同济大学计算机系

计算机组成原理课程综合实验报告

学　　号 2252429

姓　　名 蔡宇轩

专　　业 计算机科学与技术

授课老师 陈永生

目录

一、 实验内容 1
二、 CPU 数据通路设计 1
（一） 单条指令的涉及部件和输入输出关系、数据通路和指令流程图 1
（二） 完整数据通路 19
三、 CPU 控制部件设计 19
（一） 指令操作时间表 19
（二） 控制信号逻辑表达式 27
（三） 控制部件设计 28
四、 CPU 前仿真测试结果 38
（一） 单条指令前仿真波形图及对比结果 38
（二） 网站前仿真测试结果 53
五、 CPU 后仿真测试结果 53
（一） 后仿真测试结果及描述 53
（二） 时序分析结果 53
六、 CPU 下板结果 55
七、 心得体会及建议 58

一、实验内容

本次实验我们将使用 verilog HDL 语言实现 54 条 MIPS 指令的 CPU 设计和仿真，可以用单周期完成，也可以用多周期完成。本次实验采用单周期实现，根据实验要求，依次完成：

（1）新建 vivado 工程，也可以在 MIPS31 条指令 CPU 基础上编写各模块

（2）用 modelsim 前仿真逐条测试所有指令

（3）用 modelsim 前仿真逐条测试所有指令边界数据

（4）用 modelsim 前仿真测试指令序列

（5）用 modelsim 前仿真运行测试程序

（6）用 modelsim 后仿真测试指令序列

（7）用 modelsim 后仿真运行测试程序

（8）配置 XDC 文件，综合下板，并观察实验现象

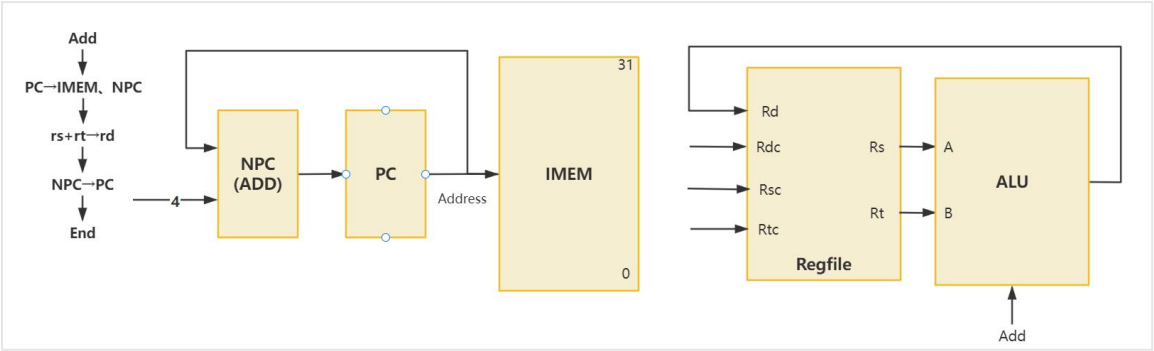
（9）按照要求书写实验报告

二、CPU 数据通路设计

（一）单条指令的涉及部件和输入输出关系、数据通路和指令流程图

1. ADD

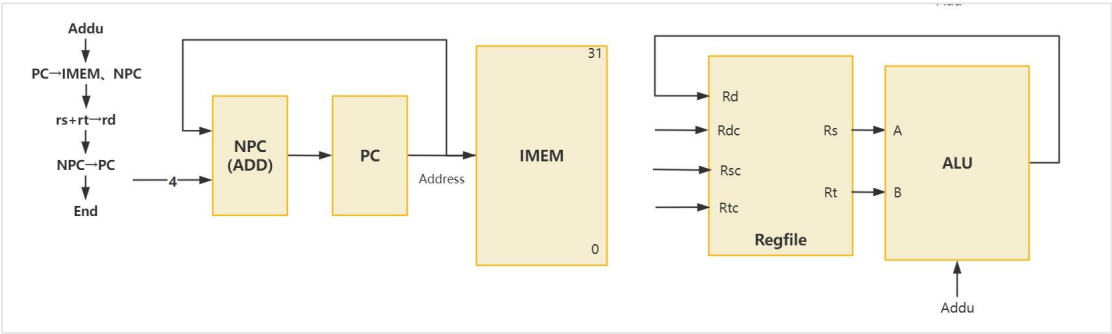
指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
ADD	NPC	PC	PC	ALU	Rs	Rt



2. ADDU

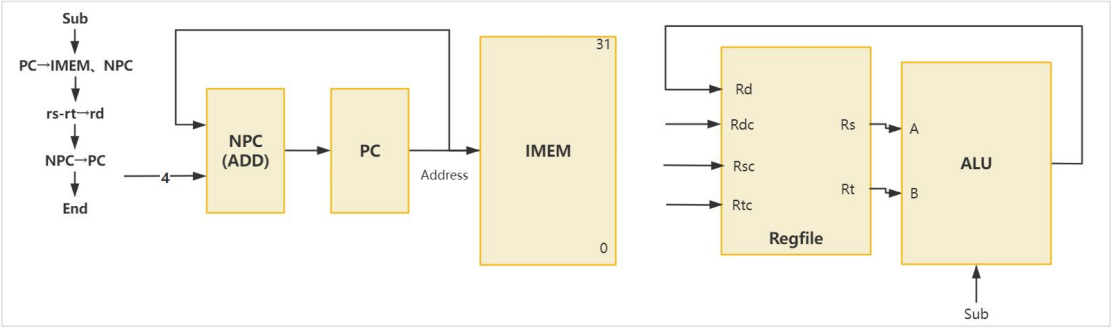
指令	PC	NPC	IMEM	Regfile	ALU

				Rd	A	B
ADDU	NPC	PC	PC	ALU	Rs	Rt



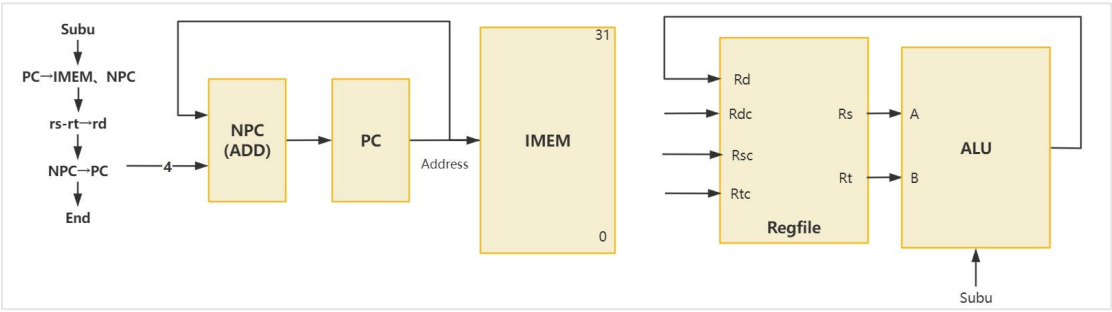
3. SUB

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
SUB	NPC	PC	PC	ALU	Rs	Rt



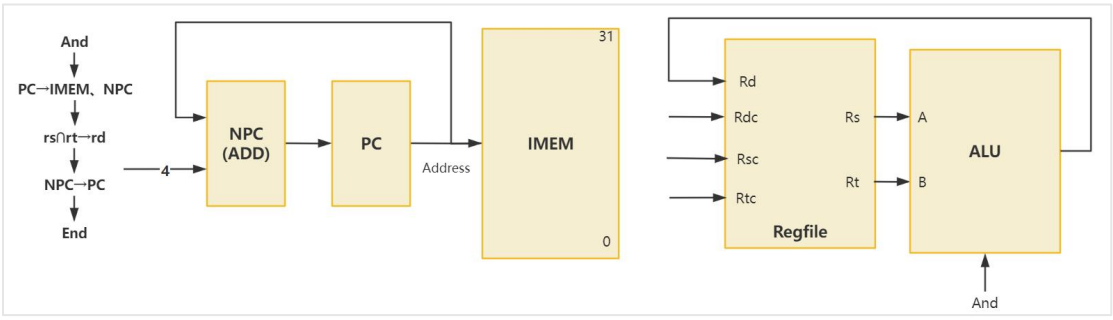
4. SUBU

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
SUBU	NPC	PC	PC	ALU	Rs	Rt



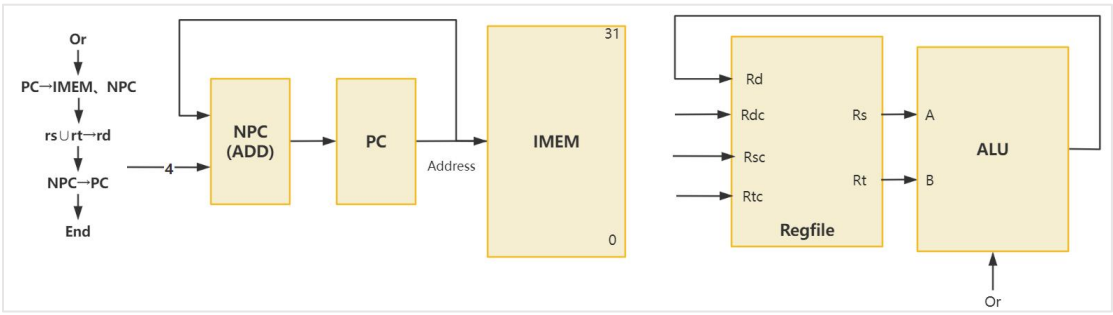
5. AND

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
AND	NPC	PC	PC	ALU	Rs	Rt



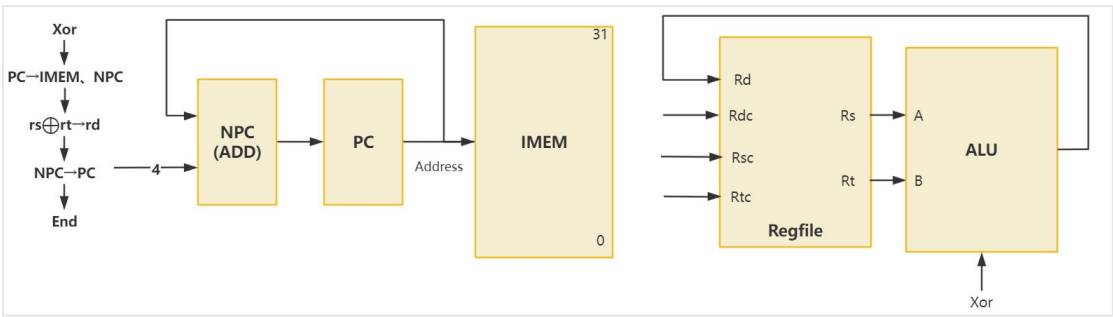
6. OR

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
OR	NPC	PC	PC	ALU	Rs	Rt



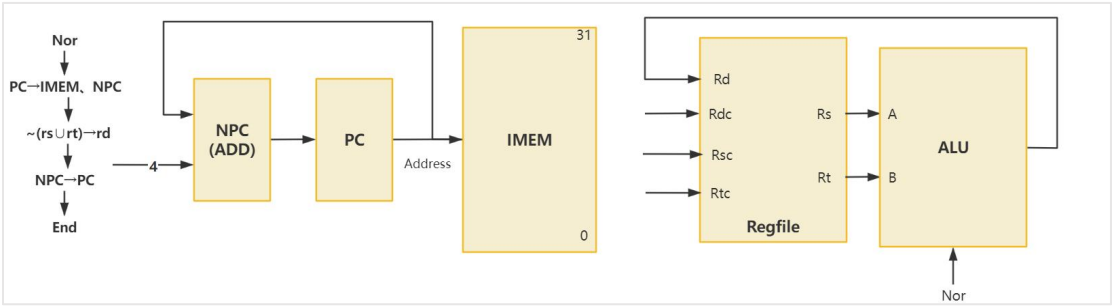
7. XOR

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
XOR	NPC	PC	PC	ALU	Rs	Rt



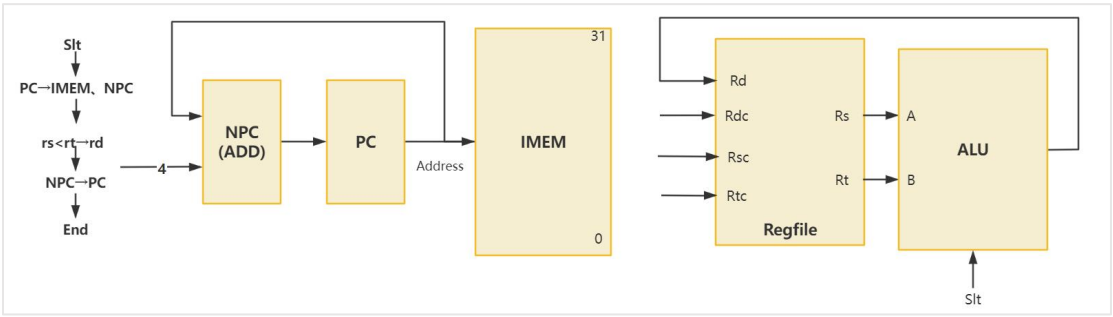
8. NOR

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
NOR	NPC	PC	PC	ALU	Rs	Rt



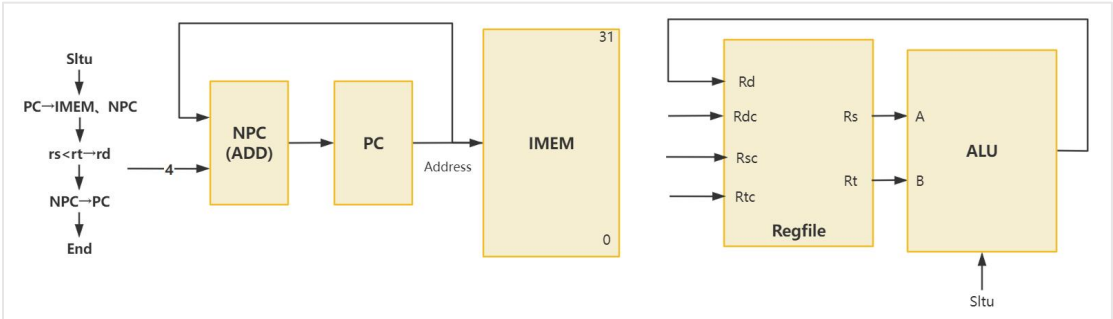
9. SLT

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
SLT	NPC	PC	PC	ALU	Rs	Rt



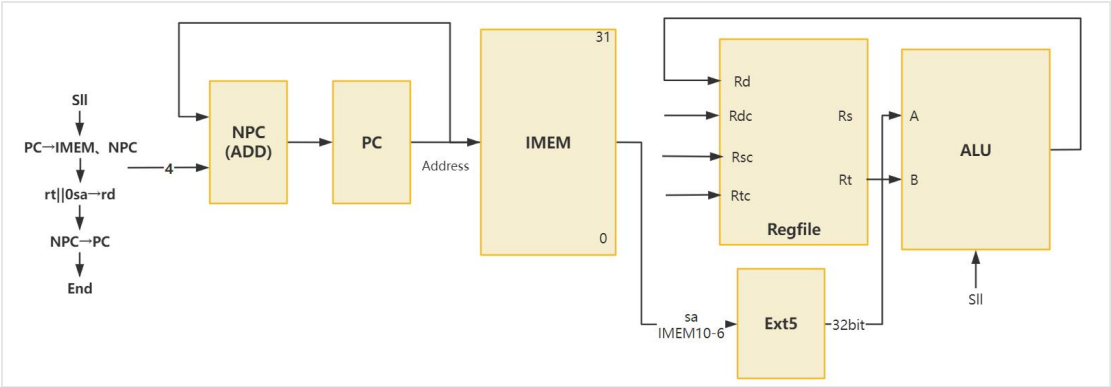
10. SLTU

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
SLTU	NPC	PC	PC	ALU	Rs	Rt



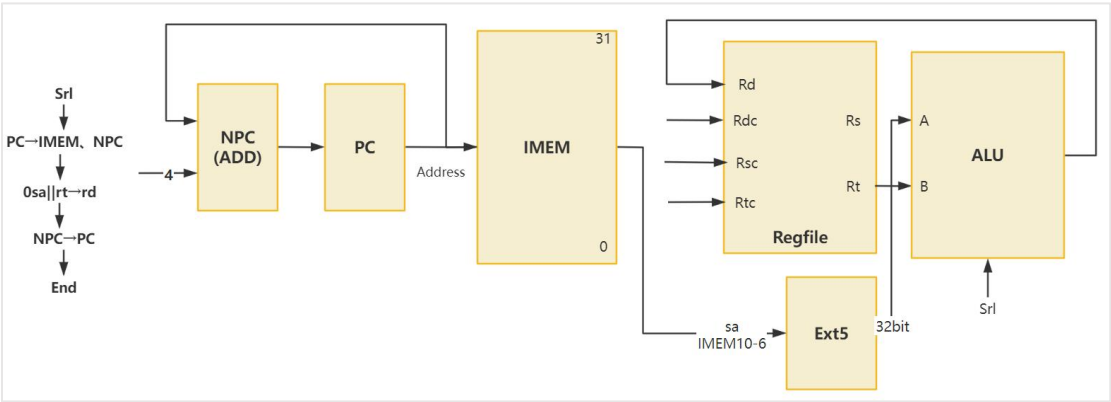
11. SLL

指令	PC	NPC	IMEM	Regfile	ALU		Ext5
				Rd	A	B	
SLL	NPC	PC	PC	ALU	Ext5	Rt	Sa



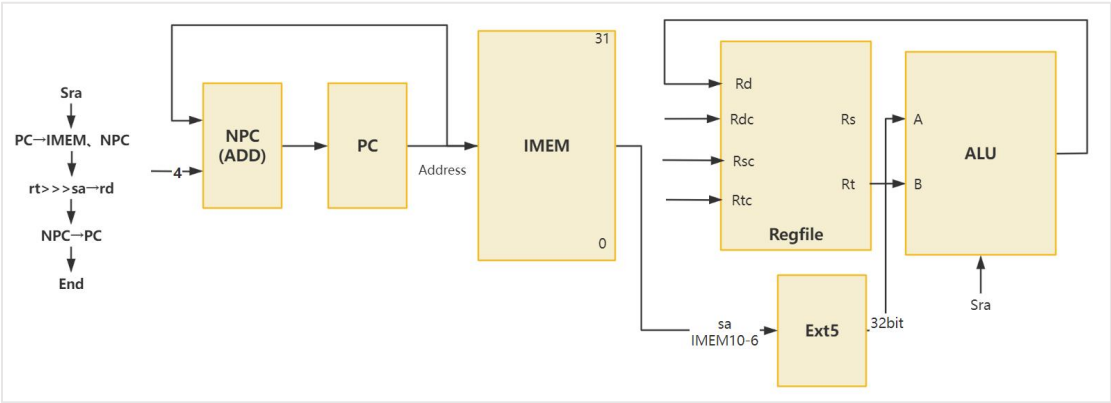
12. SRL

指令	PC	NPC	IMEM	Regfile	ALU		Ext5
				Rd	A	B	
SRL	NPC	PC	PC	ALU	Ext5	Rt	Sa



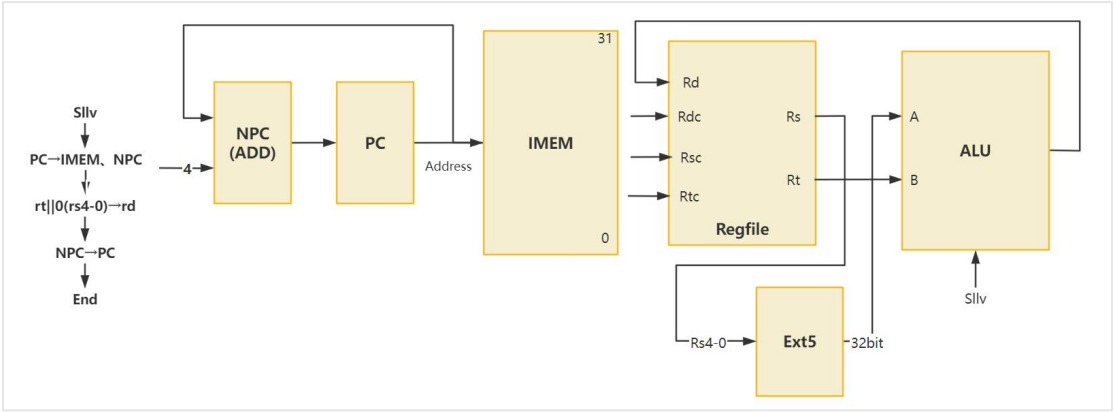
13. SRA

指令	PC	NPC	IMEM	Regfile	ALU		Ext5
				Rd	A	B	
SRA	NPC	PC	PC	ALU	Ext5	Rt	Sa



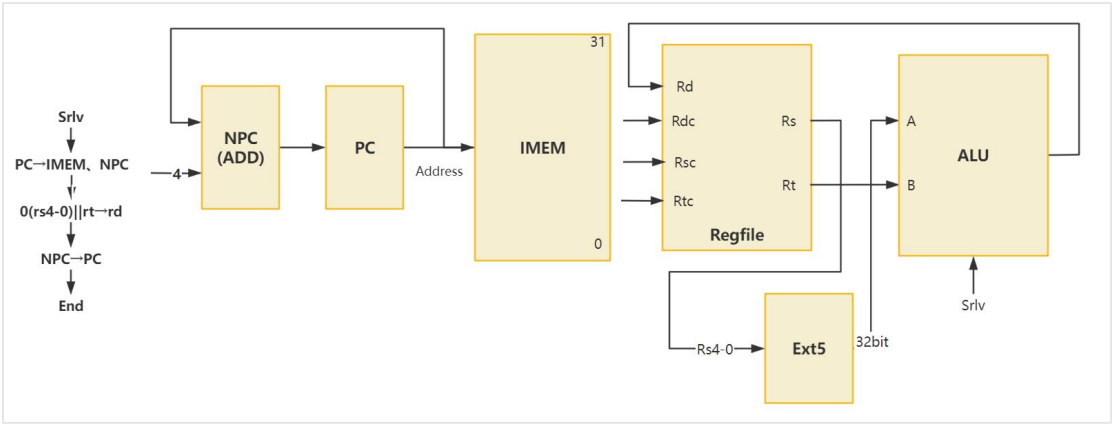
14. SLLV

指令	PC	NPC	IMEM	Regfile	ALU		Ext5
				Rd	A	B	
SLLV	NPC	PC	PC	ALU	Ext5	Rt	Rs



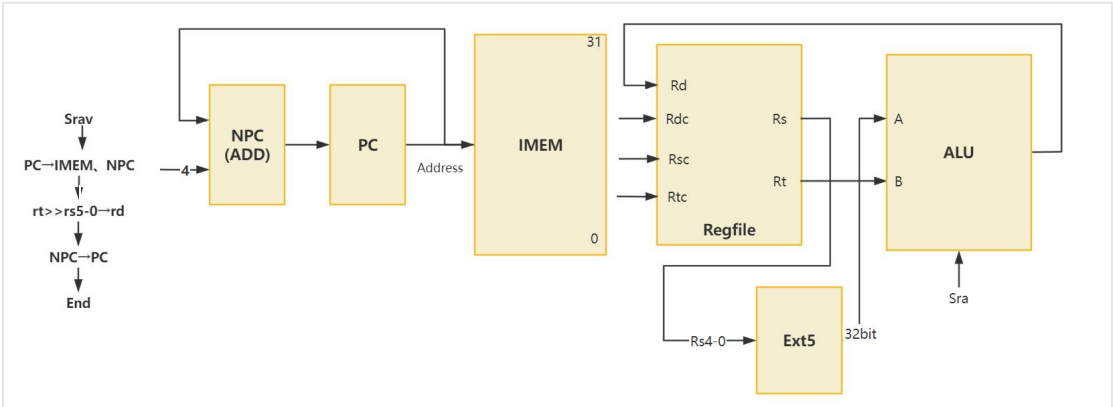
15. SRLV

指令	PC	NPC	IMEM	Regfile	ALU		Ext5
				Rd	A	B	
SRLV	NPC	PC	PC	ALU	Ext5	Rt	Rs



16. SRAV

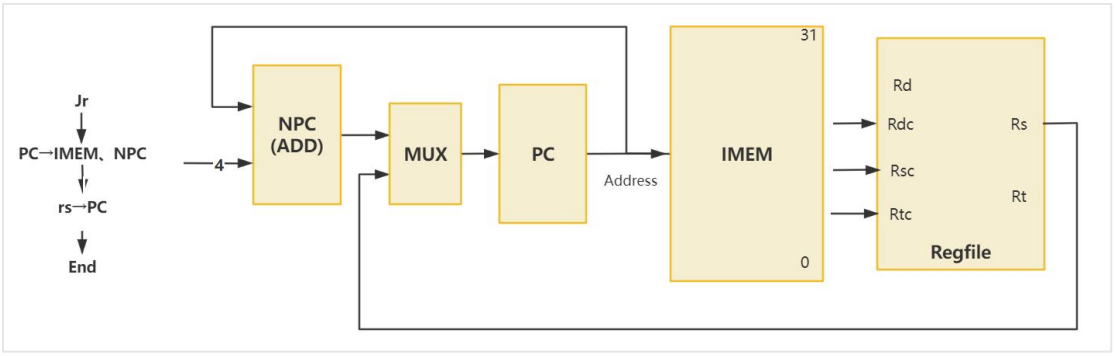
指令	PC	NPC	IMEM	Regfile	ALU		Ext5
				Rd	A	B	
SRAV	NPC	PC	PC	ALU	Ext5	Rt	Rs



17. JR

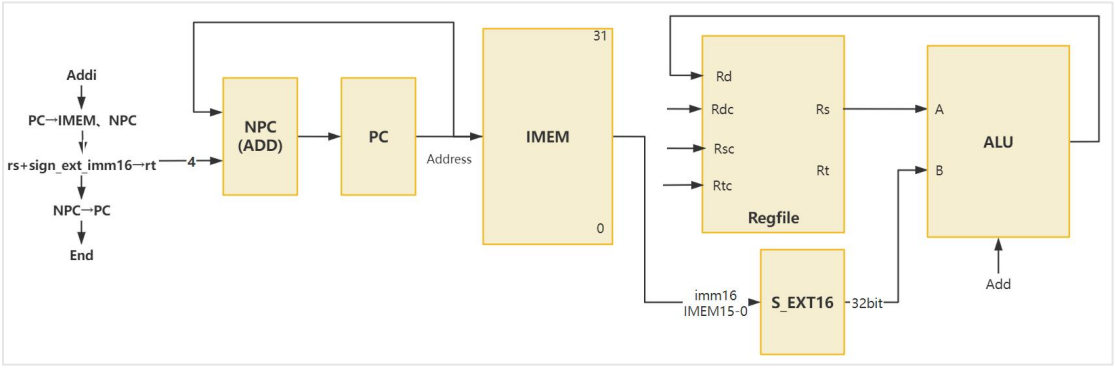
指令	PC	NPC	IMEM	Regfile	ALU		Ext5
				Rd	A	B	

