同济大学计算机系

计算机组成原理实验报告

学　　号 1952650

姓　　名 陈子翔

专　　业 信息安全

授课老师 郝老师

一、实验内容

使用 Verilog HDL 语言实现 54 条 MIPS 指令的 CPU 的设计和仿真，使其实现以下指令。

序号	31条	ucosii V2.52	指令	指令说明	指令格式	OP 31-26	RS 25-21	RT 20-16	RD 15-11	SA 10-6	FUNCT 5-0	指令码 16进制
1	√	√	addi	加立即数	addi rt, rs, immediate	001000				00000	100000	20000000
2	√	√	addiu	加立即数（无符号）	addiu rd, rs, immediate	001001						24000000
3	√	√	andi	立即数与	andi rt, rs, immediate	001100						30000000
4	√	√	ori	或立即数	ori rt, rs, immediate	001101						34000000
5	√	√	sltiu	小于立即数置 1（无符号）	sltiu rt, rs, immediate	001011						2C000000
6	√	√	lui	立即数加载高位	lui rt, immediate	001111	00000					3C000000
7	√		xori	异或（立即数）	xori rt, rs, immediate	001110			00000	00000	000000	38000000
8	√		slti	小于置 1（立即数）	slti rt, rs, immediate	001010			00000	00000	000000	28000000
9	√	√	addu	加（无符号）	addu rd, rs, rt	000000				00000	100001	00000021
10	√	√	and	与	and rd, rs, rt	000000				00000	100100	00000024
11	√	√	beq	相等时分支	beq rs, rt, offset	000100						10000000
12	√	√	bne	不等时分支	bne rs, rt, offset	000101						14000000
13	√	√	j	跳转	j target	000010						08000000
14	√	√	jal	跳转并链接	jal target	000011						0C000000
15	√	√	jr	跳转至寄存器所指地址	jr rs	000000					001000	00000009
16	√	√	lw	取字	lw rt, offset(base)	100011						8C000000
17	√	√	xor	异或	xor rd, rs, rt	000000				00000	100110	00000026
18	√	√	nor	或非	nor rd, rs, rt	000000				00000	100111	00000027
19	√	√	or	或	or rd, rs, rt	000000				00000	100101	00000025
20	√	√	sll	逻辑左移	sll rd, rt, sa	000000	00000				000000	00000000
21	√	√	sllv	逻辑左移（位数可变）	sllv rd, rt, rs	000000				00000	000100	00000004
22	√	√	sltu	小于置 1（无符号）	sltu rd, rs, rt	000000				00000	101011	0000002B
23	√	√	sra	算数右移	sra rd, rt, sa	000000	00000				000011	00000003
24	√	√	srl	逻辑右移	srl rd, rt, sa	000000	00000				000010	00000002
25	√	√	subu	减（无符号）	sub rd, rs, rt	000000				00000	100010	00000022
26	√	√	sw	存字	sw rt, offset(base)	101011						AC000000
27	√		add	加	add rd, rs, rt	000000				00000	100000	00000020
28	√		sub	减	sub rd, rs, rt	000000				00000	100010	00000022
29	√		slt	小于置 1	slt rd, rs, rt	000000				00000	101010	0000002A
30	√		srlv	逻辑右移（位数可变）	srlv rd, rt, rs	000000				00000	000110	00000006
31	√		srav	算数右移（位数可变）	srav rd, rt, rs	000000				00000	000111	00000007
32		√	clz	前导零计数	clz rd, rs	011100				00000	100000	70000020
33		√	divu	除（无符号）	divu rs, rt	000000			00000	00000	011011	0000001B
34		√	eret	异常返回	eret	010000	10000	00000	00000	00000	011000	42000018
35		√	jalr	跳转至寄存器所指地址，返回地址保存在	jalr rs	000000		00000			001001	00000008
36		√	lb	取字节	lb rt, offset(base)	100000						80000000
37		√	lbu	取字节（无符号）	lbu rt, offset(base)	100100						90000000

38		√	lhu	取半字（无符号）	lhu rt, offset(base)	100101						94000000
39		√	sb	存字节	sb rt, offset(base)	101000						A0000000
40		√	sh	存半字	sh rt, offset(base)	101001						A4000000
41			lh	取半字	lh rt, offset(base)	100001						84000000
42		√	mfc0	读 CP0 寄存器	mfc0 rt, rd	010000	00000			00000	000000	40000000
43		√	mfhi	读 Hi 寄存器	mfhi rd	000000	00000	00000		00000	010000	00000010
44		√	mflo	读 Lo 寄存器	mflo rd	000000	00000	00000		00000	010010	00000012
45		√	mtc0	写 CP0 寄存器	mtc0 rt, rd	010000	00100			00000	000000	40800000
46		√	mthi	写 Hi 寄存器	mthi rd	000000		00000	00000	00000	010001	00000011
47		√	mtlo	写 Lo 寄存器	mtlo rd	000000		00000	00000	00000	010011	00000013
48		√	mul	乘	mul rd, rs, rt	011100				00000	000010	70000002
49		√	multu	乘（无符号）	multu rs, rt	000000			00000	00000	011001	00000019
50		√	syscall	系统调用	syscall	000000					001100	0000000C
51		√	teq	相等异常	teq rs, rt	000000					110100	00000034
52		√	bgez	大于等于 0 时分支	bgez rs, offset	000001		0001				04010000
53			break	断点	break	000000					001101	0000000D
54			div	除	div rs, rt	000000			00000	00000	011010	0000001A

二、实验目的

1. 深入掌握 CPU 的构成及工作原理。
2. 设计 54 条指令的 CPU 的数据通路及控制器。
3. 使用 Verilog HDL 设计实现 54 条指令的 CPU 下板运行。

三、实验原理

3.1 实验原理

中央处理器（CPU）由运算器和控制器组成。其中，控制器的功能是负责协调并控制计算机各部件执行程序的指令序列，包括取指令、分析指令和执行指令；运算器的功能是对数据进行加工。CPU 的具体功能包括：

1）指令控制。完成取指令、分析指令和执行指令的操作，即程序的顺序控制。

2）操作控制。一条指令的功能往往由若干操作信号的组合来实现。CPU 管理并产生由内存取出的每条指令的操作信号，把各种操作信号送往相应的部件，从而控制这些部件按指令的要求进行动作。

3）时间控制。对各种操作加以时间上的控制。时间控制要为每条指令按时间顺序提供应有的控制信号。

4）数据加工。对数据进行算术和逻辑运算。

5）中断处理。对计算机运行过程中出现的异常情况和特殊请求进行处理。

单周期 CPU 是对所有指令都选用相同的执行时间来完成，称为单指令周期方案。此时每条指令都在固定的时钟周期内完成，指令之间串行执行，即下一条指令只能在前一条指令执行结束后才能启动。在单周期处理器中，一条指令执行过程中数据通路的任何资源都不能被重复使用，因此，任何需要被多次使用的资源都需要设置多个。

CPU 在执行指令的不同阶段所涉及的数据流有所不同，可分为以下几个步骤：

（1） 取指周期：根据程序计数器 PC 中的内容从主存中取出一条指令并存放在指令寄存器 IR 中。PC 存放的是指令的地址，根据此地址从内存单元中取出的是指令，取指令的同时，PC 通过自增到存储下一条指令的地址。

（2） 指令译码：对在取指周期中得到的指令进行分析并译码，确定这条指令需

要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

（3） 间址周期：取操作数有效地址，将指令中的地址码送到地址寄存器 MAR 并送至地址总线，此后控制单元 CU 向存储器发读命令，以获取有效地址并存至数据寄存器 MDR。

（4） 执行周期：根据指令寄存器 IR 中的指令字的操作码核操作数通过算数逻辑单元 ALU 操作产生执行结果。最后根据指令要求将执行结果或访问存储器得到的数据写回相应的寄存器。

3.2 指令格式

一条指令就是机器语言的一个语句，它是一组有意义的二进制代码。一条指令通常包括操作码字段和地址码字段两部分：

操作码字段	地址码字段

其中，操作码字段指出指令中该指令应该执行什么性质的操作和具有何种功能。操作码是识别指令、了解指令功能及区分操作数地址内容的组成和使用方法等的关键信息。

MIPS 框架下的 CPU 具有以下三种指令类型：

1. R-type

Bit #	31..26	25..21	20..16	15..11	10..6	5..0	
R-type	op	rs	rt	rd	shamt	func	

2. I-type

Bit #	31..26	25..21	20..16	15..0	
I-type	op	rs	rt	immediate	

3. J-type

Bit #	31..26	25..0	
J-type	op	Index	

其中：

op:31 位~26 位,6 位操作码,决定执行何种指令,R 型指令的操作码均为 000000，其余指令操作码各不相同。

rs：5 位地址码，代表第一个源操作数地址，只读。

rt：5 位地址码，代表第二个源操作数地址，可读可写。

rd：5 位地址码，代表目的操作数地址，只写。

shamt：5 位数据，专用于移位指令中代表位移量。

func：6 位功能码，与操作码配合使用，在 R 类指令中，操作码均为 0，用不同的功能码区分执行何种指令。

immediate：16 位立即数，用作无符号的逻辑操作数、有符号的算数操作数、数据加载/数据保存指令的数据地址字节偏移量和分支指令中相对程序计数器的有符号偏移量。

index：26 位地址码，用于 J 类指令中作 pc 跳转的地址。

3.3 CP0 中断

除了基本的运算功能，任何处理器均需要一些部件处理中断、配置选项以及需要某种机制来监控诸如高速缓存和定时器等功能。在 MIPS CPU 中，异常或中断发生的行为以及怎样处理都是由 CP0 控制寄存器和几条特殊指令来定义和控制的。CP0 中有一系列寄存器来完成异常控制、缓存控制等功能。本 CPU 实验需用到部分寄存器：

status：12 号寄存器

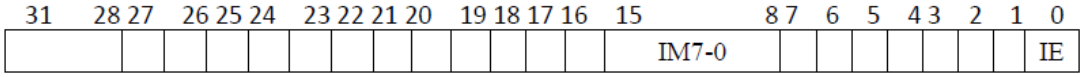


图 1 MIPS status 寄存器

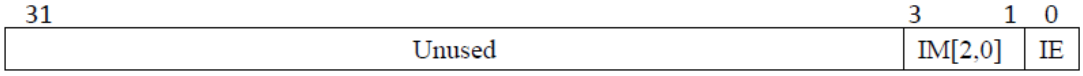


图 2 实现的 status 寄存器

status[0]‘IE’为中断禁止位，标准的 MIPS CP0 中 status 寄存器的 8 到 15 位为中断屏蔽位，如图 1，本实验中将 status[3:1]定为中断屏蔽位，status[1] 屏蔽 systcall, status[2]屏蔽 break，status[3]屏蔽 teq，如图 2。置 0 时表示禁止或屏蔽中断，我们不实现中断嵌套，所以相应中断异常时把 Status 寄存器的内容左移 5 位关中断，中断返回时再将 Status 寄存器右移 5 位恢复其内容。

cause：13 号寄存器

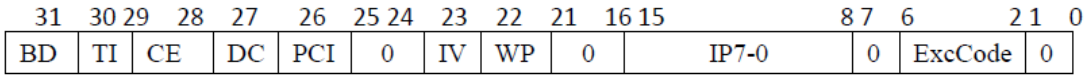


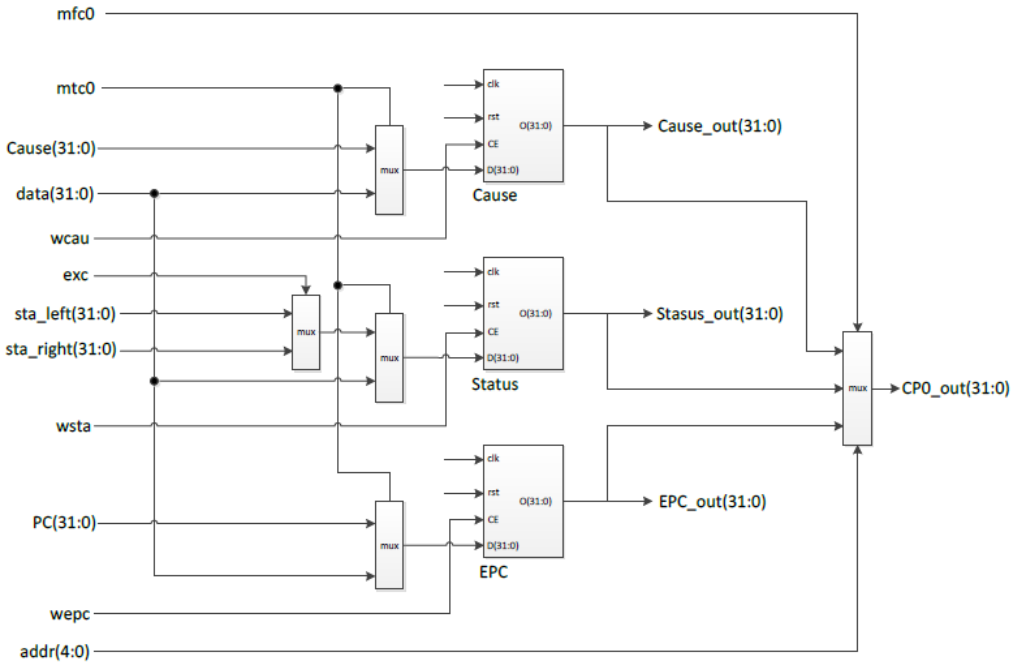
图 3 MIPS cause 寄存器

Cause 寄存器用于存放异常原因，MIPS32 架构的 cause 寄存器如图 3 所示，实验中仅用到 cause[6:2]：异常类型号 ‘ExcCode’，1000 为 systcall 异常，1001 为 break，1101 为 teq。

epc：14 号寄存器

异常发生时 epc 存放当前指令地址作为返回地址。

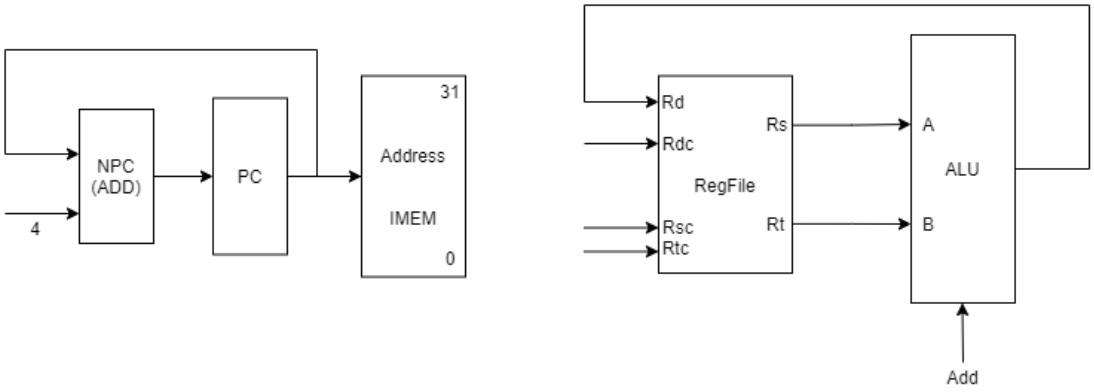
CP0 数据通路图如下：



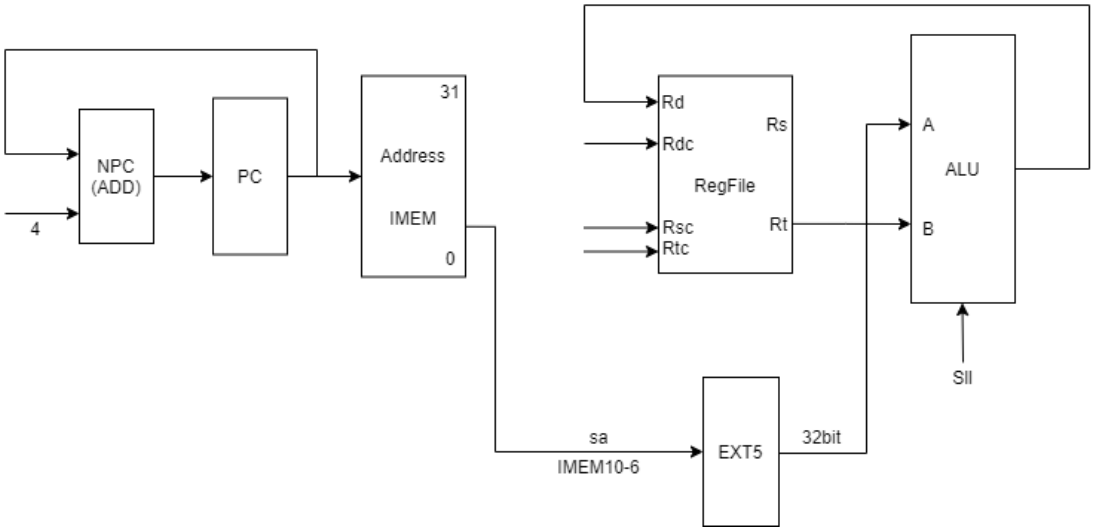
3.4 数据通路设计

对于相似的指令，相同的数据通路图不再重复画出。

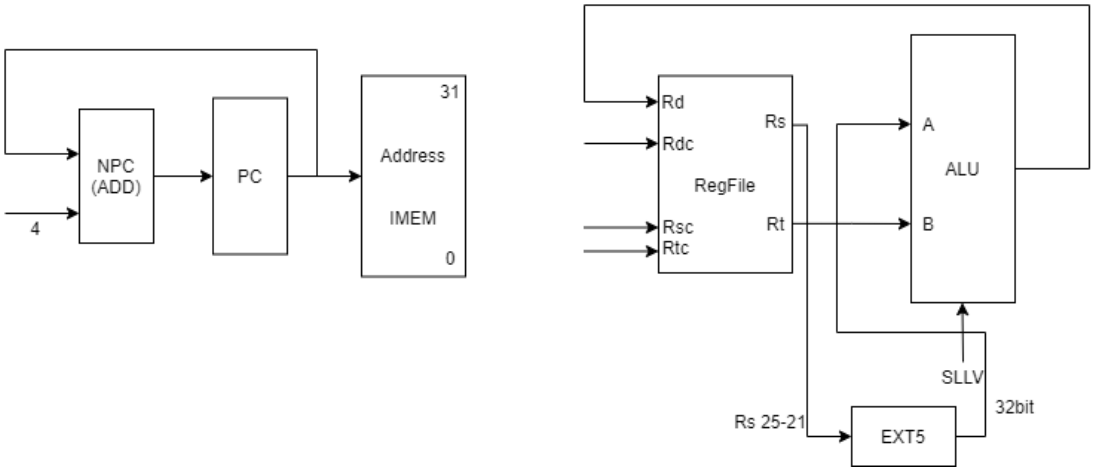
1. ADD/ADDU/SUB/SUBU/AND/OR/XOR/NOR/SLT/SLTU



