || _sh;
    assign aluc[0] = _sub || _subu || _or || _nor || _slt || _srl || _srlv
|| _ori || _beq || _bne || _slti || _teq || _bgez;

    assign mux1_choose = (_add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
_srav || _addi || _addiu || _andi || _ori || _xori || _lw || _sw || _slti
|| _sltiu || _lui || _break || _syscall || _teq || _mfc0 || _mtc0 || _clz
|| _multu || _mul || _divu || _div || _lbu || _lb || _lhu || _lh || _sb ||
_sh || _mfhi || _mflo || _mthi || _mtlo || ((_beq) && (!ZF)) || ((_bne) &&
(ZF)) || ((_bgez) && (NF)));
    assign mux10_choose = (_j || _jal);
    assign mux2_choose  = (_add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _jr || _sllv ||
_srlv || _srav || _addi || _addiu || _andi || _ori || _xori || _lw || _sw
|| _slti || _sltiu || _lui || _break || _syscall || _teq || _mfc0 || _mtc0
|| _clz || _multu || _mul || _divu || _div || _lbu || _lb || _lhu || _lh
|| _sb || _sh || _mfhi || _mflo || _mthi || _mtlo || _jalr || ((_beq) &&
(!ZF)) || ((_bne) && (ZF)) || ((_bgez) && (NF)));
    assign mux11_choose = (((_beq) && (ZF)) || ((_bne) && (!ZF)) || ((_bgez)
&& (!NF)));

    assign mux3_choose = _add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
_srav || _addi || _addiu || _andi || _ori || _xori || _slti || _sltiu ||
_lui;
    assign mux4_choose = _mfc0 || _mul || _lbu || _lb || _lhu || _lh;
    assign mux5_choose = _add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
_srav || _addi || _addiu || _andi || _ori || _xori || _lw || _slti || _sltiu
|| _lui || _jal || _jalr;
    assign mux14_choose = _add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
_srav || _addi || _addiu || _andi || _ori || _xori || _slti || _sltiu ||
_lui;
    assign mux15_choose = _lw;
    assign mux16_choose = _mfc0 || _mul;
    assign mux17_choose = _mfhi || _mflo;
    assign mux18_choose = _mul;
    assign mux19_choose = _lbu || _lb;
    assign  mux20_choose = _mfhi;

    assign mux6_choose = _add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
_srav || _clz || _mul || _mfhi || _mflo || _jalr;
    assign mux7_choose = _add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _addi || _addiu || _andi || _ori || _xori
|| _lw || _sw || _beq || _bne || _slti || _sltiu || _teq || _lbu || _lb ||
_lhu || _lh || _sb || _sh || _bgez;
    assign mux8_choose = _add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
_srav || _beq || _bne || _teq || _bgez;
    assign mux9_choose = _add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
_srav || _addi || _addiu || _andi || _ori || _xori || _lw || _slti || _sltiu
|| _lui || _mfc0 || _clz || _mul || _lbu || _lb || _lhu || _lh || _mfhi ||
_mflo || _jalr;
    assign mux27_choose = _bgez;

    assign mux12_choose = (_add || _addu || _sub || _subu || _and || _or ||
_xor || _nor || _slt || _sltu || _sll || _srl || _sra || _jr || _sllv ||
_srlv || _srav || _addi || _addiu || _andi || _ori || _xori || _lw || _sw
|| _slti || _sltiu || _lui || _j || _jal);
    assign mux13_choose = _sh;

    assign mux21_choose = _mfhi;
    assign mux22_choose = _multu || _mfhi;
    assign mux23_choose = _div;
    assign mux24_choose = _mtlo;
    assign mux25_choose = _multu || _mtlo;
    assign mux26_choose = _div;

    assign mux5_1choose = _sll || _srl || _sra;

    assign rf_we = _add || _addu || _sub || _subu || _and || _or || _xor ||
_nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv || _srav
|| _addi || _addiu || _andi || _ori || _xori || _lw || _slti || _sltiu ||
```

```verilog
_lui || _jal || _mfc0 || _clz || _mul || _lbu || _lb || _lhu || _lh || _mfhi
|| _mflo || _jalr;
    assign add_overflow = 0;
    assign ext5_signed_ext = 0;
    assign ext16_signed_ext = _addi || _addiu || _lw || _sw || _slti || _sltiu
|| _lbu || _lb || _lhu || _lh;

    assign dm_ena = _sw || _lw || _lbu || _lb || _lhu || _lh || _sb || _sh;
    assign dm_r = _lw || _lbu || _lb || _lhu || _lh;
    assign dm_w = _sw || _sb || _sh;

    assign mfc0 = _mfc0;
    assign mtc0 = _mtc0;
    assign eret = _eret;
    assign exception = _break || _syscall || (_teq&&ZF);

    assign mux5_2choose = _break;
    assign mux5_3choose = _break || _syscall;

    assign mdc[1] = _multu || _divu || _div || _mthi;
    assign mdc[0] = _multu || _divu || _div || _mtlo;

    assign div_start = 1;
    assign divu_start = 1;

    assign ext16_2_signed_ext = _lh;
    assign ext8_signed_ext = _lb;

    NPC npc(
        .a(npc_a),
        .rst(rst),
        .b(npc_b)
    );


    PC_reg pc_reg(
        .clk(clk),
        .rst(rst),
        .ena(ena),
        .PC_in(pc_in),
        .PC_out(pc_out)
    );
    Regfiles cpu_ref(
        .clk(rf_clk),
        .rst(rst),
        .we(rf_we),//高电平可以写入
        .raddr1(RsC),
        .raddr2(RtC),
        .waddr(RdC),
        .wdata(Rd),
        .rdata1(Rs),
        .rdata2(Rt)
    );

