# §3 PMAC用户指南：

HOME

用户手册的这一部分解释了怎样使用PMAC板的各种特性。它是按尽快进入令人关注的领域的主题（安全性，I/O，伺服，轨迹，等等）来组织的。这些内容是按照用户建立一个系统的典型顺序来组织。

在这部分之前的《使用PMAC入门》一章和本章的组织相似，但这一章的信息和内容具有更多的深度和广度。这一章将指导一个初学者快速地按部就班地建立一个典型系统。通过比较可以发现，用户手册的参考部分是按字母顺序来编组命令，而按数字顺序来组织变量，寄存器，跳线和接线端子的。在用户手册的这两部分之间可进行交互使用。在这部分中的任何变量，指令寄存器,跳线，或接线端子，在用户手册的适当的参考部分将含盖更多的细节。

当你在通读用户手册的这一部分时，你也许会发现其中包含了你此时并不需要的主题或内容。那么跳过这些部分，直接到那些对你来说更直接，更有用的部分。

## 用户指南综述：

1. **综述：**这部分对PMAC的作用做了一个简明的解说。
2. **同PMAC对话：**这部分解释了主机和PMAC之间的通信的基础。
3. **输入/输出：将PMAC板和机器连接起来：**这一部分对把PMAC板上的所有端口和你的机器上的别的电路连接起来提供了详细的说明。
4. **设置一个电机：**这部分解释了PMAC板上的软件的设置，它可以让一个电机按你希望的方式工作。
5. **设置PMAC的换向：**这部分说明了如果PMAC板执行电机的换相算法的参数设置。
6. **伺服闭环：**这部分介绍了伺服环的正确建立（调整）方法。
7. **确使你的应用安全：**这部分介绍了重要的安全特征和设置过程。
8. **基本电机运动：**这部分介绍了如何设置并且执行单个电机的简单运动：微动，回零，和开环运动。
9. **建立一个坐标系：**这一部分介绍了为执行一个运动程序如何将电机分配给各坐标系。
10. **计算特征：**这一部分列举了PMAC的计算能力。
11. **编写一个运动程序：**这一部分介绍了在PMAC板上编写和执行一个运动程序所要求的步骤。
12. **使PMAC与外部事件同步：**这一部分说明了如何让PMAC板与机器上别的部分（也包括别的PMAC）的事件协调起来。
13. **编写一个PLC程序：**这一部分介绍了在PMAC板上编写和执行一个PLC程序所要求的步骤。
14. **编写一个主机通信程序：**这一部分介绍了在主机上编写和执行一个程序通过串行口或总线同PMAC板通信所要求的步骤。

# §3—1 PMAC综述：

*Delta Tau Data Systems*的可编程多轴控制器（PMAC）是一个拥有高性能伺服运动控制器的系列，它通过灵活的高级语言可最多控制八轴同时运动。通过一个功能强大的数字信号处理器（DSP），PMAC给多轴控制器提供了一个前所未有的价格性能比。*Motorola*的DSP56001是PMAC的CPU，它处理了所有八轴的所有计算。

PMAC共有四种硬件形式：PMAC－PC，PMAC－Lite,PMAC-VME,和PMAC-STD。这些卡在形状，总线接口的特性，和某些特定I/O端口的性能这些方面是彼此各不相同的。所有这些形式的卡都有相同的在板固件，所以为一种版本编写的PMAC程序也可以在其它任何版本上运行。PMAC－STD在某些I/O端口有不同的存储映射。

任何版本的PMAC板既可以以脱机方式运行,又可以通过串行接口或总线接口用一台主机来控制它运行。

## 适应能力:

作为一种具有广泛用途的控制器,PMAC能应用于各种各样的设备,从那些精密到小于百万分之一英寸的精密仪器到那些需要数百千瓦或马力的大型设备。它的多种用途包括机器人、机床、纸和木材的加工作业、装配线、食品加工、印刷、包装、物料装卸、摄像机控制、自动焊接、硅片加工、激光切割和许多别的方面。

## 为一个任务的配置:

PMAC可以通过硬件设置的选择(通过选项和辅助),参数的设置,和运动及PLC程序的编写,从而实现一种特殊的用途。每一PMAC固件能够控制八根轴。这八根轴可以互相联动以进行完全协调的运动，每一根轴也可以被放入它自己的坐标系中从而得到八个完全独立的运作；或者它们中间所有的其他组合形式。

PMAC的CPU与轴的通信是通过被特殊设计的门阵列ICs（作为DSPGATE而提到)来实现的。每一个ICs能够控制四个模拟输出通道，四个作为输入的编码器，和四个来自附件的模拟驱动输入。一片PMAC板可以运用这些门阵列ICs的一到四个，从而规定硬件的配置总数，以便统计输入和输出的数量和类型。最多可以有16片PMAC板完全同步的联系到一起使用，控制总共的128根轴。

## PMAC是一台计算机:

认识到PMAC就是一台完整的计算机，这一点是很重要的。它可以通过存储在它自己内部的程序进行单独的操作。此外，它还是一台实时的，多任务的计算机，能自动对任务进行优先等级判别，从而使具有高的优先等级的任务比具有低的优先等级的任务能先被执行（这是很多个人计算机上也未能实现的）。

即使是与一台主计算机连接到一起使用，它们之间的通信应被认为是一台计算机与另一台计算机之间的通信，而决不是主计算机与它的外围设备之间的通信。在许多应用中,PMAC能够同时执行多个任务并能正确地进行优先级排序的能力，使它能够在处理时间和任务切换的复杂性这两个方面大大减轻主机（和它的编程器）的负担。

## PMAC的功能:

### 执行运动程序:

PMAC的最明显的任务是按运动程序中的顺序执行一个运动程序。当被告知执行一个运动程序时，PMAC在一次执行程序的一个指令，进行该移动命令（包括非移动的任务）的所有计算，从而为该运动的实际执行做好准备。PMAC板总是工作在实际移动之前，如果要求的话，它总能正确地与即将执行的动作相调和。用户可以参看《编写一个运动程序》一章以得到更多的细节。

### 执行PLC程序:

运动程序的顺序特性使得它能很好适应一系列的运动并相互协调其他坐标系的动作；然而当在执行那些不是用运动的顺序来直接协调的动作时，这些程序就不适于作坐标系的运动。对于这些类型的任务，PMAC提供给用户编写“PLC程序”的能力。PLC的命名是可编程逻辑控制器,因为它们以一种相似的方式工作，在处理器时间允许的情况下尽可能快地连续扫描它们的操作。这些程序对于在运动顺序上不同步的任务是非常有用的。用户可参看《编写一个PLC程序》以得到更多的细节。

### 伺服环更新:

在自动执行的任务中，对于PMAC的用户来说其实质上是不可见的。因此对于每一个电机，PMAC都

以一个固定的频率（通常是2KHz左右）对其进行伺服更新。伺服更新是这样进行的,先根据运动程序或别的运动命令得到的等式求得到要求的位置的增量（如果需要），然后将这同由反馈传感器读回的实际位置相比较,最后在两者的差的基础上发出一个输出命令使此差值变小，如此反复,直到此差值令人满意为止。这部分的功能是自动产生的,并不需要任何命令。用户可参看《闭环伺服》一章以得到更多的细节。

**换相更新：**

如果PMAC被要求给一台多相电机执行换相，那么它会自动的以一个固定的频率（通常9KHz左右）进行换相更新。换相更新，或者说是相位更新，对一台电机来说是这样的，测量并估算（或者要么只测量，要么只估算）转子磁场方向，然后再分配通过分布在电机的不同相位的伺服更新算出的命令。同样，这个功能也是自动产生的，并不需要任何的命令。用户可参看《设置PMAC换相》来获取更多的细节。

**资源管理：**

PMAC会定期自动的执行资源管理的功能，以确认整个系统是处于正常的工作状况下。这些功能还包括安全检查，例如随动误差限制，硬件超行程限制，软件超行程限制，放大器错误。通常也还包括看门狗计时器的更新。如果任何的硬件或软件的问题使这些功能不能得到执行，则看门狗计时器将会触发，从而使卡关闭。用户可参看《确使你的应用安全》一章以获取更多的细节。

**与主机通信：**

PMAC可以在任何时间与主机通信，甚至是在一个运动序列的中间。PMAC将接受一个命令，然后采取相应的动作—将命令放入一个程序缓冲区以便以后的执行，提供数据以响应主机，开始电机的移动，等等。如果命令是非法的，它将会向主机报错。

**任务优先级：**

任务是按照优先级电路组织起来的，这可以使它们得以最优化，从而让应用程序能有效、安全地运行。当优先级固定下来以后，不同任务得以执行的频率是在用户的控制下。用户可参看《设置PMAC换相》、《闭环伺服》、和《计算特征》这几章以获取更多的细节。

# §３—２ 同PMAC对话：

这一部分覆盖了主机与PMAC之间通信的基本的方法。在我们假定你正在编写一个程序来让你的主机完成这些通信功能。PMAC的可执行程序（附件9D）是这些程序中的最常用的。

如果在你的最终使用中有一台计算机，那么你需要编写你自己的通信程序，作为为了这用途的前端软件的部分。这是一个很超前的内容，在后面的《编写一个主机通信程序》这一章还将详细讨论。现在,我们将集中在实际的通信方面。

在基础的水准上，PMAC可以同主机的简易终端通信，不论是通过串行口（RS—422）或者是并行（总线）。这些信息大多是由数行来回传送的ASCII字符构成。当然,大多数时间主机都是一台有着相当多的智能的计算机,但在底层时它就象是一个终端同PMAC卡通信。PMAC的可执行PC程序有一个终端仿真模式来直接这么做。

## 通信端口：

每一种版本的PMAC板都能通过它的串行端口或者是并行（总线）端口来实现通信功能。在不同的硬件版本的PMAC板中的最主要的不同是总线界面的类型：PC、STD、或者VME。

PMAC通常都准备从两个端口中的任何一个端口接收命令（除非串行口在进行波特率跳线时被特殊设置为无效）。然而重要的是不要让PMAC板同时通过两个端口来接收命令；因为如果这样的话，字符将会混杂并且命令也会混淆。

**激活响应端口：**

当PMAC板送出它对命令的响应时，不是串行端口就是总线端口将被激活。串行端口作为被激活的端口时，PMAC板开启或是重置。可是,任何通过总线接收的命令都将使总线端口成为激活的端口（在多数总线－主机的应用中这都是立即发生的，对于用户也是很明显的）。后面的响应将返回到总线端口上。

从串行端口来的后一个命令将不会自动地把串行端口再次变成激活的响应端口，所以PMAC板将有可能通过把数据送到总线端口来响应一个从串行端口来的命令。这可能会使主计算机糊涂。要想使串行端口再次成为激活的响应端口，你必须给PMAC板送一个<CTRL-Z>字符。

如果你有一个基于总线的系统，但你只用一台辅助计算机通过串行端口做一些诊断工作，例如数据的采集或者是PMAC可执行程序的打开，那么一定要记住停止总线通信。从总线来的一个单独的命令能够使我们任意通过串行端口的慢反应过程停止。

**串行接口：**

在不同版本的PMAC板上的给PMAC串行接口端口的硬件设置会有轻微的不同。

PMAC－PC，－VME的硬件设置：

PMAC－PC和－VME都有一个在26脚IDC插座(J4)的RS-422接口。这个端口直接同主机上的标准DB－25插槽相连接,而此DB-25插槽用26芯扁平线插座相连接。对于一个DB-9主机插槽，在线的另一头应使用一个标准的9到25脚接头。

PMAC－Lite:

PMAC-Lite有一个10脚IDC插座(J4)转换的RS－232接口。这个端口直接同主机上的标准DB－9插槽相连接，而此DB－9插槽与一10股扁平线插座相连接。对于一个DB－25主机插槽，在其线缆的另一头应使用一个标准的25到9针接头。对于一个RS－422口,选件9L附件板需要增订。这就给RS－422端口提供了一个DB－25插槽。而RS－232端口将无效。

PMAC－STD：

PMAC－STD既有一个在5脚SIP插槽上的RS－232界面（底板J1）,又有一个在20脚微型IDC插槽上的RS－422界面(底板J3)。但在任何时候两个界面中只能有一个被连接上。

PMAC1.5-STD:

PMAC1.5-STD有一个在5脚SIP插槽和一个10脚IDC插槽的RS-232接口，10脚插槽可以用一根10芯扁平电缆同主机上的标准DB－9插槽直接连接起来。PMAC1.5－STD还有一个在一26脚IDC插槽的RS－422接口，它可以用一根26芯扁平电缆与主机上的标准DB－25插槽连接起来。

RS－422与RS－232：

PMAC的RS－422串行接口与许多个人电脑上拥有的RS－232接口非常相似。RS－422有0到＋5V的差动信号，而RS－232有单端－10V到＋10V信号。PMAC的RS－422接收器可以很好地接收来自RS－232的带有显著噪声容限的输入。大多数个人电脑上RS－232接收器能够很好地读入来自PMAC的RS－422的信号,但噪声容限将趋向最低，并且在这个方向上的通信可能会出错，特别是在有PWM放大器存在的地方。对于同主机RS－232的拉强通信,附件26提供了交换能力和光电隔离。当然可以直接同主机上的RS－422端口进行通讯。

波特率：

串行端口波特率是在启动时由E44~E47跳线（PMAC－PC、－Lite、1.5-STD、-VME）或者切换开关SW1－1到SW1－4（PMAC－STD），以及PMAC的主板时钟频率所决定的。串行波特率能在一块20或40MHz板上被设置到76,800波特,或在一块30或60MHz板上设置到115,200波特

。如果E44~E47都是“开”（切换开关SW1－1到SW1－4都是“关”），则串行端口失效。

信号线：

由于串行接口随着系统的不同而变化,因此PMAC提供了一个简单而灵活的接口。除信号地之外，只有四根线是所要求的（如果算上差分线的则是八根）：数据传输、数据接收、清除发送线、和准备发送线。如果需要的话,这些线可以通过跳线(E9~E16)来改变,以便同主机的设置相配。PMAC将DSR和DTR线短接到一起以便为那些有特殊要求的系统提供在此门电路转换开关的自动反回信号。

数据格式：

串行通信数据格式是8位的，1个起始位，1个停止位，如果E49跳线是“开”则没有奇偶位，如果E49跳线是“关”，则有奇偶位（PMAC－STD正好相反）。PMAC可以将它从主机接收来的每一个字符都反送回给主机；命令〈CTRL－T〉控制这项功能的开和关。不支持XON/XOFF的交接处理。逐行的校验可被计算；由变量I4控制这项功能。

