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。

一、 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：

协处理器	作　用
CP0	系统控制
CP1	FPU
CP2	特定实现
CP3	FPU

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**

二、 协处理器 CP0 中的寄存器

标　号	寄存器助记符/名称	功能描述	备　注
0	Index	TLB 阵列的入口索引	这些都是与内存管理 MMU、地址翻译快表 TLB 有关的寄存器
1	Random	产生 TLB 阵列的随机入口索引	
2	EntryLo0	偶数虚拟页的入口地址的低位部分	
3	EntryLo1	奇数虚拟页的入口地址的低位部分	
4	Context	指向内存虚拟页表入口地址的指针	
5	PageMask	控制 TLB 入口中可变页面的大小	
6	Wired	控制固定的 TLB 入口的数目	
7	保留		
8	BadVAddr	记录最近一次地址相关异常的地址	
9	***Count***	**处理器计数周期**	
10	EntryHi	TLB 入口地址的高位部分	与内存管理 MMU、地址翻译快表 TLB 有关的寄存器

11	***Compare***	定时中断控制	
12	***Status***	处理器状态和控制寄存器，包括决定CPU特权等级，使能哪些中断等字段	
13	***Cause***	保存上一次异常原因	
14	***EPC***	保存上一次异常时的程序计数器	
15	***PRId***	处理器标志和版本	
16	***Config***	配置寄存器，用来设置CPU的参数	
17	LLAddr	加载链接指令要加载的数据存储器地址	
18	WatchLo	观测点watchpoint地址的低位部分	与调试有关
19	WatchHi	观测点watchpoint地址的高位部分	
20-22	保留		
23	Debug	调试控制和异常状况	与调试有关
24	DEPC	上一次调试异常的程序计数器	
25	保留		
26	ErrCtl	控制Cache指令访问数据和SPRAM	与Cache有关
27	保留		
28	TagLo/DataLo	Cache中Tag接口的低位部分	与Cache有关
29	保留		
30	ErrorEPC	上一次系统错误时的程序计数器	
31	DESAVE	用于调试处理的暂停寄存器	与调试有关

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。

Bit	31-0
标志名	Count

2．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。

Bit	31-0
标志名	Compare

3．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：

Bit	31-28	27	26	25	24-23	22	21	20	19
标志名	CU3-CU0	RP	R	RE	0	BEV	TS	SR	NMI
Bit	18-16	15-8		7-5	4	3	2	1	0
标志名	0	IM7-IM0		R	UM	R	ERL	EXL	IE

- **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。

4．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。

Bit	31	30	29-28	27	26	25-24	23	22	21-16
标志名	BD	R	CE	DC	PCI	0	IV	WP	0
Bit	15-10			9-8		7	6-2		1-0
标志名	IP[7:2]			IP[1:0]		0	ExcCode		0

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：

ExcCode 编码	助 记 符	描　　述
0	Int	中断
1	Mod	TLB 修改异常或者保留
2	TLBL	TLB 加载异常或者取指异常或者保留
3	TLBS	TLB 存储异常或者保留
4	AdEL	加载或取指过程中，地址错误异常
5	AdES	存储过程中，地址错误异常
6	IBS	取指过程中，总线错误异常
7	DBE	加载或存储数据过程中，总线错误异常
8	Sys	系统调用指令 Syscall
9	Bp	断点异常
10	RI	执行未定义指令引起的异常
11	CpU	协处理器不可用异常
12	Ov	整数溢出异常
13	Tr	自陷指令引起的异常
14-22	-	保留
23	WATCH	访问 WatchHi\WatchLo 地址
24	MCheck	机器检测，CPU 检测到 CPU 控制系统中的灾难性错误
25-31	-	保留

