第10章 汇编语言“ trap ”例程

本章的主要目的是检验“ m40.s ”文件中的汇编语言代码，这些代码涉及中断和陷入处理。

该代码在0750行和0805行之间，有两个入口点：“ trap ”(0755)和“ call ”(0766)。有若干条不同和相关路径通过此段代码，我们跟踪其中的某些实例。

10.1 陷入和中断源

在第一部分的讨论中有3处可期望产生中断或陷入：

1)“ main ”反复调用“ fuibyte ”(1564)直至返回1个负值。在产生“ 总线超时错 ” (bus timeout error)，导致陷入到矢量单元4(0512行)之后将返回此值。

2) 已将时钟设置为运行，此后每隔16.7或20ms(一个时钟滴答)就将产生一次中断。

3) ＃1进程作为“ exec ”系统调用的一部分将执行一条“ trap ”指令。

10.2 fuibyte(0814)与fuiword(0844)

“ main ”使用“ fuibyte ”和“ fuiword ”。因为前者在非实质方面更复杂一些，所以我们将其留给读者仔细阅读，而集中说明后者。

当系统在核心态下运行时，以一个参数调用 fuiword(1602行)，该参数是一个用户态地址空间中的地址。“ fuiword ”的功能是取相应字的值，并将其作为执行结果在 r0中返回。但是，如果发生错误，则返回－1。

注意，与“ fuibyte ”相比，“ fuiword ”有一个含糊不清之处，返回值－ 1并不一定表示出错，而可能是用户空间中相应字的实际值。但请你务必相信，按照“ main ”的实际使用方法，这不会造成任何麻烦。

另外，这段代码对“ 总线超时错 ”和“ 段违例错 ”两者并不作区分。

该例程按如下方式执行：

0846：将调用参数移入r1。

0848：调用“ gword ”。

0852：将当前PS存放到栈上。

0853：将处理机优先级升为7(禁止所有中断)。

0854：“ nofault ”单元(1466)的内容存放到栈上。

0855：将“ err ”例程的地址装入“ nofault ”单元。

0856：用“ mfpi ”指令从用户地址空间 r1所指向的单元中取字。如果没有出错，那么该值将存放在核心栈上。

0857：将此值从核心栈转移至r0。

0876：恢复“ nofault ”和PS的以前值。

0878：通过0849行返回。

现在假定“ mfpi ”指令出了错，或者出现了“ 总线超时 ”。

0856：“ mfpi ”指令将异常终止。 PC将指向一条指令(0857)，然后，将通过矢量单元 4产生陷入。

0512：新PC的值将是“ trap ”。新PS将指明：

现状态=核心态

前状态=核心态

优先级=7

0756：执行的下一条指令是“ trap ”的第一条指令。它将处理机状态字保存到当前栈顶以上的第2个字。(这一点不是我们现在所关心的)。

0757：“ nofault ”包含“ err ”的地址，其值非0。

0765：将1送入SR0，重新启动存储管理单元。

0766：“ nofault ”的内容送至栈顶，覆盖以前的内容，它是“ gword ”中的返回地址。

0767：“ rtt ”不是返回到“ gword ”，而是返回到“ err ”的第1个字。

0880：“ err ”恢复“ nofault ”和PS，跳过到“ fuiword ”的返回，将－1送入r0，直接返回至调用例程。

10.3 中断

假定时钟已中断了处理机。

时钟矢量单元100和104具有相同的信息。 PC设置为标号为“ kwlp ”(0568)的单元地址，PS设置为包含下列内容：

现状态=核心态

前状态=核心态或用户态

优先级=6

注意：PS将包含真实的前状态，这与矢量单元中的值无关。

0570：矢量单元包含了新 PC值，它是标号为“ kwlp ”语句的地址。这条指令是对子程序“ call ”的调用，其参数径r0传送。

在此指令执行后， r0包含了此指令后第一代码字的地址，而该字的内容是“ clock ”，亦即r0包含了“ clock ”例程的地址的地址，“ clock ”例程在文件“ clock.c ”(3725)中。

10.4 call(0776)

0777：将PS复制到栈上。

0779：将r1复制到栈上。

0780：将前地址空间的栈指针复制到栈上。 (仅当前状态为用户态时，这才是有意义的。)

这是“ mfpi ”指令的一种特殊情况。请参见《PDP11 Processor Handbook》。

0781：将PS的最后5位值复制到栈上。其值表示中断 (或陷入)的原因。应迅速取到PS的原

值。

0783：测试前状态是核心态还是用户态。

若前状态是核心态，则进行转移(0784)。更改PS，其中前状态表示为用户态(0798)。

0799：进入由r0指向的专用中断处理例程。(在此例中，这是clock，下一章将对此进行详细讨论。)