**PC总线接口：**

对于PMAC－PC和PMAC－Lite的PC总线接口只能同PC－XT总线（8位）一起工作。尽管提供了附加的AT总线插槽,但它只能用来连接附加的AT中断线。一块PMAC在PC的I/O端口空间占有16个地址。这个空间的基本地址是由E91~E92和E66~E71跳线决定的。当然，这个地址应该加以选择以避免同PC里面的任何别的事物的地址冲突。出厂时的缺省地址设置是528（以十六进制为210）。跳线描述章节包括了典型PC的I/O映射图和可能的空地址。

通过这些地址之一，字符可以一次传送一个。支持这个接口的软件同支持串行口的软件非常相似。当然，字符通过总线端口传送可以更快。

选件2(双端口RAM)给在PMAC板和主计算机之间来回传送的数据提供了8K×16位的共享内存。这个内存的数据路径是16位宽，因此可以直接支持AT总线。

**STD总线接口**：

PMAC-STD的总线接口的工作情况实质上与PMAC-PC或-Lite上的总线接口是一样的。它既能同最初的8位STD总线连接在一起工作,也能与新的32位STD32总线连接在一起工作。在主机的I/O空间中它占据了16字的空间。这16个字的基本地址是通过在PMAC－STD的底板上的W11~W22跳线来决定的。出厂时的默认设置基本地址是61584(十六进制为F090 hex）。跳线部分包括了一个典型STD总线计算机I/O映射的详细情况和可能的空地址。

新的PMAC1.5-STD不支持具有32位的STD32总线,尽管它也能通过十六位地址线同STD80或STD32总线连接起来工作。在PMAC1.5-STD上,DIP转换开关S1－1到S1－12控制STD板的总线地址。

**VME总线接口：**

VME总线的PMAC-VME界面象是一个VME总线的从属装置。命令和响应都是通过一套16×8位的“邮件箱寄存器”来传送的。二进制数据可以通过在载选件2V双端口RAM(8K×16位)来传送。邮件箱寄存器的数据总线是8位宽，而DPRAM的数据总线则是16位宽的。地址总线可以被设置成16，24或32位。

该接口的地址和特性的设置必须这样来完成：通过由串行口向PMAC内的寄存器写值，将这些值保存到非易失性的内存中，并重置卡。对于这种模式的板的典型的初始化设置和以后的发展是用提供的PMAC可执行程序通过一台IBM－PC或别的兼容的主机带有的串行端口通讯来完成的。关于如何设置VME总线接口，用户可参看下面《编写一个主机通信程序》部分。

## 给PMAC发出命令：

PMAC基本上是一个命令驱动设备，它并不象别的寄存器那样是靠寄存器驱动的。你可以通过向PMAC发出ASCII字符串从而让它工作，而PMAC通常也以ASCII字符串的形式向主机提供信息。

如果你有选件2双端口RAM，那么你可以让PMAC更有效的工作；你可以通过向DPRAM中指定的寄存器写值来发出命令，并且PMAC也通过向这些寄存器中放入二进制的值来提供信息。但你必须已经向PMAC送出ASCII命令，让它在接收到这些值时能采取正确的动作，并将这些值放入寄存器中。

**PMAC的指令执行进程：**

当PMAC通过它的端口之一接收到一条字母数字的字符,它除了将该字符放入命令队列中并不做别的什么。它需要一个控制字符（ASCII值1到31）来使它产生实际的动作。最常用的控制字符是“回车”(ASCII值是13),它告诉PMAC将前面的字母数字字符按照一条命令进行翻译并采取相应的动作。

控制字符：

其他的控制字符将导致PMAC产生一个独立于此前发送字符的动作。这些控制字符能被送到一行字母数字命令字符的中间而不会打乱命令流程。PMAC将首先响应控制字符命令,并将内容直到回车的字符串存储起来。

**命令识别：**

PMAC的命令识别和数据响应的确切特性是由I一变量I3，I4，和I9控制的，其中I3最为重要。如果I3是1，则PMAC通过向主机送回一个“换行”（〈LF〉；ASCII值10）字符来识别一个有效的字母数字命令。如果I3是2或3，则它用〈ACK〉字符（ASCII值是6）。如果I3是0，则PMAC不提供任何识别字符。不论I3的设置如何，PMAC总是通过返回一个〈BELL〉字符（ASCII值7）来响应一条非法命令。当以终端模式与PMAC交互式地工作时,通常用〈LF〉作为应答比较好，因为它能自动地在命令中留间隔并能响应在终端屏幕上。

**数据响应：**

当命令接收到要求一个数据响应时,如果I3被设置为1或3则PMAC在数据响应的每一行前加一个换行字符。如果I3被设置为0或2，则不会那么做。不论I3的设置是怎样的,PMAC都将在数据响应的每一行加上一个回车字符作为终结。对于这些命令，命令应答字符—〈LF〉或〈ACK〉—
是在数据响应之后被送出的，其作用相当于一个传输终止字符。对响应的计算机分析，则最好将〈ACK〉用作一个唯一的EOT字符。

**数据完整性：**

变量I4决定了一些在通信中由PMAC执行的数据完整性检查，其中最重要的是逐行校验。在《编写一个主机通信程序》一章中，对这一功能有详细的叙述。

**数据响应格式：**

变量I9控制PMAC将怎样的数据格式传送到主机。它的设置决定了PMAC是否要将程序行以长的或短的格式返回给主机，是否要将I一变量的值和M一变量的定义作为完全的命令来汇报，选址I一变量值是以十进制格式进行汇报还是以十六进制的格式来汇报。

## 在线（立即执行）命令：

发给PMAC的命令中许多都是在线命令；也就是说，它们可以立即被PMAC执行，或者是产生某些动作，或者是改变某些变量，或者是把某些信息报告给主机。这种命令本身在执行后就被丢弃掉了(因此不被回显），尽管命令的效果也许依然存在。

一些命令例如P1＝1，如果没有已打开的程序缓冲区，就会被立即执行。如果有一个缓冲区被打开了，就会被存储到缓冲区中。另外一些命令不能成为在线命令，它们必须放到一个打开的缓冲区中，即使是为立即执行的特殊缓冲区，例如X1000 Y1000。如果没有打开的缓冲区，PMAC将拒绝执行这些命令（如果I6被设置为1或3,报错为ERR005）。当然还有一些命令，例如J＋，只能是在线命令，不能被放到一个程序缓冲区中去（除非以形式CMD“J＋”）。

**在线命令的类型：**

共有三种基本类型的在线命令：电机定义命令，只影响当前被主机选址的电机；坐标系定义命令，只

影响当前被主机选址的坐标系；全局命令，不论是何种选址模式都影响卡的特性。在用户手册的参考部分，每一条命令都被归入到这三种类型中的一种。

**注意：**每一个用COMMAND指令从卡发出在线指令的程序，都有其自己电机和坐标系的地址，改变主机的地址不会影响到程序。另外，控制面板通过BCD码形式的旋转开关来选电机和坐标系也不受主机的地址影响。

**电机特性命令:**
电机编址:

一个电机是通过一个#n命令来进行编址的，n是该电机的序号，其范围为1到8。电机将始终处于当前编址的位置，直到PMAC板接收的另一个#n命令。例如，命令行#1J+#2J-
告诉电机1按正方向微动，并且电机2沿反方向微动（象大多数的命令一样，微动命令只有在回车字符被接收之后才会产生效果，所以在这种情况下，两根轴将差不多同时开始动作）。

电机命令：

电机特性命令只有少数的几种类型。包括微动命令、回零点命令、开环命令，以及控制电机位置、速度、随动误差和状态的命令。

**坐标系命令：**
坐标系编址：

一个坐标系是通过一个&n命令来进行编址的，n是该坐标系的序号，其范围是1到8。坐标系将保持当前的编址直到PMAC板接收的另一个&n命令。例如，命令行&1B6R&2B8R告诉坐标系1执行运动程序6，而坐标系2执行运动程序8。

坐标系命令：

坐标系特性命令有形式多样的类型。轴定义语句在当前编址的坐标系起作用，因为电机总是与一轴在特定的坐标系中相匹配的。既然是一个坐标系来执行一个运动控制程序，所有的程序控制命令在编址的坐标系中起作用。Q－变量赋值和查询命令也是坐标系命令，因为Q－变量本身就属于一个坐标系。

要注意的是如果不止一个电机被分配给一个坐标系，那么对于坐标系下的一个命令将影响到这好几个电机。例如，假如电机4被分配给坐标系1，则对于坐标系1执行一个运动程序的一个命令能够让电机4开始移动。

**全局命令：**

一些在线命令并不依靠哪一个电机或坐标系被编址。例如，命令P1＝1把P1的值设为1不管是否有电机或坐标系被编址。所有全局在线命令都是缓冲区管理命令。PMAC有若干缓冲区，它们之一能在某一时被打开。当一个缓冲区打开时，之后输入的指令被放到该缓冲区中。

控制字符命令（ASCII值为0~31D）一般都是全局命令。那些不要求数据响应的数字符通过串行数据菊花链作用于所有的卡。这些字符包括回车〈CR〉、空格〈BS〉和一些特殊用途的字符。这就允许在同一行发出的位置命令，在命令行的末尾接收到回车字符时同时产生效果（&1R&2R〈CR〉）将使坐标系1和2都运作起来）。

## 缓冲的命令：

正如它们的名字所暗示的那样，缓冲的命令并不立即被执行，而是被保持着以便后面的执行。PMAC有许多程序缓冲区—
256个常规程序缓冲区，8个循环程序缓冲区（每个坐标系一个），和32个PLC程序缓冲区。在命令能够被放入一个缓冲区内之前必须确保该缓冲区是被打开了的（例如：OPEN PROG 3，OPEN PLC 7）。每一个程序命令都被加到打开的缓冲区的命令列表的末尾。

如果你想替换掉现在的缓冲区，在打开该缓冲区后立即用CLEAR命令，在进入新的缓冲区之前清除已存在的常量。在完成输入程序后，用CLOSE命令关闭打开的缓冲区。

**循环运动程序缓冲区：**

循环运动程序缓冲区是一个特殊的程序缓冲区，它能在执行运动程序的同时打开，以便来自主机的程序命令可以进入。如果一个打开的循环程序缓冲区正在工作，但已经执行完送给它的每一条命令，它将立即执行送给它的下一个缓冲区的程序命令。

## 多卡应用：

如果有多卡同主机相连通信，那么对于主机来说必须要有在不同的卡之间互相区分的方法。主机必须能够分别地同每一个卡通信，而有时又能同所有的卡进行通信。因此，主机必须有一种给卡编址的方法。

**总线通信：**

当处于总线接口时（例如：PC－bus,STD-bus或VME-bus），不同卡之间的区分是通过硬件编址来实现的。这就意味着不同的卡将响应不同的总线地址，通过总线选址线来选择。多卡的硬件编址的设置同单卡的一样（参看前面）；不能将两片不同的卡放到总线上的同一硬件地址中。

同时的命令：

因为命令只能是顺序地被发往不同的卡，所以一个小的特殊操作用来初始化所有卡上的同时动作。因为字符被顺序地送到所有的卡上去的速度是如此之快，以至于这个延时并不成为各卡之间动作的同时性的主要限制。命令应该被送出以便除回车字符之外的所有命令都已被送给了所有的卡；然后〈CR〉命令被高速顺序地送到每一个卡。（如果你检查写准备位，那么确认在送字符给任何卡之前所有卡的这一位应该是“真”
。）在一个典型的总线系统上，你送这些字符的每字符间的间隔大约只有百万分之一秒。这远快于PMAC上的命令解释程序的大约一毫秒的软件扫描时间，所以命令可以被有效的同时发出。

**串行通信：**

然而，如果串行通信被使用（RS－232或RS－422），PMAC的菊花链将不允许相互隔离的硬件编址，因此必须要有一个软件编址的方案。

装备有PROM1.13版本或更高版本的PMAC卡通过用RS－422端口能够进行菊花链通信。PMAC－Lite和PMAC-STDs不能通过用RS－232端口来使用菊花链通信；要求使用RS－422端口（PMAC-Lite的选件9L）。最多可有16个PMAC板能被连接并同步地使用串行端口通信。然而为了这么做，一些硬件和软件的设置程序必须用到。

连接：

当从单个的主机串行端口给多片PMAC板进行串行通信时，连接是通过一个多点“菊花链”线路来完成的。在它的一头是给主机的一个插头（一般是一个DB－25插头）。在线路的另一头是一个给在链上的每一个PMAC的插头（“点”）。该线缆的每一股从每一插头的同一引脚引出。

*多口*线:

ACC-3D提供了与单个PMAC-PC或PMAC-VME的RS-422端口的串行连接,每一个用它指定的ACC-3E都提供了一个额外的点给附加的PMAC-PC或PMAC-VME。如果与主机上的RS－232端口连接，建议使用ACC－26转换器或者相似的转换器，特别是在多点应用当中。

**注意：**如果要将一片PMAC-Lite与另一片PMAC联合起来用，则要求用到选件9L RS－422接口。既然这样，ACC－3D 26脚串行线应该被用到，而不是ACC－3L 10脚串行线。

PMAC－STD既有RS－232端口又有RS－422端口。如果想要用到菊花链，必须用RS－422端口。***Delta Tau***没有提供这种线。

串行卡编址：

软件编址是通过@n命令来完成的，其中n是一个十六进制的数（从0到F）------最多可将十六片卡在一个主机下串联在一起。@@命令对所有的卡同时进行编址，但当系统处于这种模式下时通过串行端口送出一个要求响应的查询命令是不合法的（哪一片卡将会响应？）。

设置卡的地址：

通过主机发出软件编址命令必须和特定的PMAC板的板卡号相匹配，这个板卡号在PMAC－PC、PMAC－Lite、PMAC1.5－STD和PMAC－VME上是由E40~E43决定的，而在PMAC－STD上是由开关SW1－1到SW1－4决定的。参看下面的表或《跳线描述》一章得到正确的跳线配置。在链上的一片卡必须被设置为卡@0。推荐其它的卡依次顺序地从零开始编号（@1、@2，等等）。

