**协处理器访问指令的实现**

CP0协处理器类似于HI、LO，介绍mfc0、mtc0指令的格式、作用、用法。

# 协处理器介绍

协处理器通常表示处理器的一个可选部件，负责处理指令集的某个扩展，具有与处理器核独立的寄存器。MIPS架构提供了最多4个协处理器，分别是CP0~CP3，作用如下：



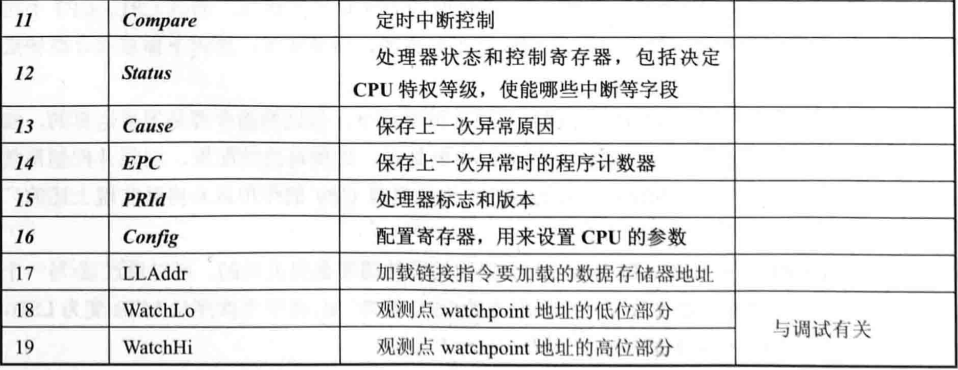
OpenMIPS没有浮点运算，CP1和CP3不用实现，CP2也没有作用，不用实现。因此只需实现CP0.

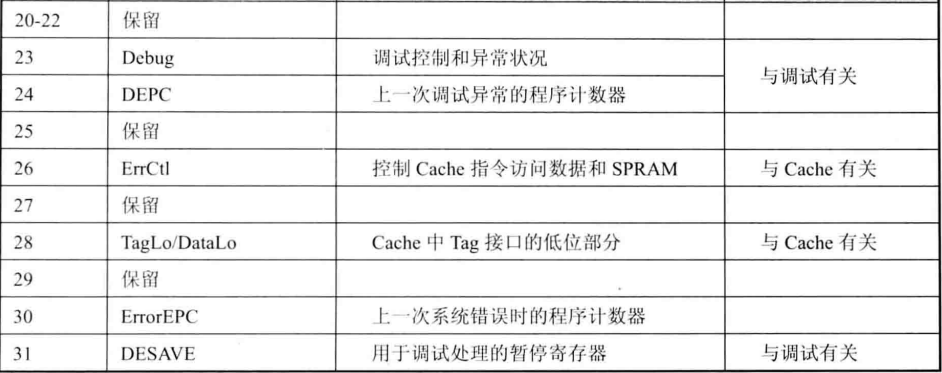
CP0主要负责的工作如下：

* **配置CPU工作状态**
* **告诉缓存控制**
* **异常控制**
* **存储管理单元控制**
* **其他，例如时钟、时间计数器、奇偶校验错误检测等**

# 协处理器CP0中的寄存器





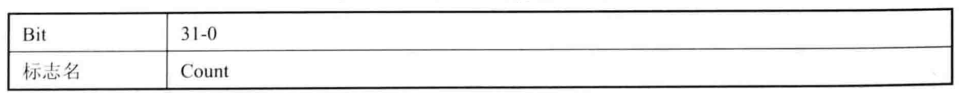


这里面有许多寄存器是关于缓存。MMU、TLB、调试有关的，OpenMIPS是一个轻量级处理器，并不用实现这些，只需实现上表中加粗的7个寄存器即可。

下面介绍这7个寄存器的格式和作用：

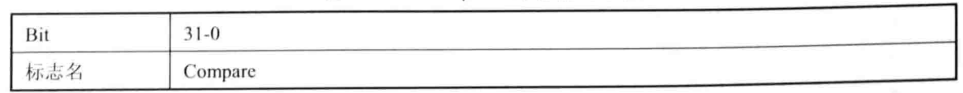
1. **Count寄存器（标号为9）**

Count寄存器为32位寄存器，不停计数，计数频率一般与CPU时钟频率相同，循环计数。Count寄存器可读可写。



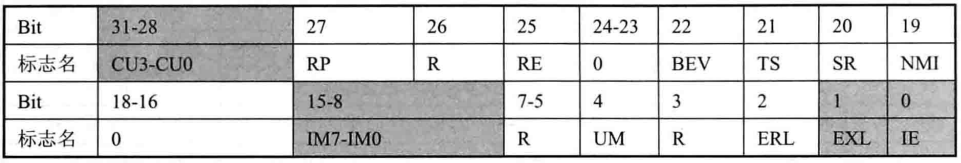
1. **Compare寄存器（标号为11）**

32位寄存器，与Count寄存器一起完成定时中断功能。当两者的值一样时，会产生定时中断。中断会一直保持，直到有数据被写入Compare寄存器。Compare寄存器可读可写。



