

# Verilog 搭建流水线 CPU 设计报告

## 一、 数据通路设计

### 1.PC

#### (1)接口定义

信号名称	方向	描述
reset	I	PC 复位至 0x00003000
NPC[31:0]	I	下一个 PC 值（下一个时钟上升沿的时候要写入 PC 寄存器的值）
PC[31:0]	O	PC 输出

#### (2) PC 功能定义

序号	功能名称	描述
1	复位	当 reset 有效时，PC 寄存器被赋值为 0x00003000
2	输出指令地址	在时钟上升沿的时候更新 PC

### 2.ADD4 模块

#### (1)接口定义

信号名	方向	描述
PC[31: 0]	I	当前 PC 值
PC4[31: 0]	O	=PC+4

#### (2) 功能定义

序号	功能名称	描述
1	输出 PC+4	纯组合逻辑，永远输出 PC4=PC+4

### 3. IM 模块

#### (1) 接口定义

信号名	方向	描述
clk,reset	I	时钟，复位信号

PC[31:0]	I	要读取的指令的地址
instr[31:0]	O	从 IM 中取出来的 32 位指令

（2）功能定义

序号	功能名称	描述
1	取指令	根据输入的 PC 值输出相应位置的指令

4、RF

GRF 由 32 个寄存器构成（其中 0 号寄存器恒为 0）。首先计算 RegA、RegB、RegD 信号的值，再进行读或写操作：读操作时，将编号为 RegA、RegB 的两个寄存器的值读出到 busA、busB。写操作时，在 RegWr 信号为 1 时，将 dataWr 写入 RegD 编号对应的寄存器中。

表格 1 GRF 端口说明

端口名	方向	说明
IR_D[31:0]	I	D 级指令，用来提取 rs,rt
R3[4:0]	I	待写入寄存器的编号
RFin[31:0]	I	待写入寄存器的数据
RegWr	I	寄存器写使能信号 0：不写入      1：写入
clk	I	时钟信号
reset	I	复位信号
PC4[31:0]	I	PC+4 值，\$display 时-4 后使用
RFOut1[31:0]	O	读出数据 1(\$rs 的值)
RFOut2[31:0]	O	读出数据 2(\$rt 的值)

表格 2 GRF 功能定义

序号	功能名称	功能描述
1	读寄存器	将编号为 R1 的寄存器中的数据输出到 RFOUt1 端口； 将编号为 R2 的寄存器中的数据输出到 RFOut2 端口

2            写寄存器        RegWr=1 时，在时钟上升沿将 RFin 写入到编号为 R3 的寄存器中；写入寄存器时进行输出操作

5、EXT（扩展器）

EXT 根据 ExtOp 对 16 位立即数 imm16 进行各类扩展。

表格 3    EXT 端口说明

端口名	方向	说明
IR_D[31:0]	I	D 级指令，用来提取 16 位立即数 imm16
ExtOp[1:0]	I	进行何种扩展的选择信号。 00：无符号扩展 01：有符号扩展 10：加载到高 16 位，低 16 位补 0 11：（未定义）
ExtOut[31:0]	O	扩展后的数据。

表格 4    EXT 功能定义

序号	功能名称	功能描述
1	无符号扩展	ExtOp=00 时，对 imm16 进行无符号扩展并输出到 ExtOut。
2	有符号扩展	ExtOp=01 时，对 imm16 进行有符号扩展并输出到 ExtOut。
3	后补 16 位 0	ExtOp=10 时，将 imm16 加载到高 16 位，在低 16 位补 0，并输出到 ExtOut。

6.CMP 模块

（1）模块接口

信号名称	方向	描述
CMPIIn1[31: 0]	I	第一个操作数

CMPIn2[31: 0]	I	第二个操作数
BrType[2: 0]	I	Br 指令类型，比较方式选择
BrTrue	O	比较结果，用于是否做 br 跳转的判断