JR	Rs	PC	PC				



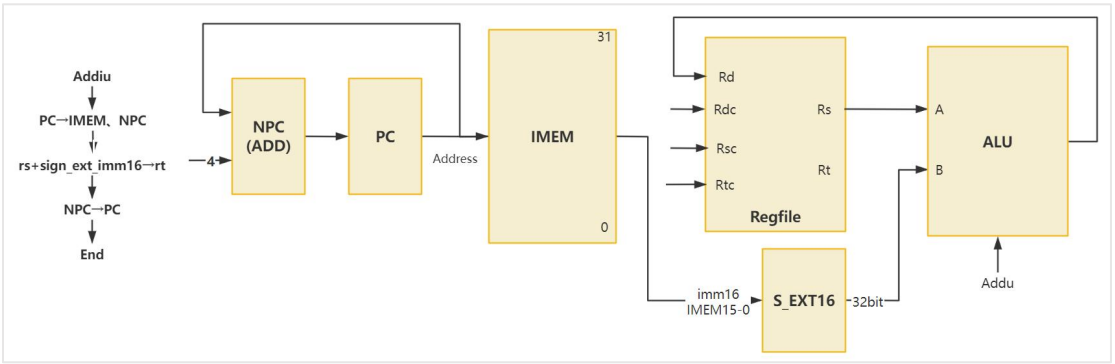
18. ADDI

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0
				Rd	A	B		
ADDI	NPC	PC	PC	ALU	Rs	S_Ext16_0		imm16



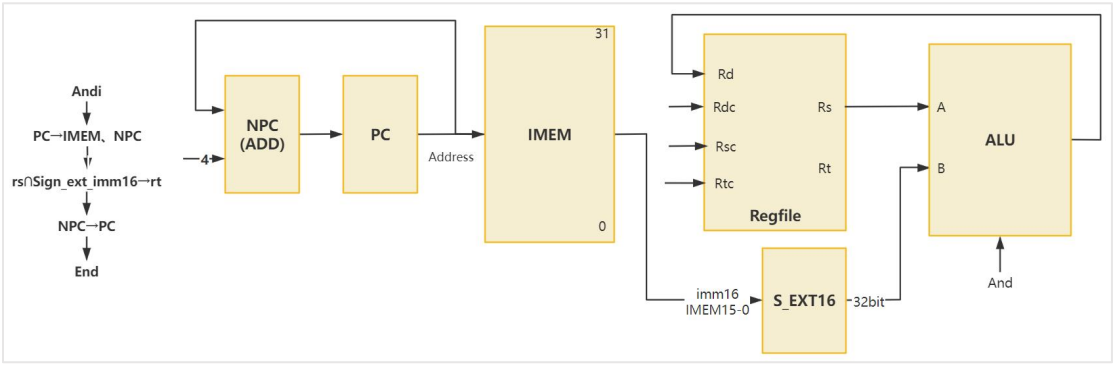
19. ADDIU

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0
				Rd	A	B		
ADDIU	NPC	PC	PC	ALU	Rs	S_Ext16_0		imm16



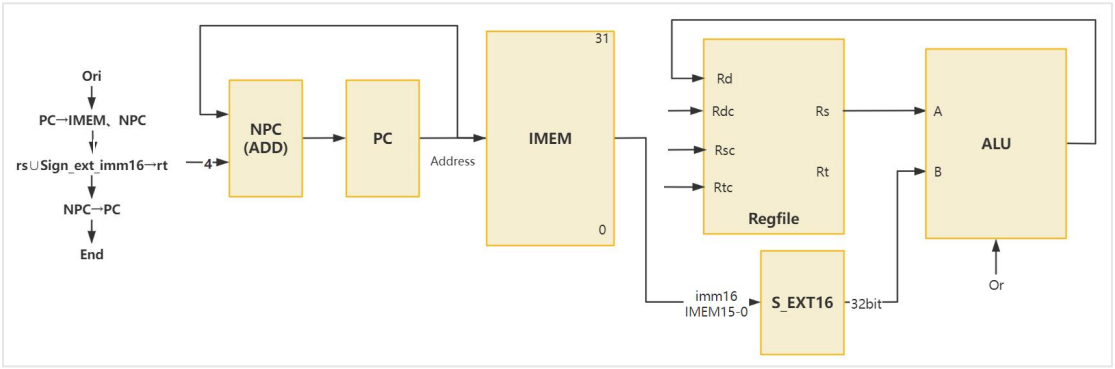
20. ANDI

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0
				Rd	A	B		
ADDI	NPC	PC	PC	ALU	Rs	Ext16		



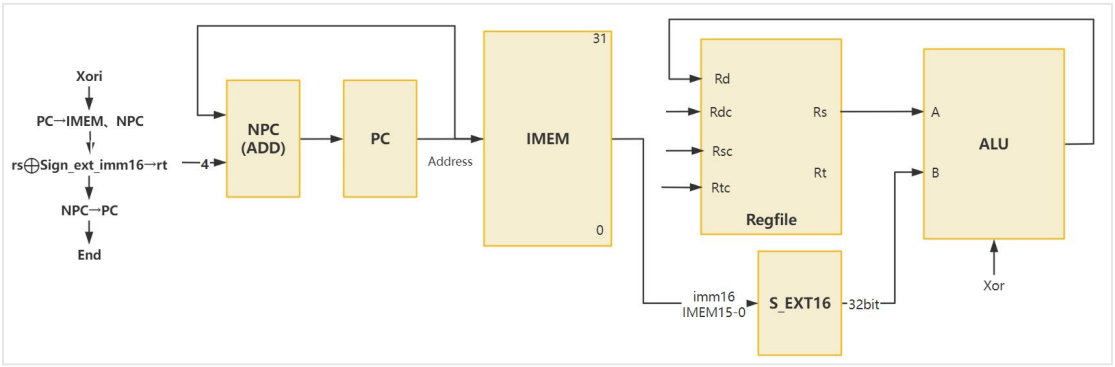
21. ORI

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0
				Rd	A	B		
ORI	NPC	PC	PC	ALU	Rs	Ext16		



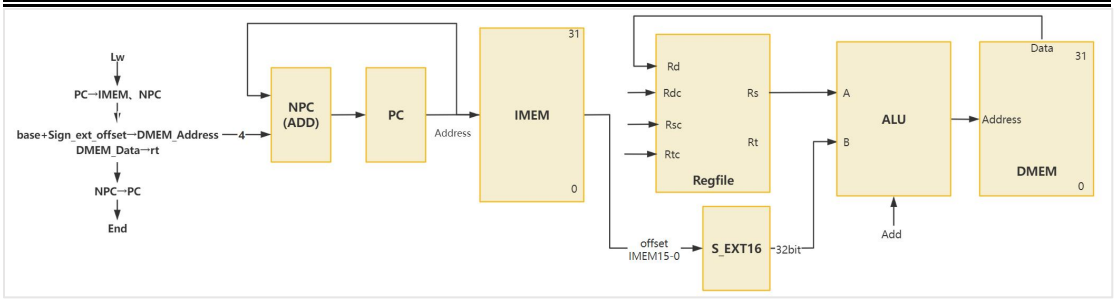
22. XORI

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0
				Rd	A	B		
XORI	NPC	PC	PC	ALU	Rs	Ext16		



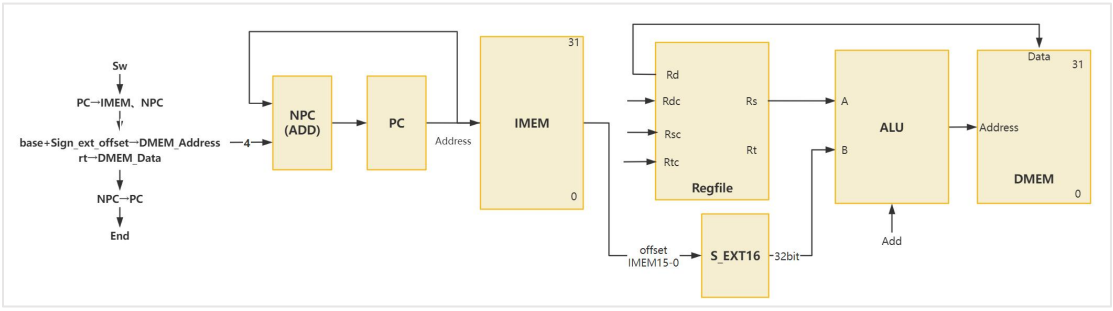
23. LW

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0	S_Ext_16_1	Ext16	DMEM	
				Rd	A	B					Addr	Data
LW	NPC	PC	PC	Data	Rs	S_Ext16_0		offset			ALU	



24. SW

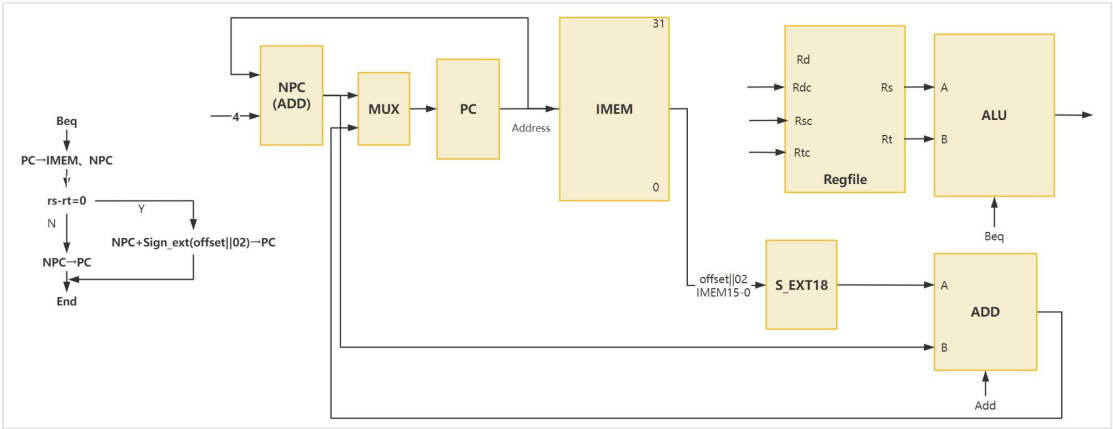
指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0	S_Ext_16_1	Ext16	DMEM	
				Rd	A	B					Addr	Data
LW	NPC	PC	PC	Data	Rs	S_Ext16_0		offset			ALU	Rt



25. BEQ(Z=1)

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B
BEQ	NPC	PC	PC		Rs	Rt

S_Ext18	S_Ext8	Ext8	ADD	
			A	B
offset			S_Ext18	NPC

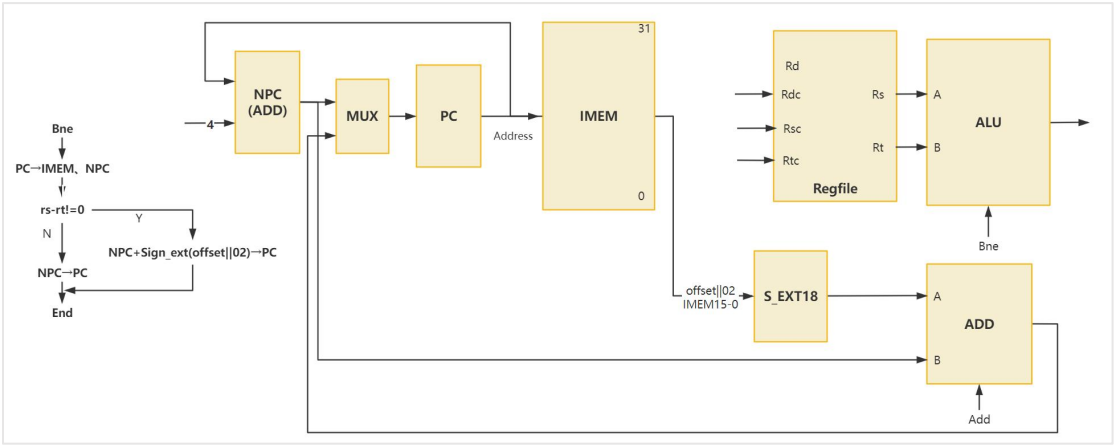


26. BNE(Z=0)

指令	PC	NPC	IMEM	Regfile	ALU	
				Rd	A	B

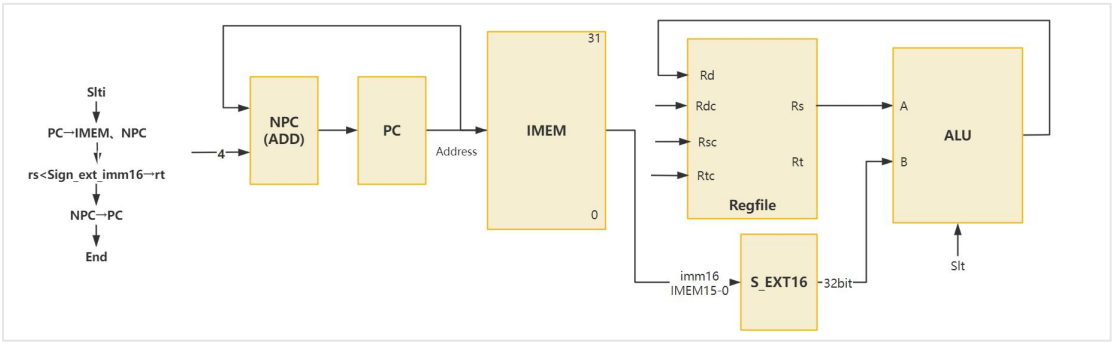
BNE	NPC	PC	PC		Rs	Rt

S_Ext18	S_Ext8	Ext8	ADD	
			A	B
offset			S_Ext18	NPC



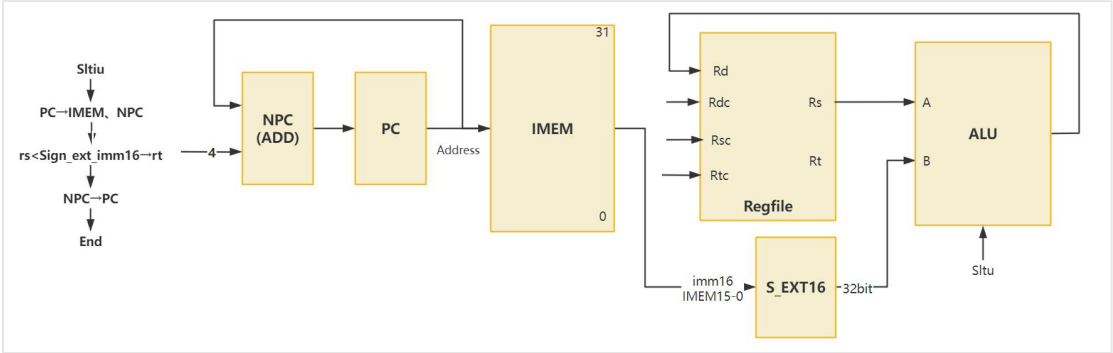
27. SLTI

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0
				Rd	A	B		
SLTI	NPC	PC	PC	ALU	Rs	S_Ext16_0		imm16



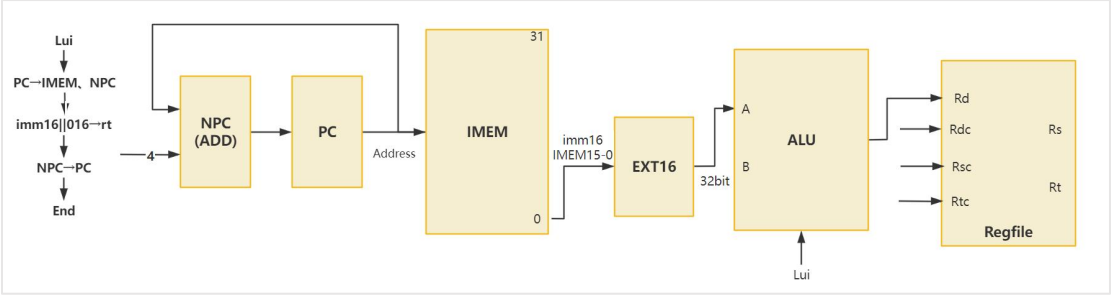
28. SLTIU

指令	PC	NPC	IMEM	Regfile	ALU		Ext5	S_Ext16_0
				Rd	A	B		
SLTI	NPC	PC	PC	ALU	Rs	S_Ext16_0		imm16



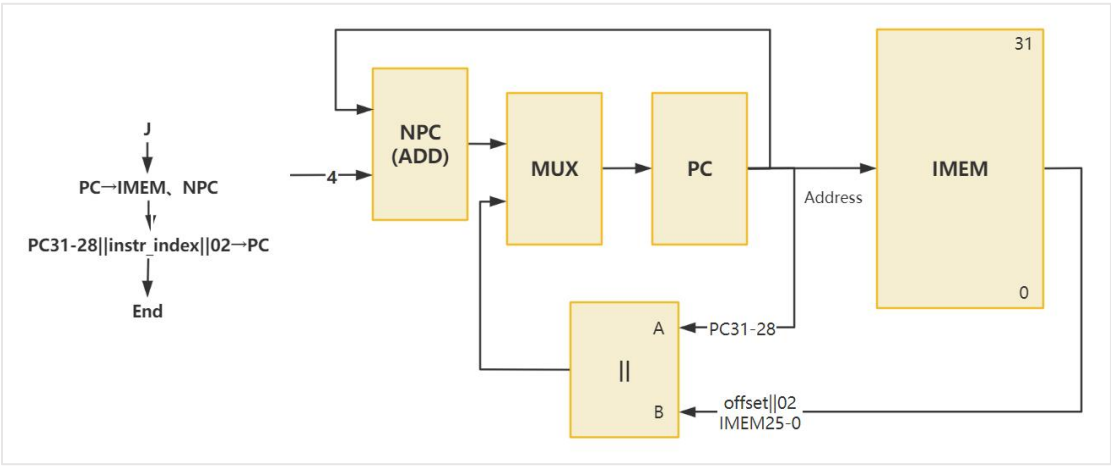
29. LUI

指令	PC	NPC	IMEM	Regfile	ALU		Ext16
				Rd	A	B	
LUI	NPC	PC	PC	ALU		Ext16	imm16



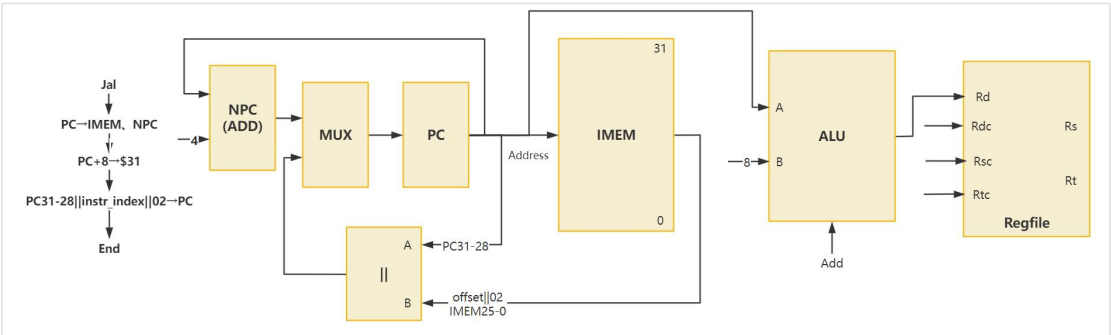
30. J

指令	PC	NPC	IMEM	Regfile	||	
				Rd	A	B
J	||	PC	PC		PC31-28	IMEM25-0



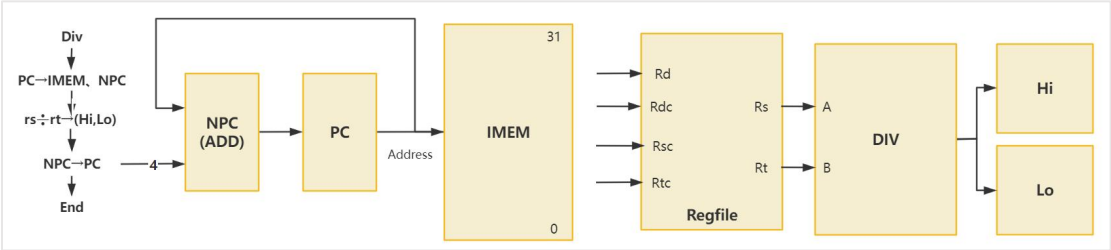
31. JAL

指令	PC	NPC	IMEM	Regfile	||	
				Rd	A	B
JAL	||	PC	PC	ALU	PC31-28	IMEM25-0



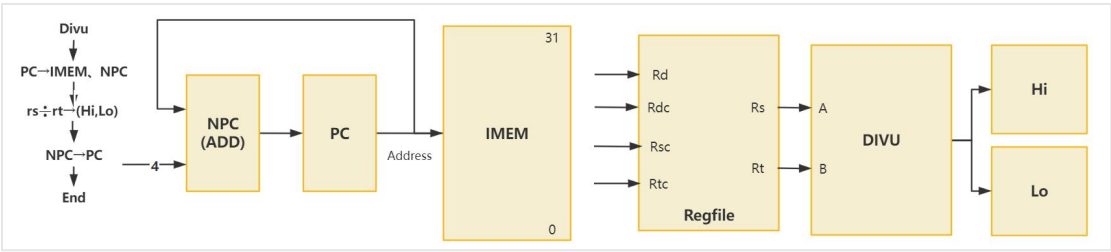
32. DIV

指令	PC	NPC	IMEM	DIV		Hi	Lo
				Dividend	Divisor		
DIV	NPC	PC	PC	Rs	Rt	r(DIV)	q(DIV)



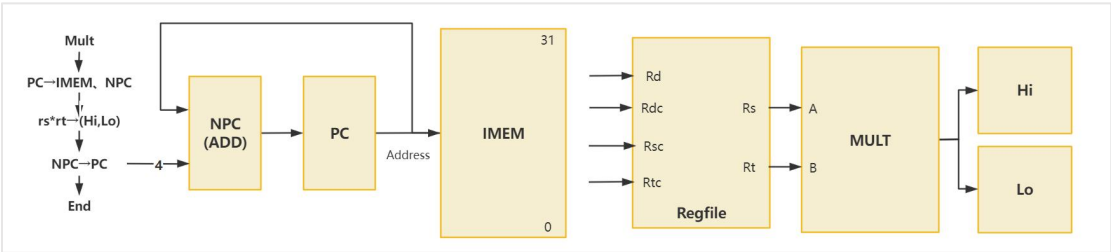
33. DIVU

指令	PC	NPC	IMEM	DIVU		Hi	Lo
				Dividend	Divisor		
DIVU	NPC	PC	PC	Rs	Rt	r(DIV)	q(DIV)



