哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2020 春季

课程名称：计算机设计与实践

实验名称：多周期 CPU 设计

实验性质：综合设计型

实验学时：42 地点：线上

学生班级：计算机类 5 班

学生学号：180110508

学生姓名：李想

评阅教师：

报告成绩：

实验与创新实践教育中心制

2020 年 5 月

注：

本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明设计的成果和特色。报告中应该叙述设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

完成单周期 CPU 设计与实现的同学根据单周期设计的内容完成实验报告。完成多周期 CPU 设计与实现的同学根据多周期设计的内容完成实验报告。

设计的功能描述（含所有实现的指令描述及模块的功能）

一、指令描述

本次实验所要实现的 MIPS 指令共 31 条，指令格式如下：

指令类型	31…26	25…21	20…16	15…11	10…6	5…0
R 型	op	rs	rt	rd	shamt	func
I 型	op	rs	rt	immediate		
J 型	op	address				

1. R 型指令

1) add 加法指令
汇编格式：add rd, rs, rt
功能描述：rd←rs + rt; PC←NPC (PC + 4)。32 位整数加法，源操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。由于本设计无溢出检测，因此该指令功能同 ADDU。

2) addu 无符号数加法指令
汇编格式：addu rd, rs, rt
功能描述：rd←rs + rt; PC←NPC (PC + 4)。32 位无符号整数加法，源操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。

3) sub 减法指令
汇编格式：sub rd, rs, rt
功能描述：rd←rs – rt; PC←NPC(PC+4)。32 位整数减法，源操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。

4) subu 无符号数减法指令
汇编格式：subu rd, rs, rt
功能描述：rd←rs – rt; PC←NPC(PC+4)。32 位无符号整数减法，源操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。

5) and 逻辑与
汇编格式：and rd, rs, rt
功能描述：rd←rs and rt; PC←NPC(PC+4)。32 位数按位逻辑与，源操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。

6) or 逻辑或
汇编格式：or rd, rs, rt
功能描述：rd←rs or rt; PC←NPC(PC+4)。32 位数按位逻辑或，源操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。

7) xor 逻辑异或
汇编格式：xor rd, rs, rt
功能描述：rd←rs xor rt; PC←NPC(PC+4)。32 位数按位逻辑异或，源操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。

8) nor 逻辑或非
汇编格式：nor rd, rs, rt
功能描述：rd←rs nor rt; PC←NPC(PC+4)。32 位数按位逻辑或非，源

操作数分别在 rs, rt 两个通用寄存器中，结果放在 rd 寄存器。

9) slt 小于则设置指令

汇编格式：slt rd, rs, rt

功能描述：if ((rs)<(rt)) then (rd)←1; else (rd)←0; PC ← NPC (PC + 4)。如果 rs 的值小于 rt 的值，则设置 rd 为 1，否则 rd 为 0。

10) sltu 无符号小于则设置指令

汇编格式：sltu rd, rs, rt

功能描述：if ((rs)<(rt)) then (rd)←1; else (rd)←0; PC ← NPC (PC + 4)。无符号数小于判断，如果 rs 的值小于 rt 的值，则设置 rd 为 1，否则 rd 为 0。

11) sll 逻辑左移

汇编格式：sll rd, rt, shamt

功能描述：(rd)←(rt)<<shamt; PC ← NPC (PC + 4)。逻辑左移，将 rt 寄存器中的 32 位数逻辑左移后赋给 rd，低位用 0 填充，移位的位数是 shamt。

12) srl 逻辑右移

汇编格式：srl rd, rt, shamt

功能描述：(rd)←(rt)>>shamt; PC ← NPC (PC + 4)。逻辑右移，将 rt 寄存器中的 32 位数逻辑右移后赋给 rd，移位的位数是 shamt。

13) sra 算术右移

汇编格式：sra rd, rt, shamt

功能描述：(rd)←(rt)>>shamt; PC ← NPC (PC + 4)。算术右移，将 rt 寄存器中的 32 位数算术右移后赋给 rd，移位的位数是 shamt。算术右移时，符号位不仅要参与移位，还要保留。

14) sllv 按寄存器值逻辑左移指令

汇编格式：sllv rd, rt, rs

功能描述：(rd)←(rt)<<(rs); PC ← NPC (PC + 4)。按寄存器值逻辑左移指令，将 rt 寄存器中的 32 位数逻辑左移后赋给 rd，低位用 0 填充，移位的位数在 rs 寄存器中。

15) srlv 按寄存器值逻辑右移指令

汇编格式：srlv rd, rt, rs

功能描述：(rd)←(rt)>>(rs); PC ← NPC (PC + 4)。按寄存器值逻辑右移指令，将 rt 寄存器中的 32 位数逻辑右移后赋给 rd，移位的位数在 rs 寄存器中。

16) srav 按寄存器值算术右移指令

汇编格式：srav rd, rt, rs

功能描述：(rd)←(rt)>>(rs); PC ← NPC (PC + 4)。按寄存器值算术右移指令，将 rt 寄存器中的 32 位数算术右移后赋给 rd，移位的位数在 rs 寄存器中。算术右移时，符号位不仅要参与移位，还要保留。

17) jr 按寄存器内容转移指令

汇编格式：jr rs

功能描述：(PC)←(rs); PC ← NPC (PC + 4)。将 rs 寄存器的内容当地址，赋给 PC，从而完成转移，通常作为过程返回语句。

2. I 型指令

1) addi 有符号立即数加法指令
汇编格式：addi rt, rs, immediate
功能描述：(rt)←(rs) + (Sign-Extend)immediate; PC ← NPC (PC + 4)。首先将 16 位有符号立即数扩展到 32 位，然后加上 rs 中的数，结果 rt 寄存器。如果结果溢出会产生内部异常中断，本设计不支持异常处理。

2) addiu 无符号立即数加法指令
汇编格式：addiu rt, rs, immediate
功能描述：(rt)←(rs) + (Sign-Extend)immediate; PC ← NPC (PC + 4)。首先将 16 位有符号立即数扩展到 32 位，然后加上 rs 中的数，结果给 rt 寄存器。与 ADDI 的不同是不会因溢出而产生内部异常中断，本设计未实现内部异常处理，因此该指令与 ADDI 相同。

3) andi 立即数逻辑与指令
汇编格式：andi rt, rs, immediate
功能描述：(rt)←(rs) AND (Zero-Extend)immediate; PC ← NPC (PC + 4)。首先将 16 位立即数零扩展到 32 位，然后同 rs 中的数按位逻辑与，结果给 rt 寄存器。

4) ori 立即数逻辑或指令
汇编格式：ori rt, rs, immediate
功能描述：(rt)←(rs) ORI (Zero-Extend)immediate; PC ← NPC (PC + 4)。首先将 16 位立即数零扩展到 32 位，然后同 rs 中的数按位逻辑或，结果给 rt 寄存 器。

5) xori 立即数逻辑异或指令
汇编格式：xori rt, rs, immediate
功能描述：(rt)←(rs) XORI (Zero-Extend)immediate; PC ← NPC (PC + 4)。首先将 16 位立即数零扩展到 32 位，然后同 rs 中的数按位逻辑异或，结果给 rt 寄存器。

6) sltiu 小于无符号立即数则设置指令
汇编格式：sltiu rt, rs, immediate
功能描述：if ((rs)<(sign_extend)immediate) then (rt)←1; else (rt)←0; PC ← NPC (PC + 4)。如果 rs 的值小于立即数 immediate 值（进行的是符号扩展），则设置 rt 为 1，否则 rt 为 0。

7) lui 立即数赋值指令
汇编格式：lui rt, immediate
功能描述：(rt)←immediate<<16 & 0FFFF0000H 即 (rt)←immediate×65536; PC ← NPC (PC + 4)。首先 16 位立即数赋给 rt 寄存器的 16 位，低 16 位用填充。也就是将 16 位立即数乘以 65536 后赋值 rt 寄存器。

8) lw 存储器读（字操作）
汇编格式：lw rt, offset(rs)
功能描述：(rt)←Memory[(rs)+(sign_extend)offset]; PC ← NPC (PC + 4)。以 rs 寄存器的内容为基地址，offset 通过符号扩展后形成 32 位的偏移，将基地址加上偏移形成一个 32 位的地址，以此地址从 RAM

中读出一个字（4 字节）赋给 rt 寄存器。

9) sw 存储器写（字操作）
   汇编格式：sw rt, offset(rs)
   功能描述：Memory[(rs)+(sign_extend)offset]←(rt); PC ← NPC (PC + 4)。以 rs 寄存器的内容为基地址，offset 通过符号扩展后形成 32 位的偏移，将基地址 加上偏移形成一个 32 位的地址，将 rt 寄存器的内容写入到 RA 中该地址开始的一个字（4 字节）单元。
10) beq 相等则转移指令
   汇编格式：beq rt, rs, immediate
   功能描述：if ((rt)=(rs)) then (PC)←(PC)+4+((Sign-Extend) offset<<2); else PC ← NPC (PC + 4)。如果 rt 和 rs 的值相等，则转移到新的地址。新地址是当前指令的下一条指令地址（PC+4）加上一个 32 位偏移量。该 32 位偏移量是将 16 位 offset 符号扩展到 32 位，然后左移 2 位（即乘 4）后取低 32 位得到。
11) bne 不相等则转移指令
   汇编格式：bne rt, rs, immediate
   功能描述：if ((rt)≠(rs)) then (PC)←(PC)+4+((Sign-Extend) offset<<2); else PC ← NPC (PC + 4)。如果 rt 和 rs 的值不等，则转移到新的地址。新地址是当前指令的下一条指令地址（PC+4）加上一个 32 位偏移量。该 32 位偏移量是将 16 位 offset 符号扩展到 32 位，然后左移 2 位（即乘 4）后取低 32 位得到。
12) bgtz 大于 0 则转移指令
   汇编格式：bgtz rs, offset
   功能描述：if ((rs)>0) then (PC)←(PC)+4+((Sign-Extend) offset<<2), rs=$1; else PC ← NPC (PC + 4)。如果 rs 的值大于 0 则跳转到指定分支。

3. J 型指令

1) j 无条件转移指令
   汇编格式：j target
   功能描述：(PC)←((Zero-Extend) address<<2); PC ← NPC (PC + 4)。无条件转移到新的地址。新地址是 26 位 address 零扩展到 32 位，然后左移 2 位（即 4）后取低 32 位得到。实际系统中只用了低 16 位地址线。
2) jal 过程调用指令
   汇编格式：JAL target
   功能描述：($31)←(PC)+4; (PC)←((Zero-Extend)address<<2); PC ← NPC(PC+4)。先将下条指令的地址((PC)+4)保存在$31（$ra）作为过程的返回地址，然后无条件转移到新的地址。新地址是 26 位 address 零扩展到 32 位，然后左移 2 位（即乘 4）后取低 32 位得到。实际系统中只用了低 16 位地址线。

二、模块功能

本设计系统共有六个模块，分别是顶层模块、控制器模块、取指模块、译码模块、执行模块和存储器模块。各个模块的功能如下：

顶层模块：实例化所有模块，用于连接各个部件并传递相关信号，输入时钟信号和复位信号，输出 debug 信号用于调试系统。

控制器模块：根据输入的指令的 opcode 和 func 段生成指令变量；并通过指令变量、时钟周期状态和输入信号（Zero、Pos）在特定的时钟周期下生成相应的控制信号；通过指令变量生成次态组合逻辑控制状态机的状态转移。

取指模块：根据 NPCOp 控制并通过其他输入信号计算 NPC，在一定时钟周期 PCWr 写使能信号控制下修改 PC 寄存器；分配指令寄存器 ROM 并根据 PC 值读出相应的指令，在 IRWr 写使能信号下修改 IR 寄存器，并输出。

