计算机体系结构实验课程第_二_次实报告

实验名称	静态 5 级流水线 CPU 实现			班级	李雨森老师
学生姓名	蒋薇	学号	2110957	指导老师	董前琨
实验地点	A308		实验时间	2023.10.09	

1、 实验目的

1. 在多周期 CPU 实验完成的提前下，深入理解 CPU 流水线的概念。
2. 熟悉并掌握流水线 CPU 的原理和设计。
3. 最终检验运用 verilog 语言进行电路设计的能力。
4. 通过亲自设计实现静态 5 级流水线 CPU,加深对计算机组成原理和体系结构理
论知识的理解。
5. 培养对 CPU 设计的兴趣，加深对 CPU 现有架构的理解和深思。

2、 实验内容说明

多周期 CPU 中只要求实现了 30 多条指令，此处要求扩展到 40 多条指令。

1. 做好预习：

1） 复习好上一次多周期 CPU 的实验，归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令，对其进行分析，完成表 9.1 的填写；

2） 认真学习流水线的概念，明白流水线的意义和架构，理解数据相关、控制相关和结构相关，尤其需要注意分支跳转指令及其延迟槽指令的处理；

3） 依据自己设计中实现的指令，编写一段不少于 50 行的汇编程序，要求包含所有实现的指令，完成表 9.2 的填写。要求标注出指令间存在的相关，指出 CPU 可能存在的阻塞；

4） 在多周期 CPU 实验的设计框图的基础上，完善课程设计的设计框图，即补充完善图 9.2；

5） 如果对 FPGA 板了解的话，可确定设计中与 FPGA 板上交互的接口，画出包含外围模块的整体设计框图，即补充完善图 9.3。

2. 实验实施：

1） 确认流水 CPU 的设计框图的正确性；

2） 编写 verilog 代码，将表 9.2 中自己编写的汇编程序翻译为二进制，以 coe 文件的方式初始化到指令 ROM 中；

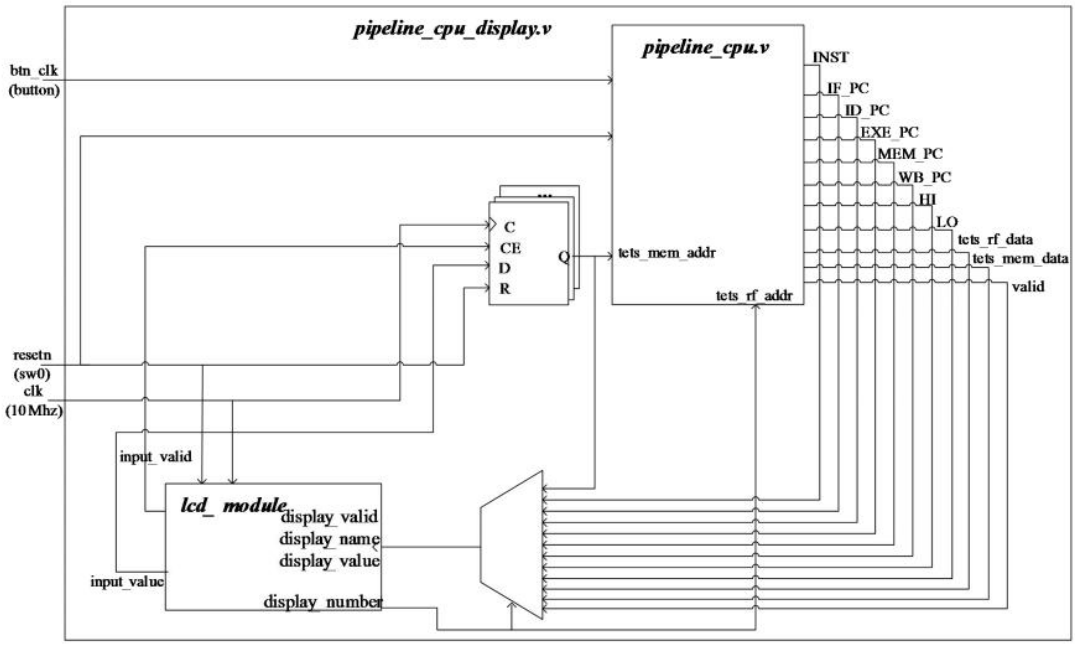
3） 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料，在仿真时需要将生成指令 ROM 时产生的.mif 文件拷贝到工程目录下，才能仿真成功；