    ALU alu(
        .a(alu_a),
        .b(alu_b),
        .aluc(aluc),
        .r(alu_r),
        .zero(ZF),
        .carry(CF),
        .negative(NF),
        .overflow(OF)
    );

    //Mux
    Mux mux1(
        .a(mux1_a),
        .b(mux1_b),
        .choose(mux1_choose),
        .z(mux1_z)
    );
    Mux mux2(
        .a(mux2_a),
        .b(mux2_b),
        .choose(mux2_choose),
        .z(mux2_z)
    );
    Mux mux3(
        .a(mux3_a),
        .b(mux3_b),
        .choose(mux3_choose),
        .z(mux3_z)
    );
    Mux mux4(
        .a(mux4_a),
        .b(mux4_b),
        .choose(mux4_choose),
        .z(mux4_z)
    );
    Mux mux5(
        .a(mux5_a),
        .b(mux5_b),
        .choose(mux5_choose),
        .z(mux5_z)
    );
    Mux mux6(
        .a(mux6_a),
        .b(mux6_b),
        .choose(mux6_choose),
        .z(mux6_z)
    );
    Mux mux7(
        .a(mux7_a),
        .b(mux7_b),
        .choose(mux7_choose),
        .z(mux7_z)
    );
    Mux mux8(
        .a(mux8_a),
        .b(mux8_b),
        .choose(mux8_choose),
        .z(mux8_z)
    );
    Mux mux9(
        .a(mux9_a),
        .b(mux9_b),
        .choose(mux9_choose),
        .z(mux9_z)
    );
    Mux mux10(
           .a(mux10_a),
           .b(mux10_b),
           .choose(mux10_choose),
           .z(mux10_z)
    );
    Mux mux11(
           .a(mux11_a),
           .b(mux11_b),
           .choose(mux11_choose),
           .z(mux11_z)
     );
         Mux mux12(
             .a(mux12_a),
             .b(mux12_b),
             .choose(mux12_choose),
             .z(mux12_z)
         );
         Mux mux13(
             .a(mux13_a),
             .b(mux13_b),
             .choose(mux13_choose),
             .z(mux13_z)
         );
         Mux mux14(
             .a(mux14_a),
             .b(mux14_b),
             .choose(mux14_choose),
             .z(mux14_z)
         );
         Mux mux15(
             .a(mux15_a),
             .b(mux15_b),
             .choose(mux15_choose),
             .z(mux15_z)
         );
         Mux mux16(
             .a(mux16_a),
             .b(mux16_b),
             .choose(mux16_choose),
             .z(mux16_z)
         );
         Mux mux17(
             .a(mux17_a),
             .b(mux17_b),
             .choose(mux17_choose),
             .z(mux17_z)
         );
         Mux mux18(
             .a(mux18_a),
             .b(mux18_b),
             .choose(mux18_choose),
             .z(mux18_z)
         );
```

```
Mux mux19(
    .a(mux19_a),
    .b(mux19_b),
    .choose(mux19_choose),
    .z(mux19_z)
);
Mux mux20(
       .a(mux20_a),
       .b(mux20_b),
       .choose(mux20_choose),
       .z(mux20_z)
);
   Mux mux21(
        .a(mux21_a),
        .b(mux21_b),
        .choose(mux21_choose),
        .z(mux21_z)
    );
    Mux mux22(
        .a(mux22_a),
        .b(mux22_b),
        .choose(mux22_choose),
        .z(mux22_z)
    );
    Mux mux23(
        .a(mux23_a),
        .b(mux23_b),
        .choose(mux23_choose),
        .z(mux23_z)
    );
    Mux mux24(
        .a(mux24_a),
        .b(mux24_b),
        .choose(mux24_choose),
        .z(mux24_z)
    );
    Mux mux25(
        .a(mux25_a),
        .b(mux25_b),
        .choose(mux25_choose),
        .z(mux25_z)
    );
    Mux mux26(
        .a(mux26_a),
                .b(mux26_b),
                .choose(mux26_choose),
                .z(mux26_z)
            );
            Mux mux27(
                .a(mux27_a),
                .b(mux27_b),
                .choose(mux27_choose),
                .z(mux27_z)
            );


    Mux_5 mux5_1(
        .a(mux5_1a),
        .b(mux5_1b),
        .choose(mux5_1choose),
        .z(mux5_1z)
    );
    Mux_5 mux5_2(
        .a(mux5_2a),
        .b(mux5_2b),
        .choose(mux5_2choose),
        .z(mux5_2z)
    );
    Mux_5 mux5_3(
        .a(mux5_3a),
        .b(mux5_3b),
        .choose(mux5_3choose),
        .z(mux5_3z)
    );

    Extend1 ext1(
        .a(ext1_a),
        .b(ext1_b)
    );
    Extend5 ext5(
        .a(ext5_a),
        .signed_ext(ext5_signed_ext),
        .b(ext5_b)
    );
    Extend16 ext16(
         .a(ext16_a),
         .signed_ext(ext16_signed_ext),
         //.left(ext16_left),
         .b(ext16_b)
    );
    Extend16 ext16_2(
         .a(ext16_2_a),
         .signed_ext(ext16_2_signed_ext),
         //.left(ext16_left),
         .b(ext16_2_b)
    );
    Extend8 ext8(
         .a(ext8_a),
         .signed_ext(ext8_signed_ext),
         //.left(ext16_left),
         .b(ext8_b)
    );
    Extend18 ext18(
         .a(ext18_a),
         .b(ext18_b)
    );
    II ii(
        .a(ii_a),
        .b(ii_b),
        .r(ii_r)
    );
    add add(
        .a(add_a),
        .b(add_b),
        .r(add_r),
        .overflow(add_overflow)
    );