**给PMAC－PC、-Lite、1.5－STD和－VME卡地址控制E点**

| E40 | E41 | E42 | E43 | 卡地址 | 缺省 |
|---|---|---|---|---|---|
| ON | ON | ON | ON | @0 | @0 |
| OFF | ON | ON | ON | @1 | |
| ON | OFF | ON | ON | @2 | |
| OFF | OFF | ON | ON | @3 | |
| ON | ON | OFF | ON | @4 | |
| OFF | ON | OFF | ON | @5 | |
| ON | OFF | OFF | ON | @6 | |
| OFF | OFF | OFF | ON | @7 | |
| ON | ON | ON | OFF | @8 | |
| OFF | ON | ON | OFF | @9 | |
| ON | OFF | ON | OFF | @A | |
| OFF | OFF | ON | OFF | @B | |
| ON | ON | OFF | OFF | @C | |
| OFF | ON | OFF | OFF | @D | |
| ON | OFF | OFF | OFF | @E | |
| OFF | OFF | OFF | OFF | @F | |

**给PMAC－STD的开关地址控制**

| SW1-1 | SW1-2 | SW1-3 | SW1-4 | 卡地址 | 缺省 |
|---|---|---|---|---|---|
| OFF | OFF | OFF | OFF | @0 | @0 |
| ON | OFF | OFF | OFF | @1 | |
| OFF | ON | OFF | OFF | @2 | |
| ON | ON | OFF | OFF | @3 | |
| OFF | OFF | ON | OFF | @4 | |
| ON | OFF | ON | OFF | @5 | |
| OFF | ON | ON | OFF | @6 | |
| ON | ON | ON | OFF | @7 | |
| OFF | OFF | OFF | ON | @8 | |
| ON | OFF | OFF | ON | @9 | |
| OFF | ON | OFF | ON | @A | |
| ON | ON | OFF | ON | @B | |
| OFF | OFF | ON | ON | @C | |
| ON | OFF | ON | ON | @D | |
| OFF | ON | ON | ON | @E | |
| ON | ON | ON | ON | @F | |

多卡模式变量：

当通过单个菊花链连接器同多块卡通信时，为了正确地通信，在链接的每一块卡的变量I1应该被设置为2或3（通常为2）。如果在连接线上只有一块卡，I1应该被设置为0或1，通常为0。如果这个在初始化连接时就应准备好的设置还没有被设置好，那么就把将I1赋值为2作为发送给卡的第一条命令，然后立即对其中之一的卡进行选址。例如，命令I1＝2
@0〈CR〉能够被用来给菊花链通信设置所有的卡，然后对＃0卡进行选址。

一旦这个设置做好后（并且用SAVE命令存储了起来），就没有必要发出这个命令，但是如果已发出了命令也没有什么害处。将I1设置成3比起设置成2来，能使CTS
处理无效，因此主机将不能从PMAC板接收字符；这是我们不希望的。

被选址的卡动作：

在任何时候被选址的卡都能接收字母数字命令并响应它们。只有它试图控制同主机的通信处理。在给定时间未被选址的卡将忽略通过串行口送给它的字母数字命令，并且它们的通信处理输出都是三态的，以便不会干扰那些已被选址的卡。未被选址的卡可以响应某些控制字符（不是那些查询卡的字符—参看以下部分），并且等候着看被选址的板卡号是否有变。

*处理数据响应：*

当发出一个要求数据响应的命令时，重要的是每一命令行只从一块卡请求数据（每一命令行由一〈CR〉字符终结），并且在查询另一块卡之前接受响应。否则当响应时将会有不止一个的卡去立即试图控制通信线。例如，命令@1P@2P〈CR〉将会导致两个卡都试图发送位置数据（直到发现〈CR〉字符，两卡才会开始执行命令）。

*同时编址：*

用@@命令可以将所有的卡同时进行编址以接受字母数字命令。既然这样，所有的卡将都能接受字母数字命令。卡@0将提供交接处理响应字符。在@@编址中，查询命令是不被允许的。如果在这种模式下主机送出了这样的一个命令，卡@0将用〈BELL〉字符响应。

启动状态：

对于设置为菊花链通信（例如，I1＝2或3,存储在EAROM中），卡@0作为已被选址的卡由启动/重置循环中产生，准备响应指令；所有别的由启动/重置循环中产生的卡未被编址，因此它们将忽略字母数字命令，直到它们被编址。

控制字符命令：

不要求数据响应的控制字符命令总是编址给在链上的所有的卡。这一类的命令有：

〈CTRL－A〉 取消所有的程序和运动
〈CTRL－D〉 使所有的PLC程序无效
〈CTRL－I〉 重复上一命令行（tab）
〈CTRL－K〉 停止所有的电机
〈CTRL－M〉 键入命令行（回车）
〈CTRL－O〉 所有坐标系进给保持
〈CTRL－Q〉 退出所有运动程序
〈CTRL－R〉 所有坐标系运行
〈CTRL－S〉 所有坐标系分步运行
〈CTRL－W〉 从（总线端口）双端口RAM取出命令行
〈CTRL－X〉 擦去命令和响应队列
〈CTRL－Z〉 使串行端口成为激活的响应端口

*注释〈CTRL－M〉：*

回车字符使得命令行结束，并让在链上的每一片卡可以接受该命令并执行它。这就允许单独命令行被送给每一块卡，但同时得到执行。如果在链上有一块特殊的卡当它看到回车字符时仍未接收到任何命令，那么它将执行一个“无操作”命令。

*例如：*

命令@0&1B4R@1&3B25R<CR>将使卡@0的坐标系1开始执行运动程序4，并且卡@1的坐标系3将开始执行运动程序25。

控制字符命令要求被当前选址的卡接受并执行数据响应，而被别的卡所忽略。在@@编址模式下面的命令将被拒绝。这一类命令有：

〈CTRL－B〉 汇报所有的电机状态字
〈CTRL－C〉 汇报所有的坐标系状态字
〈CTRL－E〉 以二进制汇报数据采集地址内容
〈CTRL－F〉 汇报所有随动误差
〈CTRL－G〉 汇报全程状态字

〈CTRL－P〉 汇报所有电机位置
〈CTRL－V〉 汇报所有电机速度
〈CTRL－Y〉 汇报并重复上一命令行

*注释:*

要求数据响应的控制字符命令将由最近被执行的编址命令所选址的卡执行。一条编址命令要到下一个回车字符才能被执行，但控制字符命令是在回车之前被执行的，所以在编址命令和控制字符命令之间发送一个回车字符是很重要的。

别的控制字符命令和它们在多卡应用中的性质是：

<CTRL-H>（退格键----
擦去被传送的最后一个字符）当它被发送给卡时，实际上是在整个数据流中得以执行。它擦去在数据流中被发送的最后一条字母数据命令。重复〈CTRL－H〉字符能够擦去自上一个回车字符的所有字母数字命令。如果其中包括编址字符，也会被一块擦除。

<CTRL-T>全双工通信（回显字符）在菊花链串行模式里是不被允许的。因此，<CTRL-H>命令（全/半双工触发）在这种模式里将会被拒绝。

# PMAC复位

PMAC被复位有好几种方法。第一种方法是先将5V电源关闭，接着将它打开。第二种方法是先让JPAN插座上的INIT/线上电平为低，接着让它为高。第三种方法是使用后连线板总线的复位线。这种方法要依赖于E39跳线的设置，如果是PC－bus版本，则依赖于E39和E94的设置。第四种方法是通过端口向PMAC发出$$$命令。

**PMAC复位动作：**

当PMAC板从前面的任意一种方式接收到复位信号或命令时，它立即停止所有激活的运算并且开始复位周期。在复位周期的开始，它使所有的输出无效，并将程序包读到激活的内存中。然后按照由硬件配置和重新初始化跳线E51的设置所决定的方式进行激活的内存的读取。

通过是EEPROM和电池备份的RAM（“标准CPU”部分，是缺省的配置，选件4和5），在电源关闭或复位周期中PMAC的内存信息可以被保留，，或者是完全通过闪存（“可选CPU”部分，是用选件4A、5A和5B）。如果是通过标准CPU部分，基本的用户卡信息（包括I－变量、交换表的设置、VME和DPRAM的地址设置）在用SAVE命令写入后就被保持在非易失性EEPROM中。用户程序、表、缓冲区和定义只是简单地通过电池保存在RAM里。在电源关断或复位周期过程中，没有命令或动作被要求来保持这些项目。

如果是通过可选CPU部分，所的用户卡上的信息在用SAVE命令写入后就被保存在非易失性闪存中。在电源故障或复位周期过程中，没有信息保留在RAM中。因此，在重置过程中必须用到SAVE命令将任何所需信息保存在卡中。

如果E51跳线是处于它的缺省状态（对于PMAC－PC、-Lite、－VME和1.5STD是OFF；对于PMAC－STD是ON），PMAC将上一次保存在非易失性存储器中的内容拷贝到激活的存储器中。对于通过标准CPU部分的PMAC这些只包括存储在EEPROM中。别的项目依然象重置之前的状态那样被保存着。对于通过可选CPU部分，这包括所有的用户设置：变量、定义、程序、缓冲区和表。

经过复位周期，所有的增量编码器的计数器被设置为零。在复位周期结束时，所有被激活电机的Ix80变量设为“1”以便能电机。其它的电机仍被留在停止的状态；等待着一个命令来激活它们。

**PMAC重新初始化动作：标准CPU**

当使用标准CPU部分，如果E51的跳线不是处于缺省状态时（对于PMAC－PC、-Lite、和－VME是ON；对于PMAC－STD是OFF），PMAC在复位周期中执行重新初始化。不同的是PMAC将从程序包EEPROM中把出厂时的缺省值拷贝到激活的存储器中，而不是从PROM中将保存的参数的值拷贝到激活的内存中。

这种重新初始化过程只有在由于软件或参数错误以及通信不能建立起来而导致卡被锁死时才是必须的

。这种类型的最普遍的例子就是那些意外地重复SEND或CMD语句（试图在重新初始化之前发出一个<CTRL-D>字符）的PLC程序，或带有太多的被激活的电机而伺服处理时间却很短。

**PMAC重新初始化动作：闪存CPU**

在PMAC使用闪存且E51跳线是ON的情况下，当PMAC执行它的复位循环时，PMAC进入了允许新的程序包下载的特殊的重新初始化模式。在这种模式下，PMAC只能通过PS/STD总线端口，或以波特率38,400通过串行端口进行通信，不论波特率跳线的设置情况如何。在这种模式下只有一个非常基础的引导程序包得以执行。

在这个引导模式下，只有很少的命令选择。PMAC将用响应*BOOTSTRAP PROM*来响应任何状态字查询命令（？，？？，或？？？）。这就允许主机知道PMAC是否处在这种模式下。PMAC将用引导程序包的序号来响应版本查询命令（例如，1.01），它可能比操作程序包版本不同。

标准重新初始化：

在这种模式下如要略过下载操作只要向PMAC发送一个<CRTL-R>字符。这就将PMAC放到一个带有现成程序包的标准操作模式中。给I一变量的出厂时的缺省值，交换表的设置，和给DPRAM和VME的总线地址都从闪存的程序包部分拷贝到激活的存储器中。这些量的被存储的值没有被使用，但它们仍然保存在闪存的用户部分。

注意：在试图升级PMAC操作程序包之前，确认已经把所有的PMAC设置存储到盘上。如果新的程序包提供了一个不同的用户存储器映射，PMAC将在新的程序包被加载之后，电源开启时清除存储器。即使不是这种情况，设立一个新的程序校验参考值的最容易的方法就是发送一条 $$$***命令，该命令将会清除缓冲区。

对于在操作程序包里的任何改变，已编译好的PLCs不得不用新的程序版本的LIST LINK文件重新编译。重要的是在试图改变操作程序包版本之前，删去所有的已编译好的PLCs（DELETE PLCCn）。正在一个版本的程序包之下执行的已编译好的PLC程序不同于那些为该程序包所编译的PLC程序，可能会引起难以预料的结果。

给PMAC下载新的操作程序包只要通过串行端口给PMAC发送一个<CTRL-O>字符,引导程序会将这作为准备下载新的操作程序包进行解释。随后通过串行端口接收到的字节都将被认为是程序机器代码的二进制编码字节，并将被写到闪存中。送完<CTRL-O>命令后,在开始下载操作程序包之前,主机至少会等待5秒。这段延迟是确认闪存已经准备好被写入。在下载后，PMAC应该被关掉电源；在这段时间中，不要试图同PMAC进行别的通信。

在关掉PMAC的电源之后，E51跳线应该被撤除。当重新打开PMAC的电源之后，PMAC板应该能用新的程序包正常操作。使用SAVE命令存储在闪存的别的段里的用户设置不会受到新的程序包的下载的影响（除非新的程序包有不同的用户存储器映射）。当在这种重新初始化模式下主机与PMAC建立通信时，PMAC执行程序V3.x以及更高版本，能自动识别PMAC所处的模式。在这种模式下，File菜单上的菜单选项“Download binary firmware file。”可以被选择，以从磁盘上取得一个二进制文件并通过串行端口拷贝给PMAC。然后，程序强迫你退出，回到操作系统。这时，你应该关掉PMAC的电源，并且撤除E51的跳线。

如果在一台PC上运行老版本的PMAC可执行程序，或者是终端仿真程序，下载新的程序包的步骤如下（记住首先将你的PMAC软件备份并删去任何已编译好的PLC程序）：

1.) 以38400波特通过串行口建立同PMAC的通信。通过查看PMAC是否用*BOOTSTRAP PROM*

来响应？命令从而确认PMAC是否处于重新初始化的模式下。如果你正在执行程序，确认除终端窗口（例如位置窗口）外的所有窗口都被关闭，以便不会有别的命令被送给PMAC。

2.) 在终端窗口内键入〈CTRL一O〉字符并立即退回到DOS。这时不要送任何别的字符给PMAC。

3.) 用DOS的COPY命令的二进制版本（/B）将容纳着新的程序包的文件下载给PMAC。
键入的DOS命令如下所示：

*COPY/B B:V115A.BIN COM1:*

其中*B：V115A.BIN*以二进制机器代码格式容纳操作程序包的目录和文件名，*COM1：*则是用于通信的串行端口。

4.) 关掉PMAC的电源并且撤除E51跳线。

5.) 重新打开PMAC的电源，并且用新的程序包开始正常操作。

6.) 如果你想要更新你的程序校验参考值以便PMAC不用报告程序校验错误，最简单的方法是发出$$$***命令，该命令将使PMAC自动计算新的参考值（但是它同时也从激活的存储器中清除了所有你的程序和缓冲区）。作为一种备选方案，你可以送几次RHX:$0794命令。如果每一次你都得到同样的值，PMAC将会因为一个错误而停止校验计算，并且汇报的值就是它为程序校验所计算的值。把这个值写入参考寄存器X:07B1。例如，如果RHX:$0794数次都返回9A3B12，给PMAC发出命令WX：$07B1,$9A3B12。要记住不论是哪一种方法，你都需要用SAVE命令将这个参考值存入闪存。