5．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：

Bit	31-0
标志名	EPC

6．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：

Bit	31-24	23-16	15-6	5-0
标志名	R	Company ID	Processor ID	Revision

字段含义为字面含义，略。

7．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：

Bit	31	30-16	15	14-13	12-10	9-7	6-4	3	2-0
标志名	M	Impl	BE	AT	AR	MT	0	VI	K0

- **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．

三、 协处理器 CP0 的实现

协处理器 CP0 的接口图如下：

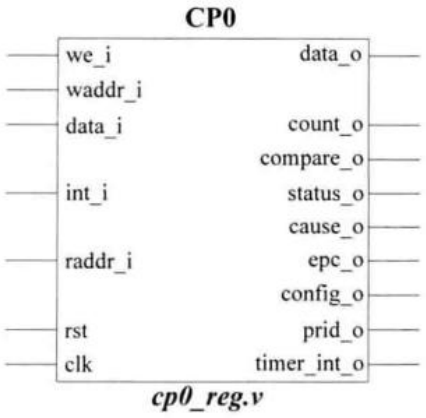


图 10-1 协处理器 CP0 的接口图

实现方式类似于 HI、LO 寄存器，接口描述如下：

序　号	接 口 名	宽度（bit）	输入/输出	作　用
1	rst	1	输入	复位信号
2	clk	1	输入	时钟信号
3	raddr_i	5	输入	要读取的 CP0 中寄存器的地址
4	int_i	6	输入	6 个外部硬件中断输入
5	we_i	1	输入	是否要写 CP0 中的寄存器
6	waddr_i	5	输入	要写的 CP0 中寄存器的地址
7	wdata_i	32	输入	要写入 CP0 中寄存器的数据
8	data_o	32	输出	读出的 CP0 中某个寄存器的值
9	count_o	32	输出	Count 寄存器的值
10	compare_o	32	输出	Compare 寄存器的值
11	status_o	32	输出	Status 寄存器的值
12	cause_o	32	输出	Cause 寄存器的值
13	epc_o	32	输出	EPC 寄存器的值
14	config_o	32	输出	Config 寄存器的值
15	prid_o	32	输出	PRId 寄存器的值
16	timer_int_o	1	输出	是否有定时中断发生

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。

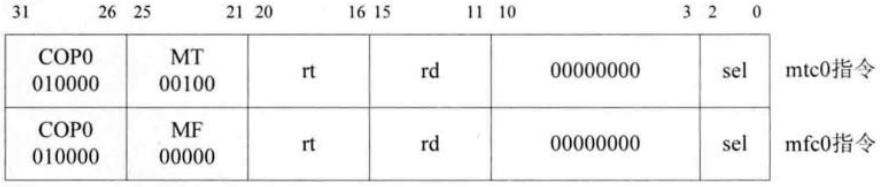
当 Compare 寄存器不为 0，且 Count 寄存器的值等于 Compare 寄存器的值是，将时钟中断输出信号 timer_int_o 置为 1，知道有新的数据写入 Compare 才改变。

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.