## (2) 功能定义

序号	功能名称	描述(C 为 BrType;A,B 为 CMPIn1,CMPIn2)
1	==(BrType=000)	C=(A==B)
2	!=(BrType=001)	C=(A!=B)
3	A>B(BrType=010)	C=(A>B)
4	A<=B(BrType=011)	C=(A<=B)
5	A<B(BrType=100)	C=(A<B)
6	A>=0(BrType=101)	C=(A>=0)

## 7.NPC 模块

### (1)接口定义(计算下一个 PC 的值，纯组合逻辑)

信号名	方向	描述
PC4[31: 0]	I	当前 PC 值
IR_D[31: 0]	I	D 级指令，用于提取 imm16 和 imm26
BrTrue	I	从 CMP 块得到的是否满足 br 指令跳转条件的信号
jPC[31:0]	O	j 指令的下一个 PC
brPC[31:0]	O	br 指令的下一个 PC

### (2) NPC 功能定义

序号	功能名称	描述
1	计算 brPC	BrTrue=0 时，brPC=PC4+4 BrTrue=1 时，brPC=PC4+sign_extend(offset  00)
2	计算 jPC	jPC= PC4[31:28]    imm26    00

## 8、ALU（算术逻辑单元）

ALU 由何种算数逻辑组成。根据 ALUctr 对 ALUin1 和 ALUin2 进行加、减、或、相等比较等操作并输出。

表格 5 ALU 端口说明

端口名	方向	说明
ALUIn1[31:0]	I	第一个待操作数。
ALUIn2[31:0]	I	第二个待操作数。
ALUOp[2:0]	I	进行何种运算的选择信号。  00: ALUIn1+ALUIn2  01: ALUIn1-ALUIn2  10: ALUIn1   ALUIn2  11: ALUIn2 << ALUIn1[4:0]
ALUOut[31:0]	O	运算后的结果。

表格 6 ALU 功能定义

序号	功能名称	功能描述
1	加（无溢出）	ALUOut=ALUin1 + ALUin2
2	减（无溢出）	ALUOut=ALUin1 - ALUin2
3	或	ALUOut=ALUin1   ALUin2

9、DM（数据存储器）

表格 7 DM 端口说明

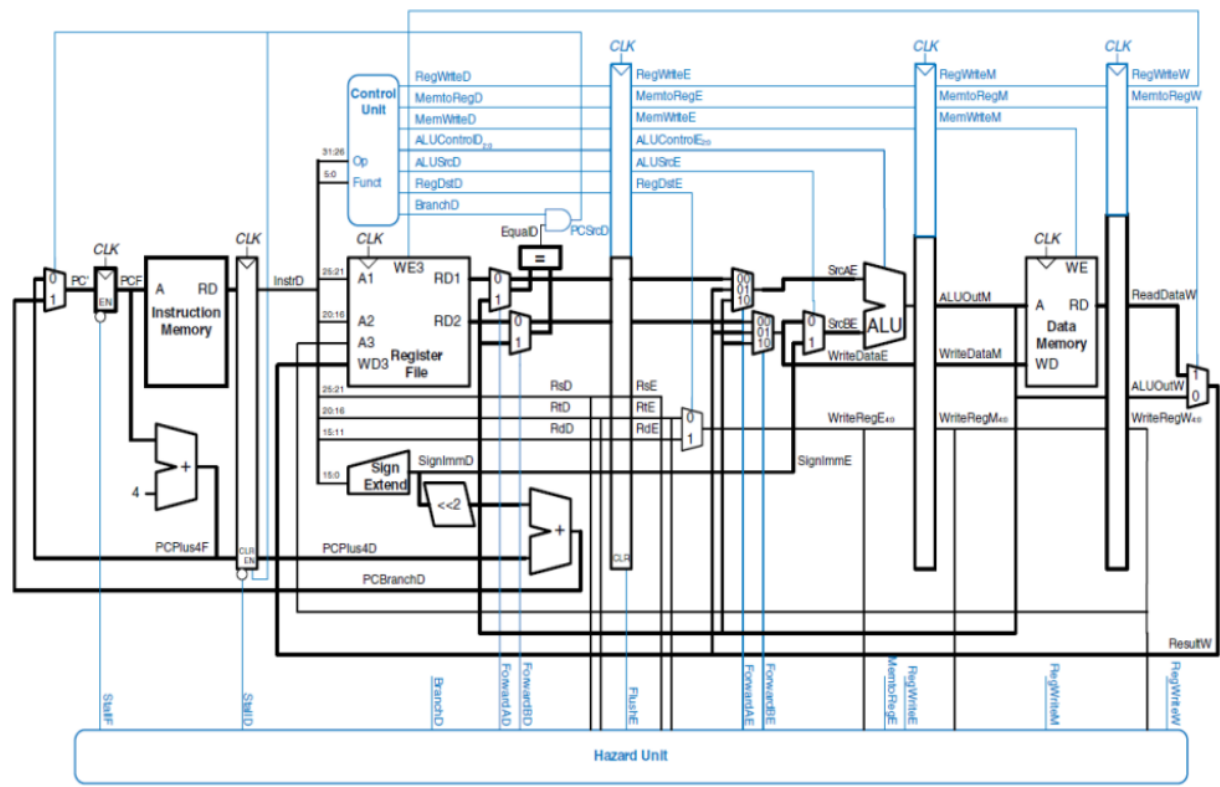
端口名	方向	说明
DMAddr[31:0]	I	DM 中的读出/写入地址，即 ALU 的输出端 ALUOut
DMIn[31:0]	I	待写入 DM 的数据,即 GRF 的输出端 RFOut2
MemWr	I	将 DMdata 写入 DM 的写使能信号。  0: 不写入    1: 写入
clk	I	时钟信号
reset	I	复位信号
PC4[31:0]	I	PC4 值，\$display 时-4 后使用