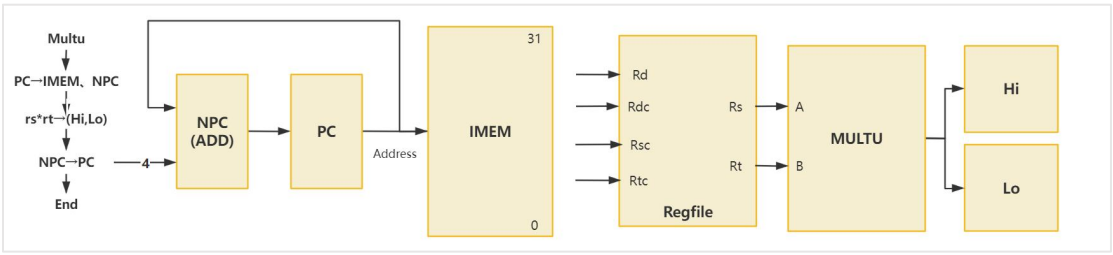
34. MULT

指令	PC	NPC	IMEM	MULT		Hi	Lo
				A	B		
MULT	NPC	PC	PC	Rs	Rt	z(MULT)63-32	z(MULT)31-0



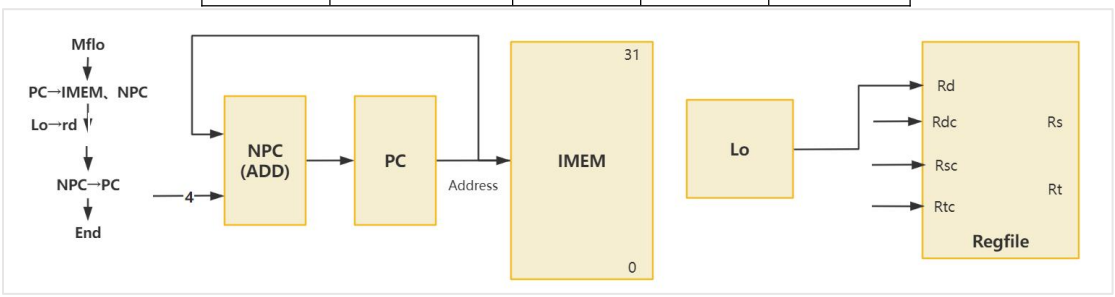
35. MULTU

指令	PC	NPC	IMEM	MULTU		Hi	Lo
				A	B		
MULT	NPC	PC	PC	Rs	Rt	z(MULT)63-32	z(MULT)31-0



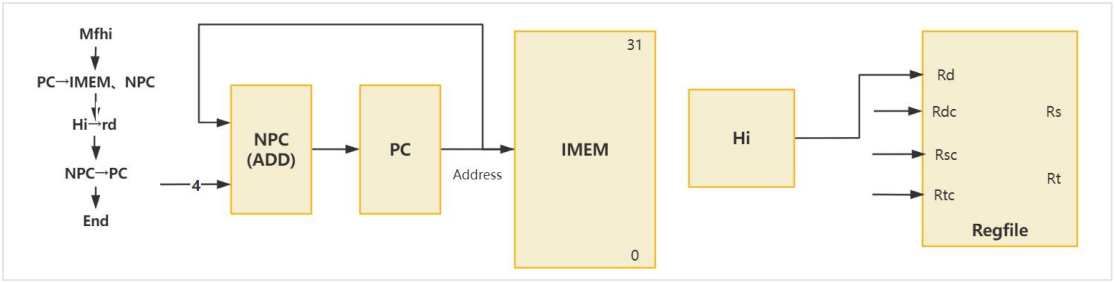
36. MFLO

指令	PC	NPC	IMEM	Regfile
				Rd
MFLO	NPC	PC	PC	Lo



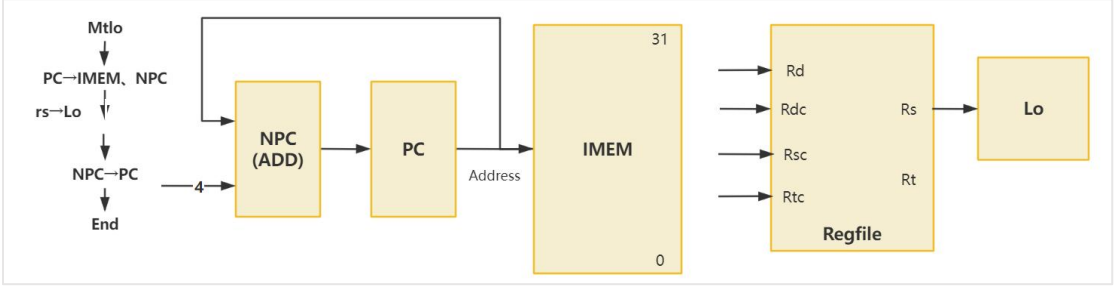
37. MFHI

指令	PC	NPC	IMEM	Regfile
				Rd
MFHI	NPC	PC	PC	Hi



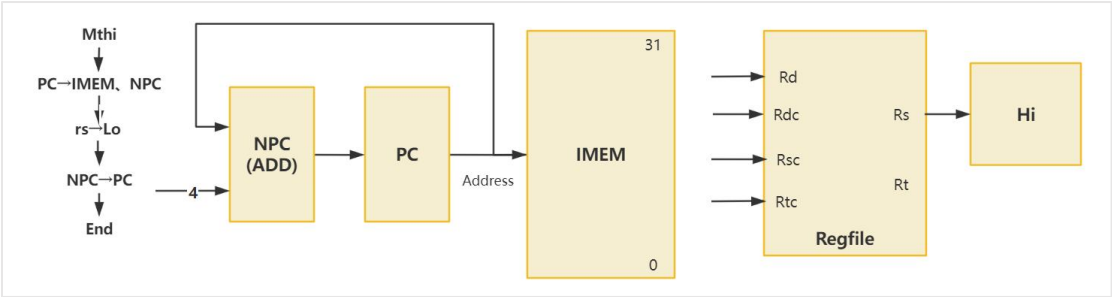
38. MTLO

指令	PC	NPC	IMEM	Lo
MTLO	NPC	PC	PC	Rs



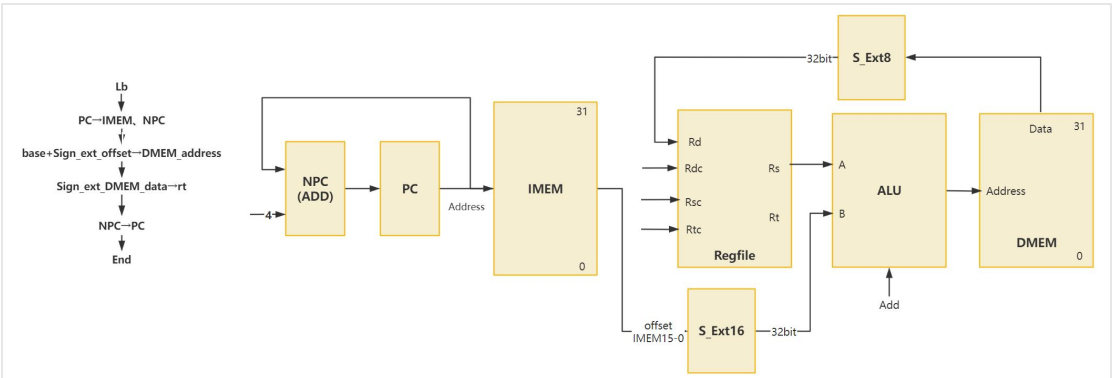
39. MTHI

指令	PC	NPC	IMEM	Hi
MTHI	NPC	PC	PC	Rs



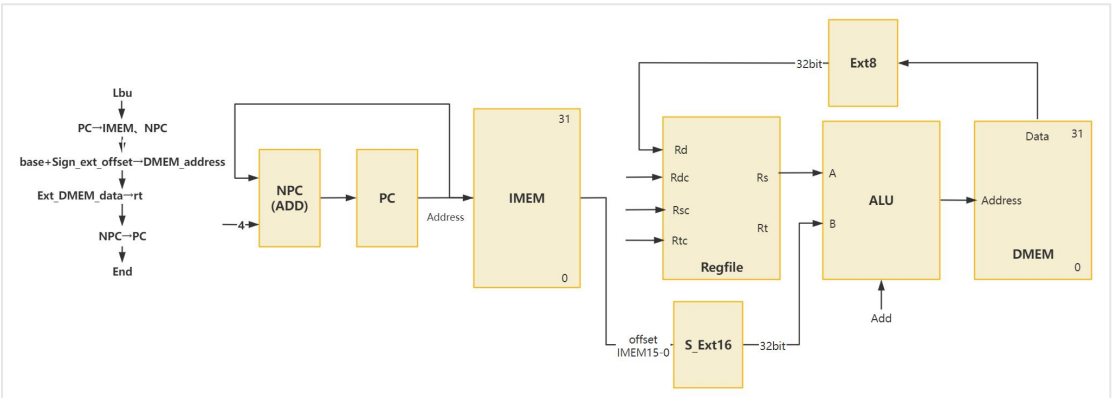
40. LB

指令	PC	NPC	IMEM	Regfile	ALU		S_Ext16_0	DMEM	S_Ext8
				Rd	A	B		Addr	
LB	NPC	PC	PC	S_Ext8	Rs	S_Ext16_0	offset	ALU	Data



41. LBU

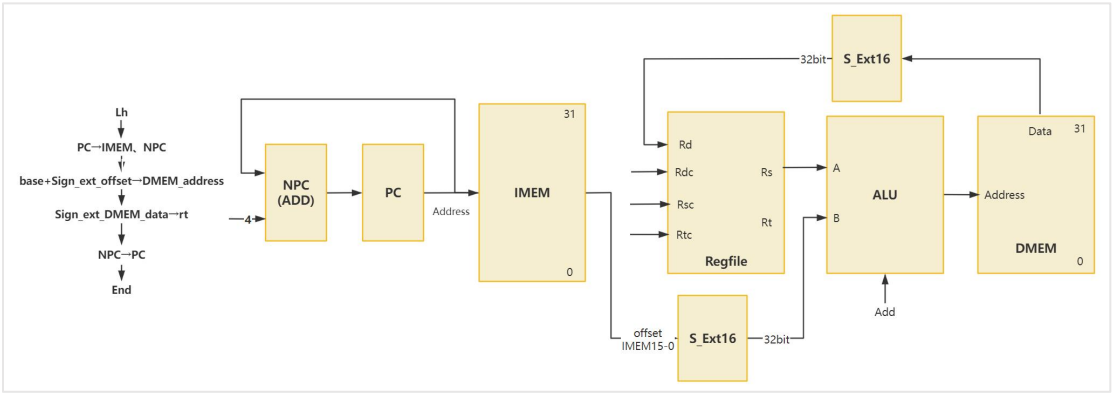
指令	PC	NPC	IMEM	Regfile	ALU		S_Ext16_0	DMEM	Ext8
				Rd	A	B		Addr	
LB	NPC	PC	PC	S_Ext8	Rs	S_Ext16_0	offset	ALU	Data



42. LH

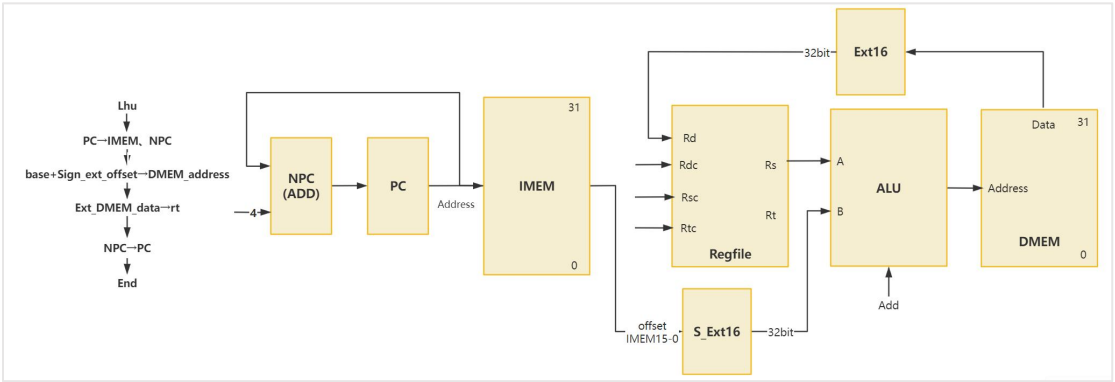
指令	PC	NPC	IME M	Regfile	ALU		S_Ext16_	DMEM	
				Rd	A	B		Addr	Data

							0		
LH	NPC	PC	PC	S_Ext16_1	Rs	S_Ext16_0	offset	ALU	



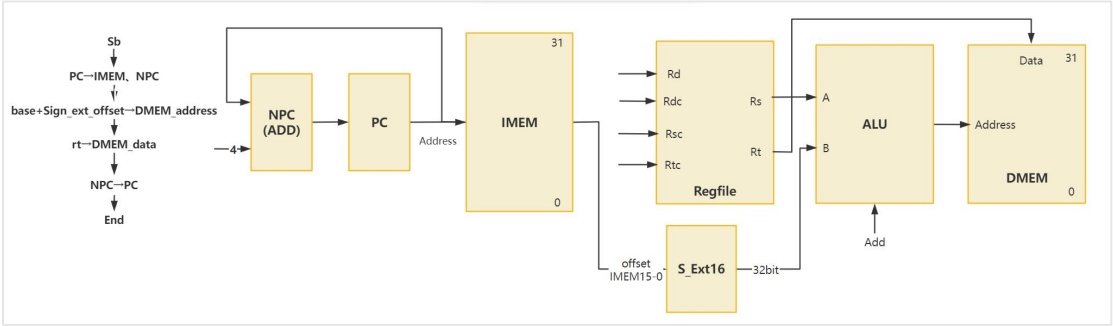
43. LHU

指令	PC	NPC	IMEM	Regfile	ALU		S_Ext16_0	DMEM	
				Rd	A	B		Addr	Data
LHU	NPC	PC	PC	S_Ext16_1	Rs	S_Ext16_0	offset	ALU	



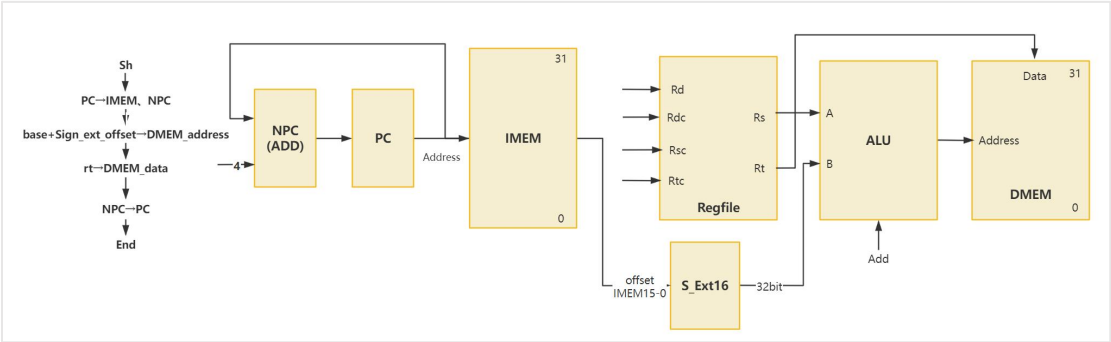
44. SB

指令	PC	NPC	IMEM	ALU		S_Ext16_0	DMEM	
				A	B		Addr	Data
SB	NPC	PC	PC	Rs	S_Ext16_0	offset	ALU	Rt7-0



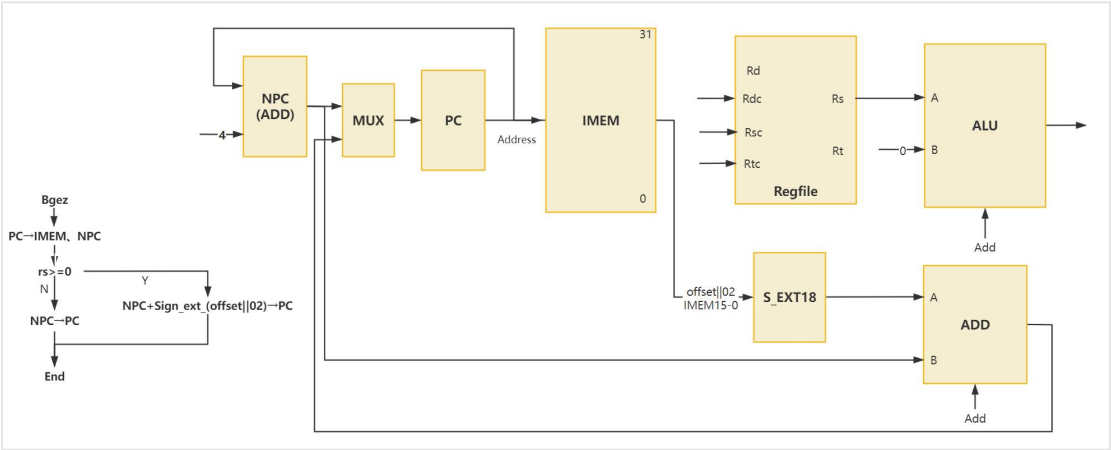
45. SH

指令	PC	NPC	IMEM	ALU		S_Ext16_0	DMEM	
				A	B		Addr	Data
SH	NPC	PC	PC	Rs	S_Ext16_0	offset	ALU	Rt15-0



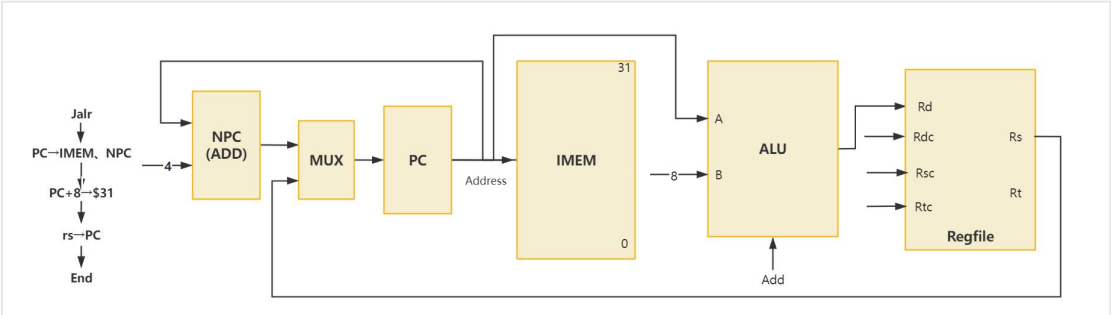
46. BGEZ(N=0)

指令	PC	NPC	IMEM	ALU		S_Ext18	ADD	
				A	B		A	B
BGEZ	ADD	PC	PC	Rs	0	offset||02	S_Ext18	NPC



47. JALR

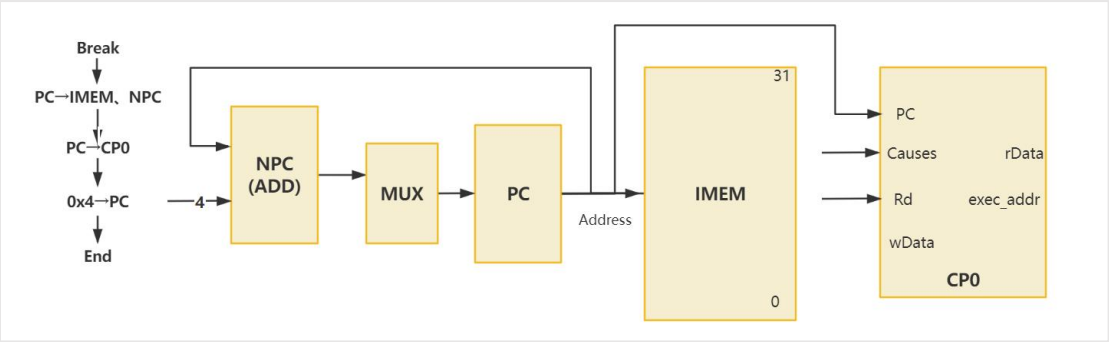
指令	PC	NPC	IMEM
JALR	Rs	PC	PC



48. BREAK

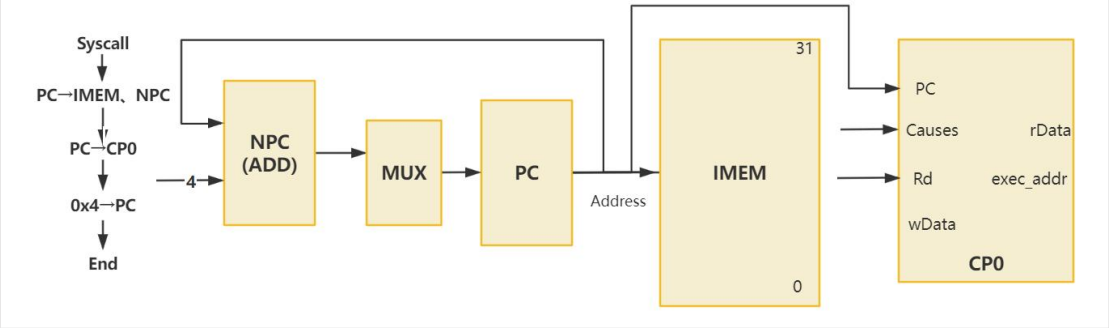
指令	PC	NPC	IMEM	CP0		
				PC	Rd	wData

BREAK	0x4	PC	PC	PC		



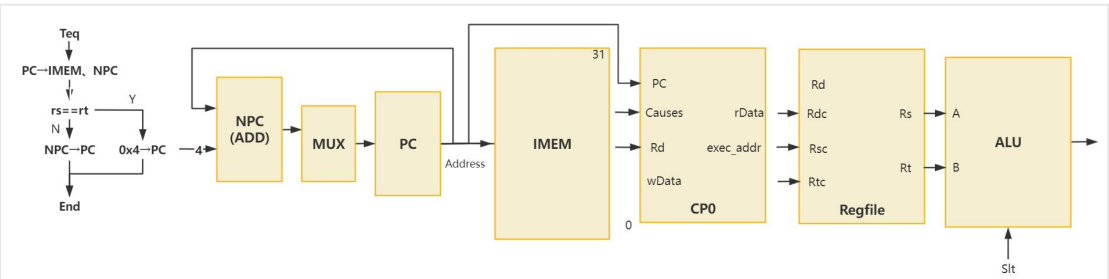
49. SYSCALL

指令	PC	NPC	IMEM	CP0		
				PC	Rd	wData
SYSCALL	0x4	PC	PC	PC		



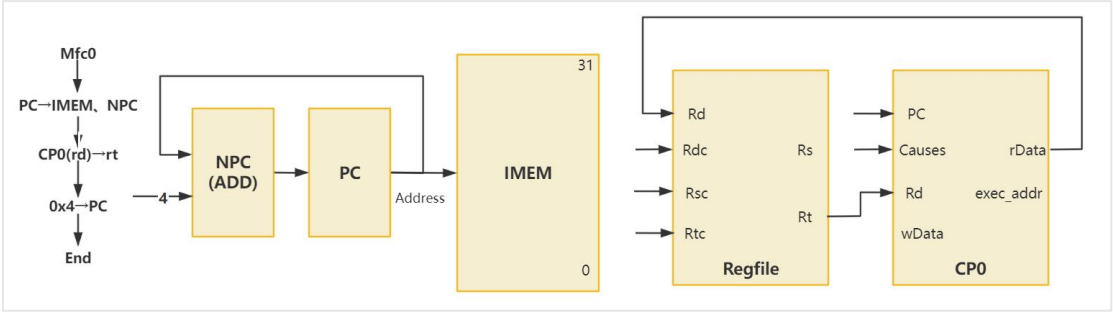
50. TEQ(Z=1)

指令	PC	NPC	IMEM	ALU		CP0		
				A	B	PC	Rd	wData
TEQ	0x4	PC	PC	Rs	Rt	PC		



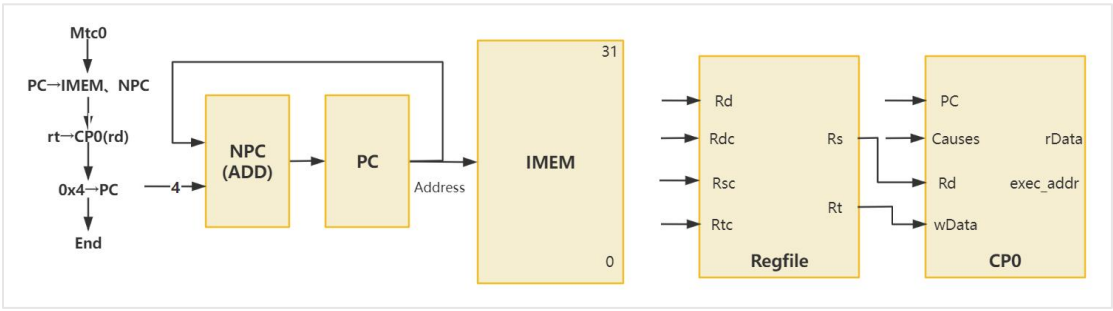
51. MFC0

指令	PC	NPC	IMEM	Regfile	CP0		
				Rd	PC	Rd	wData
MFC0	NPC	PC	PC	rData(CP0)		Rd	



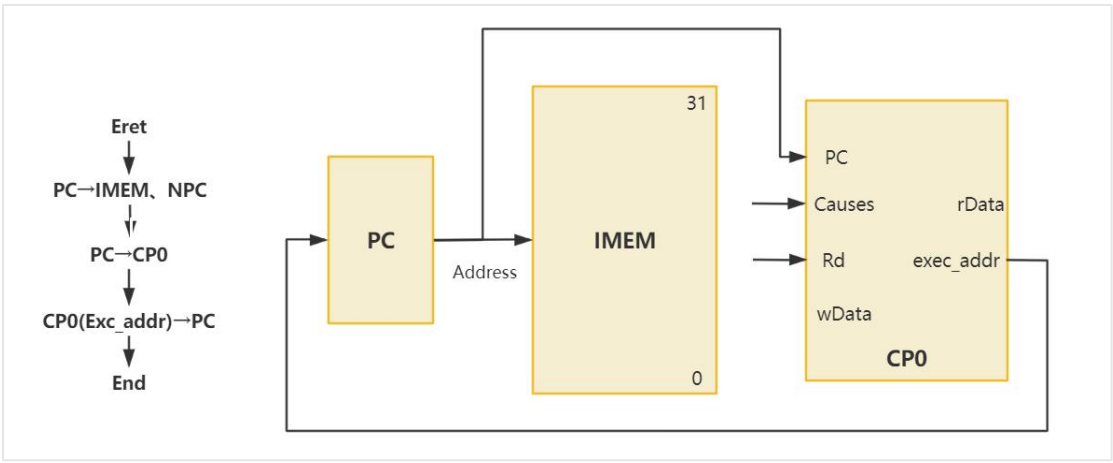
52. MTC0

指令	PC	NPC	IMEM	CP0		
				PC	Rd	wData
MTC0	NPC	PC	PC		Rd	Rt



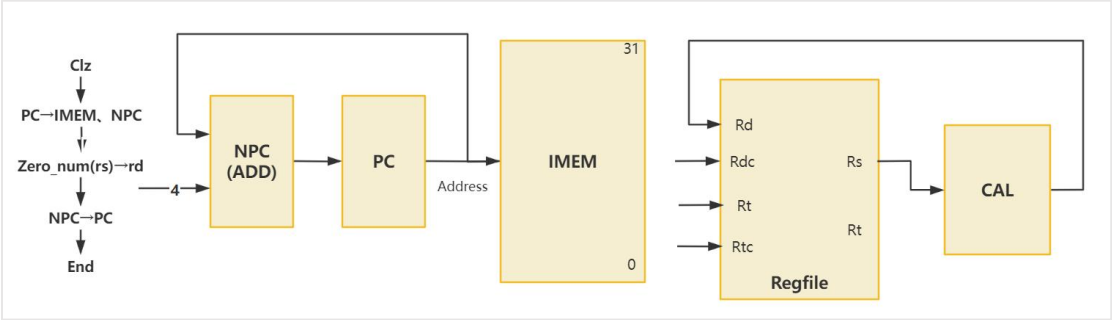
53. ERET

指令	PC	NPC	IMEM
ERET	exc_addr(CP0)	PC	PC

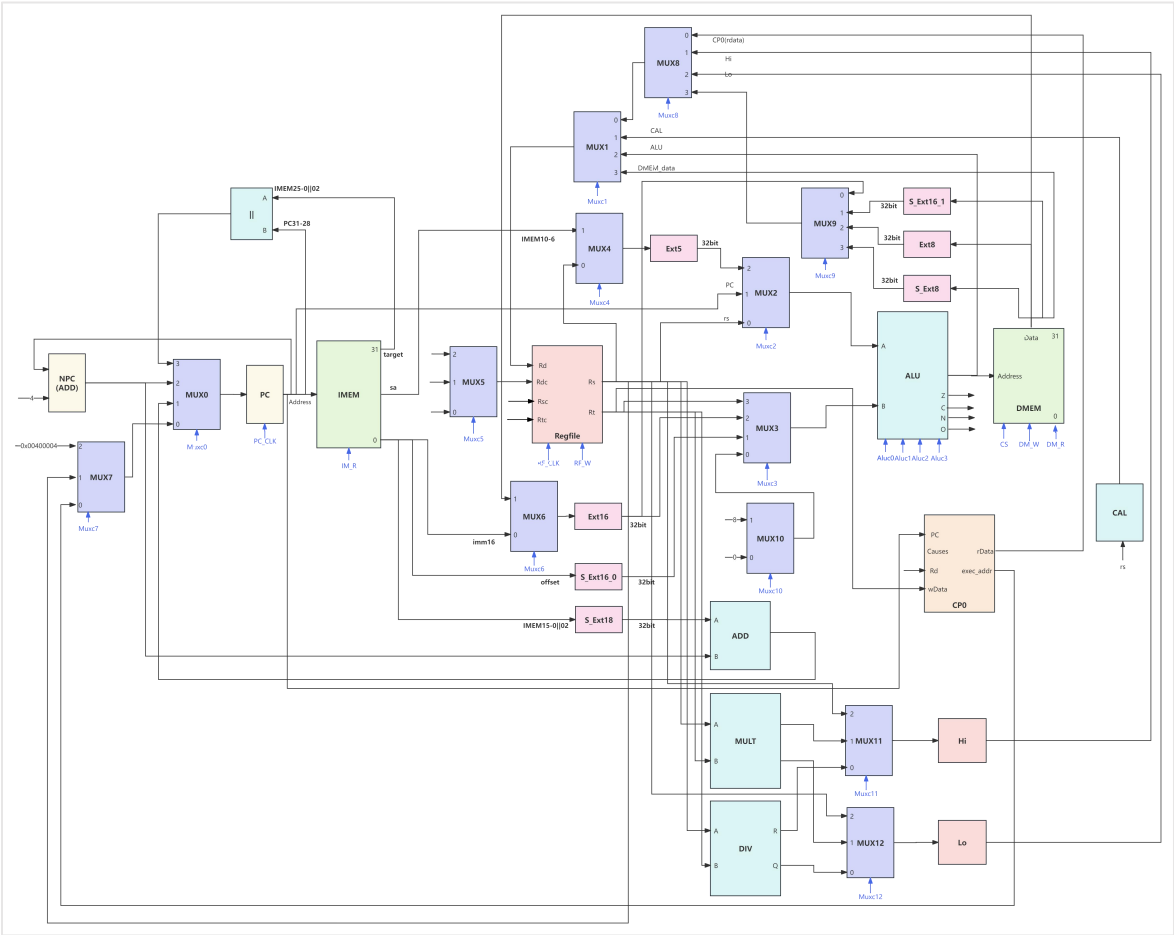


54. CLZ

指令	PC	NPC	IMEM	Regfile	CAL
				Rd	
CLZ	NPC	PC	PC	CAL	Rs



（二）完整数据通路



三、CPU 控制部件设计

（一）指令操作时间表

1. 选择器控制信号的指令操作时间表

控制器信号	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR	SLT	SLTU	SLL
mux0	2	2	2	2	2	2	2	2	2	2	2
mux1	2	2	2	2	2	2	2	2	2	2	2
mux2	0	0	0	0	0	0	0	0	0	0	2
mux3	3	3	3	3	3	3	3	3	3	3	3
mux4	0	0	0	0	0	0	0	0	0	0	1
mux5	2	2	2	2	2	2	2	2	2	2	2
mux6											
mux7											
mux8											
mux9											
mux10											
mux11											
mux12											