1. **Status寄存器（标号为12）**

也是一个32位，可读可写的寄存器，用来控制处理器的操作模式、中断使能以及诊断状态。字段如下表所示：



* **CU3-CU0**

表示协处理器是否可用（Coprocessor Usability），分别控制协处理器CP0~CP3。为0时，表示不可用，因为只有CP0可用，可以直接设置本字段为4b’0001.

* **RP**

表示是否启动低功耗模式（Reduced Power），本处理器没有实现次功能。

* **RE**

用来改变用户态模式下的字节次序，1表示改变，0表示不改变。如大端（MSB）和小端（LSB），本处理器没有实现该功能，本字段没有用。

* **BEV**

表示是否使用启动异常向量（Bootstrap Exception Vector），本处理器没有实现该功能，本字段没有用。

* **TS**

表示是否关闭TLB（TLB Shutdown），为1表示关闭，本处理器没有实现该功能，本字段没有用。

* **SR**

表示是否软重启（Soft Reset），为1表示重启异常是由软中断引起的。

* **NMI**

表示是否是不可屏蔽中断（Non-Maskable Interrupt）.

* **IM7-IM0**

表示是否屏蔽响应中断（Interrupt Mask），0表示屏蔽，1表示不屏蔽，MIPS处理器有8个中断源，对应IM字段的8位，其中6 个中断源是处理器外部硬件中断，另外两个是软件中断，中断是否能被处理器响应是由Status寄存器和Cause寄存器共同决定，当Status的IM字段与Cause的IP字段都为1，且Status寄存器IE字段也是1时，处理器才响应中断。

* **UM**

表示是否为用户模式（User Mode），为1表示处理器运行在内核模式，为0表示运行在用户模式，但是MIPS处理器没有区分内核模式和用户模式，两种状态下权限一样，故该字段没有用。

* **ERL**

表示是否处于错误级，当处理器接收到坏消息时设置本字段为1。有些MIPS处理器会设置奇偶校验位或者纠错码，但是本处理器没有实现，故本字段没有用。

* **EXL**

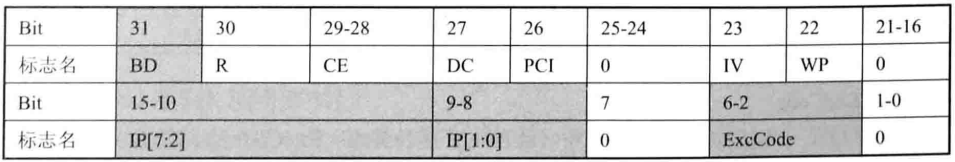
表示是否处于异常级（Exception Level），异常时，设置本字段为1，处理器会机内内核模式，并且禁止中断。

* **IE**

表示是否使能中断（Interrupt Enable），全局中断使能标志位，为1表示中断使能。

1. **Cause寄存器（标号为13）**

Cause寄存器主要记录最近一次异常发生的原因，也控制软件中断请求，字段如下表所示，除了IP[1:0]、IV和WP，其他字段都是只读的。



标识为R的字段是保留字段，灰色背景的字段需要理解。

* **BD**

当发生异常的指令处于分支延迟槽（Branch DelaySlot）时，该字段被设置为1.