DMOut[31:0]      O      DM 输出数据

表格 8    DM 功能定义

序号	功能名称	功能描述
1	读出	DMOut=DM 中 DMAddr 地址中的数据。
2	写入	MemWr=1 时，将 DMIn 写入 DM 的 DMAddr 地址中。

10、电路图总览



二、 Controller（控制器）设计

1、基本描述

Controller 根据指令中的 opcode 段和 funct 段，先利用与门确定该指令类型，再利用或门确定各控制信号。

端口名	方向	说明
instr[31:0]	I	指令

ALUSrc1	O	
ALUSrc2	O	
MemWr	O	
RegWr	O	
ExtOp[1:0]	O	控制信号
ALUOp[2:0]	O	说明见 Excel
NPCsel[1:0]	O	
RegDst[1:0]	O	
MemtoReg[1:0]	O	
BrType	O	
calr	O	指令类型
cali	O	说明见 Excel
br	O	
load	O	
store	O	
jal	O	
jr	O	
jalr	O	

序号	功能名称	功能描述
1	译码	将 instr 根据上表所述进行译码

2、控制信号真值表（见 Excel）

3、控制信号含义（见 Excel）

### 三、 中断异常设计模块

1、BRIDGE

端口名	方向	说明
-----	----	----

DMAddr[31:0]	I	DM 中的读出/写入地址，即 ALU 的输出端 ALUOut
DMIn[31:0]	I	待写入 DM 的数据,即 GRF 的输出端 RFOut2
MemWr	I	将 DMdata 写入 DM 的写使能信号。 0: 不写入    1: 写入
clk	I	时钟信号
reset	I	复位信号
PC4[31:0]	I	PC4 值，\$display 时-4 后使用
DMOut[31:0]	O	DM 输出数据

## 2、TimerCounter

端口名	方向	说明
DMAddr[31:0]	I	DM 中的读出/写入地址，即 ALU 的输出端 ALUOut
DMIn[31:0]	I	待写入 DM 的数据,即 GRF 的输出端 RFOut2
MemWr	I	将 DMdata 写入 DM 的写使能信号。 0: 不写入    1: 写入
clk	I	时钟信号
reset	I	复位信号
PC4[31:0]	I	PC4 值，\$display 时-4 后使用
DMOut[31:0]	O	DM 输出数据