7.)如果你需要重新编译PLC程序，可以用对应于新版本程序包的LISTLINK.TXT文件，也可以通过在新的程序包中向PMAC发出LIST LINK命令并将其响应存储到命名为LISTLINK.TXT的文本文件中，从而创建一个这样的文件。

重新初始化命令：

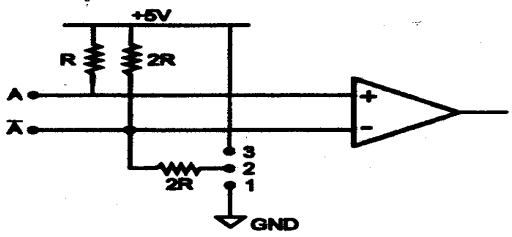
$$$***命令将使得PMAC重置和完全重新初始化。在读取缺省参数值的同时，它也清除掉电池保持RAM中的所有缓冲区：运动程序、PLC程序、表，等等。

一些用户总是在复位周期过程中让卡重新初始化；然后他们在每一个周期之后立即下载所有的参数设置和程序。在这种策略之后的逻辑是即使一块新的备用卡刚刚被放入，操作的同时启动顺序依然被使用。对于那些不希望在任何方面依赖于PMAC自己的非易失性存储（EEPROM和电池保持RAM，或闪存）的应用来说，这也是非常有用的。

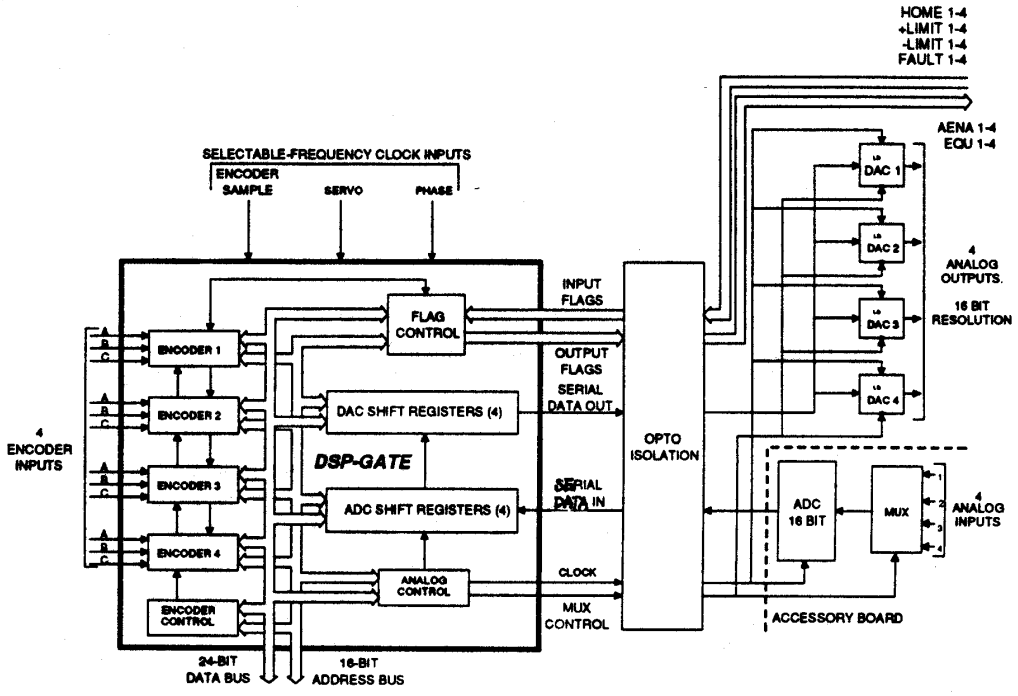
对于已知的完全初始化的PMAC，下列命令应该被加上：

```
P0..1023=0
Q0..1023=0
M0..1023->*
UNDEFINE ALL
```

记住这些命令只直接影响激活的存贮器（RAM）。要将新的设置拷贝到非易失性存储器中（EEPROM或闪存），要用SAVE命令。




# §3－3 输入与输出:PMAC与外部的连接




PMAC具有扩展的输入输出能力，模拟的和数字的，特殊功能的和一般功能的。它的I/O有多种特性来保证信号的完整性;由于采用多种I/O类型，我们将解释为提高每种类型I/O的数据的完整性而采取的步骤。

## 正交编码器输入（JMACH口）

作为一种标准特性，PMAC可接受0-5V的数字正交编码器信号。在PMAC的配置中，每个数字信号处理器可连接四个编码器。

**PMAC运动控制器的用户门阵列(图1 page3-22)**

**单端输入及差动输入**

对于每个编码器通道PMAC都有差动输入接收线，它既能接受单端输入(每个通道一根信号线)，也可接受差动输入(每个通道两根信号线，主线和辅线)。如下所示，用户通过每个编码器的跳线(E18-E21和E24-E27)自行设置。

差动输入接收线可接受主辅线间不超过+/-12V的输入，和每根线上不超过+/-12V的输入。通常在0到+5V之间。

## PMAC运动控制器的常规门阵列(图1 page3-23)

- 1与2连接到差分线+2.5V

- 连接2到3差分线+5V
- 当不连接时接+2.5V
- 单端编码器接+2.5V

- 不要用差分线驱动编码器
  - 开集电极编码器辅线接+5V(已废弃)
  - 外部XOR编码器接+5V

单端输入编码器

当由跳线设置该编码器为单端输入时，它的差动输入线被连至+2.5V;单端信号线将与之比较，在0到+5V间变化。

> 当使用单端TTL数字编码器时，差动输入线应悬空，不要接地或是接高电平，这对于PMAC的差动输入线的正常工作是必要的。

差动输入编码器

差动编码器信号通过抑制共模噪声可提高其抗噪声的能力。工业系统的现代设计标准正是采用了这种技术，特别是在产生大量电磁干扰的PWM功放中。

*集电极开路差动输入*

有两种类型的差动编码器信号。第一种是主辅通道都采用了简单的集电极开路驱动(或与之类似)。对于这种类型的编码器，跳线必须设置在差动模式，并为输入提供上拉电阻。

*差动线驱动*

全部信号采用差动线驱动是差动编码器格式的第二种类型(也是被极力推荐的)对于该类型的编码器，跳线如何设置无关紧要;大多数用户只须让跳线处于缺省位置。

*终端电阻*

当通过一根长导线驱动编码器信号时(10米或更长)，可能需要在主辅线间加上终端电阻以减小传递的瞬间干扰。PMAC为此提供了电阻槽。终端电阻的优化值是由系统决定的，但一般从330欧姆开始。

模拟编码器

如果驱动器有足够的能力在470欧姆上拉电阻的条件下工作，PMAC的差动输入接收器就可以接收模拟电压编码器输入，并且接收器上的最大差动电压在2V和12V之间。对于一个单端模拟编码器，辅助通道应接地以在电压信号正负变化时提供正确的转换。最好把跳线设在单端输入上，但不要求这样做。

对于一个差动模拟编码器，每个通道的两个信号象数字差动编码器一样被接在一起。最好把跳线设在差动输入上，但不要求这样做。在这里，12V的输入限制是信号的变化幅度。

提供电源和隔离

在PMAC的基本设置里，编码器电路并未与PMAC的数字电路隔离开，并且信号的参考点是PMAC的数字公共地。这种情况下编码器一般由PMAC的+5V来供电。编码器的总电流被看作由PMAC的电源提供。

连到PMAC上的非隔离信号的编码器也有可能使用一个独立的电源。在这种情况下，电源的一端应接在PMAC的数字公共地上来给信号提供一个公共的参考点。独立电源的+5V线不能被接在一起，因为它们会互相干扰。

*编码器的隔离信号*

在许多系统里，用户希望把编码器电路和PMAC的数字电路光电隔离开来。这在编码器与控制器距离长(大于10米或30步)或(和)电噪声高的系统中是常见的。隔离可通过使用附件8D Opt 6四通道编码器隔离板实现。对于被隔离的编码器，需要一个独立的电源来保证隔离效果，并且电源的一端不能与数字公共地连接，否则将破坏隔离。

*模拟编码器信号*

对于具有由无刷电机放大器内的R/D转换器提供信号的模拟编码器的系统，应予特殊考虑。在这些系统里，编码器信号几乎一直以放大器的信号返回为参考，它们接在PMAC的

模拟公共地AGND上。在这种情况下，为把模拟编码器信号与PMAC数字电路隔离开的最好设置就是采用附件8D Opt 6 隔离板或类似模块。这将使PMAC的数字电路与放大器保持完全的隔离。

如果模拟编码器信号的隔离不可行，那么PMAC的数字电路和放大器信号电路(包括PMAC的模拟电路)必须被接在一起以提供一个公共的参考电压。最好是打开跳线E85，E87和E88，把PMAC上的模拟和数字电路，以及模拟信号电路接在一起，需要避免的仅仅是用模拟编码器的导线连接电路。这可能导致信号的丢失甚至部件的损坏。

连线技术

对于编码器的连线有许多种有助于减轻噪声的重要技术。首先，编码器线应该尽可能地与电机的电源线隔离开。其次，这些线都必须被屏蔽，对于电源线是为了防止产生噪声，对于编码器线是为了防止引入噪声。这些屏蔽必须仅在内部端口被接地，也就是被连至那些自身被接地的设备上去。

*双绞线*

第三种重要的减轻噪声的技术是把并行的导线互相绞在一起。通过“双绞”，噪声会因为互相缠绕而抵销。

**编码器信号采样**

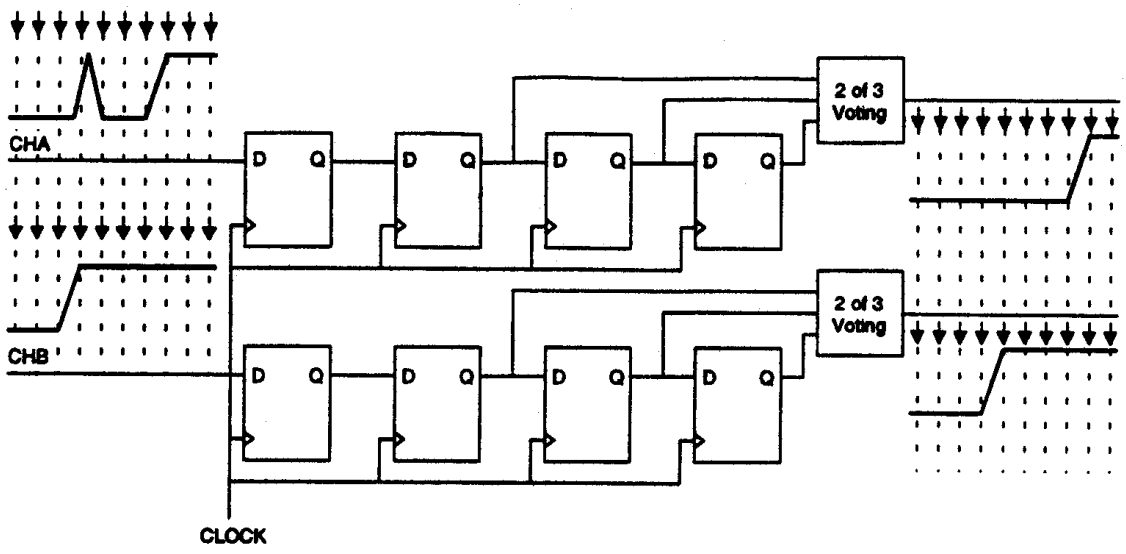
在经过差动线接收器的前端处理后，编码器信号以一个由SCLK--编码器采样时钟--频率决定的速率进行数字采样。SCLK是通过对主时钟的分频得到的，分频率由跳线E34到E38决定。缺省设置是E34为ON，对标准板上的10MHZ的主时钟进行二分频。

E35为ON时进行四分频；E36为ON时八分频;E37为ON时十六分频。当E38为ON时允许用户提供一个外部的SCLK信号(由CHC4和CHC4/输入)。使用的SCLK频率设置了可能的计数率的最高值；在实际使用中，最高的计数率要低大约20%，以允许不完善的输入信号。

**数字延迟滤波器**

每个编码器都有一个由每条线上的三个级联的D触发器组成的数字延迟滤波器，触发器的输出带一个最好的2/3门选电路。触发器的工作频率由SCLK信号提供。该滤波器仅允许保持了一个SCLK周期的状态变化通过;在这一段的任何变化都被视作噪声。通过这样做，滤波器把实际的转变延迟了两个SCLK周期--对于大部分系统是一个微小的延迟。

**编码器数字延迟滤波器(图3)**




频率的协定

SCLK频率越低，能被剔除的噪声尖峰越宽，但最高计数频率越低。用户在他的系统中必须综合这几个方面。一般地，SCLK应被设成允许PMAC保持编码器的最大的可能计数频率的最小频率。

旁路滤波器

该延迟滤波器可通过设置编码器I变量(I901，I906，etc。)为1来设置。该滤波器仅仅适用与那些使用并行子脉冲插补的用户，对于他们，延迟会导致转换错误。

**错误检测**

计数错误标志

如果一个非法的编码器转换(在一个SCLK周期内通道同时改变)通过—如果旁路，则跳过—

延迟滤波器，并到达编码器，将会设置一个计数脉冲错误标志，提醒注意丢失了位置信息。该标志为18位的编码器状态/控制字(对编码器1为X:$C000，对编码器2为X:C004，依此类推)。，M118，M218等M变量可用来存取这些位。

每转一次的检查

另外，通过PMAC位置捕捉功能并用编码器的第三通道进行每转一次的位置检查。从查看是否脉冲丢失查看PMAC与外部事件同步的位置捕捉描述和程序示例PLCMDD.PMC。PMC来获得更多细节。

## 光电隔离专用的数字输入标志(JMACH)

在JMACH PMAC的接口上，每个通道有四个专用的数字输入：+LIMn，-LIMn(行程限制)，HMFLn(回零标志)和FAULTn(放大器出错)。通过电机I变量Ix25的指定，这些变量通常作为专用的设置分配给一个电机。那些非专用的标志通过定义M变量被当作普通用途的输入。

**标志的连线**

所有的标志输入必须被短接在0V参考点上(通常是AGND)，允许电流流过光电隔离器中的LED，以被视为“0”状态。流向0V的电流只需将标志置为“1”；不需要外部的上拉电阻，并且它不会造成损坏。为作一个电气开关使用集电极开路输出方式，作为一个机械开关而常在标志脚与0V间用一个开/闭触点。