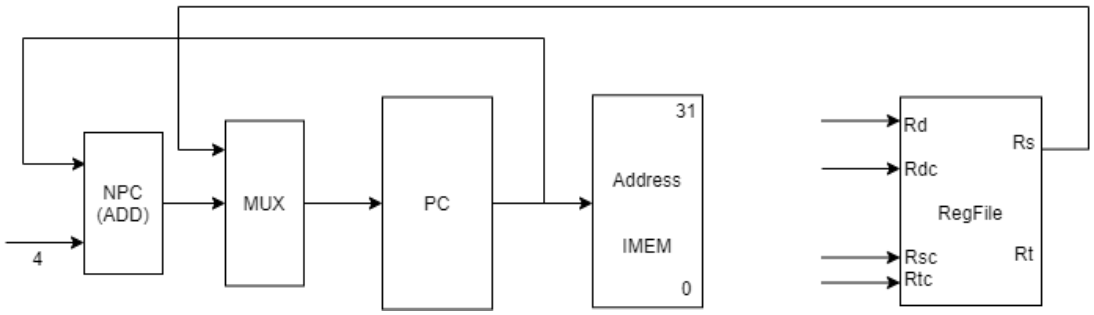
2. SLL/SRL/SRA



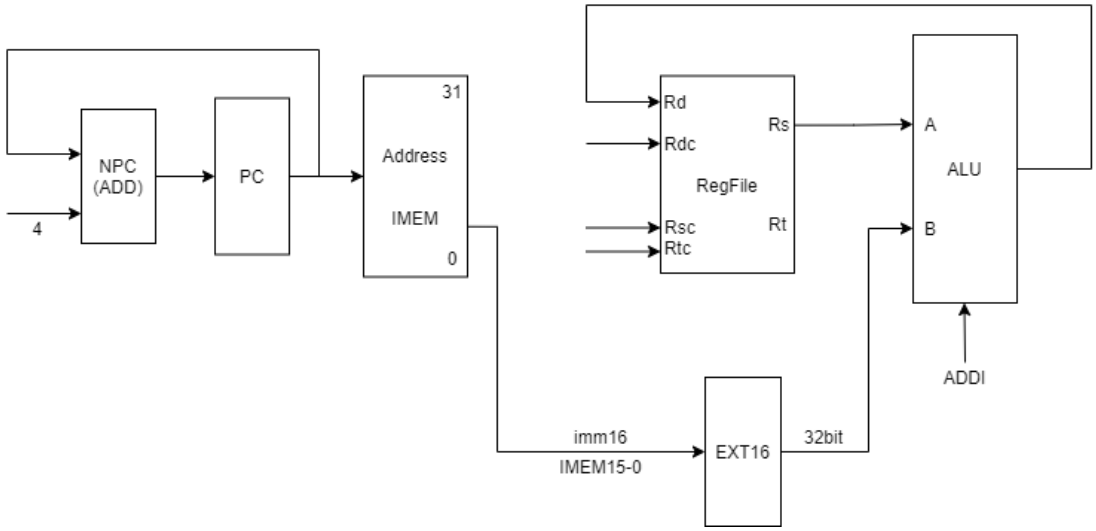
3. SLLV/SRLV/SRAV



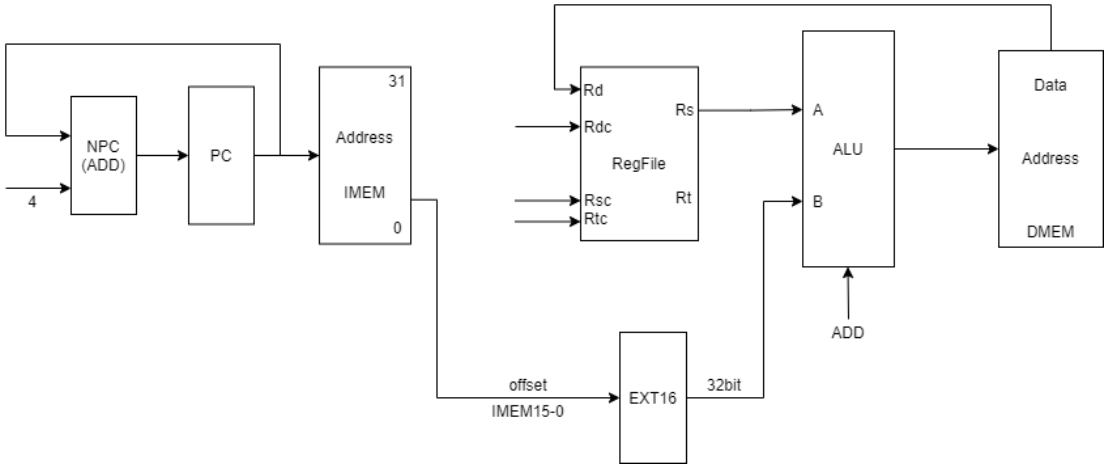
4. JR



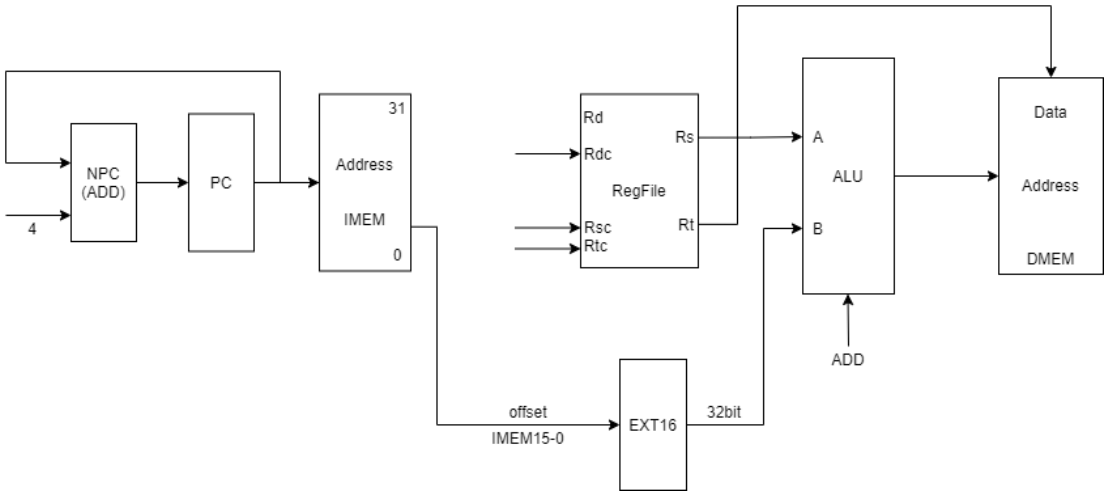
5. ADDI/ADDIU/ANDI/ORI/XORI



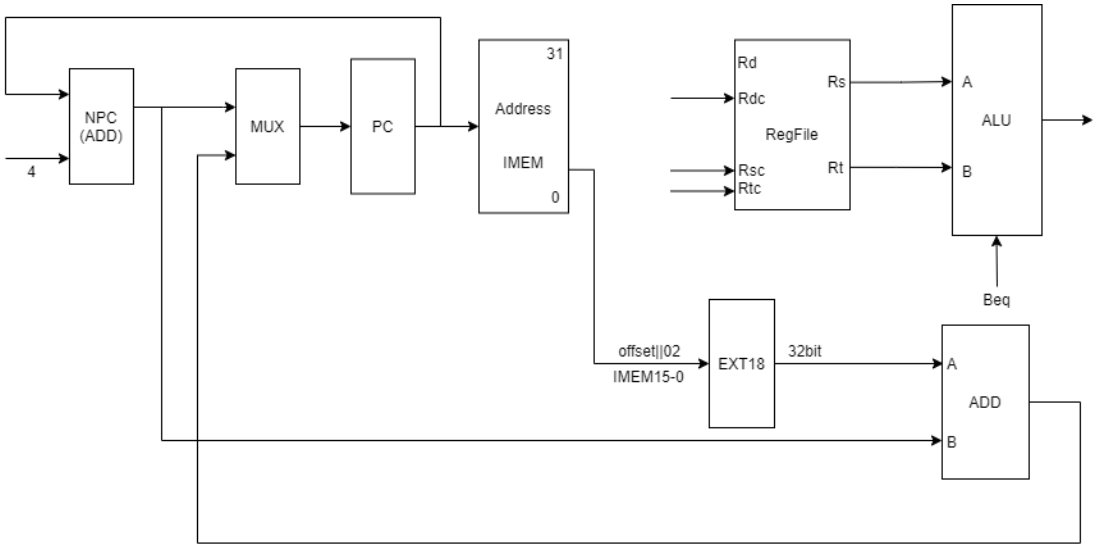
6. LW



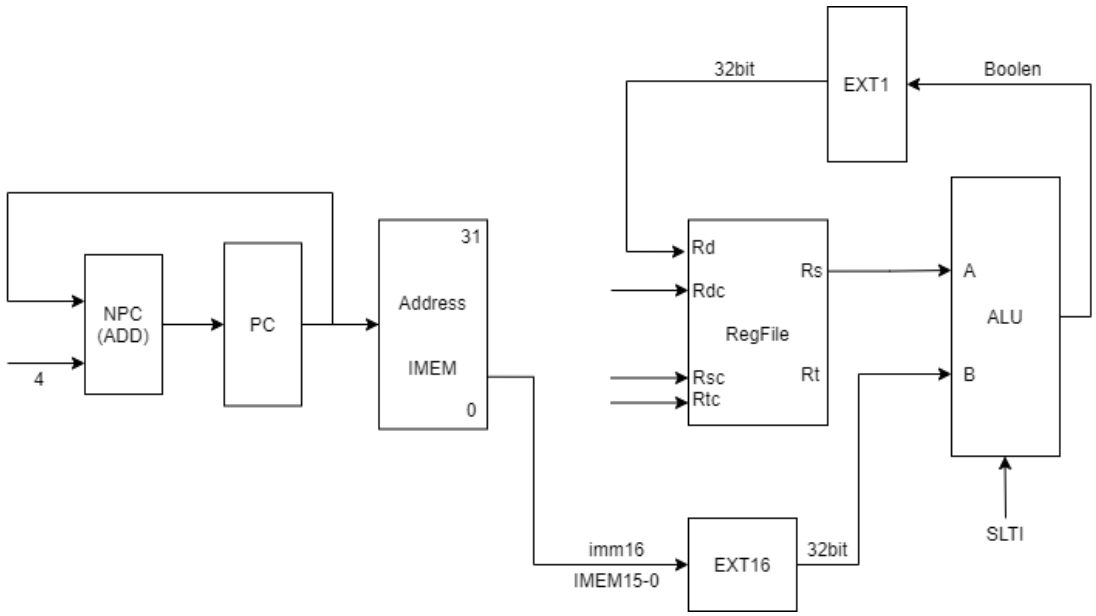
7. SW



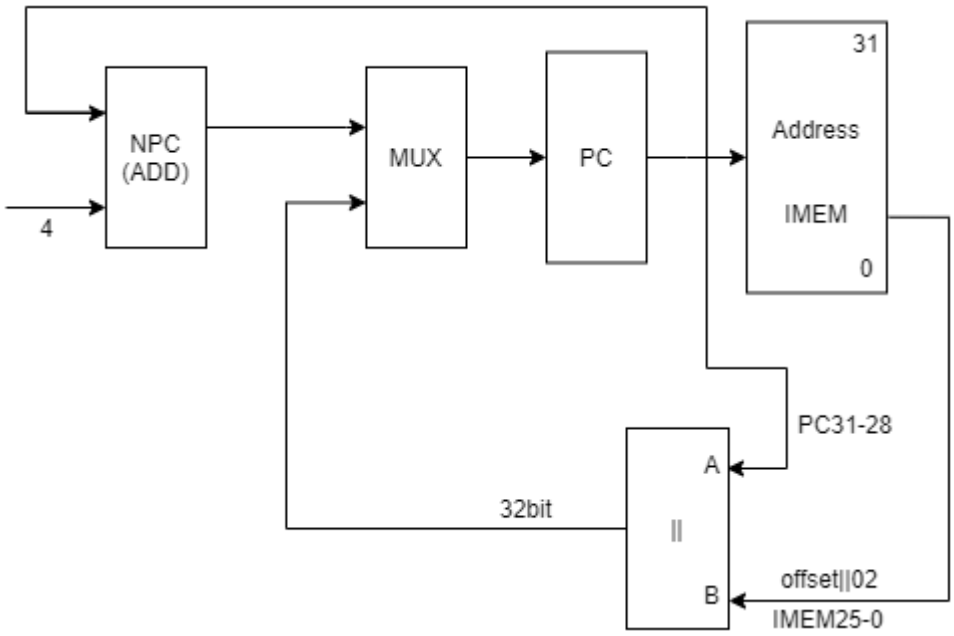
8. BEQ/BNE



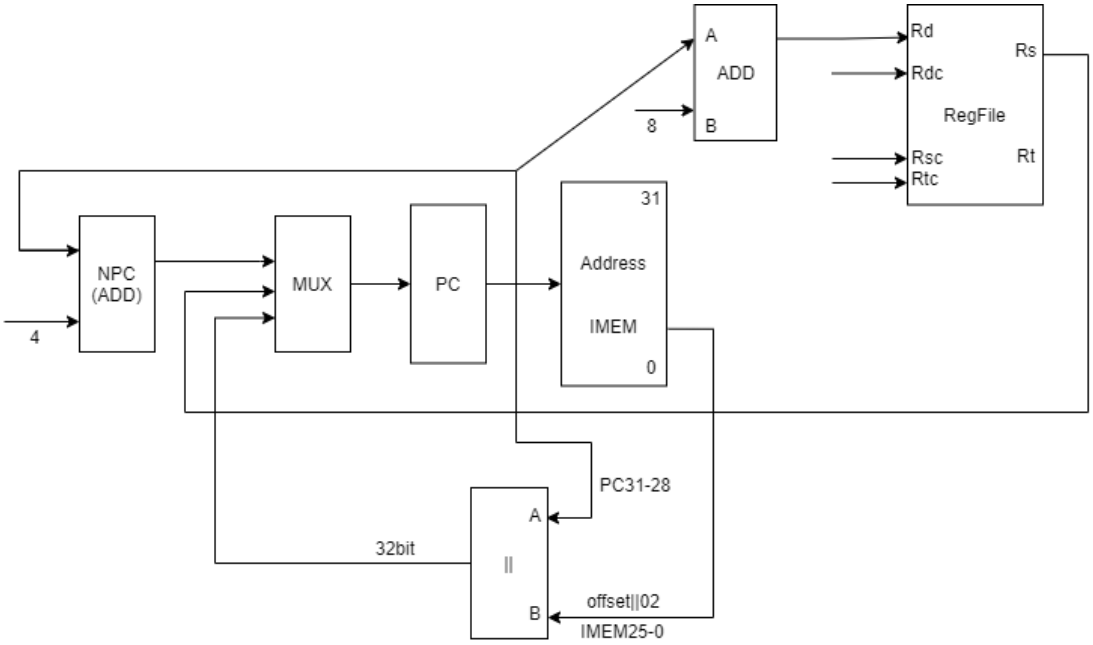
9. SLTI/SLTIU/LUI



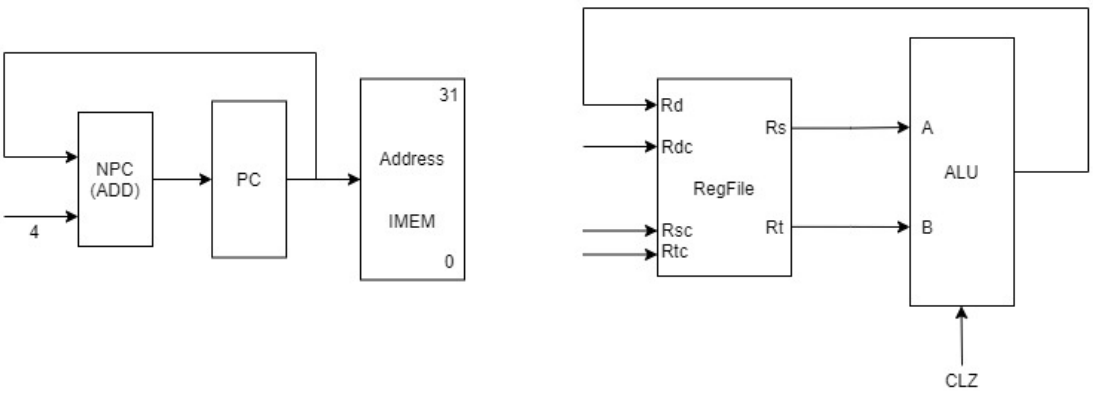
10. J


NPC
(ADD)
4
MUX
PC
31
Address
IMEM
0
PC31-28
A
||
B
32bit
offset||02
IMEM25-0

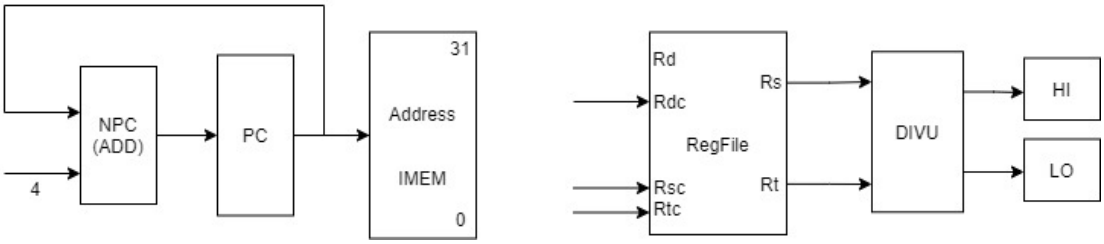
11. JAL


A
ADD
8
B
Rd
Rs
Rdc
RegFile
Rsc
Rt
Rtc
NPC
(ADD)
4
MUX
PC
31
Address
IMEM
0
PC31-28
A
||
B
32bit
offset||02
IMEM25-0