译码模块：在一定时钟周期和信号控制下从寄存器组中根据读取地址读取数据，或者根据写入选择信号和写使能信号向寄存器组写入数据；此外，扩展单元在一定时钟周期下根据不同的功能选择信号计算并输出扩展后的值。

执行模块：在一定时钟周期和 ALUOp 信号的控制下，根据输入的值完成指定的运算，并输出计算结果 ALUOut，以及部分指令所需的 Zero 和 Pos 信号。

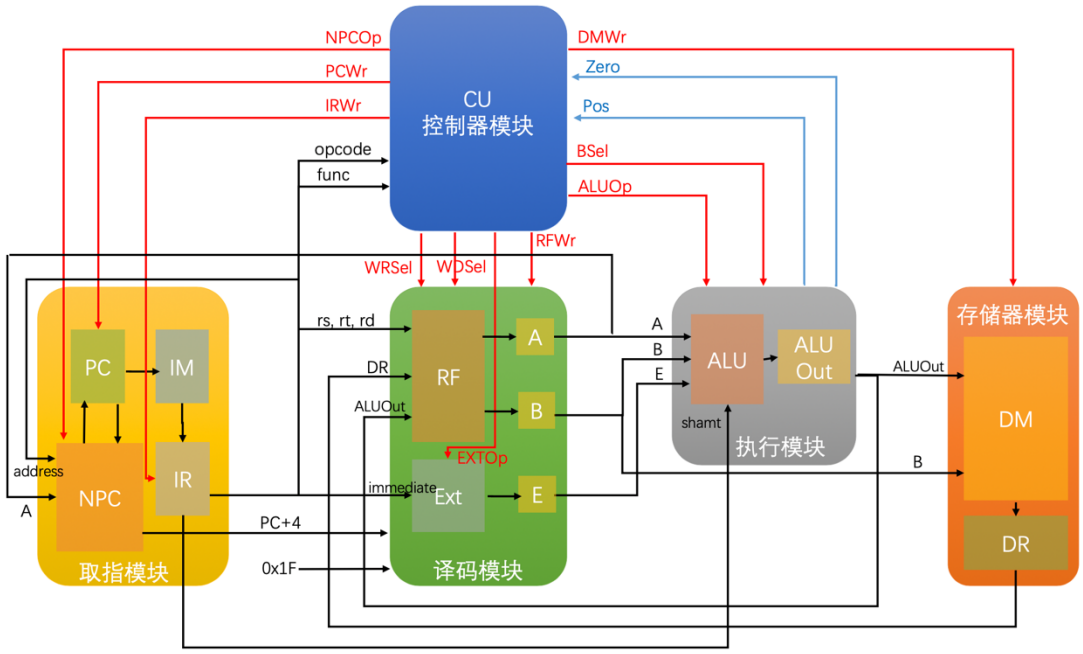
存储器模块：分配数据存储器 RAM，并在一定时钟周期和信号控制下根据地址读出/写入数据。

设计的主要特色

本设计一共实现了 31 条 MIPS 指令，整个系统被划分为了六个功能模块，顶层模块、控制器模块、取指模块、译码模块、执行模块和存储器模块，每个模块中的各功能部件明确，各模块之间的连线明了，思路清晰。采用了大量的宏定义，用于指令描述和运算单元功能选择描述，代码可读性强，便于维护。在保证仿真答案正确的前提下，系统主频最高可以达到 115MHz。控制器模块的控制信号赋值采用与门和或门的组合逻辑，代码简洁易懂。部分指令的运算功能采用相同的 ALUOp，使得连线简单，起到一定的优化作用。

设计的体系结构

（包括体系结构框图和对结构图的简要解释）



控制器模块：从 IR 寄存器输入 opcode 和 func 段（黑色线），从执行模块输入状态信号 Zero 和 Pos（蓝色线）；输出（红色线）NPCOp、PCWr、IRWr 控制信号到取指模块，WRSel、WDSel、RFWr、EXTOp 控制信号到译码模块，ALUOp、BSel 控制信号到执行模块，DMWr 控制信号到存储器模块。

取指模块：输入：控制信号，来自译码模块的 A；输出 IR、NPC.PC4 到译码模块，IR 的 shamt 段到执行模块。

译码模块：输入：控制信号，来自取指模块的 IR 的 rs, rt, rd, immediate 段以及 NPC.PC4，来自执行模块的 ALUOut，来自存储器模块的 DR；输出：RF.A 到执行模块和取指模块，RF.B 到执行模块和存储器模块，Ext.E 到执行模块。

执行模块：输入：控制信号，来自译码模块的 RF.A，RF.B，Ext.E，来自取指模块的 IR 的 shamt 段；输出：状态信号 Zero，Pos 到控制器模块，ALUOut 到存储器模块和译码模块。

存储器模块：输入：控制信号，来自执行模块的 ALUOut，来自译码模块的 RF.B；输出：DR 到译码模块。

数据通路设计

（包含数据通路主要部件说明及数据通路表）

一、 主要部件说明

1. 取指单元
   1) PC 指令地址寄存器，存储指令的地址。
   2) NPC 下指令地址计算部件，根据输入信号计算下一条指令的地址。
   3) IM 指令存储器，存储所有指令，并根据指令地址取出相应指令到 IR。
   4) IR 指令寄存器，寄存来自 IM 的指令。
2. 译码单元
   1) RF 寄存器堆，存储 32 个通用寄存器的值，并对其进行读出和写入。
   2) A 第一个读出值的寄存器，寄存来自第一个地址读出的对应的寄存器的值。
   3) B 第二个读出值的寄存器，寄存来自第二个地址读出的对应的寄存器的值。
   4) Ext 扩展单元，对输入信号进行零扩展或者符号扩展到 32 位。
   5) E 扩展单元计算输出结果寄存器。
3. 执行单元
   1) ALU 运算器单元，根据控制信号执行相应逻辑运算并输出计算结果和标志信号
   2) ALUOut 计算结果寄存器。
4. 数据存储器
   1) DM 数据存储器，读取和写入相应地址的数据。
   2) DR 从数据存储器读出的数据寄存器。

二、 数据通路表

	所属单元	取指单元					
	部件	PC	NPC			（指令存储器	IR
	输入信号	DI	PC	Imm	RA（寄存器）	A	
R型指令	add	NPC.NPC	PC.DO			PC.DO	IM
	addu	NPC.NPC	PC.DO			PC.DO	IM
	sub	NPC.NPC	PC.DO			PC.DO	IM
	subu	NPC.NPC	PC.DO			PC.DO	IM
	and	NPC.NPC	PC.DO			PC.DO	IM
	or	NPC.NPC	PC.DO			PC.DO	IM
	xor	NPC.NPC	PC.DO			PC.DO	IM
	nor	NPC.NPC	PC.DO			PC.DO	IM
	slt	NPC.NPC	PC.DO			PC.DO	IM
	sltu	NPC.NPC	PC.DO			PC.DO	IM
	sll	NPC.NPC	PC.DO			PC.DO	IM
	srl	NPC.NPC	PC.DO			PC.DO	IM
	sra	NPC.NPC	PC.DO			PC.DO	IM
	sllv	NPC.NPC	PC.DO			PC.DO	IM
	srlv	NPC.NPC	PC.DO			PC.DO	IM
	srav	NPC.NPC	PC.DO			PC.DO	IM
	jr	NPC.NPC	PC.DO		A	PC.DO	IM
I型指令	addi	NPC.NPC	PC.DO			PC.DO	IM
	addiu	NPC.NPC	PC.DO			PC.DO	IM
	andi	NPC.NPC	PC.DO			PC.DO	IM
	ori	NPC.NPC	PC.DO			PC.DO	IM
	xori	NPC.NPC	PC.DO			PC.DO	IM
	sltiu	NPC.NPC	PC.DO			PC.DO	IM
	lui	NPC.NPC	PC.DO			PC.DO	IM
	lw	NPC.NPC	PC.DO			PC.DO	IM
	sw	NPC.NPC	PC.DO			PC.DO	IM
	beq	NPC.NPC	PC.DO	IR[15:0]		PC.DO	IM
	bne	NPC.NPC	PC.DO	IR[15:0]		PC.DO	IM
	bgtz	NPC.NPC	PC.DO	IR[15:0]		PC.DO	IM
J型指令	j	NPC.NPC	PC.DO	IR[25:0]		PC.DO	IM
	jal	NPC.NPC	PC.DO	IR[25:0]		PC.DO	IM
综合		NPC.NPC	PC.DO	IR[25:0]	A	PC.DO	IM

	所属单元	译码单元							
	部件	RF（寄存器堆）				A	B	S_EXT	E
	输入信号	A1（读出1）	A2（读出2）	A3（写入）	WD（写回值）				
R型指令	add	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	addu	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	sub	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	subu	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	and	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	or	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	xor	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	nor	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	slt	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	sltu	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	sll		IR[20:16]	IR[15:11]	ALUOut		RF.RD2		
	srl		IR[20:16]	IR[15:11]	ALUOut		RF.RD2		
	sra		IR[20:16]	IR[15:11]	ALUOut		RF.RD2		
	sllv	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	srlv	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	srav	IR[25:21]	IR[20:16]	IR[15:11]	ALUOut	RF.RD1	RF.RD2		
	jr	IR[25:21]				RF.RD1			
I型指令	addi	IR[25:21]		IR[20:16]	ALUOut	RF.RD1		IR[15:0]	EXT.Ext
	addiu	IR[25:21]		IR[20:16]	ALUOut	RF.RD1		IR[15:0]	EXT.Ext
	andi	IR[25:21]		IR[20:16]	ALUOut	RF.RD1		IR[15:0]	EXT.Ext
	ori	IR[25:21]		IR[20:16]	ALUOut	RF.RD1		IR[15:0]	EXT.Ext
	xori	IR[25:21]		IR[20:16]	ALUOut	RF.RD1		IR[15:0]	EXT.Ext
	sltiu	IR[25:21]		IR[20:16]	ALUOut	RF.RD1		IR[15:0]	EXT.Ext
	lui			IR[20:16]	ALUOut			IR[15:0]	EXT.Ext
	lw	IR[25:21]		IR[20:16]	DR	RF.RD1		IR[15:0]	EXT.Ext
	sw	IR[25:21]	IR[20:16]			RF.RD1	RF.RD2	IR[15:0]	EXT.Ext
	beq	IR[25:21]	IR[20:16]			RF.RD1	RF.RD2		
	bne	IR[25:21]	IR[20:16]			RF.RD1	RF.RD2		
	bgtz	IR[25:21]				RF.RD1			
J型指令	j								
	jal			0x1F	PC				
综合		IR[25:21]	IR[20:16]	IR[15:11] IR[20:16] 0x1F	ALUOut DR PC	RF.RD1	RF.RD2	IR[15:0]	EXT.Ext

	所属单元	执行单元				数据存储器		
	部件	ALU			ALUOut	DM（数据存储器）		DR
	输入信号	A	B	C		A	WD	
R型指令	add	A	B		ALU.C			
	addu	A	B		ALU.C			
	sub	A	B		ALU.C			
	subu	A	B		ALU.C			
	and	A	B		ALU.C			
	or	A	B		ALU.C			
	xor	A	B		ALU.C			
	nor	A	B		ALU.C			
	slt	A	B		ALU.C			
	sltu	A	B		ALU.C			
	sll		B	IR[10:6]	ALU.C			
	srl		B	IR[10:6]	ALU.C			
	sra		B	IR[10:6]	ALU.C			
	sllv	A	B		ALU.C			
	srlv	A	B		ALU.C			
	srav	A	B		ALU.C			
	jr							
I型指令	addi	A	E		ALU.C			
	addiu	A	E		ALU.C			
	andi	A	E		ALU.C			
	ori	A	E		ALU.C			
	xori	A	E		ALU.C			
	sltiu	A	E		ALU.C			
	lui		E		ALU.C			
	lw	A	E		ALU.C	ALUOut		DM.RD
	sw	A	E		ALU.C	ALUOut	B	
	beq	A	B		s			
	bne	A	B					
	bgtz	A						
J型指令	j							
	jal							
综合		A	B E	IR[10:6]	ALU.C	ALUOut	B	DM.RD