    //mult
    MULT mult(
        .clk(clk),
        .reset(rst),
        .cs(1),
        .a(mult_a),
        .b(mult_b),
        .z(mult_z)
    );
    MULTU multu(
        .clk(clk),
        .reset(rst),
        .cs(1),
        .a(mult_a),
        .b(mult_b),
        .z(multu_z)
    );

    //div
    DIV div(
        .dividend(mult_a),
        .divisor(mult_b),
        .start(div_start),
        .clock(clk),
        .reset(rst),
        .q(div_q),
        .r(div_r),
        .busy(div_busy)
    );
    DIVU divu(
        .dividend(mult_a),
        .divisor(mult_b),
        .start(div_start),
        .clock(clk),
        .reset(rst),
        .q(divu_q),
        .r(divu_r),
        .busy(div_busy)
    );

    //cp0
    CP0 cp0(
      .clk(clk),
      .rst(rst),
      .mfc0(mfc0),
      .mtc0(mtc0),
      .eret(eret),
      .exception(exception),
      .cause(cause),
      .addr(cp0_addr),
      .data(cp0_data),
      .pc(epc),
      .rdata(cp0_rdata),
      .status(status),
      .exc_addr(exc_addr)
    );

    //clz
    CLZ32 clz (
      .rs(clz_rs),
      .rd(clz_rd)
    );

    //hilo
    HILO hilo(
        .clk(clk),
        .reset(rst),
        .mdc(mdc),
        .a(hilo_a),
        .b(hilo_b),
        .hi(hi),
        .lo(lo)
    );

endmodule
```

IMEM：

```
`timescale 1ns / 1ps

module IMEM(
    input [10:0] addr,
    output [31:0] instr
    );

    dist_mem_gen_0 instr_mem(
        .a(addr),
        .spo(instr)
    );
endmodule
```

DMEM：

```
`timescale 1ns / 1ps

module DMEM(
    input clk,
    input ena,//高电位有效

    input Dmem_W,//控制写
    input Dmem_R,//控制读
    input [10:0] Dmem_addr,//输入的地址
    input [31:0] Dmem_wdata,//写入的数据
    output [31:0] Dmem_rdata//读取的数据
    );

    reg [31:0] D_mem[0:1023];

    always @(posedge clk) begin
        if (Dmem_W && ena)
        begin
            D_mem[Dmem_addr] <= Dmem_wdata;
        end
    end

    assign Dmem_rdata = (Dmem_R && ena) ? D_mem[Dmem_addr] : 32'bz;//不可
读时改为高阻
endmodule
```

sccomp_dataflow：

```
`timescale 1ns / 1ps

module sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0] inst,
    output [31:0] pc
    );

    wire [31:0] instr;
    wire dw, dr, dena;
    wire [31:0] w_data, r_data;
    wire [10:0] dm_addr;
    wire [10:0] im_addr;

    assign inst = instr;
    assign im_addr = (pc - 32'h0040_0000) / 4;

    cpu_top sccpu(
        .clk(clk_in),
        .ena(1'b1),
        .rst(reset),

        .IM_inst(instr), //i ->
        .DM_rdata(r_data), //d ->

        .DM_ena(dena), //pc ->
        .DM_W(dw), //-> d
        .DM_R(dr), //-> d
        .addr(dm_addr),
        .DM_wdata(w_data), //-> d
        .PC_out(pc), //-> i
        .ALU_out(res) //-> d
    );

    IMEM imemory(
        .addr(im_addr),
        .instr(instr)
    );

    DMEM dmemory(
        .clk(clk_in),
        .ena(1'b1),

        .Dmem_W(dw),
        .Dmem_R(dr),

        .Dmem_addr(dm_addr),
        .Dmem_wdata(w_data),
        .Dmem_rdata(r_data)
    );
endmodule
```

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

test：

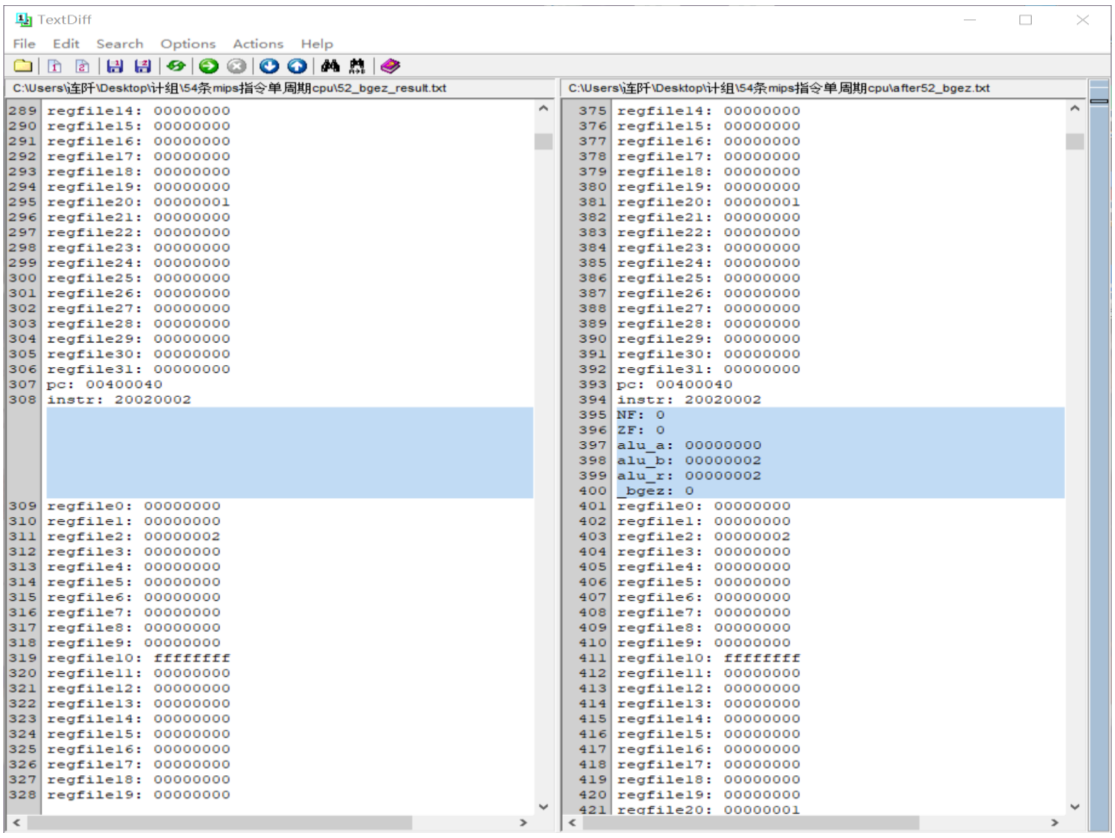
```
`timescale 100ps / 1ps

