

# 计算机组成原理

## 课程设计报告

学号\_\_\_\_\_18061225\_\_\_\_\_  
姓名\_\_\_\_\_李之龙\_\_\_\_\_  
指导教师\_\_\_\_\_宋书瀛\_\_\_\_\_  
提交日期\_\_\_\_\_2020/7/8\_\_\_\_\_

成绩评价表

报告内容	报告结构	报告最终成绩
<input type="checkbox"/> 丰富正确 <input type="checkbox"/> 基本正确 <input type="checkbox"/> 有一些问题 <input type="checkbox"/> 问题很大	<input type="checkbox"/> 完全符合要求 <input type="checkbox"/> 基本符合要求 <input type="checkbox"/> 有比较多的缺陷 <input type="checkbox"/> 完全不符合要求	
报告与 Project 功能一致性	报告图表	总体评价
<input type="checkbox"/> 完全一致 <input type="checkbox"/> 基本一致 <input type="checkbox"/> 基本不一致	<input type="checkbox"/> 符合规范 <input type="checkbox"/> 基本符合规范 <input type="checkbox"/> 有一些错误 <input type="checkbox"/> 完全不正确	

教师签字:\_\_\_\_\_

# 目录

<u>Project1 VerilogHDL 单周期处理器开发-----</u>	<u>P3</u>
<u>Project2 VerilogHDL 多周期处理器开发-----</u>	<u>P19</u>
<u>Project3 VerilogHDL MIPS 微系统开发（支持设备与中断） -----</u>	<u>P37</u>
<u>收获体会-----</u>	<u>P51</u>

# Project1 VerilogHDL 完成单周期处理器开发

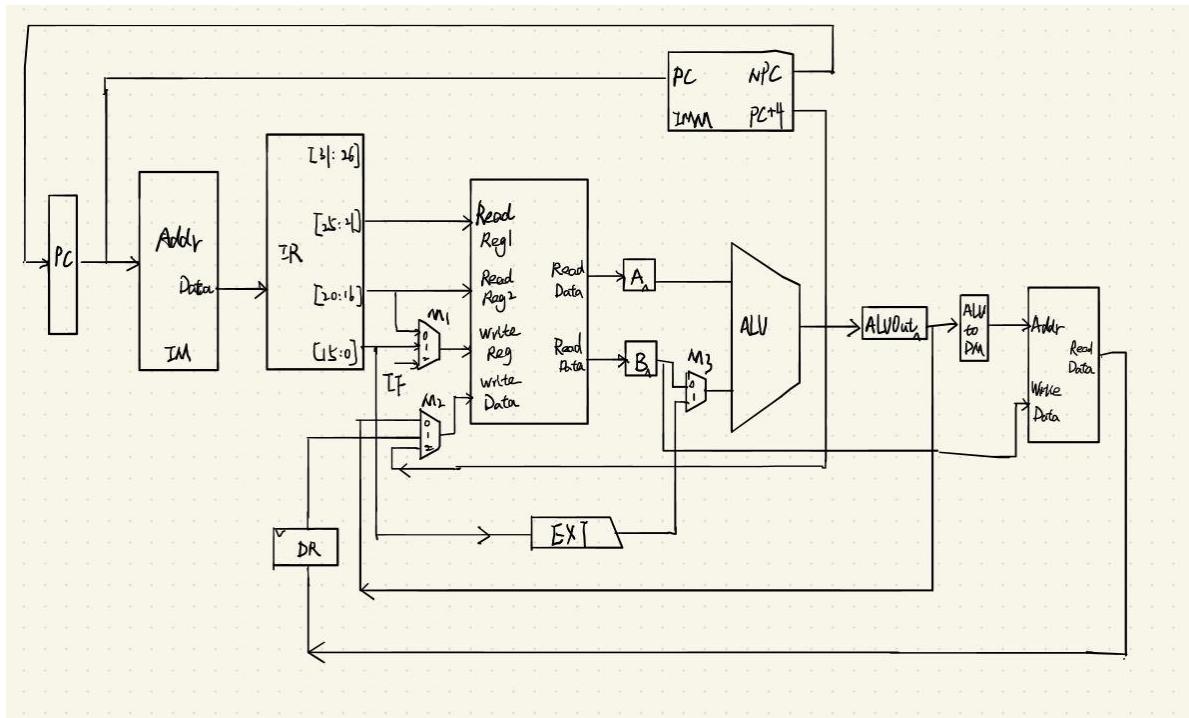
## 一、设计说明

1, 处理器应实现 MIPS-Lite1 指令集。

- a) MIPS-Lite1 = {MIPS-Lite, addi, addiu, slt, jal, jr}。
- b) MIPS-Lite 指令集: addu, subu, ori, lw, sw, beq, lui, j。
- c) addi 应支持溢出, 溢出标志写入寄存器\$30 中第 0 位。

2, 处理器为单周期设计。

## 二、数据通路



## 三、模块定义

1, ifu 模块定义

(1) 基本描述

Ifu 模块主要功能是完成输出当前指令地址并保存下一条指令地址。从大小为 1kb 的指令存储器 IM 中根据当前 pc 读取 32 位 MIPS 指令并且完成指令的译码作为该模块的输出。PC 复位后, 指向 0x0000\_3000, 此处为第一条指令的地址。并且完成普通指令 PC 的更新, 计算下一条指令的地址  $pc=pc+4$ , 以及 j, jr,jal 指令以及 beq

指令的 pc 更新。

## (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号。 1: 复位 0: 无效
npc_sel	I	Beq 指令标识
zero	I	零标志, 表示 rs,rt 寄存器值相等
Aout[31:0]	I	J 指令的跳转, rs 寄存器内的内容
jr	I	Jr 指令标识
jsome	I	J 指令和 jal 指令标识
Pcc[31:0]	O	Jal 指令需要寄存在 31 号寄存器的 PC+4
code[31:0]	O	当前 PC 对应的 32 位 MIPS 指令

## (3) 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时, PC 被设置为 0x0000_3000。
2	根据不同指令更新 PC	在每个 clock 的上升沿根据当前不同的指令更新 PC 1 如果当前指令不是 beq、j、jal、jr 指令, 则 PC PC+4 2 如果当前指令是 beq 指令, 并且 zero 为 0, 则 PC PC+4 如果当前指令是 beq 指令, 并且 zero 为 1, 则 PC PC+sign_ext(Instr[15:0]) 3 如果当前指令是 j 指令, 则 jsome 为 1, 则 PC PC+1[31:28]    insout[25:0](最低两位忽略)。 4 如果当前指令是 jal 指令, 则 jsome 为 1, 则 PC PC+1[31:28]    insout[25:0](最低两位忽略)。 5 如果当前指令是 jr 指令, 则 jr 为 1, 则 pc Rrs。
3	输出当前 PC 指令译码	通过引用 Im_1k 模块根据当前 PC 读取并译码当前指 令为 code[31:0]作为 output
4	输出当前下一条指令 PC+4	pcc 为下一条指令 PC+4, 用于在 jal 指令中回写到 GPR 31 号寄存器中。

## 2.im\_1k 模块定义

### (1) 基本描述

根据输入的 PC 地址, 从指令寄存器中取出相应的指令。

### (2) 模块接口

信号名	方向	描述
addr[9:0]	I	当前 PC 的最后 10 位地址
dout[31:0]	O	当前地址对应的 32 位 MIPS 指令

### (3) 功能定义

序号	功能名称	功能描述
1	根据当前 PC 读取指令	将 32 位 MIPS 指令从 im 指令存储器中取出

## 3.GPR 模块定义

### (1) 基本描述

寄存器组内含 32 个 32 位寄存器、写信号及其他相关逻辑。GPR 按照 rs 和 rt 提供的编号读取两个寄存器内容。当写使能有效时，根据 M1 和 M2 模块将不同的值写入不同的寄存器，当 addi 发生溢出即溢出标志位 of 有效时，将 30 号寄存器置一。每当下降沿到来时，将 0 号寄存器清零，确保 0 号寄存器的值永远是 0.

### (2) 模块接口

信号名	方向	描述
GPRWr	I	读控制信号 1: 写操作允许 2: 写操作禁止
ReadR1[31:0]	I	寄存器 rs 对应的地址
ReadR2[31:0]	I	寄存器 rt 对应的地址
clk	I	时钟信号
WrR[31:0]	I	需要写入的寄存器编号
WrData[31:0]	I	写入数据的输入
of	I	addi 溢出信号
A[31:0]	O	输出寄存器 rs 的内容
B[31:0]	O	输出寄存器 rt 的内容

### (3) 功能定义

序号	功能名称	功能描述
1	读写操作	读取: Rs 作为编号的寄存器内容输出到 A, Rt 作为编号的寄存器内容输出到 B。 写入: 当信号 GPRWr =1 时, WrData 上的数据被写入 WrR 作为编号的寄存器内。
2	时钟信号	clk 仅仅写操作时有效, 表明写入时刻 每当下降沿来临时, 将 0 号寄存器清零, 确保 0 号寄存器的值都是零。
3	溢出置位	当 addi 操作有溢出, 即 of 置 1 时, 将 30 号寄存器置 1
4	数据输出	通过组合逻辑将 A 和 B 的值通过寄存器地址赋予

## 4.ALU 模块定义

### (1) 基本描述

ALU 内含无符号加、减及或等运算，当输入两路数据后，输出相应的运算结果，如果是 lw 或 sw 指令，则计算访存地址。当 addi 溢出时，控制信号 of 置一。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
A[31:0]	I	A 寄存器的值
B[31:0]	I	B 寄存器的值
aluop[2:0]	I	又 controller 产生的计算控制信号 000: 加运算 001: 减运算 010: 与运算 011: 或运算 100: 异或运算 101: 判断两数大小，若 A<B，ALUOut 置 1 110: 判断 addi 是否溢出（双符号位判断）
ALUOut[31:0]	O	32 位数据输出，ALU 模块计算结果
zero	O	ALU 计算结果为 0 标志。（配合 beq 指令使用） 1: 计算结果为 0，两寄存器值相等 0: 计算结果非 0，两寄存器值不同
of	O	addi 溢出标志 1: 溢出 0: 未溢出

### (3) 功能定义

序号	功能名称	功能描述
1	加	A+B
2	减	A-B
3	或	A B
4	异或	A^B
4	判断溢出	当为 addi 指令时，加数与被加数最高位相同，且与结果最高位相反，则发生溢出，此时 of 为 1。
5	判断 slt 回写数据	当 slt 被减数<减数时，比较标志为 1，回写 1；否则为 0，回写 0。
6	判断 beq 是否跳转	当 beq 指令时，若 ALUOut=1 则 zero=0；反之 zero=1。

## 5.ext 模块定义

### (1) 基本描述

将 16 位立即数实现零扩展、符号扩展、和 LUI 指令特殊的低 16 位变成高 16 位然后低 16 位补零而扩展为 32 位立即数。

### (2) 模块接口

信号名	方向	描述
Clk	I	时钟信号
Imm16[15:0]	I	32 位 MIPS 指令中的低 16 位立即数
extop[1:0]	I	由 controller 产生的控制信号 00: 0 扩展 01: 符号扩展 10: lui 变换
Imm32[31:0]	O	扩展后的 32 位数据输出

### (3) 功能定义

序号	功能名称	功能描述
1	0 扩展	高 16 位补 0
2	符号扩展	高 16 位符号扩展
4	Lui 扩展	执行 lui 指令输出结果

## 6.dm\_1k 模块定义

### (1) 基本描述

实现相应的写入及输出功能，使用小端序存储结构。当 lw 指令时，将由 ALU 计算出的 PC 指令后 10 位对应的连续 4 个字节取出作为该模块的输出 dout，将来回写到寄存器。当指令为 sw 时，写使能 we 有效，将对应地址修改为输入中的 din,也就是 rt 寄存器中的值。

### (2) 模块接口

信号名	方向	描述
addr[9:0]	I	数据存储器的地址，去 MIPS 指令的低 10 位对应一个字节
din[31:0]	I	写入数据的输入
we	I	读写控制信号，写使能 1: 写操作
clk	I	时钟信号
dout[31:0]	O	32 位数据输出

### (3) 功能定义

序号	功能名称	功能描述
1	读操作	当指令为 lw 时, 根据输入的寄存器地址取出连续的四个字节内容数据作为 32 位输出 dout
3	写操作	根据输入的地址, 将输入的数据写入相应的地址空间 当 we 高有效对应 sw 指令时, 将 din 写入

## 7.M1,M2,M3 模块定义

### (1) 基本描述

- ①M1:回写地址的选择
- ②M2:回写数据的选择
- ③M3:alu2 位输入 3 数据 2 的选择

### (2) 模块接口

#### 1) M1:

信号名	方向	描述
Clk	I	时钟信号
GPRSel	I	写地址控制信号 00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里
[31:0]in0	I	rt 寄存器
[31:0]in1	I	rd 寄存器
[31:0]in2	I	31 号寄存器, 用于 jal 指令回写
WrReg[31:0]	O	选择回写的寄存器

#### 2) M2:

信号名	方向	描述
Clk	I	时钟信号
WDSel[1:0]	I	选择写入的数据 00: 将 ALUOut 的数据写回 01: 将 DMOOut 的数据写回 10: 将 pcc 地址写回
Inn0[31:0]	I	ALUOut 的数据
Inn1[31:0]	I	DMOOut 的数据
Inn2[31:0]	I	pcc 地址对应 jal 的下一条指令地址
WrData[31:0]	O	需要写回的数据

#### 3) M3

信号名	方向	描述
clk	I	时钟信号

B[31:0]	I	Rt 寄存器的内容
imm32[31:0]	I	经过 ext 扩展模块扩展后的数据
BSel	I	选择 ALU 第二路的输入 0: B, 对应 rt 寄存器的内容 1: 经过 ext 信号扩展的 32 位立即数
M3out[31:0]	O	选择后的值

### (3) 功能定义

序号	功能名称	功能描述
1	gpr 回写地址的选择	00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里对应 jal 指令
2	gpr 回写数据的选择	00: 运算器结果回写 01: 数据存储器输出数据回写 10: pc+4 回写
3	Alu 第 2 位输入数据的选择	0: 寄存器 rt 输出的 32 位数据 1: 扩展后的 32 位数据

## 8.IR 模块定义

### (1) 基本描述

将当前指令拆分译码

### (2) 模块接口

信号名	方向	描述
[31:0]irin	I	当前 PC 对应的 32 位 MIPS 指令
[15:0]imm	O	最后 16 位立即数
[31:0]rs	O	rs 寄存器
[31:0]rt	O	rt 寄存器
[31:0]rd;	O	rd 寄存器
[25:0]jadd	O	用于跳转的当前指令最后 25 位