## 3、CP0

端口名	方向	说明
A1[4:0]	I	读 CP0 寄存器编号（MFC0 产生）
A2[4:0]	I	写 CP0 寄存器编号（MTC0 产生）
CP0In[31:0]	I	CP0 写入数据（MTC0 产生，来自 GPR）
PC	I	
ExcCode[6:2]	I	中断异常类型
HWInt[5:0]	I	6 个设备中断（外设 TC0/TC1 产生）
CP0Wr	I	CP0 写使能



EXLSet	I	用于置位 SR 的 EXL（流水线在 M 阶段产生）
EXLClr	I	用于清楚 SR 的 EXL（ERET 产生）
clk	I	时钟信号
reset	I	复位信号
IntReq	O	中断请求，输出至 CPU 控制器（是 HWInt/IM/EXL/IE 的函数）
EPC[31:0]	O	EPC 寄存器输出至 NPC
CP0Out[31:0]	O	CP0 输出数据（MFC0 产生，输出至 GPR）

序号	功能名称	功能描述
1	读出	DMOut=DM 中 DMAddr 地址中的数据。
2	写入	MemWr=1 时，将 DMIn 写入 DM 的 DMAddr 地址中。

## 四、 测试 CPU（见文档）

### 1、 P6 测试

### 2、 P7 新增冲突+异常+中断

## 四、 思考题

1. 我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？

软件/硬件（RW / HW）接口：在接口之上是中断处理程序和用于不同设备的设备驱动程序，在此之下是各种设备的控制器，如 CD-ROM 控制器、硬盘控制器、键盘控制器、打印机控制器、网络控制器等，它们都属于硬件。由于设备种类繁多，故该接口相当复杂

2. 在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

现代计算机由于时钟频率十分高，但是 DM 的读写周期根本降不不到 CPU 其他部件能接受的时钟周期，因此现代 CPU 采用将 DM 从 CPU 之中分离出来的方法，在 CPU 之中加上一个高速缓存器（cache）这样可以提高时钟频率。

### 3.BE 部件对所有的外设都是必要的吗？

不是。比如这次教程要求的 Timer 不支持 sb/sh 这两条指令，那么就不需要 BE。

4.请开发一个主程序以及定时器的 **exception handler**。整个系统完成如下功能：

- ①定时器在主程序中被初始化为模式 0；
- ②定时器倒计时至 0 产生中断；
- ③handler 重置初值寄存器从而再次启动定时器的计数器。②及③被无限重复。
- ④主程序在初始化时将定时器初始化为模式 0，设定初值寄存器的初值为某个值，如 100 或 1000。（注意，主程序可能需要涉及对 CP0.SR 的编程，推荐阅读过后文后再进行。）

```
.text
#写初值寄存器
ori $10,$0,0x7f10
ori $2,100
sw $20,4($10)
#加载模式 0
ori $2,9
sw $2,0($10)

#中断处理程序
.ktext 0x00004180
ori $10,0x7f10
ori $2,9
sw $2,0($10)
eret
```

5.请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

键盘、鼠标这类的低速设备是通过中断请求的方式进行 I/O 操作的。即当键盘上按下一个按键的时候，键盘会发出一个中断信号，中断信号经过 Bridge 传到 CPU 的 CP0 寄存器，然后 CP0 根据不同的中断号执行不同的中断响应程序，并进行相应的 I/O 操作，把按下的按键编码（或者鼠标操作）读到寄存器，最后放入内存中。