各部件的设计与实现

（含模块功能，输入、输出信号及变量定义，关键代码等。控制器设计包含控制信号表）

一、 顶层模块

1. 模块功能：实例化所有模块，用于连接各个部件并传递相关信号，输入时钟信号和复位信号，输出 debug 信号用于调试系统。
2. 输入输出信号：

信号类型	信号	位宽	信号描述
Input	rst	1	复位信号
	clk	1	时钟信号
Output	debug_wb_pc	32	PC 值的 debug 信号
	debug_wb_rf_wen	1	RFWr 的 debug 信号
	debug_wb_rf_wnum	32	寄存器组 A3 写入地址的 debug 信号
	debug_wb_rf_wdata	32	寄存器组 WD 写入数据的 debug 信号

3. 变量定义：

变量	类型	位宽	描述
clock	wire	1	变频后的时钟信号
IR	wire	32	指令寄存器连线
DM_RD	wire	32	数据存储器读出数据连线
ALUOut	wire	32	ALUOut 寄存器连线
RF1	wire	32	寄存器组读出寄存器 A 连线
RF2	wire	32	寄存器组读出寄存器 B 连线
Ext	wire	32	扩展单元结果 Ext 连线
PC_4	wire	16	PC+4 结果
ALUOp	wire	5	ALU 功能选择信号连线
NPCOp	wire	2	NPC 功能选择信号连线
WRSel	wire	2	寄存器组写入地址选择信号连线
WDSel	wire	2	寄存器组写入数据选择信号连线
Zero	wire	1	ALU 计算结果零标志信号连线
Pos	wire	1	BGTZ 标志信号连线
RFWr	wire	1	寄存器组写使能控制信号连线
EXTOp	wire	1	扩展单元功能选择信号连线
DMWr	wire	1	数据存储器写使能控制信号连线
Bsel	wire	1	ALU 输入端口 B 选择信号连线
PCWr	wire	1	PC 寄存器写使能信号连线
IRWr	wire	1	IR 寄存器写使能信号连线

4. 关键代码：

```
assign debug_wb_rf_wen = RFWr;

    cpuclk cc(.clk_in1(clk), .clk_out1(clock));
```

```
control32 cu(
//input
.clk(clock), .rst(rst), .opcode(IR[31:26]),
.func(IR[5:0]), .Zero(Zero), .Pos(Pos),
//output
 .NPCOp(NPCOp), .PCWr(PCWr), .IRWr(IRWr), .RFWr(RFWr),
.EXTOp(EXTOp), .ALUOp(ALUOp), .DMWr(DMWr),
.MRFA3Sel(WRSel), .MRFWDSel(WDSel), .MALUBsel(Bsel)
);

dmemory32 dm(
.DM_RD(DM_RD), //output
.address(ALUOut), .write_data(RF2), .Memwrite(DMWr),
.clock(clock)//input
);

executs32 exe(
//input
.clk(clock), .rst(rst), .RF1(RF1), .RF2(RF2), .Ext(Ext),
.shamt(IR[10:6]), .ALUOp(ALUOp), .Bsel(Bsel),
//output
.Zero(Zero), .Pos(Pos), .ALUOut(ALUOut)
);

idecode32 dec(
//input
.rst(rst), .clock(clock), .WRSel(WRSel), .WDSel(WDSel),
.RFWr(RFWr), .EXTOp(EXTOp), .IM_D(IR),
.ALUOut(ALUOut), .DM_RD(DM_RD), .PC_4(PC_4),
//output
.RD1(RF1), .RD2(RF2), .Ext(Ext),
//debug
.debug_WD(debug_wb_rf_wdata), .debug_A3(debug_wb_rf_wnum)
);

ifetc32 ifetc(
      //input
    .clock(clock), .rst(rst), .NPCOp(NPCOp), .PCWr(PCWr),
    .IRWr(IRWr), .RF1(RF1),
     //output
    .PC_4(PC_4), .IR(IR),
     //debug
    .PC_debug(debug_wb_pc)
```

```
);
```

二、 控制器模块

1. 模块功能：

根据输入的指令的 opcode 和 func 段生成指令变量；并通过指令变量、时钟周期和输入信号（Zero、Pos）生成控制信号；通过指令变量生成次态组合逻辑控制状态机的状态转移。

2. 输入、输出信号：

信号类型	信号	位宽	信号描述
Input	clk	1	时钟信号
	rst	1	复位信号
	opcode	6	指令的 IR[31:26]段
	func	6	指令的 IR[5:0]段
	Zero	1	ALU 的输出信号，若 ALUOut = 0 则 Zero = 1；否则为 0
	Pos	1	ALU 的输出信号，用于 BGTZ 指令，若(rs)>0，Pos = 1；否则为 0
Output	NPCOp	2	NPC 功能选择信号
	PCWr	1	PC 写使能信号
	IRWr	1	IR 写使能信号
	RFWr	1	RF 写使能信号
	EXTOp	1	扩展单元功能选择信号
	ALUOp	5	ALU 功能选择信号
	DMWr	1	DM 写使能信号
	MRFA3Sel	2	RF 写入地址 A3 端多路选择器信号
	MRFWDSel	2	RF 写入数据 WD 端多路选择器信号
	MALUBSel	1	ALU 输入 B 端多路选择器信号

3. 变量定义：

变量	类型	位宽	描述
current_state	reg	3	状态机现态寄存器
next_state	reg	3	状态机次态寄存器
Rtype	wire	1	判断指令是否为 R 型
T1,T2,T3,T4,T5	wire	1	分别表示位于第几个状态（时钟周期）
add,addu,sub, subu,and_r, or_r, xor_r, nor_r, slt, sltu, sll, srl, sra, sllv, srlv, srav, jr,	wire	1	分别表示 31 条指令变量

addi, addiu, andi, ori, xori, sltiu, lui, lw, sw, beq, bne, bgtz, j, jal			

4. 关键代码：

```
//控制信号生成
assign NPCOp[0] = (beq & Zero & T3) | (bne & ~Zero & T3) |
                  (bgtz & Pos & T3) | (j & T2) | (jal & T2);
assign NPCOp[1] = (T3 & jr) | (T2 & j) | (T2 & jal);
assign PCWr = T1 | (( j | jal) & T2)| (jr & T3) | (beq & Zero & T3)
                | (bne & ~Zero & T3) | (bgtz & Pos & T3);
assign IRWr = T1;
assign RFWr = (rst==1)? 0 : ((add|addu|sub|subu|and_r |or_r|xor_r|nor_r|
                slt|sltu|sll|srl|sra|sllv|srlv|srav|addi|addiu|andi|ori|xori|sltiu|lui) & T4)
                |(lw & T5)|(jal & T2);
assign EXTOp = (addi|addiu|sltiu|lw|sw) & T2;
assign ALUOp[0] = T3 & (add|addu|addi|addiu|lw|sw|sub|subu|
                        or_r|nor_r|sltu|srl|srlv|ori|sltiu|lui|beq|bne);
assign ALUOp[1] = T3 & (and_r|or_r|slt|sltu|sra|sllv|srlv|andi|ori|sltiu|lui);
assign ALUOp[2] = T3 & (subu|xor_r|nor_r|slt|sltu|sllv|srlv|xori|sltiu|bgtz);
assign ALUOp[3] = T3 & (subu|sll|srl|sra|sllv|srlv|lui|bgtz);
assign ALUOp[4] = T3 & (add|addu|addi|addiu|lw|sw|srav);
assign DMWr = T4 & sw;
assign MRFA3Sel[0] = T4 & (add|addu|sub|subu|and_r|or_r|
                           xor_r|nor_r|slt|sltu|sll|srl|sra|sllv|srlv|srav);
assign MRFA3Sel[1] = T2 & jal;
assign MRFWDSel[0] = T5 & lw;
assign MRFWDSel[1] = T2 & jal;
assign MALUBsel = T3 & (addi|addiu|andi|ori|xori|sltiu|lui|lw|sw);
//次态组合逻辑
always @(*)
    begin
        case (current_state)
            S1: begin
                next_state <= S2;
            end
            S2: begin
                if (j | jal)
                    next_state <= S1;
                else next_state <= S3;
            end
            S3: begin
```

```
                    if (beq | bne | bgtz | jr)
                        next_state <= S1;
                    else next_state <= S4;
                end
            S4: begin
                    if (lw)
                        next_state <= S5;
                    else next_state <= S1;
                end
            S5: begin
                    next_state <= S1;
                end
            default:next_state <= next_state;
        endcase
    end
```

5. 控制信号表：

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
add	T1:+4	T1:1	T1:1	T4:1		T3:ADD		T4:RD	T4:AR	T3:B
addu	T1:+4	T1:1	T1:1	T4:1		T3:ADD		T4:RD	T4:AR	T3:B
sub	T1:+4	T1:1	T1:1	T4:1		T3:SUB		T4:RD	T4:AR	T3:B
subu	T1:+4	T1:1	T1:1	T4:1		T3:SUBU		T4:RD	T4:AR	T3:B
and	T1:+4	T1:1	T1:1	T4:1		T3:AND		T4:RD	T4:AR	T3:B
or	T1:+4	T1:1	T1:1	T4:1		T3:OR		T4:RD	T4:AR	T3:B
xor	T1:+4	T1:1	T1:1	T4:1		T3:XOR		T4:RD	T4:AR	T3:B
nor	T1:+4	T1:1	T1:1	T4:1		T3:NOR		T4:RD	T4:AR	T3:B
slt	T1:+4	T1:1	T1:1	T4:1		T3:SLT		T4:RD	T4:AR	T3:B
sltu	T1:+4	T1:1	T1:1	T4:1		T3:SLTU		T4:RD	T4:AR	T3:B
sll	T1:+4	T1:1	T1:1	T4:1		T3:SLL		T4:RD	T4:AR	T3:B
srl	T1:+4	T1:1	T1:1	T4:1		T3:SRL		T4:RD	T4:AR	T3:B
sra	T1:+4	T1:1	T1:1	T4:1		T3:SRA		T4:RD	T4:AR	T3:B
sllv	T1:+4	T1:1	T1:1	T4:1		T3:SLLV		T4:RD	T4:AR	T3:B
srlv	T1:+4	T1:1	T1:1	T4:1		T3:SRLV		T4:RD	T4:AR	T3:B
srav	T1:+4	T1:1	T1:1	T4:1		T3:SRAV		T4:RD	T4:AR	T3:B
jr	T1:+4 T3:JRPC	T1:1 T3:1	T1:1							
addi	T1:+4	T1:1	T1:1	T4:1	T2:SE	T3:ADD		T4:RT	T4:AR	T3:E32
addiu	T1:+4	T1:1	T1:1	T4:1	T2:SE	T3:ADD		T4:RT	T4:AR	T3:E32
andi	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:AND		T4:RT	T4:AR	T3:E32
ori	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:OR		T4:RT	T4:AR	T3:E32
xori	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:XOR		T4:RT	T4:AR	T3:E32
sltiu	T1:+4	T1:1	T1:1	T4:1	T2:SE	T3:SLTU		T4:RT	T4:AR	T3:E32
lui	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:LUI		T4:RT	T4:AR	T3:E32
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		T5:RT	T5:DR	T3:E32
sw	T1:+4	T1:1	T1:1		T2:SE	T3:ADD	T4:1			T3:E32
beq	T1:+4 T3:BNPC	T1:1 T3:Zero	T1:1			T3:SUB				T3:B
bne	T1:+4 T3:BNPC	T1:1 T3:~Zero	T1:1			T3:SUB				T3:B
bgtz	T1:+4 T3:BNPC	T1:1 T3:Pos	T1:1			T3:BGTZ				
j	T1:+4 T2:JNPC	T1:1 T2:1	T1:1							
jal	T1:+4 T2:JNPC	T1:1 T2:1	T1:1	T2:1				T2:+31	T2:PC4	

