【详解】中断相关的知识

版本：v1.1

Crifan Li

摘要

此文详细介绍了中断，陷阱和异常之间的区别和联系。以及中断的各种分类，包括向量中断和非向量中断，内部中断和外部中断，软件中断和硬件中断可屏蔽中断和非可屏蔽中断，并且对向量中断和非向量中断进行了实例解析

本文提供多种格式供：

在线阅读	HTML [1]	HTMLs [2]	PDF [3]	CHM [4]	TXT [5]	RTF [6]	WEBHELP [7]
下载（7zip压缩包）	HTML [8]	HTMLs [9]	PDF [10]	CHM [11]	TXT [12]	RTF [13]	WEBHELP [14]

HTML版本的在线地址为：

http://www.crifan.com/files/doc/docbook/hardware_basic/release/html/hardware_basic.html

有任何意见，建议，提交bug等，都欢迎去讨论组发帖讨论：

http://www.crifan.com/bbs/categories/hardware_basic/

修订历史		
修订 1.0	2011-11-01	crl
1. 详细解释了中断，陷阱和异常之间的区别和联系 2. 详细解释了中断的各种分类		
修订 1.1	2012-08-14	crl
1. 通过Docbook发布		

[1] http://www.crifan.com/files/doc/docbook/hardware_basic/release/html/hardware_basic.html
[2] http://www.crifan.com/files/doc/docbook/hardware_basic/release/htmls/index.html
[3] http://www.crifan.com/files/doc/docbook/hardware_basic/release/pdf/hardware_basic.pdf
[4] http://www.crifan.com/files/doc/docbook/hardware_basic/release/chm/hardware_basic.chm
[5] http://www.crifan.com/files/doc/docbook/hardware_basic/release/txt/hardware_basic.txt
[6] http://www.crifan.com/files/doc/docbook/hardware_basic/release/rtf/hardware_basic.rtf
[7] http://www.crifan.com/files/doc/docbook/hardware_basic/release/webhelp/index.html
[8] http://www.crifan.com/files/doc/docbook/hardware_basic/release/html/hardware_basic.html.7z
[9] http://www.crifan.com/files/doc/docbook/hardware_basic/release/htmls/index.html.7z
[10] http://www.crifan.com/files/doc/docbook/hardware_basic/release/pdf/hardware_basic.pdf.7z
[11] http://www.crifan.com/files/doc/docbook/hardware_basic/release/chm/hardware_basic.chm.7z
[12] http://www.crifan.com/files/doc/docbook/hardware_basic/release/txt/hardware_basic.txt.7z
[13] http://www.crifan.com/files/doc/docbook/hardware_basic/release/rtf/hardware_basic.rtf.7z
[14] http://www.crifan.com/files/doc/docbook/hardware_basic/release/webhelp/hardware_basic.webhelp.7z

【详解】中断相关的知识:

Crifan Li

版本：v1.1

出版日期 2012-08-14

[15] http://www.crifan.com/files/doc/docbook/soft_dev_basic/release/html/soft_dev_basic.html#cc_by_nc

目录

插图清单

表格清单

缩略词

NMI (NMI)	Non-Maskable Interrupt 非可屏蔽中断

正文之前

1. 此文目的

解释中断的基本概念，和其与异常和陷阱之间的区别和联系。

然后再详细解释，从不同方面来看，中断所细分出来的各种分类。

2. 声明

由于笔者水平有限，文中难免有误，欢迎批评指正:admin (at) crifan.com。

第 1 章 什么是中断

1.1. 中断出现之前

在中断出现之前，程序的执行，都是CPU一步步按照指令顺序执行下去的，中间即使有程序的跳转，但是CPU的执行的顺序，还是有程序中的代码的逻辑决定的。

首先要了解的背景知识是，CPU指令的执行速度和外设IO之间，前者很多都是纳秒（nm）级别的，后者很多都是毫秒（ms）级别的，所以时间量级上相差很大。

有了这个背景知识后，再去了解程序实际执行过程中，有时候会涉及到CPU要去和各种外设打交道。

比如，CPU发送完毕一个指令后，外设IO开始执行对应的动作，然后CPU就开始去读取对应的外设的状态，由于CPU指令执行很快，然后结果就是CPU执行了N次指令了，结果查询到的结果是，外设还没有执行完对应的动作。