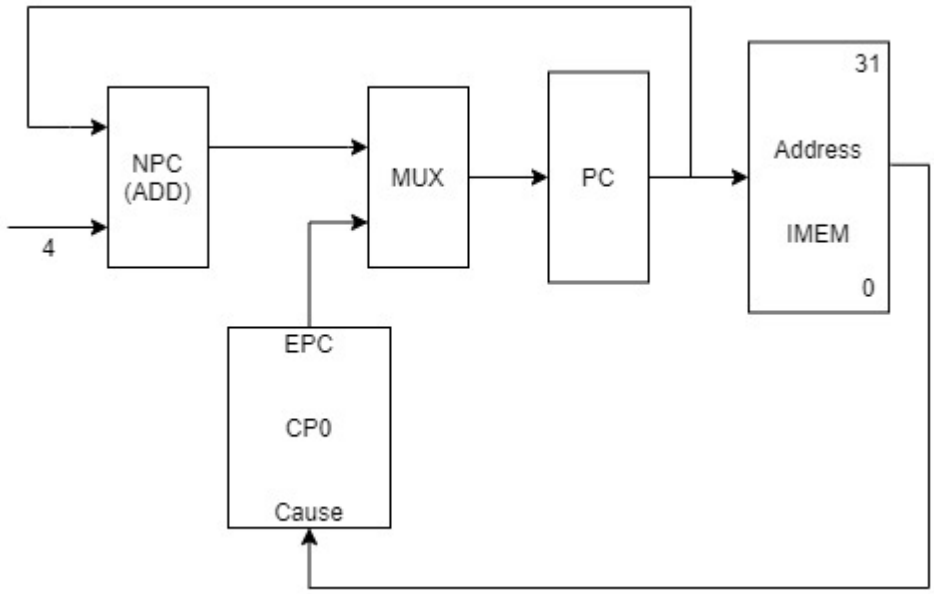
12. CLZ


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rt
Rtc
A
ALU
B
CLZ

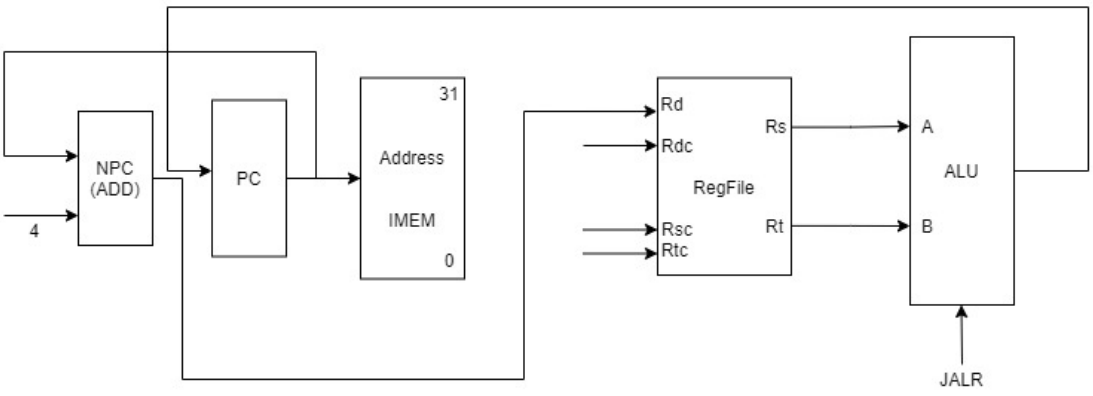
13. DIVU



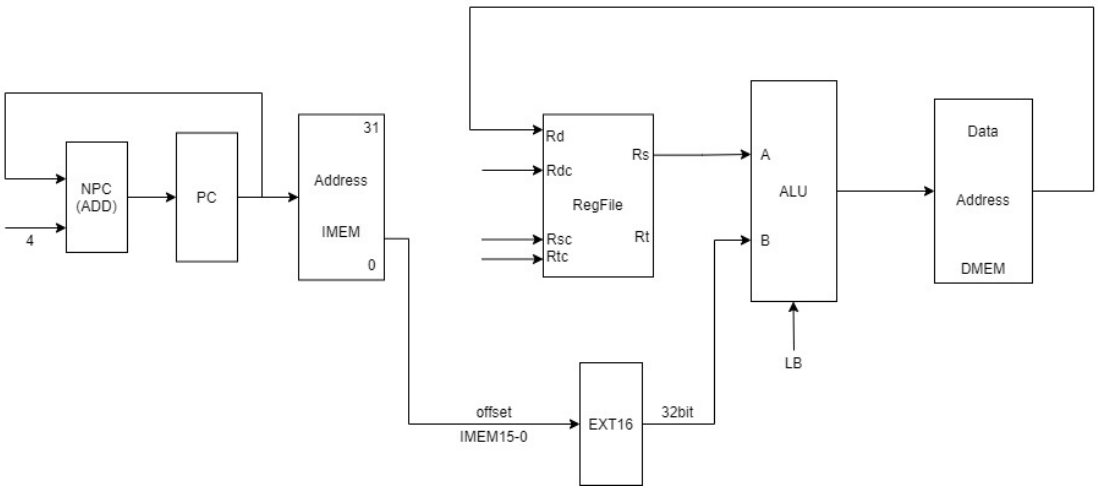
14. ERET



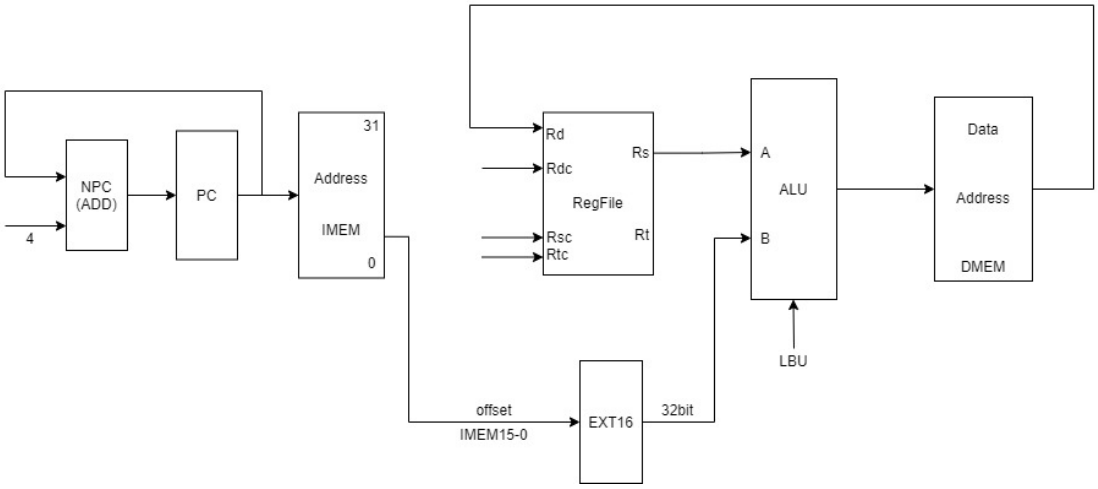
15. JALR



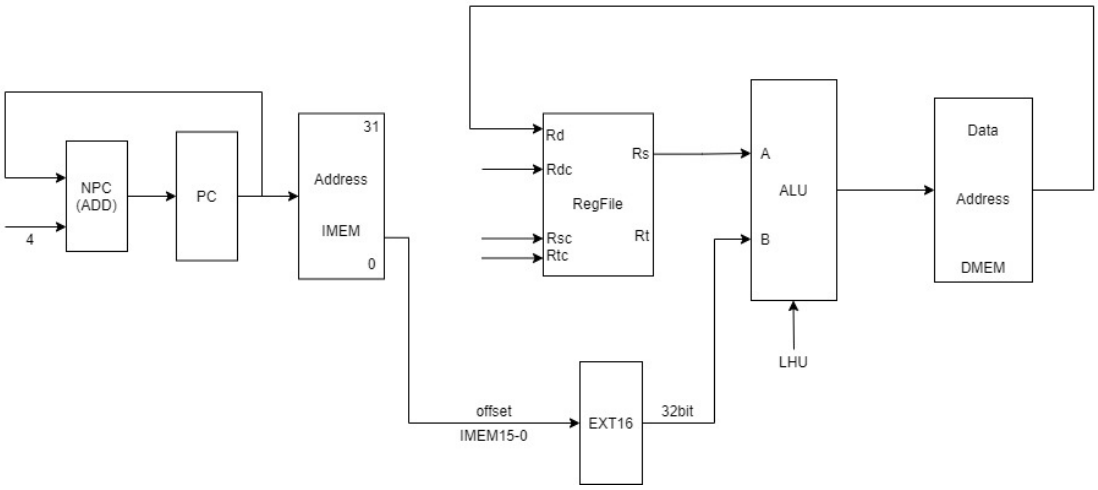
16. LB



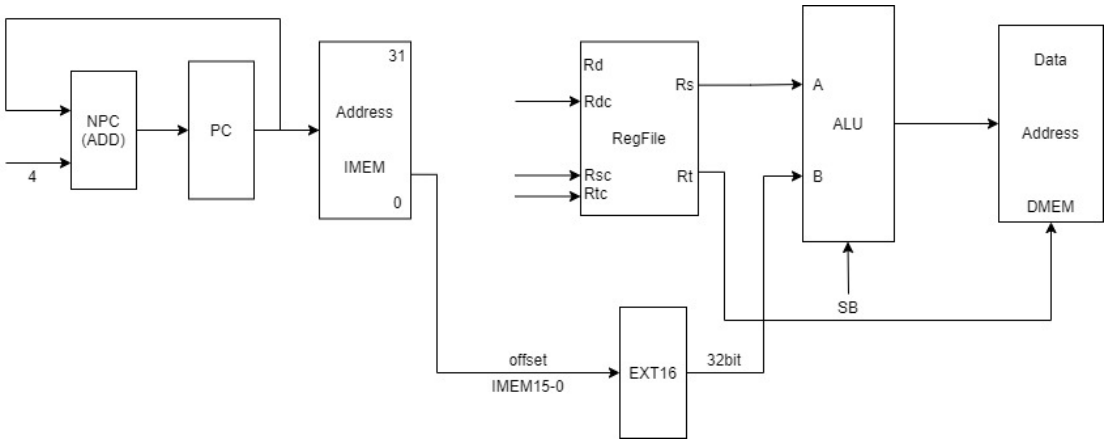
17. LBU



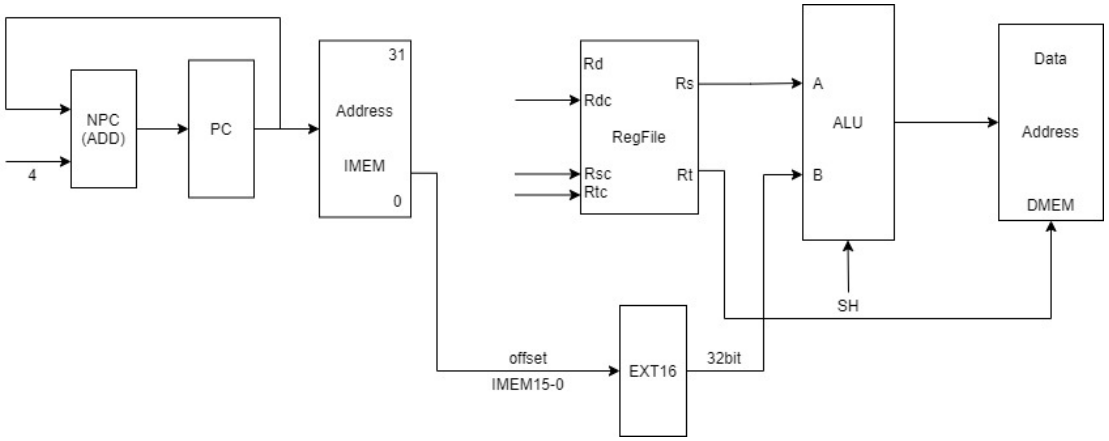
18. LHU



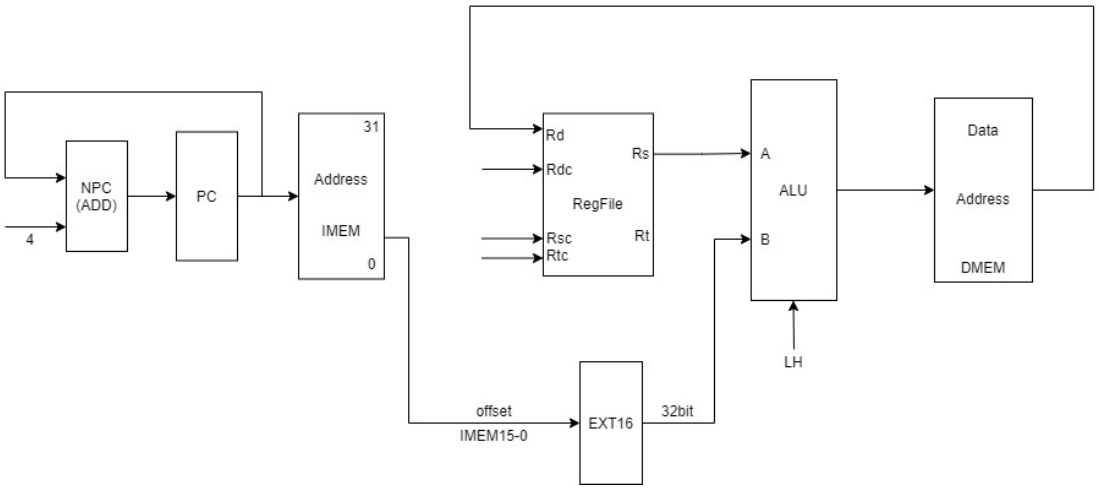
19. SB



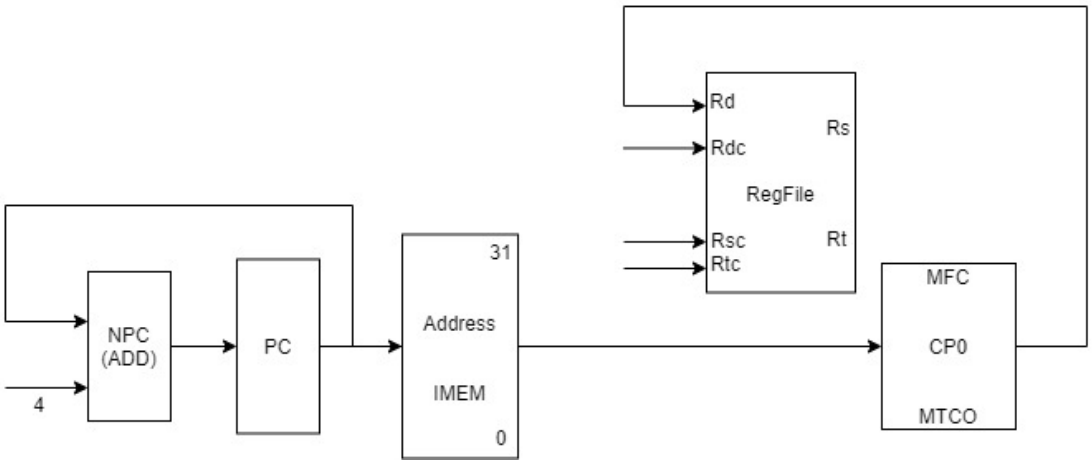
20. SH



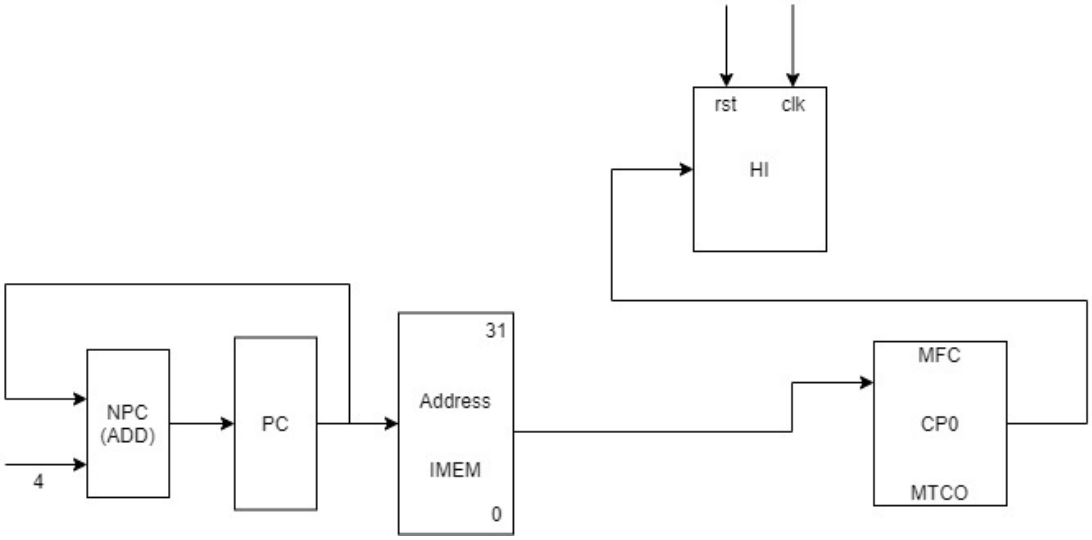
21. LH



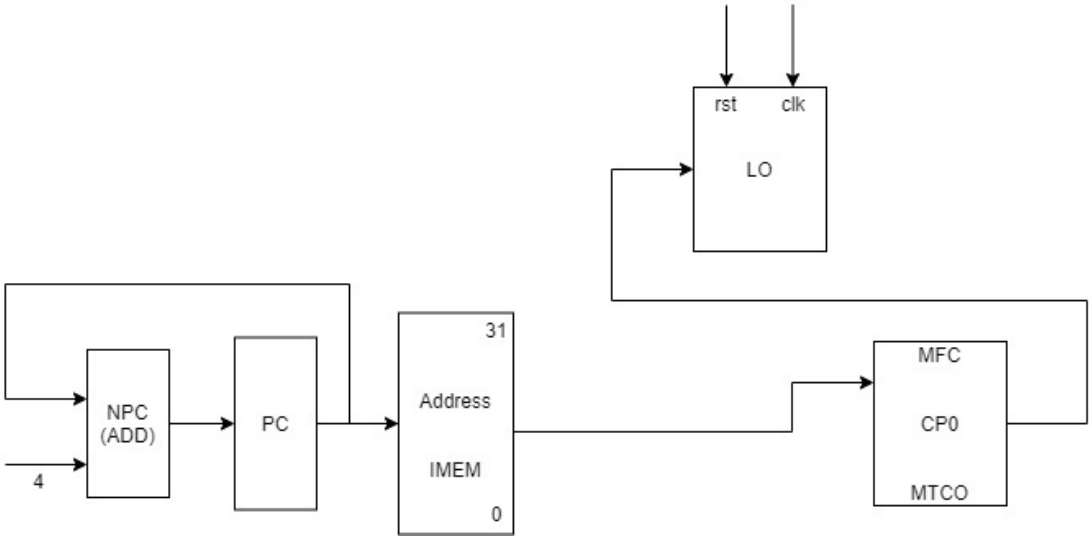
22. MFC0



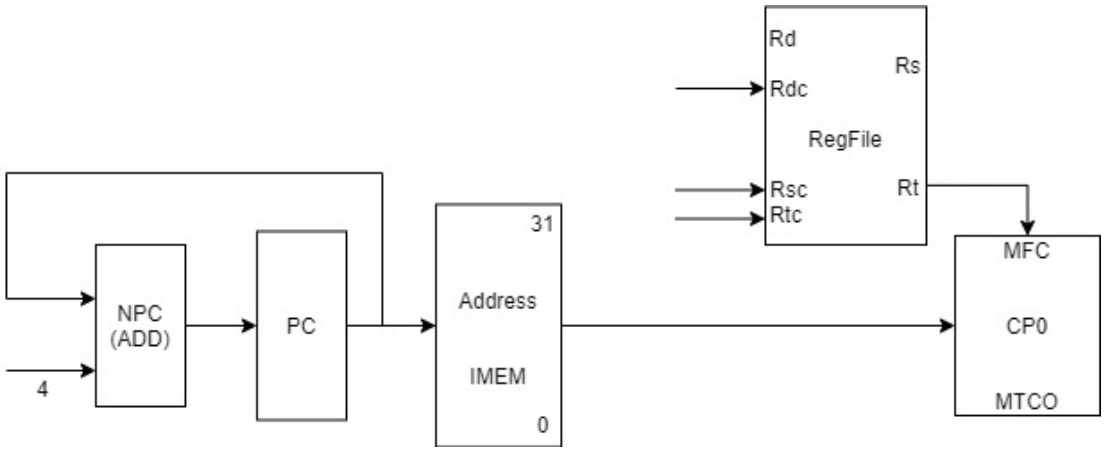
23.MFHI



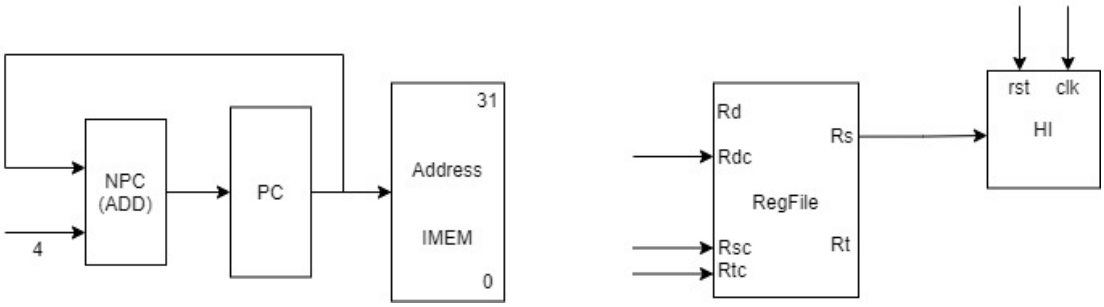
24.MFLO



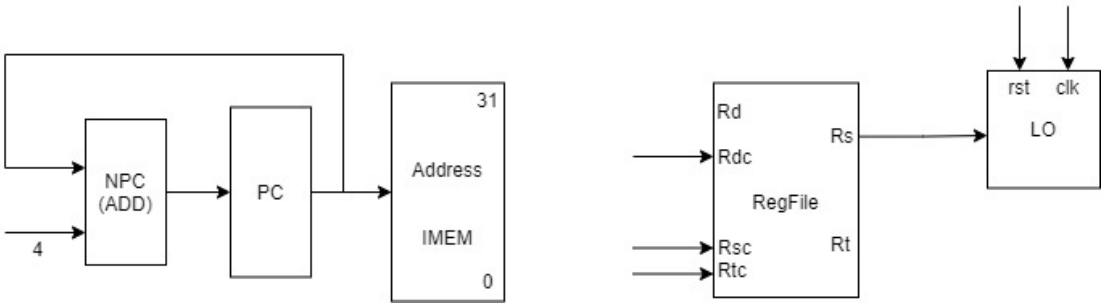
25.MTCO



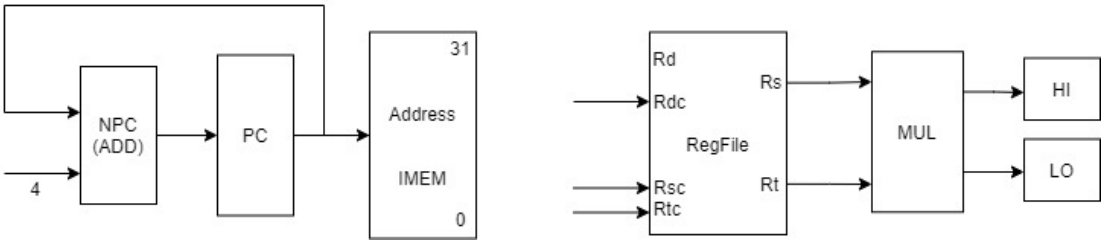
26. MTHI



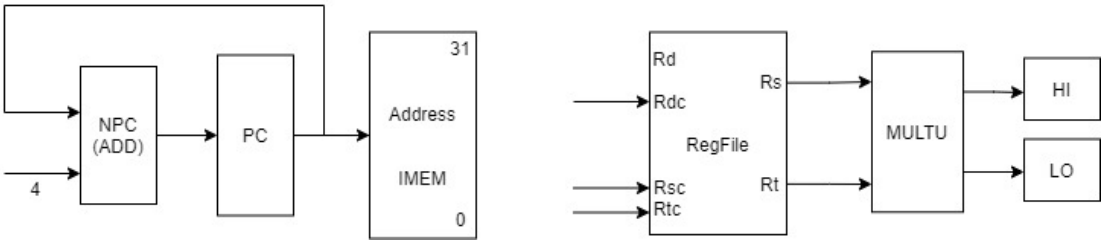
27. MTLO



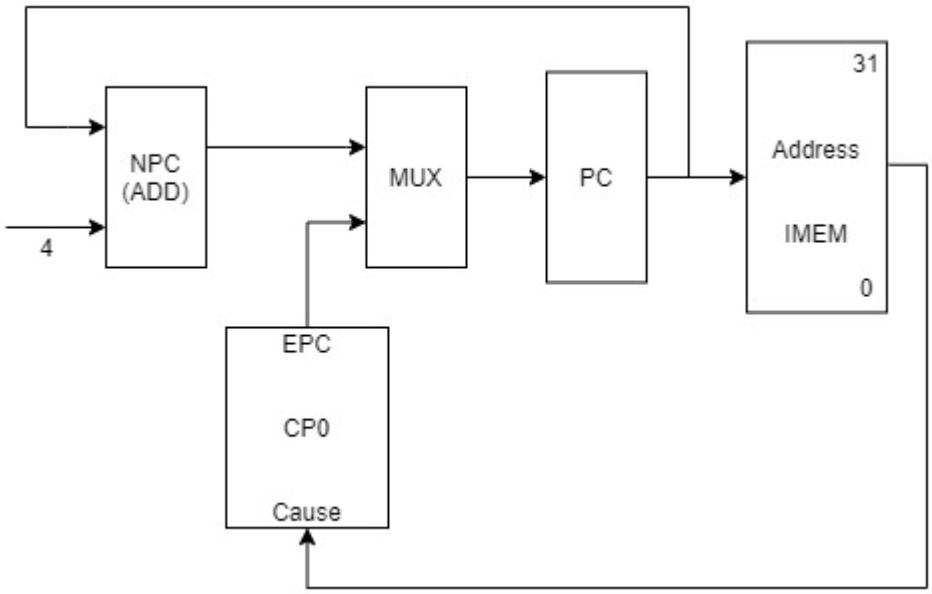
28. MUL



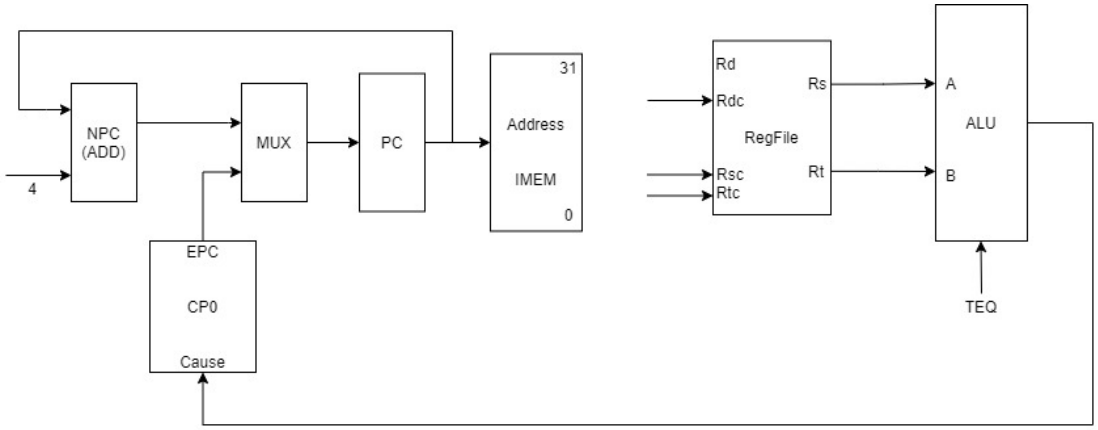
29. MULTU



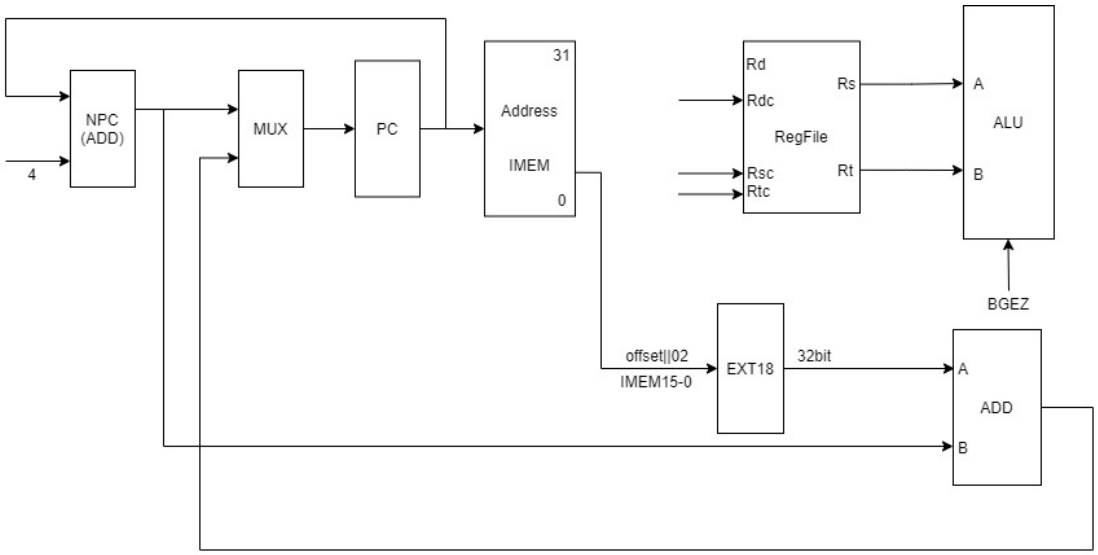
30. SYSCALL



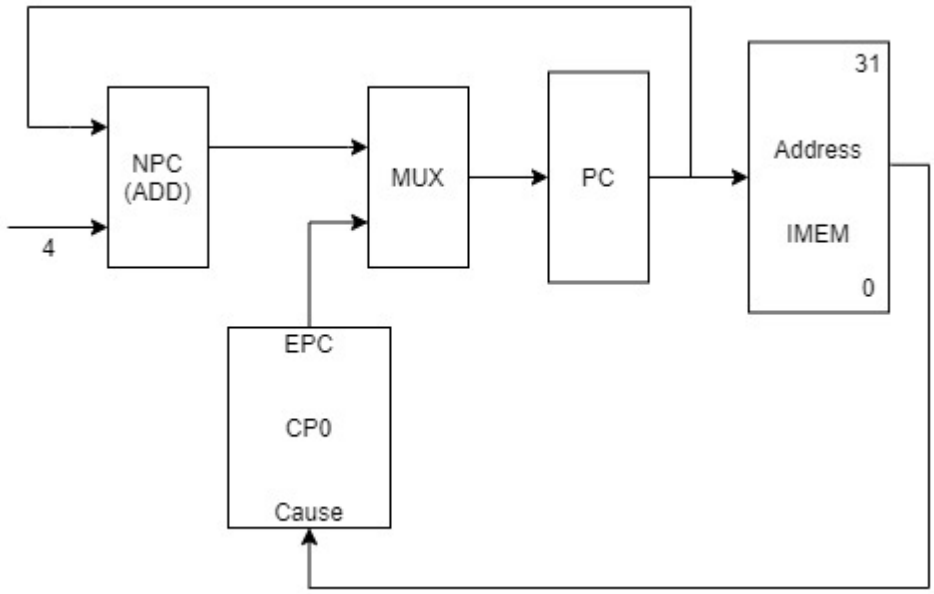
31. TEQ



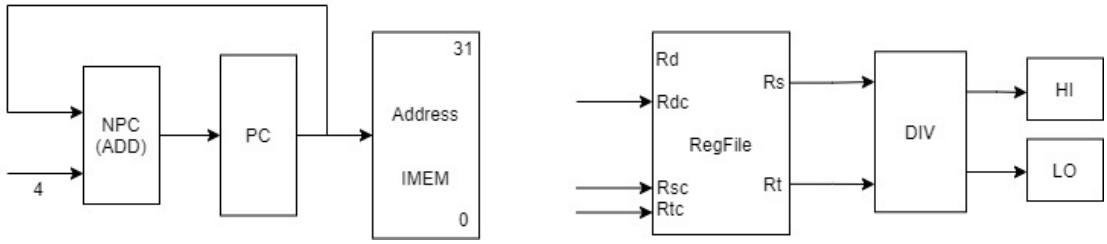
32. BGEZ



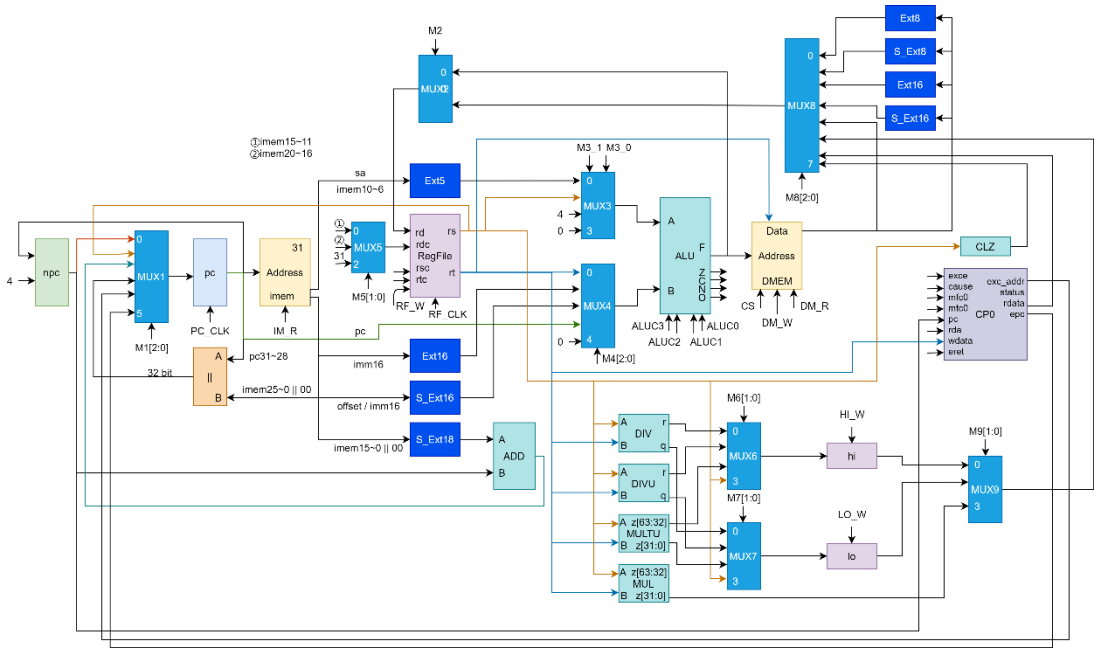
33. BREAK



34. DIV



总数据通路：



各部件输入输出关系如下图：

指令	pc	npc	imem	RegFile	ALU		Ext5	Ext16	DMEM		S_Ext16	Ext18	S_Ext18	ADD		||	
				rd	A	B			Addr	Data				A	B	A	B
ADD	npc	pc	pc	ALU	rs	rt											
ADDU	npc	pc	pc	ALU	rs	rt											
SUB	npc	pc	pc	ALU	rs	rt											
SUBU	npc	pc	pc	ALU	rs	rt											
AND	npc	pc	pc	ALU	rs	rt											
OR	npc	pc	pc	ALU	rs	rt											
XOR	npc	pc	pc	ALU	rs	rt											
NOR	npc	pc	pc	ALU	rs	rt											
SLT	npc	pc	pc	ALU	rs	rt											
SLTU	npc	pc	pc	ALU	rs	rt											
SLL	npc	pc	pc	ALU	Ext5	rt	sa										
SRL	npc	pc	pc	ALU	Ext5	rt	sa										
SRA	npc	pc	pc	ALU	Ext5	rt	sa										
SLLV	npc	pc	pc	ALU	rs	rt											
SRLV	npc	pc	pc	ALU	rs	rt											
SRAV	npc	pc	pc	ALU	rs	rt											
JR	rs	pc	pc														
ADDI	npc	pc	pc	ALU	rs	Ext16					imm16						
ADDIU	npc	pc	pc	ALU	rs	Ext16		imm16									
ANDI	npc	pc	pc	ALU	rs	Ext16		imm16									
ORI	npc	pc	pc	ALU	rs	Ext16		imm16									
XORI	npc	pc	pc	ALU	rs	Ext16		imm16									
LW	npc	pc	pc	Data	rs	S_Ext16			alu		offset						
SW	npc	pc	pc		rs	S_Ext16			alu	rt	offset						
BEQ	ADD	pc	pc		rs	rt							offset	npc	S_Ext18		
BNE	ADD	pc	pc		rs	rt							offset	npc	S_Ext18		
SLTI	npc	pc	pc	ALU	rs	S_Ext16					imm16						
SLTIU	npc	pc	pc	ALU	rs	Ext16		imm16									
LUI	npc	pc	pc	ALU	16	Ext16		imm16									
J	||	pc	pc													pc31-28	imem25-0
JAL	||	pc	pc	pc										32'h4	npc	pc31-28	imem25-0

部件功能说明：

NPC：即 PC+4，可用简单加法实现

PC：指令计数器

IMEM：指令存储器

DMEM：数据存储器

RegFile：寄存器堆

ALU：算术逻辑单元，实现算数运算与逻辑运算

Ext5：将 5 位数据无符号扩展为 32 位

Ext16：将 16 位数据无符号扩展为 32 位

S_Ext16：将 16 位数据有符号扩展为 32 位

ADD：加法器

||：数据拼接

CLZ：前置 0 个数计数器

HI：HI 寄存器

LO：LO 寄存器

MUL：有符号乘法器

MULTU：无符号乘法器

DIV：有符号除法器

DIVU：无符号除法器

3.4 CPU 控制器设计

各指令控制信号表：

指令	PC_CLK	IM_R	Rsc4~0	Rtc4~0	Rdc4~0	cp0_Rdc	exce	RF_W
ADD	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
ADDU	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SUB	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SUBU	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
AND	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
OR	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
XOR	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
NOR	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SLT	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SLTU	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SLL	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SRL	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SRA	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SLLV	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SRLV	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
SRAV	CLK	1	IM25~21	IM20~16	IM15~11	0	0	1
JR	CLK	1	IM25~21	IM20~16	0	0	0	1
ADDI	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
ADDIU	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
ANDI	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
ORI	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
XORI	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
LW	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
SW	CLK	1	IM25~21	IM20~16	0	0	0	1
BEQ	CLK	1	IM25~21	IM20~16	0	0	0	1
BNE	CLK	1	IM25~21	IM20~16	0	0	0	1
SLTI	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
SLTIU	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
LUI	CLK	1	IM25~21	IM20~16	IM20~16	0	0	1
J	CLK	1	IM25~21	IM20~16	0	0	0	1
JAL	CLK	1	IM25~21	IM20~16	31	0	0	1
CLZ	CLK	1	IM25~21	IM20~16	31	0	0	1
DIVU	CLK	1	IM25~21	IM20~16	31	0	0	1
ERET	CLK	1	IM25~21	IM20~16	31	0	1	1
JALR	CLK	1	IM25~21	IM20~16	31	0	0	1
LB	CLK	1	IM25~21	IM20~16	31	0	0	1
LBU	CLK	1	IM25~21	IM20~16	31	0	0	1
LHU	CLK	1	IM25~21	IM20~16	31	0	0	1
SB	CLK	1	IM25~21	IM20~16	31	0	0	1
SH	CLK	1	IM25~21	IM20~16	31	0	0	1
LH	CLK	1	IM25~21	IM20~16	31	0	0	1