## 9.IF 模块定义

### (1) 基本描述

判断是否为 jal 指令输出 31 号寄存器

### (2) 模块接口

信号名	方向	描述
Clk	I	时钟信号
jal	I	Jal 指令标志

[31:0]out	O	当指令为 jal 时，对应的 31 号寄存器
-----------	---	------------------------

## 10. control 模块定义

### (1) 基本描述

根据 32 位指令分析出的 opcode 和 funct 对应于相应的指令，分析出每个指令执行的数据通路的相应写使能信号和选择信号。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
code	I	32 位 MIPS 指令
ALU_OP[2:0]	O	ALU 选择信号 000: 加运算 001: 减运算 010: 与运算 011: 或运算 100: 异或运算 101: 判断两数大小 相等: zero 为 1 不相等: zero 为 0 110: 判断 addi 是否溢出
WDSel	O	寄存器写入数据选择控制信号 00: 将 ALUOut 的数据写回 01: 将 DMOut 的数据写回 10: 将 pcc 地址写回
GPRSel	O	寄存器回写寄存器选择控制信号 00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里对应 jal 指令
ExtOp	O	扩展信号 00: 0 扩展 01: 符号扩展 10: lui 变换
GPRWr	O	寄存器写使能
BSel	O	ALU 第二位操作数选择信号
DMWr	O	DM 写使能
jsome	O	J 和 jal 指令控制信号
npc_sel	O	Beq 指令控制信号
jr	O	Jr 指令控制信号
jal	O	Jal 控制信号，用于选择回写 31 号寄存器

#### (4) 单周期控制器真值表

指令	addu	subu	ori	lw	sw	beq	lui	j	addi	addiu	slt	jal	jr
ALU_OP	000	001	011	000	000	001	011	x	110	000	101	x	011
WDSel	00	00	00	01	x	x	000	x	00	00	00	x	x
GPRSel	01	01	00	00	x	0	00	x	00	00	01	x	x
ExtOp	0	0	00	01	01	0	10	x	01	01	0	x	x
GPRWr	1	1	1	1	0	x	1	0	1	1	1	0	x
BSel	0	0	1	1	1	0	1	x	1	1	0	x	x
DMWWr	0	0	0	0	1	0	0	0	0	0	0	0	x
jsome	0	0	0	0	0		0	1	0	0	0	1	x
npc_sel	0	0	0	0	0	1	0	0	0	0	0	0	x
jr	0	0	0	0	0	0	0	0	0	0	0	0	1
jal	0	0	0	0	0	0	0	0	0	0	0	1	0

## 11.mips 模块定义

### (1) 基本描述

起各模块的连接作用。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	置位信号 (恢复初始)

## 四、机器指令描述

指令名称 opcode	操作码 opcode	功能码 funct	功能	指令功能描述
addu	000000	100001	无符号数加	$R[rd] \leftarrow R[rs] + R[rt]$
subu	000000	100011	无符号减	$R[rd] \leftarrow R[rs] - R[rt]$
slt	000000	101010	小于时置位	$R[rd] \leftarrow (GPR[rs] < GPR[rt])$
addi	001000		立即数加法 (支持溢出)	未溢出时: $R[rt] \leftarrow R[rs] + immediate$ 溢出时: 不改变寄存器内的值
addiu	001001		立即数加法 (不支持溢出)	$R[rt] \leftarrow R[rs] + immediate$
ori	001101		立即数或	$R[rt] \leftarrow R[rs]   ZeroExt(imm16)$
lw	100011		取字	$R[rt] \leftarrow MEM[R[rs] + SignExt[imm16]]$
sw	101011		存字	$MEM[R[rs] + SignExt[imm16]] \leftarrow R[rt]$
beq	000100		相等时跳转	$if(R[rs] == R[rt]) then$ $PC \leftarrow PC + 4 + (signExt(imm16)    00)$
lui	001111		取立即数的高位	$R[rt] \leftarrow imm16    0^{16}$
j	000010		无条件跳转	$PC \leftarrow PC \leftarrow PC + 4[31..28]    instr    0^2$
jal	100000		跳转并链接	$$31 = pc + 4$ $PC \leftarrow PC \leftarrow PC + 4[31..28]    instr    0^2$
jr	000000	0010000	跳转至寄存器所指位置	$PC \leftarrow R[rs]$

## 五、测试要求

### 1， 测试程序

机器码	指令	注释
34100001	ori \$16, \$0, 1	#将 1 和\$0 内容做或运算放入\$16
34110003	ori \$17, \$0, 3	#将 3 和\$0 内容做或运算放入\$17
34080001	ori \$8, \$0, 1	#将 1 和\$0 内容做或运算放入\$8
340cabab	ori \$12, \$0,0xabab	#将 0xabab 和\$0 内容做或运算放入\$12
3c0d000a	lui \$13, 10	#将 10 放到\$13 中的高 16 位, 低 16 位补 0
00102021	start: addu \$4, \$0,\$16	#start: 将\$0 内容和\$16 内容无符号相加放入\$4
00082821	addu \$5, \$0,\$8	#将\$0 内容和\$8 内容无符号相加放入\$5
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd
00028021	addu \$16, \$0, \$2	#将\$0 内容和\$2 内容无符号相加放入\$16

02288823	subu \$17,\$17,\$8	#将\$17 内容和\$8 内容无符号相减放入\$17
1211ffff	beq \$16, \$17, start	#将\$16 内容和\$17 内容比较, 若相等则跳转到 start
34080004	ori \$8, \$0,4	#将 4 和\$0 内容做或运算放入\$8
3c017fff	addiu \$24,\$0,0x7fffffff	#将\$0 内容和 0x7fffffff 带符号相加放入\$24
3421ffff		
0001c021		
27090003	addiu \$9,\$24,3	#将\$24 内容和 3 带符号相加放入\$9
270a0005	addiu \$10,\$24,5	#将\$24 内容和 5 带符号相加放入\$10
23160006	addi \$22,\$24,6	#将\$24 内容和 6 带符号相加放入\$22
ad090000	start2: sw \$9, 0(\$8)	#start2: \$9 内容放到以\$8 内容为基地址偏移 0 个字节地址指向的存储器单元
8d0e0000	lw \$14, 0(\$8)	#将以\$8 内容为基地址偏移 0 个字节指向的存储器单元存放数据放入\$14
ad0a0004	sw \$10,4(\$8)	#\$10 内容放到以\$8 内容为基地址偏移 4 个字节地址指向的存储器单元
8d0f0004	lw \$15,4(\$8)	#将以\$8 内容为基地址偏移 4 个字节指向的存储器单元存放数据放入\$15
ad04fffc	sw \$4, -4(\$8)	#\$4 内容放到以\$8 内容为基地址偏移-4 个字节地址指向的存储器单元
8d12fffc	lw \$18, -4(\$8)	#将以\$8 内容为基地址偏移 0-4 个字节指向的存储器单元存放数据放入\$18
00082021	addu \$4,\$0,\$8	#将\$0 内容和\$8 内容无符号相加放入\$4
00092821	addu \$5,\$0,\$9	#将\$0 内容和\$9 内容无符号相加放入\$5
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd
0148c82a	slt \$25,\$10,\$8	#将\$10 内容和\$8 内容比较, \$10 内容小于\$8 内容则\$25 存 1, 否则存 0
13200018	beq \$25, \$0,end2	#将\$25 内容和\$0 内容比较, 若相等则跳转到 end2
0184a02a	slt \$20,\$12,\$4	#将\$12 内容和\$4 内容比较, \$12 内容小于\$4 内容则\$20 存 1, 否则存 0
12800001	beq \$20, \$0, end1	#将\$20 内容和\$0 内容比较, 若相等则跳转到 end1
3c0cffff	lui \$12, 65535	#将 65535 放到\$12 中的高 16 位, 低 16 位补 0
34000001	end1:ori \$0, \$0,1	#end1: 将 1 和\$0 内容做或运算放入\$0
3c13efef	lui \$19, 0xefef	#将 0xefef 放到\$19 中的高 16 位, 低 16 位补 0
3c01abab	addiu \$3,\$0,0xababcdcd	#将\$0 内容和 0xababcdcd 带符号相加放入\$3
3421cdcd		
00011821		
24640002	start3: addiu \$4, \$3, 2	#start3: 将\$3 内容和 2 带符号相加放入\$4
20770005	addi \$23, \$3, 5	#将\$3 内容和 5 带符号相加放入\$23
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd
00024021	addu \$8, \$0, \$2	#将\$0 内容和\$2 内容无符号相加放入\$8
00082021	addu \$4, \$0, \$8	#将\$0 内容和\$8 内容无符号相加放入\$4
00092821	addu \$5, \$0, \$9	#将\$0 内容和\$9 内容无符号相加放入\$5
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd

00024821	addu \$9, \$0, \$2	#将\$0 内容和\$2 内容无符号相加放入\$9
01004821	addu \$9, \$8, \$0	#将\$8 内容和\$0 内容无符号相加放入\$9
3c0a0069	lui \$10, 0x69	#将 0x69 放到\$9 中的高 10 位, 低 16 位补 0
11090001	beq \$8, \$9, start4	#将\$8 内容和\$9 内容比较, 若相等则跳转到 start4
1000fff4	beq \$0, \$0, start3	#将\$0 内容和\$0 内容比较, 若相等则跳转到 start3
08000c36	start4: j end	#start4: 无条件跳转到 end
00851021	newadd: addu \$2, \$4, \$5	#newadd: 将\$4 内容和\$5 内容无符号相加放入\$2
21801234	addi \$0,\$12,0x1234	#将\$12 内容和 0x1234 带符号相加放入\$0
03e00008	jr \$31	#返回到\$31 储存的地址所指向指令
201a5678	end2: addi \$26,\$0,0x5678	#end2: 将\$0 内容和 0x5678 带符号相加放入\$26
	end:	#end:

## 2. 预期测试结果

### (1) 寄存器值

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0xababcd
\$v0	2	0xababdd3
\$v1	3	0xababcd
\$a0	4	0x2babdd1
\$a1	5	0x80000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x2babdd1
\$t1	9	0x2babdd1
\$t2	10	0x00690000
\$t3	11	0x00000000
\$t4	12	0x0000abab
\$t5	13	0x0000a000
\$t6	14	0x80000002
\$t7	15	0x80000004
\$s0	16	0x00000003
\$s1	17	0x00000001
\$s2	18	0x00000002
\$s3	19	0xefef0000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0xababdd2
\$t8	24	0x7fffffff
\$t9	25	0x00000001
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x00001800
\$sp	29	0x00002ffc
\$fp	30	0x00000000
\$ra	31	0x000030b0
pc		0x0000030d8
hi		0x00000000
lo		0x00000000

### (2) 数据储存器值

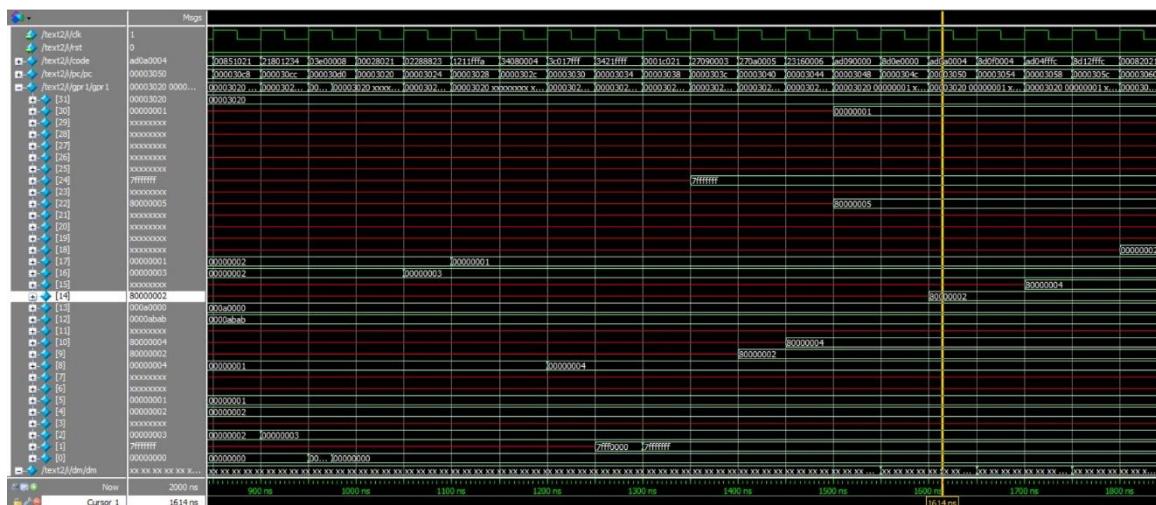
Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000002	0x80000002	0x00000004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000001a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

### 3. 实际测试结果（分步显示以及最终结果）



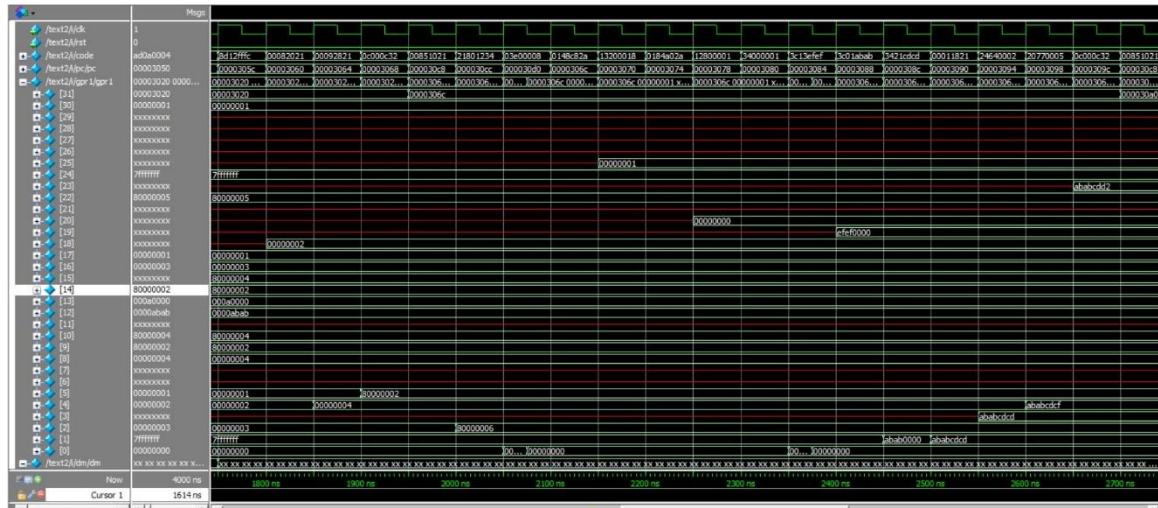
寄存器值 (0-900ns)