* **DC**

使得Count寄存器停止计数

* **IV**

中断向量（Interrupt Vector）的选择与此字段有关，该字段为0标识使用一般中断向量，反之使用特殊中断向量。OpenMIPS使用一种简单的异常向量表的方式来处理中断，这个字段没有作用。

* **IP[7:2]**

中断挂起（Interrupt Pending），标识硬件中断：

IP[7]——5号硬件中断

IP[6]——4号硬件中断

IP[5]——3号硬件中断

IP[4]——2号硬件中断

IP[3]——1号硬件中断

IP[2]——0号硬件中断

* **IP[1:0]**

软件中断：

IP[1]——1号硬件中断

IP[0]——0号硬件中断

* **ExcCode**

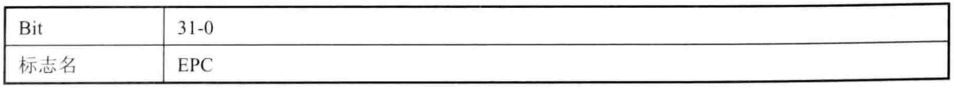
五位编码，记录异常的种类

如下表：



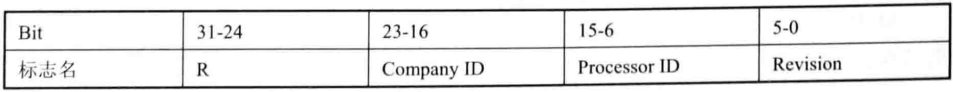
1. **EPC寄存器（标号为14）**

EPC是程序异常计数器（Exception Program Counter），用来存储异常返回地址，如果发生异常的指令的地址位于延迟槽中，则EPC存储的是前一条转移指令的地址。该寄存器可读可写。字段表如下：



1. **PRId寄存器（标号为15）**

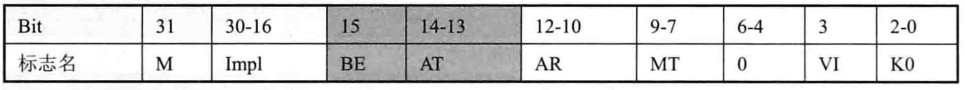
PRId寄存器是处理器表示（Processor Identifier）寄存器，包含有制造商信息，处理器类型以及处理器的版本等。字段表如下：



字段含义为字面含义，略。

1. **Config寄存器（标号为16）**

Config寄存器包含了与处理器相关的各种配置和功能信息，字段表如下：



* **M**

表示是否存在Config1寄存器，MIPS32架构中实际定义了4个配置寄存器：Config、Config1-3，OpenMIPS处理器只实现了Config寄存器，因此初始化时将该字段设置为0，表示没有Config1寄存器。

* **BE**

其值为1表示处理器工作在大端模式（MSB），为0表示处理器工作在小端模式（LSB）。OpenMIPS处理器工作在大端模式，设置本字段为1.

* **AT**

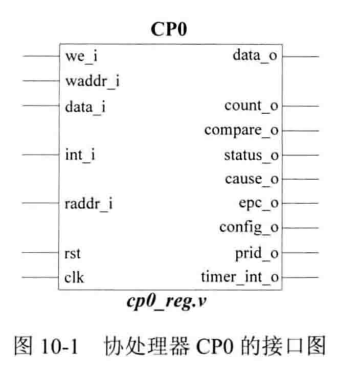
指令集发行版本（Architecture Revision）字段。

* **K0**

表示内存的Kseg0区域是否可缓存，其中Kseg0是内存中的一段空间，OpenMIPS没有实现缓存，本字段为3’b000.

# 协处理器CP0的实现

协处理器CP0的接口图如下：



实现方式类似于HI、LO寄存器，接口描述如下：





代码略。

该部分主要实现了CP0中的Count、Compare、Status、Cause、EPC、PRId、Config这7个寄存器。在这7个寄存器中，Config和PRId不可写，因此在上面的写入寄存器操作中没有写入这两个寄存器。此外，Cause寄存器只有其中的IP[1:0]、IV、WP三个字段可写，所以对Cause寄存器的写入是选择性的。

Cause寄存器的值每个时钟周期都会加一。

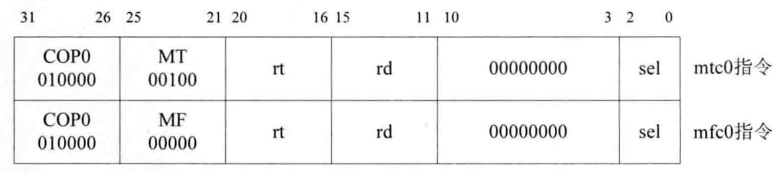
当Compare寄存器不为0，且Count寄存器的值等于Compare寄存器的值是，将时钟中断输出信号timer\_int\_o置为1，知道有新的数据写入Compare才改变。

MIPS32架构支持8个中断，有两个软件中断，支持的外部硬件中断只有6个，因此CP0模块的中断输入信号int\_i宽度是6.

Cause寄存器的第10~15bit是IP[7:2]就是控制的外部硬件中断，因此直接将int\_i赋值给Cause寄存器的10~15bit.