4） 完成调用流水 CPU 的外围模块的设计，并编写代码；

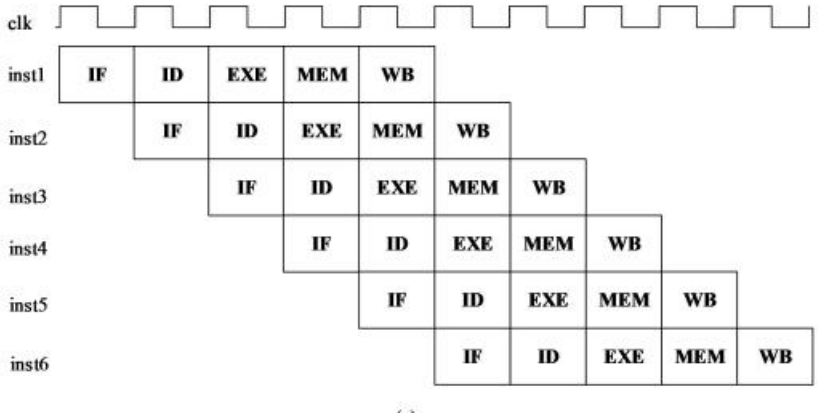
5） 对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证。

3、 实验原理图

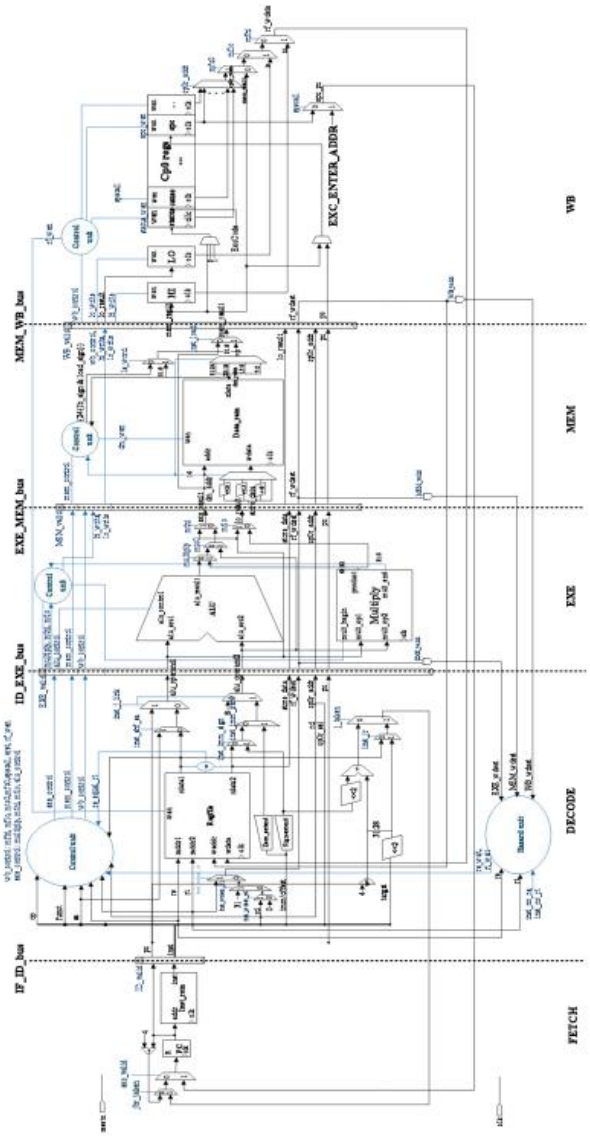
下图为实验顶层模块框图：



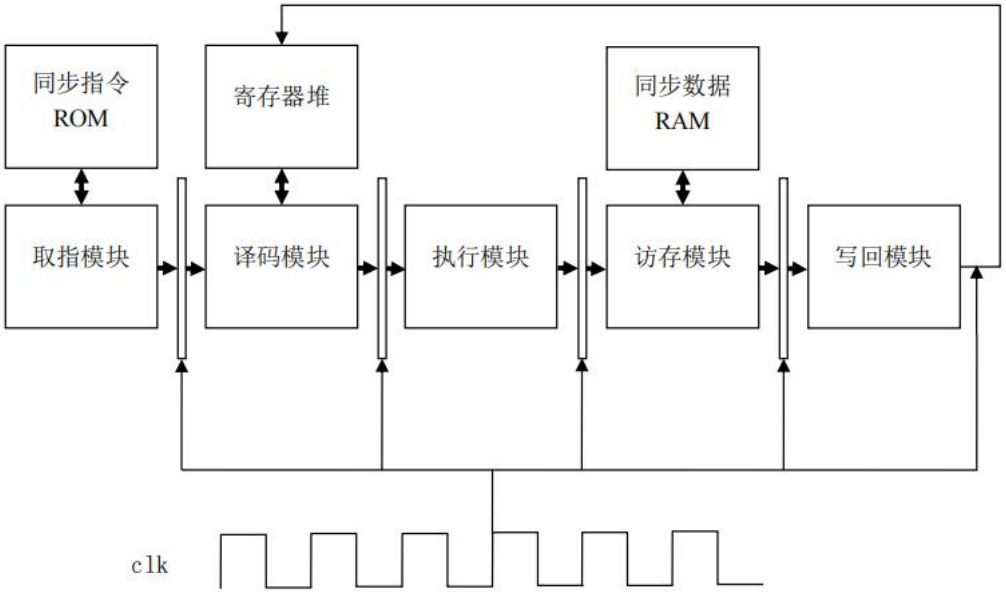
下图为五级流水线时空图：



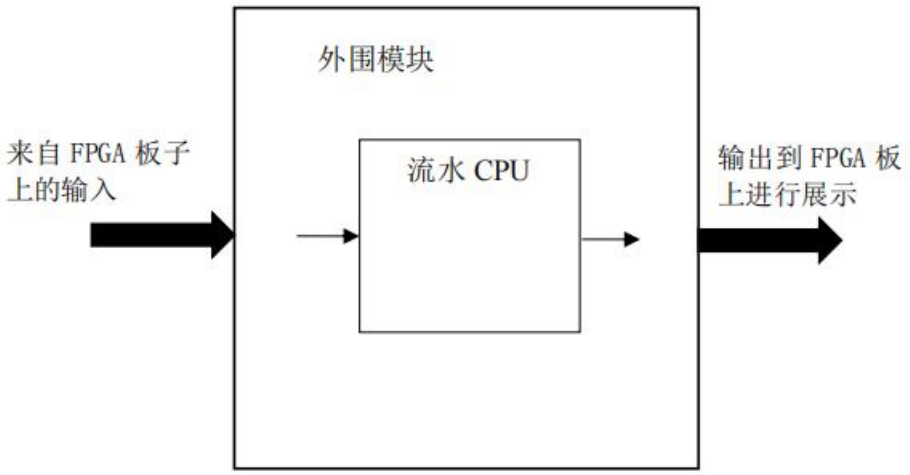
下图为 5 级流水线 CPU 的实现框图：



下图为静态 5 级流水 CPU 的大致框图，5 个部件都是同时运转的，但对每条指令而言，依然是依次工作的。



下图为 3 静态 5 级流水 CPU 设计实验的顶层模块大致框图：



4、 实验步骤

5 级流水线 CPU 实现的 mips 指令特性归纳如下：

指令类型	汇编指令	指令码	源操作数1	源操作数2	源操作数3	目的寄存器	功能描述
R型指令	addu rd,rs,rt	000000|rs|rt|rd|00000|100001	[rs]	[rt]		rd	GPR[rd]=GPR[rs]+GPR[rt]
	subu rd,rs,rt	000000|rs|rt|rd|00000|100011	[rs]	[rt]		rd	GPR[rd]=GPR[rs]-GPR[rt]
	slt rd,rs,rt	000000|rs|rt|rd|00000|101010	[rs]	[rt]		rd	GPR[rd]=(sign(GPR[rs])<sign(GPR[rt]))
	sltu rd,rs,rt	000000|rs|rt|rd|00000|101011	[rs]	[rt]		rd	GPR[rd]=(zero(GPR[rs])<zero(GPR[rt]))
	jalr rs	000000|rs|00000|11111|00000|001001	[rs]			31	GPR[31]=PC,PC=GPR[rs]
	jr rs	000000|rs|000000000000000|001000	[rs]				PC=GPR[rs]
	and rd,rs,rt	000000|rs|rt|rd|00000|100100	[rs]	[rt]		rd	GPR[rd]=GPR[rs]&GPR[rt]
	nor rd,rs,rt	000000|rs|rt|rd|00000|100111	[rs]	[rt]		rd	GPR[rd]=~(GPR[rs]|GPR[rt])
	or rd,rs,rt	000000|rs|rt|rd|00000|100101	[rs]	[rt]		rd	GPR[rd]=GPR[rs]|GPR[rt]
	xor rd,rs,rt	000000|rs|rt|rd|00000|100110	[rs]	[rt]		rd	GPR[rd]=GPR[rs]^GPR[rt]
	sll rd,rt,shf	000000|00000|rt|rd|shf|000000		[rt]		rd	GPR[rd]=zero(GPR[rt])<<shf
	sllv rd,rt,rs	000000|rs|rt|rd|00000|000100	[rs]	[rt]		rd	GPR[rd]=zero(GPR[rt])<<(GPR[rs]%32)