说明：前四条指令分别给 16、17、8、12 寄存器赋值，说明 ori 指令正确。Lui \$13,10。之后 13 号寄存器中值变为 000a0000，说明正确。4、5 号寄存器中的值分别变为 1,1 说明 addu 指令正常。jal 指令执行时，31 号寄存器中存 00003020（为下一个指令地址）。到 jr 指令执行时，pc 从 000030d0 变为 00003020，说明 jr 正确，可正常完成跳转。到 beq 指令时，16、17 号寄存器分别存 2,2，跳到 start 标签处（即 pc=00003014）则 beq 指令正确。



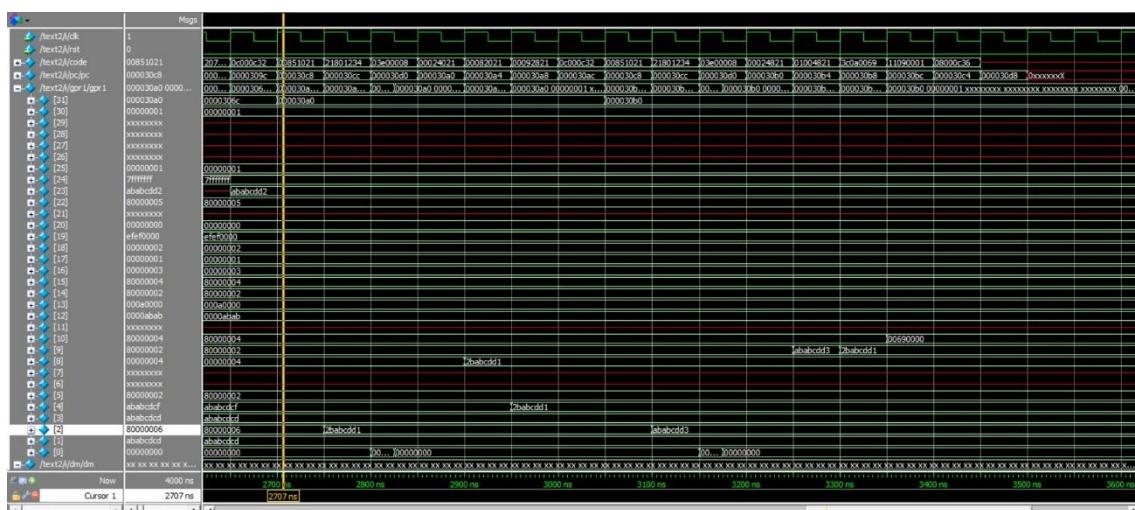
寄存器值 (900-1800ns)

说明：等到第二次 beq 指令时，此时 16、17 号寄存器中的值为 3、1，则不发生跳转。之后的 addiu 指令与 addu 同理，证明其正确。Sw 指令将 9 号寄存器中值存到 4 地址中，见下图 1550ns 处，从地址 4-7 处存 80000002（小端序）。之后 lw 指令将相同地址的值拿出存到 14 号寄存器中，也证明其正确。之后出现的指令已全部证明正确过。



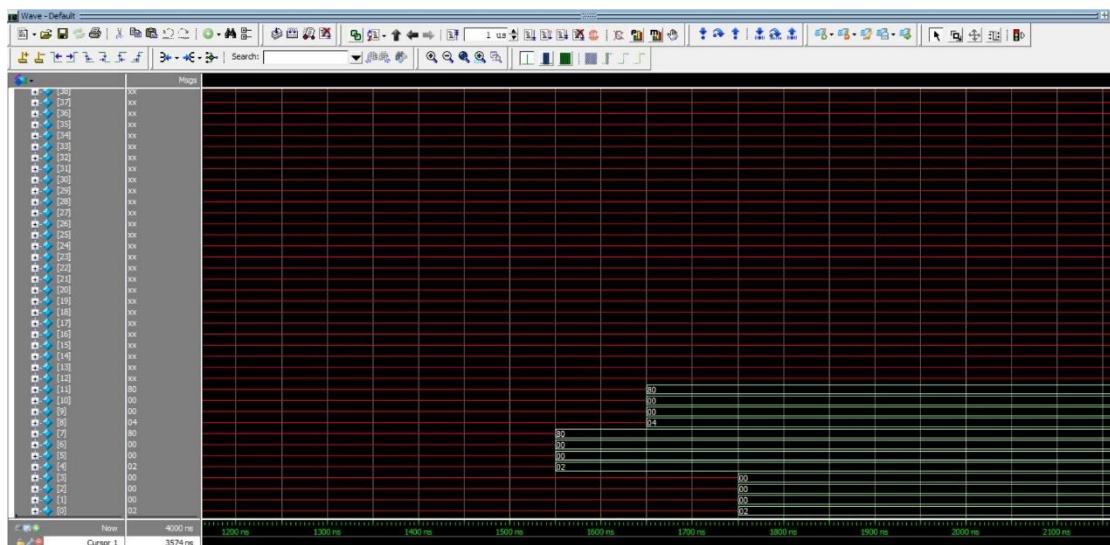
寄存器值 (1800-2700ns)

说明：基本上述所有已经证明正确过



寄存器值 (2700-3500ns)

说明：需要证明其正确的还有 3400ns 处的 j 指令，地址从 000030c4 跳转到 000030d8，其正确。



储存器值（只有 1550-1750ns 的时候发生变化）

[12]	xx
[11]	80
[10]	00
[9]	00
[8]	04
[7]	80
[6]	00
[5]	00
[4]	02
[3]	00
[2]	00
[1]	00
[0]	02

储存器的最终值

说明：结果与 MARs 环境验证完全相同，证明其正确

-	/text2/i/gpr1/gpr1	000030b0 0000...
+	[31]	000030b0
+	[30]	00000001
+	[29]	xxxxxxxx
+	[28]	xxxxxxxx
+	[27]	xxxxxxxx
+	[26]	xxxxxxxx
+	[25]	00000001
+	[24]	7fffffff
+	[23]	ababcd2
+	[22]	80000005
+	[21]	xxxxxxxx
+	[20]	00000000
+	[19]	efef0000
+	[18]	00000002
+	[17]	00000001
+	[16]	00000003
+	[15]	80000004
+	[14]	80000002
+	[13]	000a0000
+	[12]	0000abab
+	[11]	xxxxxxxx
+	[10]	00690000
+	[9]	2babcd1
+	[8]	2babcd1
+	[7]	xxxxxxxx
+	[6]	xxxxxxxx
+	[5]	80000002
+	[4]	2babcd1
+	[3]	ababcdcd
+	[2]	ababcd3
+	[1]	ababcdcd
+	[0]	00000000

### 寄存器组的最终数值

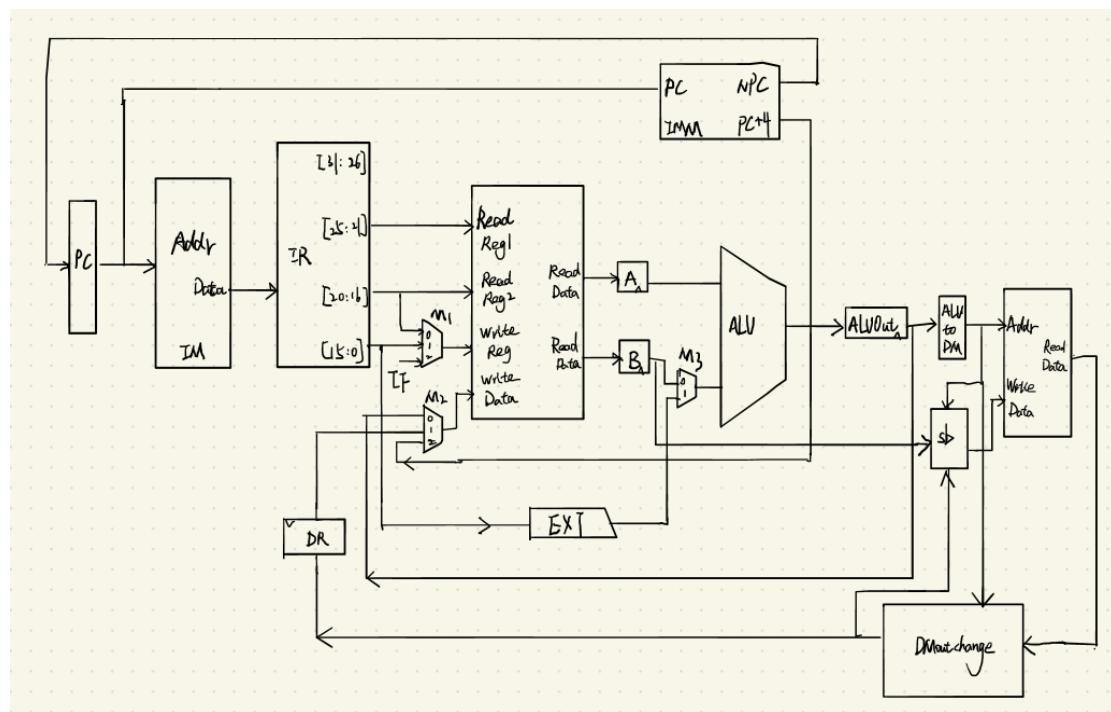
说明：结果与 MARs 环境验证完全相同，证明其正确

# Project2 VerilogHDL 完成多周期处理器开发

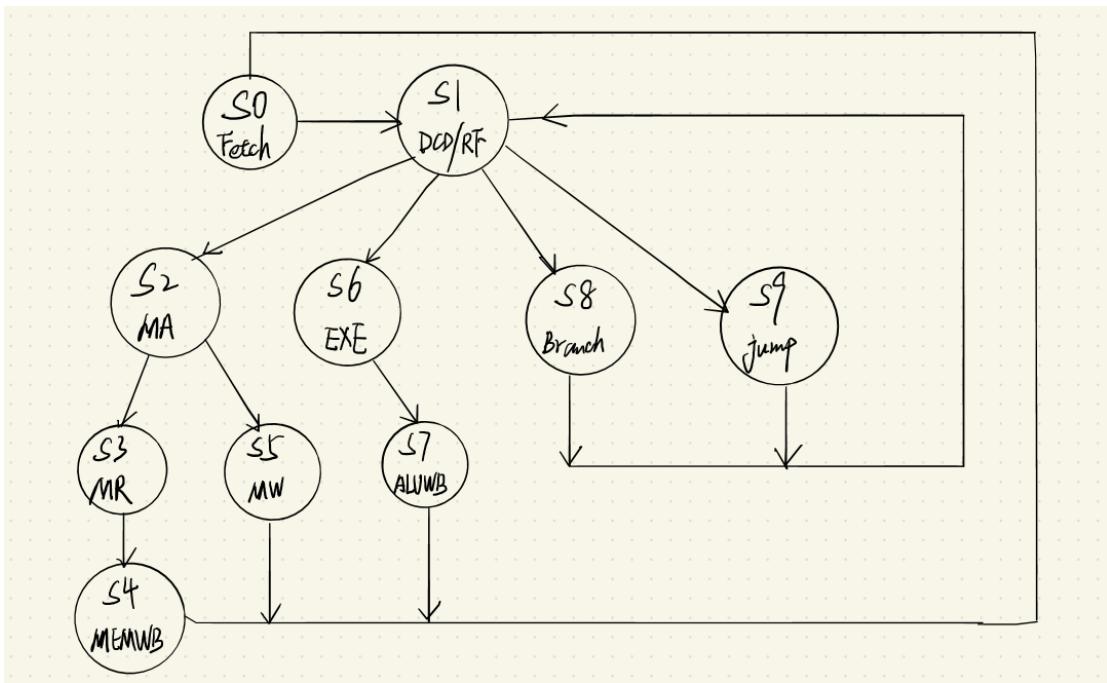
## 一、设计说明

1. 处理器应实现 MIPS-Lite2 指令集。
  - a) MIPS-Lite2 = {MIPS-Lite1, lb, sb}。
  - b) MIPS-Lite1 = {addu, subu, ori, lw, sw, beq, j, lui, addi, addiu, slt, jal, jr }。
  - c) addi 应支持溢出，溢出标志写入寄存器\$30 中第 0 位。
2. 处理器为多周期设计。

## 二、数据通路



## 三、多周期状态图



## 四、模块定义

### 1, ifu 模块定义

#### (1) 基本描述

Ifu 模块主要功能是完成输出当前指令地址并保存下一条指令地址。从大小为 1kb 的指令存储器 IM 中根据当前 pc 读取 32 位 MIPS 指令并且完成指令的译码作为该模块的输出。PC 复位后，指向 0x0000\_3000，此处为第一条指令的地址。并且完成普通指令 PC 的更新，计算下一条指令的地址  $pc=pc+4$ ，以及 j, jr,jal 指令以及 beq 指令的 pc 更新。

#### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
enpc	I	Pc 更新信号 1: 可以更新 2: 不能更新
reset	I	复位信号。 1: 复位 0: 无效
npc_sel	I	Beq 指令标识
zero	I	零标志，表示 rs,rt 寄存器值相等
Aout[31:0]	I	J 指令的跳转，rs 寄存器内的内容
jr	I	Jr 指令标识

jsome	I	J 指令和 jal 指令标识
Pcc[31:0]	O	Jal 指令需要寄存在 31 号寄存器的 PC+4
code[31:0]	O	当前 PC 对应的 32 位 MIPS 指令

### (3) 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时, PC 被设置为 0x0000_3000。
2	根据不同指令更新 PC	<p>在每个 clock 的上升沿根据当前不同的指令更新 PC</p> <p>6 如果当前指令不是 beq、j、jal、jr 指令, 则 PC PC+4</p> <p>7 如果当前指令是 beq 指令, 并且 zero 为 0, 则 PC PC+4</p> <p>如果当前指令是 beq 指令, 并且 zero 为 1, 则 PC PC+sign_ext(Instr[15:0])</p> <p>8 如果当前指令是 j 指令, 则 jsome 为 1, 则 PC PC+1[31:28]    insout[25:0](最低两位忽略)。</p> <p>9 如果当前指令是 jal 指令, 则 jsome 为 1, 则 PC PC+1[31:28]    insout[25:0](最低两位忽略)。</p> <p>10 如果当前指令是 jr 指令, 则 jr 为 1, 则 pc Rrs。</p>
3	输出当前 PC 指令译码	通过引用 Im_1k 模块根据当前 PC 读取并译码当前指令为 code[31:0]作为 output
4	输出当前下一条指令 PC+4	pcc 为下一条指令 PC+4, 用于在 jal 指令中回写到 GPR 31 号寄存器中。

## 2.im\_1k 模块定义

### (1) 基本描述

根据输入的 PC 地址, 从指令寄存器中取出相应的指令。

### (2) 模块接口

信号名	方向	描述
addr[9:0]	I	当前 PC 的最后 10 位地址
dout[31:0]	O	当前地址对应的 32 位 MIPS 指令

### (3) 功能定义

序号	功能名称	功能描述
1	根据当前 PC 读取指令	将 32 位 MIPS 指令从 im 指令存储器中取出

## 3.GPR 模块定义

### (1) 基本描述

寄存器组内含 32 个 32 位寄存器、写信号及其他相关逻辑。GPR 按照 rs 和 rt 提供的编号读取两个寄存器内容。当写使能有效时，根据 M1 和 M2 模块将不同的值写入不同的寄存器，当 addi 发生溢出即溢出标志位 of 有效时，将 30 号寄存器置一。每当下降沿到来时，将 0 号寄存器清零，确保 0 号寄存器的值永远是 0.