（续表）

控制器信号	SRL	SRA	SLLV	SRLV	SRAV	JR	ADDI	ADDIU	ANDI	ORI	XORI
mux0	2	2	2	2	2	0	2	2	2	2	2
mux1	2	2	2	2	2	2	2	2	2	2	2
mux2	2	2	2	2	2	0	0	0	0	0	0
mux3	3	3	3	3	3	3	1	1	2	2	2
mux4	1	1	0	0	0	0	0	0	0	0	0
mux5	2	2	2	2	2	2	1	1	1	1	1
mux6									0	0	0
mux7						1					
mux8											
mux9											
mux10											
mux11											
mux12											

（续表）

控制器信号	LW	SW	BEQ (Z=1)	BNE (Z=0)	SLTI	SLTIU	LUI	J	JAL	DIV	DIVU
mux0	2	2	1	1	2	2	2	3	3	2	2
mux1	3	2	2	2	2	2	2	2	2		
mux2	0	0	0	0	0	0	0	0	1		
mux3	1	1	3	3	1	1	2	3	0		

mux4	0	0	0	0	0	0	0	0	0		
mux5	1	2	2	2	1	1	1	2	0		
mux6							0				
mux7											
mux8											
mux9											
mux10									1		
mux11										0	0
mux12										0	0

（续表）

控制器信号	MULT	MULTU	MFLO	MFHI	MTLO	MTHI	LB	LBU	LH	LHU	SB
mux0	2	2	2	2	2	2	2	2	2	2	2
mux1			0	0			0	0	0	0	
mux2							0	0	0	0	0
mux3							1	1	1	1	1
mux4											
mux5			2	2			1	1	1	1	
mux6										1	
mux7											
mux8			2	1			3	3	3	3	
mux9							1	2	3	0	
mux10											
mux11	1	1				2					
mux12	1	1			2						

（续表）

控制器信号	SH	BGEZ (N=0)	JALR	BREAK	SYSCALL	TEQ (Z=1)	MFC0	MTC0	ERET	CLZ
mux0	2	1	0	0	0	0	2	2	0	2
mux1			2				0			1
mux2	0	0	1			0				
mux3	1	0	0			3				
mux4										
mux5			2				1			2
mux6										
mux7			1	2	2	2			0	
mux8							0			
mux9										
mux10		0	1							
mux11										

mux12										

2. 其余微操作信号的指令操作时间表

微操作信号	ADD	ADDU	SUB	SUBU	AND	OR	XOR
PC_CLK	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16
ALUC3	0	0	0	0	0	0	0
ALUC2	0	0	0	0	1	1	1
ALUC1	1	0	1	0	0	0	1
ALUC0	0	0	1	1	0	1	0
Rdc4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11
RF_W	1	1	1	1	1	1	1
RF_CLK	1	1	1	1	1	1	1
DM_ena	0	0	0	0	0	0	0
DM_R	0	0	0	0	0	0	0
DM_W	0	0	0	0	0	0	0
Signed_mult							
Signed_div							
exception							
Rd_CP0							
hi-ena	0	0	0	0	0	0	0
lo-ena	0	0	0	0	0	0	0

（续表）

微操作信号	NOR	SLT	SLTU	SLL	SRL	SRA	SLLV
PC_CLK	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21				IM25-21
Rtc4-0	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16
ALUC3	0	1	1	1	1	1	1
ALUC2	1	0	0	1	1	1	1
ALUC1	1	1	1	1	0	0	1
ALUC0	1	1	0	0	1	0	0
Rdc4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11
RF_W	1	1	1	1	1	1	1
RF_CLK	1	1	1	1	1	1	1
DM_ena	0	0	0	0	0	0	0
DM_R	0	0	0	0	0	0	0
DM_W	0	0	0	0	0	0	0
Signed_mult							

Signed_div							
exception							
Rd_CP0							
hi-ena	0	0	0	0	0	0	0
lo-ena	0	0	0	0	0	0	0

（续表）

微操作信号	SRLV	SRAV	JR	ADDI	ADDIU	ANDI	ORI
PC_CLK	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16					
ALUC3	1	1	0	0	0	0	0
ALUC2	1	1	0	0	0	1	1
ALUC1	0	0	0	1	0	0	0
ALUC0	1	0	0	0	0	0	1
Rdc4-0	IM15-11	IM15-11		IM20-16	IM20-16	IM20-16	IM20-16
RF_W	1	1	0	1	1	1	1
RF_CLK	1	1	0	1	1	1	1
DM_ena	0	0	0	0	0	0	0
DM_R	0	0	0	0	0	0	0
DM_W	0	0	0	0	0	0	0
Signed_mult							
Signed_div							
exception							
Rd_CP0							
hi-ena	0	0	0	0	0	0	0
lo-ena	0	0	0	0	0	0	0

（续表）

微操作信号	XORI	LW	SW	BEQ(Z=1)	BNE(Z=0)	SLTI	SLTIU
PC_CLK	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0			IM20-16	IM20-16	IM20-16		
ALUC3	0	0	0	0	0	1	1
ALUC2	1	0	0	0	0	0	0
ALUC1	1	1	1	1	1	1	1
ALUC0	0	0	0	1	1	1	0
Rdc4-0	IM20-1	IM20-1				IM20-1	IM20-1

	6	6				6	6
RF_W	1	1	0	0	0	1	1
RF_CLK	1	1	0	0	0	1	1
DM_ena	0	1	1	0	0	0	0
DM_R	0	1	0	0	0	0	0
DM_W	0	0	1	0	0	0	0
Signed_mul t							
Signed_div							
exception							
Rd_CP0							
hi-ena	0	0	0	0	0	0	0
lo-ena	0	0	0	0	0	0	0

（续表）

微操作信号	LUI	J	JAL	DIV	DIVU	MULT	MULTU
PC_CLK	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1
Rsc4-0				IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0				IM20-16	IM20-16	IM20-16	IM20-16
ALUC3	1	0	0				
ALUC2	0	0	0				
ALUC1	0	0	0				
ALUC0	0	0	0				
Rdc4-0	IM20-16		$31				
RF_W	1	0	1	0	0	0	0
RF_CLK	1	0	1	0	0	0	0
DM_ena	0	0	0	0	0	0	0
DM_R	0	0	0	0	0	0	0
DM_W	0	0	0	0	0	0	0
Signed_mult						1	0
Signed_div				1	0		
exception							
Rd_CP0							
hi-ena	0	0	0	1	1	1	1
lo-ena	0	0	0	1	1	1	1

（续表）

微操作信号	MFLO	MFHI	MTLO	MTHI	LB	LBU	LH
PC_CLK	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1
Rsc4-0			IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0							

ALUC3					0	0	0
ALUC2					0	0	0
ALUC1					1	1	1
ALUC0					0	0	0
Rdc4-0	IM15-11	IM15-11			IM20-16	IM20-16	IM20-16
RF_W	1	1	0	0	1	1	1
RF_CLK	1	1	0	0	1	1	1
DM_ena	0	0	0	0	1	1	1
DM_R	0	0	0	0	1	1	1
DM_W	0	0	0	0	0	0	0
Signed_mult							
Signed_div							
exception							
Rd_CP0							
hi-ena	0	0	0	1	0	0	0
lo-ena	0	0	1	0	0	0	0

（续表）

微操作信号	LHU	SB	SH	BGEZ(N=0)	JALR	BREAK	SYSCALL
PC_CLK	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21		
Rtc4-0		IM20-16	IM20-16				
ALUC3	0	0	0	0	0		
ALUC2	0	0	0	0	0		
ALUC1	1	1	1	1	1		
ALUC0	0	0	0	1	0		
Rdc4-0	IM20-16				IM15-11		
RF_W	1	0	0	0	1	0	0
RF_CLK	1	0	0	0	1	0	0
DM_ena	1	1	1	0	0	0	0
DM_R	1	0	0	0	0	0	0
DM_W	0	1	1	0	0	0	0
Signed_mult							
Signed_div							
exception						1	1
Rd_CP0							
hi-ena	0	0	0	0	0	0	0

lo-ena	0	0	0	0	0	0	0

（续表）

微操作信号	TEQ(Z=1)	MFC0	MTC0	ERET	CLZ		
PC_CLK	1	1	1	1	1		
IM_R	1	1	1	1	1		
Rsc4-0	IM25-21				IM25-21		
Rtc4-0	IM20-16		IM20-16				
ALUC3	0						
ALUC2	0						
ALUC1	1						
ALUC0	1						
Rdc4-0		IM20-16			IM15-11		
RF_W	0	1	0	0	1		
RF_CLK	0	1	0	0	1		
DM_ena	0	0	0	0	0		
DM_R	0	0	0	0	0		
DM_W	0	0	0	0	0		
Signed_mult							
Signed_div							
exception	Z=1						
Rd_CP0		IM15-11	IM15-11				
hi-ena	0	0	0	0	0		
lo-ena	0	0	0	0	0		

3. ALU 操作信号说明

指令	ALUC3	ALUC2	ALUC1	ALUC0	备注
Addu	0	0	0	0	ALU 完成“无符号加”
Subu	0	0	0	1	ALU 完成“无符号减”
Add	0	0	1	0	ALU 完成“有符号加”
Sub	0	0	1	1	ALU 完成“有符号减”
And	0	1	0	0	ALU 完成“按位与”
Or	0	1	0	1	ALU 完成“按位或”
Xor	0	1	1	0	ALU 完成“按位异或”
Nor	0	1	1	1	ALU 完成“按位或非”
Lui	1	0	0	x	ALU 完成“置高位立即数”
Sltu	1	0	1	0	ALU 完成“无符号小于比较”
Slt	1	0	1	1	ALU 完成“小于比较”
Sra	1	1	0	0	ALU 完成“算术右移”
Srl	1	1	0	1	ALU 完成“逻辑右移”
Sll	1	1	1	x	ALU 完成“逻辑左移与算术左移”

4. 其余操作信号说明

信号	备注
PC_CLK	CPU 工作主频
IM_R	代码存储器读信号
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
M3_1	MUX3 选择器控制端 1
M3_0	MUX3 选择器控制端 0
M4_1	MUX4 选择器控制端 1
M4_0	MUX4 选择器控制端 0
ALUC3	ALU 控制端 3
ALUC2	ALU 控制端 2
ALUC1	ALU 控制端 1
ALUC0	ALU 控制端 0
M2	MUX2 选择器控制端 1
Rdc4-0	Rd 寄存器选择输入控制端
RF_W	Regfile 写信号
RF_CLK	Regfile 时钟
M1_1	MUX1 选择器控制端
M1_0	MUX1 选择器控制端
DM_R	数据存储器读信号
DM_W	数据存储器写信号
M5	MUX5 选择器控制端

（二）控制信号逻辑表达式

Pc_clk=1

Imem_R=1

Rsc4-0=ADD+ADDU+SUB+SUBU+AND+OR+XOR+NOR+SLT+SLTU+SLLV+SRLV+SRAV+JR+ADDI+ADDIU+ANDI+ORI+XORI+LW+SW+BEQ*Z+BNE*~Z+SLTI+SLTIU+DIV+DIVU+MULT+MULTU+MLTO+MTHI+LB+LBU+LH+TEQ*Z+CLZ

Rtc4-0=ADD+ADDU+SUB+SUBU+AND+OR+XOR+NOR+SLT+SLTU+SLL+SRL+SRA+SLLV+SRLV+SRAV+SW+BEQ*Z+BNE*~Z+DIV+DIVU+MULT+MULTU+TEQ*Z+MTC0

Rdc4-0=ADD+ADDU+SUB+SUBU+AND+OR+XOR+NOR+SLT+SLTU+SLL+SR

L+SRA+SLLV+SRLV+SRAV+ADDI+ADDIU+ANDI+ORI+XORI+LW+SLTI+SLTIU+LUI+JAL+MFLO+MFHI+LB+LBU+LH+MFC0+CLZ

Aluc3=SLT+SLTU+SLL+SRL+SRA+SLLV+SRLV+SRAV+SLTI+SLTIU+LUI

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+LW+SW+BEQ*Z+BNE*~Z+SLTI+SLTIU+LB+LBU+LH+TEQ*Z

Aluc0=SUB+SUBU+OR+NOR+SLT+SRL+SRLV+ORI+BEQ*Z+BNE*~Z+SLTI+TEQ*Z

RF_W=~(JR+SW+BEQ*Z+BNE*~Z+J+DIV+DIVU+MULT+MULTU+MTLO+MTHI+TEQ*Z+MTC0+ERET)

RF_CLK=RF_W

DM_R=LW+LB+LBU+LH

DM_W=SW

DM_ena=DM_W+DM_R

Signed_mult=MULT

Signed_div=DIV

Rd_CP0=MFC0+MTC0

Hi_ena=DIV+DIVU+MULT+MULTU+MTHI

Lo_ena=DIV+DIVU+MULT+MULTU+MTLO

（三）控制部件设计

1. 设计思路

本次实验的 CPU 主要分为运算器和控制器两部分。通过代码编写或直接实例化模块，可以完成下述模块的实现，包括：算术逻辑运算单元 ALU、指令译码器 decoder、程序计数器 PC、寄存器堆 RegFile、中断寄存器 CP0、高位地位寄存器 HiLo、前导零计数器 CAL、乘法器 Multiplier、除法器 Divider 以及控制器 controller。对于拓展模块 Ext 和选择器 Mux，直接通过 Verilog 语言中的级联语法、三目运算符等实现，没有使用具体的 module。对于内存模块，分为指令存储器 IMEM 和地址数据存储器 DMEM 两部分，分别用不同的 module

实现。

（1）ALU：直接实例化上学期数字逻辑中的代码。

（2）Decoder：实例化模块，通过输入的指令内容判断解码生成当前组要完成的操作。

（3）PC：实例化 reg 变量，存储当前正在执行的指令地址，并给出下一条将要执行的指令所在地址。

（4）Regfile：直接在模块内声明合适大小的 reg 变量。

（5）MEM：存储器部分，分为 DMEM 和 IMEM 两部分。DMEM 通过声明合适大小的 reg 变量实现，IMEM 通过实例化 IP 核实现。

（6）CP0：中断寄存器部分，存储中断发生时必要的 PC 地址以及数据等，通过实例化 reg 变量实现。

（7）Multipier、Divider：乘法器和除法器部分，分为有符号和无符号运算两种。

（8）CAL：计算输入数据的前导零模块。

（9）HiLo：高位和低位寄存器模块，直接实例化 reg 变量实现。

（10）Cpu：对上述的除存 MEM 以外的其他模块进行实例化，协调这些模块元件之间的数据流向并将他们通过内部变量链接。

（11）Dataflow：顶层的数据通路，完成 cpu 各部件与存储器 mem 之间的数据通路联系，由于本次 CPU 设计中不涉及外围设备 I/O，所以我们只需要协调这两部分之间的数据传输即可。

2. 代码

（1）接口定义

```
//cpu 模块
module cpu(
    input clk,
    input ena,
    input rst_n,
    input [31:0] in_instr,                          //指令

    input [31:0] dmem_out_data,                //读取到的 dmem 中的数据
    output [31:0] dmem_in_data,               //写入 dmem 中的数据
    output dmem_ena,                             //启用 dmem 信号
    output dmem_W,                               //dmem 写信号
    output dmem_R,                               //dmem 读信号
    output [5:0] dmem_opr,                                //dmem 写入方式
    output [31:0] out_pc,                              //输出指令地址
    output [31:0] dmem_addr                    //dmem 启用地址
    );
```

（2）Verilog 代码

```
//2252429 蔡宇轩
`timescale 1ns / 1ps
//cpu 模块
module cpu(
    input clk,
    input ena,
    input rst_n,
    input [31:0] in_instr,                          //指令

    input [31:0] dmem_out_data,                 //读取到的 dmem 中的数据
    output [31:0] dmem_in_data,                //写入 dmem 中的数据
    output dmem_ena,                              //启用 dmem 信号
    output dmem_W,                                //dmem 写信号
    output dmem_R,                                //dmem 读信号
    output [5:0] dmem_opr,                              //dmem 写入方式
    output [31:0] out_pc,                          //输出指令地址
    output [31:0] dmem_addr                     //dmem 启用地址
    );
//-------------------------------------------------------------
//内部连线变量
    /*decoder 用*/
    wire is_add, is_addu, is_sub, is_subu, is_and, is_or, is_xor, is_nor, is_slt,
is_sltu,
    is_sra, is_srl, is_sll, is_srav, is_srlv, is_sllv, is_jr, is_addi, is_addiu,
is_andi,
    is_ori, is_xori, is_lw, is_sw, is_beq, is_bne, is_slti, is_sltiu, is_lui,
is_j,
    is_jal, is_div, is_divu, is_mult, is_multu, is_mflo, is_mfhi, is_mtlo, is_mthi,
is_lb,
    is_lbu, is_lh, is_lhu, is_sb, is_sh, is_bgez, is_jalr, is_break, is_syscall,
is_teq,
    is_mfc0, is_mtc0, is_eret, is_clz;              //各指令标志信息

    wire [4:0] Rsc,Rtc,Rdc;                        //寄存器对应地址
    wire [4:0] shamt;                               //偏移量
    wire [15:0] immediate;                         //立即数
    wire [25:0] address;                            //跳转地址（J 型指令用）

    /*ALU 用*/
    wire [31:0] alu_A,alu_B;                                    //运算数
    wire [3:0] aluc;                                  //ALU 功能选择信号
    wire [31:0] alu_result;                         //运算结果
    wire Z,C,N,O;                                     //运算标志位

    /*DIV 用*/
    wire signed_div;                                //有无符号功能选择信号
    wire [31:0] div_A,div_B,R,Q;                            //运算数及结果

    /*MULT 用*/
    wire signed_mult;                               //有无符号功能选择信号
    wire [31:0] mult_A,mult_B,Hi,Lo;                        //运算数及结果

    /*HiLo 用*/
    wire Hi_W,Lo_W;                                 //HILO 寄存器写使能信号
    wire [31:0] Hi_in_data, Hi_out_data;       //HI data
```

```
        wire [31:0] Lo_in_data, Lo_out_data;       //LO data

        /*regfile 用*/
        //寄存器对应地址 rsc,rtc,rdc 已在 decoder 部分定义
        //使能写信号 reg_W 在 controller 部分定义
        //此处定义数据部分
        wire [31:0] rs_out_data, rt_out_data;
        wire [31:0] rd_in_data;

        /*PC 用*/
        wire [31:0] pc_in_next_addr;
        wire [31:0] pc_out_now_addr;

        /*controller 用*/
        //所有指令标志已在 decoder 部分定义
        //Z, N 在 ALU 部分定义
        wire [1:0] muxc [12:0];           //多路选择器信号
        //aluc 信号已在 ALU 部分定义
        //signed div,signed mult 已在 MULT DIV 部分定义
        wire ext5, ext8, s_ext8,  ext16, s_ext16_0, s_ext16_1, s_ext18;    //拓展部件
使能信号
        //dmem R and W 已在本模块定义
        wire reg_W;
        //Hi Lo 使能写信号已在 HILO 部分定义
        wire cat;                     //数位拼接使能信号
        wire exception;         //CP0 使能信号
        /*CP0 用*/
        //mfc0 = is_mfc0,  mtc0 = is_mtc0,  eret = is_eret
        //pc=pc_out_now_addr
        wire [4:0] Rd;              //CP0 中寄存器编号
        //exception 在 controller 部分定义
        wire [31:0] rdata, wdata;
        wire [31:0] exec_addr;
        reg [4:0] cause;
        /*CAL 用*/
        wire [31:0] cal_in_data;
        wire [5:0] cal_out_result;

    //内部工作变量
        //各拓展模块的实际输出
        wire [31:0] ext5_out,  ext8_out,  ext16_out;
        wire signed [31:0] s_ext8_out,  s_ext16_0_out,  s_ext16_1_out,  s_ext18_out;
        //拼接器的实际输出
        wire [31:0] cat_out;
        //npc
        wire [31:0] npc;
        //多路选择器的实际输出
        reg [31:0] mux_out [12:0];

    //-----------------------------------------------------------------
    //连接各模块
        /*EXT*/
            assign ext5_out = (ext5) ? mux_out[4] : 32'hz;
            assign ext8_out = (ext8) ? dmem_out_data : 32'hz;
            assign s_ext8_out = (s_ext8) ? dmem_out_data : 32'hz;
            assign ext16_out = (ext16) ? mux_out[6] : 32'hz;
```

```
            assign s_ext16_0_out = (s_ext16_0) ? { {16{immediate[15]}} ,
immediate[15:0]} : 32'hz;
            assign s_ext16_1_out = (s_ext16_1) ? dmem_out_data : 32'hz;
            assign s_ext18_out = (s_ext18) ?
{{14{immediate[15]}},immediate[15:0],2'h0} : 32'hz;
        /*NPC*/
            assign npc=pc_out_now_addr +4;
        /*MUX*/
        always @*begin
            //mux7
            if(muxc[7]==0) mux_out[7]=exec_addr;
            else if(muxc[7]==1) mux_out[7]=rs_out_data;
            else if(muxc[7]==2) mux_out[7]=32'h0040_0004;
            //mux0
            if(muxc[0]==0) mux_out[0]=mux_out[7];
            else if(muxc[0]==1) mux_out[0]=npc+s_ext18_out;
            else if(muxc[0]==2) mux_out[0]=npc;
            else if(muxc[0]==3) mux_out[0]=cat_out;

           //mux9
            if(muxc[9]==0) mux_out[9]=ext16_out;
            else if(muxc[9]==1) mux_out[9]=s_ext16_1_out;
            else if(muxc[9]==2) mux_out[9]=ext8_out;
            else if(muxc[9]==3) mux_out[9]=s_ext8_out;

            //mux8
            if(muxc[8]==0) mux_out[8]=rdata;
            else if(muxc[8]==1) mux_out[8]=Hi_out_data;
            else if(muxc[8]==2) mux_out[8]=Lo_out_data;
            else if(muxc[8]==3) mux_out[8]=mux_out[9];

            //mu1
            if(muxc[1]==0) mux_out[1]=mux_out[8];
            else if(muxc[1]==1) mux_out[1]=cal_out_result;
            else if(muxc[1]==2) mux_out[1]=alu_result;
            else if(muxc[1]==3) mux_out[1]=dmem_out_data;

            //mux2
            if(muxc[2]==0) mux_out[2]=rs_out_data;
            else if(muxc[2]==1) mux_out[2]=pc_out_now_addr;
            else if(muxc[2]==2) mux_out[2]=ext5_out;

            //mux10
            if(muxc[10]==1) mux_out[10]=4;
            else if(muxc[10]==0) mux_out[10]=0;

            //mux3
            if(muxc[3]==0) mux_out[3]=mux_out[10];
            else if(muxc[3]==1) mux_out[3]=s_ext16_0_out;
            else if(muxc[3]==2) mux_out[3]=ext16_out;
            else if(muxc[3]==3) mux_out[3]=rt_out_data;

            //mux4
            if(muxc[4]==0) mux_out[4]=rs_out_data;
            else if(muxc[4]==1) mux_out[4]=shamt;

            //mux5：已在 decoder 部分完成 rdc 选择
```

```
    //mux6
    if(muxc[6]==1) mux_out[6]=dmem_out_data;
    else if(muxc[6]==0) mux_out[6]=immediate;

    //mux11
    if(muxc[11]==0) mux_out[11]=R;
    else if(muxc[11]==1) mux_out[11]=Hi;
    else if(muxc[11]==2) mux_out[11]=rs_out_data;

    //mux12
    if(muxc[12]==0) mux_out[12]=Q;
    else if(muxc[12]==1) mux_out[12]=Lo;
    else if(muxc[12]==2) mux_out[12]=rs_out_data;
end//end of always

/*PC*/
assign pc_in_next_addr=mux_out[0];
assign out_pc =pc_out_now_addr;
/*拼接*/
assign cat_out =cat?  {out_pc[31:28], address[25:0],2'h0} : 32'hz;

/*ALU*/
assign alu_A = mux_out[2];
assign alu_B = mux_out[3];

/**DIV*/
assign div_A = rs_out_data;
assign div_B = rt_out_data;
/*MULT*/
assign mult_A = rs_out_data;
assign mult_B = rt_out_data;
/*regfile*/
assign rd_in_data = mux_out[1];

/*DMEM*/
assign dmem_ena = (dmem_W || dmem_R) ? 1'b1 : 1'b0;
assign dmem_addr = alu_result;
assign dmem_in_data = rt_out_data;
assign dmem_opr = in_instr[31:26];

/*IMEM*/
assign out_pc = pc_out_now_addr;

/*Hi Lo*/
assign Hi_in_data = mux_out[11];
assign Lo_in_data = mux_out[12];

/*CP0*/
assign wdata = rt_out_data;
always@*begin
    if(is_break) cause=5'b01001;
    else if(is_syscall) cause=5'b01000;
    else if(is_teq) cause=5'b01101;
    else cause=5'bz;
end//end of always
assign Rd = in_instr[15:11];
```

```
    /*CAL*/
    assign cal_in_data = rs_out_data;
//------------------------------------------------------------------
//实例化各子模块
    /*instance of decoder*/
    Decoder decoder_inst(
        .in_instr(in_instr),
        .is_add(is_add),
        .is_addu(is_addu),
        .is_sub(is_sub),
        .is_subu(is_subu),
        .is_and(is_and),
        .is_or(is_or),
        .is_xor(is_xor),
        .is_nor(is_nor),
        .is_slt(is_slt),
        .is_sltu(is_sltu),
        .is_sra(is_sra),
        .is_srl(is_srl),
        .is_sll(is_sll),
        .is_srav(is_srav),
        .is_srlv(is_srlv),
        .is_sllv(is_sllv),
        .is_jr(is_jr),
        .is_addi(is_addi),
        .is_addiu(is_addiu),
        .is_andi(is_andi),
        .is_ori(is_ori),
        .is_xori(is_xori),
        .is_lw(is_lw),
        .is_sw(is_sw),
        .is_beq(is_beq),
        .is_bne(is_bne),
        .is_slti(is_slti),
        .is_sltiu(is_sltiu),
        .is_lui(is_lui),
        .is_j(is_j),
        .is_jal(is_jal),
        .is_div(is_div),
        .is_divu(is_divu),
        .is_mult(is_mult),
        .is_multu(is_multu),
        .is_mflo(is_mflo),
        .is_mfhi(is_mfhi),
        .is_mtlo(is_mtlo),
        .is_mthi(is_mthi),
        .is_lb(is_lb),
        .is_lbu(is_lbu),
        .is_lh(is_lh),
        .is_lhu(is_lhu),
        .is_sb(is_sb),
        .is_sh(is_sh),
        .is_bgez(is_bgez),
        .is_jalr(is_jalr),
        .is_break(is_break),
        .is_syscall(is_syscall),
```

```
            .is_teq(is_teq),
            .is_mfc0(is_mfc0),
            .is_mtc0(is_mtc0),
            .is_eret(is_eret),
            .is_clz(is_clz),

            .Rsc(Rsc),
            .Rtc(Rtc),
            .Rdc(Rdc),

           .shamt(shamt),
           .immediate(immediate),
           .addr(address)
        );
        /*instance of controller*/
        controller controller_inst(
            .is_add(is_add),
            .is_addu(is_addu),
            .is_sub(is_sub),
            .is_subu(is_subu),
            .is_and(is_and),
            .is_or(is_or),
            .is_xor(is_xor),
            .is_nor(is_nor),
            .is_slt(is_slt),
            .is_sltu(is_sltu),
            .is_sra(is_sra),
            .is_srl(is_srl),
            .is_sll(is_sll),
            .is_srav(is_srav),
            .is_srlv(is_srlv),
            .is_sllv(is_sllv),
            .is_jr(is_jr),
            .is_addi(is_addi),
            .is_addiu(is_addiu),
            .is_andi(is_andi),
            .is_ori(is_ori),
            .is_xori(is_xori),
            .is_lw(is_lw),
            .is_sw(is_sw),
            .is_beq(is_beq),
            .is_bne(is_bne),
            .is_slti(is_slti),
            .is_sltiu(is_sltiu),
            .is_lui(is_lui),
            .is_j(is_j),
            .is_jal(is_jal),
            .is_div(is_div),
            .is_divu(is_divu),
            .is_mult(is_mult),
            .is_multu(is_multu),
            .is_mflo(is_mflo),
            .is_mfhi(is_mfhi),
            .is_mtlo(is_mtlo),
            .is_mthi(is_mthi),
            .is_lb(is_lb),
            .is_lbu(is_lbu),
```

```
        .is_lh(is_lh),
        .is_lhu(is_lhu),
        .is_sb(is_sb),
        .is_sh(is_sh),
        .is_bgez(is_bgez),
        .is_jalr(is_jalr),
        .is_break(is_break),
        .is_syscall(is_syscall),
        .is_teq(is_teq),
        .is_mfc0(is_mfc0),
        .is_mtc0(is_mtc0),
        .is_eret(is_eret),
        .is_clz(is_clz),

        .Z(Z),
        .N(N),

        //.muxc(muxc),
        .muxc0(muxc[0]),
        .muxc1(muxc[1]),
        .muxc2(muxc[2]),
        .muxc3(muxc[3]),
        .muxc4(muxc[4]),
        .muxc5(muxc[5]),
        .muxc6(muxc[6]),
        .muxc7(muxc[7]),
        .muxc8(muxc[8]),
        .muxc9(muxc[9]),
        .muxc10(muxc[10]),
        .muxc11(muxc[11]),
        .muxc12(muxc[12]),

        .aluc(aluc),
        .signed_mult(signed_mult),
        .signed_div(signed_div),

        .ext5(ext5),
        .ext8(ext8),
        .s_ext8(s_ext8),
        .ext16(ext16),
        .s_ext16_0(s_ext16_0),
        .s_ext16_1(s_ext16_1),
        .s_ext18(s_ext18),

        .dmem_W(dmem_W),
        .dmem_R(dmem_R),
        .reg_W(reg_W),

        .Hi_W(Hi_W),
        .Lo_W(Lo_W),

        .cat(cat),

        .exception(exception)
    );
    /*instance of  ALU*/
    ALU alu_inst(
```

```
        .A(alu_A),
        .B(alu_B),
        .aluc(aluc),
        .alu_result(alu_result),
        .Z(Z),
        .C(C),
        .N(N),
        .O(O)
    );
    /*instance of  MULT DIV*/
    mult mult_inst(
        .signed_mult(signed_mult),
        .A(mult_A),
        .B(mult_B),
        .Hi(Hi),
        .Lo(Lo)
    );
    div div_inst(
        .signed_div(signed_div),
        .A(div_A),
        .B(div_B),
        .R(R),
        .Q(Q)
    );
    /*instance of regfile*/
    Regfile cpu_ref(
        .reg_clock(clk),
        .reg_ena(ena),
        .rst(rst_n),
        .reg_W(reg_W),
        .rdc(Rdc),
        .rsc(Rsc),
        .rtc(Rtc),
        .rd_in_data(rd_in_data),
        .rs_out_data(rs_out_data),
        .rt_out_data(rt_out_data)
    );
    /*instance of HILO*/
    Hi hi_inst(
        .clk(clk),
        .rst(rst_n),
        .Hi_W(Hi_W),
        .Hi_in_data(Hi_in_data),
        .Hi_out_data(Hi_out_data)
    );
    Lo lo_inst(
        .clk(clk),
        .rst(rst_n),
        .Lo_W(Lo_W),
        .Lo_in_data(Lo_in_data),
        .Lo_out_data(Lo_out_data)
    );
    /*instance of PC*/
    PC pc_inst(
        .pc_clock(clk),
        .pc_ena(ena),
        .rst(rst_n),
```