其中，NPCOp=00 对应+4，01 对应 BNPC，10 对应 JRPC，11 对应 JNPC；
EXTOp=1 对应 UE，1 对应 SE；
MRFA3Sel=00 对应 RT，01 对应 RD，10 对应+31；
MRFWDSel=00 对应 AR，01 对应 DR，10 对应 PC4；
MALUBSel=0 对应 B，1 对应 E32；
ALUOp=ADD 对应 5'b10001，SUB 对应 5'b00001， AND 对应 5'b00010，OR 对应 5'b00011，XOR 对应 5'b00100，NOR 对应 5'b00101， SLT 对应 5'b00110，SLTU 对应 5'b00111，SLL 对应 5'b01000，SRL 对应 5'b01001，SRA 对应 5'b01010，LUI 对应 5'b01011，BGTZ 对应 5'b01100，SUBU 对应 5'b01101，SLLV 对应 5'b01110，RLV 对应 5'b01111，SRAV 对应 5'b10000

三、 取指模块

1. 模块功能：
   根据 NPCOp 控制并通过其他输入信号计算 NPC，在 PCWr 写使能信号控制下修改 PC 寄存器；分配指令寄存器 ROM 并根据 PC 值读出相应的指令，在 IRWr 写使能信号下修改 IR 寄存器，并输出。
2. 输入输出信号：

信号类型	信号	位宽	信号描述
Input	clock	1	时钟信号
	rst	1	复位信号
	NPCOp	2	写入地址端多路选择器信号
	PCWr	1	写入数据端多路选择器信号
	IRWr	1	寄存器组写使能信号
	RF1	32	寄存器组读出数据 RF.RD1
Output	PC_4	16	PC+4 输出（时序逻辑）
	IR	32	指令寄存器
	PC_debug	32	PC 寄存器的 debug 信号

3. 变量定义：

变量	类型	位宽	描述
PC,PC_pre	reg	16	分别表示 PC 寄存器；存储 PC 的上一次取值
NPC	reg	16	保存计算出的下一个 PC 值
offset	wire	16	指令 IR[15:0]，offset 段
address	wire	26	指令 IR[25:0]，address 段
offset_ext	wire	16	offset 的高 16 位符号扩展值
address_ext	wire	6	address 的高 6 位零扩展
NPC_PC_4	wire	16	PC+4 值（组合逻辑）
Instruction	wire	32	IP 核指令寄存器读出的指令
flag	wire	1	PC 延迟一拍标志

4. 关键代码：

```
always @(*)
        begin
            case(NPCOp)
                2'b00: NPC = NPC_PC_4;
                2'b01: NPC = PC_pre + 4 + (({offset_ext, offset})<<2);
                2'b10: NPC = RF1;
                2'b11: NPC = ({address_ext, address})<<2;
               default: NPC = NPC;
            endcase
        end
always @(negedge clock)
        begin
            PC_4 <= PC + 4;
            if (rst == 1)
                begin
                    PC <= 15'd0;
                    flag = 0;
                end
             else if (flag == 0 && reset == 0)
                flag = 1;
            else if (PCWr == 1)
                begin
                    PC <= NPC;
                    PC_pre <= PC;
                end
            else PC <= PC;
        end
```

四、 译码模块

1. 模块功能：

从寄存器组读取数据，或者根据写入选择信号和写使能信号向寄存器组写入数据；扩展单元根据不同的功能选择信号计算并输出扩展后的值。

2. 输入输出信号：

信号类型	信号	位宽	信号描述
Input	clock	1	时钟信号
	rst	1	复位信号
	WRSel	2	写入地址端多路选择器信号
	WDSel	2	写入数据端多路选择器信号
	RFWr	1	寄存器组写使能信号
	EXTOp	1	扩展单元功能选择信号
	IM_D	32	指令 IR
	ALUOut	32	ALU 输出结果

	DM_RD	32	数据存储器读出结果
	PC_4	16	PC+4 的值
Output	RD1	32	寄存器组读出 A 端寄存器
	RD2	32	寄存器组读出 B 端寄存器
	Ext	32	扩展单元输出结果寄存器
	debug_WD	32	debug 信号连接写入数据 WD
	debug_A3	32	debug 信号连接写入地址 A3

3. 变量定义：

变量	类型	位宽	描述
RF[0:31]	reg	32	32 个寄存器形成的寄存器组
A1,A2	wire	5	寄存器组两个读出地址
sign_ext,zero_ext	wire	16	扩展单元需扩展的高 16 位
WD,A3	reg	32	WD：写入寄存器组的数据 A3：写入寄存器组的地址
WD_PC_4	reg	16	用于 jal 指令将 PC_4 从时钟上升沿开始保持一个周期

4. 关键代码：

```
always @(*)
    begin
        case (WRSel)
            2'b00: A3 = IM_D[20:16];
            2'b01: A3 = IM_D[15:11];
            2'b10: A3 = 5'b11111;
        endcase
        case (WDSel)
            2'b00: WD = ALUOut;
            2'b01: WD = DM_RD;
            2'b10: WD = WD_PC_4;
        endcase
    end
always @(posedge clock)
    begin
        RD1 <= RF[A1];
        RD2 <= RF[A2];
        Ext <= (EXTOp) ? {sign_ext, IM_D[15:0]} :
                         {zero_ext, IM_D[15:0]};
        WD_PC_4 <= PC_4;
        if (RFWr == 1)
            RF[A3] <= WD;
    end
```

五、 执行模块

1. 模块功能：在一定时钟周期和 ALUOp 信号的控制下，根据输入的值完成指定的运算，并输出计算结果 ALUOut，以及部分指令所需的 Zero 和 Pos 信号。
2. 输入输出信号：

信号类型	信号	位宽	描述
Input	clk	1	时钟信号
	rst	1	复位信号
	RF1	32	寄存器组输入 A
	RF2	32	寄存器组输入 B
	Ext	32	扩展单元结果输入
	shamt	5	IR[10:6]输入 C
	ALUOp	5	ALU 功能选择信号
	BSel	1	B 端口多路选择控制信号
Output	Zero	1	结果为零标志
	Pos	1	BGTZ 结果为正标志
	ALUOut	32	ALU 计算结果寄存器

3. 变量定义：

变量	类型	位宽	描述
A	wire	32	输入端口 A
B	wire	32	输入端口 B
C	wire	5	输入端口 C

4. 关键代码：

```
always @(*)
        begin
            if (rst == 1)
                ALUOut = 31'd0;
            case (ALUOp)
                `ADD_u: ALUOut = A + B;
                `SUB_u: ALUOut = $signed(A) -$signed(B);
                `SUBU_u: ALUOut = A - B;
                `AND_u: ALUOut = A & B;
                `OR_u:    ALUOut = A | B;
                `XOR_u: ALUOut = A ^ B;
                `NOR_u: ALUOut = ~(A | B);
                `SLT_u: ALUOut = ($signed(A)<$signed(B)) ? 1:0;
                `SLTU_u: ALUOut = (A < B) ? 1:0;
                `SLL_u: ALUOut = B << C;
                `SRL_u: ALUOut = B >> C;
                `SRA_u: ALUOut = $signed(B) >>> C;
                `SLLV_u: ALUOut = B << A;
                `SRLV_u: ALUOut = B >> A;
```

```
                    `SRAV_u: ALUOut = $signed(B) >>> A;
                    `LUI_u: ALUOut = (B << 16) & 32'h0FFF0000;
                    `BGTZ_u: Pos = ($signed(A) > 0) ? 1:0;
                    default: ALUOut = ALUOut;
                endcase
            end
```

六、 数据存储器模块

1. 模块功能：分配数据存储器 RAM，并在一定时钟周期和信号控制下根据地址读出/写入数据。
2. 输入输出信号：

信号类型	信号	位宽	描述
Input	clock	1	时钟信号
	Memwrite	1	写使能信号
	write_data	32	写入的数据
	address	32	写入的地址
Output	DM_RD	32	读出的数据

3. 变量定义：

变量	类型	位宽	描述
clk	wire	1	反向时钟信号
read_data	wire	32	读出的数据

4. 关键代码：

```
assign clk = ~clock;
    ram ram (
        .clka(clk),
        .wea(Memwrite),
        .addra(address[15:2]),
        .dina(write_data),
        .douta(read_data)
    );
    always @(*)
        begin
            DM_RD = read_data;
        end
```

设计主要测试结果（仿真截图或下载照片）

（自己实现的仿真部分代码及截图（至少包括除时钟和存储器外的 2 个模块），贴主要说明问题的时序图并对时序进行分析，可以竖贴）

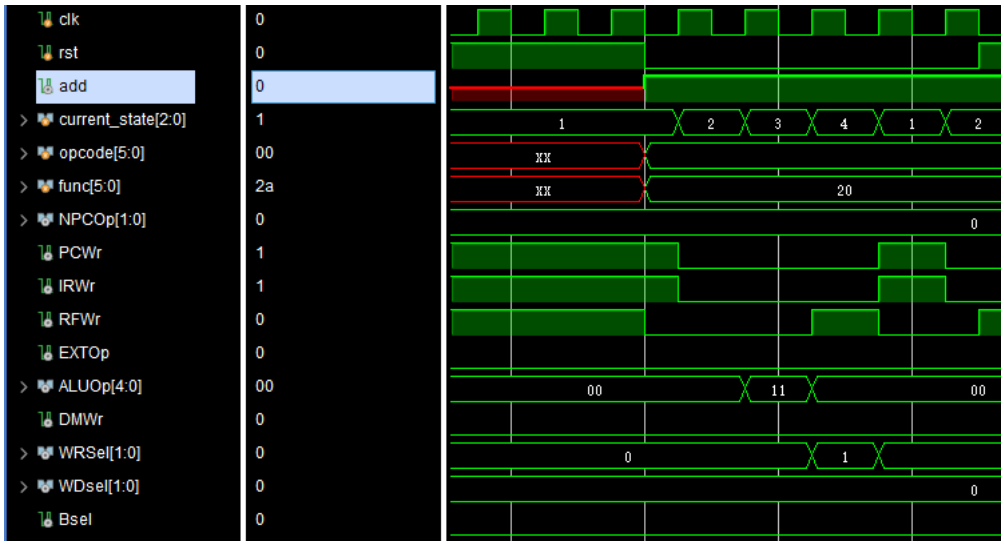
一、 控制器模块仿真（control32_sim.v）

1. 仿真部分代码

```
always #5 clk = ~clk;
    initial begin
        #500 begin opcode = 5'd0; func = `ADD;rst = 1'b0;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0;opcode = 5'd0; func = `SLT;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0; opcode = 5'd0; func = `SLL;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0;opcode = 5'd0; func = `JR;end
        #50 rst = 1'b1;
        //I TYPE
        #10 begin rst = 1'b0;opcode = `ADDI;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0;opcode = `LUI;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0;opcode = `LW;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0;opcode = `SW;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0;opcode = `BEQ; Zero = 1'b1;end
        #50 rst = 1'b1;
        #10 begin rst = 1'b0; opcode = `BEQ; Zero = 1'b0;end
        #50 rst = 1'b1;
        //J TYPE
        #10 begin rst = 1'b0;opcode = `JAL;end
        #50 rst = 1'b1;
         $finish;
    end
```