MFC0	CLK	1	IM25~21	IM20~16	31	IM15~11	0	1
MFHI	CLK	1	IM25~21	IM20~16	31	0	0	1
MFLO	CLK	1	IM25~21	IM20~16	31	0	0	1
MTC0	CLK	1	IM25~21	IM20~16	31	IM15~11	0	1
MTHI	CLK	1	IM25~21	IM20~16	31	0	0	1
MTLO	CLK	1	IM25~21	IM20~16	31	0	0	1
MUL	CLK	1	IM25~21	IM20~16	31	0	0	1
MULTU	CLK	1	IM25~21	IM20~16	31	0	0	1
SYSCALL	CLK	1	IM25~21	IM20~16	31	0	1	1
TEQ	CLK	1	IM25~21	IM20~16	31	0	Z==0	1
BGEZ	CLK	1	IM25~21	IM20~16	31	0	0	1
BREAK	CLK	1	IM25~21	IM20~16	31	0	1	1
DIV	CLK	1	IM25~21	IM20~16	31	0	0	1

指令	RF_CLK	CS	DM_R	DM_W	HI_W	LO_W	ALU3	ALU2
ADD	CLK	0	0	0	0	0	0	0
ADDU	CLK	0	0	0	0	0	0	0
SUB	CLK	0	0	0	0	0	0	0
SUBU	CLK	0	0	0	0	0	0	0
AND	CLK	0	0	0	0	0	0	1
OR	CLK	0	0	0	0	0	0	1
XOR	CLK	0	0	0	0	0	0	1
NOR	CLK	0	0	0	0	0	0	1
SLT	CLK	0	0	0	0	0	1	0
SLTU	CLK	0	0	0	0	0	1	0
SLL	CLK	0	0	0	0	0	1	1
SRL	CLK	0	0	0	0	0	1	1
SRA	CLK	0	0	0	0	0	1	1
SLLV	CLK	0	0	0	0	0	1	1
SRLV	CLK	0	0	0	0	0	1	1
SRAV	CLK	0	0	0	0	0	1	1
JR	CLK	0	0	0	0	0	0	0
ADDI	CLK	0	0	0	0	0	0	0
ADDIU	CLK	0	0	0	0	0	0	0
ANDI	CLK	0	0	0	0	0	0	1
ORI	CLK	0	0	0	0	0	0	1
XORI	CLK	0	0	0	0	0	0	1
LW	CLK	1	1	0	0	0	0	0
SW	CLK	1	0	1	0	0	0	0
BEQ	CLK	0	0	0	0	0	0	0
BNE	CLK	0	0	0	0	0	0	0
SLTI	CLK	0	0	0	0	0	1	0
SLTIU	CLK	0	0	0	0	0	1	0
LUI	CLK	0	0	0	0	0	1	0
J	CLK	0	0	0	0	0	0	0
JAL	CLK	0	0	0	0	0	0	0
CLZ	CLK	0	0	0	0	0	0	0
DIVU	CLK	0	0	0	1	1	0	0
ERET	CLK	0	0	0	0	0	0	0
JALR	CLK	0	0	0	0	0	0	0
LB	CLK	1	1	0	0	0	0	0
LBU	CLK	1	1	0	0	0	0	0
LHU	CLK	1	1	0	0	0	0	0
SB	CLK	1	0	1	0	0	0	0
SH	CLK	1	0	1	0	0	0	0
LH	CLK	1	1	0	0	0	0	0
MFC0	CLK	0	0	0	0	0	0	0
MFHI	CLK	0	0	0	0	0	0	0
MFLO	CLK	0	0	0	0	0	0	0
MTC0	CLK	0	0	0	0	0	0	0
MTHI	CLK	0	0	0	1	0	0	0
MTLO	CLK	0	0	0	0	1	0	0
MUL	CLK	0	0	0	1	1	0	0
MULTU	CLK	0	0	0	1	1	0	0
SYSCALL	CLK	0	0	0	0	0	0	0
TEQ	CLK	0	0	0	0	0	0	0
BGEZ	CLK	0	0	0	0	0	1	0
BREAK	CLK	0	0	0	0	0	0	0
DIV	CLK	0	0	0	1	1	0	0

指令	ALU1	ALU0	M1_2	M1_1	M1_0	M2	M3_1	M3_0
ADD	1	0	0	0	0	0	0	1
ADDU	0	0	0	0	0	0	0	1
SUB	1	1	0	0	0	0	0	1
SUBU	0	1	0	0	0	0	0	1
AND	0	0	0	0	0	0	0	1
OR	0	1	0	0	0	0	0	1
XOR	1	0	0	0	0	0	0	1
NOR	1	1	0	0	0	0	0	1
SLT	1	1	0	0	0	0	0	1
SLTU	1	0	0	0	0	0	0	1
SLL	1	0	0	0	0	0	0	0
SRL	0	1	0	0	0	0	0	0
SRA	0	0	0	0	0	0	0	0
SLLV	1	0	0	0	0	0	0	1
SRLV	0	1	0	0	0	0	0	1
SRAV	0	0	0	0	0	0	0	1
JR	0	0	0	0	1	0	0	0
ADDI	1	0	0	0	0	0	0	1
ADDIU	0	0	0	0	0	0	0	1
ANDI	0	0	0	0	0	0	0	1
ORI	0	1	0	0	0	0	0	1
XORI	1	0	0	0	0	0	0	1
LW	0	0	0	0	0	1	0	1
SW	0	0	0	0	0	0	0	1
BEQ	0	1	0	Z==1	0	0	0	1
BNE	0	1	0	Z==0	0	0	0	1
SLTI	1	1	0	0	0	0	0	1
SLTIU	1	0	0	0	0	0	0	1
LUI	0	0	0	0	0	0	0	0
J	0	0	0	1	1	0	0	0
JAL	1	0	0	1	1	0	1	0
CLZ	0	0	0	0	0	1	0	1
DIVU	0	0	0	0	0	0	0	1
ERET	0	0	1	0	1	0	0	0
JALR	1	0	0	0	1	0	1	0
LB	0	0	0	0	0	1	0	1
LBU	0	0	0	0	0	1	0	1
LHU	0	0	0	0	0	1	0	1
SB	0	0	0	0	0	0	0	1
SH	0	0	0	0	0	0	0	1
LH	0	0	0	0	0	1	0	1
MFC0	0	0	0	0	0	1	0	0
MFHI	0	0	0	0	0	1	0	0
MFLO	0	0	0	0	0	1	0	0
MTC0	0	0	0	0	0	0	0	0
MTHI	0	0	0	0	0	0	0	1
MTLO	0	0	0	0	0	0	0	1
MUL	0	0	0	0	0	1	0	1
MULTU	0	0	0	0	0	0	0	1
SYSCALL	0	0	1	0	0	0	0	0
TEQ	0	1	Z==0	0	0	0	0	1
BGEZ	1	1	0	N==1	0	0	1	1
BREAK	0	0	1	0	0	0	0	0
DIV	0	0	0	0	0	0	0	1

指令	M4_2	M4_1	M4_0	M5_1	M5_0	M6_1	M6_0	M7_1
ADD	0	0	0	0	0	0	0	0
ADDU	0	0	0	0	0	0	0	0
SUB	0	0	0	0	0	0	0	0
SUBU	0	0	0	0	0	0	0	0
AND	0	0	0	0	0	0	0	0
OR	0	0	0	0	0	0	0	0
XOR	0	0	0	0	0	0	0	0
NOR	0	0	0	0	0	0	0	0
SLT	0	0	0	0	0	0	0	0
SLTU	0	0	0	0	0	0	0	0
SLL	0	0	0	0	0	0	0	0
SRL	0	0	0	0	0	0	0	0
SRA	0	0	0	0	0	0	0	0
SLLV	0	0	0	0	0	0	0	0
SRLV	0	0	0	0	0	0	0	0
SRAV	0	0	0	0	0	0	0	0
JR	0	0	0	0	0	0	0	0
ADDI	0	1	0	0	1	0	0	0
ADDIU	0	1	0	0	1	0	0	0
ANDI	0	0	1	0	1	0	0	0
ORI	0	0	1	0	1	0	0	0
XORI	0	0	1	0	1	0	0	0
LW	0	1	0	0	1	0	0	0
SW	0	1	0	0	0	0	0	0
BEQ	0	0	0	0	0	0	0	0
BNE	0	0	0	0	0	0	0	0
SLTI	0	1	0	0	1	0	0	0
SLTIU	0	0	1	0	1	0	0	0
LUI	0	0	1	0	1	0	0	0
J	0	0	0	0	0	0	0	0
JAL	0	1	1	1	0	0	0	0
CLZ	0	0	0	0	0	0	0	0
DIVU	0	0	0	0	0	0	1	0
ERET	0	0	0	0	0	0	0	0
JALR	0	1	1	0	0	0	0	0
LB	0	1	0	0	1	0	0	0
LBU	0	1	0	0	1	0	0	0
LHU	0	1	0	0	1	0	0	0
SB	0	1	0	0	0	0	0	0
SH	0	1	0	0	0	0	0	0
LH	0	1	0	0	1	0	0	0
MFC0	0	0	0	0	1	0	0	0
MFHI	0	0	0	0	0	0	0	0
MFLO	0	0	0	0	0	0	0	0
MTC0	0	0	0	0	0	0	0	0
MTHI	0	0	0	0	0	1	1	1
MTLO	0	0	0	0	0	1	1	1
MUL	0	0	0	0	0	0	0	0
MULTU	0	0	0	0	0	1	0	1
SYSCALL	0	0	0	0	0	0	0	0
TEQ	0	0	0	0	0	0	0	0
BGEZ	1	0	0	0	0	0	0	0
BREAK	0	0	0	0	0	0	0	0
DIV	0	0	0	0	0	0	0	0

指令	M7_0	M8_2	M8_1	M8_0	M9_1	M9_0
ADD	0	0	0	0	0	0
ADDU	0	0	0	0	0	0
SUB	0	0	0	0	0	0
SUBU	0	0	0	0	0	0
AND	0	0	0	0	0	0
OR	0	0	0	0	0	0
XOR	0	0	0	0	0	0
NOR	0	0	0	0	0	0
SLT	0	0	0	0	0	0
SLTU	0	0	0	0	0	0
SLL	0	0	0	0	0	0
SRL	0	0	0	0	0	0
SRA	0	0	0	0	0	0
SLLV	0	0	0	0	0	0
SRLV	0	0	0	0	0	0
SRAV	0	0	0	0	0	0
JR	0	0	0	0	0	0
ADDI	0	0	0	0	0	0
ADDIU	0	0	0	0	0	0
ANDI	0	0	0	0	0	0
ORI	0	0	0	0	0	0
XORI	0	0	0	0	0	0
LW	0	1	0	0	0	0
SW	0	0	0	0	0	0
BEQ	0	0	0	0	0	0
BNE	0	0	0	0	0	0
SLTI	0	0	0	0	0	0
SLTIU	0	0	0	0	0	0
LUI	0	0	0	0	0	0
J	0	0	0	0	0	0
JAL	0	0	0	0	0	0
CLZ	0	1	1	1	0	0
DIVU	1	0	0	0	0	0
ERET	0	0	0	0	0	0
JALR	0	0	0	0	0	0
LB	0	0	0	1	0	0
LBU	0	0	0	0	0	0
LHU	0	0	1	0	0	0
SB	0	0	0	0	0	0
SH	0	0	0	0	0	0
LH	0	0	1	1	0	0
MFC0	0	1	1	0	0	0
MFHI	0	1	0	1	0	0
MFLO	0	1	0	1	0	1
MTC0	0	0	0	0	0	0
MTHI	1	0	0	0	0	0
MTLO	1	0	0	0	0	0
MUL	0	0	0	0	1	0
MULTU	0	0	0	0	0	0
SYSCALL	0	0	0	0	0	0
TEQ	0	0	0	0	0	0
BGEZ	0	0	0	0	0	0
BREAK	0	0	0	0	0	0
DIV	0	0	0	0	0	0