module test();

    reg clk_in, reset;
    wire [31:0] inst, pc;
    integer file_open;

    initial begin
        file_open = $fopen("D:/…/…/…/…/…/~.txt", "w");
        clk_in = 0;
        reset = 1;

        #2;
        reset = 0;
    end

    always begin
        #2 clk_in = ~clk_in;
        if(clk_in == 1'b1) begin
                        $fdisplay(file_open, "regfile0: %h",
test.uut.sccpu.cpu_ref.array_reg[0]);
                        $fdisplay(file_open, "regfile1: %h",
test.uut.sccpu.cpu_ref.array_reg[1]);
                        $fdisplay(file_open, "regfile2: %h",
test.uut.sccpu.cpu_ref.array_reg[2]);
                        $fdisplay(file_open, "regfile3: %h",
test.uut.sccpu.cpu_ref.array_reg[3]);
                        $fdisplay(file_open, "regfile4: %h",
test.uut.sccpu.cpu_ref.array_reg[4]);
                        $fdisplay(file_open, "regfile5: %h",
test.uut.sccpu.cpu_ref.array_reg[5]);
                        $fdisplay(file_open, "regfile6: %h",
test.uut.sccpu.cpu_ref.array_reg[6]);
                        $fdisplay(file_open, "regfile7: %h",
test.uut.sccpu.cpu_ref.array_reg[7]);
                        $fdisplay(file_open, "regfile8: %h",
test.uut.sccpu.cpu_ref.array_reg[8]);
                        $fdisplay(file_open, "regfile9: %h",
test.uut.sccpu.cpu_ref.array_reg[9]);
                        $fdisplay(file_open, "regfile10: %h",
test.uut.sccpu.cpu_ref.array_reg[10]);
                        $fdisplay(file_open, "regfile11: %h",
test.uut.sccpu.cpu_ref.array_reg[11]);
                        $fdisplay(file_open, "regfile12: %h",
test.uut.sccpu.cpu_ref.array_reg[12]);
                        $fdisplay(file_open, "regfile13: %h",
test.uut.sccpu.cpu_ref.array_reg[13]);
                        $fdisplay(file_open, "regfile14: %h",
test.uut.sccpu.cpu_ref.array_reg[14]);
                        $fdisplay(file_open, "regfile15: %h",
test.uut.sccpu.cpu_ref.array_reg[15]);
                        $fdisplay(file_open, "regfile16: %h",
test.uut.sccpu.cpu_ref.array_reg[16]);
                        $fdisplay(file_open, "regfile17: %h",
test.uut.sccpu.cpu_ref.array_reg[17]);
```

```
                        $fdisplay(file_open, "regfile18: %h",
test.uut.sccpu.cpu_ref.array_reg[18]);
                        $fdisplay(file_open, "regfile19: %h",
test.uut.sccpu.cpu_ref.array_reg[19]);
                        $fdisplay(file_open, "regfile20: %h",
test.uut.sccpu.cpu_ref.array_reg[20]);
                        $fdisplay(file_open, "regfile21: %h",
test.uut.sccpu.cpu_ref.array_reg[21]);
                        $fdisplay(file_open, "regfile22: %h",
test.uut.sccpu.cpu_ref.array_reg[22]);
                        $fdisplay(file_open, "regfile23: %h",
test.uut.sccpu.cpu_ref.array_reg[23]);
                        $fdisplay(file_open, "regfile24: %h",
test.uut.sccpu.cpu_ref.array_reg[24]);
                        $fdisplay(file_open, "regfile25: %h",
test.uut.sccpu.cpu_ref.array_reg[25]);
                        $fdisplay(file_open, "regfile26: %h",
test.uut.sccpu.cpu_ref.array_reg[26]);
                        $fdisplay(file_open, "regfile27: %h",
test.uut.sccpu.cpu_ref.array_reg[27]);
                        $fdisplay(file_open, "regfile28: %h",
test.uut.sccpu.cpu_ref.array_reg[28]);
                        $fdisplay(file_open, "regfile29: %h",
test.uut.sccpu.cpu_ref.array_reg[29]);
                        $fdisplay(file_open, "regfile30: %h",
test.uut.sccpu.cpu_ref.array_reg[30]);
                        $fdisplay(file_open, "regfile31: %h",
test.uut.sccpu.cpu_ref.array_reg[31]);
                        $fdisplay(file_open, "pc: %h", pc);
                        $fdisplay(file_open, "instr: %h", inst);
        end;
    end