**行程限制输入**

当被指定为专用时，这些信号提供重要的安全和精度保障。+LIMn和-LIMn是有方向行程限制，它们必须保持低电平(从脚到地的源电流)以允许某个方向上的运动。

+LIMn和-LIMn的方向性与许多人的直觉相反。+LIMn应被置在行程的反端，而-LIMn应被置在行程的正端。

**回零标志输入**

HMTLn输入通常被用作回零或其它利用PMAC的硬件位置俘获特性的定位功能。编码器/标志I变量2和3决定哪个信号和哪个触发沿导致一个捕捉操作。

**放大器出错输入**

FAULTn输入通常被当作一个放大器出错时送出的信号。Ix25变量控制是高电平还是低电平有效。

要了解关于这些标志操作的更多细节，请参考“使用安全”和使PMAC与外部事件同步”部分。

**标志信号隔离**

这些输入信号与其它电路保持隔离。如果由模拟电源(+15V)供电并接在模拟地(AGND)上，它们将与PMAC的数字电路隔离。如果由数字电源(+12V)供电并接在数字地上，它们将与PMAC的模拟电路隔离。如果由独立电源(OPTO+V:12到24V)供电并接回电源自身的地，它们将与PMAC的模拟和数字电路隔离。跳线E89和E90的设置控制标志使用何种电源，也就同时控制哪个电路将被隔离。

## 专用数字输出标志(JMACH，JEQU)

在硬件配置中PMAC的每个通道有两个专用数字输出：放大器使能/方向信号(AENA/DIRn)和比较-相等信号(EQUn)。

**放大器使能/方向输出**

AENA/DIRn输出是一个接在同一电路上的光电隔离的输出。所以，它可与PMAC的数字电路光隔，或与模拟电路，或与两者同时隔离开来(见上)。

放大器使能的使用

对于由PMAC控制的放大器，这些输入通常被用作使能线。为了确保在需要时放大器能被完全的切断，这个控制功能是非常重要的。(不能依靠零输出电压;很容易发生偏移而导致一个零命令不能使之停止。在一个速度模式驱动器里模拟输出偏移表现为爬行;在一个扭矩模式的轻载驱动条件下，高速时将导致失控。)

转换

当PMAC发现放大器输出的一个出错信号时，它将自动关掉电机，把放大器使能信号置在无效状态。许多放大器，当它们由于任何原因被置成无效后，将送一个出错信号给控制器。即使放大器出现错误后，PMAC允许用户把它置成有效(把放大器使能线从无效改为有效)。但在下一个每几个毫秒，执行后台的“housekeeping”任务中的错误扫描时，如果放大器仍然出现错误，PMAC将再次使轴无效。

开集电极驱动器

这些输出的缺省驱动器是开集电极电路，需要外部的上拉电阻。它们通常被直接连在放大器的光电隔离器的阴极上(负端)。所用的ULN2803A额定值为10mA和24V；内部的二极管保护电路限制了送给模拟电源的最高电压，通常为+15V。为使保护失效并使输出超过+15V，IC的10管脚必须被去掉。

源极驱动

在PMAC的新版本中，通过跳线E101和E102，一个UDN2981A发射极开路(源)驱动器可替代标准的集电极开路驱动器。这可通过交换槽中的IC并改变跳线E101和E102来实现。

极性控制

这些输出的极性由跳线E17控制。对PMAC-PC，E17为ON则低有效(缺省);E17为OFF则高有效。对于所有的4或8条线，PMAC-PC有一个单跳线E17；PMAC-Lite，-VME，和-STD对于每个通道有独立的跳线E17A到E17H。对PMAC-Lite，-VME，和-STD，跳线为OFF则低有效(对PMAC-Lite和-VME为缺省)；跳线为ON则高有效(对PMAC-STD为缺省)。极性由硬件而不是软件控制的原因，是必须确认即使在软件失效的情况下，也可正确地使放大器无效。

失效保护极性

在放大器使能信号使用缺省驱动器时，低有效的极性(低电压--导通--为有效;高电压--不导通--为无效)为无电源提供了较好的保护。当失去PMAC计算部分的+5V电源，或模拟的+15V电源时，放大器可自动被断开，因为输出电阻会进入不导通状态。如果你需要这种保护但不能把这个极性信号直接连在放大器上，你必须使用中间电路改变信号格式。对于可选的源驱动器，高有效极性可提供更好的失效保护。

方向位使用

这些输出的一个可选使用是对于需要符号及绝对值命令的驱动器系统的方向位。一些伺服放大器需要这种命令格式，并且如果信号需要通过一个电压-频率转换器例如附件8D OPT 2来为步进驱动器产生一个方向脉冲时，使用这种格式。在这种情况下使用输出，对于使用该线的电机Ix25变量的16位必须置为1。这将禁止使用放大器使能线，同时I×02的16位也必须置1这使模拟输出为绝对值，并在输出中替换符号位。

普通用途

如果输出不是专用的，通过把M变量定义给该位(M变量定义的M114，M124，etc。)把它作为普通输出使用。

**比较-相等输出**

比较-相等输出的一个专门用途是在编码器位置到达一个给定值时提供一个信号沿。这对扫描和测量功能非常有用。这些输出的使用指导在以下关于“PMAC与外部事件同步”部分有详细叙述。

PMAC-PC

对于PMAC-PC，EQU输出没有一个固定的接口。但这些信号可通过在13个E跳线对，E53到E65—包括8条EQU线—上放置一个26针的IDC接口而引出。这些输出是驱动能力很低的TTL电路;它们在驱动任何实际设备之前，它们必须加缓冲。ACC27，通常作为多路拨码开关的I/O缓冲器，可被用来驱动一些EQU线。附件27带的26针导线适用于13个跳线对E53-E65。

PMAC-VME

对于PMAC—VME，这些信号由J7接口(JEQU)引出，参考点为数字地(GND)。出厂时，它们为集电极开路输出，用

一个ULN2803A驱动，额定值为24V，100 mA。
一个UDN2981A发射极开路(源)驱动器可替代它。这可通过插槽的IC并改变跳线E93和E94。

PMAC-Lite

对于PMAC-Lite，
这些信号由与数字电路光电隔离的J8接口(JEQU)引出，参考点为模拟地(AGND)或外部的标志源的地。出厂时， 它们为集电极开路输出，用一个ULN2803A驱动，额定值为24V，100mA。
一个UDN2981A发射极开路(源)驱动器可替代它。这可通过交换槽中的IC并改变跳线E101和E102来实现。

PMAC-STD

对于PMAC-STD，
这些信号由在每个级联式插件板上的J6接口(JEQU)引出，它们是内部带1千欧上拉电阻的集电极开路输出，额定值为5V。

对于PMAC-STD1.5， 这些信号由与数字电路光电隔离的J8接口(JEQU)引出，参考点为模拟地(AGND)或外部的标志源的地。出厂时，
它们为集电极开路输出，用一个ULN2803A驱动，额定值为24V，100mA。
一个UDN2981A发射极开路(源)驱动器可替代它。这可通过交换槽中的IC并改变跳线E101和E102。来实现。

## 光电隔离模拟输出（JMACH口)

PMAC可在JMACH接口上提供高精度的模拟输出，它们普遍地作为一个速度命令，一个扭矩命令，或一个相电流命令(对)来控制伺服放大器。PMAC
的每个通道提供互补的由16位数模转换器得到的DAC和DAC/输出。每个DAC输出范围为-10V到+10V，分辨率为300μV/位。

**连接**

如果放大器为单端输入，DACn将被用作命令线，AGND为返回点。如果放大器为差动输入，DACn将被用作命令线，DACn/为返回点。此时，放大器输入的公共端仍需接在PMAC的AGND上。

**隔离**

模拟命令输出电路与PMAC的数字逻辑电路是光电隔离的。模拟电路通常从放大器(大多数放大器为此提供+/-
15V)获得电源。可以使用跳线E85，E87，E88和E90将模拟电路的电源跳至主板的数字端，但这样破坏了光电隔离；在任何高功率或高噪声环境中，最好不要这样做，特别是在PMAC使用后连线板或非光电隔离的串联线与主机连接时。

**驱动能力**

模拟输出驱动倾向于大电流的高阻抗输入。220欧的输出电阻在所有情况下可保持采用电流小于50mA并防止损坏输出电流，但任何超过10mA的取用电流都会导致可察觉的信号失真。

**普通用途**

任何不作为专用伺服用途的模拟输出都可被用作普通用途的模拟输出。通常通过定义一个M变量给数模转换器寄存器(M变量定义M102,M202，etc.)，然后给它赋值来这样做。

## 通用的数字输入和输出(JOPTO口)

PMAC的JOPTO口(PMAC-PC，-Lite，-
VME的J5)提供八个普通用途的数字输入和八个普通用途的数字输出。在相对的列上每个输入和输出都有它自己的响应地。34针接口易于与OPTO-22或类似的光电隔离I/O模块连接。为此Delta Tau提供了一个长六英尺的扁平电缆，附件21F。

PMAC-
STD的接口形式与其它版本的PMAC不同。它的JOPT接口有24个I/O,可由软件选择作为输入或输出。除非特别提及,我们将不涉及PMAC-STD的接口。有关细节见“PMAC-

STD硬件指南”的有关JOPT接口部分。

**软件存取**

这些输入和输出通常使用定义M变量有软件进行读取。在M变量的定义中，变量M1到M8分别用于读取输出1到8，M11到M18分别用于读取输入1到8。该口进入内存的地址为Y:$FFC2。

**标准集电极开路输出**

出厂时，PMAC的八个输出由ULN2803A驱动。这些输出的电流可达100mA，但需要有有关上拉电阻。

不要把这些输出直接接在电压源上，否则过大的电流将损坏PMAC。

用户可给JOPTO接口的33针上提供一个高电压(+5V-+24V)，并通过连接跳线E1的1和2针来提高输出。对于ULN2803A的输出，跳线E2的1和2针也必须连上。

**警告：E1，E2的设置错误将损坏集成电路。**

**源输出的选项**

通过用UDN2981A替换ULN2803A来使这些输出成为源驱动器是可能的。这个集成电路(PMAC-PC的U3，PMAC-Lite的U26，PMAC-VME的U33)是插槽式的，很容易替换。这种驱动器必须使用下拉电阻。对于UDN2981A驱动器，必须连接跳线E1的2，3针和E2的2，3针。

**警告：E1，E2的设置错误将损坏集成电路。**

**输入源/集电极开路控制**

跳线E7控制八个输入的配置。如果它连接1，2针(缺省配置)，输入被偏置+5V为“OFF”状态，并且它们被下拉至“ON”状态。如果它连接2，3针，输入被偏置地到“OFF”状态，并且它们被上拉至”ON”状态。在任一状态下，高电压将被PMAC软件译为“0”，而低电压将被译为“1”。

## 多路拨码开关I/O口(JTHW口)

**多路拨码开关的使用**

JTHW(J3)接口上的多路拨码开关口，或多路拨码开关口有8根输入线和8根输出线。输出线可用来用作大量的输入和输出，并且Delta Tau提供了附件板和软件结构(特殊M变量定义)来实现这个特性。多至32和多路复用I/O板可以任意组合，形成菊花链。

辅助口

附件18为多路拨码开关提供多至16个BCD拨码位或每板64个离散的TTL输入。该板使用M变量的TWD和TWB形式。

附件34，-34A和-34B串联多路复用I/O板提供每板64个与PMAC光电隔离的I/O点。这些板使用M变量的TWS形式。

附件8D Option 7旋转转换板提供多至4个解算通道，它们的绝对位置可被该口读出。该板使用M变量的TWR形式。

附件8D Option 9 Yaskawa™绝对编码器接口板可与多至4个编码器连接。绝对位置在加电时通过多路拨码开关口串行读入。

**非多路复用的使用**

如果没有使用附件板，该口上的输入和输出将被当作离散的非多路复用I/O。它们进入PMAC处理器地址为Y：$FFC1。M变量定义为M40到M47为八个输出，M50到M57为八个输入。在非多路复用的形式下，附件27光电隔离I/O板为I/O提供缓冲，每点额定值为24V，100mA。

## 控制面板I/O接口(JPAN口)

JPAN接口(PMAC-PC，-Lite，-VME的J2，PMAC-STD的顶板)是一个26针的接口。包括专用控制输入，专用指示输出，一个正交编码器输入和一个模拟输入。控制输入内部有上拉电阻，为低有效。它们有预先定义功能除非控制面板无效I变量(I2)已被设为1。在这里，它们可通过分配M变量到它们的响应内存位置(Y：$FFC0)被用作通用输入。

**离散输入**

指令输入

JOG-/，JOG+/，PREJ/(返回上一个微动位置)和HOME/对由FDPn/线选择的电机起作用。STRT/(运行)，STEP/，STOP/(放弃)和HOLD/(保持进给)对由FDPn/线选择的坐标系起作用。

选择输入

四个低有效的BCD码输入线FDP0(LSbit)，FDP1/，FDP2/和FDP3/(MSbit)组成一个低有效的选择电机和坐标系(同时)的BCD码四位组。通常由一个单四位电机/坐标系选择开关控制。由这些输入线选择的电机将响应特定的电机输入。它也将打开它的位置跟随功能(Ix06自动设为1)；没被选择的电机将关闭它的位置跟随功能(Ix06自动设为0)。

当某个微动输入为低时不要改变选择开关的输入。解除微动输入将不能停止前一个被选中的电机。这将导致危险。

**可选用途**

如果I2被设为1，离散输入可被用作并行数据伺服反馈或基本位置。附件39手轮编码器接口板提供从一个正交编码器到这些输入的8位并行计数数据。处理数据请参考附件39手册和设置电机下的并行位置反馈转换部分。

**初始化输入**

输入INIT/(重置)对整个卡有作用。它与加电或主机的$$$命令作用相同。它属于硬连线，所以即使I2被设成1时它依然起作用。

**手轮输入**

手轮输入HWCA和HWCB可用跳线E22和E23连在PMAC的第二个编码器的计数器上。。如果这些跳线为“ON”，则编码器2上不应接其它东西。由I905控制，信号可被译成正交或脉冲(HWCA)和方向(HWCB)。I905也控制输入的方向性。确认编码器2跳线E26设在单端信号输入上，即连接1，2针。

**模拟输入**

WIPER模拟输入(以数字地为参考点，PMAC-PC，-VME和-STD为0到10V，PMAC-Lite为-10V到10V)提供一个25KHZ/V电压频率转换器(V/F)的输入，频率范围0-250KHZ。V/F的输出可通过跳线E72和E73连至编码器4的计数器上。
如果这些跳线为"ON"，则编码器4上不应接其它东西。
确认编码器4跳线E24设在单端信号输入上，即连接1，2针。