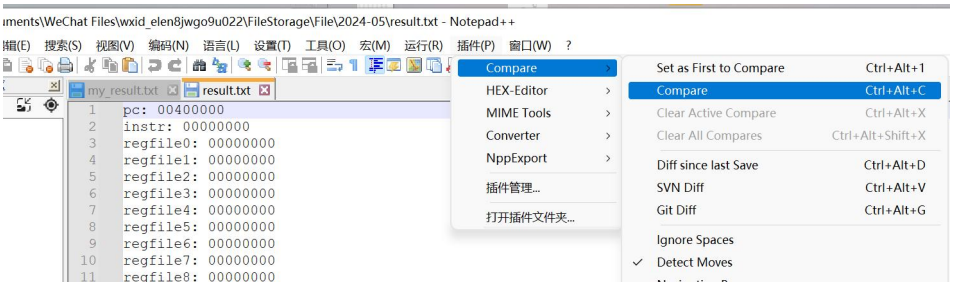
```
        .pc_in_next_addr(pc_in_next_addr),
        .pc_out_now_addr(pc_out_now_addr)
    );
    /*instance of  CP0*/
    cp0 cp0_inst(
        .clk(clk),
        .rst(rst_n),
        .mfc0(is_mfc0),
        .mtc0(is_mtc0),
        .pc(pc_out_now_addr),
        .Rd(Rd),
        .wdata(wdata),
        .exception(exception),
        .eret(is_eret),
        .cause(cause),
        .rdata(rdata),
        .exec_addr(exec_addr)
    );
    /*instance of CAL*/
    CAL cal_inst(
        .cal_in_data(cal_in_data),
        .cal_out_result(cal_out_result)
    );
//------------------------------------------------------------------
endmodule
```

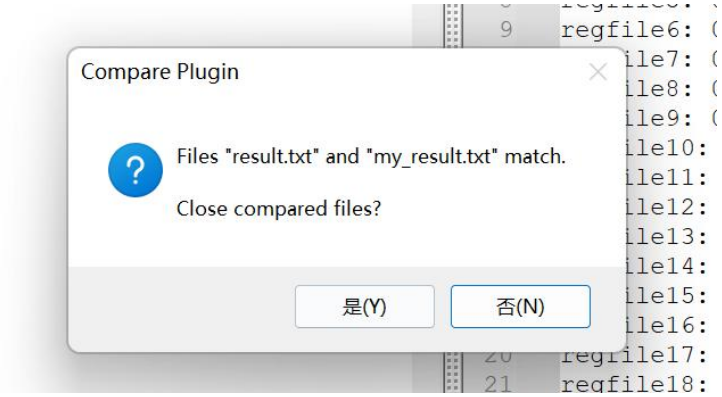
四、CPU 前仿真测试结果

（一）单条指令前仿真波形图及对比结果

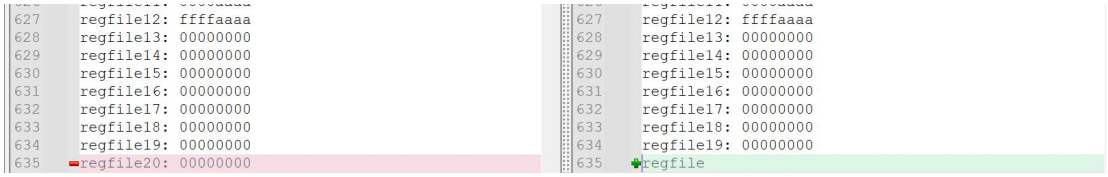
前仿真测试主要用于测试所编写的代码电平逻辑是否正确，无需考虑延时，因此是一种理想情况下的测试。编写了测试文件 cpu31_tb.v 进行前仿真测试，通过跟踪如下变量的值及其波形变化来检验前仿真测试是否成功：clk，rst，[31:0]inst 和[31:0]pc。

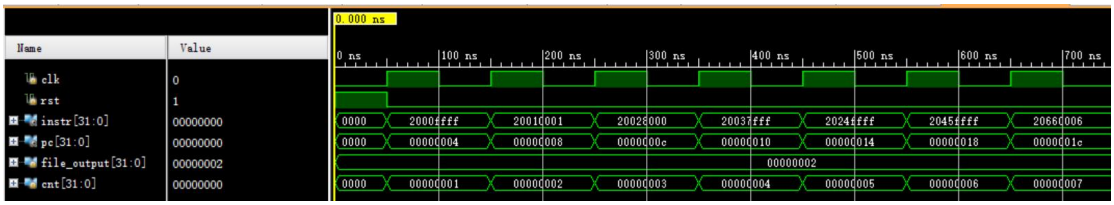
使用 modelsim 对单条指令进行前仿真测试的波形图如下。同时将每条指令执行后通用寄存器堆中的 32 个寄存器的值打印输出至 result.txt 中，使用 notepad++的 compare 插件对生成结果与标准结果进行对比，对比结果一并展示如下：



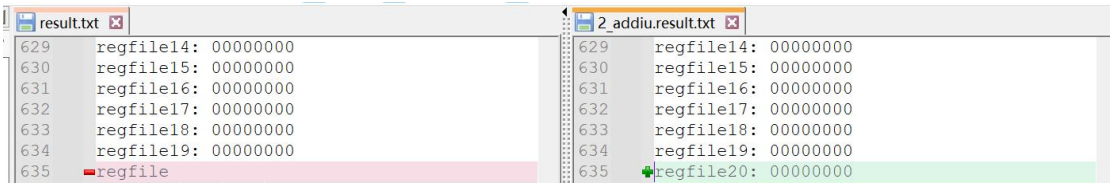


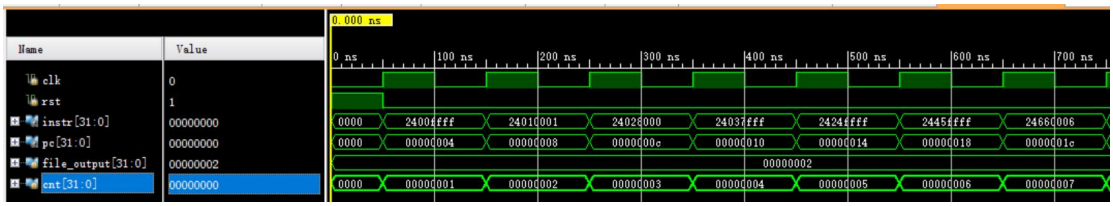
1.Addi



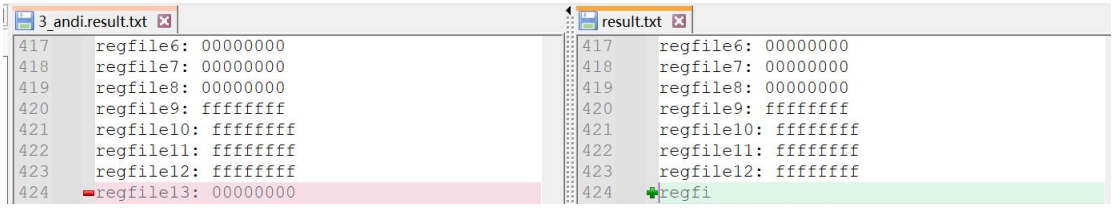


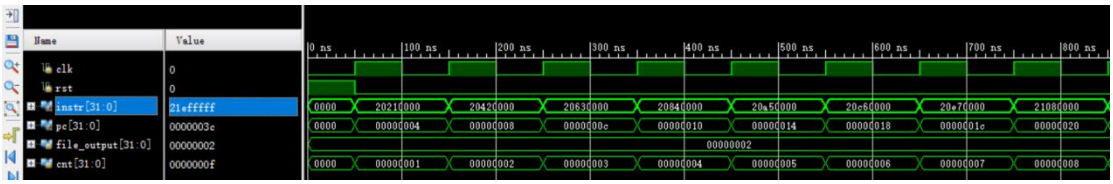
2.Addiu



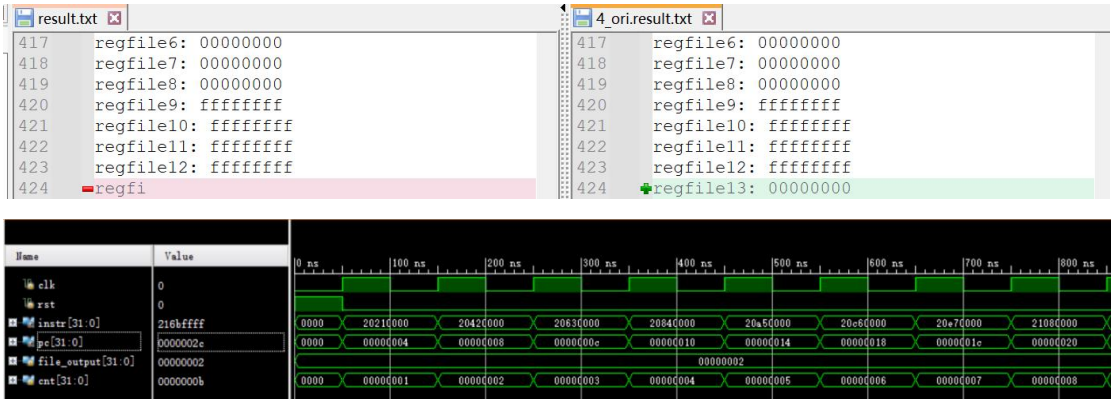


3.Andi

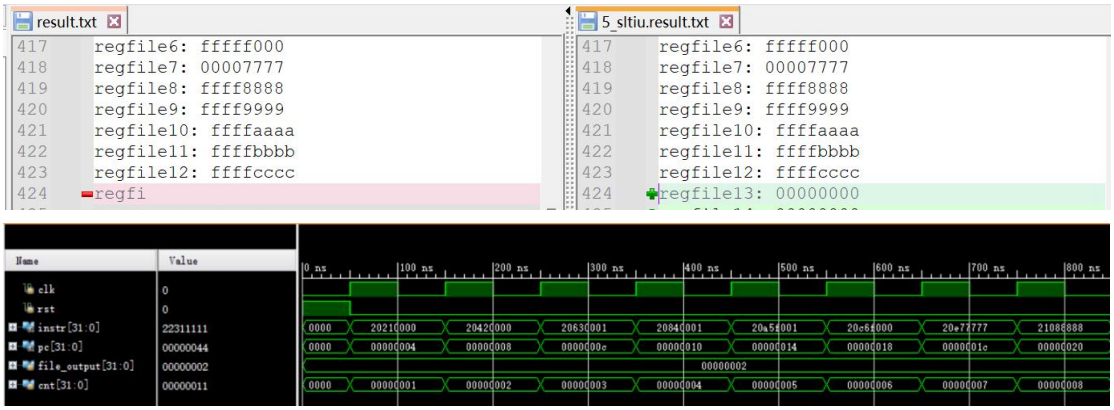




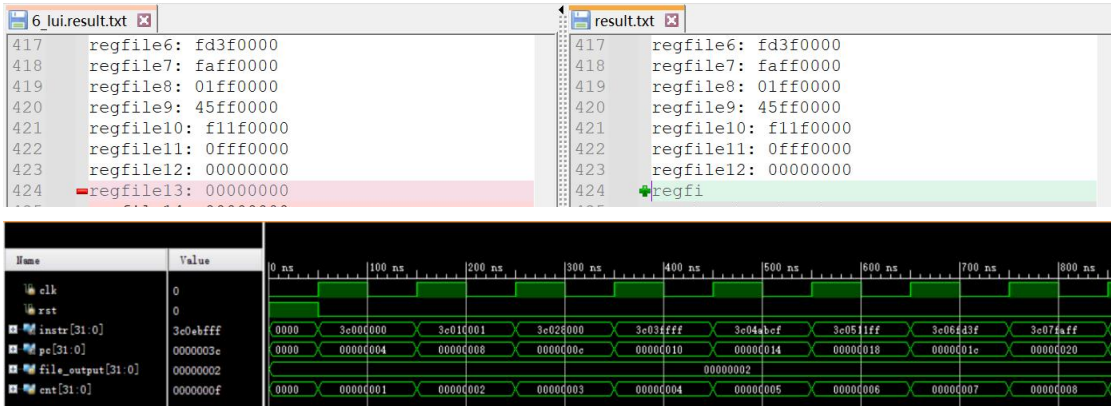
4.Ori



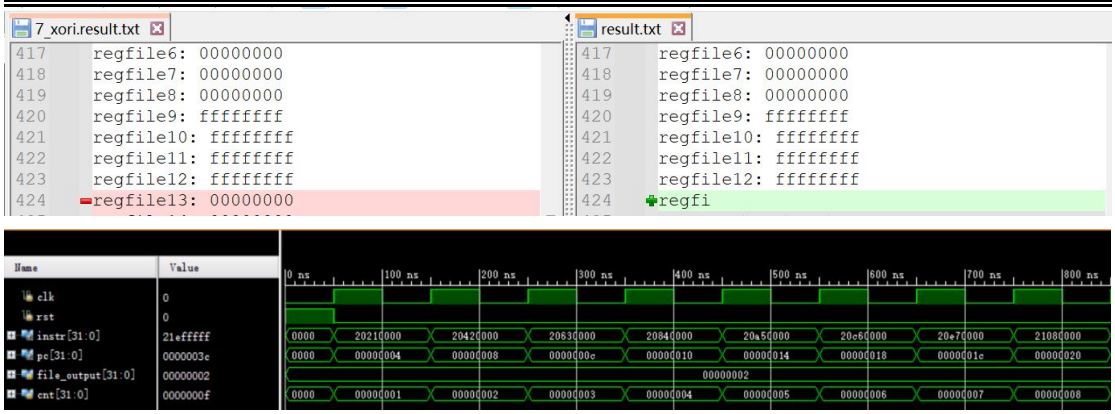
5.Sltiu



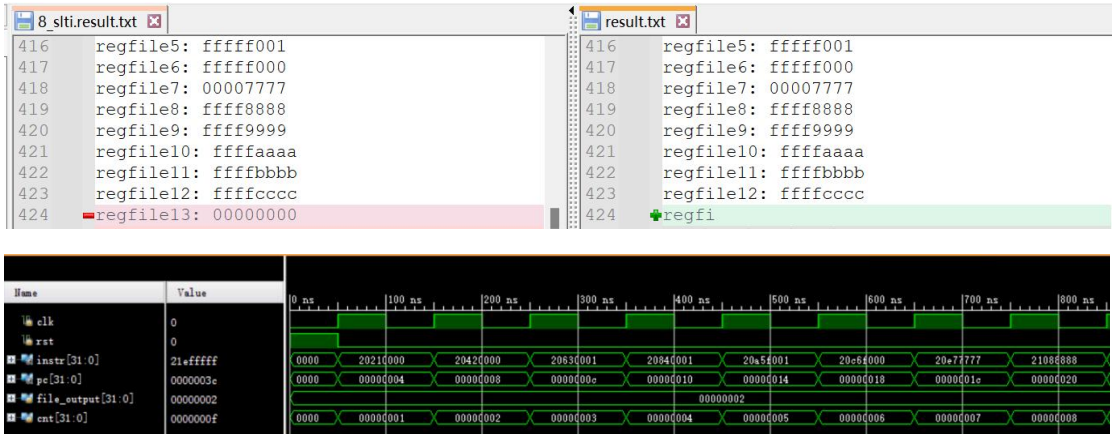
6.Lui



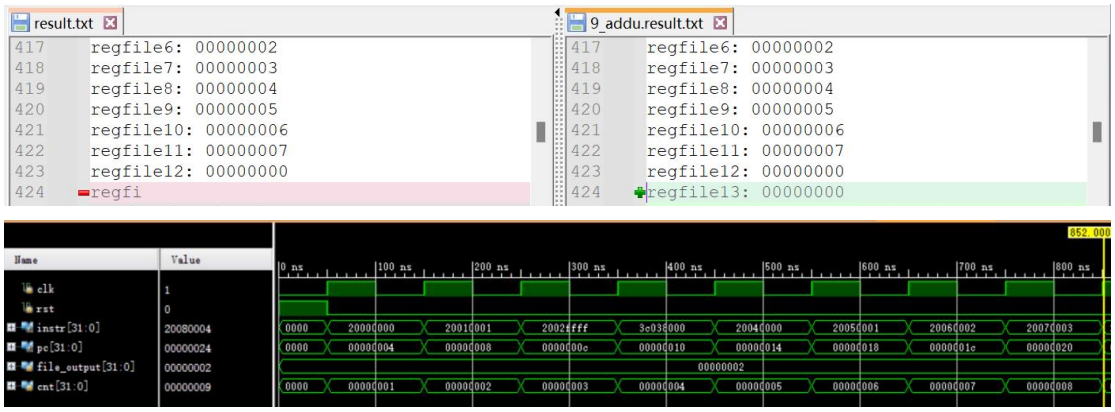
7.Xori



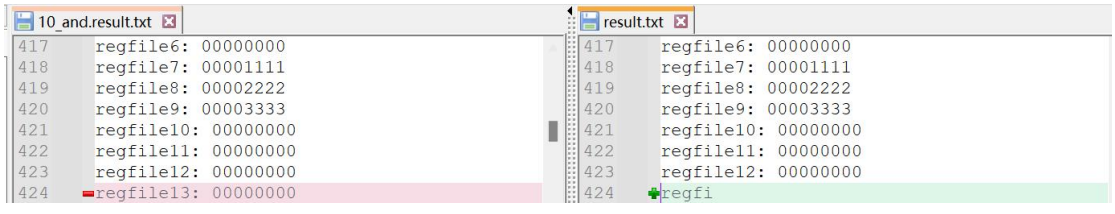
8.Slti

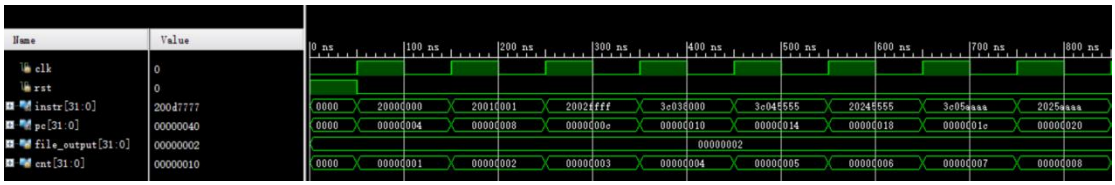


9.Addu

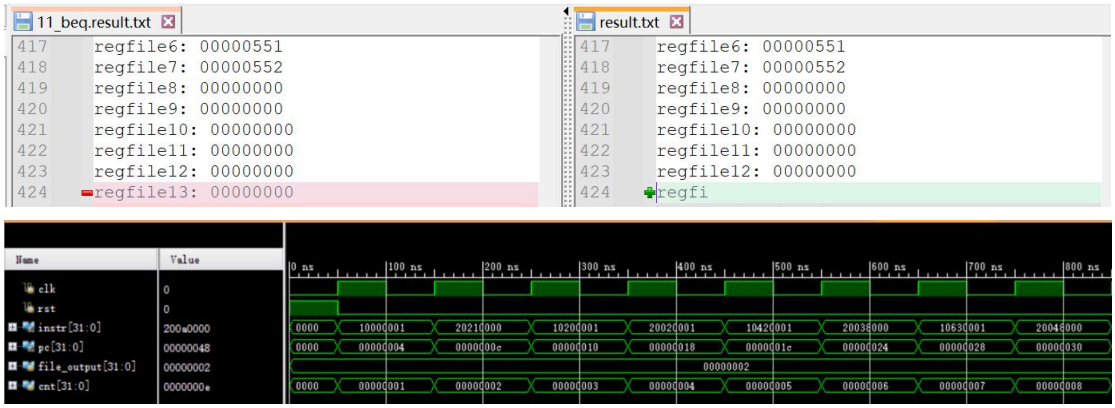


10.And

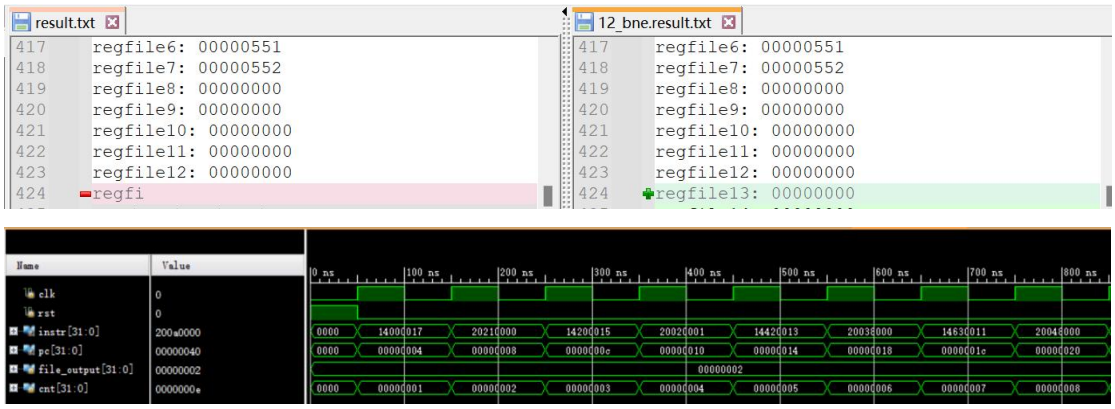




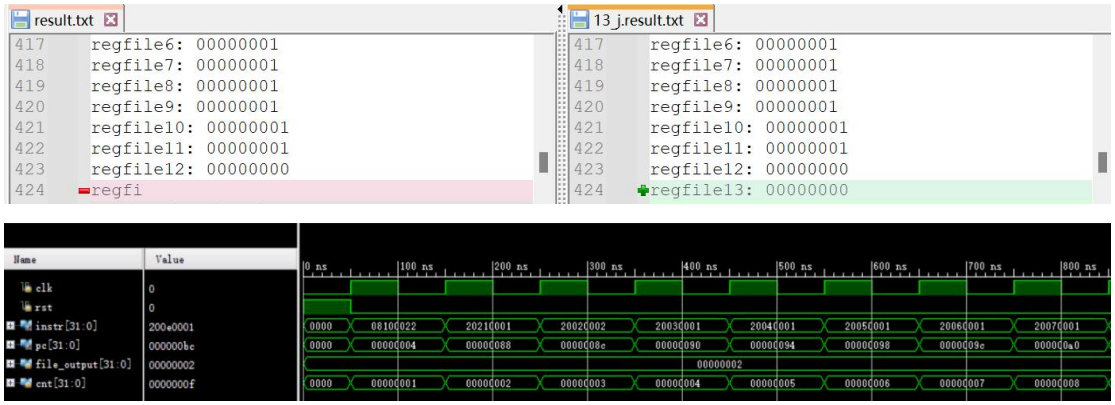
11.Beq



12.Bne



13.J



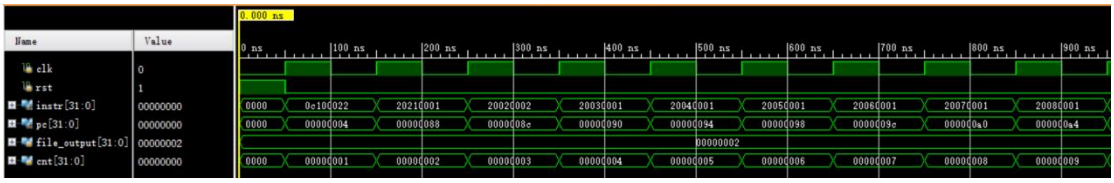
14.Jal

result.txt

```
841    regfile22: 00000001
842    regfile23: 00000001
843    regfile24: 00000000
844    regfile25: 00000000
845    regfile26: 00000000
846    regfile27
847
```