控制信号说明：
PC_CLK：pc 寄存器时钟
IM_R：指令寄存器 IMEM 读有效信号
Rsc4~0：rs 寄存器选择输入控制端
Rtc4~0：rt 寄存器选择输入控制端
Rdc4~0：rd 寄存器选择输入控制端
CP0_Rdc：CP0 中 rd 寄存器选择输入控制端
EXCE：异常处理标志
RF_W：RegFile 写信号
RF_CLK：RegFile 时钟
CS：数据存储器片选信号
DM_R：数据存储器读信号
DM_W：数据存储器写信号

HI_W：HI 寄存器写信号
LO_W：LO 寄存器写信号
ALU3：ALU 控制端 3
ALU2：ALU 控制端 2
ALU1：ALU 控制端 1
ALU0：ALU 控制端 0
M1_2：MUX1 选择器控制端 2
M1_1：MUX1 选择器控制端 1
M1_0：MUX1 选择器控制端 0
M2：MUX2 选择器控制端
M3_1：MUX3 选择器控制端 1
M3_0：MUX3 选择器控制端 0
M4_2：MUX4 选择器控制端 2
M4_1：MUX4 选择器控制端 1
M4_0：MUX4 选择器控制端 0
M5_1：MUX5 选择器控制端 1
M5_0：MUX5 选择器控制端 0
M6_1：MUX6 选择器控制端 1
M6_0：MUX6 选择器控制端 0
M7_1：MUX7 选择器控制端 1
M7_0：MUX7 选择器控制端 0
M8_2：MUX8 选择器控制端 2
M8_1：MUX8 选择器控制端 1
M8_0：MUX8 选择器控制端 0
M9_1：MUX9 选择器控制端 1
M9_0：MUX9 选择器控制端 0

各微指令逻辑表达式如下图：

信号	逻辑表达式
PC_CLK	CLK
IM_R	1
Rsc4~0	IM25~21
Rtc4~0	IM20~16
Rdc4~0	IM15~11(ADD+ADDU+SUB+SUBU+AND+OR+XOR+NOR+SLT+SLTU+SLL+SRL+SRA+SLLV+SRLV+SRAV+JALR+MFHI+MFLO+CLZ)+IM20~16(ADDI+ADDIU+ANDI+ORI+XORI+LW+SLTI+SLTIU+LUI+LBU+LHU+LB+LH+MFC0)+31(JAL)
Cp0Rdc	IM15~11(MFC0+MTC0)
EXCE	BREAK+SYSCALL+ERET+TEQ(Z==0)
RF_W	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+LBU+LHU+LB+LH+MFHI+MFLO+MFC0+CLZ
RF_CLK	CLK
CS	LW+SW+LBU+LHU+LB+LH+SB+SH
DM_R	LW+LBU+LHU+LB+LH
DM_W	SW+SB+SH
HI_W	DIV+DIVU+MUL+MULTU+MTHI
LO_W	DIV+DIVU+MUL+MULTU+MTLO
ALUC3	SLT+SLTU+SLL+SRL+SRA+SLLV+SRLV+SRAV+SLTI+SLTIU+LUI+BGEZ
ALUC2	AND+OR+XOR+NOR+SLL+SRL+SRA+SLLV+SRLV+SRAV+ANDI+ORI+XORI
ALUC1	ADD+SUB+XOR+NOR+SLT+SLTU+SLL+SLLV+ADDI+XORI+SLTI+SLTIU+JAL+BGEZ+JALR
ALUC0	SUB+SUBU+OR+NOR+SLT+SRL+SRLV+ORI+BEQ+BNE+SLTI+BGEZ+TEQ

M1_2	BREAK+SYSCALL+ERET+TEQ(Z==0)
M1_1	BEQ(Z==1)+BNE(Z==0)+J+JAL+BGEZ(N==1)
M1_0	JR+J+JAL+JALR+ERET
M2	LW+LBU+LHU+LB+LH+MFHI+MFLO+MFC0+MUL+CLZ
M3_1	JAL+BGEZ+JALR
M3_0	ADD+ADDU+SUB+SUBU+AND+OR+XOR+NOR+SLT+SLTU+SLLV+SRLV+SRAV+ADDI+ ADDIU+ANDI+ORI+XORI+LW+SW+BEQ+BNE+SLTI+SLTIU+DIV+DIVU+MULT+MULTU+BGEZ+LBU+LHU+LB+LH+SB+SH+MTHI+MTLO+CLZ+TEQ
M4_2	BGEZ
M4_1	ADDI+ADDIU+LW+SW+SLTI+JAL+JALR+LBU+LHU+LB+LH+SB+SH
M4_0	ANDI+ORI+XORI+SLTIU+LUI+JAL+JALR
M5_1	JAL
M5_0	ADDI+ADDIU+ANDI+ORI+XORI+LW+SLTI+SLTIU+LUI+LBU+LHU+LB+LH+MFC0
M6_1	MULTU+MTHI+MTLO
M6_0	DIVU+MTHI+MTLO

M7_1	MULTU+MTHI+MTLO
M7_0	DIVU+MTHI+MTLO
M8_2	LW+MFHI+MFLO+MFC0+MUL+CLZ
M8_1	LHU+LH+MFC0+MUL+CLZ
M8_0	LB+LH+MFHI+MFLO+CLZ
M9_1	MUL
M9_0	MFLO

四、后仿真与下板

4.1 后仿真功能说明

run behavioral simulation-----行为级仿真，行为级别的仿真通常也说功能仿真。
post-synthesis function simulation-----综合后的功能仿真。
post-synthesis timing simulation-----综合后带时序信息的仿真，综合后带时序信息的仿真比较接近于真实的时序。
post-implementation function simulation-----布线后的功能仿真。
post-implementation timing simulation-----（布局布线后的仿真） 执行后的时序仿真，该仿真时最接近真实的时序波形

前仿真即为行为级仿真，不涉及时序信息。后仿真即为综合后、布线后的带有时序信息的仿真。

4.2 后仿真代码修改

4.2.1 顶层模块接口更改

```
module sccomp_dataflow(
    input clk_in,
    input reset,
    output [7:0] o_seg,
    output [7:0] o_sel
    //output [31:0] inst,
    //output [31:0] pc
    );
```

由于后仿真需要进行管脚设置，需输出的值为七段数码管相关变量，故添加七段数码管段选与位选信号。

4.2.2 对系统时钟进行分频操作

```
reg [21:0] cnt;
    always @ (posedge clk_in, posedge reset)
      if (reset)
        cnt <= 0;
      else
```

```
        cnt <= cnt + 1'b1;

wire new_clk=cnt[21];
```

降低 CPU 主频控制 pc 值在七段数码管上的显示速度。

4.2.3 七段数码管模块

```
module seg7x16(
    input clk,
     input reset,
     input cs,
     input [31:0] i_data,
     output [7:0] o_seg,
     output [7:0] o_sel
    );

    reg [14:0] cnt;
     always @ (posedge clk, posedge reset)
      if (reset)
        cnt <= 0;
      else
        cnt <= cnt + 1'b1;

    wire seg7_clk = cnt[14];

     reg [2:0] seg7_addr;

     always @ (posedge seg7_clk, posedge reset)
       if(reset)
          seg7_addr <= 0;
       else
          seg7_addr <= seg7_addr + 1'b1;

     reg [7:0] o_sel_r;

     always @ (*)
       case(seg7_addr)
          7 : o_sel_r = 8'b01111111;
          6 : o_sel_r = 8'b10111111;
          5 : o_sel_r = 8'b11011111;
          4 : o_sel_r = 8'b11101111;
          3 : o_sel_r = 8'b11110111;
          2 : o_sel_r = 8'b11111011;
          1 : o_sel_r = 8'b11111101;
          0 : o_sel_r = 8'b11111110;
```

```
   endcase

reg [31:0] i_data_store;
always @ (posedge clk, posedge reset)
  if(reset)
     i_data_store <= 0;
   else if(cs)
     i_data_store <= i_data;

reg [7:0] seg_data_r;
always @ (*)
  case(seg7_addr)
     0 : seg_data_r = i_data_store[3:0];
     1 : seg_data_r = i_data_store[7:4];
     2 : seg_data_r = i_data_store[11:8];
     3 : seg_data_r = i_data_store[15:12];
     4 : seg_data_r = i_data_store[19:16];
     5 : seg_data_r = i_data_store[23:20];
     6 : seg_data_r = i_data_store[27:24];
     7 : seg_data_r = i_data_store[31:28];
   endcase

reg [7:0] o_seg_r;
always @ (posedge clk, posedge reset)
  if(reset)
     o_seg_r <= 8'hff;
   else
     case(seg_data_r)
       4'h0 : o_seg_r <= 8'hC0;
     4'h1 : o_seg_r <= 8'hF9;
     4'h2 : o_seg_r <= 8'hA4;
     4'h3 : o_seg_r <= 8'hB0;
     4'h4 : o_seg_r <= 8'h99;
     4'h5 : o_seg_r <= 8'h92;
     4'h6 : o_seg_r <= 8'h82;
     4'h7 : o_seg_r <= 8'hF8;
     4'h8 : o_seg_r <= 8'h80;
     4'h9 : o_seg_r <= 8'h90;
     4'hA : o_seg_r <= 8'h88;
     4'hB : o_seg_r <= 8'h83;
     4'hC : o_seg_r <= 8'hC6;
     4'hD : o_seg_r <= 8'hA1;
     4'hE : o_seg_r <= 8'h86;
     4'hF : o_seg_r <= 8'h8E;
```

```
            endcase

        assign o_sel = o_sel_r;
        assign o_seg = o_seg_r;

endmodule
```

4.3 下板操作

完成后仿真后便可进行下板验证，由于七段数码管选择输出程序计数器 pc 的值，故开发板七段数码管处可稳定显示计数器 pc 的值。可以通过对时钟分频控制 pc 在数码管中的显示速度。

五、模块建模

5.1 sccomp_dataflow //顶层模块，调用 cpu、imem、dram 三个模块

```
`timescale 1ns / 1ps
module sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0] inst,
    output [31:0] pc
    );

    wire CS,DM_W,DM_R;
    wire [31:0] DM_addr;
    wire [31:0] DM_WData,DM_RData;

    wire [31:0] instr_addr;
    wire [31:0] dm_addr;
    assign instr_addr = pc - 32'h0040_0000;
    //assign instr_addr = pc;
    assign dm_addr = (DM_addr - 32'h1001_0000) / 4;


    imen IM(instr_addr[12:2],inst);

    Dram Dram(clk_in,CS,DM_W,DM_R,dm_addr,DM_WData,DM_RData);

    cpu sccpu(clk_in,reset,inst,DM_addr,DM_RData,DM_WData,CS,DM_W,DM_R,
pc);

endmodule
```

5.2 cpu //CPU 模块，实现指令译码、数据通路设计、控制器设计等核心功能

```
module cpu(
    input clk,
    input rst,
    //data(instruction) from iram
    input [31:0] IM,
    //data exchanged with dram
    output [31:0] DM_addr,
    input [31:0] DM_RData,
    output [31:0] DM_WData,
    //control of dram
    output CS,
    output DM_W,
    output DM_R,
    //output of pcre
    output [31:0] PC_out
    );

    //pcreg
    wire PC_CLK;
    wire ena;
    wire [31:0] PC_in;
    //wire [31:0] PC_out;

    //NPC
    wire [31:0] NPC;

    //regfile
    wire [31:0] rs;
    wire [31:0] rt;
    wire [31:0] rd;
    wire [4:0] rdc;
    wire [4:0] rsc;
    wire [4:0] rtc;
    wire RF_CLK;
    wire RF_W;


    reg [31:0] temp;

    always @(posedge clk)
    begin
        temp <= rs;
    end
```

```
//alu
wire [3:0] aluc;
wire [31:0] alu_a;
wire [31:0] alu_b;
wire [31:0] alu_c;
wire zero;
wire carry;
wire negative;
wire overflow;

//DIV
wire DIV_start;
wire [31:0] DIV_dividend;
wire [31:0] DIV_divisor;
wire [31:0] DIV_q;
wire [31:0] DIV_r;
wire DIV_busy;
//DIVU
wire DIVU_start;
wire [31:0] DIVU_dividend;
wire [31:0] DIVU_divisor;
wire [31:0] DIVU_q;
wire [31:0] DIVU_r;
wire DIVU_busy;
//MULT
wire [31:0] MULT_a;
wire [31:0] MULT_b;
wire [63:0] MULT_z;

//multu
wire [31:0] multu_a;
wire [31:0] multu_b;
wire [63:0] multu_z;

//HI
wire [31:0] HI_data_in;
wire [31:0] HI_data_out;
wire HI_W;

//LO
wire [31:0] LO_data_in;
wire [31:0] LO_data_out;
wire LO_W;
```

```
//CP0
wire [31:0] CP0_pc;
wire [4:0] CP0_Rd;
wire [31:0] CP0_wdata;
wire CP0_exception;
wire [4:0] CP0_cause;
wire [31:0] CP0_rdata;
wire [31:0] CP0_status;
wire [31:0] CP0_epc;
wire [31:0] CP0_exc_addr;

//Iram
//wire [31:0] IM;

//EXT
wire [31:0] Ext5;
wire [31:0] Ext16;
wire [31:0] S_Ext16;
wire [31:0] S_Ext18;

wire [7:0] DExt8in;
wire [7:0] DS_Ext8in;
wire [15:0] DExt16in;
wire [15:0] DS_Ext16in;

wire [31:0] DExt8;
wire [31:0] DS_Ext8;
wire [31:0] DExt16;
wire [31:0] DS_Ext16;

//MUX
wire M1_2;
wire M1_1;
wire M1_0;
wire [31:0] M1_out;
wire M2;
wire [31:0] M2_out;
wire M3_1;
wire M3_0;
wire [31:0] M3_out;
wire M4_2;
wire M4_1;
wire M4_0;
```

```
wire [31:0] M4_out;
wire M5_1;
wire M5_0;
wire [4:0] M5_out;
wire M6_1;
wire M6_0;
wire [31:0] M6_out;
wire M7_1;
wire M7_0;
wire [31:0] M7_out;
wire M8_2;
wire M8_1;
wire M8_0;
wire [31:0] M8_out;
wire M9_1;
wire M9_0;
wire [31:0] M9_out;


//ADD
wire [31:0] ADD_A;
wire [31:0] ADD_B;
wire [31:0] ADD_C;

//Connect||
wire [3:0] Connect_A;
wire [27:0] Connect_B;
wire [31:0] Connect_C;

//CLZ
wire [31:0] CLZ_in;
wire [31:0] CLZ_out;

//31 条指令
wire ADD = (IM[31:26]==6'b0) && (IM[5:0]==6'b100000);
wire ADDU = (IM[31:26]==6'b0) && (IM[5:0]==6'b100001);
wire SUB = (IM[31:26]==6'b0) && (IM[5:0]==6'b100010);
wire SUBU = (IM[31:26]==6'b0) && (IM[5:0]==6'b100011);
wire AND = (IM[31:26]==6'b0) && (IM[5:0]==6'b100100);
wire OR = (IM[31:26]==6'b0) && (IM[5:0]==6'b100101);
wire XOR = (IM[31:26]==6'b0) && (IM[5:0]==6'b100110);
wire NOR = (IM[31:26]==6'b0) && (IM[5:0]==6'b100111);
wire SLT = (IM[31:26]==6'b0) && (IM[5:0]==6'b101010);
wire SLTU = (IM[31:26]==6'b0) && (IM[5:0]==6'b101011);
```

```
    wire SLL = (IM[31:26]==6'b0) && (IM[5:0]==6'b000000);
    wire SRL = (IM[31:26]==6'b0) && (IM[5:0]==6'b000010);
    wire SRA = (IM[31:26]==6'b0) && (IM[5:0]==6'b000011);
    wire SLLV = (IM[31:26]==6'b0) && (IM[5:0]==6'b000100);
    wire SRLV = (IM[31:26]==6'b0) && (IM[5:0]==6'b000110);
    wire SRAV = (IM[31:26]==6'b0) && (IM[5:0]==6'b000111);
    wire JR = (IM[31:26]==6'b0) && (IM[5:0]==6'b001000);

    wire ADDI = (IM[31:26]==6'b001000);
    wire ADDIU = (IM[31:26]==6'b001001);
    wire ANDI = (IM[31:26]==6'b001100);
    wire ORI = (IM[31:26]==6'b001101);
    wire XORI = (IM[31:26]==6'b001110);
    wire LW = (IM[31:26]==6'b100011);
    wire SW = (IM[31:26]==6'b101011);
    wire BEQ = (IM[31:26]==6'b000100);
    wire BNE = (IM[31:26]==6'b000101);
    wire SLTI = (IM[31:26]==6'b001010);
    wire SLTIU = (IM[31:26]==6'b001011);
    wire LUI = (IM[31:26]==6'b001111);

    wire J = (IM[31:26]==6'b000010);
    wire JAL = (IM[31:26]==6'b000011);

    //cpu54 剩余指令
    wire CLZ = (IM[31:26]==6'b011100) && (IM[5:0]==6'b100000);
    wire DIVU = (IM[31:26]==6'b0) && (IM[5:0]==6'b011011);
    wire ERET = (IM[31:26]==6'b010000) && (IM[25:21]==6'b10000) && (IM[
5:0]==6'b011000);
    wire JALR = (IM[31:26]==6'b0) && (IM[20:16]==5'b0) && (IM[5:0]==6'b
001001);
    wire LB = (IM[31:26]==6'b100000);
    wire LBU = (IM[31:26]==6'b100100);
    wire LHU = (IM[31:26]==6'b100101);
    wire SB = (IM[31:26]==6'b101000);
    wire SH = (IM[31:26]==6'b101001);
    wire LH = (IM[31:26]==6'b100001);
    wire MFC0 = (IM[31:26]==6'b010000) && (IM[25:21]==5'b0) && (IM[5:0]
==6'b0);
    wire MFHI = (IM[31:26]==6'b0) && (IM[5:0]==6'b010000);
    wire MFLO = (IM[31:26]==6'b0) && (IM[5:0]==6'b010010);
    wire MTC0 = (IM[31:26]==6'b010000) && (IM[25:21]==5'b00100) && (IM[
5:0]==6'b0);
    wire MTHI = (IM[31:26]==6'b0) && (IM[5:0]==6'b010001);
```

```
    wire MTLO = (IM[31:26]==6'b0) && (IM[5:0]==6'b010011);
    wire MUL = (IM[31:26]==6'b011100) && (IM[5:0]==6'b000010);
    wire MULTU = (IM[31:26]==6'b0) && (IM[5:0]==6'b011001);
    wire SYSCALL = (IM[31:26]==6'b0) && (IM[5:0]==6'b001100);
    wire TEQ = (IM[31:26]==6'b0) && (IM[5:0]==6'b110100);
    wire BGEZ = (IM[31:26]==6'b000001) && (IM[20:16]==5'b00001);
    wire BREAK = (IM[31:26]==6'b0) && (IM[5:0]==6'b001101);
    wire DIV = (IM[31:26]==6'b0) && (IM[5:0]==6'b011010);


    //pcreg
    assign PC_CLK = clk;
    assign ena = 1;
    assign PC_in = M1_out;

    //Regfiles
    assign rd = M2_out;
    assign rdc = M5_out;
    assign rsc = IM[25:21];
    assign rtc = IM[20:16];
    assign RF_W = ADD||ADDU||SUB||SUBU||AND||OR||XOR||NOR||SLT||SLTU||S
LL||SRL||SRA||SLLV||SRLV||
                  SRAV||ADDI||ADDIU||ANDI||ORI||XORI||LW||SLTI||SLTIU||
LUI||JAL||JALR||LBU||LHU||
                  LB||LH||MFHI||MFLO||MFC0||CLZ||MUL;
    assign RF_CLK = clk;

    //alu
    assign aluc[3] = SLT||SLTU||SLL||SRL||SRA||SLLV||SRLV||SRAV||SLTI||
SLTIU||LUI||BGEZ;
    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||XO
RI||SLTI||SLTIU||JAL||BGEZ||JALR;
    assign aluc[0] = SUB||SUBU||OR||NOR||SLT||SRL||SRLV||ORI||BEQ||BNE|
|SLTI||BGEZ||TEQ;
    assign alu_a = M3_out;
    assign alu_b = M4_out;

    //MUX
    assign M1_2 = BREAK||SYSCALL||ERET||(TEQ && (zero == 0));
    assign M1_1 = (BEQ && (zero == 1))||(BNE && (zero == 0))||J||JAL||(
BGEZ && (negative == 1'b0));
    assign M1_0 = JR||J||JAL||JALR||ERET;
```

```
    assign M2 = LW||LBU||LHU||LB||LH||MFHI||MFLO||MFC0||CLZ||MUL;
    assign M3_1 = JAL||JALR;
    assign M3_0 = ADD||ADDU||SUB||SUBU||AND||OR||XOR||NOR||SLT||SLTU||S
LLV||SRLV||SRAV||ADDI||
                ADDIU||ANDI||ORI||XORI||LW||SW||BEQ||BNE||SLTI||SLTIU||
DIV||DIVU||MUL||MULTU||
                BGEZ||LBU||LHU||LB||LH||SB||SH||MTHI||MTLO||CLZ||TEQ;
    assign M4_2 = BGEZ;
    assign M4_1 = ADDI||ADDIU||LW||SW||SLTI||JAL||JALR||LBU||LHU||LB||L
H||SB||SH;
    assign M4_0 = ANDI||ORI||XORI||SLTIU||LUI||JAL||JALR;
    assign M5_1 = JAL;
    assign M5_0 = ADDI||ADDIU||ANDI||ORI||XORI||LW||SLTI||SLTIU||LUI||L
BU||LHU||LB||LH||MFC0;
    assign M6_1 = MTHI||MTLO||MULTU;
    assign M6_0 = DIVU||MTHI||MTLO;
    assign M7_1 = MTHI||MTLO||MULTU;
    assign M7_0 = DIVU||MTHI||MTLO;
    assign M8_2 = LW||MFHI||MFLO||MFC0||CLZ||MUL;
    assign M8_1 = LHU||LH||MFC0||CLZ;
    assign M8_0 = LB||LH||MFHI||MFLO||CLZ||MUL;
    assign M9_1 = MUL;
    assign M9_0 = MFLO;

    //Dram
    assign CS = LW||SW||LBU||LHU||LB||LH||SB||SH;
    assign DM_R = LW||LBU||LHU||LB||LH;
    assign DM_W = SW||SB||SH;
    assign DM_addr = alu_c;
    assign DM_WData = (SH && DM_addr[1]==1'b0) ? {DM_RData[31:16], rt[1
5:0]} : (SH && DM_addr[1]==1'b1) ? {rt[15:0], DM_RData[15:0]}:(SB && DM
_addr[1:0]==2'b00) ? {DM_RData[31:8], rt[7:0]} : (SB && DM_addr[1:0]==2
'b01) ? {DM_RData[31:15], rt[7:0], DM_RData[7:0]}:(SB && DM_addr[1:0]==
2'b10) ?{DM_RData[31:24], rt[7:0], DM_RData[15:0]}:(SB && DM_addr[1:0]=
=2'b11) ?{rt[7:0], DM_RData[23:0]}:rt;

    //Connect||
    assign Connect_A = PC_out[31:28];
    assign Connect_B = {IM[25:0],2'b00};
    assign Connect_C = {Connect_A,Connect_B};

    //ADD
    assign ADD_A = S_Ext18;
    assign ADD_B = NPC;
```

```
    assign ADD_C = ADD_A + ADD_B;

    //CLZ
    assign CLZ_in = rs;

    //DIV
    assign DIV_start = DIV&&~DIV_busy;
    assign DIV_dividend = rs;
    assign DIV_divisor = rt;
    //DIVU
    assign DIVU_start = DIVU&&~DIVU_busy;
    assign DIVU_dividend = rs;
    assign DIVU_divisor = rt;
    //MULT
    assign MULT_a = rs;
    assign MULT_b = rt;
    //multu
    assign multu_a = rs;
    assign multu_b = rt;

    //HI
    assign HI_data_in = M6_out;
    assign HI_W = DIV||DIVU||MULTU||MTHI;
    //LO
    assign LO_data_in = M7_out;
    assign LO_W = DIV||DIVU||MULTU||MTLO;

    //CP0
    assign CP0_pc = NPC;
    assign CP0_Rd = IM[15:11];
    assign CP0_wdata = rt;
    assign CP0_exception = BREAK||SYSCALL||(TEQ && (zero == 0));
    assign CP0_cause = BREAK ? 5'b01001:SYSCALL ? 5'b01000:TEQ ? 5'b011
01:0;


    //assign NPC = PC_out + 4;
    assign NPC = (DIV_busy||DIVU_busy) ? PC_out : (PC_out + 4);


    //EXT
    assign DS_Ext8in = (LB && DM_addr[1:0]==2'b00) ? DM_RData[7:0]:(LB
&& DM_addr[1:0]==2'b01) ? DM_RData[15:8]:(LB && DM_addr[1:0]==2'b10) ?
DM_RData[23:16]:(LB && DM_addr[1:0]==2'b11) ? DM_RData[31:24]:8'b0;
```

```
    assign DExt8in = (LBU && DM_addr[1:0]==2'b00) ? DM_RData[7:0]:(LBU
&& DM_addr[1:0]==2'b01) ? DM_RData[15:8]:(LBU && DM_addr[1:0]==2'b10) ?
 DM_RData[23:16]:(LBU && DM_addr[1:0]==2'b11) ? DM_RData[31:24]:8'b0;
    assign DS_Ext16in = (LH && DM_addr[1]==1'b0) ? DM_RData[15:0]:(LH &
& DM_addr[1]==1'b1) ? DM_RData[31:16]:16'b0;
    assign DExt16in = (LHU && DM_addr[1]==1'b0) ? DM_RData[15:0]:(LHU &
& DM_addr[1]==1'b1) ? DM_RData[31:16]:16'b0;

    Ext5 E5(IM[10:6],Ext5);
    Ext16 E16(IM[15:0],Ext16);
    S_Ext16 S_E16(IM[15:0],S_Ext16);
    S_Ext18 S_E18({IM[15:0],2'b00},S_Ext18);

    Ext8 DE8(DExt8in,DExt8);
    S_Ext8 S_DE8(DS_Ext8in,DS_Ext8);
    Ext16 DE16(DExt16in,DExt16);
    S_Ext16 S_DE16(DS_Ext16in,DS_Ext16);



    CLZ clz(CLZ_in,CLZ_out);


    MUX8 MUX1(NPC,temp,ADD_C,Connect_C,CP0_exc_addr,CP0_epc,CP0_rdata,0
,M1_2,M1_1,M1_0,M1_out);
    assign M2_out = M2==0 ? alu_c:M8_out;
    MUX MUX3(Ext5,rs,4,0,M3_1,M3_0,M3_out);
    MUX8 MUX4(rt,Ext16,S_Ext16,PC_out,0,0,0,0,M4_2,M4_1,M4_0,M4_out);
    MUX_5bit MUX5(IM[15:11],IM[20:16],5'b11111,0,M5_1,M5_0,M5_out);
    MUX MUX6(DIV_r,DIVU_r,multu_z[63:32],rs,M6_1,M6_0,M6_out);
    MUX MUX7(DIV_q,DIVU_q,multu_z[31:0],rs,M7_1,M7_0,M7_out);
    MUX MUX9(HI_data_out,LO_data_out,MULT_z[31:0],0,M9_1,M9_0,M9_out);
    MUX8 MUX8(DExt8,DS_Ext8,DExt16,DS_Ext16,DM_RData,M9_out,CP0_rdata,C
LZ_out,M8_2,M8_1,M8_0,M8_out);


    MULT MULT(clk,rst,MULT_a,MULT_b,MULT_z);
    multu multu(clk,rst,multu_a,multu_b,multu_z);
    DIV div(DIV_dividend,DIV_divisor,DIV_start,clk,rst,DIV_q,DIV_r,DIV_
busy);
    DIVU divu(DIVU_dividend,DIVU_divisor,DIVU_start,clk,rst,DIVU_q,DIVU
_r,DIVU_busy);


    pcreg pcreg_HI(clk,rst,HI_W,HI_data_in,HI_data_out);
```

```
    pcreg pcreg_LO(clk,rst,LO_W,LO_data_in,LO_data_out);

    pcreg pcreg(PC_CLK,rst,ena,PC_in,PC_out);

    alu alu(alu_a,alu_b,aluc,alu_c,zero,carry,negative,overflow);

    regfile cpu_ref(RF_CLK,rst,1'b1,RF_W,rsc,rtc,rdc,rd,rs,rt);

    CP0 CP0(clk,rst,MFC0,MTC0,CP0_pc,CP0_Rd,CP0_wdata,CP0_exception,ERE
T,CP0_cause,CP0_rdata,CP0_status,CP0_epc,CP0_exc_addr);


endmodule
```