频率译码

此时，编码器4必须通过设置I915为0或4设为脉冲-方向译码。通常设I915为4，因为当CHB4(方向)不被连接时，正电压将导致计数器开始计数。

电源

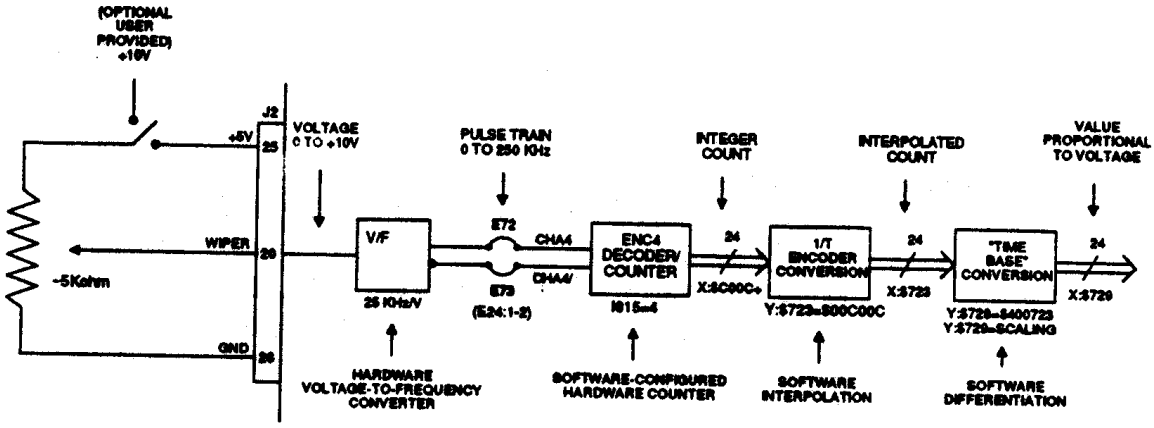
要使V/F工作，PMAC必须有参考于数字地的+/-12V电源。如果PMAC与总线连接，它通常来自总线接口的总线电源。在脱机运行时，该电源仍需从总线接口获得(PMAC-Lite为接线盒电源)，或者由跳线E85，E87和E88跳至模拟端，这样破坏了板上的光电隔离。

PMAC-Lite的特殊限制

因为PMAC-

Lite的WIPER输入有双极能力，它有一些特殊设置，如果使用双级模式，偏置电位计(R18)应被调节至零节点的最小端(可由JMACH!接口的CHA4检测)。如果不使用双极模式，R18必须被调至某一点以确认0V不会产生频率；跳线E73也应保持"OFF"以保证没有失真信号进入计数器的符号位。

对于PMAC-
Lite，脉冲和方向信号可分别从JMACH1接口的CHA4和CHB4针引出。它们被用来控制步进电机的驱动。DAC4输出可被连至WIPER输出，可提供伺服环需要的反馈和驱动器的命令信号。这使PMAC-Lite不需要特别的附件板就能驱动一个步进电机。




**使用PMAC的控制面板模拟输入(WIPER)（图4)**

软件处理

编码器转换表可提取每个伺服周期计数器的差别并放大它，提供一个与频率成正比的值，然后送给输入电压。这通常用于进给率超调(基于时间的控制)，但得到的值可用于任何用途。有关细节可见使PMAC与外部事件同步的时基控制部分。

**状态输出**

JPAN接口上有5个专用的低有效的输出，通常用于点亮LED灯。它们是BRLD/(请求缓冲区LED)，IPLD(到达位置LED)，EROR/(错误条件LED)，F1LD/(1st--警告--
以下错误LED)和F2LD/(当监视时钟出错时为真)。BRLD/，ERLD/和F2LD/为全局的状态线。IPLD/和F1LD/为坐标系特定的状态线。如果I2=0，它们将参照由面板选择的坐标系(FDPn/)。如果I2=1，它们将参照由主机选择的坐标系(&n)

如果I2=0当没有坐标系被选中(所有的FDPn/输入悬空或为高)，这些线可被作为普通用途输出，由Y：$FFC2的20-23位指定(PMAC-STD为Y：$FFFD的0-3位)。

## 显示口输出(JDISP口)

JDISP接口(J1)允许附件12或附件12A连接的液晶显示，或附件12C的真空荧光显示。使用DISPLAY命令可在运动或PLC程序的运行中在这些显示器上显示文本和变量值。

# §3—4 设置一个电机：

一个电机，对于PMAC来说就是一个有反馈、输出、标志位，和潜在的一个主控单元。你可以通过给它分配这些属性并且激活它来设置一个电机。这是通过使用I—变量（初始化变量）来完成的。位置信息是通过一个称作编码器交换表的结构进行预处理的，解释如下:

## 定义电机：

**电机I—变量：**

设置几个I—变量可为PMAC“定义”电机。也就是说，它们告诉PMAC从哪儿得到输入，在哪儿给出输出。通过使所有的这些位置由变量的值设置，PMAC在建立系统时将提供极强的灵活性。每一个电机都有一套相同的I—变量。I—变量序号的百位的数字对应着电机的序号。为了更方便地参考一个电机I—变量，我们用字母x代替百位数字，x代表了这一时候我们想要处理的任何一个电机(例如，Ix03能够代表I103,I203,I303,等等,直到I803)。

这些变量的默认值提供了大多数用户想要使用的设置，因此他们并不需要改变这些设置。但是，如果一个不同于默认值的设置是需要的，那么改变一两个变量也是一件简单的事情。

**激活电机：**

电机x的变量Ix00控制着PMAC是否做给这台电机的运算。如果你将要使用电机x，那么你需要设置Ix00为1（被激活的）。如果你根本就不会用到电机x,那么你需要将Ix00设置为0（不被激活的），这样PMAC将不会浪费处理时间来为一台现在不在的电机做计算。

一台被激活的电机既可以使它有效，也可以使它无效；激活仅仅意味着PMAC在注意该电机上发生着的事。

**PMAC给电机换向吗？**

实际上所有的电机在一定程度上都需要换向—
一个重要的例外是音圈电机。在这儿，重要的问题是PMAC需不需要做换向的工作。如果换向已在电机（例如换向式电动机）或放大器的内部被做了，那么PMAC就不需要再做换向工作，并且Ix01必须被设置为0。如果是这样的话，对电机只需要一个模拟输出，并且不用理会换向I—变量（Ix70~Ix83）的设置如何。

如果PMAC将要给电机执行换向，Ix01必须被设置为1。既然这样,电机将要求两个模拟输出，并且Ix70~Ix83必须被设置以便电机能正确的换向。参见后面的《设置PMAC换向》一章。

**地址I—变量：**

每一个电机都有一些“地址I—变量”。这些指针变量包括PMAC能自动读写数据的PMAC存储器和寄存器I/O空间的地址。这些变量包括Ix02,Ix03,Ix04,Ix05,Ix10,Ix25,Ix81,和Ix83。因为PMAC有一条16位的地址总线，所以它通过16位（4个十六进制数）来指定一个地址。然而，地址I—变量是24位的值，因此高八位能够被用来指定为使用特定的寄存器的变更模式。如果所有的高位都为零，寄存器是按照默认模式来用。可以通过参考《软件参考》中的单个I—变量的描述，来了解这些I—变量的每一个的变更用法模式的详细情况。对于初级用户来说，需要指定输入和输出的地址可能是一件比较麻烦的事。但是，为了基本的应用，大多数用户能够使用那些切合实际的默认值（电机n使用DACn,编码器n,和标志n），而随意分配输入和输出的能力在一个更加复杂的多的应用当中提供了前所未有的柔性。

**十六进制和十进制形式：**

如果I9是1或0，PMAC将以十进制数报告地址I—变量的值。如果I9的值是2（默认）或3，PMAC将以十六进制数报告这些值。通常以十六进制数时比较容易被解释，特别是当变更模式被使用时。因为你能直接看到地址位上的数值本身。例如，设置I102为
$C003(49155)指定了＃1电机DAC1的使用，命令输出处于正常的双极性模式下。设置第16位为1告诉PMAC在单极性模式（大小和方向）下使用该寄存器。用十六进制格式，PMAC将汇报此值为$1C003

，所以地址仍然是明显的，但是用十进制模式，PMAC将汇报此值为114691，使得该地址变得完全不明白了。

ADDRESS I一变量

- 低16位（4个十六进制数）指定地址
- 当高8位是零时，地址被用在正常模式下
- 当一位或更多位为1时，地址被用在不同模式下
- I9＝2或3：PMAC以十六进制报告地址变量值

| 十六进制 | 0 | | | | 1 | | | | C | | | | 0 | | | | 0 | | | | 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 二进制 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

(高8位,即表中的01是方式;低4位,即表中的C003为地址。)

**选择输出：**

变量Ix02确定在每一周期电机x的给定输出放在哪一个寄存器里（或是一对寄存器，如果PMAC换向的话）。Ix02的值就是该寄存器的地址。这几乎总是一个数模转换（DAC）寄存器。Ix02的默认值是该寄存器的地址（例如，缺省条件下电机1用DAC1）。

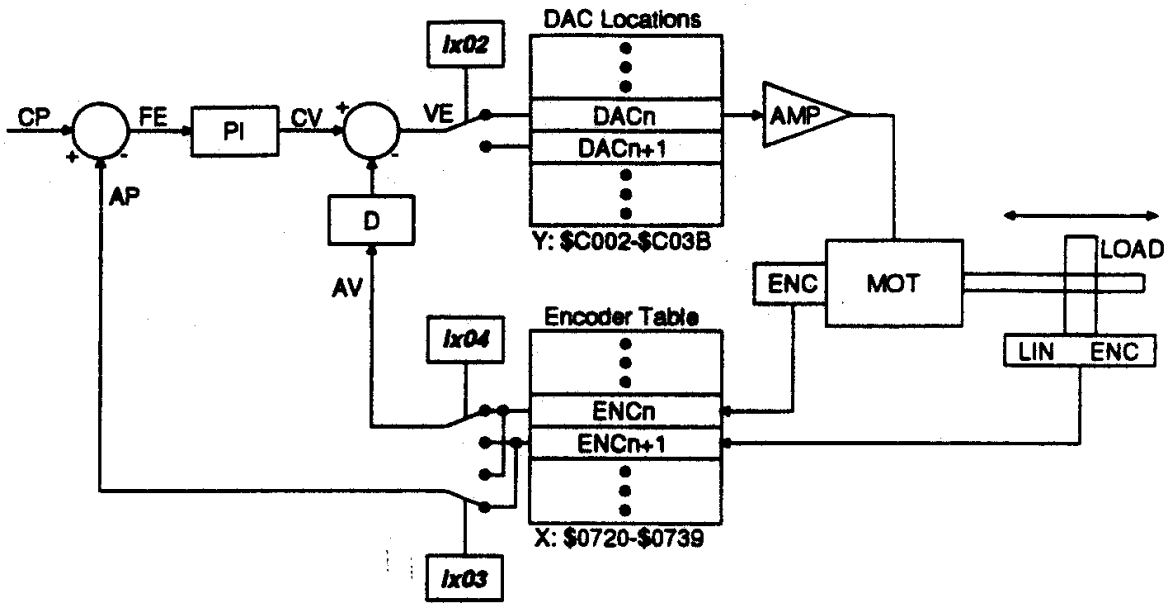
脉冲和方向输出:

PMAC能够命令接受脉冲和方向输入的电机－步进电机和步进装置的伺服电机。来自PMAC的模拟输出通过在选件板2附件8D上的伏频转换器转换成一系列的脉冲链。PMAC的输出必须通过将Ix02的第16位设置为1而被设置成方向和脉冲模式。脉冲链可以为形成一个模拟的伺服环而被反馈给PMAC，否则一个实际的编码器将被使用。

（PAGE 3－39 图）

I×02　DAC输出地址
I×03　位置环反馈地址
I×04　速度环反馈地址

**选择位置环反馈：**




变量Ix03决定了电机x从哪一个寄存器获得它的实际位置的信息，从而在每一次伺服周期闭上位置环。Ix03的值就是该寄存器的地址。在编码器转换表中它是一个寄存器地址。有来自反馈元件的处理过的信息，通过编码器转换表的默认设置，Ix03的默认值就是来自编码器x的处理过数据的寄存器地址（例如，电机3在缺省条件下使用编码器3）。

**选择速度环反馈：**

变量Ix04的值决定了电机x从哪一个寄存器获得它的实际位置的信息，从而在每一次伺服周期闭上速度环。Ix04的值就是该寄存器的地址。在编码器转换表中的它是一个寄存器地址。有来自反馈元件的处理过的信息，通过编码器转换表的默认设置，Ix04的默认值就是来自编码器x的处理过数据的寄存器地址（例如，电机3在缺省条件下使用编码器3）。

双反馈系统：

在大多数的系统中，这个寄存器和用来闭上位置环的寄存器是同样的寄存器，这就意味着Ix03等于Ix04
。然而，双反馈的概念在现在的运动系统中正逐步变得日益广泛。在这样的一个系统中，电机和负载都有位置传感器。

精度性和稳定性：
一个在负载上的传感器（通常是一线性标尺）比一个在电机上的传感器提供了更为精确的位置测量，因为它的精度性不受在电机负载联接中缺陷的影响。但是，负载上的传感器也使轴变得更加不稳定,因为这些联接缺陷(典型的有柔性和间隙），现在都处在反馈环内部了。在电机上的传感器，可能会精度降低，但却提供了更好的稳定性，因为那些缺陷都不在反馈环内部。

在许多情况下，通过在电机和负载上都使用传感器从而同时获得高精度和稳定性是可能的。在一个PMAC系统中，仅仅只要通过负载编码器闭上位置环（获得准确性），用Ix03指向该编码器，同时通过电机编码器合上速度环（获得稳定性），用Ix04指向该编码器。

注意：当使用双重反馈时，由Ix25（见后）指定的电机标志应该和位置环编码器有一样的序号。否则，为引导的硬件位置捕获功能将不会工作，而只能用低精度的软件位置捕获功能。例如，如果速度环编码器是ENC1（Ix04＝$0720）并且位置环编码器是ENC2(Ix03=$0721)，那么为了使用硬件位置捕获功能电机标志必须为标志2（Ix25=$C004）。如果标志不是2，那么只能通过给Ix03的第16位设置为1（例如，Ix03＝$10721）设置软件位置捕获功能。

