异常与中断

异常与中断

同学们可能在之前的程序设计学习过程中接触过异常这个概念， Python、Java 等语言中都有异常处理的机制。而我们现在所谈论的是与上述“异常”有所不同的东西——**硬件异常**，比如加法**溢出**，乘除法**除零**这些不在原本部件设计范围内的运算发生后， CPU 就会产生一个硬件异常。而所谓的中断，更多情况是来自外部，例如计时器、输入设备（更加详细准确的定义请同学们参考相应的资料）、存储设备等。从这个角度出发，在笔者看来中断比起异常是更为**宽泛**的，我们可以把异常看做是一种 CPU 自身产生的中断（内部中断），因此相应的 CPU 就能获得更多异常相关的信息。**下文中为方便表述将异常与中断统称为中断。**总的来看，异常和中断就是 CPU 因为某种内部事件或者外部事件，正常运行的运行进程被中止，需要进行相关的处理。

精确异常

对于硬件异常，我们能明确指出是哪条指令导致了异常，并称这条指令为异常受害指令。精确异常的特性是，在异常受害指令**前面的所有指令都执行完毕**，而**受害指令及其后续指令都像从来没有开始**（准确说是当异常处理结束后重新执行这些指令，与未发生异常时执行这些指令的效果一样）。这样的处理思路使得从使用者的角度来看， CPU 执行异常处理是顺序执行的，而隐藏了流水线设计的细节。

相关指令

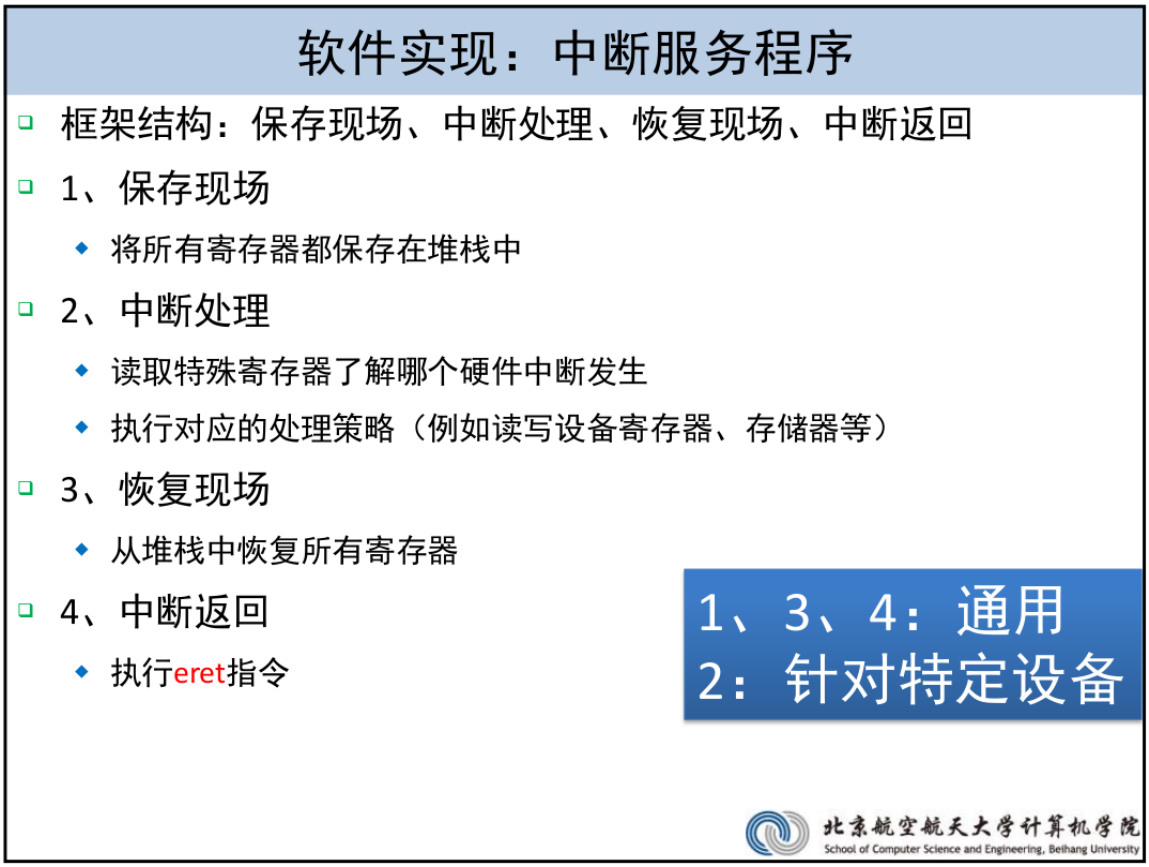
从这个角度看，引入中断和异常机制后，我们就拥有了一个**有多种状态**的 CPU ，一种是正常流水状态，一种是中断处理状态，为了对此进行相关的控制，我们需要引入新的部件 **CP0（Coprocessor 0）** ， CP0 的具体功能与设计我们在此先按下不表。而拥有了 CP0 后的 CPU 就能够支持如下的指令（详情请参阅指令集）：