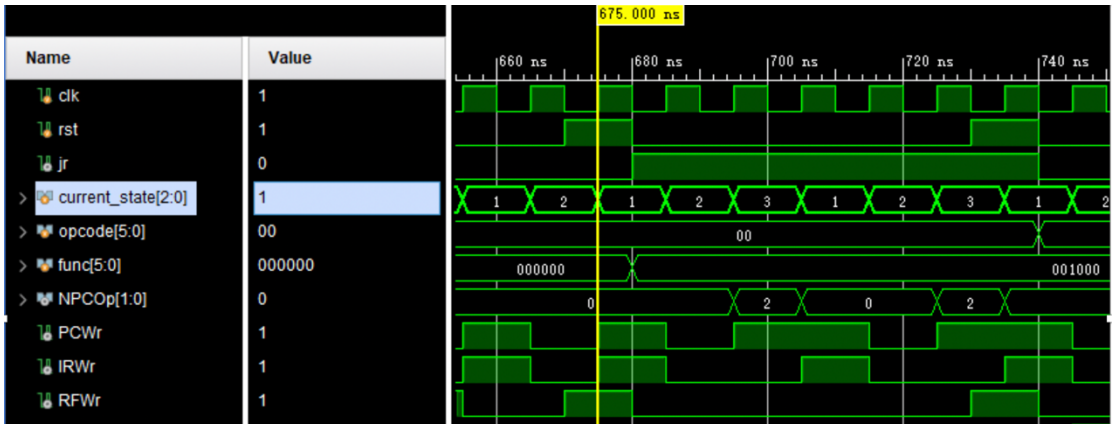
2. 截图及时序分析

1) Add 指令



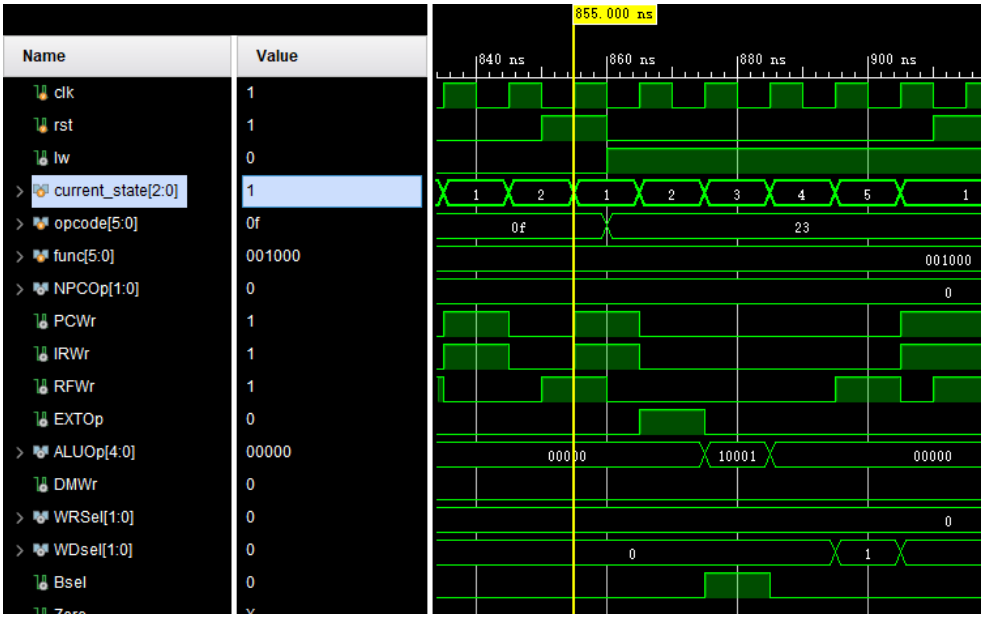
该指令共 4 个周期，在 T1（current_state=1）时拉高 PCWr 和 IRWr，NPCOp=0 执行 PC+4；T3 时得到 add 指令对应的 ALUOp，BSel=0，选择 RF.B 端口数据；T4 时拉高 RFWr，WRSel=1，WDSel=0 将 ALUOut 写入寄存器组，地址为 IR[15:11]。

2) Jr 指令



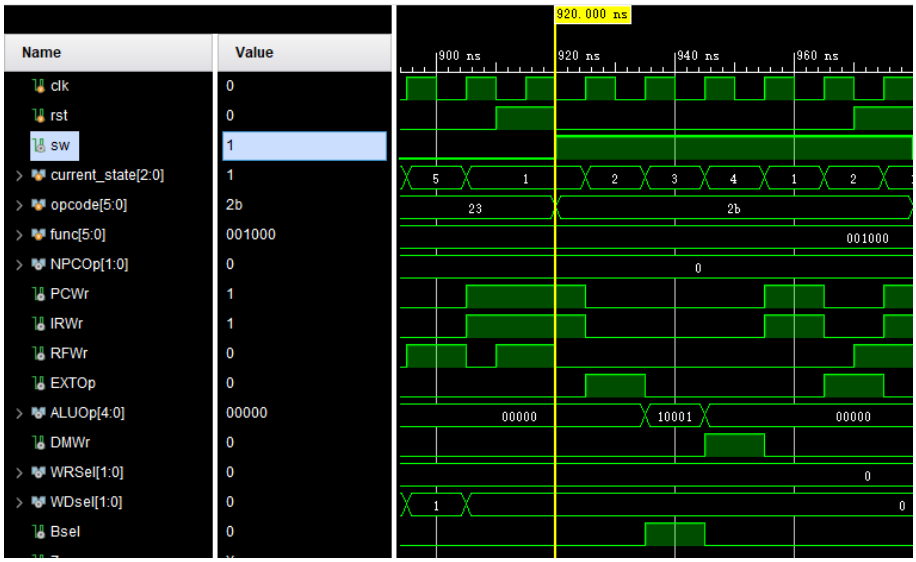
该指令共 3 个周期，在 T1 拉高 PCWr 和 IRWr，NPCOp=1 执行 PC+4；T3 时再拉高 PCWr，并且给到 NPCOp=2（JRPC），其他控制信号默认为 0。

3) Lw 指令



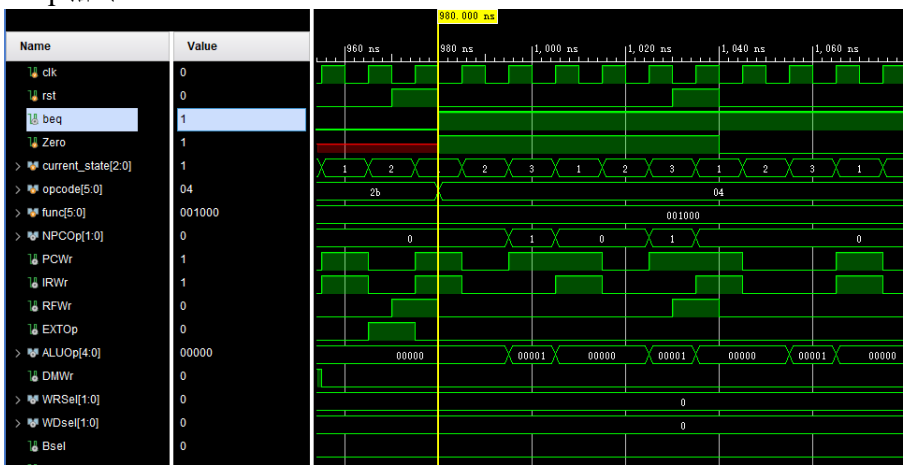
该指令共 5 个周期，T1 时拉高 PCWr 和 IRWr，并取 NPCOp=0 执行 PC+4；T2 时 EXTOp=1，进行符号扩展；T3 时得到 ALUOp=10001（加法计算），并拉高 BSel=1，ALU 输入 B 端取扩展单元的输出结果；T4 为从数据存储器取数据阶段，无控制信号；T5 时 RFWr=1，WRSel=WDSel=0，将数据写回寄存器组。

4) Sw 指令



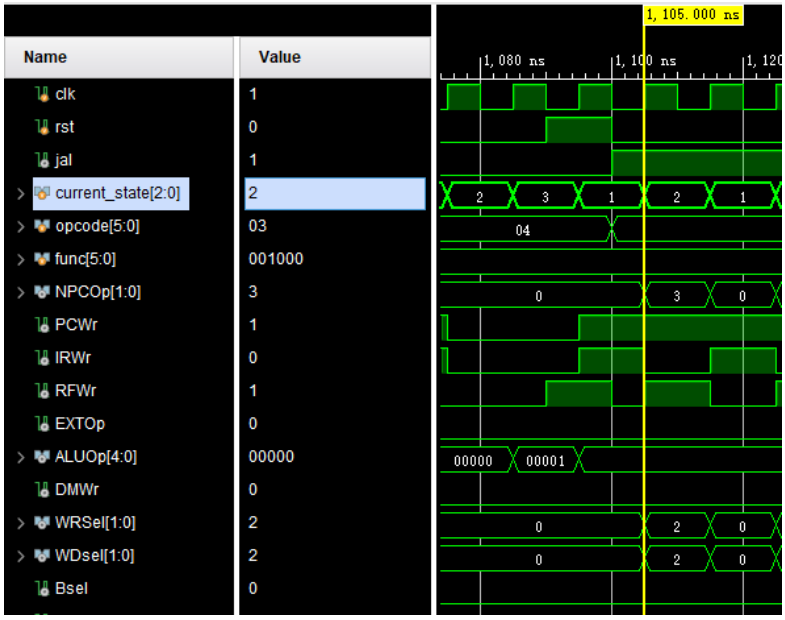
该指令共 4 个周期，T1 时拉高 PCWr 和 IRWr，并取 NPCOp=0 执行 PC+4；T2 时 EXTOp=1，进行符号扩展；T3 时得到 ALUOp=10001（加法计算），并拉高 BSel=1，ALU 输入 B 端取扩展单元的输出结果；T4 时拉高 DMWr，将数据写入存储器。

5) Beq 指令



该指令共 3 个周期，T1 时拉高 PCWr 和 IRWr，并取 NPCOp=0 执行 PC+4；T3 时 ALUOp=1（减法操作），并根据 ALU 返回的标志信号 Zero 来判断是否需要跳转（修改 PC）：Zero=1，T3 时 PCWr 拉高，NPCOp=1（BNPC）；Zero=0，T3 时 PCWr 仍为 0，不修改 PC。

6) Jal 指令



该指令共 2 个时钟周期，T1 时拉高 PCWr 和 IRWr，并取 NPCOp=0 执行 PC+4；T2 时拉高 PCWr，NPCOp=3（JNPC），同时拉高 RFWr，WRSel=WDSel=2，将 PC 值写入寄存器组$31。