    sccomp_dataflow uut(
        .clk_in(clk_in),
        .reset(reset),
        .inst(inst),
        .pc(pc)
    );

endmodule
```

五、实验结果

前仿真：

比对通过：



TextDiff

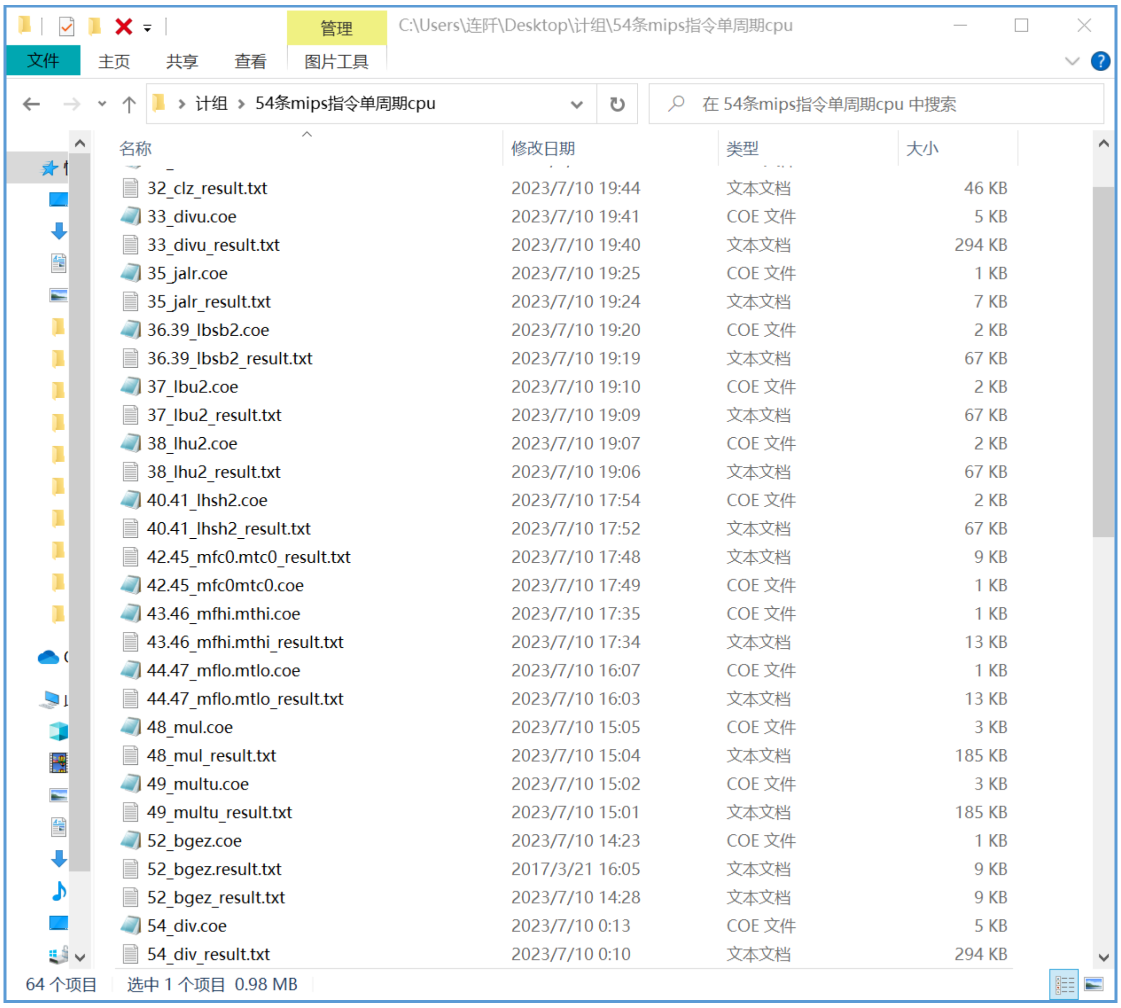
File Edit Search Options Actions Help

C:\Users\连阡\Desktop\计组\54条mips指令单周期cpu\52_bgez_result.txt

```
regfile14: 00000000
regfile15: 00000000
regfile16: 00000000
regfile17: 00000000
regfile18: 00000000
regfile19: 00000000
regfile20: 00000001
regfile21: 00000000
regfile22: 00000000
regfile23: 00000000
regfile24: 00000000
regfile25: 00000000
regfile26: 00000000
regfile27: 00000000
regfile28: 00000000
regfile29: 00000000
regfile30: 00000000
regfile31: 00000000
pc: 00400040
instr: 20020002