	sra rd,rt,shf	000000|00000|rt|rd|shf|000011		[rt]		rd	GPR[rd]=sign(GPR[rt])>>shf
	srav rd,rt,rs	000000|rs|rt|rd|00000|000111	[rs]	[rt]		rd	GPR[rd]=sign(GPR[rt])>>(GPR[rs]%32)
	srl rd,rt,shf	000000|00000|rt|rd|shf|000010		[rt]		rd	GPR[rd]=zero(GPR[rt])>>shf
	srlv rd,rt,rs	000000|rs|rt|rd|00000|000110	[rs]	[rt]		rd	GPR[rd]=zero(GPR[rt])>>GPR[rs]
	mult rs,rt	000000|rs|rt|0000000000|011000	[rs]	[rt]		(HI,LO)	(HI,LO)=sign(GPR[rs])*sign(GPR[rt])
	mflo rd	000000|0000000000|rd|00000|010010	[LO]			rd	GPR[rd]=[LO]
	mfhi rd	000000|0000000000|rd|00000|010000	[HI]			rd	GPR[rd]=[HI]
	mtlo rs	000000|rs|000000000000000|010011	[rs]			LO	[LO]=GPR[rs]
	mthi rs	000000|rs|000000000000000|010001	[rs]			HI	[HI]=GPR[rs]
I型指令	addiu rt,rs,imm	001001|rs|rt|imm	[rs]	sign_ext(imm)		rt	GPR[rt]=GPR[rs]+sign_ext(imm)
	slti rt,rs,imm	001010|rs|rt|imm	[rs]	sign_ext(imm)		rt	GPR[rt]=(sign(GPR[rs])<sign_ext(imm))
	sltiu rt,rs,imm	001011|rs|rt|imm	[rs]	sign_ext(imm)		rt	GPR[rt]=(zero(GPR[rs])<sign_ext(imm))

指令类型	汇编指令	指令码	源操作数 1	源操作数 2	源操作数 3	目的寄存器	功能描述
I 型指令	beq rs,rt,offset	000100|rs|rt|offset	[rs]	[rt]			if GPR[rs]=GPR[rt] then PC= next_pc+ sign_ext (offset)<<2
	bgez rs,offset	000001|rs|00001|offset	[rs]				if GPR[rs]≥0 then PC= next_pc + sign_ext (offset)<<2
	bgtz rs,offset	000111|rs|00000|offset	[rs]				if GPR[rs]>0 then PC= next_pc + sign_ext (offset)<<2
	blez rs,offset	000110|rs|00000|offset	[rs]				if GPR[rs]≤0 then PC= next_pc + sign_ext (offset)<<2
	bltz rs,offset	000001|rs|00000|offset	[rs]				if GPR[rs]<0 then PC= next_pc + sign_ext (offset)<<2