14_jal.result.txt

```
841    regfile22: 00000001
842    regfile23: 00000001
843    regfile24: 00000000
844    regfile25: 00000000
845    regfile26: 00000000
846    regfile27: 00000000
847    regfile28: 00000000
```



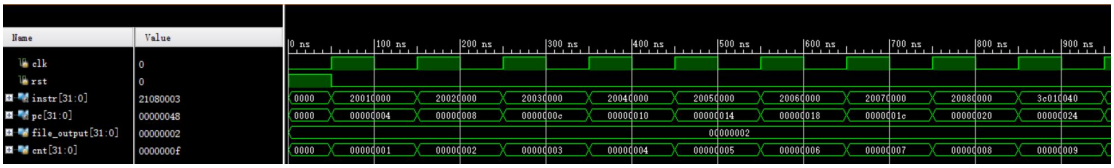
15.Jr

15_jr.result.txt

```
417    regfile6: 00000000
418    regfile7: 00000000
419    regfile8: 00000000
420    regfile9: 00000000
421    regfile10: 00000000
422    regfile11: 00000000
423    regfile12: 00000000
424    regfile13: 00000000
```

result.txt

```
417    regfile6: 00000000
418    regfile7: 00000000
419    regfile8: 00000000
420    regfile9: 00000000
421    regfile10: 00000000
422    regfile11: 00000000
423    regfile12: 00000000
424    regfi
```



16.

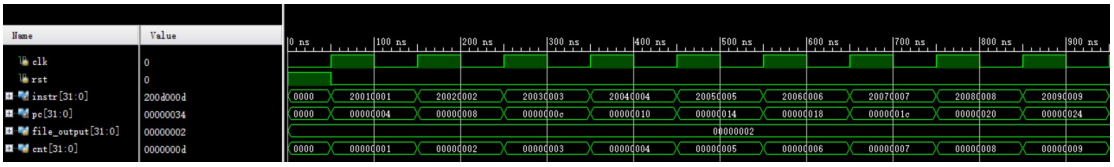
LwSw

result.txt

```
418    regfile7: 00000007
419    regfile8: 00000008
420    regfile9: 00000009
421    regfile10: 0000000a
422    regfile11: 0000000b
423    regfile12: 0000000c
424    regfi
```

16.26_lwsw.result.txt

```
418    regfile7: 00000007
419    regfile8: 00000008
420    regfile9: 00000009
421    regfile10: 0000000a
422    regfile11: 0000000b
423    regfile12: 0000000c
424    regfile13: 00000000
```



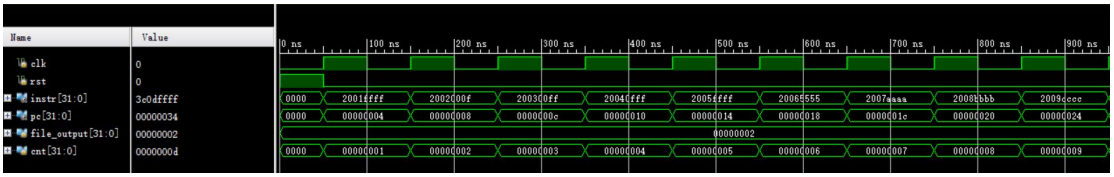
LwSw2

result.txt

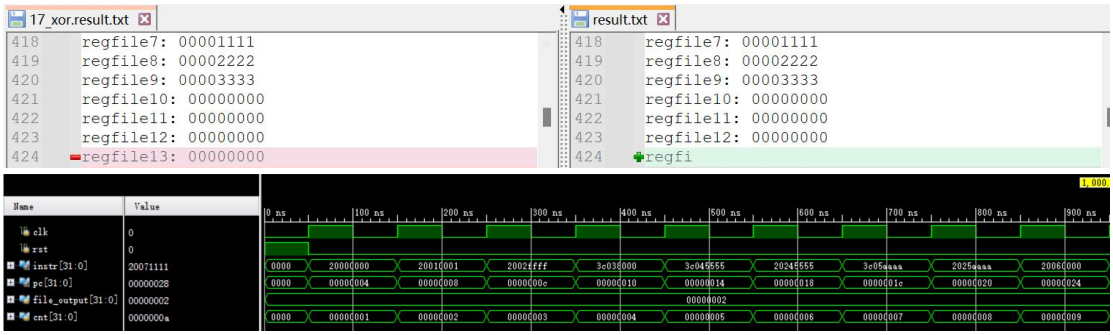
```
418    regfile7: ffffaaaa
419    regfile8: ffffbbbb
420    regfile9: ffffcccc
421    regfile10: ffffdddd
422    regfile11: ffffeeee
423    regfile12: ffffffff
424    regfi
```

16.26_lwsw2.result.txt

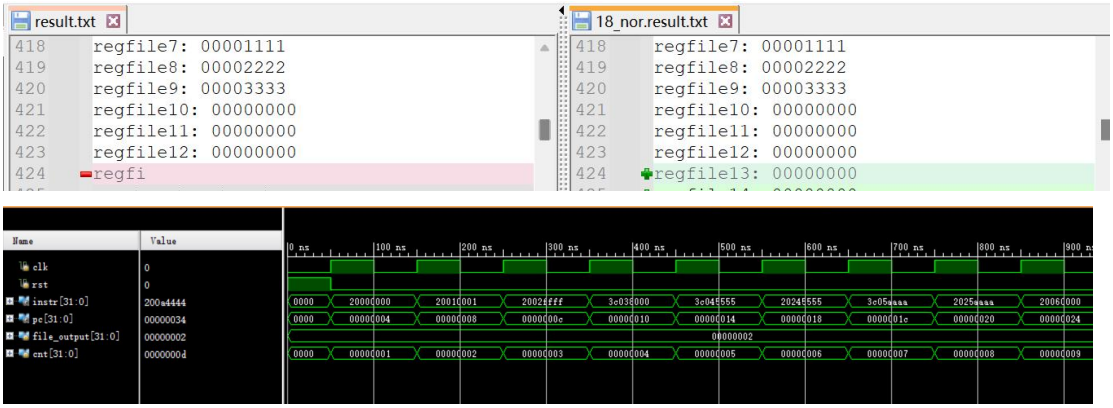
```
418    regfile7: ffffaaaa
419    regfile8: ffffbbbb
420    regfile9: ffffcccc
421    regfile10: ffffdddd
422    regfile11: ffffeeee
423    regfile12: ffffffff
424    regfile13: 00000000
```



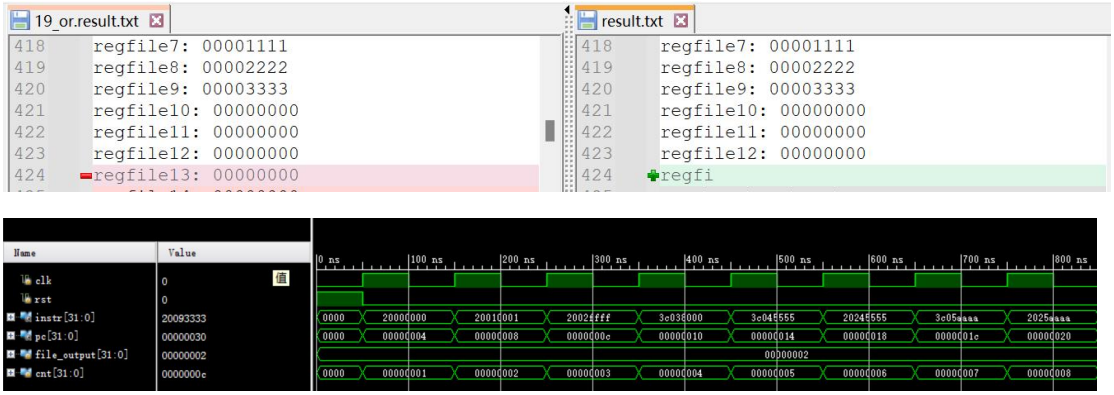
17.Xor



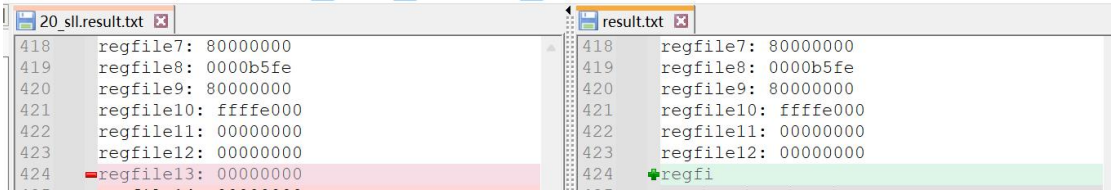
18.Nor

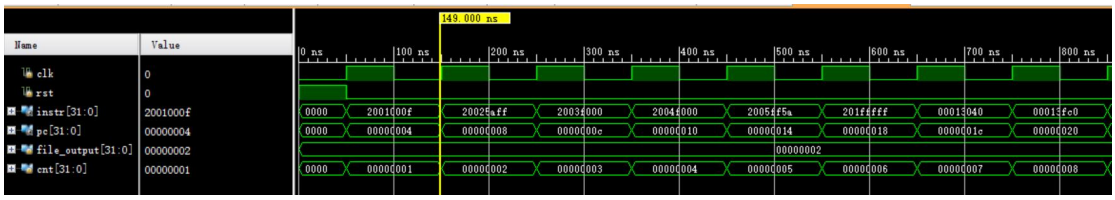


19.Or

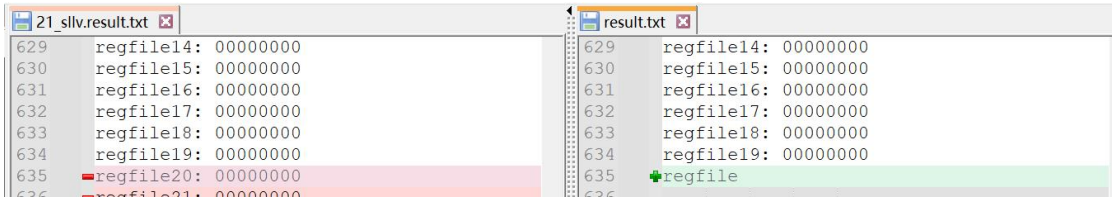


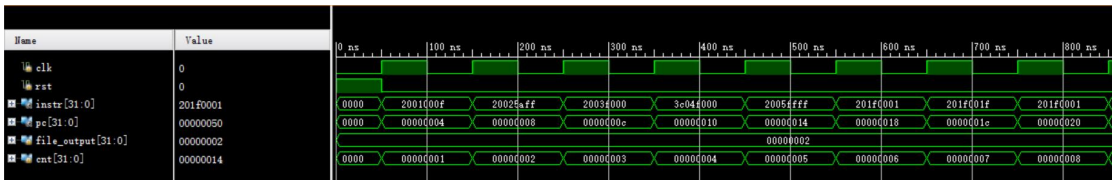
20.Sll



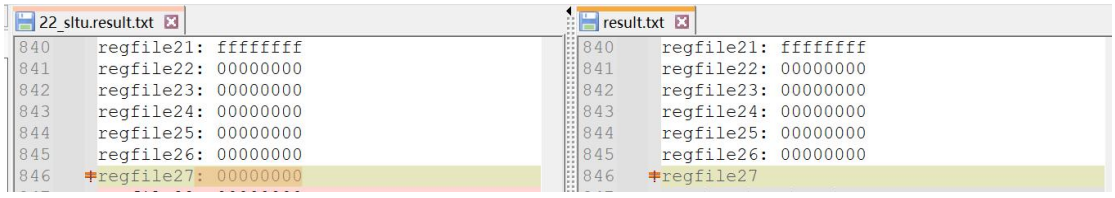


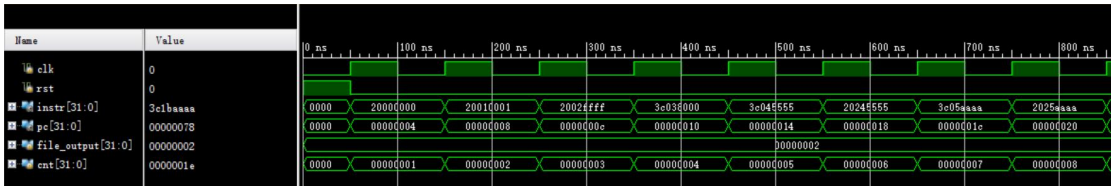
21.Sllv



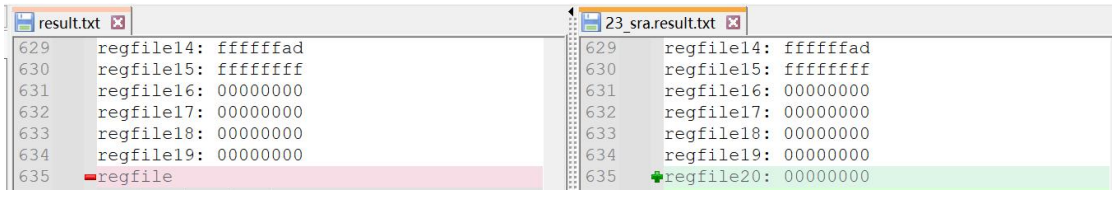


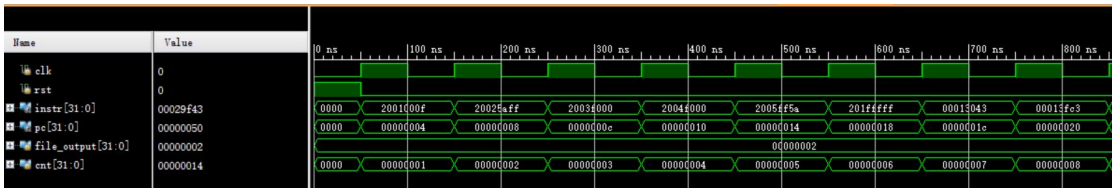
22.Sltu





23.Sra





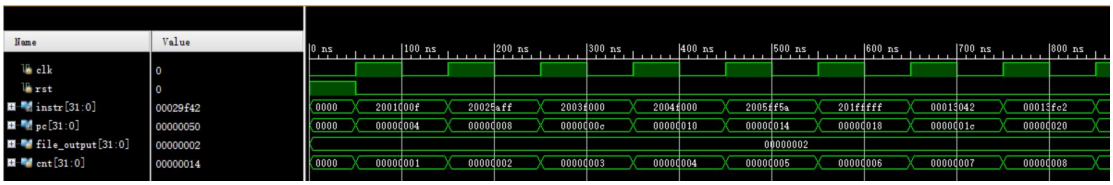
24.Srl

result.txt

```
629     regfile14: 7fffffad
630     regfile15: 00000001
631     regfile16: 00000000
632     regfile17: 00000000
633     regfile18: 00000000
634     regfile19: 00000000
635   - regfile
```

24_srl.result.txt

```
629     regfile14: 7fffffad
630     regfile15: 00000001
631     regfile16: 00000000
632     regfile17: 00000000
633     regfile18: 00000000
634     regfile19: 00000000
635   + regfile20: 00000000
```



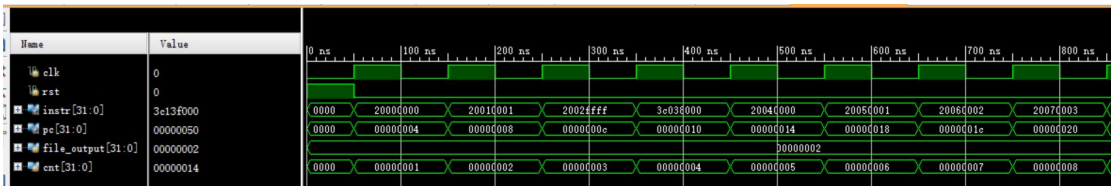
25.Subu

result.txt

```
629     regfile14: 0000000a
630     regfile15: 0000000b
631     regfile16: 0000000c
632     regfile17: 0000000d
633     regfile18: 00000000
634     regfile19: 00000000
635   - regfile
```

25_subu.result.txt

```
629     regfile14: 0000000a
630     regfile15: 0000000b
631     regfile16: 0000000c
632     regfile17: 0000000d
633     regfile18: 00000000
634     regfile19: 00000000
635   + regfile20: 00000000
```



26.16 已经测试

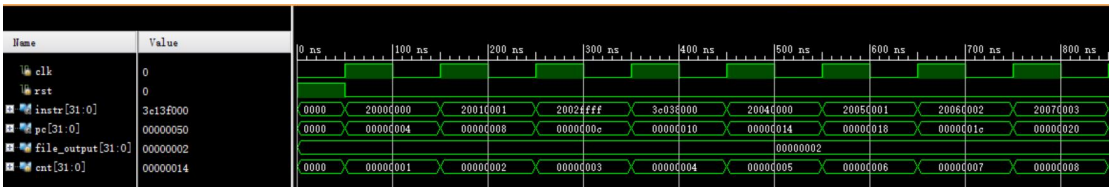
27.Add

result.txt

```
629     regfile14: 0000000a
630     regfile15: 0000000b
631     regfile16: 0000000c
632     regfile17: 0000000d
633     regfile18: 00000000
634     regfile19: 00000000
635   - regfile
```

27_add.result.txt

```
629     regfile14: 0000000a
630     regfile15: 0000000b
631     regfile16: 0000000c
632     regfile17: 0000000d
633     regfile18: 00000000
634     regfile19: 00000000
635   + regfile20: 00000000
```



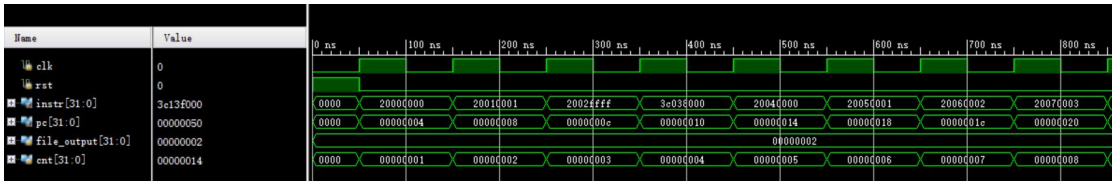
28.Sub

result.txt

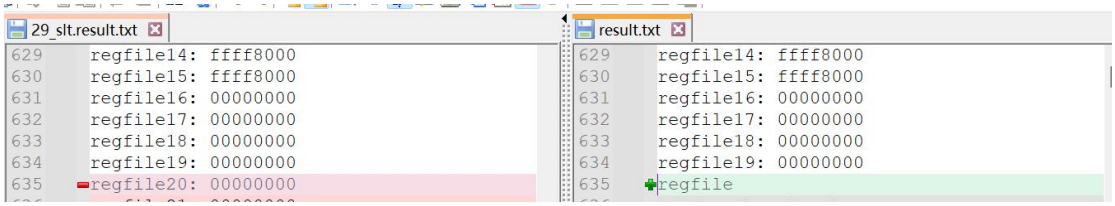
```
629     regfile14: 0000000a
630     regfile15: 0000000b
631     regfile16: 0000000c
632     regfile17: 0000000d
633     regfile18: 00000000
634     regfile19: 00000000
635   - regfile
```

28_sub.result.txt

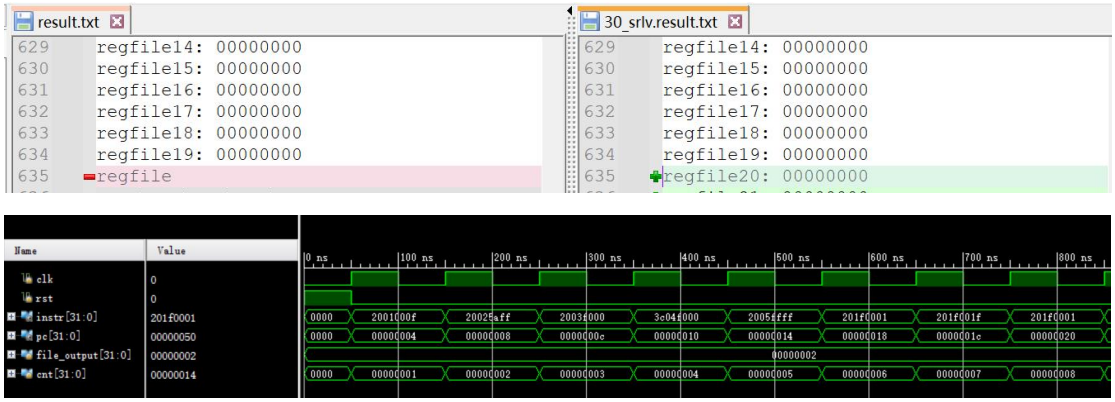
```
629     regfile14: 0000000a
630     regfile15: 0000000b
631     regfile16: 0000000c
632     regfile17: 0000000d
633     regfile18: 00000000
634     regfile19: 00000000
635   + regfile20: 00000000
```



