电厂测控微机的资源分配矛盾及其危害

杨华舒 施延华 昆明理工大学电力工程学院(650051)

Abstract

Found on research of occasional malfunction and experiment of computer-IC at servomechanism of power plant,the antinomy and the blooie features on resources-appointing of computer are analyzed,and the efficient solutions are propounded.

Keywords:automated measurement & control, microcomputer,resources-appointing

摘 要

基于对发电厂自动测控设备偶然性故障的研究和对工控微机芯片的实验，分析了存在于设备中的若干种微机资源分配的矛盾和故障特点，提出了行之有效的解决方案。

关键词：自动监测与控制，微机，资源分配

0 引言

由于计算机各种功能芯片的寿命大大超过分立元件，因此许多技术人员在测控系统的具体研制过程中，就特别强调其适用性、经济性而忽视了它的连续可靠性，致使系统的无故障运行时间大大缩短。较为常见的是在设计过程中省略了资源分配的步骤，不恰当地重复使用芯片的各种硬件资源以求降低成本，这些资源的使用也许并不矛盾，因此设计被认为是成功的；然而一旦窜入或者采集到某些特定的信息、又或运算出某些特定的数据时，资源的使用就会出现偶发性的矛盾，从而导致故障甚至重大事故的发生。

某电厂在更新励磁设备的自动调节装置后，曾经出现过相邻机组甩负荷导致运行机组欠励、甚至 ZTL 偶然失效的故障。制造厂家改善了设备微机的低通滤波后似乎解决了问题，但又在大修期间发现大功率焊机的电弧偶尔会对附近机组的励磁产生明显的影响。对微机的软硬件分析和试验均未找到异常原因，后经专家会诊，证明是堆栈与变量缓冲区在子程序调用及中断的某种组合下出现了矛盾所致。

某新建高水头电站在检测高压钢管道的过程中，多次发生 #1 或 #3 机组转速异常波动、或者智能仪表对转速的误报，长时间未能发现原因。由于偶然机会，才发现大功率对讲机使用时产生的高频电磁场偶尔会对附近的水机调速设备产生干扰。在抑制电磁噪声的努力失败以后，经过近半年的软件修改调试，才终于找到了问题的关键——通用寄存器在将运算结果转移到数据存储区之前，就被噪声偶然调用的中断服务程序修改了，亦即出现了寄存器使用矛盾。

在电厂或者电力系统进行自动化改造的过程中，发生上述类型的故障并不是个别的，施工、调试、或者大修现场的技术人员往往都能讲述一些莫名其妙的偶然性故障。资源分配矛盾主要影响二次回路中的自动测量和自动控制系统，在电厂等超强度电磁噪声环境中，很难将各种频率和幅度的干扰信号完全屏蔽于微机以外，因此必须提高设备的容噪性能以保障其连续工作的可靠性。资源分配不当，是降低微机容噪性能的重要原因，其特点是窜入的噪声往往产生变异的、偶然的故障现象，并且不易从硬件或者软件上找出原因。这些故障的发生次数虽然不多，然而却可能导致重大安全事故，笔者在应邀参加的事故鉴定中，就曾经遇到过智能机械因微机资源分配不当而引起机毁人亡的教训。为此，我们研究了微机芯片的内部结构和编程的一般规律，现将研制微机监控设备的过程中容易出现矛盾的资源和解决的方法列举如下，供有关人员参考。

1 程序与数据合用存储器

常见个人微机(PC)的运行环境较好、并且偶然性或者临时性故障带来的危害不大，故主要考虑成本因素，通常采用同一组 DRAM 分段来分别存放程序或者数据，如图 1 所示。尽管 PC 微机的用户大多遇到过与存储器密切相关的“死机”现象，然而由于没有造成太大的破坏，大家也就认可了这种设计。出于成本的考虑，现在工控计算机系统的程序存储器与数据存储器也趋向于合用同一个器件；并且非易失性 SRAM 器件的推广使这种方案获得了更多的赞同。干扰信号的多样化组合使其不可能在室内实施充分的模拟重现，因此这类产品一般被检验为合格，只有在现场运行中遇到某些特定的干扰、致使噪声恰好窜入存储器的程序段时，其隐患才会显露出来。

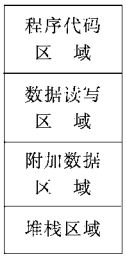

图 1 PC 机的 DRAM 分配

程序与数据在存储器内虽然均以二进制的形式存在,但程序与数据在本质上绝非同一种代码:在正常运行过程中,程序不允许修改、具有“只读”的特征。此前进行的电磁干扰研究表明[1],各类存储器件中 ROM 的抗扰能力最强,其次依序为 PROM、EPROM、EEPROM、SRAM,抑噪性能最差的是动态随机读写存储器 DRAM。据此可以认为,SRAM 和 DRAM 只能用于存放数据,因为数据被噪声修改后有许多成熟的方法来纠错或者补偿;而程序一旦被修改,则可能带来预料之外的故障和危害,出现计算机失控的现象。此外,处于同一个芯片的程序和数据具有相同的选片信号,在环境干扰较强时就有可能把数据区域内的代码当作指令取用,而且无法落入软件陷阱自动复原,从而增加了计算机出错和死机的机会。