# 协处理器访问指令说明

MIPS32指令集架构中定义了2条协处理器访问指令，**mtc0、mfc0**，前者写入，后者读取，指令格式如下：



观察这两条指令，发现跟之前的指令略有不同，他们的前面6位指令码都是一样的，都为6b’010000，仅能通过21~25bit来区分，并且0~2bit为sel，实际上这个域取决于具体的MIPS32架构处理器，对我们这个OpenMIPS而言，是没有作用的。

指令具体用法和作用：

* **mtc0：**

用法： mtc0 rt, rd

作用： CPR[0,rd] <- GPR[rt]，将地址为rt的通用寄存器的值赋给协处理器CP0中地址为rd的寄存器。

* **mfc0：**

用法： mfc0 rt, rd

作用： GPR[rt] <- CPR[0,rd]，读出协处理器CP0中的地址为rd的寄存器的值，并赋给地址为rt的通用寄存器。

# 协处理器访问指令实现思路

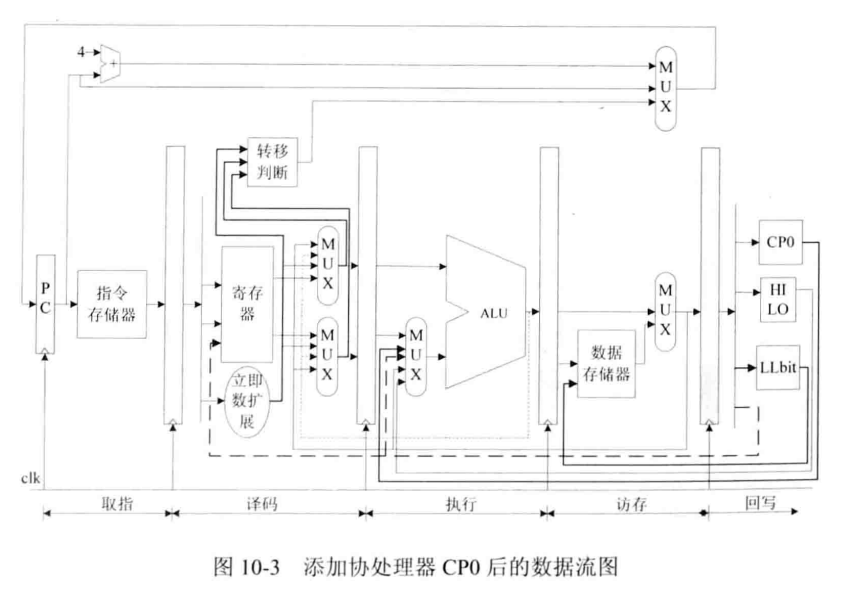
和HI、LO寄存器的访问一样，对CP0中所有寄存器的操作都放在**回写**阶段。