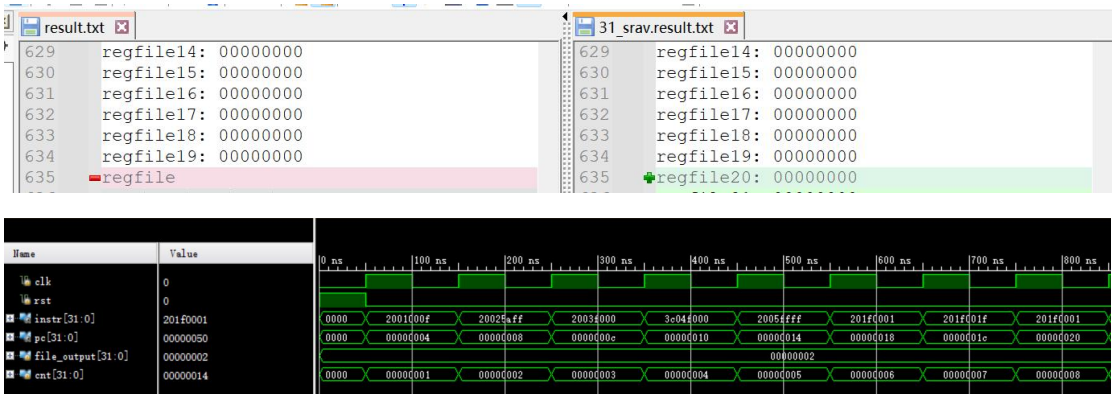
29.Slt



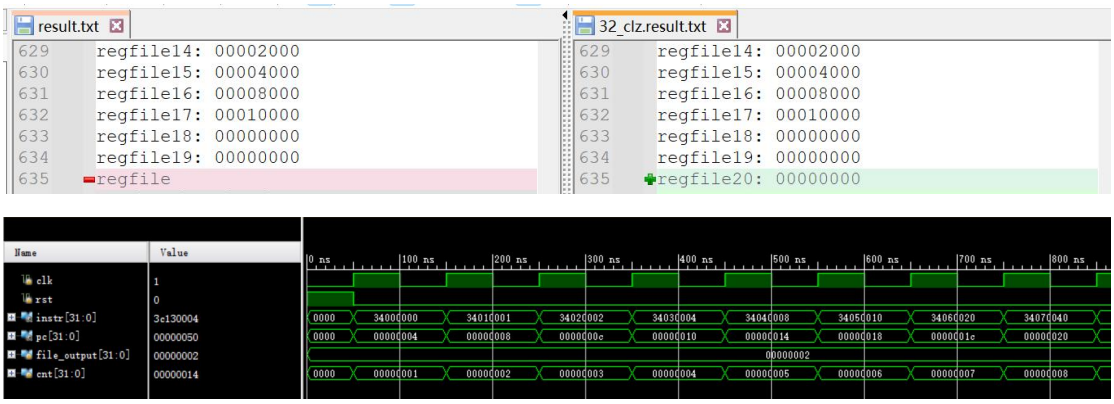
30.Srlv



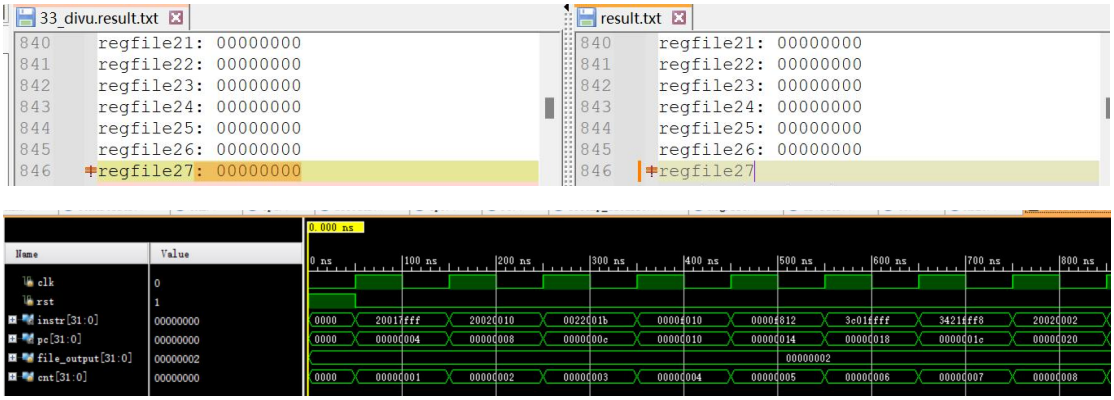
31.Srav



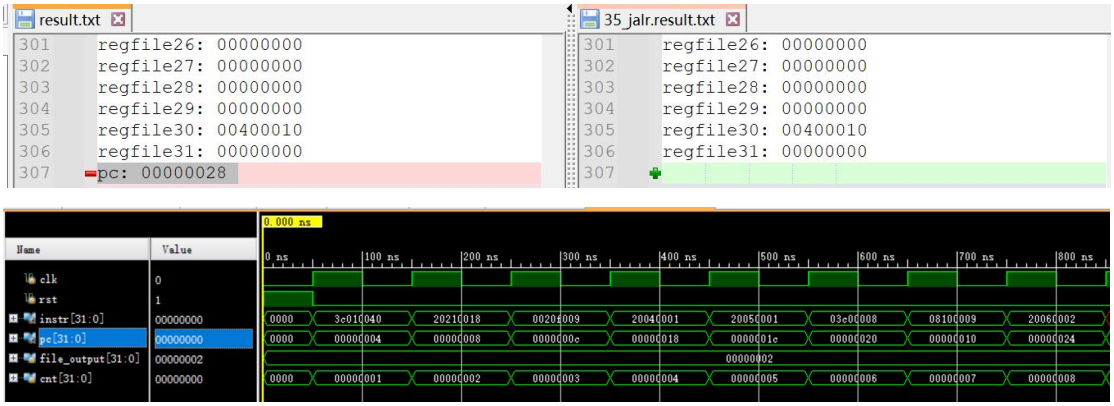
32.Clz



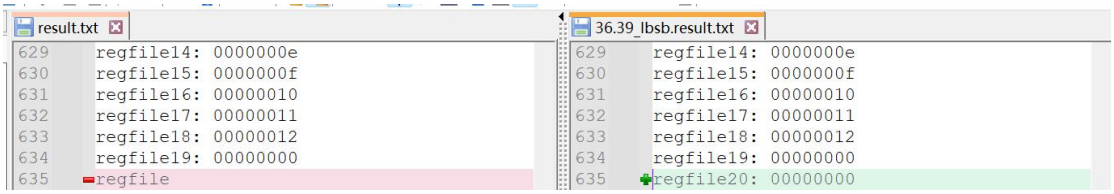
33.Divu

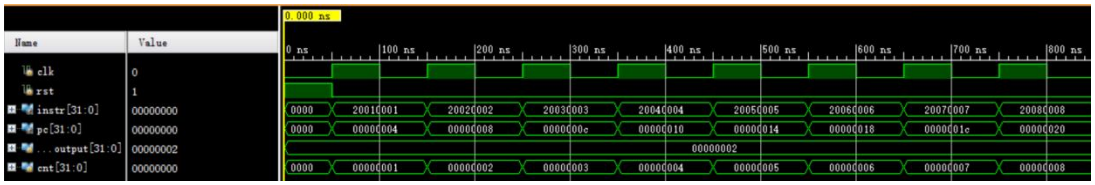


35. Jalr

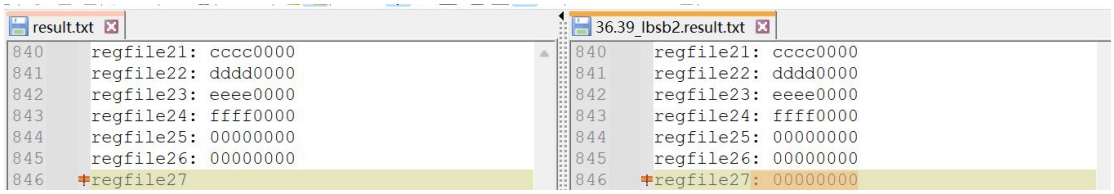


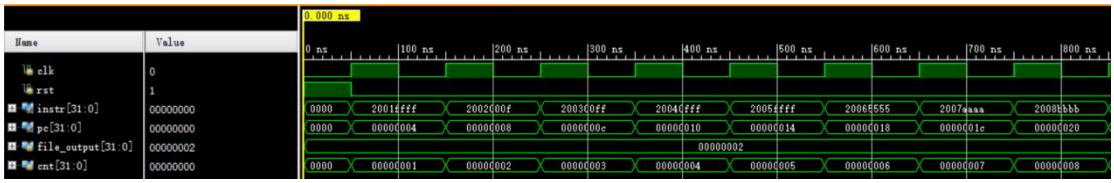
36.39 LbSb



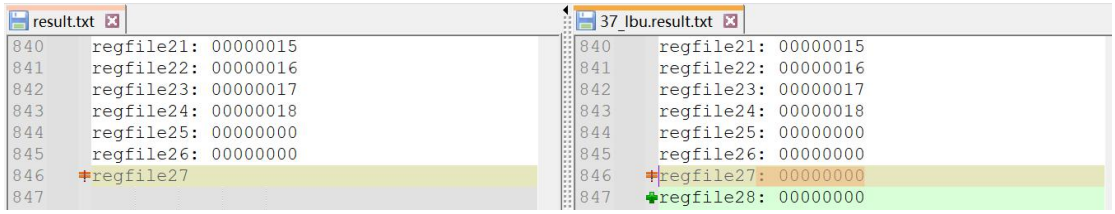


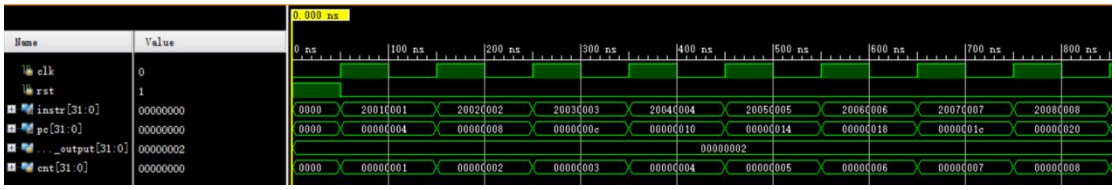
LbSb2



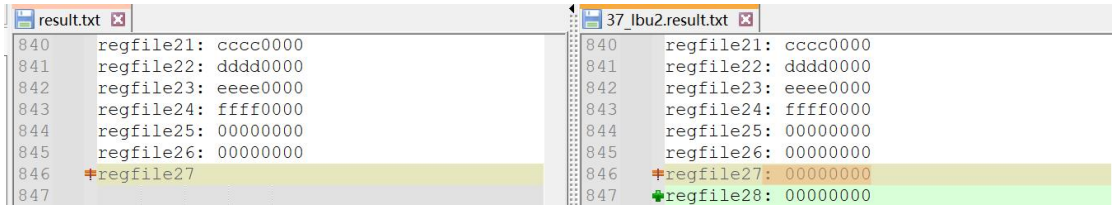


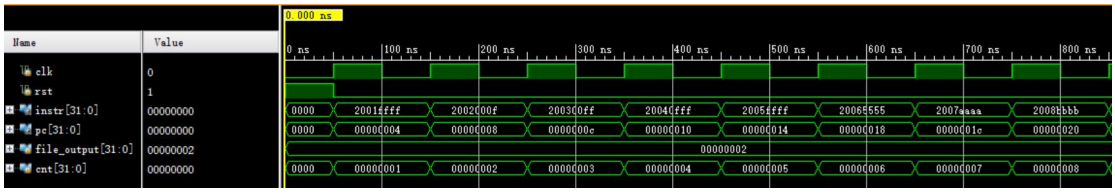
37.Lbu





Lbu2





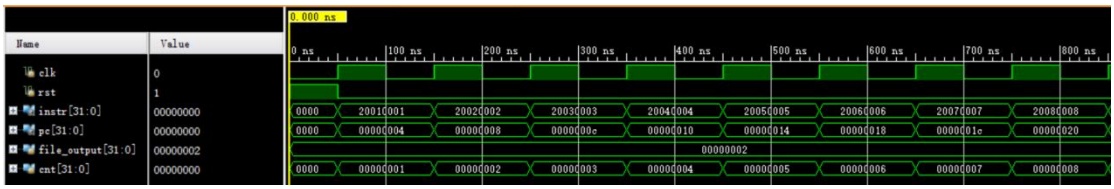
38.Lhu

result.txt

```
840  regfile21: 00000015
841  regfile22: 00000016
842  regfile23: 00000017
843  regfile24: 00000018
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27
```

38_lhu.result.txt

```
840  regfile21: 00000015
841  regfile22: 00000016
842  regfile23: 00000017
843  regfile24: 00000018
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27: 00000000
```



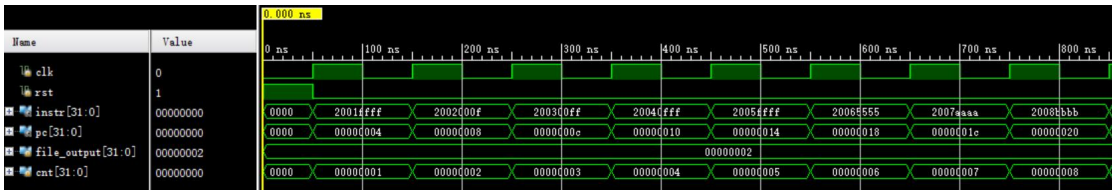
39.Lhu2

result.txt

```
840  regfile21: cccc0000
841  regfile22: dddd0000
842  regfile23: eeee0000
843  regfile24: ffff0000
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27
```

38_lhu2.result.txt

```
840  regfile21: cccc0000
841  regfile22: dddd0000
842  regfile23: eeee0000
843  regfile24: ffff0000
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27: 00000000
```



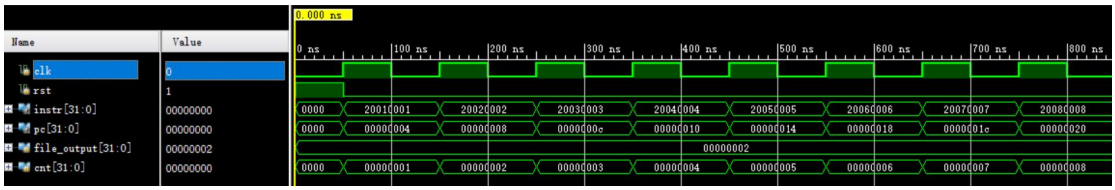
40.Lhsh

result.txt

```
840  regfile21: 00000015
841  regfile22: 00000016
842  regfile23: 00000017
843  regfile24: 00000018
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27
847
```

40.41_lhsh.result.txt

```
840  regfile21: 00000015
841  regfile22: 00000016
842  regfile23: 00000017
843  regfile24: 00000018
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27: 00000000
847  regfile28: 00000000
```



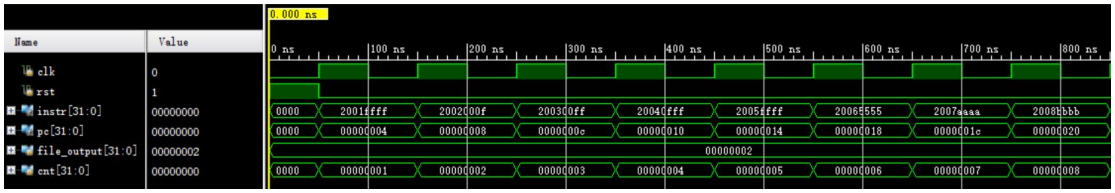
41.Lhsh2

result.txt

```
840  regfile21: cccc0000
841  regfile22: dddd0000
842  regfile23: eeee0000
843  regfile24: ffff0000
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27
847
```

40.41_lhsh2.result.txt

```
840  regfile21: cccc0000
841  regfile22: dddd0000
842  regfile23: eeee0000
843  regfile24: ffff0000
844  regfile25: 00000000
845  regfile26: 00000000
846  regfile27: 00000000
847  regfile28: 00000000
```



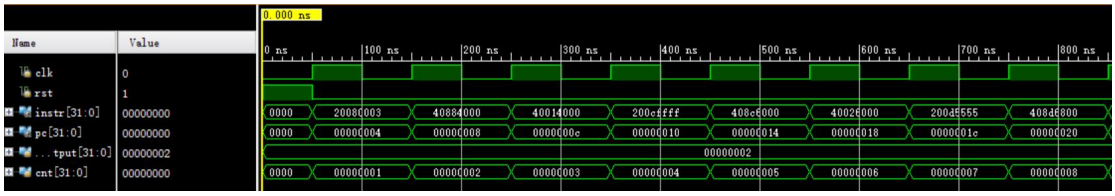
42.45 Mfc0 Mtc0

result.txt

```
437    regfile26: 00000000
438    regfile27: 00000000
439    regfile28: 00000000
440    regfile29: 00000000
441    regfile30: 00000000
442    regfile31: 00000000
443    pc: 00000034
```

42.45_mfc0mtc0.result.txt

```
437    regfile26: 00000000
438    regfile27: 00000000
439    regfile28: 00000000
440    regfile29: 00000000
441    regfile30: 00000000
442    regfile31: 00000000
443
```



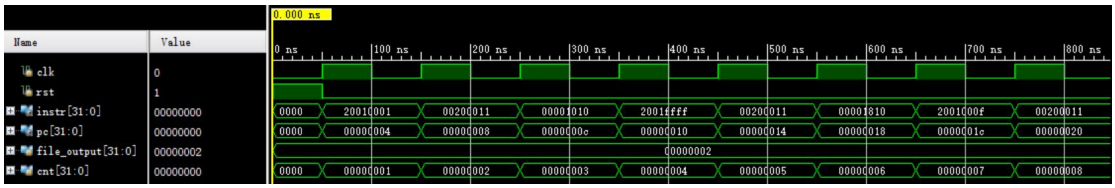
43.46 Mfhi Mthi

result.txt

```
629    regfile14: 00000000
630    regfile15: 00000000
631    regfile16: 00000000
632    regfile17: 00000000
633    regfile18: 00000000
634    regfile19: 00000000
635    regfile
```

43.46_mfhi.mthi.result.txt

```
629    regfile14: 00000000
630    regfile15: 00000000
631    regfile16: 00000000
632    regfile17: 00000000
633    regfile18: 00000000
634    regfile19: 00000000
635    regfile20: 00000000
```



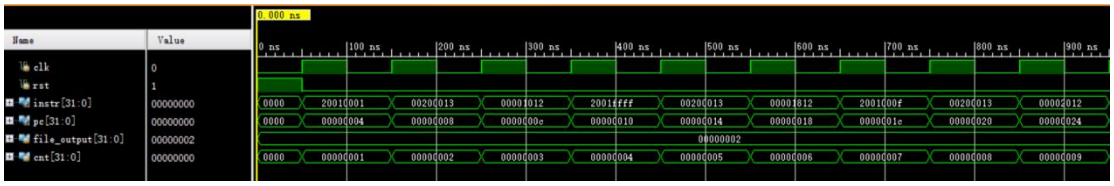
44.47 Mflo Mtlo

result.txt

```
629    regfile14: 00000000
630    regfile15: 00000000
631    regfile16: 00000000
632    regfile17: 00000000
633    regfile18: 00000000
634    regfile19: 00000000
635    regfile
```

44.47_mflo.mtlo.result.txt

```
629    regfile14: 00000000
630    regfile15: 00000000
631    regfile16: 00000000
632    regfile17: 00000000
633    regfile18: 00000000
634    regfile19: 00000000
635    regfile20: 00000000
```



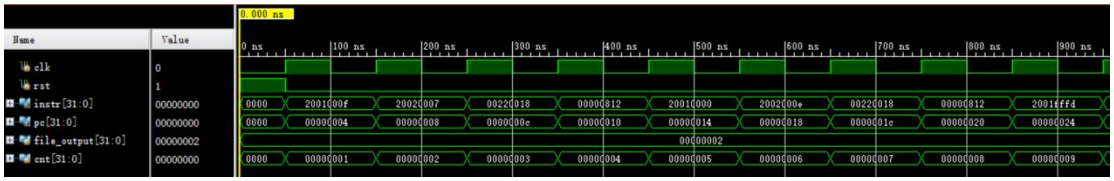
48. Mult

result.txt

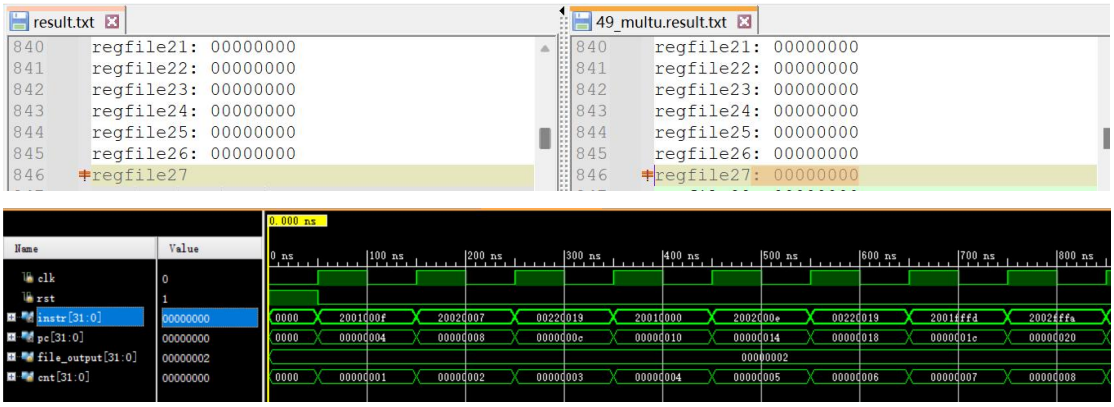
```
840    regfile21: 00000000
841    regfile22: 00000000
842    regfile23: 00000000
843    regfile24: 00000000
844    regfile25: 00000000
845    regfile26: 00000000
846    regfile27
```

48_mult.result.txt

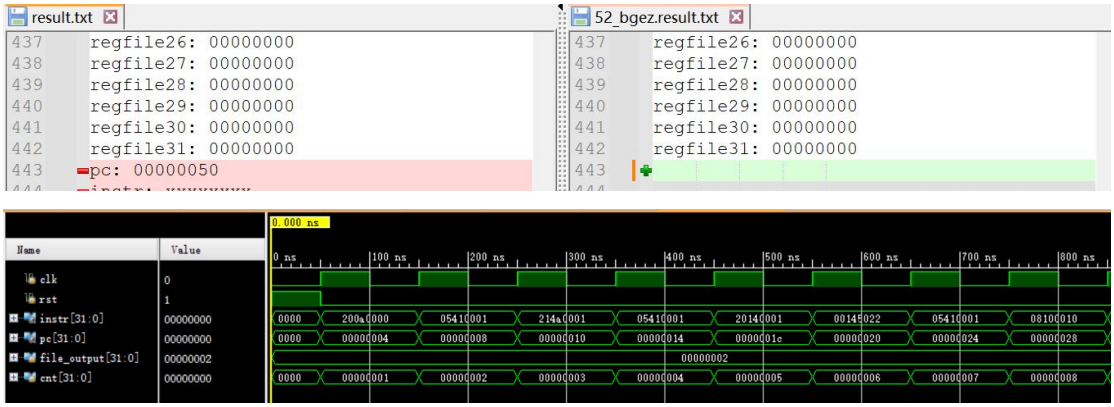
```
840    regfile21: 00000000
841    regfile22: 00000000
842    regfile23: 00000000
843    regfile24: 00000000
844    regfile25: 00000000
845    regfile26: 00000000
846    regfile27: 00000000
```



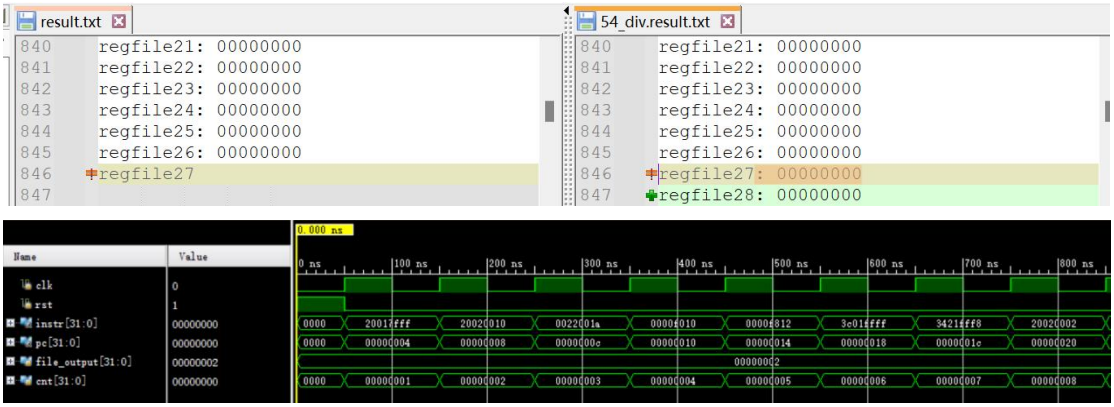
49. Multu



52. Bgez



54. Div



（二）网站前仿真测试结果

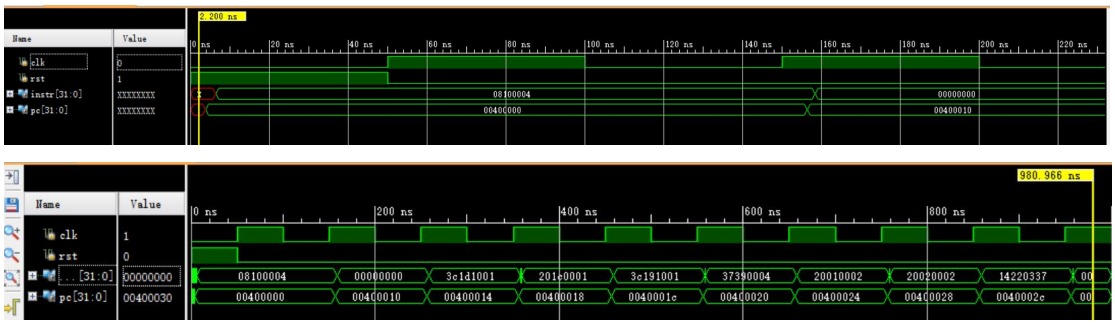
实验	结果	提交时间	
10	AC	2024/7/17 18:31:11	--

五、CPU 后仿真测试结果

（一）后仿真测试结果及描述

完成前仿真以后，后仿真测试可以真实地模拟实际情况下存在延时的效果，从而进行更加符合真实情况的仿真测试。

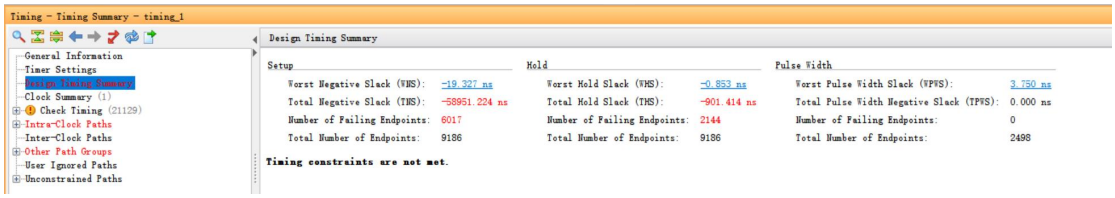
配置好相关的时钟约束：时钟变量、input delay、output delay 以后，即可通过 modelsim 与 vivado 链接生成后仿真波形图，经过比对可以得出结果正确：



（二）时序分析结果

在 vivado 下进行后仿真测试，可以通过 report timing summary 生成具体的时序分析报告内容。时序报告中可以详细地查看每条路径的具体时延报告，分为 summary，source clock path，data path 以及 destination path 等部分。我们通过改进逻辑布线布局，降低扇出以及减少逻辑门级数等操作可以有效地减少延迟，优化电路。

本次实验的时序分析结果如下：



Intra-Clock Paths - clk_pin - Setup

Name	...	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination C
Path 1	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[16]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 2	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[17]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 3	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[18]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 4	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[19]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 5	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[20]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 6	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[21]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 7	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[22]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 8	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[23]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 9	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[24]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin
Path 10	-19.327	34	1030	sccpu/pc_inst/pc_reg_reg[2]/C	mem/Dmem/dmem_out_data1_retimed_reg[25]/D	24.194	6.316	17.878	5.000	clk_pin	clk_pin