- MTC0：通用寄存器值写入 CP0 寄存器
- MFC0：读取 CP0 寄存器至通用寄存器
- ERET：从中断中返回

细心的同学可能会发现，其中并没有进入中断状态的指令，这是由于中断产生时，我们的 CPU 会直接改写 PC 至中断处理程序所在地址，自主进入中断处理程序。然后通过 MFC0 、 MTC0 等指令从 CP0 中获取相关信息，设置 CPU 状态，进行相关处理后，通过 ERET 返回正常状态。

中断处理程序

前文已叙，当 CPU 遇到中断时，会改写相应 PC，进入中断处理程序。而中断处理程序的大致框架如下：



所谓**保存现场**，即是将当前 CPU 运行状态给保留下来，通过之前对函数调用的学习我们可以知道，CPU 运行状态中最重要的就是寄存器的状态，我们需要在中断处理程序中把所有寄存器存入堆栈，在返回时再予以恢复，以此来达成对状态的恢复。同时我们需要注意的是，在中断处理程序中，我们可能需要对 CP0 中寄存器进行操作，请大家参阅《See MIPS Run Linux》了解更多信息。另外，针对外部中断，我们也需要对外部硬件进行操作。

中断处理程序为软件实现，不需要同学们在 P7 实现，仅作为了解即可。

中断和返回

那么，在中断/异常产生时，我们的 CPU 都要做哪些事情呢?

简单来说，在中断/异常产生时，CPU 会跳到预先设定好的异常处理地址。这个地址就像 CPU 执行从 0x3000 开始一样，是 CPU 的根本属性。在我们的 CPU 中，这个地址是 0x4180。一旦 CP0 判断进入中断、异常，PC 就跳转至 0x4180。

此外，还要将发生中断/异常时的 PC 写入 CP0 中的 EPC 寄存器，并设置一些 CP0 中的寄存器，比如设置中断异常种类等。具体信息请参考 CP0 协处理器设计约束（暂未开放）。中断处

理程序会根据此时填入的信息，进行相应的处理。

特别地，由于我们的 CPU 是不支持异常重入的，在 CPU 处于异常处理状态时，不能再继续响应中断信号。当然，CPU 状态这个信息，也是记录在 CP0 的寄存器中的。

在异常返回时（ERET 指令），CPU 会跳到 EPC 记录的地址，并修改 CP0 中的若干状态信息。值得一提的是，中断处理前后的 PC 可能并不相同，因为中断处理程序可能会更改 EPC 的值。同样的，通用寄存器的值在异常处理前后也有可能不同。具体保存哪些寄存器，恢复哪些寄存器，都是由软件来决定的。

利用 MARS 验证中断处理框架

尽管在 MARS 中，我们只能针对异常进行模拟，无法模拟外部中断。但由我们对异常与中断的了解可以知道，两者的处理是类似的。因此你可以在 MARS 中先验证你的中断/异常处理框架是否正确。方法是：你让某条指令出错（溢出等），然后看 MARS 能否进入你写的exception handler。至于你如何处理这个错误，则是次要问题。

你可以按照下图的方式添加 Exception handler：

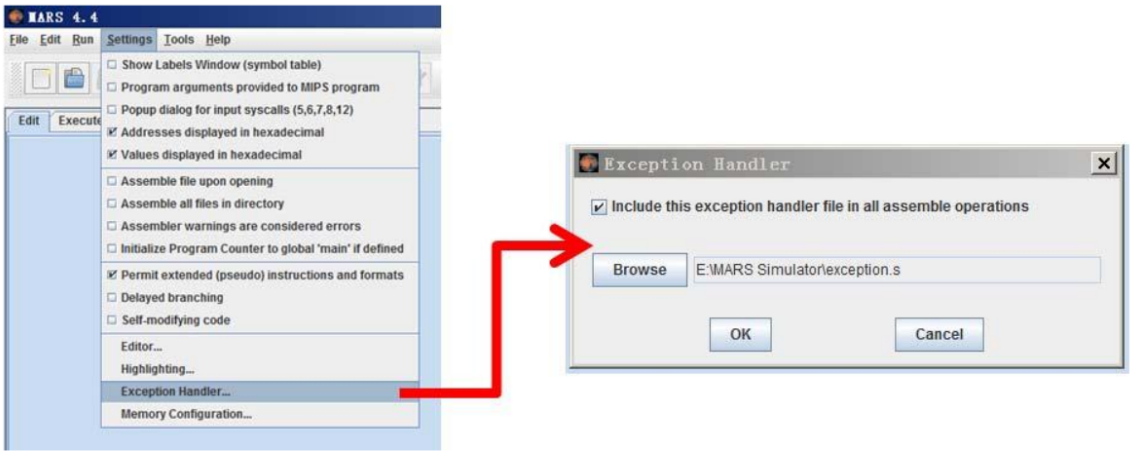


图2 在MARS中验证Exception handler机制

另外，由下图 MARS 的内存分配图我们可以看到，MARS 中的异常入口正是在 0x00004180 处：

因此，若你的中断处理程序为 exception.s， 样例如下： （注意：该程序中只包含一条 `eret` 指令，直接返回了正常状态。）

```
.ktext 0x00004180
eret
```

思考题

1、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）定时器倒计数至 0 产生中断；

（3）handler 设置使能 Enable 为 1 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。）

2、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?