**选择主位置资源：**

变量Ix05决定了电机x从哪一个寄存器获得它的主位置的信息。Ix05的值就是该寄存器的值。这是含有来自位置传感器的处理数据的编码器转换表中的一个寄存器。通过编码器转换表的默认设置，Ix05的缺省值就是来自编码器2的处理数据的寄存器地址（例如，所有电机将编码器2作为主位置）。这一设置允许一个单独的主位置编码器通过控制面板端口（J2）被引入，并且如果电机的随动功能有效，可使任何一台电机跟随该主位置编码器。

主位置是给PMAC的位置随动功能的数据的来源（通常被称为电子齿轮传动）。这一主题将在《使PMAC与外部事件同步》一章中详细叙述。

**选择标志寄存器：**
变量Ix25决定了电机x
将哪一个寄存器用作它的标志输入(极限、回零标志、放大器出错和检索通道）和输出（放大器使能/方向）。Ix25的值就是该寄存器的地址。这在DSPGATE
IC中总是一个控制/状态寄存器。Ix25的缺省值是给编码器 x
的控制/状态寄存器的寄存器地址（例如，电机4使用+LIM4、－LIM4、HMFL4、FAULT4和AENA4/DIR4）。为了使用精确的硬件位置捕获功能，标志设置的数值必须和Ix03指定的位置环编码器的数值相匹配。

**选择启动模式：**
变量Ix80决定了在启动/重置周期结束时电机是被使能还是被禁止。如果Ix80
是1，处于闭环、零速度状态，且在此时命令位置设置等于实际位置，电机将在启动/重置周期结束时自动使能。如果PMAC换向电机要求相位搜索，它也将被自动完成。如果Ix80是0，则电机将被禁止；需要一个命令来使电机重新使能：对于一台PMAC换向的电机必须用$命令；对于一台不是由PMAC换向的电机，命令$或J/都是可以的，或者命令A给在坐标系中的所有电机,或者〈CRTR－A〉命令给PMAC的所有电机。

## 位置传感器的类型：

PMAC可以不需要任何附件而接受增量编码反馈。使用合适的附件，它也能够接受旋转变压器、绝对编码器、模拟量，或者磁致伸缩线性位移传感器等的反馈。这些性能将在下面进行更加深入细致的解释。

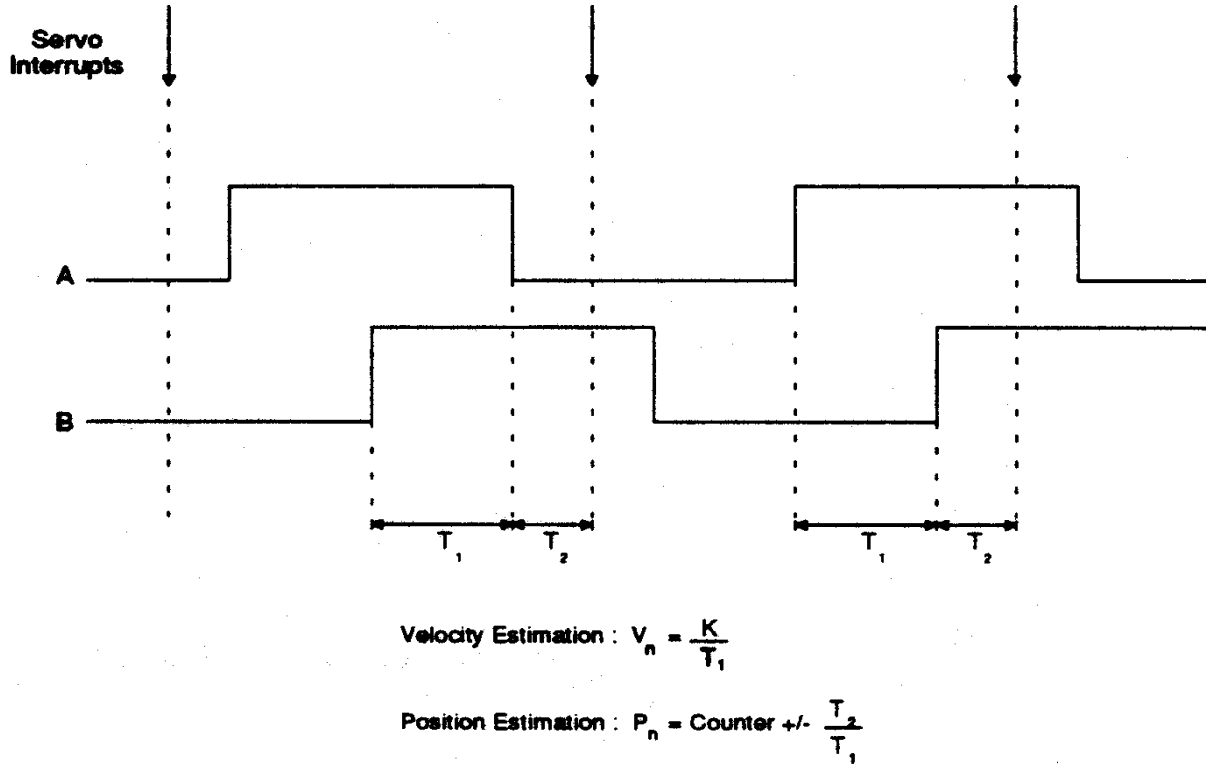
**正交编码器反馈：**

软件设置译码选择，PMAC通过x1，x2
或x4译码（脉冲和方向译码也是可能的）能够将正交编码信号作为位置反馈来接收。编码器I一变量0（I900，I905，等等）决定了每一个编码器的译码方式和方向。24位的硬件计数器被软件扩展到36位（可计数64亿）。软件参数（Ix27）允许位置按用户指定的值滚改：这对于旋转轴特别有用。在DSP GATE IC中任何没有用到的编码计数器可能被用作一个硬件计时器（参看I900的描述）。

1/T 子脉冲插补：

在PMAC上有两种可选的方式通过增量反馈来完成子脉冲分辨，第一种是1/T译码。每一个编码器通道都有两个计时寄存器与它相关联。第一个寄存器保持在前两次编码阶跃之间的时间。速度值就是这个时间的反比 -----这是非常准确的计算，特别是在低速时。

**（PAGE 3－41 PMAC 1/T EXTENSION图）**

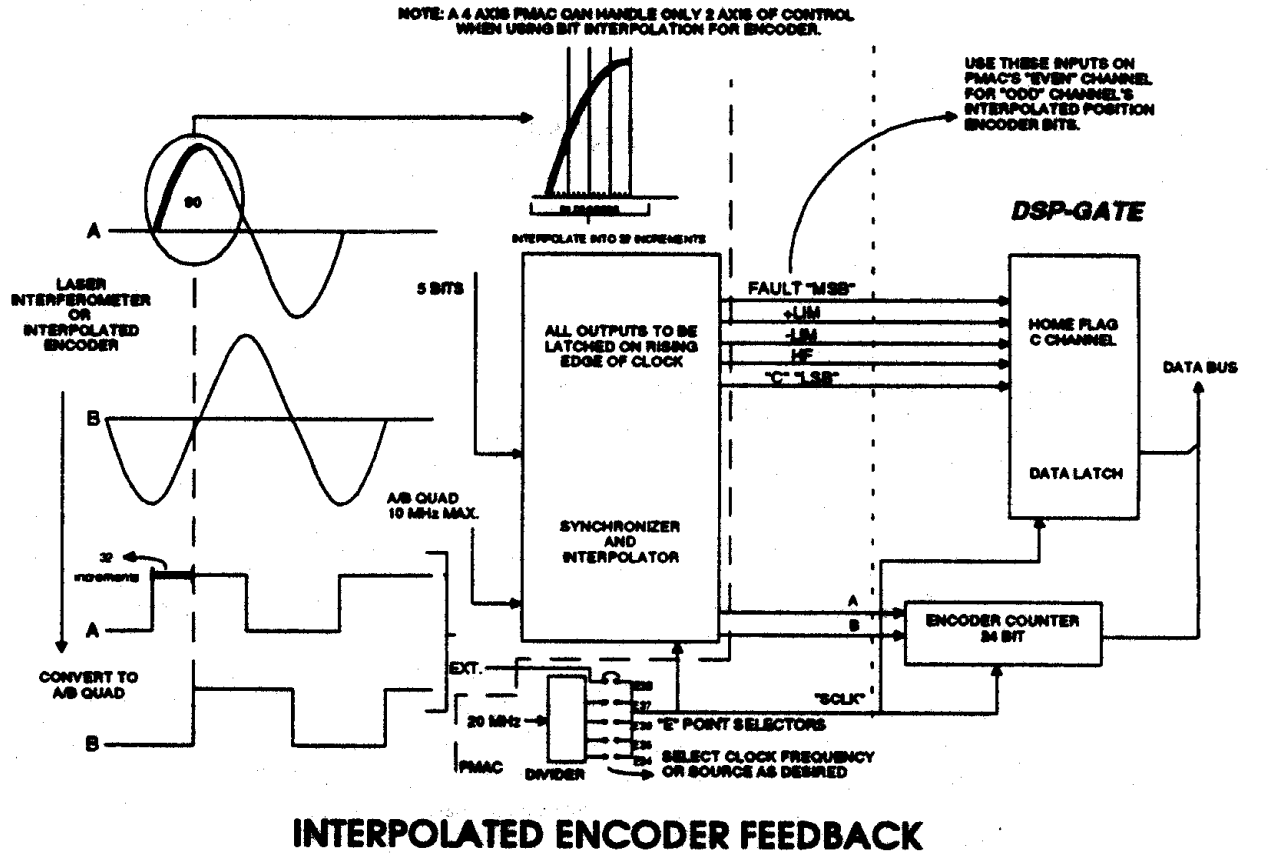



第二个计时寄存器保持从前一次阶跃时的时间。将第二个计时寄存器的值被第一个计时寄存器的值除（见图），得到的是自前一次阶跃以来经过的距离。这种插值给低速运动提供了运行的平滑性,但它没有提供静态的准确插值。1/T译码要求$00转换格式（见下）。

并行子脉冲插补：

插值的第二种方式允许PMAC读入5位并行的小数信息来补充整型正交计数。通常，这个信息是将编码器或干涉仪的模拟“正弦/余弦”正交信号经过模数转换电路而得到的。产生数字正交和并行小数位的电路必须是PMAC外部的电路，可以是用户提供的电路，也可以用***Delta Tau***的附件8D选件8模拟编码插补板。PMAC将给正交计数器和并行输入提供同步锁定，以确保数据的同步性。这种插值方式不象1/T方式，它给运动状态和静态都提供了准确的插值。假如A/D转换电路在高速时不能提供准确的插值，它可能在1/T方式和并行方式之间进行动态切换。

**（PAGE 3－42 INTERPOLATED ENCODER FEEDBACK图）**




**INTERPOLATED ENCODER FEEDBACK**

注意：当编码器进行位插补使用时4轴PMAC只能用作2轴控制

这种类型的子脉冲扩展只可奇数编码器上做。这5位是同下一个较高编号的编码器相关联的5个输入：FAULT（最高位）、＋LIM、－LIM、HMFL和CHC（最低位）。并行子脉冲扩展要求$80转换格式（见下面）。

硬件改变：

为了正确地实施这种类型的反馈，在硬件和软件上的几处设置的默认值需要加以改变。首先，给标志位用作插位的光电隔离器必须被去掉，代以硬接线的支路，以便信号不会被延迟。这将会把标志电路和数字电路联系到一起。既然如此，通常是用数字电路的＋12V来给保持“真”的标志供电（通过移动E90跳线），并且将低端接到数字地（GND）；这样就能在数字电路和模拟电路之间保持隔离。

软件改变：

在软件上，重要的是要使在编码器输入上的数字延迟滤波器失效，这些编码器输入既包括主编码器的，也包括匹配标志位编码器的（即使这个编码器并不用在该功能）。这是通过将所有这些编码器的编码器I－变量（I901、I906、I911，等等）都设置为1来完成的。

**并行位置反馈：**

PMAC能够通过它的I/O扩展板（附件14D/V）接收并行位置反馈（例如，来自一个绝对编码器、激光干涉仪，或者已经转变的模拟信号）。每块附件14D/V板有48位的输入，因此它可以同两个并行反馈设备连接起来，其中每一片最多可有24位，或者同一个多于24位的反馈设备连接。一片PMAC最多可以和6片附件14D板连接。并行反馈设备必须直接提供二进制数据，而不是格雷码。如果并行反馈设备浮动的话，PMAC的内部寄存器会自动延长计数（除非PMAC寄存器也被设置为浮动的）。
并行位置反馈需要$20, $30, $60或$70格式中的其中之一(见下面)

并行绝对反馈：

当用一个绝对编码器作为反馈设备时，给PMAC的数据是处于并行格式。所有的线必须被一起接入；高字低字选择电路是不被允许的。由于设备的绝对特征，启动/重置位置并不自动为零。对于这种设备，PMAC可以用Ix10参数来读绝对的启动/重置位置，其宽度最大可到48位。如果Ix10被设置为0，绝对启动/重置位置读入功能无效，并且不论传感器的设置如何，启动/重置位置被置为零，顺序位置读入相应增加地参考该零位置。要获得更多的信息，参考《绝对启动位置》一章，还有《软件参考》上关于Ix10的描述。

传感器浮动：

如果轴的总的移动超出了绝对设备的范围，PMAC将自动地在软件上扩展位置以处理浮动。在这种情况下，设备应该被认为是并行增量设备（参见下一部分）。为了换向的目的，一个设备可以被认为是绝对的，因此启动相位搜索是不必要的（如果PMAC在做换向时，设置Ix81＝1），但对于总的机器位置功能仍是增量式的。在大多数系统中，单圈旋转变压器和绝对编码器有这项功能。参考《设置PMAC换向》中的《参考绝对传感器调整相位》部分，可以获得关于这类设置的细节问题。

用这种反馈设备时重要的是在启动/重置之后，在第一个编程移动之前,执行PMATCH（位置匹配）功能。通常这是通过设置I14等于1来自动完成的。如果没有做这一步，PMAC将假定一个零起点代替真实位置来为电机计算第一步移动，而这将导致不可预料的后果。

并行增量反馈：

一台象激光干涉仪这样的设备通常提供的是并行反馈数据，但这种设备根本上仍是增量式的，所以在启动/重置时它并不知道自己所处的位置。PMAC可以设置成象接收一台绝对并行设备的反馈那样接收这种类型的反馈，但是用户必须认识到回零过程是必须的。对于这类设备，让Ix10等于零，以便PMAC不用执行绝对启动/重置位置读功能。