（path1 详细展开）

Summary

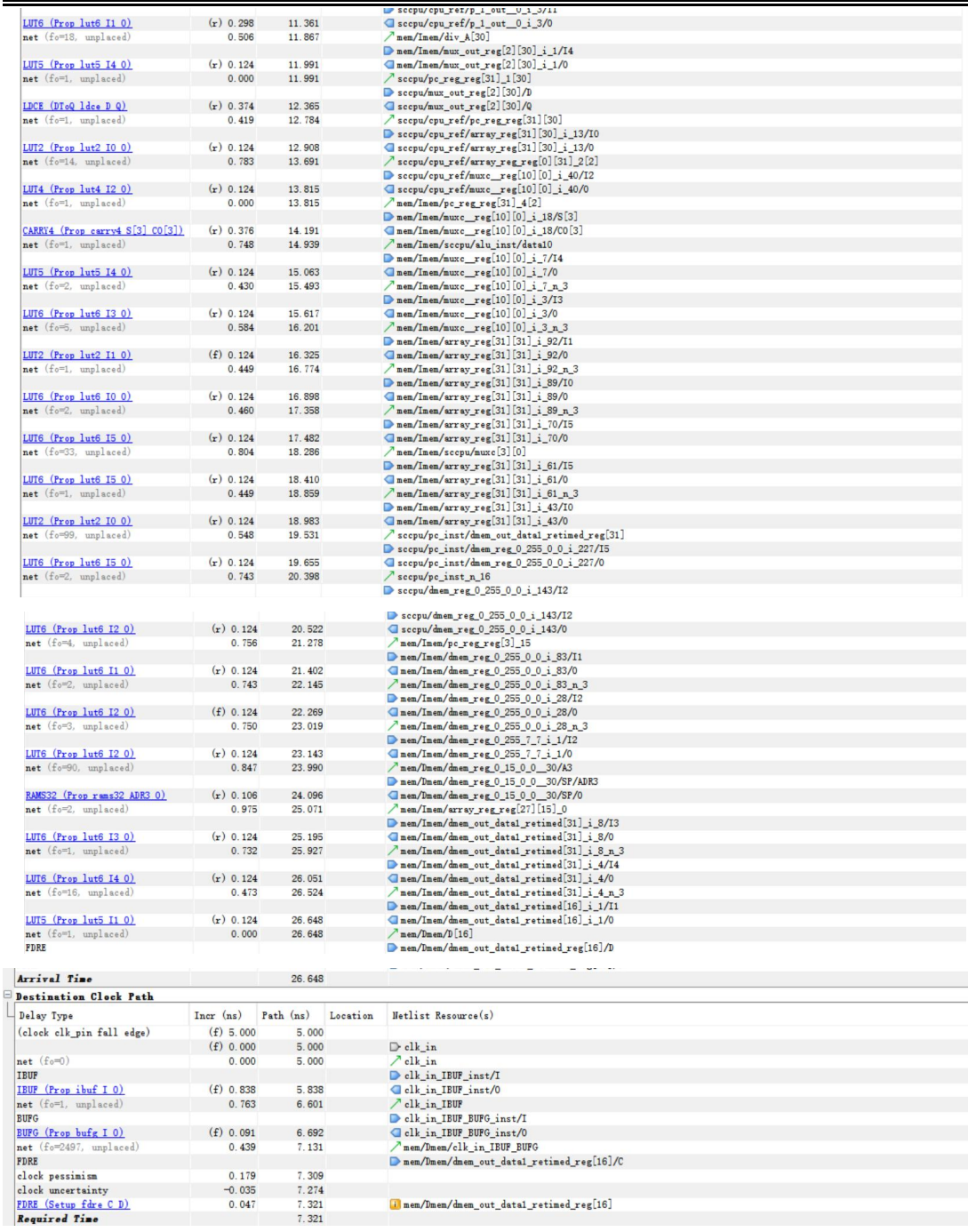
Name	Path 1
Slack	-19.327ns
Source	sccpu/pc_inst/pc_reg_reg[2]/C (rising edge-triggered cell FDRE clocked by clk_pin {rise@0.000ns fall@5.000ns period=10.000ns})
Destination	mem/Dmem/dmem_out_data1_retimed_reg[16]/D (falling edge-triggered cell FDRE clocked by clk_pin {rise@0.000ns fall@5.000ns period=10.000ns})
Path Group	clk_pin
Path Type	Setup (Max at Slow Process Corner)
Requirement	5.000ns (clk_pin fall@5.000ns - clk_pin rise@0.000ns)
Data Path Delay	24.194ns (logic 6.316ns (26.106%) route 17.878ns (73.894%))
Logic Levels	34 (CARRY4=1 LDCE=1 LUT2=4 LUT3=1 LUT4=1 LUT5=3 LUT6=17 MUXF7=3 MUXF8=1 RAMS32=1 RAMS64E=1)
Clock Path Skew	-0.145ns
Clock Uncertainty	0.035ns

Source Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk_pin rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000		clk_in
net (fo=0)	0.000	0.000		clk_in
				clk_in_IBUF_inst/I
IBUF (Prop ibuf I O)	(r) 0.971	0.971		clk_in_IBUF_inst/O
net (fo=1, unplaced)	0.803	1.774		clk_in_IBUF
				clk_in_IBUF_BUFG_inst/I
BUFG (Prop bufg I O)	(r) 0.096	1.870		clk_in_IBUF_BUFG_inst/O
net (fo=2497, unplaced)	0.584	2.454		sccpu/pc_inst/clk_in_IBUF_BUFG
FDRE				sccpu/pc_inst/pc_reg_reg[2]/C

Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
FDRE (Prop fdre C Q)	(r) 0.478	2.932		sccpu/pc_inst/pc_reg_reg[2]/Q
net (fo=1030, unplaced)	1.024	3.956		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/A0
				mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/RAMS64E_C/ADR0
RAMS64E (Prop rams64e ADR0 O)	(f) 0.376	4.332		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/RAMS64E_C/O
net (fo=1, unplaced)	0.000	4.332		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/OC
				mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/F7.B/I1
MUXF7 (Prop muxf7 I1 O)	(f) 0.247	4.579		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/F7.B/O
net (fo=1, unplaced)	0.000	4.579		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/O0
				mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/F8/I0
MUXF8 (Prop muxf8 I0 O)	(f) 0.098	4.677		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30/F8/O
net (fo=1, unplaced)	0.717	5.394		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/ram_reg_1792_2047_30_30_n_0
				mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/spo[30]_INST_0_i_2/I0
LUT6 (Prop lut6 I0 O)	(f) 0.319	5.713		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/spo[30]_INST_0_i_2/O
net (fo=1, unplaced)	0.000	5.713		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/spo[30]_INST_0_i_2_n_0
				mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/spo[30]_INST_0/I1
MUXF7 (Prop muxf7 I1 O)	(f) 0.247	5.960		mem/Imem/imem_ip/U0/synth_options.dist_mem_inst/gen_sp_ram.spram_inst/spo[30]_INST_0/O
net (fo=53, unplaced)	0.819	6.779		mem/Imem/spo[30]
				mem/Imem/p_1_out_i_434/I0
LUT3 (Prop lut3 I0 O)	(r) 0.298	7.077		mem/Imem/p_1_out_i_434/O
net (fo=3, unplaced)	0.437	7.514		mem/Imem/p_1_out_i_434_n_3
				mem/Imem/p_1_out_i_439/I2
LUT6 (Prop lut6 I2 O)	(r) 0.124	7.638		mem/Imem/p_1_out_i_439/O
net (fo=1, unplaced)	0.449	8.087		mem/Imem/p_1_out_i_439_n_3
				mem/Imem/p_1_out_i_432/I5
LUT6 (Prop lut6 I5 O)	(r) 0.124	8.211		mem/Imem/p_1_out_i_432/O
net (fo=1, unplaced)	0.449	8.660		mem/Imem/p_1_out_i_432_n_3
				mem/Imem/p_1_out_i_293/I0
LUT6 (Prop lut6 I0 O)	(r) 0.124	8.784		mem/Imem/p_1_out_i_293/O
net (fo=5, unplaced)	0.477	9.261		mem/Imem/sccpu/decoder_inst/Rsc0
				mem/Imem/p_1_out_i_431/I1
LUT2 (Prop lut2 I1 O)	(r) 0.124	9.385		mem/Imem/p_1_out_i_431/O
net (fo=256, unplaced)	0.572	9.957		sccpu/cpu_ref/Rsc[0]
				sccpu/cpu_ref/p_1_out__0_i_88/I4
LUT6 (Prop lut6 I4 O)	(r) 0.124	10.081		sccpu/cpu_ref/p_1_out__0_i_88/O
net (fo=1, unplaced)	0.000	10.081		sccpu/cpu_ref/p_1_out__0_i_88_n_3
				sccpu/cpu_ref/p_1_out__0_i_22/I1
MUXF7 (Prop muxf7 I1 O)	(r) 0.247	10.328		sccpu/cpu_ref/p_1_out__0_i_22/O
net (fo=1, unplaced)	0.735	11.063		sccpu/cpu_ref/p_1_out__0_i_22_n_3
				sccpu/cpu_ref/p_1_out__0_i_3/I1



LUT6 (Prop_lut6_I1_O)	(r) 0.298	11.361	sccpu/cpu_ref/p_1_out__0_i_3/O
net (fo=18, unplaced)	0.506	11.867	mem/Imem/div_A[30]
			mem/Imem/mux_out_reg[2][30]_i_1/I4
LUT5 (Prop_lut5_I4_O)	(r) 0.124	11.991	mem/Imem/mux_out_reg[2][30]_i_1/O
net (fo=1, unplaced)	0.000	11.991	sccpu/pc_reg_reg[31]_1[30]
			sccpu/mux_out_reg[2][30]/D
LDCE (DToQ_ldce_D_Q)	(r) 0.374	12.365	sccpu/mux_out_reg[2][30]/Q
net (fo=1, unplaced)	0.419	12.784	sccpu/cpu_ref/pc_reg_reg[31][30]
			sccpu/cpu_ref/array_reg[31][30]_i_13/I0
LUT2 (Prop_lut2_I0_O)	(r) 0.124	12.908	sccpu/cpu_ref/array_reg[31][30]_i_13/O
net (fo=14, unplaced)	0.783	13.691	sccpu/cpu_ref/array_reg_reg[0][31]_2[2]
			sccpu/cpu_ref/muxc__reg[10][0]_i_40/I2
LUT4 (Prop_lut4_I2_O)	(r) 0.124	13.815	sccpu/cpu_ref/muxc__reg[10][0]_i_40/O
net (fo=1, unplaced)	0.000	13.815	mem/Imem/pc_reg_reg[31]_4[2]
			mem/Imem/muxc__reg[10][0]_i_18/S[3]
CARRY4 (Prop_carry4_S[3]_CO[3])	(r) 0.376	14.191	mem/Imem/muxc__reg[10][0]_i_18/CO[3]
net (fo=1, unplaced)	0.748	14.939	mem/Imem/sccpu/alu_inst/data10
			mem/Imem/muxc__reg[10][0]_i_7/I4
LUT5 (Prop_lut5_I4_O)	(r) 0.124	15.063	mem/Imem/muxc__reg[10][0]_i_7/O
net (fo=2, unplaced)	0.430	15.493	mem/Imem/muxc__reg[10][0]_i_7_n_3
			mem/Imem/muxc__reg[10][0]_i_3/I3
LUT6 (Prop_lut6_I3_O)	(r) 0.124	15.617	mem/Imem/muxc__reg[10][0]_i_3/O
net (fo=5, unplaced)	0.584	16.201	mem/Imem/muxc__reg[10][0]_i_3_n_3
			mem/Imem/array_reg[31][31]_i_92/I1
LUT2 (Prop_lut2_I1_O)	(f) 0.124	16.325	mem/Imem/array_reg[31][31]_i_92/O
net (fo=1, unplaced)	0.449	16.774	mem/Imem/array_reg[31][31]_i_92_n_3
			mem/Imem/array_reg[31][31]_i_89/I0
LUT6 (Prop_lut6_I0_O)	(r) 0.124	16.898	mem/Imem/array_reg[31][31]_i_89/O
net (fo=2, unplaced)	0.460	17.358	mem/Imem/array_reg[31][31]_i_89_n_3
			mem/Imem/array_reg[31][31]_i_70/I5
LUT6 (Prop_lut6_I5_O)	(r) 0.124	17.482	mem/Imem/array_reg[31][31]_i_70/O
net (fo=33, unplaced)	0.804	18.286	mem/Imem/sccpu/muxc[3][0]
			mem/Imem/array_reg[31][31]_i_61/I5
LUT6 (Prop_lut6_I5_O)	(r) 0.124	18.410	mem/Imem/array_reg[31][31]_i_61/O
net (fo=1, unplaced)	0.449	18.859	mem/Imem/array_reg[31][31]_i_61_n_3
			mem/Imem/array_reg[31][31]_i_43/I0
LUT2 (Prop_lut2_I0_O)	(r) 0.124	18.983	mem/Imem/array_reg[31][31]_i_43/O
net (fo=99, unplaced)	0.548	19.531	sccpu/pc_inst/dmem_out_data1_retimed_reg[31]
			sccpu/pc_inst/dmem_reg_0_255_0_0_i_227/I5
LUT6 (Prop_lut6_I5_O)	(r) 0.124	19.655	sccpu/pc_inst/dmem_reg_0_255_0_0_i_227/O
net (fo=2, unplaced)	0.743	20.398	sccpu/pc_inst_n_16
			sccpu/dmem_reg_0_255_0_0_i_143/I2
			sccpu/dmem_reg_0_255_0_0_i_143/I2
LUT6 (Prop_lut6_I2_O)	(r) 0.124	20.522	sccpu/dmem_reg_0_255_0_0_i_143/O
net (fo=4, unplaced)	0.756	21.278	mem/Imem/pc_reg_reg[3]_15
			mem/Imem/dmem_reg_0_255_0_0_i_83/I1
LUT6 (Prop_lut6_I1_O)	(r) 0.124	21.402	mem/Imem/dmem_reg_0_255_0_0_i_83/O
net (fo=2, unplaced)	0.743	22.145	mem/Imem/dmem_reg_0_255_0_0_i_83_n_3
			mem/Imem/dmem_reg_0_255_0_0_i_28/I2
LUT6 (Prop_lut6_I2_O)	(f) 0.124	22.269	mem/Imem/dmem_reg_0_255_0_0_i_28/O
net (fo=3, unplaced)	0.750	23.019	mem/Imem/dmem_reg_0_255_0_0_i_28_n_3
			mem/Imem/dmem_reg_0_255_7_7_i_1/I2
LUT6 (Prop_lut6_I2_O)	(r) 0.124	23.143	mem/Imem/dmem_reg_0_255_7_7_i_1/O
net (fo=90, unplaced)	0.847	23.990	mem/Dmem/dmem_reg_0_15_0_0__30/A3
			mem/Dmem/dmem_reg_0_15_0_0__30/SP/ADR3
RAMS32 (Prop_rams32_ADR3_O)	(r) 0.106	24.096	mem/Dmem/dmem_reg_0_15_0_0__30/SP/O
net (fo=2, unplaced)	0.975	25.071	mem/Imem/array_reg_reg[27][15]_0
			mem/Imem/dmem_out_data1_retimed[31]_i_8/I3
LUT6 (Prop_lut6_I3_O)	(r) 0.124	25.195	mem/Imem/dmem_out_data1_retimed[31]_i_8/O
net (fo=1, unplaced)	0.732	25.927	mem/Imem/dmem_out_data1_retimed[31]_i_8_n_3
			mem/Imem/dmem_out_data1_retimed[31]_i_4/I4
LUT6 (Prop_lut6_I4_O)	(r) 0.124	26.051	mem/Imem/dmem_out_data1_retimed[31]_i_4/O
net (fo=16, unplaced)	0.473	26.524	mem/Imem/dmem_out_data1_retimed[31]_i_4_n_3
			mem/Imem/dmem_out_data1_retimed[16]_i_1/I1
LUT5 (Prop_lut5_I1_O)	(r) 0.124	26.648	mem/Imem/dmem_out_data1_retimed[16]_i_1/O
net (fo=1, unplaced)	0.000	26.648	mem/Dmem/D[16]
FDRE			mem/Dmem/dmem_out_data1_retimed_reg[16]/D

Arrival Time		26.648		
Destination Clock Path				
Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk_pin fall edge)	(f) 5.000	5.000		
	(f) 0.000	5.000		clk_in
net (fo=0)	0.000	5.000		clk_in
IBUF				clk_in_IBUF_inst/I
IBUF (Prop_ibuf_I_O)	(f) 0.838	5.838		clk_in_IBUF_inst/O
net (fo=1, unplaced)	0.763	6.601		clk_in_IBUF
BUFG				clk_in_IBUF_BUFG_inst/I
BUFG (Prop_bufg_I_O)	(f) 0.091	6.692		clk_in_IBUF_BUFG_inst/O
net (fo=2497, unplaced)	0.439	7.131		mem/Dmem/clk_in_IBUF_BUFG
FDRE				mem/Dmem/dmem_out_data1_retimed_reg[16]/C
clock pessimism	0.179	7.309		
clock uncertainty	-0.035	7.274		
FDRE (Setup_fdre_C_D)	0.047	7.321		mem/Dmem/dmem_out_data1_retimed_reg[16]
Required Time		7.321		

六、CPU 下板结果

1. 下板实验内容及结果

下板实验的核心内容为点亮 7 段数码管，下板时可以将 PC 寄存器或通用寄存器结果输出到 7 段数码管上，通过检测寄存器值的正确性来检验所写的 cpu 程序是否正确。编写测试文件，实例化 CPU 和七段数码管模块，并进行相关的管脚约束。由于七段数码管的显

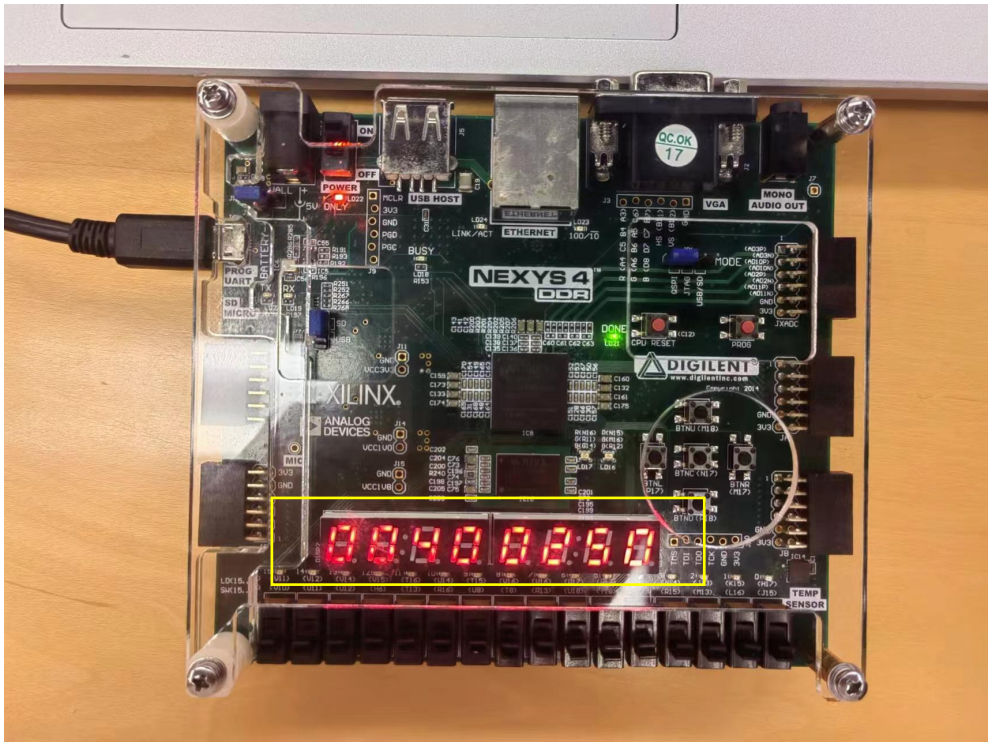
示需要使得人眼肉眼可分辨，但是 CPU 执行指令的时间转瞬即逝，因此需要对时钟进行分频。

```
`timescale 1ns / 1ps
module show(
    input clk_in,
    input reset,
    output [7:0] o_seg,
    output [7:0] o_sel
);
    wire [31:0] clk_cpu;
    wire [31:0] pc;
    sccomp_dataflow sc(
        .clk_in(clk_cpu),
        .reset(reset),
        .inst(inst),
        .pc(pc)
    );
    seg7x16 seg(
        .clk(clk_in),
        .reset(reset),
        .cs(1'b1),
        .i_data(pc),
        .o_seg(o_seg),
        .o_sel(o_sel)
    );
    Divider #(.n(100000000)) di(
        .I_CLK(clk_in),
        .rst(reset),
        .O_CLK(clk_cpu)
    );
endmodule
module Divider#(parameter n=20)(
    input I_CLK, //输入时钟信号，上升沿有效
    input rst, //同步复位信号，高电平有效
    output O_CLK //输出时钟
);
    reg [31:0]count;
    reg clk;
    assign O_CLK=clk;
        initial
        begin
        clk<=0;count<=0;
        end
        always@(posedge I_CLK)
            begin
            if(rst)
            begin
                clk<=1'b0;count<=32'b0;
            end
            else if(count==n/2-1)
            begin
                clk<=~clk;count<=32'b0;
            end
            else
                count<=count+1;
```

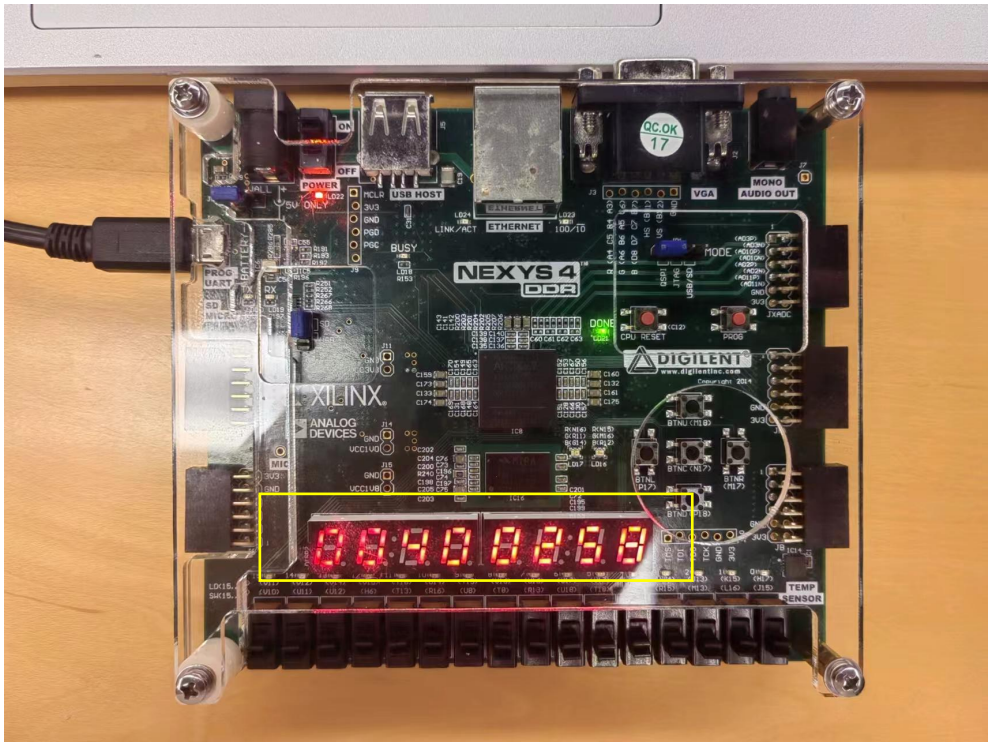
```
        end//end of always
endmodule
```

运行.bit 文件，每秒数码管上的值更新一次，显示当前 PC 程序计数器的值：

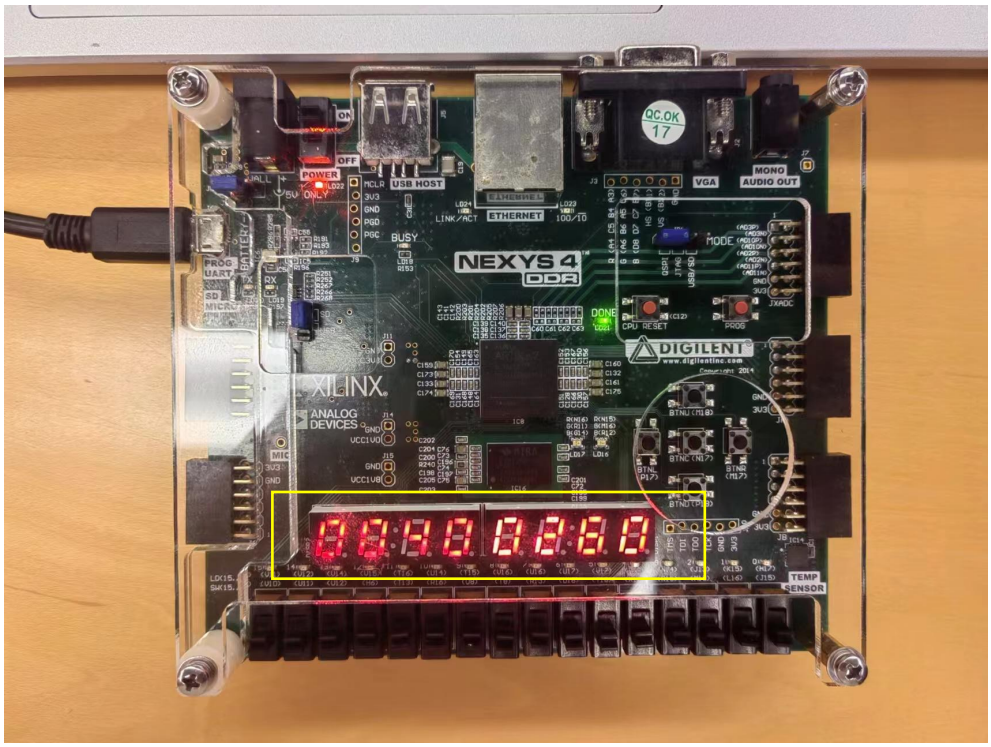
（1）PC=00400250:



（2）PC=00400258:



（3）PC=00400260：



2. 实验分析与结论

七段数码管上可以显示当前执行的程序地址（即 PC 寄存器中的值）。可以看到，显示的 PC 寄存器值从 0x00400000 开始，当执行一段时间以后，程序进入循环，会一直反复显示某几个 PC 值，这属于正常现象。

七、心得体会及建议

在本次 CPU 设计的实验中，我经历了一系列挑战和反思。尽管最终成功实现了包含 54 条 MIPS 指令的单周期 CPU，但过程中也遭遇了一些疏忽导致的错误。在 CPU 的设计过程中，我逐步实现了 IMEM、PC、ALU 、controller、CP0 等核心模块，并通过分析数据通路图，建立了各功能部件之间的逻辑联系。IMEM 和 DMEM 作为存储单元，与 CPU 紧密协作，实现了数据的读取与写入。通过顶层模块的实例化，以及 CPU 与 MEM(IMEM、DMEM)之间的协调，我成功实现了单周期 CPU 的全部功能。

通过这次实验，我对单周期 CPU 的设计有了更深刻的理解。我不仅掌握了从数据输入输出关系到指令流程再到 CPU 最终实现的全过程，还更加清晰地认识到部件之间的输入输

出关系以及数据在不同部件之间的传递方式。这些实践经验不仅加深了我对课堂知识的理解，还让我对 CPU 设计过程中的优化方向有了更明确的认识。