四、 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。

五、 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

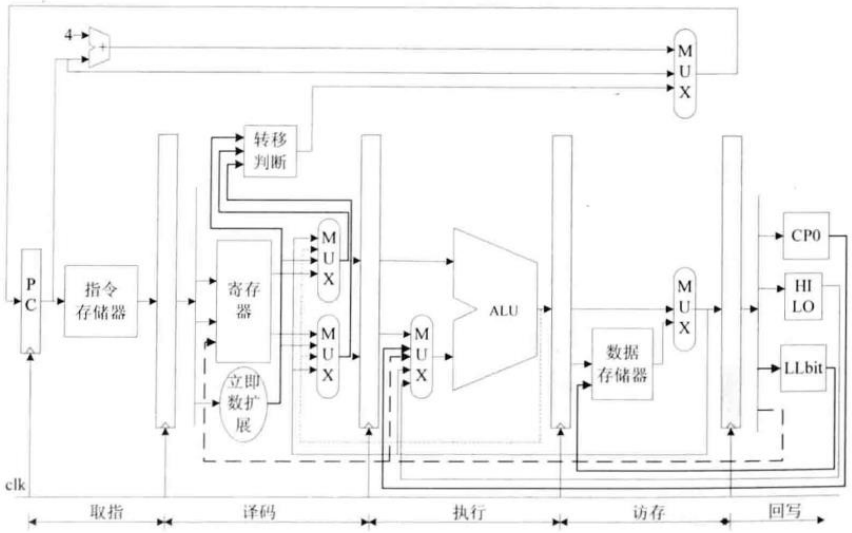


图 10-3 添加协处理器 CP0 后的数据流图

5.2 系统结构的修改

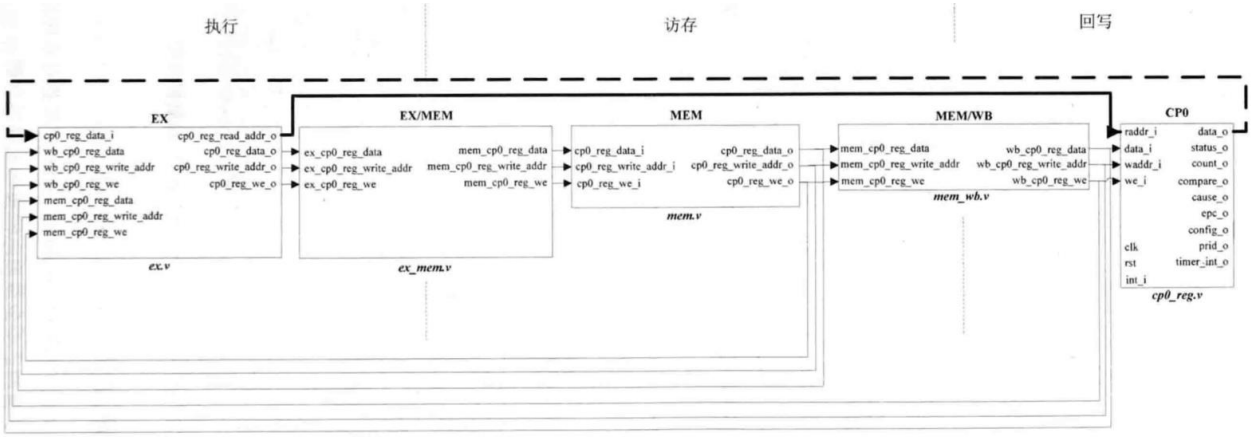


图 10-4 为实现协处理器访问指令而对系统结构做的修改

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：

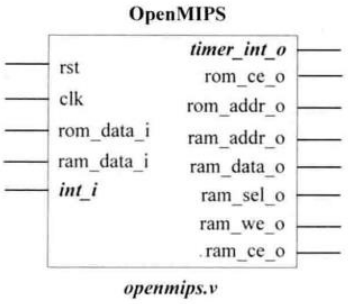


图 10-5 添加协处理器 CP0 后的 OpenMIPS 处理器接口图

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。

六、 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：

序　号	接 口 名	宽度（bit）	输入/输出	作　用
1	cp0_reg_data_i	32	输入	从 CP0 模块读取的指定寄存器的值
2	mem_cp0_reg_we	1	输入	访存阶段的指令是否要写 CP0 中的寄存器
3	mem_cp0_reg_write_addr	5	输入	访存阶段的指令要写的 CP0 中寄存器的地址
4	mem_cp0_reg_data	32	输入	访存阶段的指令要写入 CP0 中寄存器的数据
5	wb_cp0_reg_we	1	输入	回写阶段的指令是否要写 CP0 中的寄存器
6	wb_cp0_reg_write_addr	5	输入	回写阶段的指令要写的 CP0 中寄存器的地址
7	wb_cp0_reg_data	32	输入	回写阶段的指令要写入 CP0 中寄存器的数据
8	cp0_reg_read_addr_o	5	输出	执行阶段的指令要读取的 CP0 中寄存器的地址
9	cp0_reg_we_o	1	输出	执行阶段的指令是否要写 CP0 中的寄存器
10	cp0_reg_write_addr_o	5	输出	执行阶段的指令要写的 CP0 中寄存器的地址
11	cp0_reg_data_o	32	输出	执行阶段的指令要写入 CP0 中寄存器的数据

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：

序　号	接 口 名	宽度（bit）	输入/输出	作　用
1	ex_cp0_reg_we	1	输入	执行阶段的指令是否要写 CP0 中的寄存器
2	ex_cp0_reg_write_addr	5	输入	执行阶段的指令要写的 CP0 中寄存器的地址
3	ex_cp0_reg_data	32	输入	执行阶段的指令要写入 CP0 中寄存器的数据
4	mem_cp0_reg_we	1	输出	访存阶段的指令是否要写 CP0 中的寄存器
5	mem_cp0_reg_write_addr	5	输出	访存阶段的指令要写的 CP0 中寄存器的地址
6	mem_cp0_reg_data	32	输出	访存阶段的指令要写入 CP0 中寄存器的数据

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：

序　号	接 口 名	宽度（bit）	输入/输出	作　用
1	cp0_reg_we_i	1	输入	访存阶段的指令是否要写 CP0 中的寄存器
2	cp0_reg_write_addr_i	5	输入	访存阶段的指令要写的 CP0 中寄存器的地址
3	cp0_reg_data_i	32	输入	访存阶段的指令要写入 CP0 中寄存器的数据
4	cp0_reg_we_o	1	输出	访存阶段的指令最终是否要写 CP0 中的寄存器
5	cp0_reg_write_addr_o	5	输出	访存阶段的指令最终要写的 CP0 中寄存器的地址
6	cp0_reg_data_o	32	输出	访存阶段的指令最终要写入 CP0 中寄存器的数据

没啥好说的…

6.3.2 修改 MEM/WB 模块

接口描述如下表：

序　号	接 口 名	宽度（bit）	输入/输出	作　用
1	mem_cp0_reg_we	1	输入	访存阶段的指令是否要写 CP0 中的寄存器
2	mem_cp0_reg_write_addr	5	输入	访存阶段的指令要写的 CP0 中寄存器的地址
3	mem_cp0_reg_data	32	输入	访存阶段的指令要写入 CP0 中寄存器的数据

序　号	接 口 名	宽度（bit）	输入/输出	作　用
4	wb_cp0_reg_we	1	输出	回写阶段的指令是否要写 CP0 中的寄存器
5	wb_cp0_reg_write_addr	5	输出	回写阶段的指令要写的 CP0 中寄存器的地址
6	wb_cp0_reg_data	32	输出	回写阶段的指令要写入 CP0 中寄存器的数据

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。

七、 测试结果及分析

使用下面的测试程序：

```
    .org 0x0
    .set noat
    .set noreorder
    .set nomacro
    .global _start
_start:
    ori $1,$0,0xf
    mtc0 $1,$11,0x0   #写compare寄存器，开始计时
    lui $1,0x1000
    ori $1,$1,0x401
    mtc0 $1,$12,0x0   #将0x401写如status寄存器
    mfc0 $2,$12,0x0   #读status寄存器，$2=0x401

_loop:
    j _loop
    nop
```

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：

31 26	25 21	20 16	15 11	10 3	2 0
010000	00000	Rt	rd	00000000	sel
6	5	5	5	8	3

汇编格式： MFC0 rt, rd, sel

功能描述： 从协处理器 0 的寄存器取值

操作定义： GPR[rt] ← CP0[rd, sel]

例　外： 无

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？