二、 执行部件模块仿真

1. 仿真代码

```
always #5 clk = ~clk;
```

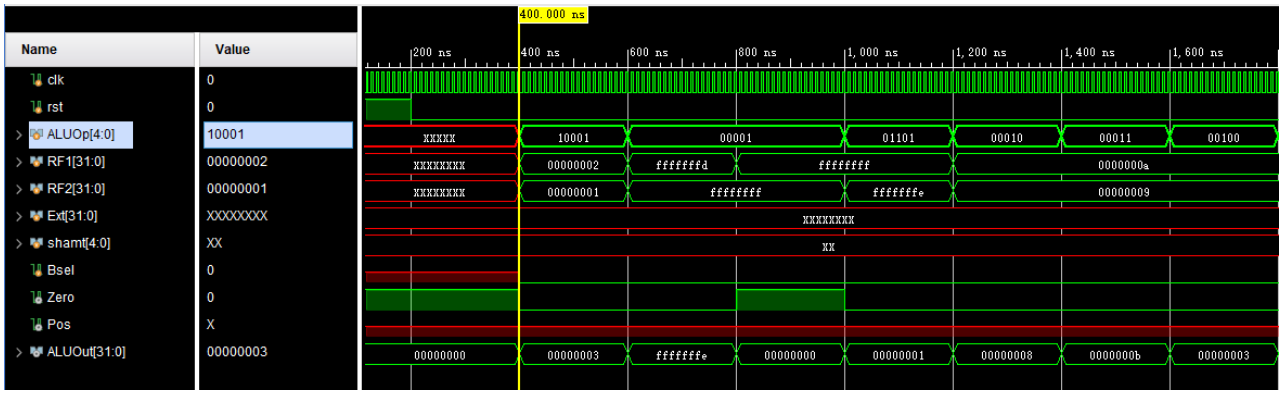
```
initial begin
    #200 rst = 1'b0;
    #200 begin
            ALUOp = `ADD_u;
            RF1 = 32'h00000002;
            RF2 = 32'h00000001;
            Bsel = 0;
        end
    #200 begin
            ALUOp = `SUB_u;
            RF1 = 32'hfffffffd; //-3
            RF2 = 32'hffffffff; //-1
            Bsel = 0;
            //-3 - -1 = -2, Zero = 0
        end
    #200 begin
            ALUOp = `SUB_u;
            RF1 = 32'hffffffff; //-1
            RF2 = 32'hffffffff; //-1
            Bsel = 0;
            //-1 - -1 = 0, Zero = 1
        end
    #200 begin
            ALUOp = `SUBU_u;
            RF1 = 32'hffffffff;
            RF2 = 32'hfffffffe; // ans = 1
            Bsel = 0;
        end
    #200 begin
            ALUOp = `AND_u;
            RF1 = 32'h0000000A; //1010
            RF2 = 32'h00000009; //1001 ans = 1000 (8)
            Bsel = 0;
        end
    #200 begin
            ALUOp = `OR_u;
            RF1 = 32'h0000000A; //1010
            RF2 = 32'h00000009; //1001 ans = 1011 (B)
            Bsel = 0;
        end
    #200 begin
            ALUOp = `XOR_u;
            RF1 = 32'h0000000A; //1010
            RF2 = 32'h00000009; //1001 ans = 0011 (3)
```

```
                Bsel = 0;
        end
#200 begin
                ALUOp = `NOR_u;
                RF1 = 32'h0000000A; //1010
                RF2 = 32'h00000009; //1001 ans = 0100 (4)
                Bsel = 0;
        end
#200 begin
                ALUOp = `SLT_u;
                RF1 = 32'hffffffff; //-1
                RF2 = 32'h00000001; //1, ans = 1 (<)
                Bsel = 0;
        end
#200 begin
                ALUOp = `SLTU_u;
                RF1 = 32'hffffffff; //
                RF2 = 32'h00000001; //1, ans = 0 (>)
                Bsel = 0;
        end
#200 begin
                ALUOp = `SLL_u;
                RF2 = 32'h00000001;
                shamt = 5'b00010; // 0001 << 2    ans =    0004H
        end
#200 begin
                ALUOp = `SRL_u;
                RF2 = 32'h80000008;
                shamt = 5'b00010; //    >> 2    ans = 2000 0002H
        end
#200 begin
                ALUOp = `SRA_u;
                RF2 = 32'h80000008;
                shamt = 5'b00010; //    >>> 2    ans = e000 0002H
        end
#200 begin
                ALUOp = `SLLV_u;
                RF2 = 32'h00000001;
                RF1 = 32'h00000002; // 0001 << 2    ans =    0004H
        end
#200 begin
                ALUOp = `SRLV_u;
                RF2 = 32'h80000008;
                RF1 = 32'h00000002; //    >> 2    ans = 2000 0002H
```

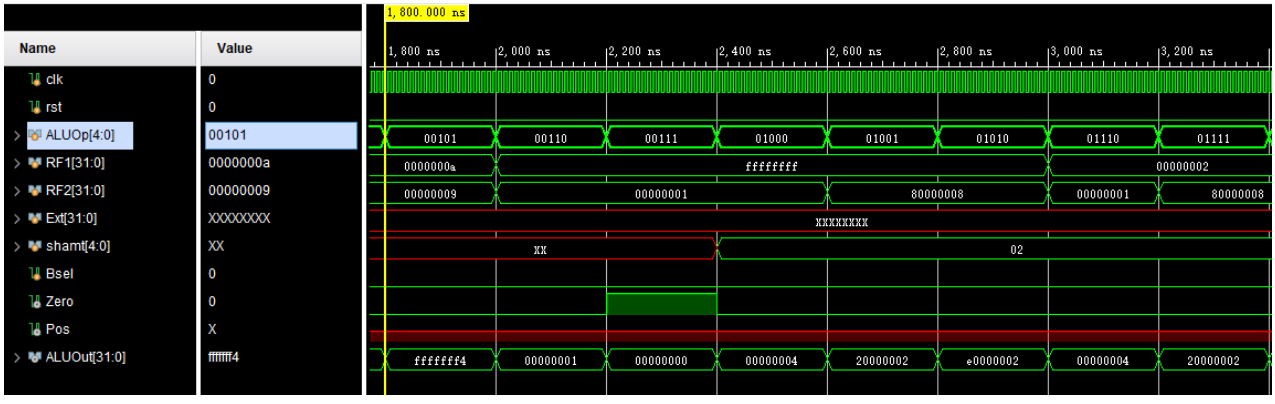
```
        end
    #200 begin
                ALUOp = `SRAV_u;
                RF2 = 32'h80000008;
                RF1 = 32'h00000002; //    >>> 2    ans = e000 0002H
        end
     #200 begin
                ALUOp = `ADD_u;    //addi
                RF1 = 32'h00000002;
                Ext = 32'h00000001; //ans = 3
                Bsel = 1;
        end
     #200 begin
                ALUOp = `AND_u;    //andi
                RF1 = 32'h0000000A; //1010
                Ext = 32'h00000009; //1001 ans = 1000 (8)
                Bsel = 1;
        end
    #200 begin
                ALUOp = `OR_u; //ori
                RF1 = 32'h0000000A; //1010
                Ext = 32'h00000009; //1001 ans = 1011 (B)
                Bsel = 1;
        end
    #200 begin
                ALUOp = `XOR_u; //xori
                RF1 = 32'h0000000A; //1010
                Ext = 32'h00000009; //1001 ans = 0011 (3)
                Bsel = 1;
        end
    #200 begin
                ALUOp = `SLTU_u; //sltui
                RF1 = 32'hffffffff; //
                Ext = 32'h00000001; //1, ans = 0 (>)
                Bsel = 1;
        end
    #200 begin
                ALUOp = `LUI_u;
                Ext = 32'h00000002; //      ans = 0002 0000 H
                Bsel = 1;
        end
    #200 begin
                ALUOp = `BGTZ_u;
                RF1 = 32'hffffffff; //    Pos = 0
```

```
            end
        #200 begin
                ALUOp = `BGTZ_u;
                RF1 = 32'h00000001; //    Pos = 1
            end
    end
```

2. 截图及时序分析

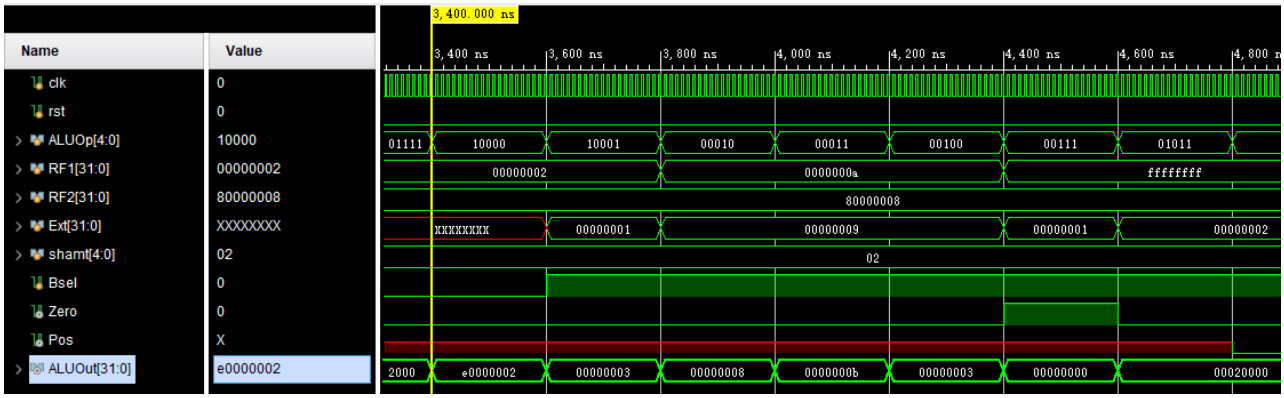


1) ALUOp=10001 执行 ADD 运算，A=2，B=1，ALUOut=3
2) ALUOp=00001 执行 SUB 运算，A= -3, B= -1, ALUOut = -2, Zero = 0; A = -1, B = -1, ALUOut = 0, Zero = 1
3) ALUOp=11101 执行 SUBU 运算，A = ffffffff, B = fffffffe, ALUOut = 1, Zero = 0
4) ALUOp=00010 执行 AND 运算，A = 1010b(0aH), B = 1001b(09H), ALUOut=1000b(08H)
5) ALUOp=00011 执行 OR 运算，A = 1010b(0aH), B = 1001b(09H), ALUOut=1011b(0bH)
6) ALUOp=00100 执行 XOR 运算，A = 1010b(0aH), B = 1001b(09H), ALUOut=0011b(03H)

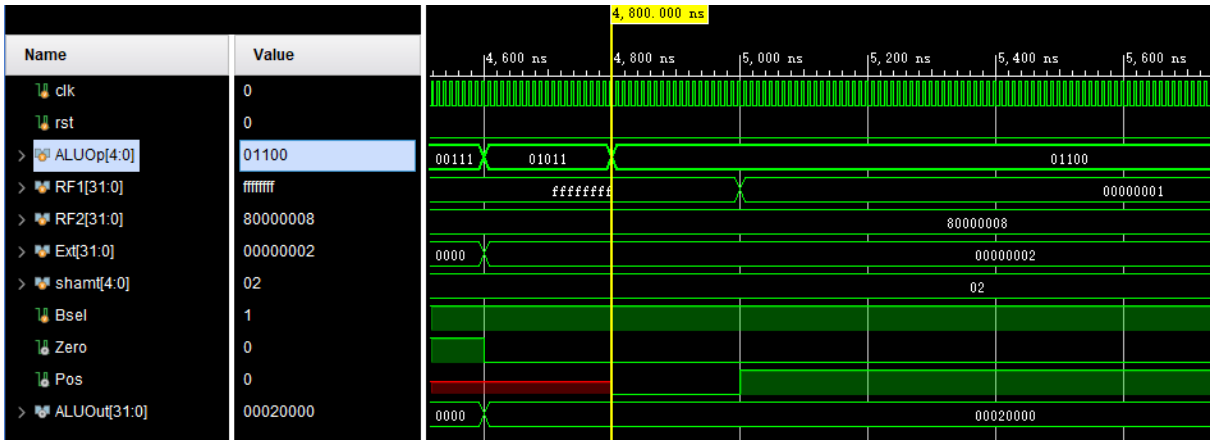


7) ALUOp=00101 执行 NOR 运算，A = 1010b(0aH), B = 1001b(09H), ALUOut=ffffffff4H
8) ALUOp=00110 执行 SLT 运算，A = ffffffff, B = 1, A < B, ALUOut = 1
9) ALUOp=00111 执行 SLTU 运算，A = ffffffff, B = 1, A > B, ALUOut = 0
10) ALUOp=01000 执行 SLL 运算，B = 00000001, shamt = 2, B<<shamt, ALUOut = 00000004