7、在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？

相应的测试样例是什么样的？请有条理的罗列出来。（非常重要）

暂停

类型	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
R 型	LD-E-RS	lw	D	RS	lw \$4, 0(\$5) addu \$4, \$4, \$5
	LD-E-RT	lw	D	RT	lw \$4, 0(\$5) addu \$4, \$5, \$4
I 型	LD-E-RS	lw	D	RS	lw \$4, 0(\$5) ori \$5, \$4, 0xffff
LD 型	LD-E-RS	lw	D	RS	lw \$4, 0(\$5) lw \$3, 4(\$4)
ST 型	LD-E-RS	lw	D	RS	lw \$4, 0(\$5) sw \$3, 4(\$4)
JR	R-E-RS	addu	D	RS	addu \$4, \$4, \$5 jr \$4
	I-E-RS	ori	D	RS	ori \$4, 0xffff jr \$4
	LD-E-RS	lw	D	RS	lw \$4, 0(\$5) jr \$4
	LD-M-RS	lw	D	RS	lw \$4, 0(\$5) nop jr \$4
B 型	R-E-RS	addu	D	RS	addu \$4, \$4, \$5 beq \$4, \$5, loop
	I-E-RS	ori	D	RS	ori \$4, 0xffff beq \$4, \$5, loop
	LD-E-RS	lw	D	RS	lw \$4, 0(\$5) beq \$4, \$5, loop
	LD-M-RS	lw	D	RS	lw \$4, 0(\$5) nop beq \$4, \$5, loop
	R-E-RT	addu	D	RT	addu \$4, \$4, \$5

					beq \$5, \$4, loop
	I-E-RT	ori	D	RT	ori \$4, 0xffff beq \$5, \$4, loop
	LD-E-RT	lw	D	RT	lw \$4, 0(\$5) beq \$5, \$4, loop
	LD-M-RT	lw	D	RT	lw \$4, 0(\$5) nop beq \$5, \$4, loop

# 转发

类型	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
R 型 (以 addu 为例)	R-W-RS	addu	D	RS	addu \$4, \$4, \$5 nop nop addu \$4, \$4, \$5
	I-W-RS	ori	D	RS	ori \$4, \$5, 0xffff nop nop addu \$4, \$4, \$5
	LD-W-RS	lw	D	RS	lw \$4, 0(\$5) nop nop addu \$4, \$4, \$5
	JAL-W-RS	jal	D	RS	jal loop nop nop addu \$1, \$31, \$1
	R-W-RT	addu	D	RT	addu \$4, \$4, \$5 nop nop addu \$4, \$5, \$4
	I-W-RT	ori	D	RT	ori \$4, \$5, 0xffff nop nop addu \$4, \$5, \$4
	LD-W-RT	lw	D	RT	lw \$4, 0(\$5) nop nop addu \$4, \$5, \$4
	JAL-W-RT	jal	D	RT	jal loop

					nop nop addu \$1, \$1, \$31
	R-W-RS	addu	E	RS	addu \$4, \$4, \$5 nop addu \$4, \$4, \$5 nop
	I-W-RS	ori	E	RS	ori \$4, \$5, 0xffff nop addu \$4, \$4, \$5 nop
	LD-W-RS	lw	E	RS	lw \$4, 0(\$5) nop addu \$4, \$4, \$5 nop
	JAL-W-RS	jal	E	RS	jal loop nop addu \$1, \$31, \$1 nop
	R-M-RS	addu	E	RS	addu \$4, \$4, \$5 addu \$4, \$4, \$5 nop
	I-M-RS	ori	E	RS	ori \$4, \$5, 0xffff addu \$4, \$4, \$5 nop
	JAL-M-RS	jal	E	RS	jal loop addu \$1, \$31, \$1 nop
	R-W-RT	addu	E	RT	addu \$4, \$4, \$5 nop addu \$4, \$5, \$4 nop
	I-W-RT	ori	E	RT	ori \$4, \$5, 0xffff nop addu \$4, \$5, \$4 nop
	LD-W-RT	lw	E	RT	lw \$4, 0(\$5) nop addu \$4, \$5, \$4 nop
	JAL-W-RT	jal	E	RT	jal loop nop addu \$1, \$1, \$31 nop