最后的最后，好不容易等到外设执行外对应的动作了，然后才接着执行后面的指令。

此循环地，不断地去等待外设完成对应的动作，叫做轮询（Poll）。

可以发现，CPU在轮询过程中，就是在做无用功，而为了提高CPU的效率，不要让其浪费生命在无谓的轮询上，所以才会引出中断这个概念。

1.2. 为何要有中断

所谓的中断的方式，就是当CPU发送完指令，让外设开始去执行其对应的动作之后，不再像之前一样傻傻等待，不断地轮询，而是发送完指令后，让外设去干其该干的活，然后自己就返回了，继续执行其他的指令了，然后等过了会，外设干完活了，硬件上，会自动发送一个中断给CPU，通知其说，我活已经干完了，你可以过来处理后续该要处理的事情了，比如将对应的数据拷贝到对应的其他内容中等等之类的事情。

这套机制，就是对应的中断的概念，简要描述就是：

CPU对外设等设备做好对应的准备工作后，然后就继续执行其他CPU指令了，然后外设做完自己的事情之后，硬件上，会自动发送对应的中断给CPU的，而CPU发现有中断了，然后才会跳转到对应的中断服务程序，即常说的ISR（Interrupt Service Routine），做接下来的事情，比如将数据拷贝到对应的内容中等等。

关于中断的概念方面，此处不再赘述，因为这里有更加清晰易懂的解释：

【转】ARM9 2410移植之ARM中断原理, 中断嵌套的误区，中断号的怎么来的 [1]

[1] http://www.crifan.com/switch_arm9_2410_transplant_arm_interrupt_principle_the_error_interrupt_nesting_how_come_the_interrupt_number/

第 2 章 中断和异常，陷阱的区别和联系

说到中断，那么多很多人会想到其他几个相关的词，即异常和陷阱。

这几个概念，很多人都会搞混淆，所以，在此特别地，详细地解释一下。

首先，各自名字所对应的英文单词叫法分别是，中断是interrupt，异常是exception，陷阱是trap。

对于中断，异常和陷阱这几个词来说，其实还有其他一些相关的词：故障/错误fault，终止abort。对于这些名词，其实没有一个完全统一和精确的解释。

因此也出现，不同的厂家，不同的作者，有不同的各自的解释。

不过，相对来说，这么多名词中，最常用的，还是上面这三个：中断，异常和陷阱。

此处，对于其各自的含义，也采用大家所最常用的解释。

只是，如果你在别处看到对这些名词有不同的解释，也别太惊讶就行了。

2.1. 中断Interrupt

此处的中断，一般也成为硬件中断，即外部的硬件发生了某些事件了，然后就通知到了当前CPU，CPU跳转到对应的ISR去执行了。

此处的外部的硬件中断，和当前执行的指令，是没有任何关系的。

而且往往都是外部设备，需要引起CPU的注意，比如某个按键被按下了，定时器超时了等等，才会去打断CPU，CPU才会跳转到ISR执行，然后执行完对应的ISR，再回来继续执行下一条指令。

可见，CPU此处是被动的，也不知道什么时候，属于不可预知的，就会受到外部的硬件中断，而外部的硬件中断，和当前CPU所执行的指令，没有任何关系，即属于异步关系。

2.2. 异常Exception

异常，可以称为自动产生的陷阱，不是我们所期望产生的，而是被动的强制产生的。

我们执行的指令遇到了一些异常情况，比如（除法中出现了）除零，（指令执行时候遇到了）非法指令，内存非法访问（比如向只读的内存中写入数据）等，就会触发异常，CPU发现你程序在做非法的事情了，发现你在干坏事，那么当前要出来处理这些紧急事务了，即跳转到对应的异常处理程序中去，接下来要做的事情，可以有多种做法和选择，比如尝试修正此错误，或者简单地中止程序的运行，或者是打印出相关的错误信息（比如Windows遇到非法地址访问，会跳出对应的对话框提示你，某某地址，常见的是0地址，不能访问）。