regfile0: 00000000
regfile1: 00000000
regfile2: 00000002
regfile3: 00000000
regfile4: 00000000
regfile5: 00000000
regfile6: 00000000
regfile7: 00000000
regfile8: 00000000
regfile9: 00000000
regfile10: ffffffff
regfile11: 00000000
regfile12: 00000000
regfile13: 00000000
regfile14: 00000000
regfile15: 00000000
regfile16: 00000000
regfile17: 00000000
regfile18: 00000000
regfile19: 00000000
```

C:\Users\连阡\Desktop\计组\54条mips指令单周期cpu\after52_bgez.txt

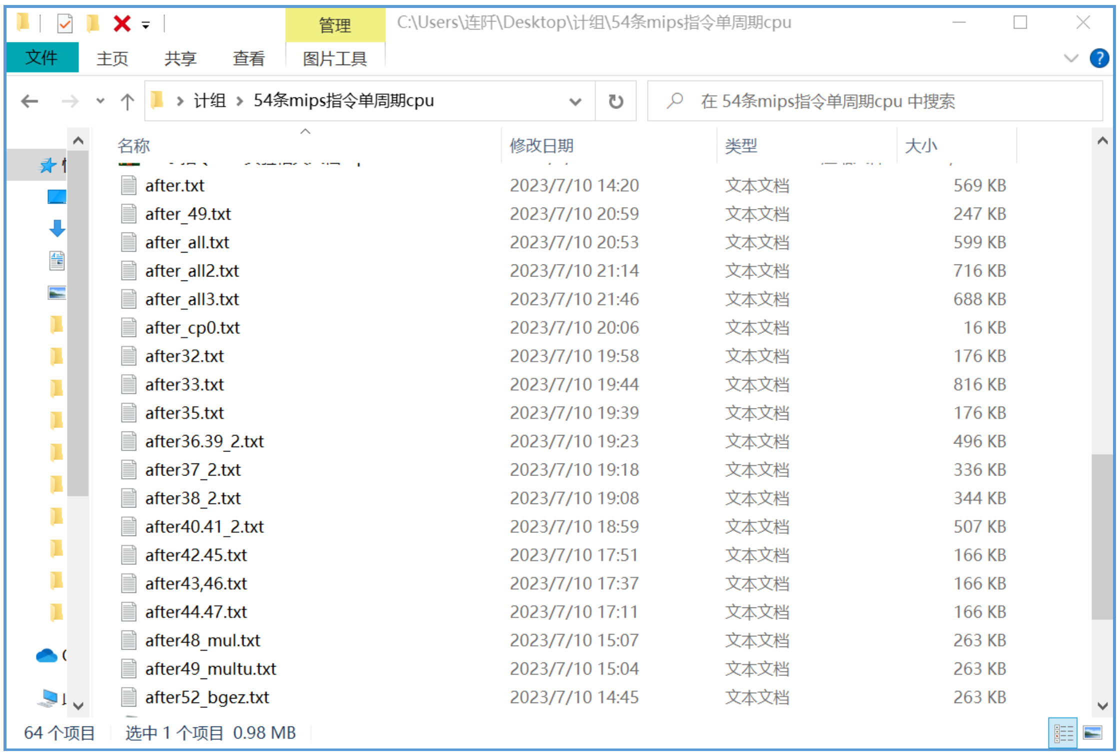
```
regfile14: 00000000
regfile15: 00000000
regfile16: 00000000
regfile17: 00000000
regfile18: 00000000
regfile19: 00000000
regfile20: 00000001
regfile21: 00000000
regfile22: 00000000
regfile23: 00000000
regfile24: 00000000
regfile25: 00000000
regfile26: 00000000
regfile27: 00000000
regfile28: 00000000
regfile29: 00000000
regfile30: 00000000
regfile31: 00000000
pc: 00400040
instr: 20020002
NF: 0
ZF: 0
alu_a: 00000000
alu_b: 00000002
alu_r: 00000002
_bgez: 0
regfile0: 00000000
regfile1: 00000000
regfile2: 00000002
regfile3: 00000000
regfile4: 00000000
regfile5: 00000000
regfile6: 00000000
regfile7: 00000000
regfile8: 00000000
regfile9: 00000000
regfile10: ffffffff
regfile11: 00000000
regfile12: 00000000
regfile13: 00000000
regfile14: 00000000
regfile15: 00000000
regfile16: 00000000
regfile17: 00000000
regfile18: 00000000
regfile19: 00000000
regfile20: 00000001
```



C:\Users\连阡\Desktop\计组\54条mips指令单周期cpu

名称	修改日期	类型	大小
32_clz_result.txt	2023/7/10 19:44	文本文档	46 KB
33_divu.coe	2023/7/10 19:41	COE 文件	5 KB
33_divu_result.txt	2023/7/10 19:40	文本文档	294 KB