### (2) 模块接口

信号名	方向	描述
GPRWr	I	读控制信号 1: 写操作允许 2: 写操作禁止
ReadR1[31:0]	I	寄存器 rs 对应的地址
ReadR2[31:0]	I	寄存器 rt 对应的地址
clk	I	时钟信号
WrR[31:0]	I	需要写入的寄存器编号
WrData[31:0]	I	写入数据的输入
of	I	addi 溢出信号
A[31:0]	O	输出寄存器 rs 的内容
B[31:0]	O	输出寄存器 rt 的内容

### (3) 功能定义

序号	功能名称	功能描述
1	读写操作	读取：Rs 作为编号的寄存器内容输出到 A, Rt 作为编号的寄存器内容输出到 B。 写入：当信号 GPRWr =1 时，WrData 上的数据被写入 WrR 作为编号的寄存器内。
2	时钟信号	clk 仅仅写操作时有效，表明写入时刻 每当下降沿来临时，将 0 号寄存器清零，确保 0 号寄存器的值都是零。
3	溢出置位	当 addi 操作有溢出，即 of 置 1 时，将 30 号寄存器置 1
4	数据输出	通过组合逻辑将 A 和 B 的值通过寄存器地址赋予

## 4.ALU 模块定义

### (1) 基本描述

ALU 内含无符号加、减及或等运算，当输入两路数据后，输出相应的运算结果，如果是 lw 或 sw 指令，则计算访存地址。当 addi 溢出时，控制信号 of 置一。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
A[31:0]	I	A 寄存器的值
B[31:0]	I	B 寄存器的值

aluop[2:0]	I	由 controller 产生的计算控制信号 000: 加运算 001: 减运算 010: 与运算 011: 或运算 100: 异或运算 101: 判断两数大小, 若 A<B, ALUOut 置 1 110: 判断 addi 是否溢出 (双符号位判断)
ALUOut[31:0]	O	32 位数据输出, ALU 模块计算结果
zero	O	ALU 计算结果为 0 标志。(配合 beq 指令使用) 1: 计算结果为 0, 两寄存器值相等 0: 计算结果非 0, 两寄存器值不同
of	O	addi 溢出标志 1: 溢出 0: 未溢出

### (3) 功能定义

序号	功能名称	功能描述
1	加	A+B
2	减	A-B
3	或	A B
4	异或	A^B
4	判断溢出	当为 addi 指令时, 加数与被加数最高位相同, 且与结果最高位相反, 则发生溢出, 此时 of 为 1。
5	判断 slt 回写数据	当 slt 被减数<减数时, 比较标志为 1, 回写 1; 否则为 0, 回写 0。
6	判断 beq 是否跳转	当 beq 指令时, 若 ALUOut=1 则 zero=0; 反之 zero=1。

## 5.ext 模块定义

### (1) 基本描述

将 16 位立即数实现零扩展、符号扩展、和 LUI 指令特殊的低 16 位变成高 16 位然后低 16 位补零而扩展为 32 位立即数。

### (2) 模块接口

信号名	方向	描述
Clk	I	时钟信号
Imm16[15:0]	I	32 位 MIPS 指令中的低 16 位立即数
extop[1:0]	I	由 controller 产生的控制信号 00: 0 扩展

		01: 符号扩展 10: lui 变换
Imm32[31:0]	O	扩展后的 32 位数据输出

### (3) 功能定义

序号	功能名称	功能描述
1	0 扩展	高 16 位补 0
2	符号扩展	高 16 位符号扩展
4	Lui 扩展	执行 lui 指令输出结果

## 6.dm\_1k 模块定义

### (1) 基本描述

实现相应的写入及输出功能，使用小端序存储结构。当 lw 指令时，将由 ALU 计算出的 PC 指令后 10 位对应的连续 4 个字节取出作为该模块的输出 dout，将来回写到寄存器。当指令为 sw 时，写使能 we 有效，将对应地址修改为输入中的 din,也就是 rt 寄存器中的值。

### (2) 模块接口

信号名	方向	描述
addr[9:0]	I	数据存储器的地址，去 MIPS 指令的低 10 位对应一个字节
din[31:0]	I	写入数据的输入
we	I	读写控制信号，写使能 1: 写操作
clk	I	时钟信号
dout[31:0]	O	32 位数据输出

### (3) 功能定义

序号	功能名称	功能描述
1	地址的规格化	当为 sb 指令时，ALUOut 出来的地址一般不是 4 的倍数，此时，dm 会取出具体地址所对应的整个字，再替换其中的字节就可以再次写入 dm 中，因此需要对 ALU 直接传入的地址进行规格化（比如地址为 7 的话，规格化完就是 4）保证以这个字节对应整个字的第一个字节为基地址，确保其有序。
2	读操作	当指令为 lw 或 lb 时，根据输入的寄存器地址取出连续的四个字节内容数据作为 32 位输出 dout
3	写操作	根据输入的地址，将输入的数据写入相应的地址空间 当 we 高有效对应 sw 或是 sb 指令时，将 din 写入

## 7. sb 模块定义

### (1) 基本描述

根据地址从 DM 中提取整个字的内容，若为 sb 指令则再根据地址更换其中字节的内容，之后重新填回 dm 中，形成 dm 中数据的更换；若为 sw 指令，则直接根据地址写入。

### (2) 模块接口

信号名	方向	描述
Clk	I	时钟信号
Sb	I	写地址控制信号 0：不执行 sb 1：执行 sb
[31:0]DMOut	I	DM 出来的这个字的值
[31:0]Bout	I	来自 B 寄存器的值
[9:0]addr	I	字所在的地址，便于寻找字节
SBOut[31:0]	O	模块的输出值

### (3) 功能描述

序号	功能名称	功能描述
1	往 DM 写字节	得到要写的地址的字值，改变对应地址的内容以后放回 DM。

## 8.doutchange(lb)模块定义

### (1) 基本描述

从 DM 里拿出字节写回寄存器。

### (2) 模块接口

信号名	方向	描述
lb	I	写地址控制信号 0：不执行 lb 1：执行 lb
[31:0]dout	I	来自 DM 的值
[9:0]addr	I	字所在的地址，便于寻找字节
doutt[31:0]	O	模块的输出值

### (3) 功能描述

序号	功能名称	功能描述
1	往寄存器写字节	若为 lb 指令根据 dm 中加工过的地址拿出整个字的内容，再根据原本地址，选取这个字中具体的某个字节，进行符号扩展后，传回 gpr 中；若是 lw 指令则直接传回 gpr 中。

## 9.register 模块定义

### (1) 模块描述

作为锁存器，及传递数据。

### (2) 模块接口

信号名	方向	描述
Clk	I	时钟信号
[31:0]din	I	写入里面保存的值
dout[31:0]	O	模块的输出值

### (3) 功能描述

序号	功能名称	功能描述
1	锁存	通过时钟信号控制的锁存器。用于 A、B、ALUOut、dr 寄存器。

## 10.M1,M2.M3 模块定义

### (1) 基本描述

- ①M1:回写地址的选择
- ②M2:回写数据的选择
- ③M3:alu2 位输入 3 数据 2 的选择

### (2) 模块接口

1) M1:

信号名	方向	描述
Clk	I	时钟信号
GPRSel	I	写地址控制信号 00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里
[31:0]in0	I	rt 寄存器
[31:0]in1	I	rd 寄存器
[31:0]in2	I	31 号寄存器, 用于 jal 指令回写
WrReg[31:0]	O	选择回写的寄存器

2) M2:

信号名	方向	描述
Clk	I	时钟信号
WDSel[1:0]	I	选择写入的数据 00: 将 ALUOut 的数据写回 01: 将 DMOut 的数据写回 10: 将 pcc 地址写回
Inn0[31:0]	I	ALUOut 的数据
Inn1[31:0]	I	DMOut 的数据
Inn2[31:0]	I	pcc 地址对应 jal 的下一条指令地址
WrData[31:0]	O	需要写回的数据

3) M3

信号名	方向	描述
clk	I	时钟信号
B[31:0]	I	Rt 寄存器的内容
imm32[31:0]	I	经过 ext 扩展模块扩展后的数据
BSel	I	选择 ALU 第二路的输入 0: B, 对应 rt 寄存器的内容 1: 经过 ext 信号扩展的 32 位立即数
M3out[31:0]	O	选择后的值

(3) 功能定义

序号	功能名称	功能描述
1	gpr 回写地址的选择	00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里对应 jal 指令
2	gpr 回写数据的选择	00: 运算器结果回写 01: 数据存储器输出数据回写 10: pc+4 回写
3	Alu 第 2 位输入数据的选择	0: 寄存器 rt 输出的 32 位数据 1: 扩展后的 32 位数据

## 11.ir 模块定义

### (1) 基本描述

将当前指令拆分译码

### (2) 模块接口

信号名	方向	描述
[31:0]irin	I	当前 PC 对应的 32 位 MIPS 指令
[15:0]imm	O	最后 16 位立即数
[31:0]rs	O	rs 寄存器
[31:0]rt	O	rt 寄存器
[31:0]rd;	O	rd 寄存器
[25:0]jadd	O	用于跳转的当前指令最后 25 位

## 12.IF 模块定义

### (1) 基本描述

判断是否为 jal 指令输出 31 号寄存器

### (2) 模块接口

信号名	方向	描述
Clk	I	时钟信号
jal	I	Jal 指令标志
[31:0]out	O	当指令为 jal 时, 对应的 31 号寄存器

## 13. control 模块定义

### (1) 基本描述

根据 32 位指令分析出的 opcode 和 funct 对应于相应的指令, 分析出每个指令执行的数据通路的相应写使能信号和选择信号。多周期控制信号, 分成以下的不同状态,

使不同的指令进行跳转（根据老师给的进行改编）：

S0 (Fetch/DCD)	取址、译码
S1 (RF/DCD)	机器码缓存
S2 (MA)	sw sb lw lb 分支
S3 (MR)	访问存储器
S4 (MEMWB)	写回 gpr
S5 (MW)	写 DM
S6 (EXE)	执行 (ALU 计算等)
S7 (ALUWB)	ALU 写回 gpr
S8 (Branch)	Beq 跳转指令 (包括译码)
S9 (jump)	J jal jr 指令 (包括译码)

## (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
code	I	32 位 MIPS 指令
ALU_OP[2:0]	O	ALU 选择信号 000: 加运算 001: 减运算 010: 与运算 011: 或运算 100: 异或运算 101: 判断两数大小 相等: zero 为 1 不相等: zero 为 0 110: 判断 addi 是否溢出
WDSel	O	寄存器写入数据选择控制信号 00: 将 ALUOut 的数据写回 01: 将 DMOut 的数据写回 10: 将 pcc 地址写回
GPRSel	O	寄存器回写寄存器选择控制信号 00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里对应 jal 指令
ExtOp	O	扩展信号 00: 0 扩展 01: 符号扩展 10: lui 变换
GPRWr	O	寄存器写使能
BSel	O	ALU 第二位操作数选择信号
DMWr	O	DM 写使能
jsome	O	J 和 jal 指令控制信号

npc_sel	O	Beq 指令控制信号
jr	O	Jr 指令控制信号
jal	O	Jal 控制信号，用于选择回写 31 号寄存器
sb	O	sb 指令控制信号
lb	O	lb 指令控制信号
enpc	O	Pc 值更新控制信号

## 11.mips 模块定义

### (1) 基本描述

起各模块的连接作用。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	置位信号（恢复初始）

## 五、机器指令描述

指令名称	操作码 opcode	功能码 funct	功能	指令功能描述
addu	000000	100001	无符号数加	$R[rd] \leftarrow R[rs]+R[rt]$
subu	000000	100011	无符号减	$R[rd] \leftarrow R[rs]-R[rt]$
slt	000000	101010	小于时置位	$R[rd] \leftarrow (GPR[rs] < GPR[rt])$
addi	001000		立即数加法 (支持溢出)	未溢出时: $R[rt] \leftarrow R[rs]+immdiate$ 溢出时: 不改变寄存器内的值
addiu	001001		立即数加法 (不支持溢出)	$R[rt] \leftarrow R[rs]+immdiate$
ori	001101		立即数或	$R[rt] \leftarrow R[rs]   ZeroExt(imm16)$
lw	100011		取字	$R[rt] \leftarrow MEM\{R[rs]+SignExt[imm16]\}$
sw	101011		存字	$MEM\{R[rs]+SignExt[imm16]\} \leftarrow R[rt]$
beq	000100		相等时跳转	$if(R[rs]==R[rt]) then$ $PC \leftarrow PC+4+(signExt(imm16)   00)$
lui	001111		取立即数的高位	$R[rt] \leftarrow imm16   0^{16}$
j	000010		无条件跳转	$PC \leftarrow PC \leftarrow PC+4[31..28]    instr    0^2$
jal	100000		跳转并链接	$$31=pc+4$ $PC \leftarrow PC \leftarrow PC+4[31..28]    instr    0^2$
jr	000000	0010000	跳转至寄存器所指位置	$PC \leftarrow R[rs]$
sb	101000		存字节	$MEM\{R[rs]+SignExt[imm16]\} \leftarrow R[rt][7:0]$
lb	100000		取字节	$GPR[rt] \leftarrow memory[GPR[base] + offset]$