可见，异常，对于CPU来说，也是被动的，不知道何时会发生的，但是和当前指令是有关系的，即当前执行指令出现错误的时候，才会出现异常，即是同步的。

2.3. 陷阱Trap

陷阱，也常被叫做软件中断（software interrupt）。80x86系统中，有对应的INT指令，执行对应的INT指令，就会跳转到对应的函数中去执行对应的ISR了。

此处的陷阱，可以看出，是无条件执行的，而且是有意为之，是软件写的代码，故意的，让其产生对应的陷阱的。即只要执行了INT指令后，就会跳转到对应的ISR中去，而不像中断和异常那样需要满足相关的条件。

陷阱主要是用于调试的，比如用来通知调试器，某条执行被执行了，然后触发调试器做出对应的动作。

对于CPU来说，陷阱是由于执行了INT等指令而产生的，所以是（和当前指令的执行是）同步的（synchronous），对于CPU也是预知的，知道是什么时候会发生陷阱的，属于主动的，而不是被动的。

2.4. 中断和异常，陷阱的区别和联系

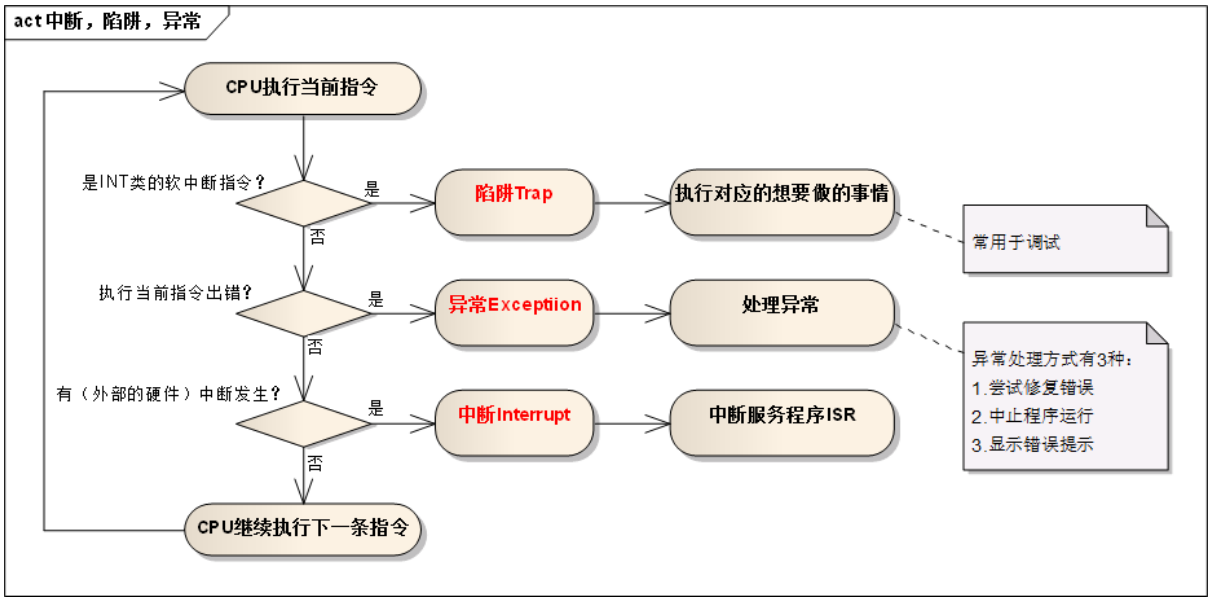
对于中断，异常和陷阱的各种关系，总结如下：

表 2.1. 中断，异常和陷阱的区别和联系

	对CPU来说是	和当前CPU所执行的指令的关系		CPU接下来的事情	程序员和用户的态度
中断	被动的	异步的	没关系	跳转到对应的ISR	希望有对应的中断，以使得CPU可以响应对应的中断，执行对应的ISR
异常	被动的	同步的	有关系，因为就是当前指令执行出问题，才有的异常	跳转到对应的异常处理	不希望出现异常，如果出现了，那往往是指令执行出现某些错误了
陷阱	主动的	同步的	有关系，就是执行当前软中断指令，才进入的软中断	执行对应的软中断处理函数	对于想要实现调试功能的程序员，有需要此陷阱的必要，其他人不用关心此点