11) ALUOp=01001 执行 SRL 运算， B = 80000008, shamt = 2, B>>shamt, ALUOut = 20000002
12) ALUOp=01010 执行 SRA 运算， B = 80000008, shamt = 2, B>>>shamt, ALUOut = e0000002
13) ALUOp=01110 执行 SLLV 运算，A =2, B = 00000001, B<<A, ALUOut = 00000004
14) ALUOp=01111 执行 SRLV 运算，A =2, B = 80000008, B>>A, ALUOut = 20000002



15) ALUOp=10000 执行 SRAV 运算，A =2, B = 80000008, B>>>A, ALUOut = e0000002
16) ALUOp=10001 执行 ADD 运算， BSel = 1, A=2， Ext=1， ALUOut=3
17) ALUOp=00010 执行 AND 运算， BSel = 1, A = 1010b(0aH), Ext = 1001b(09H), ALUOut=1000b(08H)
18) ALUOp=00011 执行 OR 运算， BSel = 1, A = 1010b(0aH), Ext = 1001b(09H), ALUOut=1011b(0bH)
19) ALUOp=00100 执行 XOR 运算， BSel = 1,A = 1010b(0aH), Ext = 1001b(09H), ALUOut=0011b(03H)
20) ALUOp=00111 执行 SLTU 运算， BSel = 1,A = ffffffff, Ext = 1, A > B, ALUOut = 0
21) ALUOp=01011 执行 LUI 运算， BSel = 1,Ext = 00000002, ALUOut = 00020000

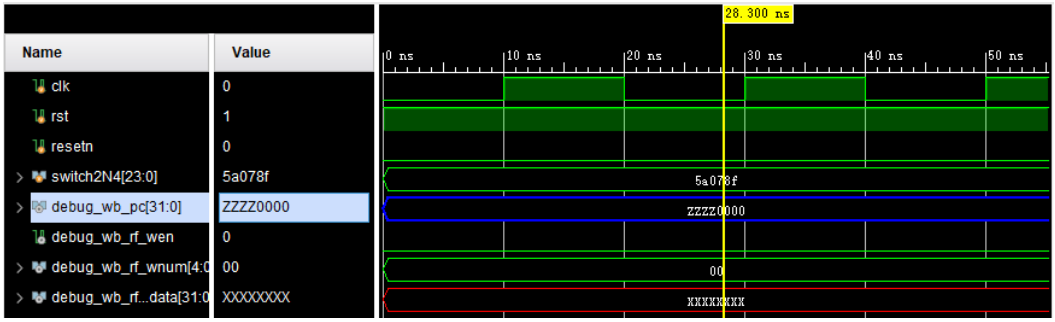


22) ALUOp =01100 执行 BGTZ 运算，RF1= ffffffff < 0, Pos = 0; RF = 00000001 > 0, Pos =1

设计过程中遇到的问题及解决方法

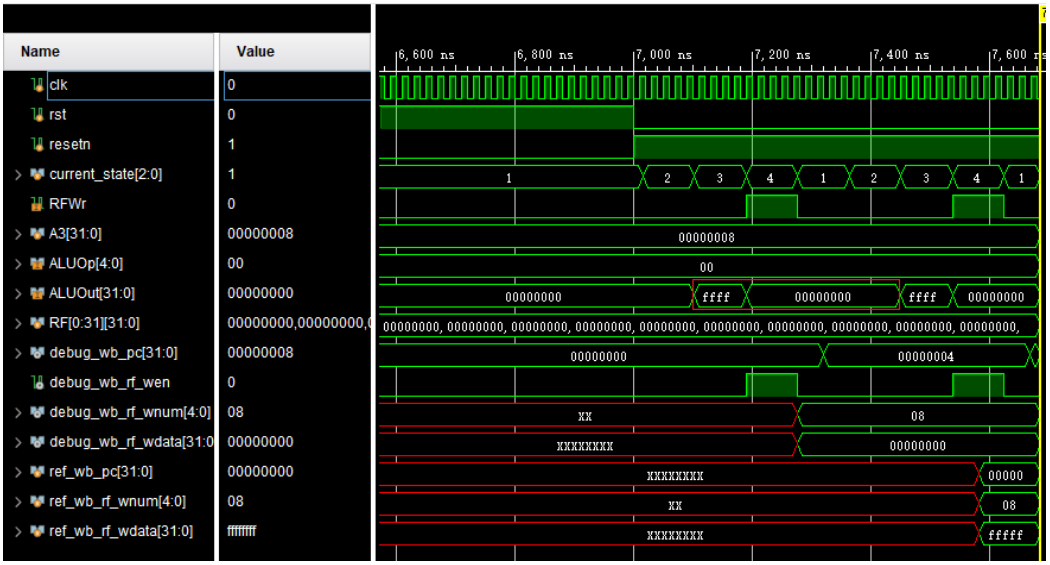
（包括设计过程中的错误及测试过程中遇到的问题）

1. debug_wb_pc 信号是 32 位，而系统使用的 PC 是 16 位，不能直接相连导致前 15 位输出为高阻态。
   解决方法：将 PC 信号拼接上高 16 位补零连接到 debug 信号线。



2. DM 中 IP 核的输出信号 read_data 不能直接定义为 reg 型。
   解决方法：再定义一个 wire 类型的信号接 IP 核的输出，再将该信号给到 DM 的寄存器。

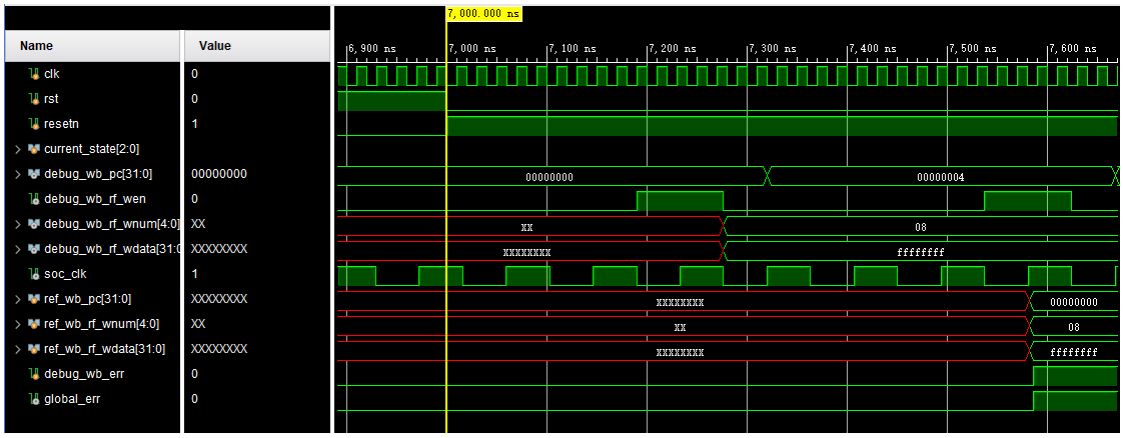
3. ALUOut 的结果没有保持：因为 ALUOp 为 0 时定义的是执行加法运算，导致 T4 时默认 ALUOp 为 0，组合逻辑的 ALUOut 因为输入的变化执行加法运算后瞬间改变，写入 RF 的值错误。
   解决方法：将 ALUOp=0 设置为 default 值，ALUOut 保持不变；修改加法运算 ADD 的 ALUOp 值。



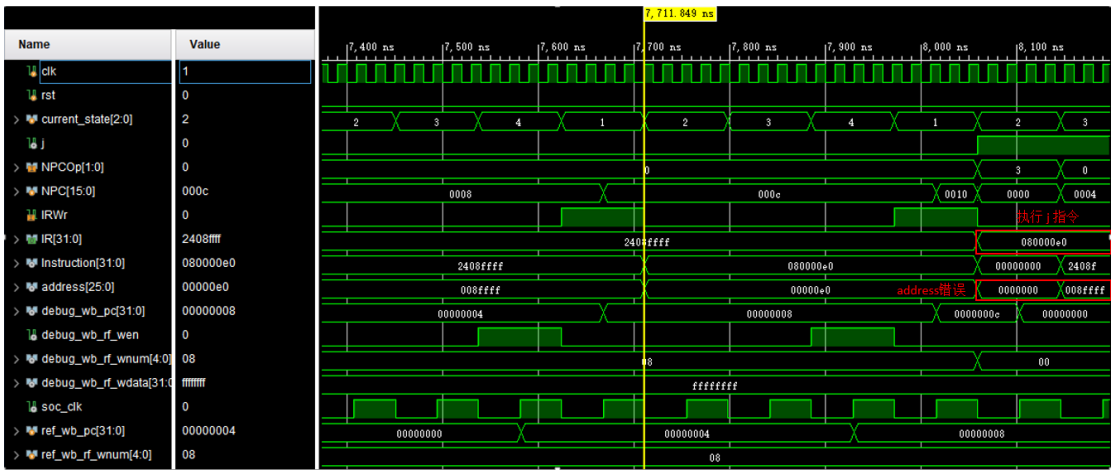
4. 控制器的状态机中每个状态都持续了两个时钟周期：因为次态组合逻辑的 always 块采用了时钟上升沿触发，导致次态的变化慢了一个周期，每个状态都持续了两个时钟周期。
   解决方法：将次态组合逻辑的 always 块采用*触发。

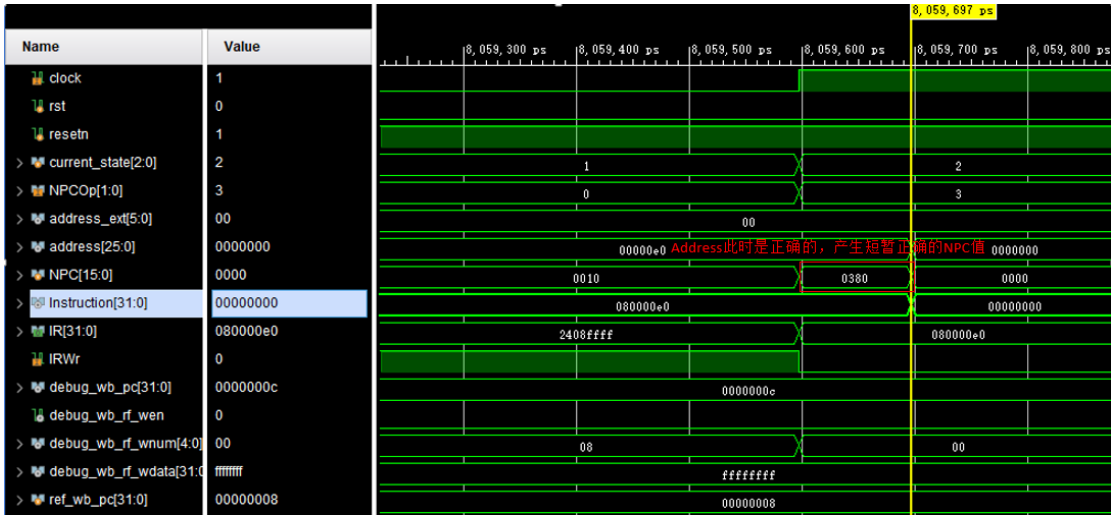
5. 在测试时 debug_wb_pc 比 ref 快了很多个周期。
   解决方法：在 reset 信号为高电平期间设置 RFWr 信号为低电平，golden_trace 文件删去所