	R-M-RT	addu	E	RT	addu \$4, \$4, \$5 addu \$4, \$5, \$4 nop
	I-M-RT	ori	E	RT	ori \$4, \$5, 0xffff addu \$4, \$5, \$4 nop
	JAL-M-RT	jal	E	RT	jal loop addu \$1, \$1, \$31 nop
I 型 (以 ori 为 例)	R-W-RS	addu	D	RS	addu \$4, \$4, \$5 nop nop ori \$4, \$4, 0xffff
	I-W-RS	ori	D	RS	ori \$4, \$5, 0xffff nop nop ori \$4, \$4, 0x0000
	LD-W-RS	lw	D	RS	lw \$4, 0(\$5) nop nop ori \$4, \$4, 0xffff
	JAL-W-RS	jal	D	RS	jal loop nop nop ori \$1, \$31, 0xffff
	R-W-RS	addu	E	RS	addu \$4, \$4, \$5 nop ori \$4, \$4, 0xffff nop
	I-W-RS	ori	E	RS	ori \$4, \$5, 0xffff nop ori \$4, \$4, 0x0f0f nop
	LD-W-RS	lw	E	RS	lw \$4, 0(\$5) nop ori \$4, \$4, 0xffff nop
	JAL-W-RS	jal	E	RS	jal loop nop ori \$1, \$31, 0xffff nop
	R-M-RS	addu	E	RS	addu \$4, \$4, \$5 ori \$4, \$4, 0xffff nop

	I-M-RS	ori	E	RS	ori \$4, \$5, 0xffff ori \$4, \$4, 0xf0f0 nop
	JAL-M-RS	jal	E	RS	jal loop ori \$1, \$31, 0xffff0 nop
LD 型	R-W-RS	addu	D	RS	addu \$4, \$4, \$5 nop nop lw \$5, 0(\$4)
	I-W-RS	ori	D	RS	ori \$4, \$5, 0xffff nop nop lw \$5, 0(\$4)
	LD-W-RS	lw	D	RS	lw \$4, 0(\$5) nop nop lw \$5, 0(\$4)
	JAL-W-RS	jal	D	RS	jal loop nop nop lw \$5, 0(\$31)
	R-W-RS	addu	E	RS	addu \$4, \$4, \$5 nop lw \$5, 0(\$4) nop
	I-W-RS	ori	E	RS	ori \$4, \$5, 0xffff nop lw \$5, 0(\$4) nop
	LD-W-RS	lw	E	RS	lw \$4, 0(\$5) nop lw \$5, 0(\$4) nop
	JAL-W-RS	jal	E	RS	jal loop nop lw \$5, 0(\$31) nop
	R-M-RS	addu	E	RS	addu \$4, \$4, \$5 lw \$5, 0(\$4) nop
	I-M-RS	ori	E	RS	ori \$4, \$5, 0xffff lw \$5, 0(\$4) nop

	JAL-M-RS	jal	E	RS	jal loop lw \$5, 0(\$31) nop
ST 型	R-W-RS	addu	D	RS	addu \$4, \$4, \$5 nop nop sw \$5, 0(\$4)
	I-W-RS	ori	D	RS	ori \$4, \$5, 0xffff nop nop sw \$5, 0(\$4)
	LD-W-RS	lw	D	RS	lw \$4, 0(\$5) nop nop sw \$5, 0(\$4)
	JAL-W-RS	jal	D	RS	jal loop nop nop sw \$5, 0(\$31)
	R-W-RT	addu	D	RT	addu \$4, \$4, \$5 nop nop sw \$4, 0(\$5)
	I-W-RT	ori	D	RT	ori \$4, \$5, 0xffff nop nop sw \$4, 0(\$5)
	LD-W-RT	lw	D	RT	lw \$4, 0(\$5) nop nop sw \$4, 0(\$6)
	JAL-W-RT	jal	D	RT	jal loop nop nop sw \$31, 0(\$5)
	R-W-RS	addu	E	RS	addu \$4, \$4, \$5 nop sw \$5, 0(\$4) nop
	I-W-RS	ori	E	RS	ori \$4, \$5, 0xffff nop sw \$5, 0(\$4) nop
	LD-W-RS	lw	E	RS	lw \$4, 0(\$5)