下面，用图表总结了，中断，陷阱和异常的区别：

图 2.1. 中断，陷阱和异常的区别



注意

无论是中断，还是异常和陷阱，对应的处理函数，一般都可以称其为中断服务程序ISR，都只是一个函数

具体函数里面要做什么事情，是由你写程序的人决定的

比如中断中处理对应的事情，异常中自己决定如何响应出现的异常，陷阱中决定做什么事情。

第 3 章 中断的分类

中断，从不同的角度来区分的话，可以有不同的分类。

下面就来从不同的方面，不同的角度，不同的侧重点，来对常见的分类，做个详细的解释。

3.1. 内部中断和外部中断

根据中断来源的方向不同，可分为内部中断和外部中断。

首先我们要搞懂，此处所说的内部中断和外部中断中的“内部”和“外部”，都是相对于CPU，或者说相对于CPU所执行的指令，来说的。

另外，广义上说，中断，就是中止，打断，即中止打断正在执行指令的CPU（而让其去执行别的ISR程序）的，都可以叫做中断。

所以，不论是狭义上的普通所说的中断，还是陷阱和异常，对于CPU执行正常的指令这个过程来说，都是会打断正常执行指令的顺序，所以都可以叫做中断。

所以，就很好理解这里的外部和内部的含义了：

针对于CPU执行指令来说，从外界过来干扰自己的，那种普通外设所产生的硬件的中断，就是外部中断；

而CPU本身执行的指令产生错误了或者本身就是执行的是INT等中断指令，那么就会产生异常或陷阱，就属于内部中断。

即对于广义上的中断的概念来说：

普通的硬件中断，属于外部中断；

陷阱和异常，属于内部中断。

3.2. 软件中断和硬件中断

根据中断的来源的性质不同，中断可分为软件中断和硬件中断。

根据前面对于中断，异常和陷阱的详细解释，所以此处也很好理解所谓的软件中断和硬件中断了。

即，从中断的来源来说，来自外部的外设所产生的中断，叫做硬件中断；

而从CPU内部所执行的（类似于80x86中的INT）指令所产生的中断，成为软件中断，是软件写的中断指令，即软件代码，所产生的中断，此处就是指陷阱。

所以，可以简单的理解为：

软件中断就是陷阱；

硬件中断就是普通的外设的中断；

3.3. 向量中断和非向量中断

中断根据如何系统如何处理和响应的方式，可以分为向量中断和非向量中断。

向量者，矢量也，即指方向，门路。

简单的说就是：

向量中断由硬件提供中断服务程序入口地址；

非向量中断由软件提供中断服务程序入口地址；

向量中断，是CPU得到此中断后，直接跳转到，对应的，固定的某个地址，该地址原先已经存放好了该中断所对应的中断服务程序ISR的地址了，即某个ISR函数的地址，所以CPU直接跳转到该位置，也就是直接跳转到对应的ISR，执行对应程序了。

对于非向量中断，是对应着多个函数，共享这个非向量中断的总入口地址，即可以理解为，多个非向量中断，其入口地址都是同一个地址，即普通的IRQ中断的那个地址，然后此地址中存放了通用的ISR函数的地址，该ISR函数中，会去经过一系列的判断，找出是具体真正的哪个外设的中断，然后再在对应的之前设置好的某个中断服务程序表中，找到对应的该中断所对应中断函数，去执行对应的真正的ISR函数。

3.3.1. 向量中断和非向量中断实例解析

以ARM系统为例，物理地址最开始的几个地址，存放的是对应的几个最常见的向量中断，如数据中止异常，软中断，预取指错误异常等，还有一个是其他非向量中断的总入口IRQ。