为了提高测控系统或产品的可靠性,应当把程序以及具有“只读”性质的其它代码(如实验表、转码表等)单独存放在不易被噪声修改的器件中,并且工控计算机的程序存储器应当与数据存储器在器件上分离。在研制阶段,为了软件的修改方便可采用 EPROM 存放程序;在中量或者小量生产中,建议采用 PROM;在规模化生产中建议尽量采用 ROM 以提高性价比。倘若要经常与磁盘交换程序代码,亦可采用专设的 SRAM,但必须对这个器件考虑周密的抗干扰措施,并且设置大大高于其“读取”电压范围的“写入电压”等。

2 数据存储器分配的矛盾

工控微机的数据存储器主要可分为堆栈、工作变量暂存区、结果数据存储区、寄存器保护区域,在系统设计的资源分配阶段应当列表逐项说明数据存储器每个单元的用途。一些设计人员为图省事,略去了“资源分配”步骤,并且把数据存储器仅分为堆栈、变量两大块使用。而通常情况下,这四部分数据区域都是必要的,并且应当相互独立。各部分资源分配太多会造成浪费而加大成本,分配不足或者相互重叠都将使得有用的数据丢失,从而导致偶然性故障。

2.1 堆栈区域

堆栈主要用来保护断点地址(PC 值)和有关寄存器、工作变量的内容,有时亦用于各个过程之间的参数传递。当其分配的空间不足时,对于递增型堆栈(例如 MCS-51 系列)很可能在存储器高端与其它区域冲突,导致栈顶附近若干字节的内容与另一段区域某些地址的内容相互覆盖而引起错误;对于递减型堆栈(例如 INTELX86 系列),则可能在栈顶字节(存储器低端)重复压栈而冲掉此前栈内的有用信息。这种错误主要发生于堆栈用量较大的程序段,尤其会出现在多级嵌套中断请求的时候。由于各个中断请求的时间不能预先确定、它们嵌套的场合与方式亦难以预先估计,使得堆栈的预留容量很难精确计算。较为保守的办法是以“最大嵌套数”作为堆栈容量 K_1 的估算依据:

$$K_1=n(D+Z)+\sum G \quad (1)$$

式中:n —— 程序指针 PC 的大小(bytesize);

D —— 允许发生的中断总数,或者中断服务程序的总数;

Z —— 可能发生的主动调用子程序的最大嵌套层数;

$\sum G$ —— 各中断服务程序和被嵌套调用的各子程序中,需要压栈保护的寄存器以及工作变量的字节数量之总和(主要是累加器和各段程序公用的数据临时存储器单元)。

2.2 重要寄存器的保护区域

CPU 及外围芯片复位时,其中的各种控制寄存器、状态及标志寄存器、I/O 端口寄存器等都会恢复到默认值。因此当 PC 或者 SP 被噪声改写、致使计算机被 WDT 强制复位时,将打乱正常运行状态的持续。极端的状况是,本来受控静止的电动机可能会自动运转,从而引起事故。有效的解决方案为:①转变“0 关 1 开”的固有观念,而把控制“开”的电平信号设置为与 I/O 端口寄存器的默认状态相反;②在不受 CPU 复位影响的数据存储器上开辟专门的保护区域,以便在程序设计过程中主动备份相关寄存器被改变后的内容;③当计算机被 WDT 强制复位并正常运行后,首先执行数据恢复程序,把控制字、端口、状态等重要寄存器被保护的内容恢复还原。显然,在 RAM 的资源分配时如果对这个保护区域的独立性、容量的要求或者对 CPU 复位的影响考虑不周,都可能由某些特定的干扰导致运行混乱。保护区域的大小 K_2 可由下列算式计算:

$$K_2=\sum J+\sum S \quad (2)$$

式中:$\sum J$ —— CPU 芯片上的堆栈指针、通用寄存器、状态及标志寄存器的字节总数;