## 六、测设要求

### 1， 测试程序

机器码	指令	注释
-----	----	----

34100001	ori \$16, \$0, 1	#将 1 和\$0 内容做或运算放入\$16
34110003	ori \$17, \$0, 3	#将 3 和\$0 内容做或运算放入\$17
34080001	ori \$8, \$0, 1	#将 1 和\$0 内容做或运算放入\$8
340cabab	ori \$12, \$0,0xabab	#将 0xabab 和\$0 内容做或运算放入\$12
3c0d000a	lui \$13, 10	#将 10 放到\$13 中的高 16 位, 低 16 位补 0
00102021	start: addu \$4, \$0,\$16	#start: 将\$0 内容和\$16 内容无符号相加放入\$4
00082821	addu \$5, \$0,\$8	#将\$0 内容和\$8 内容无符号相加放入\$5
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd
00028021	addu \$16, \$0, \$2	#将\$0 内容和\$2 内容无符号相加放入\$16
02288823	subu \$17,\$17,\$8	#将\$17 内容和\$8 内容无符号相减放入\$17
1211ffff	beq \$16, \$17, start	#将\$16 内容和\$17 内容比较, 若相等则跳转到 start
34080004	ori \$8, \$0,4	#将 4 和\$0 内容做或运算放入\$8
3c017fff	addiu \$24,\$0,0x7fffffff	#将\$0 内容和 0x7fffffff 带符号相加放入\$24
3421ffff	ori \$1,\$1,0x0000ffff	#将 0xffff 和\$1 内容做或运算放入\$1
0001c021	addu \$24,\$0,\$1	#将\$0 内容和\$1 内容无符号相加放入\$24
27090003	addiu \$9,\$24,3	#将\$24 内容和 3 带符号相加放入\$9
270a0005	addiu \$10,\$24,5	#将\$24 内容和 5 带符号相加放入\$10
23160006	addi \$22,\$24,6	#将\$24 内容和 6 带符号相加放入\$22
00000021	addu \$0,\$0,\$0	#\$0+\$0 放到\$0 寄存器
ad09fffc	start2: sw \$9, -4(\$8)	#start2: \$9 内容放到以\$8 内容为基址偏移 0 个字节地址指向的存储器单元
ad010000	sw \$1, 0(\$8)	#将\$1 的内容放到以\$8 内容为基址偏移 0 个字节的位置
810e0003	lb \$14, 3(\$8)	#将以\$8 内容为基址偏移 3 个字节指向的存储器单元存放数据放入\$14
a10c0007	sb \$12,7(\$8)	#\$12 内容放到以\$8 内容为基址偏移 7 个字节地址指向的存储器单元
8d0f0004	lw \$15,4(\$8)	#将以\$8 内容为基址偏移 4 个字节指向的存储器单元存放数据放入\$15
a104ffffd	sb \$4, -3(\$8)	#将\$4 的内容放到以\$8 内容为基址偏移-3 个字节的位置
8112ffff	lb \$18, -1(\$8)	#将以\$8 内容为基址偏移-1 个字节指向的存储器单元存放数据放入\$18
00082021	addu \$4,\$0,\$8	#将\$0 内容和\$8 内容无符号相加放入\$4
00092821	addu \$5,\$0,\$9	#将\$0 内容和\$9 内容无符号相加放入\$5
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd
0148c82a	slt \$25,\$10,\$8	#将\$10 内容和\$8 内容比较, \$10 内容小于\$8 内容则\$25 存 1, 否则存 0
13200018	beq \$25, \$0,end2	#将\$25 内容和\$0 内容比较, 若相等则跳转到 end2
0184a02a	slt \$20,\$12,\$4	#将\$12 内容和\$4 内容比较, \$12 内容小于\$4 内容则\$20 存 1, 否则存 0
12800001	beq \$20, \$0, end1	#将\$20 内容和\$0 内容比较, 若相等则跳转到 end1
3c0cffff	lui \$12, 65535	#将 65535 放到\$12 中的高 16 位, 低 16 位补 0

34000001	end1:ori \$0, \$0,1	#end1: 将 1 和\$0 内容做或运算放入\$0
3c13fefef	lui \$19, 0xefef	#将 0xefef 放到\$19 中的高 16 位, 低 16 位补 0
3c01abab	addiu \$3,\$0,0xababcdcd	#将\$0 内容和 0xababcdcd 带符号相加放入\$3
3421cdcd	ori \$1,\$1,0x0000cdcd	#将 0xcdcd 和\$1 内容做或运算放入\$1
00011821	addu \$3,\$0,\$1	#将\$0 内容和\$1 内容无符号相加放入\$3
24640002	start3: addiu \$4, \$3, 2	#start3: 将\$3 内容和 2 带符号相加放入\$4
20770005	addi \$23, \$3, 5	#将\$3 内容和 5 带符号相加放入\$23
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd
00024021	addu \$8, \$0, \$2	#将\$0 内容和\$2 内容无符号相加放入\$8
00082021	addu \$4, \$0, \$8	#将\$0 内容和\$8 内容无符号相加放入\$4
00092821	addu \$5, \$0, \$9	#将\$0 内容和\$9 内容无符号相加放入\$5
0c000c32	jal newadd	#将下一条指令的地址放入\$31 并跳转到 newadd
00024821	addu \$9, \$0, \$2	#将\$0 内容和\$2 内容无符号相加放入\$9
01004821	addu \$9, \$8, \$0	#将\$8 内容和\$0 内容无符号相加放入\$9
3c0a0069	lui \$10, 0x69	#将 0x69 放到\$9 中的高 10 位, 低 16 位补 0
11090001	beq \$8, \$9, start4	#将\$8 内容和\$9 内容比较, 若相等则跳转到 start4
1000ffff4	beq \$0, \$0, start3	#将\$0 内容和\$0 内容比较, 若相等则跳转到 start3
08000c36	start4: j end	#start4: 无条件跳转到 end
00851021	newadd: addu \$2, \$4, \$5	#newadd: 将\$4 内容和\$5 内容无符号相加放入\$2
21801234	addi \$0,\$12,0x1234	#将\$12 内容和 0x1234 带符号相加放入\$0
03e00008	jr \$31	#返回到\$31 储存的地址所指向指令
201a5678	end2: addi \$26,\$0,0x5678	#end2: 将\$0 内容和 0x5678 带符号相加放入\$26
	end:	#end:

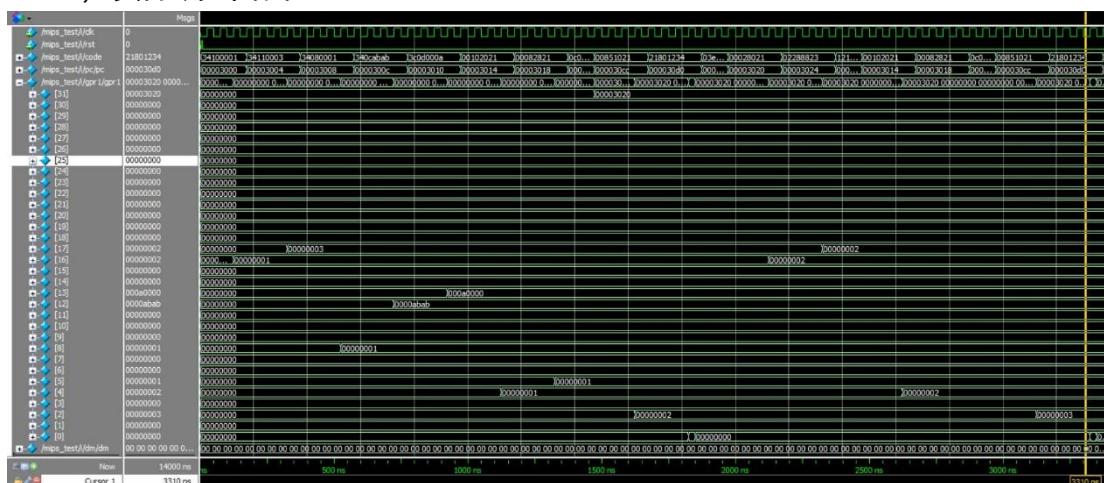
## 2, 预测测试结果

### (1) 寄存器值

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0xababcd
\$v0	2	0xababcd3
\$v1	3	0xababcd
\$a0	4	0x2babcd1
\$a1	5	0x80000002
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x2babcd1
\$t1	9	0x2babcd1
\$t2	10	0x00690000
\$t3	11	0x00000000
\$t4	12	0x0000abab
\$t5	13	0x000a0000
\$t6	14	0x0000007f
\$t7	15	0xab000000
\$s0	16	0x00000003
\$s1	17	0x00000001
\$s2	18	0xfffffff80
\$s3	19	0xefef0000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0xababcd2
\$t8	24	0x7fffffff
\$t9	25	0x00000001
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x00001800
\$sp	29	0x00002ffc
\$fp	30	0x00000000
\$ra	31	0x000030b4
pc		0x000030dc
hi		0x00000000
lo		0x00000000

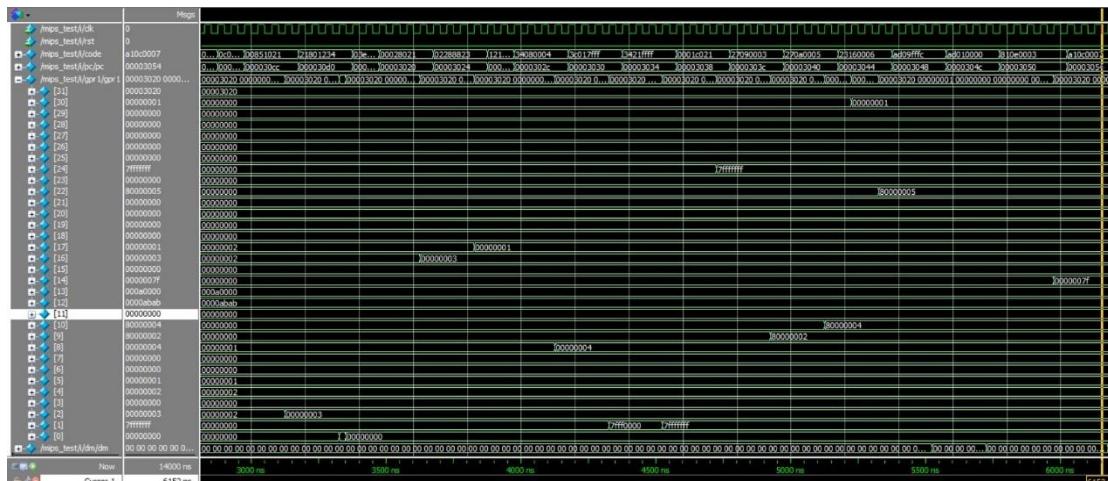
## (2) 数据储存器值

### 3. 实际测试结果



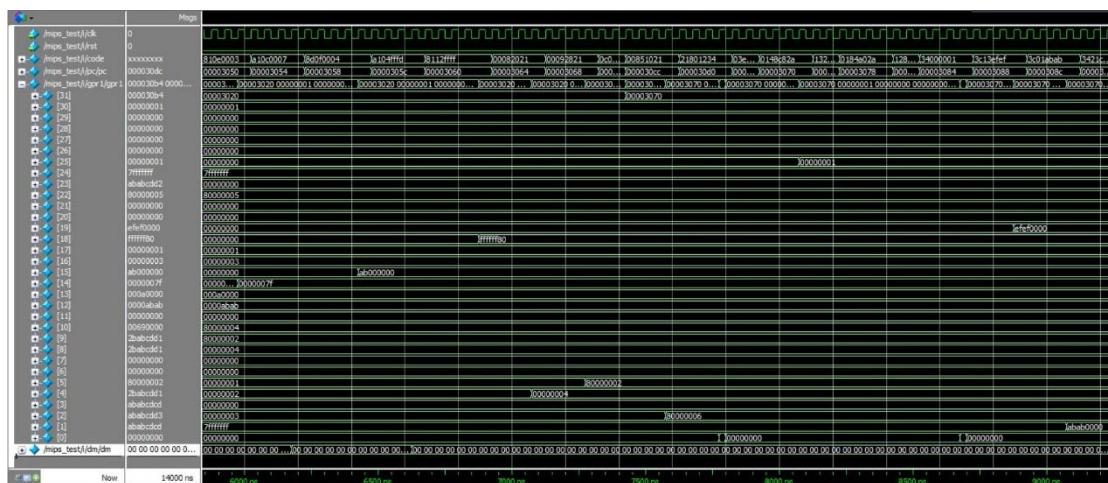
寄存器值 (0-300ns)

说明：前四条指令分别给 16、17、8、12 寄存器赋值，说明 ori 指令正确。Lui \$13,10. 之后 13 号寄存器中值变为 000a0000，说明正确。4、5 号寄存器中的值分别变为 1,1 说明 addu 指令正常。jal 指令执行时，31 号寄存器中存 00003020（为下一个指令地址）。到 jr 指令执行时，pc 从 000030d0 变为 00003020，说明 jr 正确，可正常完成跳转。到 beq 指令时，16、17 号寄存器分别存 2,2，跳到 start 标签处（即 pc=00003014）则 beq 指令正确。



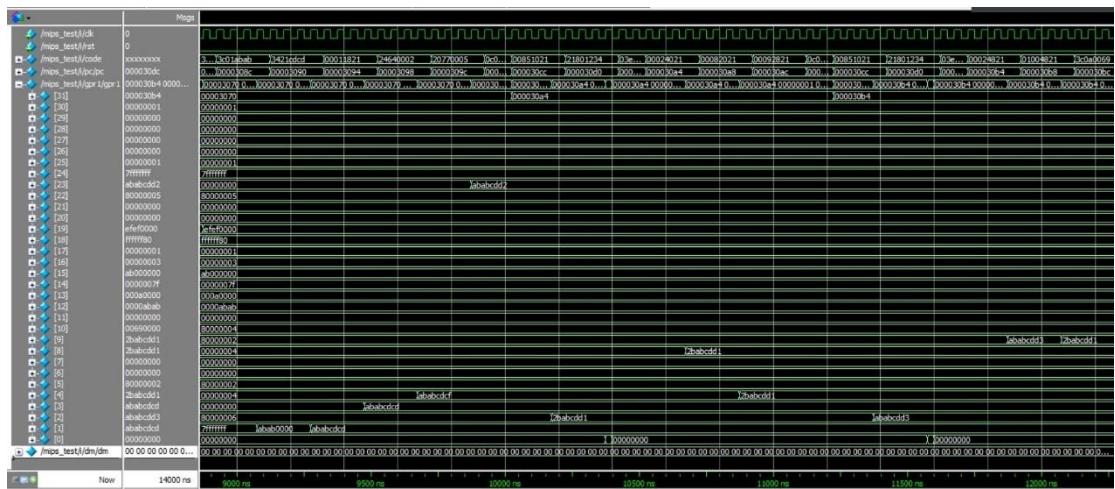
寄存器值 (3000-6000ns)

说明：等到第二次 beq 指令时，此时 16、17 号寄存器中的值为 3、1，则不发生跳转。之后的 addiu 指令与 addu 同理，证明其正确。Addi 产生溢出信号存在 30 号寄存器中。Sw 指令将 9 号寄存器中值存到 0 (4-4) 地址中，见下图 1550ns 处，从地址 0-3 处存 80000002 (小端序)。之后 lb 指令将 7 (3+4) 地址中一个字节的内容取出存在 14 号寄存器中为 0000007f，正确。Sb 指令将 12 号寄存器中内容 0000abab (取最低位的一个字节) 存到 13 (7+4) 地址处，同样也验证正确。在 8075ns 处，10 与 8 号寄存器中内容相比，10 号内存的数小，25 号寄存器置 1，即 slt 指令正确。