具体实现如下：

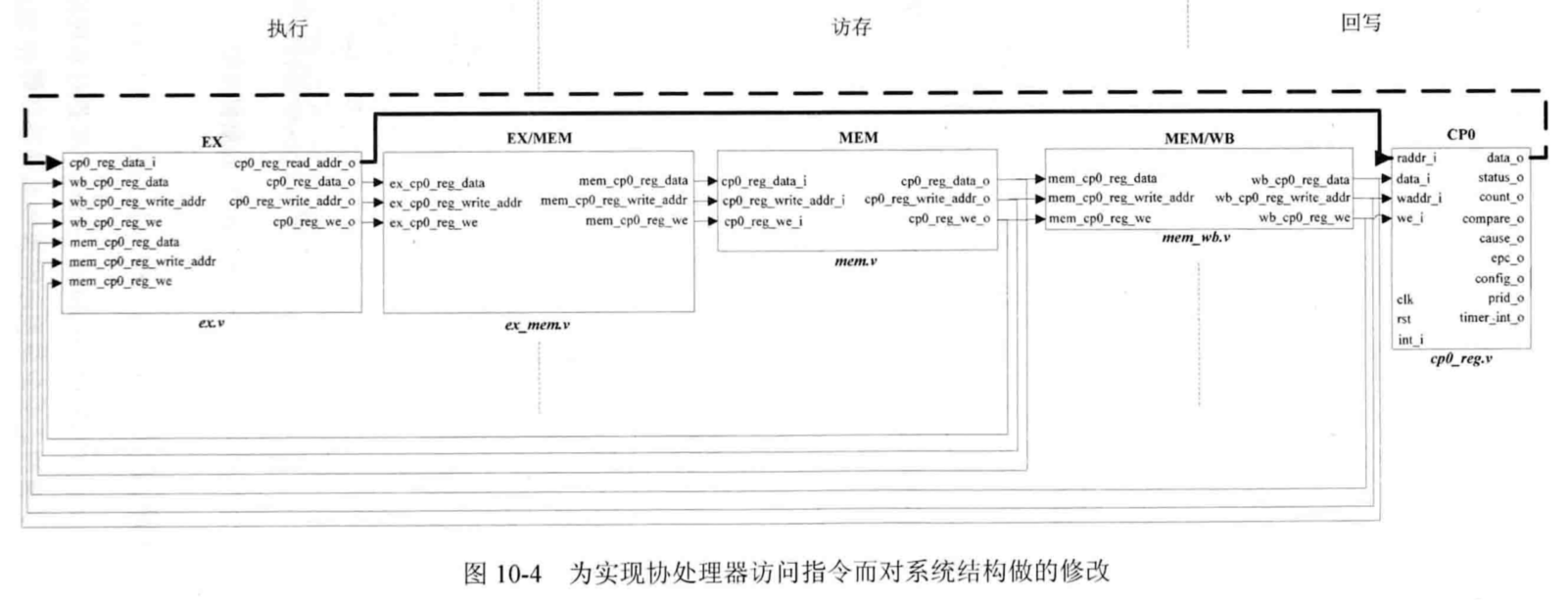
1. 在译码阶段取指令，读出地址为rt的通用寄存器的值；
2. 在执行阶段确定要写入的CP0的值，实际上就是地址为rt的通用寄存器的值，将信息传递到访存阶段；
3. 访存阶段将信息传递到回写阶段；
4. 回写阶段再根据这些信息修改地址为rd的通用寄存器

对于mfc0来说，在执行阶段获取CP0中指定寄存器的值，再传递到回写阶段，再修改地址为rt的通用寄存器。

## 5.1 数据流图的修改



## 5.2 系统结构的修改

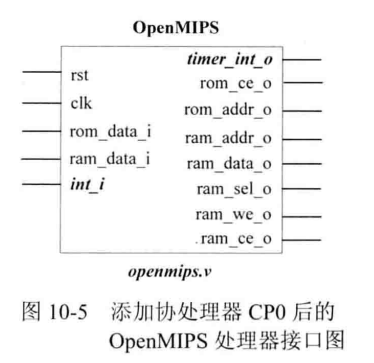


我们观察流图，发现若执行读指令mfc0，在执行阶段EX模块会直接通过接口cp0\_reg\_read\_addr\_o将地址送入CP0，并将data\_o回送回EX的cp0\_reg\_data\_i接口。

若为写操作，那么通过cp0\_reg\_data\_o, cp0\_reg\_write\_addr\_o, cp0\_reg\_we\_o一路传递到回写阶段，将其值送入CP0.

值得注意的是，对于读取CP0中的数据时，会发生**数据相关**问题，这个问题已经是老生常谈了，仅需使用数据前推就可以解决。我们将访存和回写阶段对CP0中寄存器的写信息前推到执行阶段的EX模块，由EX模块判断得到最新的值。

另外，CP0模块的输入int\_i是OpenMIPS处理器的输入，CP0的输出timer\_int\_o是OpenMIPS处理器的输出，因此可以得到OpenMIPS的接口示意图：