5.3 alu //算术逻辑单元模块，实现多种逻辑运算与算术运算

```
module alu(
    input [31:0] a,
    input [31:0] b,
    input [3:0] aluc,
    output reg [31:0] r,
    output reg zero,
    output reg carry,
    output reg negative,
    output reg overflow
    );
    reg [31:0] oTmpt;
    always@(*)
        case(aluc)
            4'b0000:r=a+b;
            4'b0010:r=$signed(a)+$signed(b);
            4'b0001:r=a-b;
            4'b0011:r=a-b;
            4'b0100:r=a&b;
            4'b0101:r=a|b;
            4'b0110:r=a^b;
            4'b0111:r=~(a|b);
            4'b1000:r={b[15:0],16'b0};
            4'b1001:r={b[15:0],16'b0};
            4'b1011:r=($signed(a)<$signed(b))?1:0;
            4'b1010:r=(a<b)?1:0;
            4'b1100:r=$signed(b)>>>a;
            4'b1110:r=b<<a;
            4'b1111:r=b<<a;
```

```
        4'b1101:r=b>>a;
        default:r=32'h00000000;
    endcase

//zero
always@(*)
    if(aluc==4'b1011||aluc==4'b1010)
        if(a==b)
            zero=1'b1;
        else
            zero=1'b0;
    else if(r==0)
        zero=1'b1;
    else
        zero=1'b0;

//negative
always@(*)
    if(aluc==4'b0010||aluc==4'b0011)
        if($signed(r)<0)
            negative=1'b1;
        else
            negative=1'b0;
    else if(aluc==4'b1011)
        if(r==1'b1)
            negative=1'b1;
        else
            negative=1'b0;
    else
        if(r[31]==1'b1)
            negative=1'b1;
        else
            negative=1'b0;

//carry
always@(*)
    if(aluc==4'b0000)
        if(a[31]==1'b1&&b[31]==1'b1)
            carry=1'b1;
        else if(a[31]==1'b1&&b[31]==1'b0&&r[31]==1'b0)
            carry=1'b1;
        else if(a[31]==1'b0&&b[31]==1'b1&&r[31]==1'b0)
            carry=1'b1;
        else
```

```
                carry=1'b0;
        else if(aluc==4'b0001)
            if(a<b)
                carry=1'b1;
            else
                carry=1'b0;
        else if(aluc==4'b1010)
            if(a<b)
                carry=1'b1;
            else
                carry=1'b0;
        else if(aluc==4'b1100)
            begin
            oTmpt=$signed(b)>>>(a-1);
            carry=oTmpt[0];
            end
        else if(aluc==4'b1101)
            begin
            oTmpt=b>>>(a-1);
            carry=oTmpt[0];
            end
        else if(aluc==4'b1110||aluc==4'b1111)
            begin
            oTmpt=b<<(a-1);
            carry=oTmpt[31];
            end

    //overflow
    always@(*)
        if(aluc==4'b0010)
            if(a[31]==b[31]&&~r[31]==a[31])
                overflow=1'b1;
            else
                overflow=1'b0;
        else if(aluc==4'b0011)
            if(a[31]==0&&b[31]==1&&r[31]==1)
                overflow=1'b1;
            else if(a[31]==1&&b[31]==0&&r[31]==0)
                overflow=1'b1;
            else
                overflow=1'b0;

endmodule
```

5.4 Dram //数据存储器模块，用于向内存中读写数据

```
module Dram(
    input clk,
    input ena,
    input DM_W,
    input DM_R,
    input [31:0] addr,
    input [31:0] data_in,
    output [31:0] data_out
    );

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

    assign data_out = (ena && (DM_R || DM_W)) ? RAM[addr] : 32'hzzzzzzz
z;

    always @ (posedge clk)
    begin
        if(ena && (DM_W))
            RAM[addr] <= data_in;
    end

endmodule
```

5.5 pcreg //指令寄存器模块，每个时钟下降沿更新一次

```
module pcreg(
    input clk,
    input rst,
    input ena,

    input [31:0] data_in,
    output reg [31:0] data_out
    );
    always@(negedge clk or posedge rst)
        begin
            if(rst==1'b1)
                data_out=32'h00400000;
            else
                if(ena==1'b1)
                    data_out=data_in;
                else if(ena==1'b0)
                    data_out=data_out;
        end
endmodule
```

5.6 regfile //寄存器堆模块，包括 32 个寄存器，根据地址内容进行读写

```
module regfile(
    input clk,
    input rst,
    input en,
    input rf_write,
    input [4:0] rsc,
    input [4:0] rtc,
    input [4:0] rdc,
    input [31:0] rd,
    output [31:0] rs,
    output [31:0] rt
    );
    reg [31:0] array_reg[31:0];
    reg [5:0] i;
    assign rs = en ? array_reg[rsc] : 32'bz;
    assign rt = en ?array_reg[rtc] : 32'bz;
    always @(negedge clk or posedge rst) begin
      if (rst) begin
        for(i=0;i<32;i=i+1)
          array_reg[i] <= 0;
        end
    end
    always@(posedge clk) begin begin
      if (rf_write && en && (rdc != 0))
        array_reg[rdc] <= rd;
      end
    end
endmodule
```

5.7 MUX //四选一数据选择器

```
module MUX(
    input [31:0] iC0,
    input [31:0] iC1,
    input [31:0] iC2,
    input [31:0] iC3,
    input iS1,
    input iS0,
    output [31:0] oZ
    );
reg[31:0] tmpt;
always@(*)
begin
   if(iS1==0)
```

```
        begin
            if(iS0==0)
                tmpt = iC0;
            else
                tmpt = iC1;
        end
   else
        begin
            if(iS0==0)
                tmpt = iC2;
            else
                tmpt = iC3;
        end
end
assign oZ = tmpt;
endmodule
```

5.8 MUX_5bit //五位四选一数据选择器

```
module MUX_5bit(
    input [4:0] iC0,
    input [4:0] iC1,
    input [4:0] iC2,
    input [4:0] iC3,
    input iS1,
    input iS0,
    output [4:0] oZ
    );
reg[31:0] tmpt;
    always@(*)
    begin
       if(iS1==0)
            begin
                if(iS0==0)
                    tmpt = iC0;
                else
                    tmpt = iC1;
            end
       else
            begin
                if(iS0==0)
                    tmpt = iC2;
                else
                    tmpt = iC3;
```

```
            end
    end
    assign oZ = tmpt;
endmodule
```

5.9 MUX8 //八选一数据选择器

```
module MUX8(
    input [31:0] iC0,
    input [31:0] iC1,
    input [31:0] iC2,
    input [31:0] iC3,
    input [31:0] iC4,
    input [31:0] iC5,
    input [31:0] iC6,
    input [31:0] iC7,
    input iS2,
    input iS1,
    input iS0,
    output [31:0] oZ
    );
reg[31:0] tmpt;
always@(*)
begin
    if({iS2,iS1,iS0} == 3'b000)
        tmpt = iC0;
    else if({iS2,iS1,iS0} == 3'b001)
        tmpt = iC1;
    else if({iS2,iS1,iS0} == 3'b010)
        tmpt = iC2;
    else if({iS2,iS1,iS0} == 3'b011)
        tmpt = iC3;
    else if({iS2,iS1,iS0} == 3'b100)
        tmpt = iC4;
    else if({iS2,iS1,iS0} == 3'b101)
        tmpt = iC5;
    else if({iS2,iS1,iS0} == 3'b110)
        tmpt = iC6;
    else
        tmpt = iC7;

end
assign oZ = tmpt;
endmodule
```

5.10 Ext5 //将 5 位数据无符号扩展为 32 位

```
module Ext5(
    input [4:0] data_in,
    output [31:0] data_out
    );
    assign data_out = {27'b0,data_in};
endmodule
```

5.11 Ext8 //将 8 位数据无符号扩展为 32 位

```
module Ext8(
    input [7:0] data_in,
    output [31:0] data_out
    );
    assign data_out = {24'b0,data_in};
endmodule
```

5.12 Ext16 //将 16 位数据无符号扩展为 32 位

```
module Ext16(
    input [15:0] data_in,
    output [31:0] data_out
    );
    assign data_out = {16'b0,data_in};
endmodule
```

5.13 S_Ext8 //将 8 位数据有符号扩展为 32 位

```
module S_Ext8(
    input [7:0] data_in,
    output [31:0] data_out
    );
    assign data_out = {{24{data_in[7]}},data_in};
endmodule
```

5.14 S_Ext16 //将 16 位数据有符号扩展为 32 位

```
module S_Ext16(
    input [15:0] data_in,
    output [31:0] data_out
    );
    assign data_out = {{16{data_in[15]}},data_in};
```

```
endmodule
```

5.15 S_Ext18 //将 18 位数据有符号扩展为 32 位

```
module S_Ext18(
    input [17:0] data_in,
    output [31:0] data_out
    );
    assign data_out = {{14{data_in[17]}},data_in};
endmodule
```

5.16 MULT//有符号乘法器，将两数乘积扩展为有符号 64 位输出

```
module MULT(
input clk, //乘法器时钟信号
input reset, //复位信号，低电平有效
input [31:0] a, //输入数 a(被乘数)
input [31:0] b, //输入数 b（乘数）
output [63:0] z //乘积输出 z
);

  reg [63:0] temp;
  wire [31:0] a0,b0;//a，b 绝对值
  reg [31:0] a1=0,b1=0;//时钟时刻 a，b
  reg as=0,bs=0,zs=0;

  reg [63:0] stored00=0; reg [63:0] stored01=0; reg [63:0] stored02=0;
reg [63:0] stored03=0; reg [63:0] stored04=0; reg [63:0] stored05=0; re
g [63:0] stored06=0; reg [63:0] stored07=0;
  reg [63:0] stored08=0; reg [63:0] stored09=0; reg [63:0] stored10=0;
reg [63:0] stored11=0; reg [63:0] stored12=0; reg [63:0] stored13=0; re
g [63:0] stored14=0; reg [63:0] stored15=0;
  reg [63:0] stored16=0; reg [63:0] stored17=0; reg [63:0] stored18=0;
reg [63:0] stored19=0; reg [63:0] stored20=0; reg [63:0] stored21=0; re
g [63:0] stored22=0; reg [63:0] stored23=0;
  reg [63:0] stored24=0; reg [63:0] stored25=0; reg [63:0] stored26=0;
reg [63:0] stored27=0; reg [63:0] stored28=0; reg [63:0] stored29=0; re
g [63:0] stored30=0; reg [63:0] stored31=0;

 reg [63:0] add00_01, add02_03, add04_05, add06_07, add08_09, add10_11,
 add12_13, add14_15,add16_17, add18_19, add20_21, add22_23, add24_25, a
dd26_27, add28_29, add30_31;
```

```
 reg [63:0] add00_03, add04_07, add08_11, add12_15, add16_19, add20_23,
 add24_27, add28_31;

 reg [63:0] add00_07, add08_15, add16_23, add24_31;

 reg [63:0] add00_15, add16_31;

  assign a0 = as ? (~a1+1) : a1;
  assign b0 = bs ? (~b1+1) : b1;
  assign z = zs ? (~temp+1):temp;
  always @(*)
  begin
   if(reset)
   begin
     temp<=0;  as=0;bs=0;zs=0; a1=0;b1=0;
     stored00<=0; stored01<=0; stored02<=0; stored03<=0; stored04<=0; s
tored05<=0; stored06<=0; stored07<=0;
     stored08<=0; stored09<=0; stored10<=0; stored11<=0; stored12<=0; s
tored13<=0; stored14<=0; stored15<=0;
     stored16<=0; stored17<=0; stored18<=0; stored19<=0; stored20<=0; s
tored21<=0; stored22<=0; stored23<=0;
     stored24<=0; stored25<=0; stored26<=0; stored27<=0; stored28<=0; s
tored29<=0; stored30<=0; stored31<=0;

     add00_01 <= 0; add02_03 <= 0; add04_05 <= 0; add06_07 <= 0; add08_
09 <= 0; add10_11 <= 0; add12_13 <= 0; add14_15 <= 0;
     add16_17 <= 0; add18_19 <= 0; add20_21 <= 0; add22_23 <= 0; add24_
25 <= 0; add26_27 <= 0; add28_29 <= 0; add30_31 <= 0;

     add00_03 <= 0; add04_07 <= 0; add08_11 <= 0; add12_15 <= 0; add16_
19 <= 0; add20_23 <= 0; add24_27 <= 0; add28_31 <= 0;

     add00_07 <= 0; add08_15 <= 0; add16_23 <= 0; add24_31 <= 0;

     add00_15 <= 0; add16_31 <= 0;
   end
   else
   begin
    as <= a[31];    bs <= b[31];
    a1 <= a;        b1 <= b;

    stored00 <= b0[0]? {32'b0, a0} :64'b0;         stored01 <= b0[1]? {
31'b0, a0, 1'b0} :64'b0;
```

```
    stored02 <= b0[2]? {30'b0, a0, 2'b0} :64'b0;   stored03 <= b0[3]? {
29'b0, a0, 3'b0} :64'b0;
    stored04 <= b0[4]? {28'b0, a0, 4'b0} :64'b0;   stored05 <= b0[5]? {
27'b0, a0, 5'b0} :64'b0;
    stored06 <= b0[6]? {26'b0, a0, 6'b0} :64'b0;   stored07 <= b0[7]? {
25'b0, a0, 7'b0} :64'b0;
    stored08 <= b0[8]? {24'b0, a0, 8'b0} :64'b0;   stored09 <= b0[9]? {
23'b0, a0, 9'b0} :64'b0;
    stored10 <= b0[10]? {22'b0, a0, 10'b0} :64'b0; stored11 <= b0[11]?
{21'b0, a0, 11'b0} :64'b0;
    stored12 <= b0[12]? {20'b0, a0, 12'b0} :64'b0; stored13 <= b0[13]?
{19'b0, a0, 13'b0} :64'b0;
    stored14 <= b0[14]? {18'b0, a0, 14'b0} :64'b0; stored15 <= b0[15]?
{17'b0, a0, 15'b0} :64'b0;
    stored16 <= b0[16]? {16'b0, a0, 16'b0} :64'b0; stored17 <= b0[17]?
{15'b0, a0, 17'b0} :64'b0;
    stored18 <= b0[18]? {14'b0, a0, 18'b0} :64'b0; stored19 <= b0[19]?
{13'b0, a0, 19'b0} :64'b0;
    stored20 <= b0[20]? {12'b0, a0, 20'b0} :64'b0; stored21 <= b0[21]?
{11'b0, a0, 21'b0} :64'b0;
    stored22 <= b0[22]? {10'b0, a0, 22'b0} :64'b0; stored23 <= b0[23]?
{9'b0, a0, 23'b0} :64'b0;
    stored24 <= b0[24]? {8'b0, a0, 24'b0} :64'b0;  stored25 <= b0[25]?
{7'b0, a0, 25'b0} :64'b0;
    stored26 <= b0[26]? {6'b0, a0, 26'b0} :64'b0;  stored27 <= b0[27]?
{5'b0, a0, 27'b0} :64'b0;
    stored28 <= b0[28]? {4'b0, a0, 28'b0} :64'b0;  stored29 <= b0[29]?
{3'b0, a0, 29'b0} :64'b0;
    stored30 <= b0[30]? {2'b0, a0, 30'b0} :64'b0;  stored31 <= b0[31]?
{1'b0, a0, 31'b0} :64'b0;

    add00_01 <= stored00+stored01; add02_03 <= stored02+stored03; add04
_05 <= stored04+stored05; add06_07 <= stored06+stored07;
    add08_09 <= stored08+stored09; add10_11 <= stored10+stored11; add12
_13 <= stored12+stored13; add14_15 <= stored14+stored15;
    add16_17 <= stored16+stored17; add18_19 <= stored18+stored19; add20
_21 <= stored20+stored21; add22_23 <= stored22+stored23;
    add24_25 <= stored24+stored25; add26_27 <= stored26+stored27; add28
_29 <= stored28+stored29; add30_31 <= stored30+stored31;

    add00_03 <= add00_01+add02_03; add04_07 <= add04_05+add06_07; add08
_11 <= add08_09+add10_11; add12_15 <= add12_13+add14_15;
    add16_19 <= add16_17+add18_19; add20_23 <= add20_21+add22_23; add24
_27 <= add24_25+add26_27; add28_31 <= add28_29+add30_31;
```

```
    add00_07 <= add00_03+add04_07; add08_15 <= add08_11+add12_15; add16
_23 <= add16_19+add20_23; add24_31 <= add24_27+add28_31;

    add00_15 <= add00_07+add08_15; add16_31 <= add16_23+add24_31;

    temp <= add00_15+add16_31;

    zs<=as+bs;
   end
   end

endmodule
```