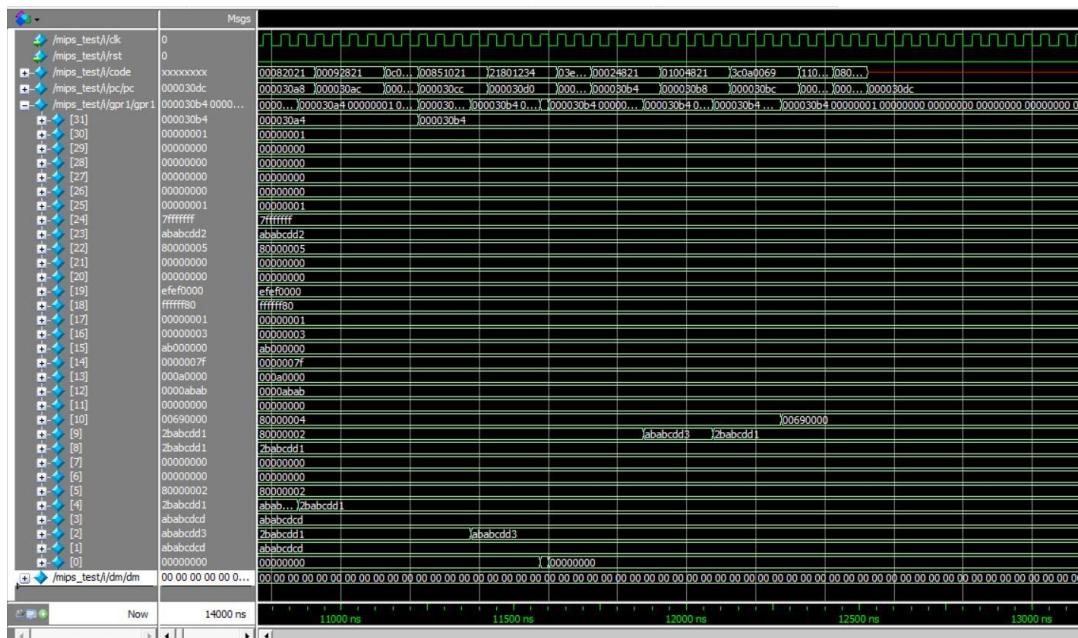
寄存器值 (6000-9000ns)

说明：基本上述所有已经证明正确过



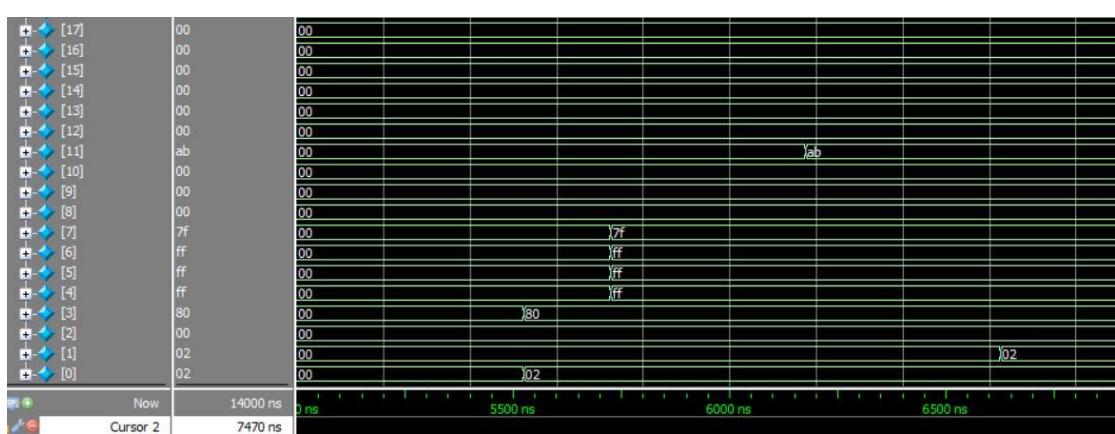
寄存器值 (9000-12000ns)

说明：基本上述所有已经证明正确过



寄存器值 (11000-13000ns)

说明：需要证明其正确的还有 3400ns 处的 j 指令，地址从 000030c8 跳转到 000030dc，其正确。



### 储存器值（只有 5500-6700ns 的时候发生变化）

[15]	00	00
[14]	00	00
[13]	00	00
[12]	00	00
[11]	ab	ab
[10]	00	00
[9]	00	00
[8]	00	00
[7]	7f	7f
[6]	ff	ff
[5]	ff	ff
[4]	ff	ff
[3]	80	80
[2]	00	00
[1]	02	02
[0]	02	02

### 数据储存器组的最终数值

说明：结果与 MARs 环境验证完全相同，证明其正确

[31]	000030b4
[30]	00000001
[29]	00000000
[28]	00000000
[27]	00000000
[26]	00000000
[25]	00000001
[24]	7fffffff
[23]	ababcd2
[22]	80000005
[21]	00000000
[20]	00000000
[19]	efef0000
[18]	fffffff80
[17]	00000001
[16]	00000003
[15]	ab000000
[14]	0000007f
[13]	000a0000
[12]	0000abab
[11]	00000000
[10]	00690000
[9]	2babcd1
[8]	2babcd1
[7]	00000000
[6]	00000000
[5]	80000002
[4]	2babcd1
[3]	ababcdcd
[2]	ababcd3
[1]	ababcdcd
[0]	00000000

### 寄存器组的最终数值

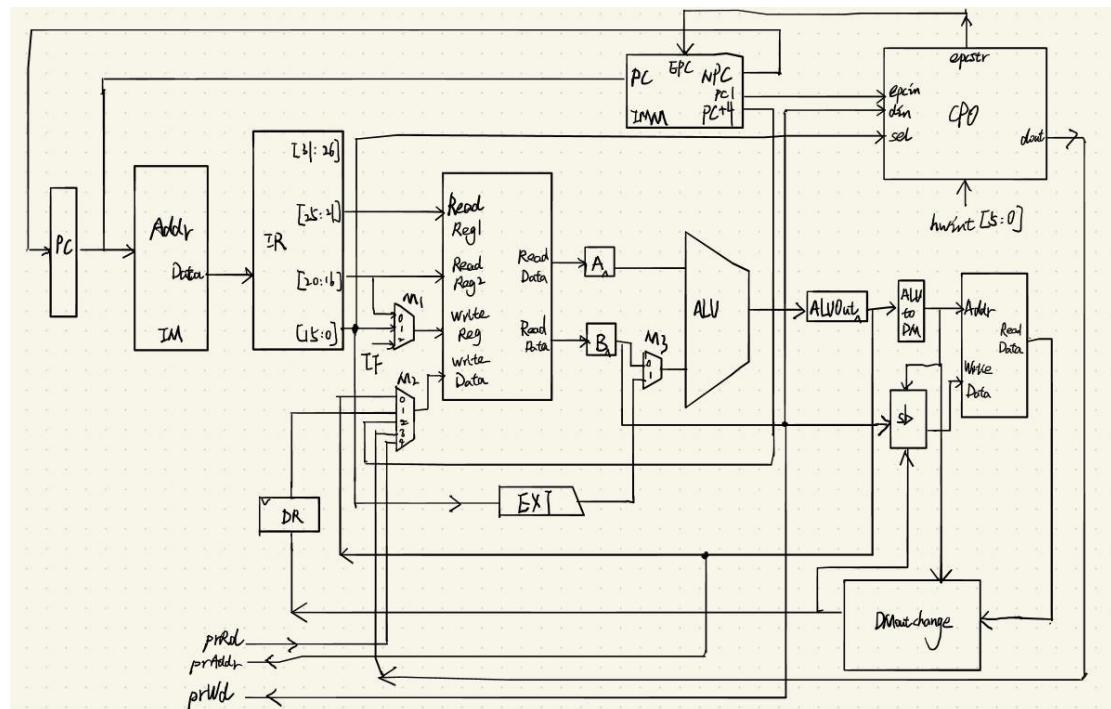
说明：结果与 Mars 环境验证完全相同，证明其正确

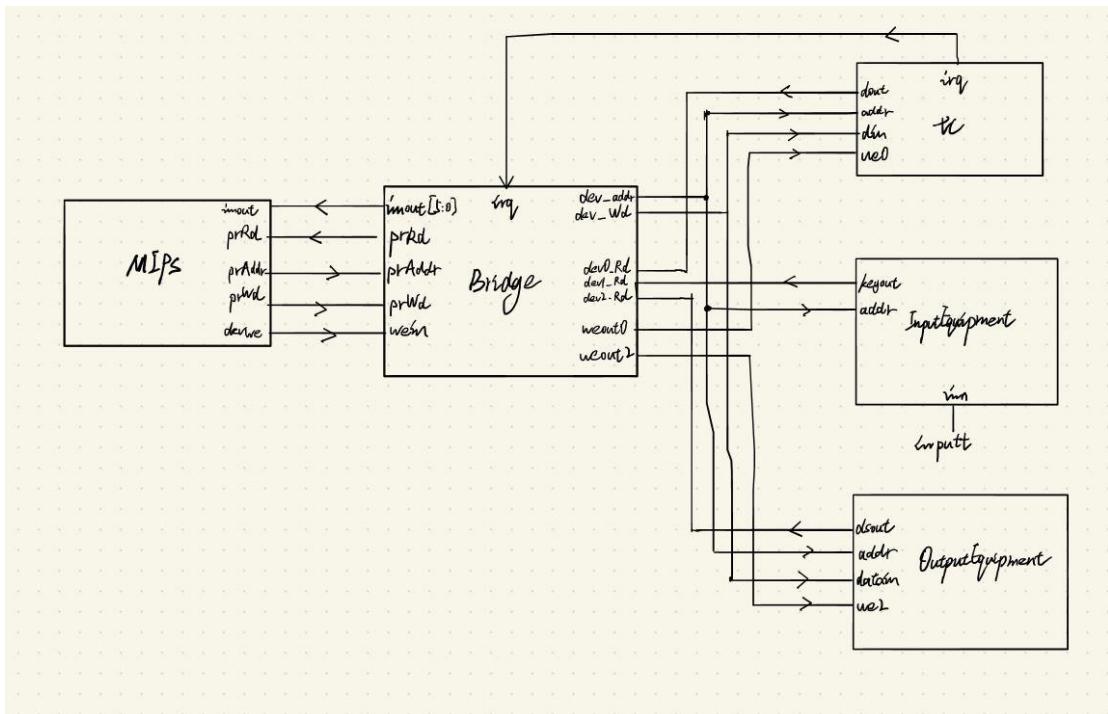
# Project3 VerilogHDL 完成 MIPS 微系统开发(支持设备与中断)

## 一、设计说明

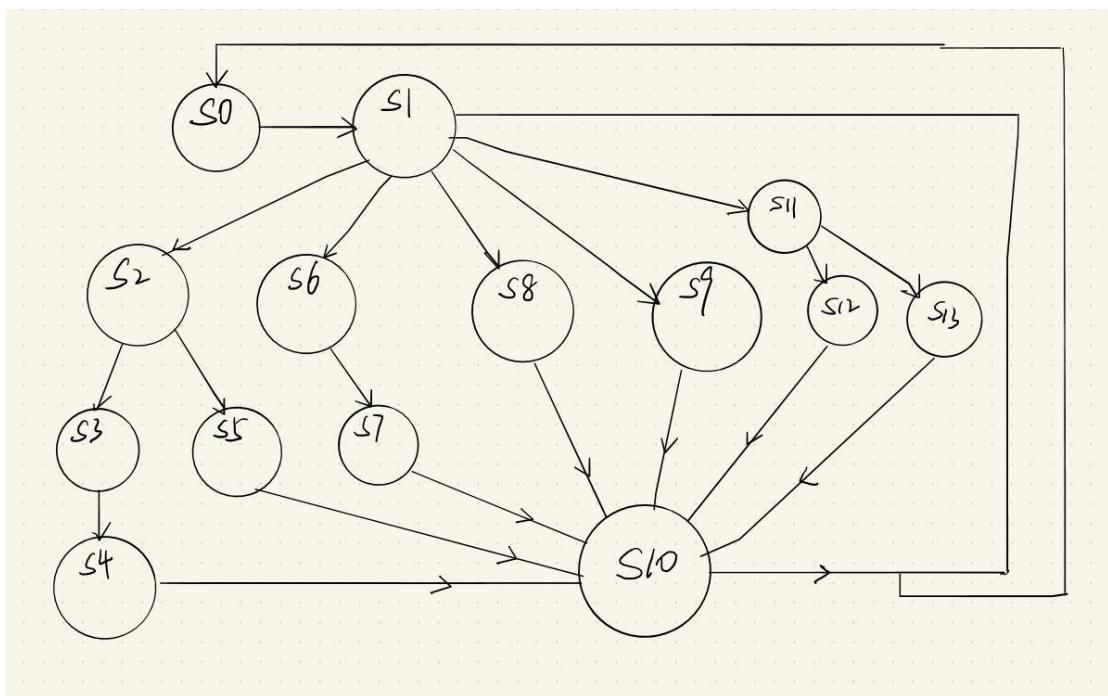
- 1, MIPS 微系统应包括: MIPS 处理器、系统桥和 1 个定时器, 32 位输入设备、32 位输出设备。
  - 2, MIPS 处理器应实现 MIPS-Lite3 指令集。
    - A) MIPS-Lite3 = {MIPS-Lite2, ERET、MFC0、MTC0 }。
    - B) MIPS-Lite2 = {addu, subu, ori, lw, sw, beq, lui, addi, addiu, slt, j, jal, jr, lb, sb }。
    - C) addi 应支持溢出, 溢出标志写入寄存器\$30 中第 0 位。
  - 3, MIPS 处理器为多周期设计。
  - 4, MIPS 微系统支持定时器硬件中断。

## 二、数据通路





### 三、多周期状态图



### 四、模块定义

注: mips 架构 cpu 中除了 controller, 其他模块基本没有改变, 故不再此一一列出, 如需要, 请见 p20 页。

#### 1.bridge 模块定义

### (1) 基本描述

Bridge 模块的主要功能是用于连接 cpu 和外设之间数据写入与读取数据之间的桥梁, 由 `addr[3:2]` 来确定选中了哪个寄存器, 然后将不同的地址分配给不同的外设。可以通过 `cpu` 读取信号来选择读取具体外设中的某一数据。

### (2) 模块接口

信号名	方向	描述
<code>prAddr</code>	I	从 ALUOut 中来的地址
<code>prWd</code>	I	Cpu 给外设的写数据
<code>dev0_Rd</code>	I	定时器需要回写给 cpu 的数据
<code>dev1_Rd</code>	I	输入设备需要回写给 cpu 的数据
<code>dev2_Rd</code>	I	输出设备需要回写给 cpu 的数据
<code>wein</code>	I	总外设写使能信号
<code>irqin0</code>	I	从定时器来的中断信号 (一位)
<code>prRd</code>	O	外设传送到 cpu 的内容
<code>dev_Wd</code>	O	外设写数据
<code>dev_addr</code>	O	外设对应的地址
<code>imout</code>	O	中断信号 (六位)
<code>weout0</code>	O	计时器的写入使能信号
<code>weout2</code>	O	输出设备的写入使能信号