					nop sw \$5, 0(\$4) nop
	JAL-W-RS	jal	E	RS	jal loop nop sw \$5, 0(\$31) nop
	R-W-RT	addu	E	RT	addu \$4, \$4, \$5 nop sw \$4, 0(\$5) nop
	I-W-RT	ori	E	RT	ori \$4, \$5, 0xffff nop sw \$4, 0(\$5) nop
	LD-W-RT	lw	E	RT	lw \$4, 0(\$5) nop sw \$4, 0(\$6) nop
	JAL-W-RT	jal	E	RT	jal loop nop sw \$31, 0(\$5) nop
	R-W-RT	addu	M	RT	addu \$4, \$4, \$5 sw \$4, 0(\$5) nop nop
	I-W-RT	ori	M	RT	ori \$4, \$5, 0xffff sw \$4, 0(\$5) nop nop
	LD-W-RT	lw	M	RT	lw \$4, 0(\$5) sw \$4, 0(\$6) nop nop
	JAL-W-RT	jal	M	RT	jal loop sw \$31, 0(\$5) nop nop
	R-M-RS	addu	E	RS	addu \$4, \$4, \$5 sw \$5, 0(\$4) nop
	I-M-RS	ori	E	RS	ori \$4, \$4, \$5 sw \$5, 0(\$4)

					nop
	JAL-M-RS	jal	E	RS	jal loop sw \$5, 0(\$31) nop
JR	R-W-RS	addu	D	RS	addu \$4, \$4, \$5 nop nop jr \$4
	I-W-RS	ori	D	RS	ori \$4, \$5, 0xffff nop nop jr \$4
	LD-W-RS	lw	D	RS	lw \$4, 0(\$5) nop nop jr \$4
	JAL-W-RS	jal	D	RS	jal loop nop nop jr \$31
	R-M-RS	addu	D	RS	addu \$4, \$4, \$5 nop jr \$4
	I-M-RS	ori	D	RS	ori \$4, \$5, 0xffff nop jr \$4
	JAL-M-RS	jal	D	RS	jal loop nop jr \$31
行为未 定义	JAL-E-RS	jal	D	RS	jal loop jr \$31
B 型	R-W-RS	addu	D	RS	addu \$4, \$4, \$5 nop nop beq \$4, \$3, loop
	I-W-RS	ori	D	RS	ori \$4, \$5, 0xffff nop nop beq \$4, \$3, loop
	LD-W-RS	lw	D	RS	lw \$4, 0(\$5) nop nop beq \$4, \$3, loop
	JAL-W-RS	jal	D	RS	jal loop

					nop nop beq \$31, \$3, loop
	R-W-RT	addu	D	RT	addu \$4, \$4, \$5 nop nop beq \$3, \$4, loop
	I-W-RT	ori	D	RT	ori \$4, \$5, 0xffff nop nop beq \$3, \$4, loop
	LD-W-RT	lw	D	RT	lw \$4, 0(\$5) nop nop beq \$3, \$4, loop
	JAL-W-RT	jal	D	RT	jal loop nop nop beq \$3, \$31, loop
	R-M-RS	addu	D	RS	addu \$4, \$4, \$5 nop beq \$4, \$3, loop
	I-M-RS	ori	D	RS	ori \$4, \$5, 0xffff nop beq \$4, \$3, loop
	JAL-M-RS	jal	D	RS	jal loop nop beq \$31, \$3, loop
	R-M-RT	addu	D	RT	addu \$4, \$4, \$5 nop beq \$3, \$4, loop
	I-M-RT	ori	D	RT	ori \$4, \$5, 0xffff nop beq \$3, \$4, loop
	JAL-M-RT	jal	D	RT	jal loop nop beq \$3, \$31, loop
行为未 定义	JAL-E-RS	jal	D	RS	jal loop beq \$31, \$3, loop
	JAL-E-RT	jal	D	RT	jal loop beq \$3, \$31, loop

附： 比对程序（C）

```
#include<stdio.h>
#include<string.h>
char a[100];
char b[100];
int main(){
    FILE *fp1=fopen("mips.txt","r");
    FILE *fp2=fopen("verilog.txt","r");
    //freopen("res.txt","w+",stdout);
    while((fscanf(fp1,"%s",a))!=EOF){
        fscanf(fp2,"%s",b);
        if(strcmp(a,b)==0)printf("right:r:%s w:%s\n",b,a);
        else {printf("wrong:r:%s w:%s\n",b,a);break;}
    }
    return 0;
}
```