5.17 MULTU//无符号乘法器，将两数乘积扩展为无符号 64 位输出

```
module multu(
input clk, //乘法器时钟信号
input reset, //复位信号，低电平有效
input [31:0] a, //输入数 a(被乘数)
input [31:0] b, //输入数 b（乘数）
output [63:0] z //乘积输出 z
);

  reg [63:0] temp;

  reg [63:0] stored00; reg [63:0] stored01; reg [63:0] stored02; reg [6
3:0] stored03; reg [63:0] stored04; reg [63:0] stored05; reg [63:0] sto
red06; reg [63:0] stored07;
  reg [63:0] stored08; reg [63:0] stored09; reg [63:0] stored10; reg [6
3:0] stored11; reg [63:0] stored12; reg [63:0] stored13; reg [63:0] sto
red14; reg [63:0] stored15;
  reg [63:0] stored16; reg [63:0] stored17; reg [63:0] stored18; reg [6
3:0] stored19; reg [63:0] stored20; reg [63:0] stored21; reg [63:0] sto
red22; reg [63:0] stored23;
  reg [63:0] stored24; reg [63:0] stored25; reg [63:0] stored26; reg [6
3:0] stored27; reg [63:0] stored28; reg [63:0] stored29; reg [63:0] sto
red30; reg [63:0] stored31;

 reg [63:0] add00_01, add02_03, add04_05, add06_07, add08_09, add10_11,
 add12_13, add14_15,add16_17, add18_19, add20_21, add22_23, add24_25, a
dd26_27, add28_29, add30_31;
```

```
 reg [63:0] add00_03, add04_07, add08_11, add12_15, add16_19, add20_23,
 add24_27, add28_31;

 reg [63:0] add00_07, add08_15, add16_23, add24_31;

 reg [63:0] add00_15, add16_31;

  always @(*)
  begin
   if(reset)
   begin
     temp<=0;

     stored00<=0; stored01<=0; stored02<=0; stored03<=0; stored04<=0; s
tored05<=0; stored06<=0; stored07<=0;
     stored08<=0; stored09<=0; stored10<=0; stored11<=0; stored12<=0; s
tored13<=0; stored14<=0; stored15<=0;
     stored16<=0; stored17<=0; stored18<=0; stored19<=0; stored20<=0; s
tored21<=0; stored22<=0; stored23<=0;
     stored24<=0; stored25<=0; stored26<=0; stored27<=0; stored28<=0; s
tored29<=0; stored30<=0; stored31<=0;

     add00_01 <= 0; add02_03 <= 0; add04_05 <= 0; add06_07 <= 0; add08_
09 <= 0; add10_11 <= 0; add12_13 <= 0; add14_15 <= 0;
     add16_17 <= 0; add18_19 <= 0; add20_21 <= 0; add22_23 <= 0; add24_
25 <= 0; add26_27 <= 0; add28_29 <= 0; add30_31 <= 0;

     add00_03 <= 0; add04_07 <= 0; add08_11 <= 0; add12_15 <= 0; add16_
19 <= 0; add20_23 <= 0; add24_27 <= 0; add28_31 <= 0;

     add00_07 <= 0; add08_15 <= 0; add16_23 <= 0; add24_31 <= 0;

     add00_15 <= 0; add16_31 <= 0;
   end
   else
   begin
    stored00 <= b[0]? {32'b0, a} :64'b0;         stored01 <= b[1]? {31'
b0, a, 1'b0} :64'b0;
    stored02 <= b[2]? {30'b0, a, 2'b0} :64'b0;   stored03 <= b[3]? {29'
b0, a, 3'b0} :64'b0;
    stored04 <= b[4]? {28'b0, a, 4'b0} :64'b0;   stored05 <= b[5]? {27'
b0, a, 5'b0} :64'b0;
    stored06 <= b[6]? {26'b0, a, 6'b0} :64'b0;   stored07 <= b[7]? {25'
b0, a, 7'b0} :64'b0;
```

```
    stored08 <= b[8]? {24'b0, a, 8'b0} :64'b0;   stored09 <= b[9]? {23'
b0, a, 9'b0} :64'b0;
    stored10 <= b[10]? {22'b0, a, 10'b0} :64'b0; stored11 <= b[11]? {21
'b0, a, 11'b0} :64'b0;
    stored12 <= b[12]? {20'b0, a, 12'b0} :64'b0; stored13 <= b[13]? {19
'b0, a, 13'b0} :64'b0;
    stored14 <= b[14]? {18'b0, a, 14'b0} :64'b0; stored15 <= b[15]? {17
'b0, a, 15'b0} :64'b0;
    stored16 <= b[16]? {16'b0, a, 16'b0} :64'b0; stored17 <= b[17]? {15
'b0, a, 17'b0} :64'b0;
    stored18 <= b[18]? {14'b0, a, 18'b0} :64'b0; stored19 <= b[19]? {13
'b0, a, 19'b0} :64'b0;
    stored20 <= b[20]? {12'b0, a, 20'b0} :64'b0; stored21 <= b[21]? {11
'b0, a, 21'b0} :64'b0;
    stored22 <= b[22]? {10'b0, a, 22'b0} :64'b0; stored23 <= b[23]? {9'
b0, a, 23'b0} :64'b0;
    stored24 <= b[24]? {8'b0, a, 24'b0} :64'b0;  stored25 <= b[25]? {7'
b0, a, 25'b0} :64'b0;
    stored26 <= b[26]? {6'b0, a, 26'b0} :64'b0;  stored27 <= b[27]? {5'
b0, a, 27'b0} :64'b0;
    stored28 <= b[28]? {4'b0, a, 28'b0} :64'b0;  stored29 <= b[29]? {3'
b0, a, 29'b0} :64'b0;
    stored30 <= b[30]? {2'b0, a, 30'b0} :64'b0;  stored31 <= b[31]? {1'
b0, a, 31'b0} :64'b0;

    add00_01 <= stored00+stored01; add02_03 <= stored02+stored03; add04
_05 <= stored04+stored05; add06_07 <= stored06+stored07;
    add08_09 <= stored08+stored09; add10_11 <= stored10+stored11; add12
_13 <= stored12+stored13; add14_15 <= stored14+stored15;
    add16_17 <= stored16+stored17; add18_19 <= stored18+stored19; add20
_21 <= stored20+stored21; add22_23 <= stored22+stored23;
    add24_25 <= stored24+stored25; add26_27 <= stored26+stored27; add28
_29 <= stored28+stored29; add30_31 <= stored30+stored31;

    add00_03 <= add00_01+add02_03; add04_07 <= add04_05+add06_07; add08
_11 <= add08_09+add10_11; add12_15 <= add12_13+add14_15;
    add16_19 <= add16_17+add18_19; add20_23 <= add20_21+add22_23; add24
_27 <= add24_25+add26_27; add28_31 <= add28_29+add30_31;

    add00_07 <= add00_03+add04_07; add08_15 <= add08_11+add12_15; add16
_23 <= add16_19+add20_23; add24_31 <= add24_27+add28_31;

    add00_15 <= add00_07+add08_15; add16_31 <= add16_23+add24_31;
```

```
    temp <= add00_15+add16_31;

   end
   end

  assign z=temp;
endmodule
```

5.18 DIV//有符号乘法器，将两有符号数相除输出商和余数

```
module DIV(
    input [31:0] dividend,      //被除数
    input [31:0] divisor,       //除数
    input start,                //开始运算
    input clock,                //时钟
    input reset,                //复位
    output [31:0] q,            //商
    output [31:0] r,            //余数
    output reg busy             //除法器忙标志位
    );
    reg [5:0] count;            //计数器
    reg [31:0] reg_q;
    reg [31:0] reg_r;
    reg [31:0] reg_b;
    wire [31:0] reg_r2;
    reg r_sign;
    wire [32:0] sub_add = r_sign? ({reg_r, reg_q[31]} + {1'b0, reg_b}):
({reg_r, reg_q[31]} - {1'b0, reg_b});    //加减法器
    assign reg_r2 = r_sign ? reg_r + reg_b : reg_r;
    assign r = dividend[31] ? (~reg_r2 + 1) : reg_r2;
    assign q = (divisor[31]^dividend[31]) ? (~reg_q + 1) : reg_q;

    always @ (posedge clock or posedge reset)
    begin
        if (reset)
        begin
            count <= 0;                         //重置
            busy <= 0;
        end

        else
        begin
            if (start)                          //开始除法运算，初始化
            begin
```

```
                reg_r <= 32'b0;
                r_sign <= 0;
                if (dividend[31] == 1)
                    reg_q <= ~dividend+1;
                else
                    reg_q <= dividend;
                if(divisor[31] == 1)
                    reg_b <= ~divisor+1;
                else
                    reg_b <= divisor;
                count <= 0;
                busy <= 1;
            end
            else if (busy)
            begin
                reg_r <= sub_add[31:0];          //循环操作
                r_sign <= sub_add[32];           //部分余数
                reg_q <= {reg_q[30:0],~sub_add[32]};
                count <= count + 1;              //计数器加一
                if(count == 31)                  //结束除法运算
                    busy <= 0;
            end
        end
    end
endmodule
```

5.19 DIVU//无符号乘法器，将两无符号数相除输出商和余数

```
module DIVU(
    input [31:0] dividend,          //被除数
    input [31:0] divisor,           //除数
    input start,                    //开始运算
    input clock,                    //时钟
    input reset,                    //复位
    output [31:0] q,                //商
    output [31:0] r,                //余数
    output reg busy                 //除法器忙标志位
    );
    reg [4:0] count;
    reg [31:0] reg_q;
    reg [31:0] reg_r;
    reg [31:0] reg_b;
    reg r_sign;
```

```
    wire [32:0] sub_add = r_sign?({reg_r,q[31]} + {1'b0,reg_b}):({reg_r
,q[31]} - {1'b0,reg_b});    //加、减法器
    assign r = r_sign? reg_r + reg_b : reg_r;
    assign q = reg_q;

    always @ (posedge clock or posedge reset)
    begin
        if (reset == 1)
        begin                                                   //重置
            count <=5'b0;
            busy <= 0;
        end

        else
        begin
            if (start)
            begin                                               //开始除法运算，
初始化
                reg_r <= 32'b0;
                r_sign <= 0;
                reg_q <= dividend;
                reg_b <= divisor;
                count <= 5'b0;
                busy <= 1'b1;
            end
            else if (busy)
            begin                                               //循环操作
                reg_r <= sub_add[31:0];                         //部分余数
                r_sign <= sub_add[32];                          //如果为负，下次
相加
                reg_q <= {reg_q[30:0],~sub_add[32]};
                count <= count +5'b1;                           //计数器加一
                if (count == 31)
                    busy <= 0;                                  //结束除法运算
            end
        end
    end
endmodule
```

5.20 CLZ //实现输出数据前导 0 的个数

```
/*module CLZ(
    input [31:0] in,
    output [31:0] out
```

```
    );
    assign out = in[31]==1 ? 0:in[30]==1 ? 1:in[29]==1 ? 2:in[28]==1 ?
3:in[27]==1 ? 4:in[26]==1 ? 5:in[25]==1 ? 6:in[24]==1 ? 7:in[23]==1 ? 8
:
                in[22]==1 ? 9:in[21]==1 ? 10:in[20]==1 ? 11:in[19]==1 ?
 12:in[18]==1 ? 13:in[17]==1 ? 14:in[16]==1 ? 15:in[15]==1 ? 16:in[14]=
=1 ? 17:
                in[13]==1 ? 18:in[12]==1 ? 19:in[11]==1 ? 20:in[10]==1
? 21:in[9]==1 ? 22:in[8]==1 ? 23:in[7]==1 ? 24:in[6]==1 ? 25:in[5]==1 ?
 26:
                in[4]==1 ? 27:in[3]==1 ? 28:in[2]==1 ? 29:in[1]==1 ? 30
:in[0]==1 ? 31:32;

endmodule*/
module CLZ(
    input [31:0]num,
    output[31:0]cnt
);
assign cnt=num[31]==1?0:num[30]==1?1:num[29]==1?2:num[28]==1?3:num[27]=
=1?4:num[26]==1?5:num[25]==1?6:num[24]==1?7:num[23]==1?8:num[22]==1?9:n
um[21]==1?10:num[20]==1?11:num[19]==1?12:num[18]==1?13:num[17]==1?14:nu
m[16]==1?15:num[15]==1?16:num[14]==1?17:num[13]==1?18:num[12]==1?19:num
[11]==1?20:num[10]==1?21:num[9]==1?22:num[8]==1?23:num[7]==1?24:num[6]=
=1?25:num[5]==1?26:num[4]==1?27:num[3]==1?28:num[2]==1?29:num[1]==1?30:
num[0]==1?31:32;
endmodule
```

5.21 CP0 //对 CPU 发生中断或异常时的处理

```
module CP0(
    input clk,
    input rst,
    input mfc0,
    input mtc0,
    input [31:0]pc,
    input [4:0]addr,
    input [31:0]data,
    input exception,
    input eret,
    input [4:0]cause,
    output reg [31:0]CP0_out,
    output [31:0]status,
    output [31:0]epc_out,
    output [31:0]exc_addr
```

```
);
    reg [31:0]memory[31:0];
    integer i;
    initial begin
        for(i = 0; i <= 31; i = i + 1)
        begin
          if(i == 12)
              memory[i] = 32'hf;
          else
                memory[i] = 32'b0;
        end
    end

    reg exception_valid = 1'b0;
    always @(*) begin
        if(exception & memory[12][0]) begin
            case(cause[3:0])
                4'b1000: begin //systcall
                    if(memory[12][1]) begin
                        exception_valid = 1'b1;
                    end else begin
                        exception_valid = 1'b0;
                    end
                end
                4'b1001: begin //break
                    if(memory[12][2]) begin
                        exception_valid = 1'b1;
                    end else begin
                        exception_valid = 1'b0;
                    end
                end
                4'b1101: begin //teq
                    if(memory[12][3]) begin
                        exception_valid = 1'b1;
                    end else begin
                        exception_valid = 1'b0;
                    end
                end
                default: begin
                    exception_valid = 1'bx;
                end
            endcase
        end else begin
            exception_valid = 1'b0;
```

```
        end
    end

    reg in_exception = 1'b0;
    always @(negedge clk or posedge rst) begin
        if(rst) begin
            memory[12] <= 32'h0000000f;
            memory[13] <= 32'b0;
            memory[14] <= 32'b0;
            in_exception <= 1'b0;
        end else begin
            if(mtc0) begin
                memory[addr] <= data;
            end else begin
                if(exception_valid & (~in_exception)) begin
                    memory[12] <= {memory[12][26:0], 5'b0};
                    memory[13] <= {26'b0 ,cause, 2'b0};
                    memory[14] <= pc ;//- 32'h4;
                    in_exception <= 1'b1;
                end else if(eret & in_exception) begin
                    memory[12] <= {5'b0, memory[12][31:5]};
                    in_exception <= 1'b0;
                end
            end
        end
    end

    always @(*) begin
        if(mfc0) begin
            CP0_out = memory[addr];
        end else begin
            CP0_out = 32'bx;
        end
    end

    wire [31:0]cause_ = memory[13];
    assign status = memory[12];//status_reg;
    assign epc_out = memory[14];//epc_reg;

    assign exc_addr=32'h00400004;
endmodule
```

5.22 imem //指令寄存器模块，通过调用 ip 核实现

```
`timescale 1ns/1ps

(* DowngradeIPIdentifiedWarnings = "yes" *)
module imen (
  a,
  spo
);

input wire [10 : 0] a;
output wire [31 : 0] spo;

  dist_mem_gen_v8_0_10 #(
    .C_FAMILY("artix7"),
    .C_ADDR_WIDTH(11),
    .C_DEFAULT_DATA("0"),
    .C_DEPTH(2048),
    .C_HAS_CLK(0),
    .C_HAS_D(0),
    .C_HAS_DPO(0),
    .C_HAS_DPRA(0),
    .C_HAS_I_CE(0),
    .C_HAS_QDPO(0),
    .C_HAS_QDPO_CE(0),
    .C_HAS_QDPO_CLK(0),
    .C_HAS_QDPO_RST(0),
    .C_HAS_QDPO_SRST(0),
    .C_HAS_QSPO(0),
    .C_HAS_QSPO_CE(0),
    .C_HAS_QSPO_RST(0),
    .C_HAS_QSPO_SRST(0),
    .C_HAS_SPO(1),
    .C_HAS_WE(0),
    .C_MEM_INIT_FILE("imen.mif"),
    .C_ELABORATION_DIR("./"),
    .C_MEM_TYPE(0),
    .C_PIPELINE_STAGES(0),
    .C_QCE_JOINED(0),
    .C_QUALIFY_WE(0),
    .C_READ_MIF(1),
    .C_REG_A_D_INPUTS(0),
    .C_REG_DPRA_INPUT(0),
    .C_SYNC_ENABLE(1),
    .C_WIDTH(32),
```

```
    .C_PARSER_TYPE(1)
  ) inst (
    .a(a),
    .d(32'B0),
    .dpra(11'B0),
    .clk(1'D0),
    .we(1'D0),
    .i_ce(1'D1),
    .qspo_ce(1'D1),
    .qdpo_ce(1'D1),
    .qdpo_clk(1'D0),
    .qspo_rst(1'D0),
    .qdpo_rst(1'D0),
    .qspo_srst(1'D0),
    .qdpo_srst(1'D0),
    .spo(spo),
    .dpo(),
    .qspo(),
    .qdpo()
  );
endmodule
```

六、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

```
`timescale 1ns / 1ps
module cpu_tb(

    );
        reg clk_in;
        reg reset;
        reg start;
        wire[31:0] inst;
        wire[31:0] pc;


        sccomp_dataflow sc(clk_in, reset, inst, pc);

        /*wire [31:0] M1_out = sc.sccpu.M1_out;
        wire [31:0] NPC = sc.sccpu.NPC;
        wire [31:0] rd = sc.sccpu.rd;
        wire [31:0] rs = sc.sccpu.rs;
```

```
wire [31:0] rt = sc.sccpu.rt;
wire [31:0] alu_a = sc.sccpu.alu_a;
wire [31:0] alu_b = sc.sccpu.alu_b;
wire [31:0] alu_c = sc.sccpu.alu_c;
wire [31:0] Ext5 = sc.sccpu.Ext5;
wire [31:0] Ext16 = sc.sccpu.Ext16;
wire [31:0] S_Ext16 = sc.sccpu.S_Ext16;
wire [31:0] S_Ext18 = sc.sccpu.S_Ext18;
wire [4:0] rsc = sc.sccpu.rsc;
wire [4:0] rtc = sc.sccpu.rtc;
wire M4_1 = sc.sccpu.M4_1;
wire M4_0 = sc.sccpu.M4_0;

wire [31:0] DM_addr = sc.DM_addr;
wire [31:0] dm_addr = sc.dm_addr;

wire CS = sc.CS;
wire DM_R = sc.DM_R;*/
wire [31:0] CLZ_out = sc.sccpu.CLZ_out;
//wire [31:0] M8_out = sc.sccpu.M8_out;
wire [31:0] rd = sc.sccpu.rd;
wire [31:0] rs = sc.sccpu.rs;
wire [31:0] rsc = sc.sccpu.rsc;
wire [31:0] rtc = sc.sccpu.rtc;
wire [31:0] rdc = sc.sccpu.rdc;
wire [31:0] RF_W = sc.sccpu.RF_W;

wire [31:0] CP0_rdata = sc.sccpu.CP0_rdata;
wire MTHI = sc.sccpu.MTHI;
wire MFHI = sc.sccpu.MFHI;
wire [31:0] LO_data_out = sc.sccpu.LO_data_out;
//wire M_LO = sc.sccpu.M_LO;
wire LO_W = sc.sccpu.LO_W;
wire [31:0] LO_data_in = sc.sccpu.LO_data_in;

wire [63:0] MULT_z = sc.sccpu.MULT_z;
wire [31:0] M9_out = sc.sccpu.M9_out;
wire [31:0] M8_out = sc.sccpu.M8_out;
wire [31:0] M2_out = sc.sccpu.M2_out;
wire [31:0] M7_out = sc.sccpu.M7_out;
wire [31:0] multu_z = sc.sccpu.multu_z;

wire M8_2 = sc.sccpu.M8_2;
wire M8_1 = sc.sccpu.M8_1;
```

```
wire M8_0 = sc.sccpu.M8_0;

wire [31:0] DExt8 = sc.sccpu.DExt8;
wire [31:0] DS_Ext8 = sc.sccpu.DS_Ext8;
wire [31:0] DExt16 = sc.sccpu.DExt16;
wire [31:0] DS_Ext16 = sc.sccpu.DS_Ext16;

wire [31:0] DM_RData = sc.sccpu.DM_RData;
wire [31:0] DM_WData = sc.sccpu.DM_WData;
wire [31:0] DM_addr = sc.sccpu.DM_addr;
wire [31:0] rt = sc.sccpu.rt;
wire DM_R = sc.sccpu.DM_R;
wire DM_W = sc.sccpu.DM_W;
wire CS = sc.sccpu.CS;

wire [31:0] DS_Ext8in = sc.sccpu.DS_Ext8in;

wire LB = sc.sccpu.LB;
wire LBU = sc.sccpu.LBU;
wire LHU = sc.sccpu.LHU;
wire SB = sc.sccpu.SB;
wire SH = sc.sccpu.SH;
wire LH = sc.sccpu.LH;

wire [31:0] M1_out = sc.sccpu.M1_out;
wire M1_2 = sc.sccpu.M1_2;
wire M1_1 = sc.sccpu.M1_1;
wire M1_0 = sc.sccpu.M1_0;
wire negative = sc.sccpu.negative;
wire BGEZ = sc.sccpu.BGEZ;

wire [31:0] M3_out = sc.sccpu.M3_out;
wire [31:0] M4_out = sc.sccpu.M4_out;

wire [31:0] CP0_exc_addr = sc.sccpu.CP0_exc_addr;


integer file_output;
integer counter = 0;

initial
begin
    file_output = $fopen("D:/MyResult.txt");
end
```

```
        initial
        begin
            clk_in = 1;
            start  = 0;
            forever begin
                #50 clk_in = ~clk_in;
            end
        end

        initial begin
            reset     = 0;
            #6 reset  = 1;
            #50 reset = 0;
            start = 1;
        end

        always @(negedge clk_in) begin
            if (start)begin
                counter  = counter + 1;

                $fdisplay(file_output, "pc: %h", pc);
                $fdisplay(file_output, "instr: %h", inst);

                $fdisplay(file_output, "regfile0: %h", sc.sccpu.cpu_ref
.array_reg[0]);
                $fdisplay(file_output, "regfile1: %h", sc.sccpu.cpu_ref
.array_reg[1]);
                $fdisplay(file_output, "regfile2: %h", sc.sccpu.cpu_ref
.array_reg[2]);
                $fdisplay(file_output, "regfile3: %h", sc.sccpu.cpu_ref
.array_reg[3]);
                $fdisplay(file_output, "regfile4: %h", sc.sccpu.cpu_ref
.array_reg[4]);
                $fdisplay(file_output, "regfile5: %h", sc.sccpu.cpu_ref
.array_reg[5]);
                $fdisplay(file_output, "regfile6: %h", sc.sccpu.cpu_ref
.array_reg[6]);
                $fdisplay(file_output, "regfile7: %h", sc.sccpu.cpu_ref
.array_reg[7]);
                $fdisplay(file_output, "regfile8: %h", sc.sccpu.cpu_ref
.array_reg[8]);
                $fdisplay(file_output, "regfile9: %h", sc.sccpu.cpu_ref
.array_reg[9]);
```