此部分内容，可以参考Uboot中的ARM的初始化部分start.S中的代码来解释：

其相关代码如下：

```
_start: b       reset
 ldr pc, _undefined_instruction
 ldr pc, _software_interrupt
 ldr pc, _prefetch_abort
 ldr pc, _data_abort
 ldr pc, _not_used
 ldr pc, _irq
 ldr pc, _fiq

_undefined_instruction: .word undefined_instruction
_software_interrupt: .word software_interrupt
_prefetch_abort: .word prefetch_abort
_data_abort:  .word data_abort
_not_used:  .word not_used
_irq:   .word irq
_fiq:   .word fiq
。。。
/*
 * exception handlers
 */
 .align  5
undefined_instruction:
 get_bad_stack
 bad_save_user_regs
 bl  do_undefined_instruction

 .align 5
software_interrupt:
 get_bad_stack
 bad_save_user_regs
 bl  do_software_interrupt

 .align 5
prefetch_abort:
```

```
 get_bad_stack
 bad_save_user_regs
 bl  do_prefetch_abort

 .align 5
data_abort:
 get_bad_stack
 bad_save_user_regs
 bl  do_data_abort

 .align 5
not_used:
 get_bad_stack
 bad_save_user_regs
 bl  do_not_used
。。。

 .align 5
irq:
 sub lr, lr, #4           @ the return address
 ldr sp, IRQ_STACK_START         @ the stack for irq
 stmdb sp!,  { r0-r12,lr } @ save registers

 ldr lr, =int_return          @ set the return addr
 ldr pc, =IRQ_Handle          @ call the isr
int_return:
 ldmia sp!,  { r0-r12,pc }^ @ return from interrupt

 .align 5
fiq:
 get_fiq_stack
 /* someone ought to write a more effiction fiq_save_user_regs */
 irq_save_user_regs
 bl  do_fiq
 irq_restore_user_regs
```

可看出，物理内存最开始的存放的内容是：

地址0x0: reset整个系统

地址0x04:放了一个指令，该指令是将_undefined_instruction存入PC，即实现PC跳转到_undefined_instruction的地址中去；

地址0x08:同理，PC跳转到_software_interrupt

地址0x0C:同理，PC跳转到_prefetch_abort

地址0x10:同理，PC跳转到_data_abort

地址0x14:同理，PC跳转到_not_used

地址0x18:同理，PC跳转到_irq

地址0x1C:同理，PC跳转到_fiq

其中，对于_undefined_instruction，很明显，就是我们之前所解释的异常，即指令执行出了对应的问题了，PC会直接跳转到此处的0x04的地址，然后该地址中，就是把PC跳转到对应的_undefined_instructio的位置，去执行对应的异常处理。

其他的_software_interrupt和_data_abort等，都是同样道理，不多解释。

而上述这些异常或_software_interrupt，就都是所谓的中断向量，都是由硬件架构决定的，固定好的了地址，作为软件开发人员，只要把对应的指令写好，到时候发生对应的异常，系统自动会跳转到此处的地址，实现对应的PC的跳转，去做对应的处理。

而对于0x18处的_irq，就是我们所说的所有的非向量中断的总的入口地址，即系统发现有中断了，此处发现是普通的中断，那么就会跳转到0x18的地址这里，然后执行的是：

PC跳转到_irq，而_irq地址所对应的内容是：保存对应的当前的环境，即上下文，然后执行“ldr pc, =IRQ_Handle”，即跳转到IRQ_Handle函数中去。