	bne rs,rt,offset	000101|rs|rt|offset	[rs]	[rt]			if GPR[rs]≠GPR[rt] then PC= next_pc + sign_ext (offset)<<2
	lw rt,offset(b)	100011|b|rt|offset	[b]	sign_ext (offset)		rt	GPR[rt]=Mem[GPR[b]+sign_ext (offset)]
	sw rt,offset(b)	101011|b|rt|offset	[b]	sign_ext (offset)	[rt]		Mem[GPR[b]+sign_ext (offset)]=GPR[rt]
	lb rt,offset(b)	100000|b|rt|offset	[b]	sign_ext (offset)		rt	GPR[rt]=sign(Mem[GPR[b]+sign_ext (offset)])
	lbu rt,offset(b)	100100|b|rt|offset	[b]	sign_ext (offset)		rt	GPR[rt]=zero(Mem[GPR[b]+sign_ext (offset)])
	sb rt,offset(b)	101000|b|rt|offset	[b]	sign_ext (offset)	[rt]		Mem[GPR[b]+sign_ext (offset)]=GPR[rt]
	andi rt,rs,imm	001100|rs|rt|imm	[rs]	zero_ext (imm)		rt	GPR[rt]=GPR[rs]&zero_ext (imm)
	lui rt,imm	001111|00000|rt|imm		{imm, 16'd0}		rt	GPR[rt]= {imm, 16'd0}
	ori rt,rs,imm	001101|rs|rt|imm	[rs]	zero_ext (imm)		rt	GPR[rt]=GPR[rs]|zero_ext (imm)
	xori rt,rs,imm	001110|rs|rt|imm	[rs]	zero_ext (imm)		rt	GPR[rt]=GPR[rs]^zero_ext (imm)
J 型指令	j target	000010|target					[PC]={next_pc[31:28],target<<2}
	jal target	000011|target					GPR[31]=PC,PC={next_pc[31:28],target <<2}
cp0 指令	mfc0 rt,cs	010000|00000|rt|cs|00000000|sel	CPR[cs .sel]			rt	GPR[rt]=CPR[cs.sel]
	mtc0 rt,cd	010000|00100|rt|cd|00000000|sel		GPR[rt]		CPR[cd. sel]	CPR[cd.sel]=GPR[rt]
	syscall	000000|code|001100					CPR[14.0]=PC,CPR[13.0][6:2]=01000,CPR[12.0][1]=1,PC=CPR[15.1]+0x180 并跳转
	Eret	010000|1|0000000000000000000|011000					CPR[12.0][1]=0,PC=CPR[14.0]并跳转