### (3) 功能定义

序号	功能名称	功能描述
1	分配外设	根据不同的 <code>addr</code> 地址选择三个不同的外设
2	输出外设写使能	当总写使能有效且该外设被选中时, 输出该外设的写使能信号。 $weout0 = HitDev0 \& wein$
3	输出 imout 作为 6 位中断信号输出	为了与 cp0 中 SR 匹配, 将由定时器传来的 <code>irqin0</code> 中断扩展成 6 位的 <code>dmout</code> 输出 $imout = \{5'b00000, irqin0\}$
4	将 <code>prwd</code> 转换为 <code>dev_wd</code>	将 cou 传来的写数据转换成对应外设的写数据 $dev\_Wd = prWd;$
5	选择具体的外设写数据	通过具体选取哪一个外设来读取外设的数据 $prRd = HitDev0 ? dev0_Rd : HitDev1 ? dev1_Rd : dev2_Rd;$ 数据

## 2.cp0 模块定义

### (1) 基本描述

Cp0 中含有 4 个寄存器: SR、Cause、EPC、PRId, SR: 用于对系统进行控制, 指令可

读可写。Cause: 指令读取, 硬件控制写入 IP[7:2]: 对应外部 6 个中断源。EPC: 用于保存异常/中断发生时的 PC, 保存 PC: 硬件控制写入, 指令读取: 中断服务程序。PRId: 处理器 ID 可以用于实现个性的编码, 为实现以下两条指令而服务。

MFC0: 读取 CP0 寄存器至通用寄存器

MTC0: 通用寄存器值写入 CP0 寄存器

## (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	清零信号
epcin	I	从 ifu 来, 存储中断此时的下一条指令
din	I	输入, 给寄存器赋值
hwint	I	从 bridge 传来的 6 位中断信号 000001
sel	I	寄存器选择信号
wen	I	写使能
exlset	I	从 contro 来的, 代表中断状态
exlclr	I	代表当前中断已经返回
intreq	O	中断产生信号 (传向 contro)
epcstr	O	Epc 存的下一条地址值
dout	O	将 cp0 中的寄存器的值传回到通用寄存器

## (3) 功能定义

序号	功能名称	功能描述
1	输出总中断信号	$\text{intreq} = (\text{hwint}[5:0] \& \text{sr}[15:10]) \& \text{sr}[0] \& \sim\text{sr}[1];$
2	Mfc0	$\text{dout} = \text{data}[\text{sel}]$ ; cp0 到通用寄存器
3	Mtc0	$\text{data}[\text{sel}] \leq \text{din}$ 通用寄存器到 cp0
	中断	产生中断信号 接收允许中断信号
4	Exlset 有效时, 将 12 号寄存器第一位置一	当中断有效时, 12 号寄存器的第一位保护位置一, 防止其他中断乱入
5	Exlclr, 将 12 号寄存器地位清零	Eret 代表当前有效中断已经返回, 其他中断又可以进入
6	锁存数据	主要需要锁存的是 epc 内存的值

## 3.tc 模块定义

### (1) 基本描述

主要用于倒计数, 并且为整个程序提供中断信号。(最源头的信号)

## (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	清零信号
Addr[3:2]	I	从 aluout 经过 bridge 来, 提供要写入外设的地址
Din[31:0]	I	输入来自 bridge, 要写入的值
We	I	来自 bridge 允许写入信号 1: 允许向地址写 2: 不允许向地址写
Irq	O	中断信号
dout	O	模块输出 (到 bridge)

## (3) 功能定义

序号	功能名称	功能描述
1	输出倒计数	输出倒计数
2	允许倒计数	由 ctrl[0]控制 1: 开始倒计数 2: 停止倒计数
3	倒计数方式	由 ctrl[1]控制 0: 方式 0 1: 方式 1
4	中断信号发出	由 ctrl[3]控制 0: 不允许中断 1: 允许中断
5	写入 control 和 preset 的值	当写信号有效时, 允许将得到的值写到对应地址里。 提供启动控制信号及其初始倒计时值。

## 3.inpuquipment 模块定义

### (1) 基本描述

输入显示。

### (2) 模块接口

信号名	方向	描述
Inn	I	Text 程序输入
Addr[3:2]	I	地址
keyout	O	模块输出

### (3) 功能定义

序号	功能名称	功能描述
1	输出输入显示	输出输入显示

## 4. outputequipment 模块定义

### (1) 基本描述

输出显示。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
we	I	写使能信号
Addr	I	要写入的地址
datain	I	要写入的数据
dsout	O	模块输出

### (3) 功能定义

序号	功能名称	功能描述
1	输出显示	输出写入地址的值 并显示

## 5. contro 模块定义

### (1) 基本描述

根据 32 位指令分析出的 opcode 和 funct 对应于相应的指令，分析出每个指令执行的数据通路的相应写使能信号和选择信号。同时作为多周期，会根据不同指令的执行情况经过的过程分为不同的状态以下为状态描述：

S0	取址、译码
S1	机器码缓存
S2	sw sb lw lb 分支
S3	访问存储器
S4	写回 gpr
S5	写 DM
S6	执行 (ALU 计算等)
S7	ALU 写回 gpr
S8	Beq 跳转指令
S9	J jal jr eret 指令
S10	中断状态
S11	mfc0 mtc0
S12	Mfc0 执行
S13	Mtc0 执行

## (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
code	I	32 位 MIPS 指令
Intreq	I	从 cp0 传来的中断信号
doutreg	I	ALUOut 算出的地址
ALU_OP[2:0]	O	ALU 选择信号 000: 加运算 001: 减运算 010: 与运算 011: 或运算 100: 异或运算 101: 判断两数大小 相等: zero 为 1 不相等: zero 为 0 110: 判断 addi 是否溢出
WDSel	O	寄存器写入选择控制信号 00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里对应 jal 指令
GPRSel	O	寄存器回写选择控制信号 00: 写到 rt 对应的寄存器里 01: 写到 rd 对应的寄存器里 10: 写到 31 号寄存器里对应 jal 指令
ExtOp	O	扩展信号 00: 0 扩展 01: 符号扩展 10: lui 变化
GPRWr	O	寄存器写使能
BSel	O	ALU 第二位操作数选择信号
DMWr	O	DM 写使能
jsome	O	J 指令, jal 指令控制信号
npc_sel	O	Beq 指令控制信号
jr	O	Jr 指令控制信号
jal	O	Jal 控制信号, 用于选择回写 31 号寄存器
lb	O	Lb 控制信号
Sb	O	Sb 控制信号
Eret	O	从中断跳回信号
Mfc0	O	指令 mfc0 控制信号
Mtc0	O	指令 mtc0 控制信号
Cp0en	O	是 cp0 写使能信号
Exlset	O	Cp0 中 sr 的第一位置 1
Exlclr	O	Cp0 中 sr 的第一位置 0

Intctr	O	Intreg 在 s10 状态有效置 1，传给 ifu 实现中断跳转
Usedev	O	判断出来是外设的地址
We	O	外设的总写使能有效

## 6. mips 模块定义

### (1) 基本描述

起各模块的连接作用。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	置位信号
Imout	I	中断信号
prRd	I	写回寄存器选择信号
prWd	O	要写入外设的内容
prAddr	O	写入的外设地址
devwe	O	写外设的允许信号

### (3) 模块功能

序号	功能名称	功能描述
1	cpu 顶层文件	完成各个模块间的相互调用。
2	为外设提供服务	提供写入的地址、内以及写信号。
3	中断信号	为 CP0 提供来自外设的中断信号

## 7.mach 模块定义

### (1) 基本描述

起 cpu 与 bridge 与外设的连接作用。

### (2) 模块接口

信号名	方向	描述
clk	I	时钟信号
rst	I	置位信号
Inputt	I	给输入设备的值

### (3) 模块功能

序号	功能名称	功能描述
1	顶层文件	完成各个模块间的相互调用。
2	更新值	提供要更新的值。

## 五、机器指令描述

指令名称	操作码 opcode	功能码 funct	功能	指令功能描述
addu	000000	100001	无符号数加	$R[rd] \leftarrow R[rs] + R[rt]$
subu	000000	100011	无符号减	$R[rd] \leftarrow R[rs] - R[rt]$
slt	000000	101010	小于时置位	$R[rd] \leftarrow (GPR[rs] < GPR[rt])$
addi	001000		立即数加法 (支持溢出)	未溢出时: $R[rt] \leftarrow R[rs] + immediate$ 溢出时: 不改变寄存器内的值
addiu	001001		立即数加法 (不支持溢出)	$R[rt] \leftarrow R[rs] + immediate$
ori	001101		立即数或	$R[rt] \leftarrow R[rs]   ZeroExt(imm16)$
lw	100011		取字	$R[rt] \leftarrow MEM\{R[rs] + SignExt[imm16]\}$
sw	101011		存字	$MEM\{R[rs] + SignExt[imm16]\} \leftarrow R[rt]$
beq	000100		相等时跳转	$if(R[rs] == R[rt]) then$ $PC \leftarrow PC + 4 + (signExt(imm16)    00)$
lui	001111		取立即数的高位	$R[rt] \leftarrow imm16    0^{16}$
j	000010		无条件跳转	$PC \leftarrow PC + 4[31..28]    instr    0^2$
jal	100000		跳转并链接	$\$31 = pc + 4$
jr	000000	0010000	跳转至寄存器所指位置	$PC \leftarrow R[rs]$
sb	101000		存字节	$MEM\{R[rs] + SignExt[imm16]\} \leftarrow R[rt][7:0]$
lb	100000		取字节	$GPR[rt] \leftarrow memory[GPR[base] + offset]$
Mtc0	010000	00100	写 cp0	$CPR[0, rd, sel] \leftarrow GPR[rt]$
Mfc0	010000	00000	读 cp0	$: GPR[rt] \leftarrow CPR[0, rd, sel]$
Eret	010000	011000	中断程序返回	To return from interrupt, exception, or error trap.

## 六、测设要求

### 1，测试程序

Main 代码

机器码	指令	注释
34017f00	ori \$1,\$0,0x7f00	给\$1 赋值 0x000007f00 (计时器)
34027f10	ori \$2,\$0,0x7f10	给\$2 赋值 0x000007f10(输入设备)
34037f20	ori \$3,\$0,0x7f20	给\$3 赋值 0x000007f20(输出设备)
8c440000	lw \$4,(\$2)	将\$2 偏移 0 的地址的内容放到\$4 寄存器里 (得到输入的值)
ac640000	sw \$4,(\$3)	将\$4 寄存器的内容放到\$3 偏移 0 的地址里 (值送到输出)
ac640004	sw \$4,4(\$3)	将\$4 寄存器的内容放到\$3 偏移 4 的地址里 (值也送到输出显示)
34050401	ori \$5,\$0,0x0401	给\$5 赋值 0x00000401
40856000	mtc0 \$5,\$12	将\$5 寄存器的内容放到\$12 里 (把 00000401 作为 sr 的控制信号)
401d7800	mfc0 \$29,\$15	将\$15 寄存器的内容放到\$29 里 (将本人的学号取出到 15 号寄存器中)
3406000a	ori \$6,\$0,10	给\$6 赋值 10
ac260004	sw \$6,4(\$1)	将\$6 寄存器的内容放到\$1 偏移 4 的地址里 (preset 赋值)
34070009	ori \$7,\$0,9	给\$7 赋值 9(0x1001)
ac270000	sw \$7,(\$1)	将\$7 寄存器的内容放到\$1 偏移 0 的地址里 (1001 作为计时器的控制信号)
08000c0d	lop: j lop	跳转到 lop

Sub 代码

机器码	指令	注释
8c480000	lw \$8,(\$2)	将\$2 偏移 0 的地址的内容放到\$8 寄存器里 (取出输入设备的值)
8c690000	lw \$9,(\$3)	将\$3 偏移 0 的地址的内容放到\$9 寄存器里 (取出输出设备的值)

11090003	beq \$8,\$9,equ	\$8 和\$9 相等则跳转到 equ
ac680000	sw \$8,(\$3)	将\$8 寄存器的内容放到\$3 偏移 0 的地址里(若不等, 将输入的值放入输出中)
ac680004	sw \$8,4(\$3)	将\$8 寄存器的内容放到\$3 偏移 4 的地址里
10000003	beq \$0,\$0,exit	直接跳转到 exit
8c6a0004	equ: lw \$10,4(\$3)	将\$3 偏移 4 的地址的内容放到 \$10 寄存器里
254a0001	addiu \$10,\$10,1	\$10 自加 1(输出显示的值加 1)
ac6a0004	sw \$10,4(\$3)	将\$10 寄存器的内容放到\$3 偏移 4 的地址里
340b000a	exit: ori \$11,\$0,10	给\$11 赋值 10
ac2b0004	sw \$11,4(\$1)	将\$11 寄存器的内容放到\$1 偏移 4 的地址里(preset 重新赋值)
340c0009	ori \$12,\$0,9	给\$12 赋值 9(0x1001)
ac2c0000	sw \$12,(\$1)	将\$12 寄存器的内容放到\$1 偏移 0 的地址里 (1001 作为控制信号给计时器控制器)
42000018	eret	返回

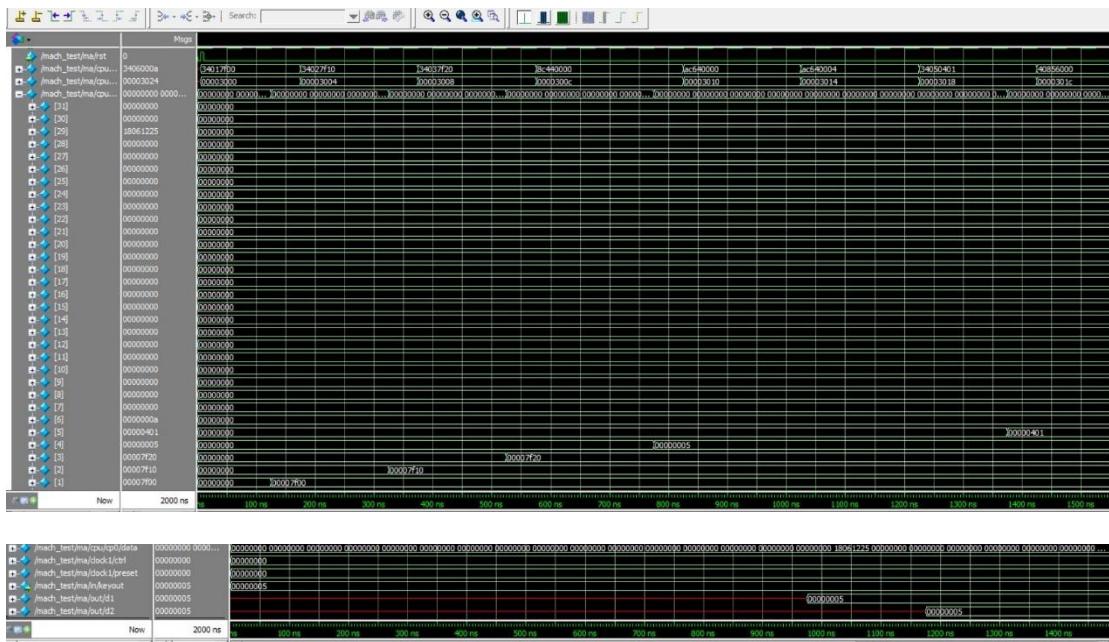
## 2, 预测测试结果

因为 Mars 无法完全模拟输入以及定时器中断, 预期测试结果便用口头描述:

开始时, 将定时器、输入设备、输出设备的地址分别赋给 1,2,3 号寄存器, 将输入的值赋到输出设备且显示上, 后将 0x0401 传给 sr 作为中断控制信号, 从 prid 中取出学号放到 29 号寄存器中, 之后对 tc 中 preset 和 contro 赋值, 此时倒计时开始, 并且主程序进入死循环。

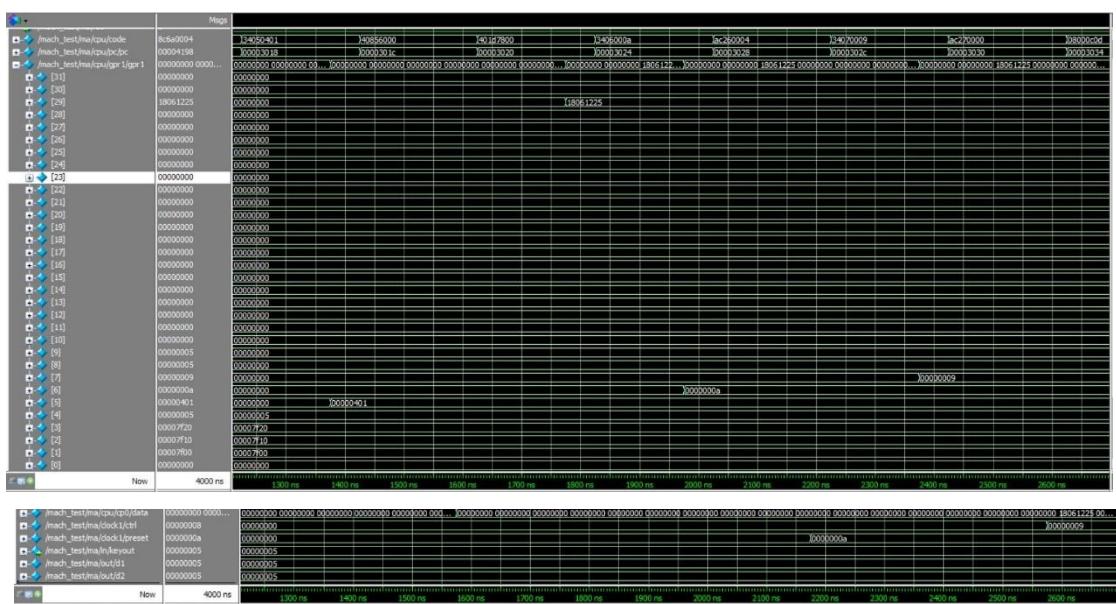
到中断程序时, 先把输入和输出设备的值分别取出进行比较, 若相等将 d2 的值加一, 在重新给 preset 和 contro 赋值再次进入倒计时; 若不相等, 将输入的值直接赋值给输出, 再对 preset 和 contro 赋值进入倒计时, 计时到 0 时, 产生中断信号。

## 3, 实际测试结果



(0-1500ns)

说明： 前三条指令对 1,2,3 号寄存器进行赋值（外设地址）。800ns 时，lw 从输入设备取到值 5，1000ns 是 sw 指令将 5 置入 d1 中，之后同理置入 d2 中。



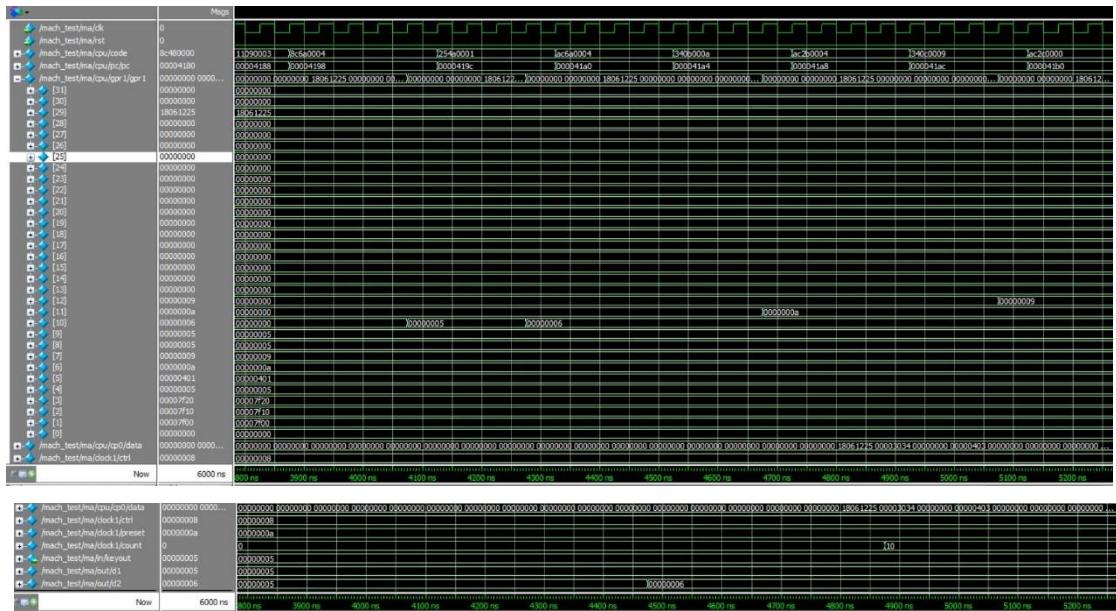
(1300-2600ns)

说明： 1800ns 时，29 号寄存器内容变为本人学号，证明 mfc0 正确。之后在 2200ns 和 2600ns，preset 和 tccontrol 的值分别变为 10 和 9，再次 sw 指令正确。此时开始倒计时，同时主程序进入死循环。



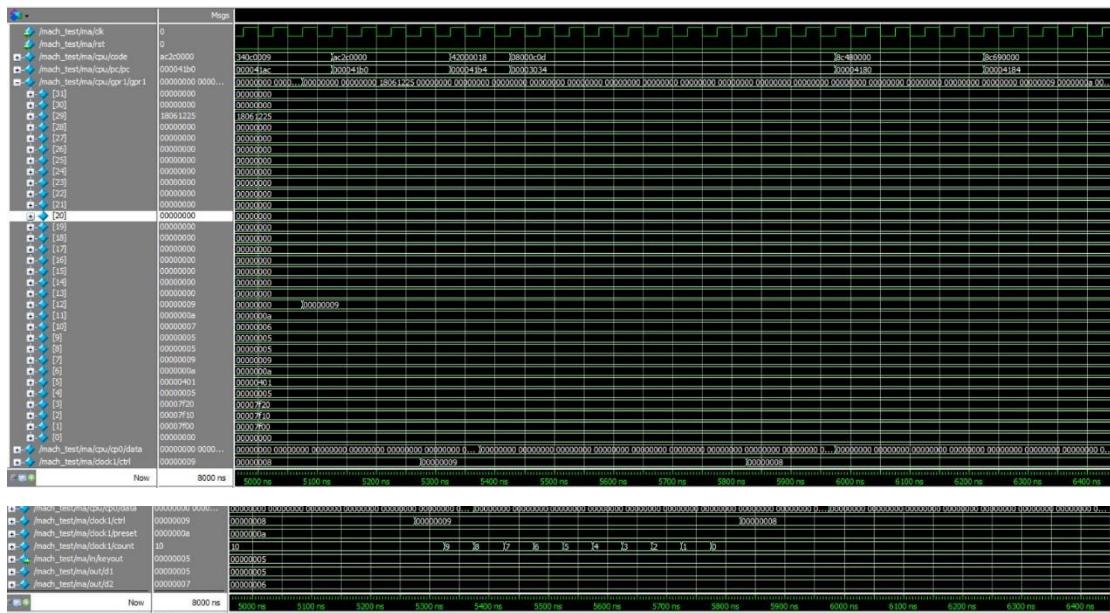
(2500-3900ns)

说明：可以看到在 2600ns 时 count 开始倒计时，此时主程序一直在执行 j lop 指令，倒计时到 0 时，产生中断信号，地址跳转到 0x00004180 处，开始实行中断子程序。在 3550 和 3750ns 时将输入和输出设备的值拿出做比较，相同则输出设备显示的值加 1，之后计时器再次启动。



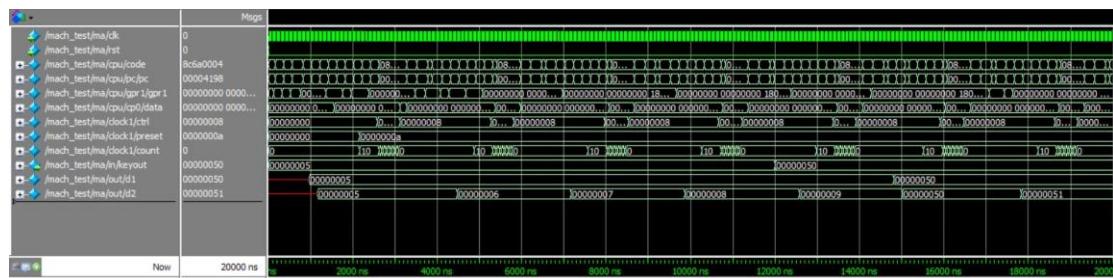
(3800-5200ns)

说明：计时器再次启动



(5000-6400ns)

说明： eret 指令使返回主程序继续执行 j lop，证明其正确，此时倒计时倒数至 0 时，再次产生中断，此后不断循环。



(0-20000ns)

说明： 这是一个不断跳转的过程，d2 的值不断变化，到 15000ns 处，输入的值改变，使 d1, d2 也改变，此后因输入输出值又相同则 d2 不断加 1。满足预测测试结果，程序正确。

## 收获体会

这两周过得还是相当充实的，最大的一个感悟还是理论和真正的实践还是有着很大的区别，正常上课的理论学习只用关心这一个点的理解掌握就 ok 了，但实践操作中这是远远不够的，你需要掌握每一个需要的知识点，以及需要更为严谨的逻辑，才能一个模块一个模块的实现到最后一个系统的整体穿合起来。

记得刚开始时做 Project 1 的时候，不怕大家笑话，那时的我甚至连字节和字的关系都不是理解的很清楚，就那种水平的我，先是把整个第四、五章从头到尾的先看了一遍，然后跟着老师给的 Modelsim 的演示视频一点点把 ifu 搭了出来，那时虽然还是有很多的不明白的地方，但还是硬着头皮从 gpr、alu、dm 模块开始一点一点的做了下去。第一次重大的危机发生在 project 1 整个测试时，将指令文件加载后硬是不出波形，也问了很多的人相关的问题，问题依然还是没有解决，但发现了很多写法上的问题：比如刚开始写的时候，我还是怀着一种 c 语言等高级语言的写作逻辑在写代码，总喜欢用一些 if else 等语句，虽然明面上看着没什么问题，但我忘了最重要的一点：verilog 是硬件描述语言，我应该像用 logelsim 开发单周期控制器一样，具体到每一个寄存器或是每一条线路上去。这时我也刚刚明白 reg 和 wire 变量的真正含义：reg 代表寄存器、wire 代码连线等等，知道了这些，我的思路一下就被打开了，能用数据流描述的地方，绝对不用 always 描述。紧接着我以为波形不出现是中间模块出的问题，就给每一个模块写了一个测试文件，都确保无误的时候，工作进度一下被中断了（甚至以为是软件本身的问题），鼠标又点到了不知道点过多少遍的 text2.v（这是我给 mips 写的测试文件），发现清零信号我在输入端和应用的写法并不一致（rst 和 reset），一下难受坏了，就这一个小问题就困扰了我一两天的时间，之后在 simulate 波形就正常的出现了。这也让我在后面进行 project2 project3 的工作时，变得更为认真仔细，绝不让这些笔误阻碍正确结果的产生。

在实现 project1 之后，project2 就显得简单了很多，状态机完全在控制器中实现，我最初是按课件 5-4 进行设计的，但后面这测试时发现 beq、j、jal 等跳转指令回到状态 0 时 pc 的值会再次加 4，相当于一条完整的指令没有运行（我设计的跳转指令会在 s8, s9 状态时便进行 pc 的更换，所以会出现 pc 连续变化

两次的问题)，因此我将全部跳转指令完成跳转后的下一个转态改为 s1 遍完美得解决了这个问题。相较于多周期的实现，sb、lb 指令的实现就难了很多，因为不让在 dm 模块中增添多的引脚，又使得任务量增加了两三倍，后面想到了整字替换的办法，就是讲现在地址所对应的整个字拿出，若为 sb 指令就再根据地址替换其中的一个字节再放入 dm 中，lb 指令的实现也是相同的道理，也把整个字拿出，根据地址取出相应的字节进行符号扩展再传给 gpr 就 ok，dm 中地址的规格化也很重要，详情请见 p24。

Project3 确实是这整个课设中最难的一个，思路和逻辑要非常严谨才能实现以及进行整体联调。对我来说一个大的收获便是理解了 mips 架构的微系统中，不同地址段存放的内容是不同的（虽然说我以前也知道知道这个，但当自己真正实现了那理解就是深了不少）。还有我们所模拟的硬件中断发生，中断信号的不断传递也是一个很有趣的过程（起初中断请求喜好从 tc 中产生，传到 bridge 加工成 6 位，再传到 cp0 中判断是否相应中断，再传到 contro 中，与 s10 状态与形成最终的中断信号，最后传入 ifu 中实现中断时的地址跳转）。我还想说的是在 cp0 总的 epc 寄存器，我设计的锁存器自我感觉也是非常巧妙的，把写入的条件严严限制死，就不会有地址乱入的问题发生。

说实话，我很享受写代码的感觉，在整个过程中，我能一点一点地完成一个又一个小问题，到最后自己的结果与老师所给的一模一样时，那种成就感是无法比拟的，这也就是我为什么当初转专业的原因。因为喜欢这个，就来到了这里，就是一个如此简单的理由，儿时最初的兴趣最终还是决定了我的职业轨迹，但做自己喜欢的事又是多么幸福的一件事呢。最后谢谢老师这几日的付出，检查和批阅这么多人的代码真的很辛苦，老师辛苦了！