0800：当“ clock ”例程(或某个其他中断处理程序)返回时，删除栈顶上的两个字。它们是PS经屏蔽处理后的副本和栈指针的副本。

0802：从栈中恢复r1。

0803：从栈中删除PS副本。

0804：从栈中恢复r0的值。

0805：最后，“ rtt ”指令返回至被中断的“ 核心 ”态例程。若前状态是用户态，那么是否立即恢复执行被中断的例程是不确定的。

0778：在从专用中断处理例程 (在此例中是clock)返回后，检查“ runrun ”是否大于0，以判断是否有较当前进程优先级高的进程已准备好运行。若判断结果是使当前进程继续运行，则关键点是：在执行从中断返回指令 (rtt指令)之前将所有寄存器恢复至中断时的状态，也就是好像没有被中断一样。因此，在本测试检查之前，使处理机优先级升为 7级(0787行)，在恢复为用户态之前，保证不会再发生中断。 (但是在恢复为用户态后，就可能立即发生另一次中断。)

若“ runrun＞0 ”，那么至少有另一个更高优先权的进程正在等待占用处理机运行。将处理机优先级设置为0，则允许任一挂起未决的中断得到处理。然后“ swtch ”(2178)使得具较高优先权的进程执行以获得进展。当进程从“ swtch ”返回时，此程序重复对“ runrun＞0 ”的测试。

上面的讨论显然可扩展至所有中断。唯一特殊之处是专用于 clock中断的例程“ clock ”。

10.5 用户程序陷入

“ 系统调用 ”机制使用户态程序能调用操作系统以得到帮助，该机制的一个重要组成部分是使用户态程序可以执行256种“ trap ”指令中的一种。(“ 种 ”在这里指的是指令字低字节的值。)

0518：在用户态程序中执行“ trap ”指令产生陷入，陷入矢量单元是34，它使标号“ trap ”的值装入PC(0512，0755行)。新PS表明：

现状态=核心态

前状态=用户态

优先级=7

0756：执行的下一条指令是：“ trap ”的第1条指令。它将处理机状态字保存到当前栈顶以上的第2个字。

由于PS字中包含所发生陷入的类型信息，所以在能改变它之前尽可能快将它保存起来是非常重要的。这条“ move ”指令的特殊用途是使陷入处理与中断处理相兼容，这样就可进一

步使用同样的代码。

0757：“nofault”将是0，所以不产生转移。

0759：在以后将需要存储管理状态寄存器的情况下，存放该寄存器，然后重新启动存储管理部件。

0762：转移至子程序call1，同时将r0的值存入栈，而r0的新值是下次要进入的例程入口地址的地址(在本例中，是文件trap.c(2693)中的例程trap)。

0772：栈指针被调整为指向已包含PS副本的单元。

0773：处理机优先级设置为0。

0774：转移至“call”的第2条指令。

从此处开始，陷入处理的路径与中断处理相同。

10.6 核心态栈

图10-1中示出了在进入“trap”过程(C语言编写)或特定中断处理例程时的核心态栈的状态。

图 10-1

第2列和第3列给出了分别相对于标号为r0和tpc栈字位置的各栈字的位置。第1列和第2列解释了文件“reg.h”的内容。

“dev”、“sp”、“r1”、“nps”、“r0”、“pc”和“ps”是在“trap”(2693)和“clock”(3725)过程中按序使用的参数名。

注意，在进入“ trap ”(C语言版)或其他中断处理例程之前，并没有在栈上存取 r2、r3、r4和r5寄存器的值。这些存栈操作是由调用“ csv ”(1420)实现的，“ C ”编译器在每个被编译过程的开始部分自动插入“ csv ”。调用“ csv ”的形式等效于汇编指令：

```
jsr r5,csv
```

它将r5的当前值存放到栈上，然后将“ C ”过程中下一条指令的地址送入r5。

1421：将r5的值复制到r0中。

1422：将栈指针的当前值复制到r5中。

注意：此时，r5指向栈中包含r5以前值的单元，亦即它指向一个指针链的链首位置，每一次过程调用就在该链中增加一个成员，除链首指针外，其他各成员都在栈中。当从一“ C ”过程中退出时，实际上返回至“ cret ”(1430)，其中使用r5的值将栈、r2、r3和r4恢复至它们的原先状态(亦即进入该过程之前时刻的状况)。由于r5起到的这种特殊作用，所以常将其称为环境指针(environment pointer)。