其相较多周期 CPU 的指令集增加 9 条指令如下：

指令名称	汇编指令	功能描述
有符号乘法	mult rs,rt	(HI,LO)=sign(GPR[rs])*sign(GPR[rt])
从 LO 寄存器取值	mflo rd	GPR[rd]=[LO]
从 HI 寄存器取值	mfhi rd	GPR[rd]=[HI]
向 LO 寄存器存值	mtlo rs	[LO]=GPR[rs]
向 HI 寄存器存值	mthi rs	[HI]=GPR[rs]
从 cp0 寄存器取值	mfc0 rt,cs	GPR[rt]=CPR[cs]
向 cp0 寄存器存值	mtc0 rt,cd	CPR[cd]=GPR[rt]
系统调用	syscall	CPR[14.0]=PC, CPR[13.0][6:2]=01000, CPR[12.0][1]=1, PC=CPR[15.1]+0x180
异常返回	eret	CPR[12.0][1]=0, PC=CPR[14.0]

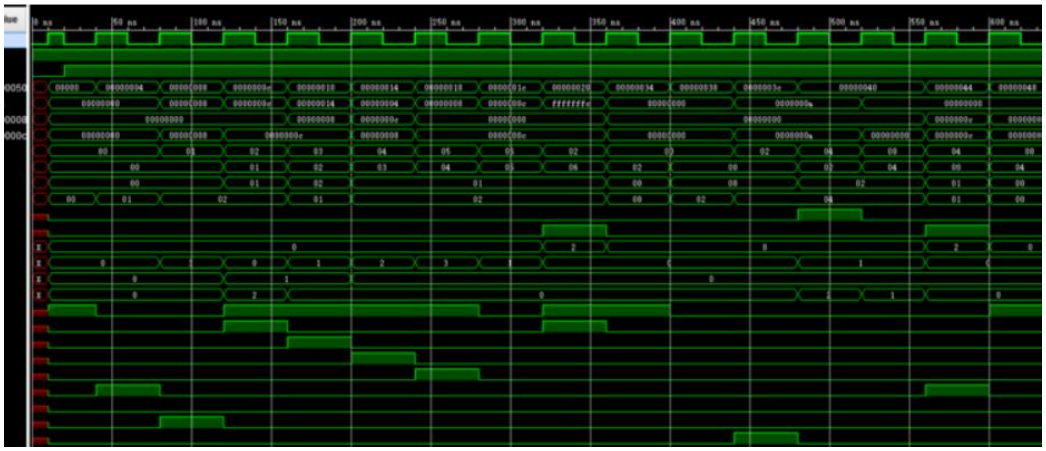
主要是乘法指令，与 HI、LO、协处理器 0（Coprocessor 0, cp0）寄存器传送数据的指令以及特权指令，共实现了 45 条指令。