其中，int\_i宽度为6bit，表示6个外部硬件中断输入，timer\_int\_o宽度为1，表示是否有定时中断发生。

# 修改代码

## 6.1 修改译码阶段

· mfc0（读CP0）

需要写入通用寄存器，故wreg\_o为WriteEnable，地址为inst\_i[20:16]，即赋值给wd\_o.该指令不需要读取通用寄存器。

* mtc0（写CP0）

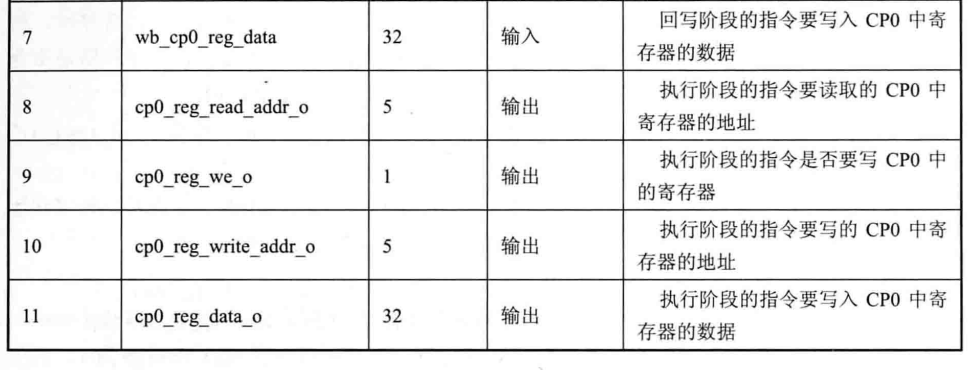
不需要写入通用寄存器，因此wreg\_o为WriteDisable，要读取一个通用寄存器，地址为inst\_i[20:16]，因此最终ID的输出reg1\_o就是地址为rt的通用寄存器的值。

## 6.2 修改执行阶段

### 6.2.1修改EX模块

新增接口表如下：





很简单没什么说的，需要判断数据相关。

### 6.2.2 修改EX/MEM模块

主要用来传递mtc0指令的数据到下一个流水，没什么难度，下表是新增接口：





## 6.3 修改访存阶段

### 6.3.1 修改MEM模块

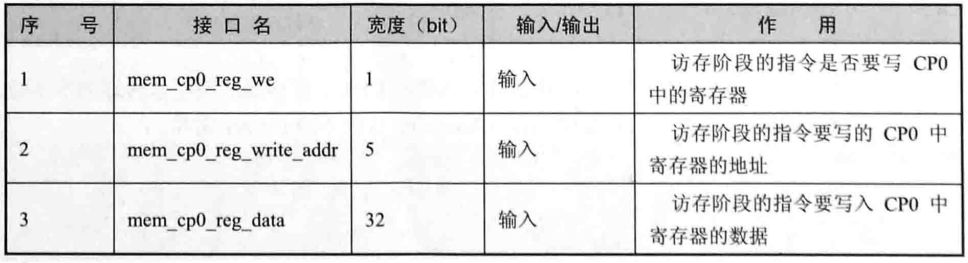
传递到回写阶段，接口描述如下表：



没啥好说的…

### 6.3.2 修改MEM/WB模块

接口描述如下表：



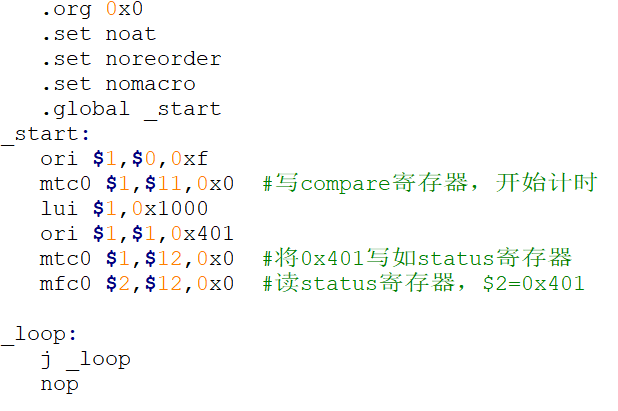


## 6.4 修改OpenMIPS顶层模块

细心即可，注意OpenMIPS本身新加了两个接口int\_i和timer\_int\_o，略。

# 测试结果及分析

使用下面的测试程序：

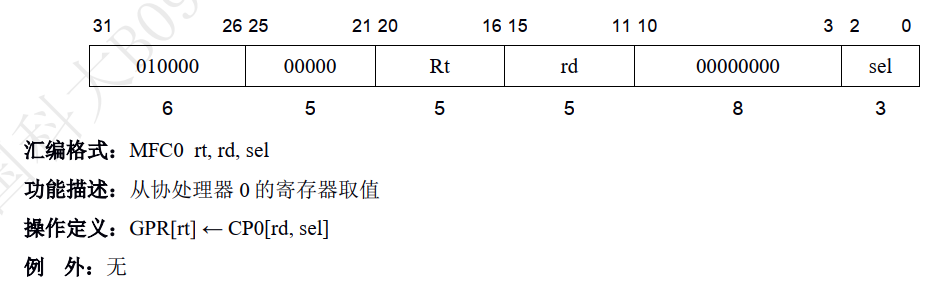


仿真结果如下图：



对于汇编指令我有一个疑问：mtc0 $1,$11,0x0表示什么意思，为什么会有三个操作数？

答：实际上mtc0和mfc0指令格式是这样的：



有三个操作数是在后面加上了sel，在这里sel默认都是0.

还有为什么汇编指令后面要用一个死循环？