35_jalr.coe	2023/7/10 19:25	COE 文件	1 KB
35_jalr_result.txt	2023/7/10 19:24	文本文档	7 KB
36.39_lbsb2.coe	2023/7/10 19:20	COE 文件	2 KB
36.39_lbsb2_result.txt	2023/7/10 19:19	文本文档	67 KB
37_lbu2.coe	2023/7/10 19:10	COE 文件	2 KB
37_lbu2_result.txt	2023/7/10 19:09	文本文档	67 KB
38_lhu2.coe	2023/7/10 19:07	COE 文件	2 KB
38_lhu2_result.txt	2023/7/10 19:06	文本文档	67 KB
40.41_lhsh2.coe	2023/7/10 17:54	COE 文件	2 KB
40.41_lhsh2_result.txt	2023/7/10 17:52	文本文档	67 KB
42.45_mfc0.mtc0_result.txt	2023/7/10 17:48	文本文档	9 KB
42.45_mfc0mtc0.coe	2023/7/10 17:49	COE 文件	1 KB
43.46_mfhi.mthi.coe	2023/7/10 17:35	COE 文件	1 KB
43.46_mfhi.mthi_result.txt	2023/7/10 17:34	文本文档	13 KB
44.47_mflo.mtlo.coe	2023/7/10 16:07	COE 文件	1 KB
44.47_mflo.mtlo_result.txt	2023/7/10 16:03	文本文档	13 KB
48_mul.coe	2023/7/10 15:05	COE 文件	3 KB
48_mul_result.txt	2023/7/10 15:04	文本文档	185 KB
49_multu.coe	2023/7/10 15:02	COE 文件	3 KB
49_multu_result.txt	2023/7/10 15:01	文本文档	185 KB
52_bgez.coe	2023/7/10 14:23	COE 文件	1 KB
52_bgez.result.txt	2017/3/21 16:05	文本文档	9 KB
52_bgez_result.txt	2023/7/10 14:28	文本文档	9 KB
54_div.coe	2023/7/10 0:13	COE 文件	5 KB
54_div_result.txt	2023/7/10 0:10	文本文档	294 KB

64 个项目 | 选中 1 个项目 0.98 MB

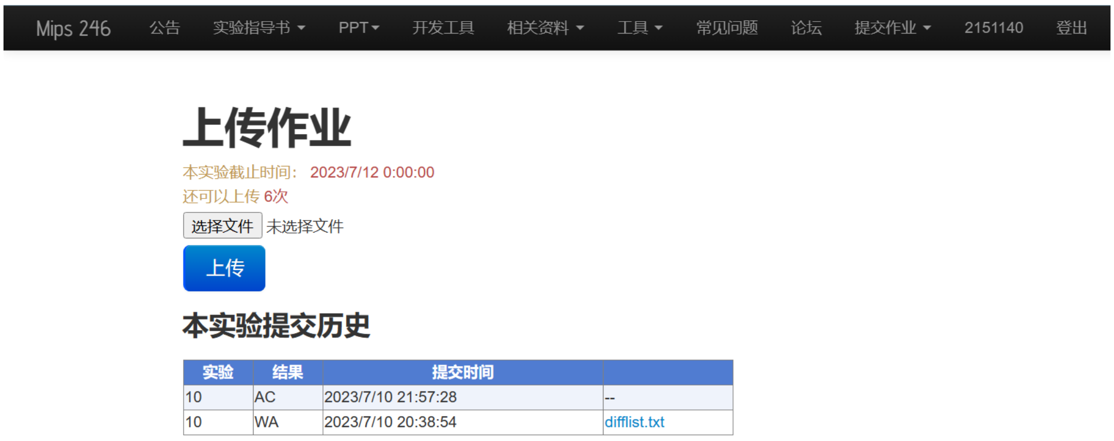


C:\Users\连阡\Desktop\计组\54条mips指令单周期cpu

名称	修改日期	类型	大小
after.txt	2023/7/10 14:20	文本文档	569 KB
after_49.txt	2023/7/10 20:59	文本文档	247 KB
after_all.txt	2023/7/10 20:53	文本文档	599 KB
after_all2.txt	2023/7/10 21:14	文本文档	716 KB
after_all3.txt	2023/7/10 21:46	文本文档	688 KB
after_cp0.txt	2023/7/10 20:06	文本文档	16 KB
after32.txt	2023/7/10 19:58	文本文档	176 KB
after33.txt	2023/7/10 19:44	文本文档	816 KB