5、 实验结果分析

下图为部分指令地址较小的 5 级流水线 CPU 测试所用汇编程序：

指令地址	汇编指令	结果描述	机器指令的机器码	
			16 进制	二进制
Exception 入口地址，在 SYSCALL 指令执行后进入此处执行				
00H	sw $1,#0($0)	Mem[0000_0000H] = 0000_0008H	AC010000	1010_1100_0000_0001_0000_0000_0000_0000
04H	sw $2,#4($0)	Mem[0000_0004H] = 0000_0010H	AC020004	1010_1100_0000_0010_0000_0000_0000_0100
08H	sw $3,#8($0)	Mem[0000_0008H] = 0000_0011H	AC030008	1010_1100_0000_0011_0000_0000_0000_1000
0CH	sw $4,#12($0)	Mem[0000_000CH] = 0000_0004H	AC04000C	1010_1100_0000_0100_0000_0000_0000_1100
10H	sw $5,#16($0)	Mem[0000_0010H] = 0000_000DH	AC050010	1010_1100_0000_0101_0000_0000_0001_0000
14H	sw $6,#24($0)	Mem[0000_0018H] = FFFF_FFE2H	AC060018	1010_1100_0000_0110_0000_0000_0001_1000
18H	sw $7,#112($0)	Mem[0000_0070H] = FFFF_FFF3H	AC070070	1010_1100_0000_0111_0000_0000_0111_0000
1CH	sw $25,#116($0)	Mem[0000_0074H] = 0000_0001H	AC190074	1010_1100_0001_1001_0000_0000_0111_0100
20H	sw $13,#24($0)	Mem[0000_0078H] = 0000_0000H	AC0D0078	1010_1100_0000_1101_0000_0000_0111_1000
24H	mfc0 $1, cp0(14.0)	[$1] = 0000_0104H	40017000	0100_0000_0000_0001_0111_0000_0000_0000
28H	addiu $1,$1,#4	[$1] = 0000_0108H	24210004	0010_0100_0010_0001_0000_0000_0000_0100
2CH	mtc0 $1, cp0(14.0)	cp0(14.0) = 0000_0108H	40817000	0100_0000_1000_0001_0111_0000_0000_0000
30H	eret	返回 108H	42000018	0100_0010_0000_0000_0000_0000_0001_1000
CPU 复位地址 0000_0034H				
34H	addiu $1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001_0000_0000_0000_0001
38H	sll $2, $1,#4	[$2] = 0000_0010H	00011100	0000_0000_0000_0001_0001_0001_0000_0000
3CH	addu $3, $2,$1	[$3] = 0000_0011H	00411821	0000_0000_0100_0001_0001_1000_0010_0001
40H	srl $4, $2,#2	[$4] = 0000_0004H	00022082	0000_0000_0000_0010_0010_0000_1000_0010
44H	slti $25,$4,#5	[$25] = 0000_0001H	28990005	0010_1000_1001_1001_0000_0000_0000_0101
48H	bgez $25,#14	跳转到 84H	0721000E	0000_0111_0010_0001_0000_0000_0000_1110
4CH	subu $5, $3,$4	[$5] = 0000_000DH	00642823	0000_0000_0110_0100_0010_1000_0010_0011
50H	sw $5, #20($0)	Mem[0000_0014H] = 0000_000DH	AC050014	1010_1100_0000_0101_0000_0000_0001_0100
54H	nor $6, $5,$2	[$6] = FFFF_FFE2H	00A23027	0000_0000_1010_0010_0011_0000_0010_0111
58H	or $7, $6,$3	[$7] = FFFF_FFF3H	00C33825	0000_0000_1100_0011_0011_1000_0010_0101
5CH	xor $8, $7,$6	[$8] = 0000_0011H	00E64026	0000_0000_1110_0110_0100_0000_0010_0110
60H	beq $8, $3,#2	跳转到 6CH	11030002	0001_0001_0000_0011_0000_0000_0000_0010
64H	sw $8, #28($0)	Mem[0000_001CH] = 0000_0011H	AC08001C	1010_1100_0000_1000_0000_0000_0001_1100
68H	slt $9, $1,$2	不执行	0022482A	0000_0000_0010_0010_0100_1000_0010_1010