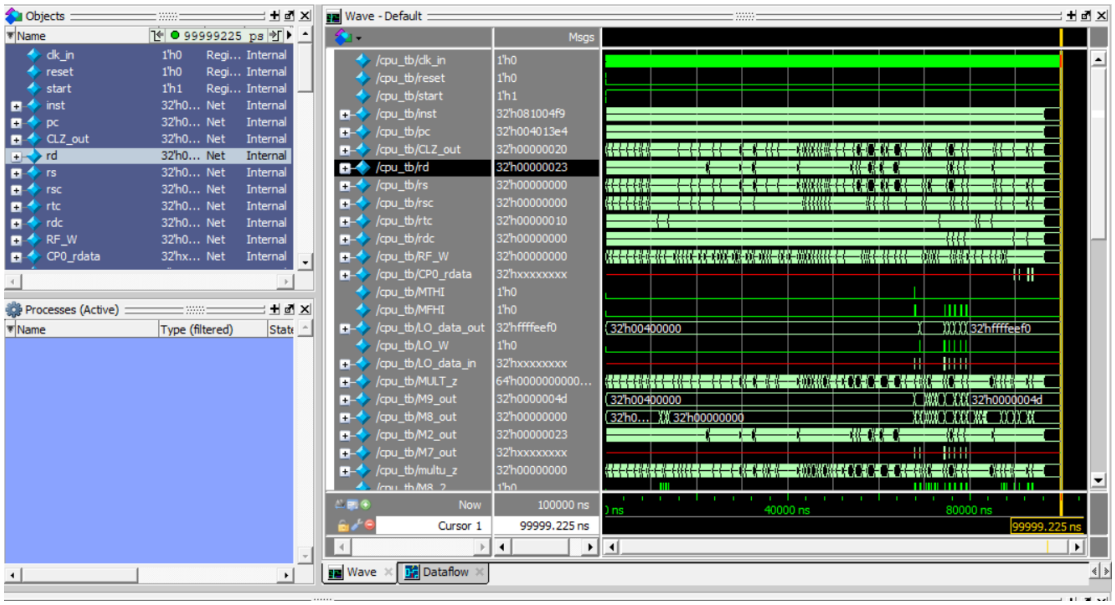
```
                $fdisplay(file_output, "regfile10: %h", sc.sccpu.cpu_re
f.array_reg[10]);
                $fdisplay(file_output, "regfile11: %h", sc.sccpu.cpu_re
f.array_reg[11]);
                $fdisplay(file_output, "regfile12: %h", sc.sccpu.cpu_re
f.array_reg[12]);
                $fdisplay(file_output, "regfile13: %h", sc.sccpu.cpu_re
f.array_reg[13]);
                $fdisplay(file_output, "regfile14: %h", sc.sccpu.cpu_re
f.array_reg[14]);
                $fdisplay(file_output, "regfile15: %h", sc.sccpu.cpu_re
f.array_reg[15]);
                $fdisplay(file_output, "regfile16: %h", sc.sccpu.cpu_re
f.array_reg[16]);
                $fdisplay(file_output, "regfile17: %h", sc.sccpu.cpu_re
f.array_reg[17]);
                $fdisplay(file_output, "regfile18: %h", sc.sccpu.cpu_re
f.array_reg[18]);
                $fdisplay(file_output, "regfile19: %h", sc.sccpu.cpu_re
f.array_reg[19]);
                $fdisplay(file_output, "regfile20: %h", sc.sccpu.cpu_re
f.array_reg[20]);
                $fdisplay(file_output, "regfile21: %h", sc.sccpu.cpu_re
f.array_reg[21]);
                $fdisplay(file_output, "regfile22: %h", sc.sccpu.cpu_re
f.array_reg[22]);
                $fdisplay(file_output, "regfile23: %h", sc.sccpu.cpu_re
f.array_reg[23]);
                $fdisplay(file_output, "regfile24: %h", sc.sccpu.cpu_re
f.array_reg[24]);
                $fdisplay(file_output, "regfile25: %h", sc.sccpu.cpu_re
f.array_reg[25]);
                $fdisplay(file_output, "regfile26: %h", sc.sccpu.cpu_re
f.array_reg[26]);
                $fdisplay(file_output, "regfile27: %h", sc.sccpu.cpu_re
f.array_reg[27]);
                $fdisplay(file_output, "regfile28: %h", sc.sccpu.cpu_re
f.array_reg[28]);
                $fdisplay(file_output, "regfile29: %h", sc.sccpu.cpu_re
f.array_reg[29]);
                $fdisplay(file_output, "regfile30: %h", sc.sccpu.cpu_re
f.array_reg[30]);
                $fdisplay(file_output, "regfile31: %h", sc.sccpu.cpu_re
f.array_reg[31]);
```

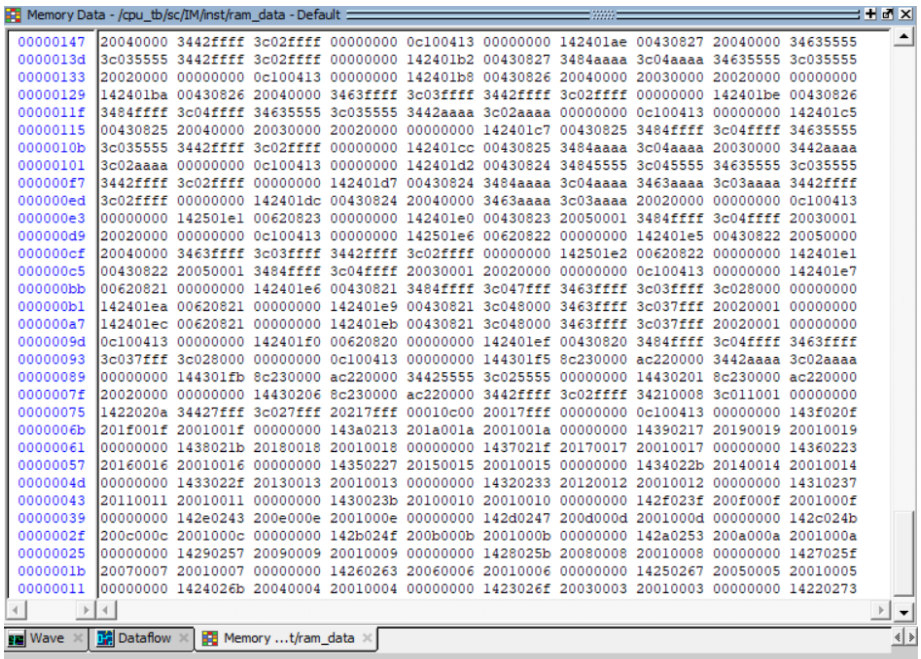
```
            end
        end
endmodule
```

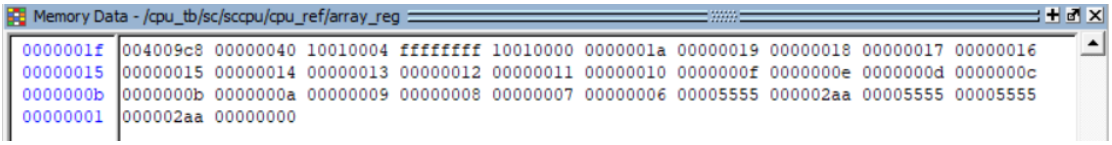
七、实验结果

6.1 前仿真

6.1.1 modelsim 后仿真波形图

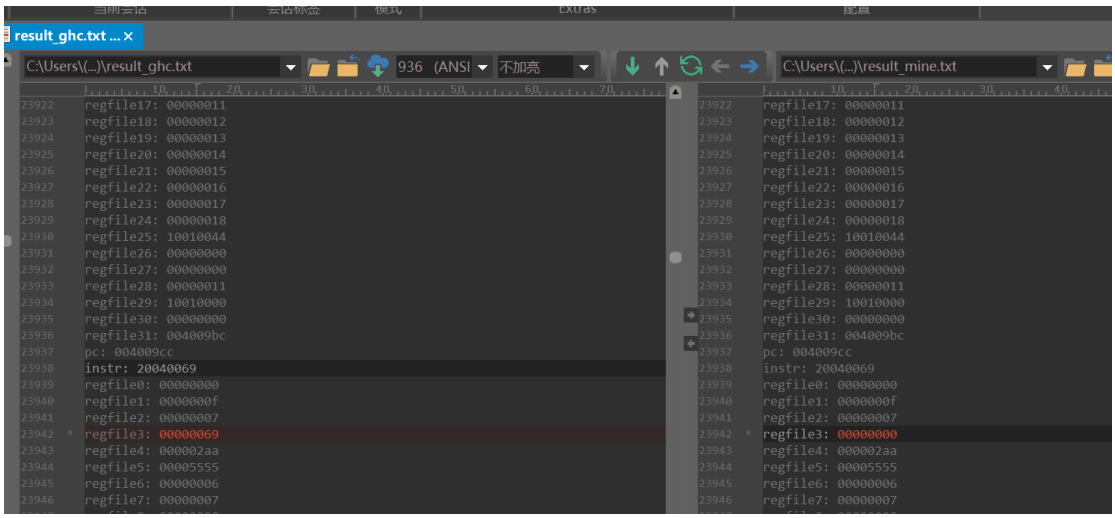




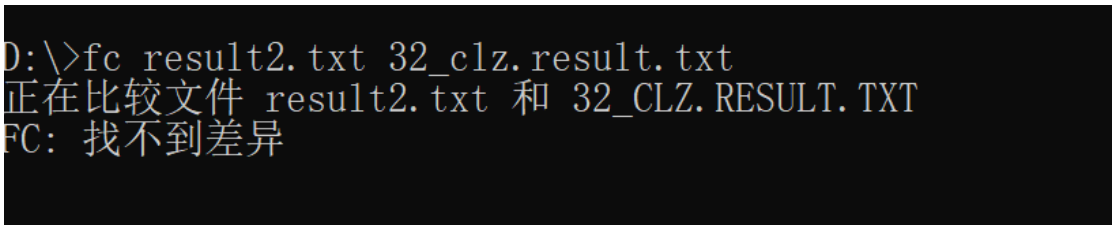


由上图我们可以看到，指令寄存器 IM 内存已读入相关指令译码后的结果，指令寄存器运行正常。同时寄存器堆 regfile 中已成功写入数据，寄存器堆也正常工作。

6.1.2 指令执行结果比对



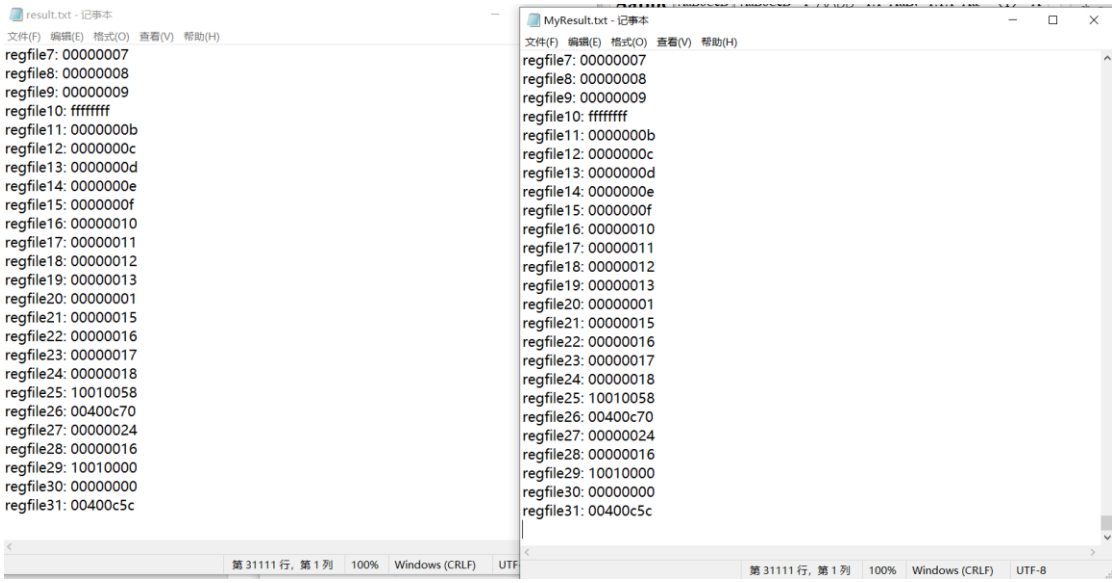
我通过采用 cmd 控制台的比较文件指令分别对 31 条指令以及最终版指令运行结果与标准结果进行比对，依次确认比对结果与标准结果并无差异以确认指令运行正确，CPU 设计正确。



```
D:\>fc result2.txt 36.39_lbsb2.result.txt /N
正在比较文件 result2.txt 和 36.39_LBSB2.RESULT.TXT
FC: 找不到差异


D:\>fc result2.txt 37_lbu.result.txt
正在比较文件 result2.txt 和 37_LBU.RESULT.TXT
FC: 找不到差异


D:\>fc result2.txt 37_lbu2.result.txt
正在比较文件 result2.txt 和 37_LBU2.RESULT.TXT
FC: 找不到差异


D:\>fc result2.txt 38_lhu.result.txt
正在比较文件 result2.txt 和 38_LHU.RESULT.TXT
FC: 找不到差异


D:\>fc result2.txt 38_lhu2.result.txt
正在比较文件 result2.txt 和 38_LHU2.RESULT.TXT
FC: 找不到差异


D:\>fc result2.txt 40.41_lhsh.result.txt
正在比较文件 result2.txt 和 40.41_LHSH.RESULT.TXT
FC: 找不到差异
```

```
D:\>fc result.txt MyResult.txt
正在比较文件 result.txt 和 MYRESULT.TXT
FC: 找不到差异
```

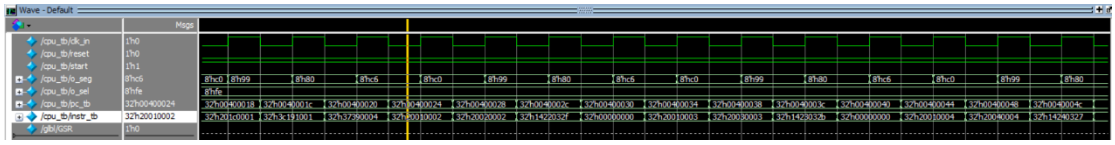
通过文件比对可以看出运行结果与标准并无差异，唯一一处不同在于运行结尾循环终止出，由此证明运行正常。



6.2 后仿真

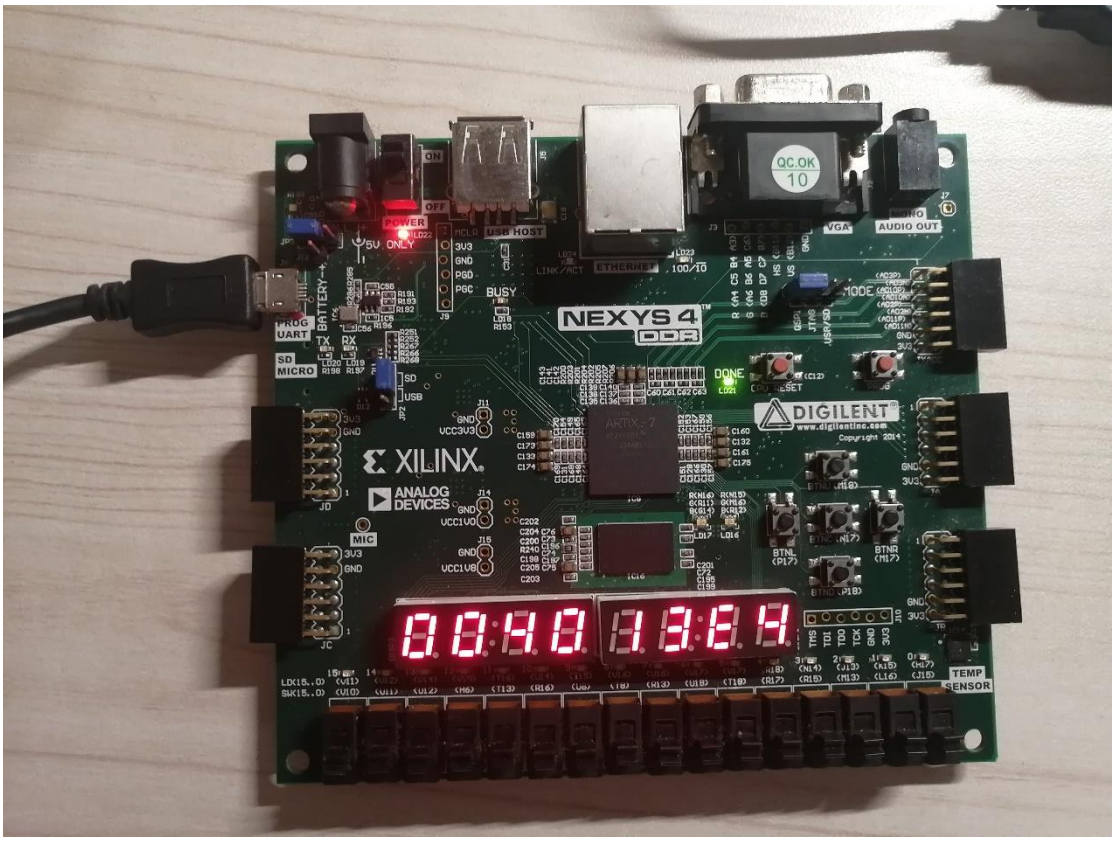
6.2.1 modelsim 后仿真波形图

6.2.2 相应 modelsim 前仿真波形图



由 modelsim 仿真结果我们可以看出前仿真与后仿真波形图基本一致，唯一不同的点在于后仿真波形图 pc 与 instr 部分有略微的延迟。

6.3 下板验证



可以看出下板后七段数码管可以稳定显示程序中的 PC 值，故下板成功。

八、总结与体会

总体来说，由于有了 31 条指令 CPU 的设计基础以及设计流程的过程熟悉。本次 54 条指令 CPU 设计相对来说更有方向，设计初期画图、列部件、写表达式等过程顺畅了许多。首先是 CPU 数据通路的设计，我根据指令的功能，确定每条指令所用到的部件；其次根据各个指令所用的部件，表格列出，同时在表格中填入每个部件的数据输入来源，再根据数据输入来源画出每条指令的数据通路，最后将所有指令数据通路合并成一个总的数据通路。合并过程涉及数据选择器 MUX，对于同一部件有不同输入来源的情况，根据数据选择器输入选择端决定在不同指令执行情况下将什么输入来源送到该部件。

由于本次 54 条指令 CPU 实验涉及到了 CPU 内部对异常以及中断的处理，需要我们首先设计一个中断处理器 CP0，我对 CPU 内部中断处理的机制、寄存器等内部构成有了一定的了解与认识，对 CPU 的整体框架有了更加深刻的理解。

在设计过程中，我遇到了许多由于疏忽而产生的 bug，有些错误甚至极难发现修改，debug 过程还是比较艰难痛苦的。首先是在将数据通路与控制器通过代码呈现时无法正常读入诸如 IP 核中的指令数据等，导致 modelsim 仿真以及寄存器结果文件输出无有效值，均为未知值 x 或 z。我通过排查数据通路是否连接、命名是否正确，最终发现问题出在顶层模块调用子模块时命名出现了错误，instr 写成了 inst，导致无法正常写入。更多的错误出在依次对单条指令进行执行比对过程中寄存器结果、地址、程序计数器出现不一致的情况。具体的原因包括：控制信号逻辑表达式有误、数据选择器有误，经过仔细排查，重新设计更改指令数

据通路，重新写不同微指令的逻辑表达式，最终解决了相关问题。这个过程需要对出错处的指令进行分析，将该指令所涉及到的部件控制信号在 tb 文件中利用 Modelsim 打印出来分析出错原因进行倒退回溯，最终寻找出出错原因。

CPU 的设计以及后期的 debug 过程让我深刻感受到了设计一个已有指令的 CPU 的过程，也通过自己设计一个 CPU 对 CPU 的工作机理有了深刻的认识，总之收获颇丰。