$\sum S$ —— CPU 以外需要保护的重要寄存器的字节总数,主要包括各输出端口的状态、以及各扩展芯片的控制信息等。

2.3 工作变量暂存区域

为了节省资源,通常在数据存储器中开辟少量的

(4~32 Bytes)单元作为工作变量或者工作寄存器,如通用寄存器一样提供给各个子程序以及程序的各种功能模块,作为公用的临时存储器,典型的如 MCS-51 系列芯片内定义的工作寄存器 R_n。工作变量的特点是在主程序、各个子程序、以及中断服务程序中都可能频繁地、重复地使用,这一特性决定了其中的数据在转移到相对单一用途的"结果存储区域"之前很容易被嵌套的内层子程序修改,中断请求被响应后尤其如此。倘若各子程序单独调试正确而全程运行出错,一般可考虑是对工作变量的内容保护不力所致。可行的解决措施包括:①所有子程序(尤其是中断服务程序)的开始部分,应当首先把本程序段中所要用到的通用寄存器和工作变量中的数据压入堆栈保护;②在退出本子程序(返回)之前,再依据"先进后出"的原则将这些数据弹出堆栈、恢复被子程序使用过的工作变量和通用寄存器的原值;③在选择区域时注意避开堆栈的使用范围、避免数据相互覆盖。

2.4 结果数据存储区域

上述各部分之外的 RAM 单元一般可作为存放采集数据、运算结果、外设控制或者显示信息的相对固定的容器,即将其用作"结果变量"单元使用,可以直接或间接寻址。对于这些单元的要求,主要是数量充足、性能稳定,尽量采用静态存储器 SRAM。在资源分配过程中,应当避开前述各区域,并且列表注明 RAM 中每个字节(或字、位)的用途,这样既便于编程时查询、又能避免重复使用,还可由此计算出数据存储器的合理大小。

3 程序存储器使用的矛盾

在资源分配阶段,"程序"可以定义为"正常运行情况下不允许变化的所有代码",因此程序存储器通常分为存放功能性指令、只读数据(表)区域、以及中断矢量、软件陷阱这四个部分,其中"中断矢量"由 CPU 的生产厂家设定,其地址是不可更改的。如果其它区域覆盖了允许响应的中断矢量地址,则中断服务程序就无法得到执行,还可能执行到某几条指令的部分代码组合而出现意外的结果。但在实际应用中,多数情况是把指令区域和数表区域混在一起使用,以便最大限度地利用存储器资源、降低产品成本。这样编程时就应当使用伪指令(如 ORG)来规定指令或者表的位置,以绕开用到的中断在存储器内的矢量地址,并且指定对应的中断服务程序的首址。此外,数表区域应当自成整体并且尽可能置于程序存储器地址的高端,这样可以降低 PC 被噪声修改后误把数表当作指令的可能性。

软件陷阱则是为了恢复程序指针的正常运行而专门布置的一段额外的程序,它可以明显改善计算机系统的容噪性能,大大降低指令错误组合和死机的概率。PC 的内容被噪声窜改后,其值即使不再是某条指令的首址,但只要还在存储器的有效地址范围内,就可以采用这个方案使其自动恢复正常。依据系统所用 CUP 的指令代码的最大允许长度 L(bytes),在各功能程序模块编程结束后,都插入 L-1 个没有实际功能的单字节指令(例如"空操作"指令),再后缀无条件转移指令,PC 内代码的错误组合在此陷阱内即可得到纠正。同理,在数表结束或者程序存储器编程后剩余的单元内,也应当填入上述的单字节指令,并在最后几个单元中填入 1 条无条件转移指令。

例如,MCS-51 系列微机的指令代码长度不超过 4 字节,于是在各子程序之后都可以插入 3 条 NOP 和 1 条 LJMP 指令:

```
ZCX:  <某子程序首行>
      …… ……
      …… ……
      RET
      NOP
      NOP
      NOP
      LJMP  ZCX
```

假定 PC 被噪声窜改后指向某子程序域内的地址、但并不是某条指令的首址,因此本子程序的各条指令都未得到正确执行。然而无论 PC 内的代码如何错误组合,CPU 终究会取到一条完整的 NOP 或者 LJMP 指令,从而转回标号 ZCX,重新正确地执行本段子程序并借以恢复正常。可以看出,PC 正常时陷阱程序不会对系统产生任何影响。

4 结束语

存储器和其它微机硬件资源的分配是否得当,直接关系到工控微机运行的可靠性。电厂环境内的干扰种类庞杂,各种设备中的微机测控系统出现的难以定位的莫名故障或者误报、偶然性死机、甚至重大事故,都有可能起因于程序对资源的使用产生了矛盾。因此建议微机工控系统的研制人员充分重视资源分配环节,坚持数据与程序的存储硬件分设的观念,郑重推算存储器各区域的用量和地址范围,这样才能在保证控制系统或产品可靠性的前提下有效地降低其硬件成本和维护成本。

参考文献

1 杨华舒,褚福涛.单片计算机抗干扰的软件途径[J].电子技术应用,2001(3)
2 郑学坚,等.微型计算机原理及应用[M].清华大学出版社,1995
3 朱宇光,等.单片机应用新技术教程[M].电子工业出版社 2000
4 周斌,等.机电一体化实用技术手册[M].兵器工业出版社,1994
5 黄一夫.微型计算机控制技术[M]. 机械工业出版社,1990

[收稿日期:2002.2.8]