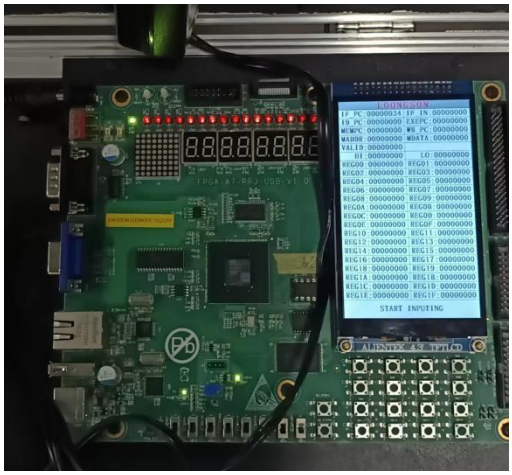
结合测试数据，仿真波形图：



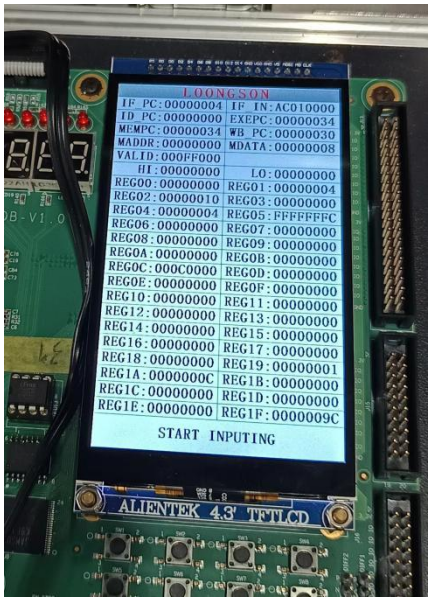
32 个寄存器的值，各级 PC 的值等。并且需要利用触摸功能输入特定数据 RAM 地址，从该 RAM 的调试端口读出数据显示在屏上。

```
#时钟信号连接
set_property PACKAGE_PIN AC19 [get_ports clk]

#脉冲开关，用于输入作为复位信号，低电平有效
set_property PACKAGE_PIN Y3 [get_ports resetn]

#脉冲开关，用于输入作为单步执行的 clk
set_property PACKAGE_PIN Y5 [get_ports btn_clk]
```

指令代码分析：



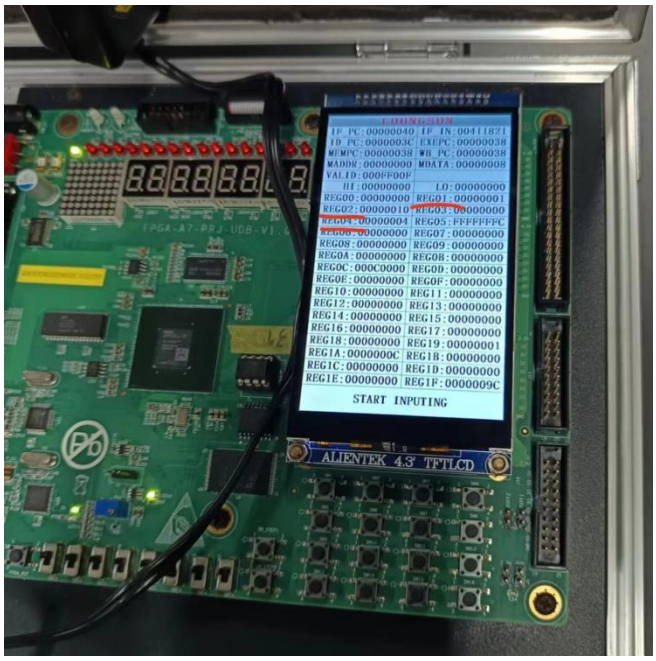
CPU 复位地址 0000_0034H



00H	sw $1,#0($0)	Mem[0000_0000H] = 0000_0008H	AC010000	1010_1100_0000_0001_0000_0000_0000_0000

IF_PC 为 00000004，IF_IN 为 AC010000,该指令为 00H 的机器指令机器码，也可知取指 PC、译码 PC、执行 PC、访存 PC、回写 PC，在地址 00000000 处，数据为 00000008，对应

Mem[0000_0000H]=0000_00008H,汇编指令为 sw $1,#0($0),二进制源码为
1010_1100_0000_0001_0000_0000_0000_0000,可继续 clk 单步执行



34H	addiu $1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001_0000_0000_0000_0001
38H	sll $2, $1,#4	[$2] = 0000_0010H	00011100	0000_0000_0000_0001_0001_0001_0000_0000
40H	srl $4, $2,#2	[$4] = 0000_0004H	00022082	0000_0000_0000_0010_0010_0000_1000_0010

寄存器$1 结果 0000_0001H:指令地址 34H，二进制源码
0010_0100_0000_0001_0000_0000_0000_0001,指令含义：addiu $1，$0,#1,
寄存器$2 结果 0000_0010H:指令地址 38H，二进制源码
0000_0000_0000_0001_0001_0001_0000_0000,指令含义：sll $2,$1,#4
寄存器$4 结果 0000_0004H:指令地址 40H，二进制源码
0000_0000_0000_0010_0010_0000_1000_0010,指令含义：srl $4, $2,#2

6、 总结感想

多周期 CPU 改进成五级流水线 CPU:
流水线是数字系统中一种提高系统稳定性和工作速度的方法，广泛应用于高档 CPU 的架构中。根据 MIPS 处理器的特点，将整体的处理过程分为取指令（IF）、指令译码（ID）、执行（EX）、存储器访问（MEM）和寄存器会写（WB）五级，对应多周期的五个处理阶段。一个指令的执行需要 5 个时钟周期，每个时钟周期的上升沿来临时，此指令所代表的一系列数据和控制信息将转移到下一级处理。
（1） IF 级：取指令部分。
包括指令储存器和 PC 寄存器及其更新模块，负责根据 PC 寄存器的值从指令存储器中取出指令编码和对 PC 的值进行更新。
（2） ID 级：指令译码部分。
根据独处的指令编码形成控制信号和读寄存器堆输出的寄存器的值。