而以TQ2440的S3C2410为例，其代码为：

interrupts.c (opt\embedsky\u-boot-1.1.6\cpu\arm920t\s3c24x0)

```
void Isr_Init(void)
{
    int i = 0;
    intregs = S3C24X0_GetBase_INTERRUPT();

    for (i = 0; i < sizeof(isr_handle_array) / sizeof(isr_handle_array[0]); i++ )
    {
        isr_handle_array[i] = Dummy_isr;
    }

    intregs->INTMOD=0x0;                // All=IRQ mode
    intregs->INTMSK=BIT_ALLMSK;         // All interrupt is masked.

    //pISR_URXD0=(unsigned)Uart0_RxInt;
    //rINTMSK=~(BIT_URXD0);             //enable UART0 RX Default value=0xffffffff

    isr_handle_array[ISR_TIMER4_OFT] = IsrTimer4;
    isr_handle_array[ISR_WDT_OFT]  = IsrWatchdog;

#ifdef CONFIG_USB_DEVICE
    isr_handle_array[ISR_USBD_OFT] = IsrUsbd;
    isr_handle_array[ISR_DMA2_OFT] = IsrDma2;
    ClearPending(BIT_DMA2);
    ClearPending(BIT_USBD);
#endif
}

void IRQ_Handle(void)
{
    unsigned long oft = intregs->INTOFFSET;
    S3C24X0_GPIO * const gpio = S3C24X0_GetBase_GPIO();

//  printk("IRQ_Handle: %d\n", oft);

    if( oft == 4 ) gpio->EINTPEND = 1<<7;
    intregs->SRCPND = 1<<oft;
    intregs->INTPND = intregs->INTPND;

    /* run the isr */
    isr_handle_array[oft]();
}
```

可见，其中IRQ_Handle做的事情，就是去读取对应的寄存器，然后经过计算，找到真正的中断源的偏移量，然后再通过偏移量，在中断函数表中，去获得对应该中断的中断服务程序ISR。

而其中的中断函数表isr_handle_array是在程序最开始初始化时候去调用Isr_Init来初始化好的，已经见每个中断多对应的ISR函数存放了对应的位置了。

向量中断的优点是，反应速度快，有了中断，CPU直接跳转到对应的位置，去执行对应的代码了，属于速度快，但是无法扩展，由硬件设计时候觉得的，固定好了，没法改变。

而非向量中断，由于多了一层调用关系，而且在总的普通中断的入口函数中，要去读取寄存器，再去计算到底是哪个中断，所以，速度上，就相对较慢了，属于速度慢，但是扩展性较好。

简单的说就是：

硬件中断，由硬件提供ISR地址，速度较快；

软件中断，由软件计算出中断源，再去找出对应的ISR，速度相对慢。

3.4. 可屏蔽中断和非可屏蔽（NMI）中断

而普通的中断，即前面所说的外部的硬件的中断，根据其中断的性质，是否可以被屏蔽掉，而分为可屏蔽中断和非可屏蔽中断。

对于如果是可屏蔽的中断，是通过设置硬件上对应的标志位flag来实现屏蔽对应的中断的。

一般的中断服务程序中，很多时候，你都会看到，在进入ISR的时候，首先做的事情就是去关闭中断，此时，指的就是设置对应的中断屏蔽寄存器的标志位，去关闭后续的可能发生的中断。

常见的系统中，都有对应的mask 寄存器的，针对每一个中断，设置其是否被屏蔽。

一般来说，向量中断，都是不可屏蔽的，而非向量中断，多数都是可以屏蔽的。

参考书目

[1] Interrupts, Traps, and Exceptions Chapter 17[1]

[2] 浅评中断、陷阱、异常 (转)[2]

[3] 【转】关于向量中断和非向量中断[3]

[4] 向量中断和非向量中断的区别[4]

[5] 【转】ARM9 2410移植之ARM中断原理, 中断嵌套的误区，中断号的怎么来的[5]

[6] Trap (computing)[6]

[7] Difference b/w software interrupt/exception/trap[7]

[8] What is key difference between a trap and interrupt?[8]

[9] 异常及非屏蔽中断[9]

[1] http://webster.cs.ucr.edu/AoA/DOS/pdf/ch17.pdf
[2] http://blog.csdn.net/littlehedgehog/article/details/2740406
[3] http://hi.baidu.com/houxn22/blog/item/9bd4df1f55a8c21f304e15dc.html
[4] http://www.laogu.com/laogubbs/see_93533.htm
[5] http://www.crifan.com/switch_arm9_2410_transplant_arm_interrupt_principle_the_error_interrupt_nesting_how_come_the_interrupt_number/
[6] http://en.wikipedia.org/wiki/Trap_(computing)
[7] http://kerneltrap.org/node/6314
[8] http://wiki.answers.com/Q/What_is_key_difference_between_a_trap_and_interrupt
[9] http://oss.org.cn/kernel-book/ch03/3.1.3.htm