after35.txt	2023/7/10 19:39	文本文档	176 KB
after36.39_2.txt	2023/7/10 19:23	文本文档	496 KB
after37_2.txt	2023/7/10 19:18	文本文档	336 KB
after38_2.txt	2023/7/10 19:08	文本文档	344 KB
after40.41_2.txt	2023/7/10 18:59	文本文档	507 KB
after42.45.txt	2023/7/10 17:51	文本文档	166 KB
after43,46.txt	2023/7/10 17:37	文本文档	166 KB
after44.47.txt	2023/7/10 17:11	文本文档	166 KB
after48_mul.txt	2023/7/10 15:07	文本文档	263 KB
after49_multu.txt	2023/7/10 15:04	文本文档	263 KB
after52_bgez.txt	2023/7/10 14:45	文本文档	263 KB

64 个项目 | 选中 1 个项目 0.98 MB

网站提交结果 AC：



Mips 246 公告 实验指导书 PPT 开发工具 相关资料 工具 常见问题 论坛 提交作业 2151140 登出

上传作业

本实验截止时间：2023/7/12 0:00:00

还可以上传 6次

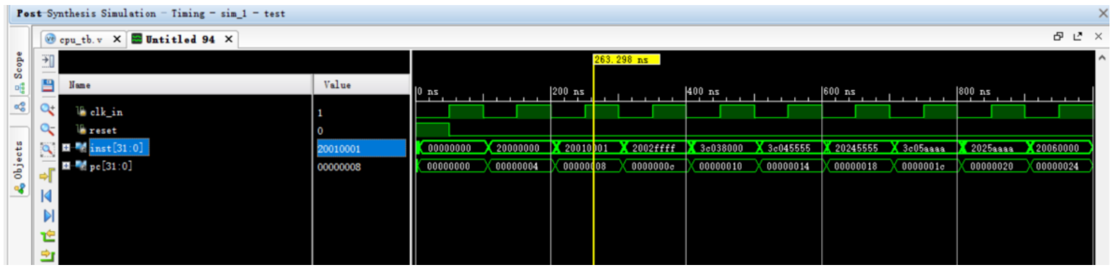
选择文件 未选择文件

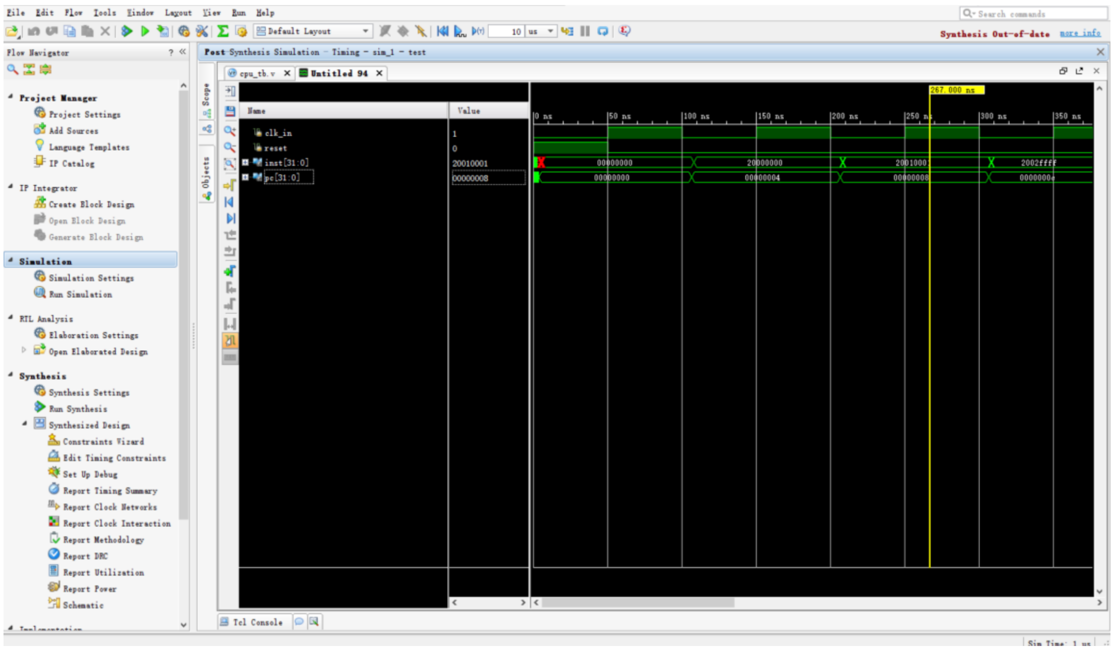
上传

本实验提交历史

实验	结果	提交时间	
10	AC	2023/7/10 21:57:28	--
10	WA	2023/7/10 20:38:54	difflist.txt

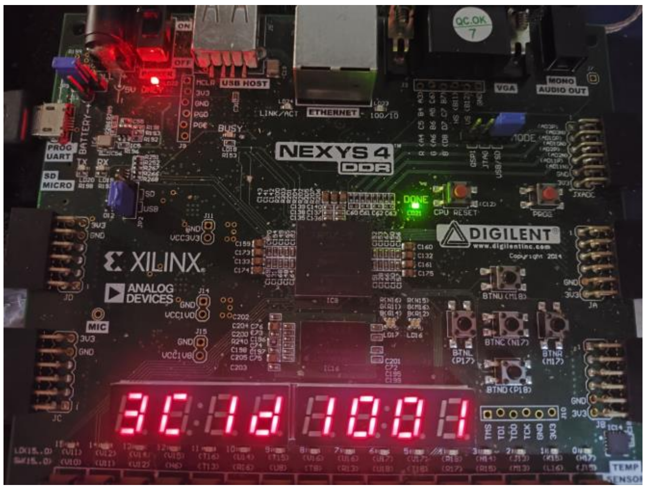
后仿真：

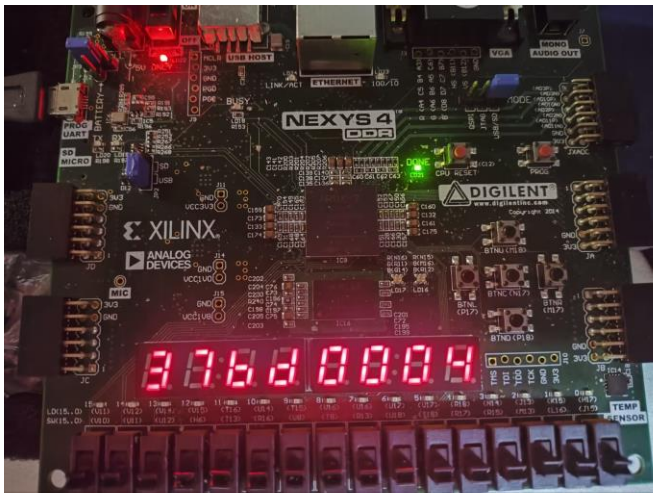

Post-Synthesis Simulation - Timing - sim_1 - test
cpu_tb.v
Untitled 94
263.298 ns
Name
Value
clk_in 1
reset 0
inst[31:0] 20010001
pc[31:0] 00000008
0 ns
200 ns
400 ns
600 ns
800 ns
00000000 20000000 20010001 2002ffff 3c038000 3c045555 20245555 3c05aaaa 2025aaaa 20060000
00000000 00000004 00000008 0000000c 00000010 00000014 00000018 0000001c 00000020 00000024

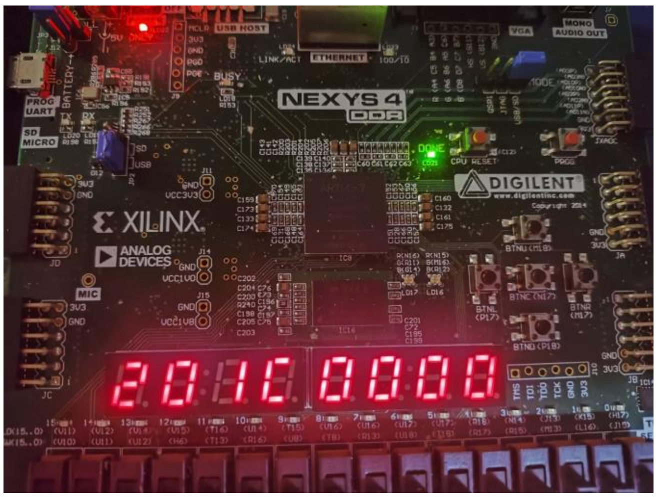


下板：

```
3c1d1001
37bd0004
201c0000
3c1b1001
0c100413
00000000
20010002
20020002
```


NEXYS 4 DDR
DIGILENT
XILINX
ANALOG DEVICES
3C1d1001


NEXYS 4 DDR
DIGILENT
XILINX
ANALOG DEVICES
37bd0004


NEXYS 4 DDR
DIGILENT
XILINX
ANALOG DEVICES
201C0000