流水线冒险检测也在该级进行，冒险检测电路需要上一条指令的 MemRead，即在检测到冒险条件成立时，冒险检测电路产生 stall 信号清空 ID/EX 寄存器，插入一个流水线气泡。

（3） EX 级：执行部分。

根据指令的编码进行算数或者逻辑运算或者计算条件分支指令的跳转目标地址。此外 LW、SW 指令所用的 RAM 访问地址也是在本级上实现。控制信号有 ALUCode、ALUSrcA、ALUScrB 和 RegDst，根据这些信号确定 ALU 操作、选择两个 ALU 操作数 A、B，并确定目标寄存器。另外，数据转发也在该级完成。数据转发控制电路产生 ForwardA 和 ForwardB 两组控制信号。

（4） MEM 级：存储器访问部分。

只有在执行 LW、SW 指令时才对存储器进行读写，对其他指令只起到一个周期的作用。该级只需存储器写操作允许信号 MemWrite。

（5） WB 级：寄存器堆写回部分。

该级把指令执行的结果回写到寄存器文件中。

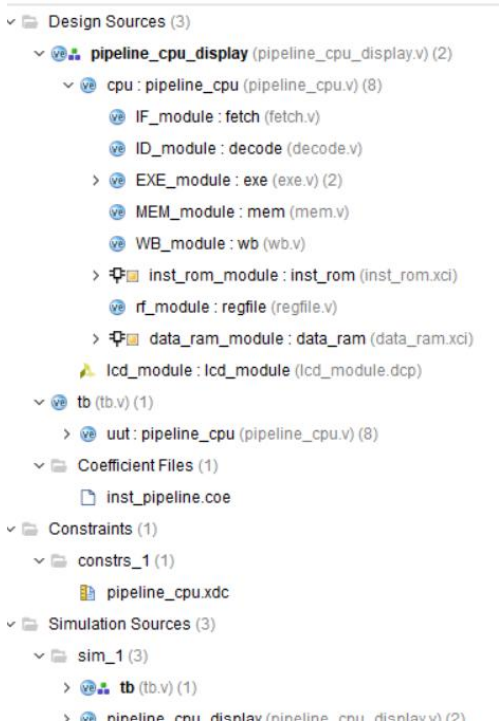
该级设置信号 MemtoReg 和寄存器写操作允许信号 RegWrite，其中 MemtoReg 决定写入寄存器的数据来自于 MEM 级上的缓冲值或来自于 MEM 级上的存储器。

设计的关键和难点在于流水线的控制，有数据相关时就需要堵在流水线中。

延迟槽，MIPS 架构中分支和跳转指令参与计算的 PC 值均为延迟槽指令对应的 PC(即分支跳转指令的 PC+4),需要注意分支跳转指令的偏移量。

2. 由于使用的是同步 IP 核形式的存储器（inst_rom 和 data_ram）,在五级流水线 CPU 的代码中也存在类似的时钟问题/分支跳转问题。

data_ram,inst_rom，



3、现有的五级流水线 CPU 功能比较简单，并且效率较低，请针对现有 CPU 提出一些优化改进思路，包括但不限于调整流水线级数、分支预测、时钟优化、指令扩展等思路。

1.分析静态 5 级流水 CPU 中的流水线阻塞情况，包括数据相关、控制相关、结构相关等，优化流水线设计，尽可能减少流水线阻塞情况，比如前递技术等。

2.对于分支跳转指令，mips 架构中有延迟槽指令的设定，利用这一点，在静态 5 级流水 CPU 中，可实现分支指令永远不阻塞后续指令。

3.此时，分支跳转指令就不需要进行转移猜测了，但大家可以将流水线结构改为 x86 中无延迟槽技术的设定，此时分支跳转指令与后续真正需要执行的指令至少会堵塞一拍，此时可以考虑实现转移预测技术，提升流水线结构。

4.目前课程设计实现的指令存储器和数据存储器是同步读的机制的，故在当前拍数（时钟周期）发出读数据的地址请求时，在下一拍才能获得读的数据，因此取值级和访存级的 load 都需要两拍时间。其实发地址请求在下一拍获得读的数据，明显也是一个可以流水做的工作，可以考虑对流水线设计方案作稍微修改，使得取指级和访存级的 load 不需要多等一拍。