有重复的第一行，只剩一行 1 00000000 8 ffffffff（删后同 31 条指令测试包的 golden_trace 文件），PC 延迟一拍。

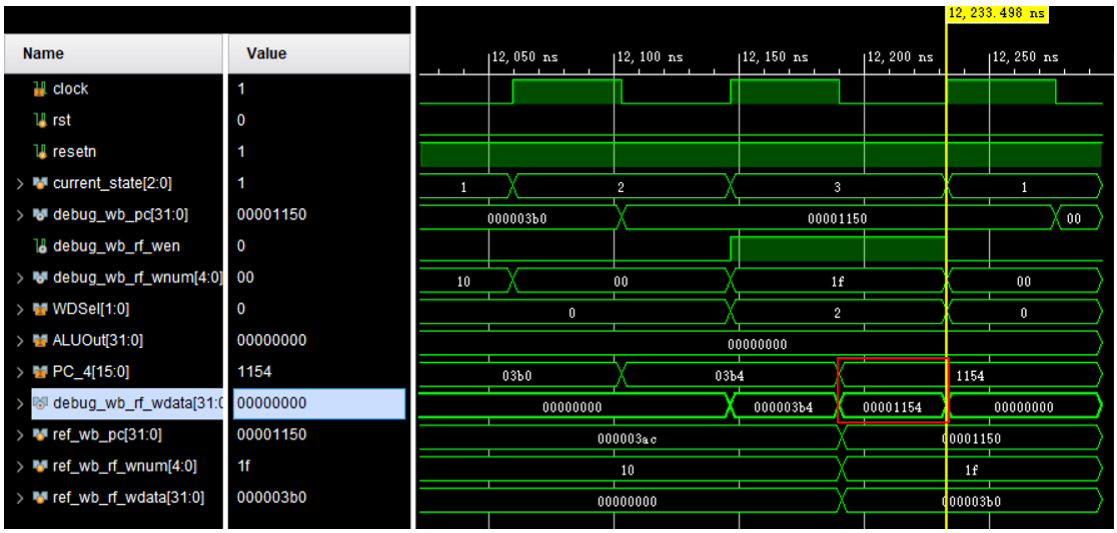


6. 在执行 j 指令时，NPC 计算错误，因为 address 读取错误。
解决方法：address 应该从 IR 寄存器读出而不是直接从 ROM 中读出，因为系统采用的是 Block Memory 的 IP 核，ROM 读出的内容是下一条指令。

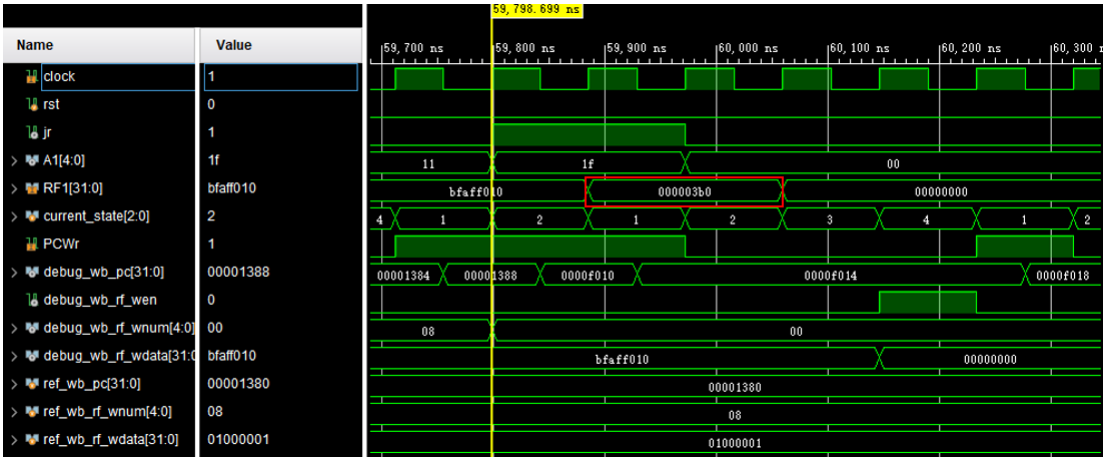




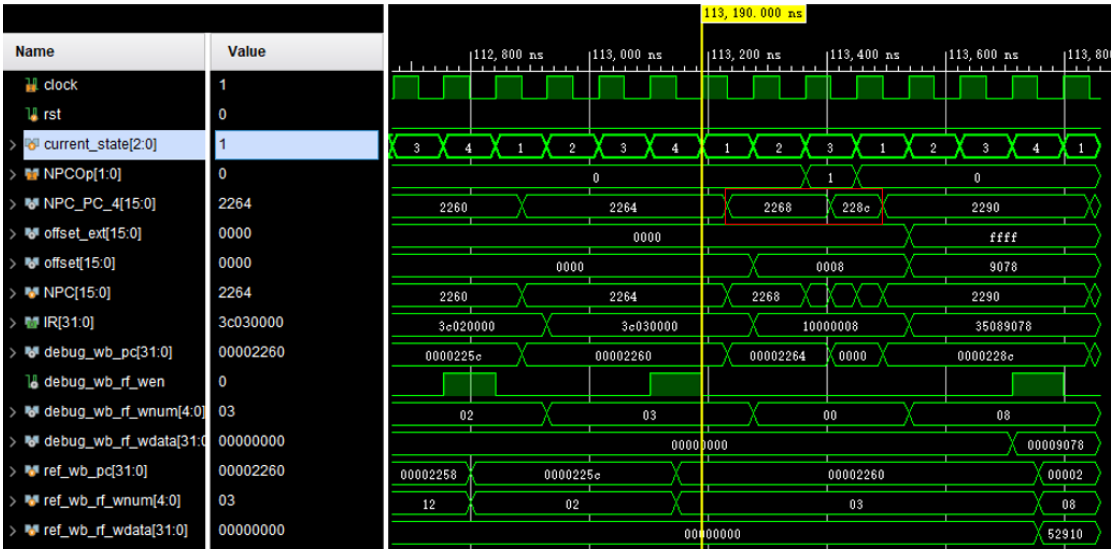
7. PC=3ac，jal 指令在 T3 需要将下条指令的 PC 写入寄存器时，debug_wd_rf_wdata 随着 PC_4（PC+4）的变化而突然变化，导致保存在寄存器的 PC 值错误，跳转到子程序后无法正常回到主程序中执行。
   解决方法：新增加一个变量 WD_PC_4 在时钟上升沿保存 PC_4 的值，维持一个周期，当执行 jal 指令时，将 WD_PC_4 的值写入寄存器，便不会因为 PC_4 在下降沿的变化而写入错误了。



8. 执行 jr 指令时 RF1（RF.RD1）因为在时钟上升沿才读出导致慢了一个时钟周期，在 T2 时 PC 无法得到正确的要跳转的指令地址。
   解决方法：将 jr 指令改为 3 个时钟周期，在 T3 时才改变 NPC 写入 PC。



9. 执行 PC=2260 的 beq 指令时，在 T3 计算 NPC 时产生计算错误。因为 beq 指令计算 NPC 时用的是 NPC_PC_4 信号，随着 PC 在 T1 变为 2264+4=2268，它也变成 2268，而计算 beq 跳转的指令地址应该是 2260+4+ ((Sign-Extend) offset<<2)，而不是 2264+4+ ((Sign-Extend) offset<<2)。
   解决方法：增加 PC_pre 变量记录上一次的 PC 值，计算 beq 指令的 NPC 值时采用 PC_pre+4+ ((Sign-Extend) offset<<2)的计算方式。


Name
Value
clock
1
rst
0
current_state[2:0]
1
NPCOp[1:0]
0
NPC_PC_4[15:0]
2264
offset_ext[15:0]
0000
offset[15:0]
0000
NPC[15:0]
2264
IR[31:0]
3c030000
debug_wb_pc[31:0]
00002260
debug_wb_rf_wen
0
debug_wb_rf_wnum[4:0]
03
debug_wb_rf_wdata[31:0]
00000000
ref_wb_pc[31:0]
00002260
ref_wb_rf_wnum[4:0]
03
ref_wb_rf_wdata[31:0]
00000000
113, 190. 000 ns
112, 800 ns
113, 000 ns
113, 200 ns
113, 400 ns
113, 600 ns

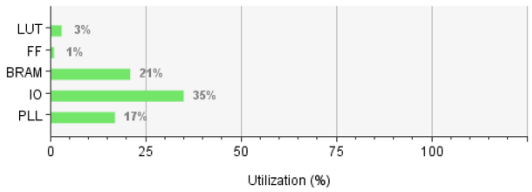
设计的性能分析

（资源使用情况、主频、功耗数据和自我分析）

一、资源使用情况

Name	Slice LUTs (63400)	Block RAM Tile (135)	Bonded IOB (285)	BUFGCTRL (32)	PLLE2_ADV (6)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)
minisys	1875	29	99	4	1	1293	319	31	750	1875
cc (cpuclk)	0	0	0	2	1		0	0	0	0
cu (control32)	349	0	0	0	0		3	0	155	349
dec (idecode32)	1207	0	0	0	0		257	0	617	1207
dm (dmemory32)	31	14.5	0	0	0		0	0	12	31
exe (executs32)	9	0	0	0	0		0	0	41	9
ifetc (ifetc32)	279	14.5	0	0	0		59	31	139	279

Resource	Utilization	Available	Utilization %
LUT	1875	63400	2.96
FF	1293	126800	1.02
BRAM	29	135	21.48
IO	99	285	34.74
PLL	1	6	16.67



二、主频

在保证仿真测试正确的前提下，本设计主频最高可以达到 115MHz。

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.167 ns	Worst Hold Slack (WHS):	0.204 ns	Worst Pulse Width Slack (WPWS):	2.633 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2625	Total Number of Endpoints:	2625	Total Number of Endpoints:	1278

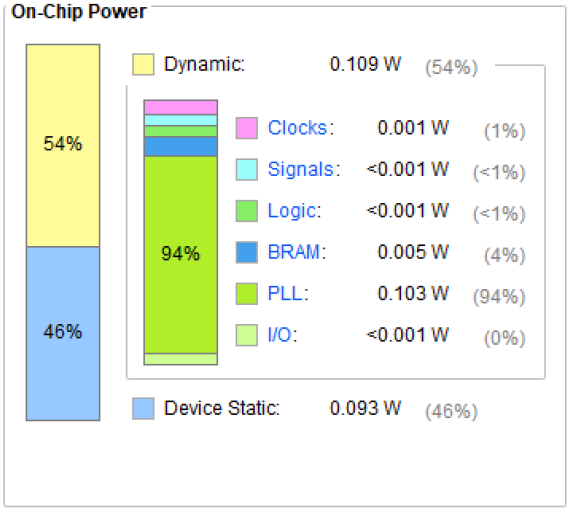
三、功耗数据

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.202 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**
Thermal Margin:	59.5°C (22.1 W)
Effective ϑJA:	2.7°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



四、自我分析

总片上的功耗达到了 0.202W，功耗较大，其中 PLL 的功耗占比最大。IO 资源使用最多，其次是 Block RAM。建立时间为 0.167ns，保持时间为 0.204ns。

项目总结

（包括设计的总结和还需改进的内容以及收获）

1. 总结：在本次计算机设计与实践的多周期 CPU 设计实验中，通过完成数据通路表和控制信号表建立起对整个系统框架的认识，从宏观的搭建框架开始，设计每个模块、每个部件，细化到每个寄存器和变量的设置，在单周期 CPU 工程的基础上，增加了状态机控制信号和各个部件的寄存器，共设计了六个模块，编写并通过了两个模块的功能仿真测试，调高时钟频率到 115MHz，最终在 190000ns 左右的时间内完成了 31 条指令的测试，并对系统的性能进行了简要分析。在实验过程中，遇到问题时反复核对波形图，找到问题的根源，并翻阅相关 PPT 和资料，咨询老师和同学，最终解决。

2. 需改进的内容：对关键路径进行分析，减小组合逻辑电路的延迟；可以用更少的 ALUOp 实现各条指令的不同运算；NPC 的计算部分和 Ext 扩展单元的计算部分有重复。

3. 收获：通过本次实验，对 CPU 中各个部件的功能和它们之间的相互关联的理解更加深刻，深入了解了各个部件的工作原理，对硬件语言的使用更加熟悉，在 debug 时磨练了自己的耐心和细心，独立跟进了自己的系统从零到完善的过程，培养了独立思考和发现问题、解决问题的能力。