可是，既然位置信息不是绝对的，既然PMAC有能力在软件上扩展位置范围，所以用不着把设备上所有的线都同PMAC连接起来。这可以节约在连接设备上的花费。所有需要的只是足够的线（从LSB开始的），因此在一个简单的伺服环中会有一半的线是用不上的。例如，一台典型的干涉仪的界面有32位的并行的数据。即使用到的伺服周期是1毫秒，这对于PMAC来说是较慢的，只要最大速度低于32,768计数/毫秒，或32,768,000计数/秒，将低16位的线同PMAC连接起来就足够了。

*回零软件捕获：*

使用这种设备进行位置反馈的电机必须在回零搜寻移动中执行一软件位置捕获，代替增量编码器执行的硬件位置捕获。这是通过给电机设置变量Ix03的第16位为1（如果Ix03是$0720,它将变成$10720）来完成的。软件捕获的延迟可能会有几毫秒；为了得到高的回零精度，回零搜索移动的速度可能需要被限制。

**线性位移传感器反馈：**

PMAC可以通过它的附件29接口板从线性位移传感器接收反馈。（这类设备的最著名的品牌是MTS公司的“Temposonics”。）这类设备工作起来很象声纳，测量的是“回声”返回之前的时间。附件29使用它的在载DSP－GATE ICs的时间寄存器来记录这个信息；时间越大，距离越长。

对于PMAC本身，这种反馈看起来更象一个绝对编码器。数据的来源是合适的计时寄存器，而不是附件14D
I/O寄存器。$20或$30转换格式将被使用，并且数据将被发现在编码器9中，通过测量前两个脉冲的时间间隔的编码器16（Y:$C020,Y:$C024,...Y:$C03C）。关于在并行位置反馈入口下建立这种类型反馈的软件设置的介绍见下。

对于这类设备，PMAC能够用Ix10参数来读绝对启动/重置位置，最多可达24位宽。如果Ix10是0，绝对启动/重置位置读功能失效，并且不论传感器的设置如何，启动/重置位置都被设为0，顺序的位置将参考该零位置相应增加地被读入。参考下面的*绝对启动位置，*《软件参考》上关于Ix10的描述，和附件29MLDT接口手册，可以获得更多的细节。

用这种反馈设备时重要的是在启动/重置之后，在第一个编程移动之前,执行PMATCH（位置匹配）功

能。通常这是通过设置I14等于1来自动完成的。如果没有做这一步，PMAC将假定一个零起点代替真实位置来为电机计算第一步移动，而这将导致不可预料的后果。

**模拟位置反馈：**

如果模拟反馈是要求的，例如从一台电位计或一LVDT，PMAC通过它的模数转换器卡（附件23或附件28）之一能够接收高带宽的模拟反馈。任何被调制过的模拟位置信号，在它被送到PMAC系统中以便用一个DC电平代表一个固定的位置之前，都必须被解调。PMAC和它的附件板卡并不支持模拟位置反馈的软件扩展，因此浮动是不允许的。通过附件23或附件28的模拟反馈要求转换格式$10（见下）。如果模拟数据通过PMAC外部的电路或PMAC的附件而被转换成数字格式，那么它就可以象一个并行数据字一样被反馈给PMAC，而PMAC将象对待一个绝对编码器那样对待它。

对于这类设备，PMAC能用Ix10参数来读绝对启动/重置位置，最多可到16位宽。如果Ix10被设为0，绝对启动/重置读功能无效，并且不论传感器的设置如何启动/重置位置都被设为零，而顺序位置将参考该零位置相应增加的被读入。参考下面的绝对启动位置，《软件参考》上关于Ix10的描述，和附件28或附件36 A/D转换器手册，可以获得更多的细节。

启动1.15版的固件，它有可能为伺服环反馈而使用在一个单独的附件36上的A/D转换器。I60和I61用来指定在定相中断时将被自动拷贝进RAM的附件36ADC寄存器的地址和序号。伺服环反馈功能将从这些RAM寄存器中读入数据，并将这些数据看作12位的并行位置反馈（见以上部分）。对于这类设备，PMAC能用Ix10参数来读绝对启动/重置位置，最多可到12位宽。如果Ix10被设为0，绝对启动/重置读功能无效，并且不论传感器的设置如何启动/重置位置都被设为零，而顺序位置将参考该零位置相应增加的被读入。

用这种反馈设备时重要的是在启动/重置之后，在第一个编程移动之前,执行PMATCH（位置匹配）功能。通常这是通过设置I14等于1来自动完成的。如果没有做这一步，PMAC将假定一个零起点代替真实位置来为电机计算第一步移动，而这将导致不可预料的后果。

**旋转变压器反馈：**

PMAC可以通过它的附件8D选项7R/D转换来接收旋转变压器反馈。这种能够按双通道和四通道设置被购买的卡，通过两种方法来接收旋转数据：一是接收一绝对字（在旋转变压器的一旋转中），二是接收一正交信号。在每一个旋转变压器电子周期两者都是4096计数。一个电子周期是一个极对，因此一个四极旋转变压器在每一机械旋转中有2个电子周期，或者是8192计数。

绝对字的读入是如此之慢，以致不能在每个伺服周期都得以执行，因此在典型使用中，这只是在启动/重置时被执行。正在进行的位置是从正交编码计数器得到的，该计数器是和转变后的正交信号相连接的。对于PMAC的软件来说，这看起来就象一个真的正交编码器。

对于这类设备，PMAC能够用Ix10、I9x和I8x参数来读单个旋转变压器的绝对启动/重置位置，或2到3个旋转变压器构成的系统的绝对启动/重置位置。如果Ix10被设为0，绝对启动/重置读功能无效，并且不论传感器的设置如何启动/重置位置都被设为零，而顺序位置将参考该零位置相应增加的被读入。如果但单圈旋转变压器的绝对位置将只是被用来防止启动相位搜索，Ix81被用来指定绝对启动相位位置读入。参考下面的*绝对启动位置，*《软件参考》上关于Ix10的描述，和附件8D选项7 R/D转换器手册，可以获得更多的细节。

## 绝对启动位置：

在一些应用中，在启动或重置后做回零搜索移动是不允许的或不要求的。在这些应用中，一个绝对位置传感器被用来使得在启动/重置时的真实位置能立即被知道，而不需要移动到一个已知的回零位置。为该目的而使用的典型传感器是绝对编码器和旋转变压器。PMAC从这两种传感器都能支持绝对启动位置读入。

**绝对位置范围：**

要获得充分的绝对启动位置而不需要回零搜索移动，位置传感器必须在轴的整个移动范围都是绝对的。如果移动包含电机的多种旋转，而传感器只在电机的一种转动上是绝对的，那么回零搜索将仍是要求的。尽管这样的一个传感器能够被用来给非换向电机的启动相位，为了这些目的，该传感器应该被

看作一个增量式的传感器。参考本手册的换向部分中的《参考绝对传感器定相》，和Ix75和Ix81的描述，可获得关于启动定相的更多的信息。

如果你想要系统的启动绝对位置，并且不想要该位置有任何的翻转，则绝对传感器的翻转点必须在移动范围之外。如果你将绝对位置信息看作是无符号量，翻转点就是传感器的零位置。如果你将绝对位置信息看作是符号量，那么翻转点间距就是两零位置的一半。

PMAC上的每一个电机都有变量Ix10,I9x,和I8x来支持绝对启动位置读入。Ix10指定绝对传感器在PMAC里的寄存器地址，以及读它的方式。
I9x和I8x是在一个啮合的旋转变压器系统被用来决定启动位置时，用来指定第二个和第三个旋转变压器。

**并行数据位置：**

Ix10能够指定两种类型的反馈。如果绝对位置数据被作为一个并行字送给PMAC，通常是通过附件14 I/O
卡，那么在Ix10的低16位指定的地址就是包含该数据的‘Y’PMAC寄存器的地址（例如，$FFD1）。Ix10的高8位指定了在该寄存器（以及更高位的寄存器）所用位的序号，其中最高有效位指定了数据是被作为符号数值来对待还是作为无符号数值来对待，第二重要的位（22位）指定了数据来自X－寄存器还是来自Y－寄存器。

在这种模式下给这些位的序号的有效值是8到48（$08到$30）。如果最高有效位（值为$80）被设为1，这给定了$88到$B0的高8位字节一个范围，将使得从传感器读入的范围在－($2^{N-1}$)到+$2^{N+1}$-1之间的数将被作为一个符号数对待，其中N是位数。如果Ix10的最高位是零，在范围0到$2^{N-1}$内的传感器的值将被作为一个无符号数来对待。实质上所有的并行I/O资源都与PMAC里的Y－寄存器映射，因此，第22位（X/Y寄存器指定位）通常总是设为0，来指定一个Y数据资源。

例子：

作为一个示例，要从在第一块附件14
（Y:$FFD1）的端口B上的22位绝对编码器中读入一个无符号数，Ix10应该被设置为$16FFD1(16 hex就是22
decimal)；如果要作为符号数被读入，Ix10应该被设置成为$96FFD1。对于一个32位的绝对传感器，它的低24位在第一片附件14(Y:$FFD0)的端口A，而高8位在端口B（Y:$FFD1），
要从它读入一个无符号数，Ix10将被设置为$20FFD0(20
hex就是32decimal)；如要从它读入一个符号数，Ix10将被设置成$A0FFD0。

注意：有着并行数据输出的传感器不必是一个绝对传感器。激光干涉仪通常以并行方式提供其位置数据，但它却是增量传感器，用它给位置反馈的电机仍然需要引导。对于任何增量传感器，Ix10应该被设置为0（没有绝对启动读入）。

**旋转变压器位置：**

另外一种能够用Ix10指定的启动位置数据是串行数据，它是通过“指轮”多路拨码开关端口从附件8D选项7旋转变压器到数字（R/D）转换器板卡来的。在这种格式下，Ix10的低16位指定该端口的多路拨码开关的地址，其值是十进制数0到256，同用DIP开关在板卡上设置的地址相配；值256（$0100）用来指定多路拨码开关地址0，因为PMAC将值0解释为没有启动位置读入。

Ix10的高8位包含了一个从0到7的值来指定在该多路拨码开关内的特定的R/D转换器的位置-----在每一个多路拨码开关地址有8个。同样，最高有效位(值$80)指定了该位置是作为一个符号数来对待还是作为一个无符号数来对待。如果最高位被设为0，从解算器读入的值将被作为无符号数对待，其范围是0到4095；如果最高位被设为1，从旋转变压器读入的值将被作为符号数对待，其范围是－2048到2047。

例如，用在多路拨码开关地址2的位置3的R/D转换器，将读入的值作为一无符号数对待，Ix10将被设置为$030002；将读入的值作为一符号数对待，Ix10将被设置为$830002；用在多路拨码开关地址0的位置0的R/D转换器，将读入的值作为一无符号数对待，Ix10将被设置为$000100；将读入的值作为一

符号数对待，Ix10被设置为$800100。

带传动比的旋转变压器：

典型的，在电机背部的单个的旋转变压器并不足以决定启动位置。如果要求确切的启动位置信息，将用到一套啮合的旋转变压器，每一个都减速到一个较低的速度，因此旋转变压器的辨率比在啮合之前要粗糙一些。第一片旋转变压器通常是在电机的背部随着电机一起转动，它转的最快，它的分辨率最高。它被称为精细旋转变压器或第一旋转变压器。在啮合链上的最后的旋转变压器转得最慢，分辨率也最低。实际上在一个电子周期，它只转不超过一圈，因此它被称为粗糙旋转变压器。

理论上，任何数目的啮合的旋转变压器都能被用来确认启动位置。实际上，大多数系统都只用两个或三个旋转变压器。在一个两旋转变压器系统中，它们分别被称为精细旋转变压器和粗糙旋转变压器。在一个三旋转变压器系统中，它们分别被称为精细的、中等的和粗糙的旋转变压器。既然PMAC能够同两旋转变压器系统或三系统连接，名词“精细旋转变压器”将用在直接和电机轴相连的旋转变压器上。和第一器啮合的旋转变压器------
在两旋转变压器系统中被称为粗糙旋转变压器，在三旋转变压器系统中被称为中等的旋转变压器------将被称为“第二旋转变压器”。再接下去的旋转变压器将被称为“第三旋转变压器”。

如果一套啮合的旋转变压器被PMAC用来决定启动位置，变量I9x，可能还有变量I8x必须从它们的默认值（0）加以改变。第二旋转变压器将同R/D转换器连接起来，该转换器的多路拨码开关地址和与精细旋转变压器相连的转换器的多路拨码开关地址是一样的，但其在该地址中的位置比和精细旋转变压器相连的转换器的位置要高。I9x代表了精细旋转变压器和第二旋转变压器之间的啮合比。它必须是个整数。如果第二旋转变压器从精细旋转变压器减速的比率是16：1，则I9x应该被设为16。I9x的值为0告诉PMAC没有次级旋转变压器。

如果还有第三旋转变压器,I8x将被用来指定第二旋转变压器和第三旋转变压器之间的啮合比。第三旋转变压器必须和在同一多路拨码开关地址上比第二旋转变压器位置高的R/D转换器连接起来。啮合比必须是一个整数。如果两者之间的啮合比是36：1，I8x将被设置为36。I8x的值为0，告诉PMAC没有第三旋转变压器。

在一个啮合旋转变压器系统中，Ix10的最高有效位决定了该组合量是被作为一个无符号数值来对待，还是作为一个符号数值来对待。如果被作为无符号数来对待，零位置将被设置在越过移动的负端的位置。以便启动位置不会在零的负边。如果被作为符号数值对待，零位置将被设置在轴的最大可用移动范围的中间。

在任何一台使用一个旋转变压器或一套旋转变压器装置作为位置反馈的电机上,所有在初始启动读入之后，在伺服环中被用到的位置信息，都来自给精细旋转变压器的R/D转换器的正交信号合成，通过PMAC中的一个编码计数器来计数。其软件设置（Ix03,Ix04,转换表）和一个实际的正交编码器是一样的。不需要用到第二或第三旋转变压器生成的正交信号给电机。

**轴偏移：**

如果绝对传感器的零位置不是你所要求的，那么你为了编程的轴的零位置又是怎样的呢？由于传感器非常难以做到准确的线性化，并且由于如果位置信息被作为一个无符号数值来对待时，传感器的零位置必定在移动范围的外面，所以这是一种十分普遍的情况。

传感器（电机）零和轴零之间的不同可以通过给轴的轴定义声明中的轴偏置参数来设置。带有计数位的该参数在传感器位置是零时将包括轴的位置。它和在同一轴定义声明中的比例系数是相互独立的。

例如，将1#电机分配给每单元10000计数的X轴，如果你想要轴的零位置处在绝对传感器读了50247计数的点上，那么轴的位置在传感器读入为零时将处在-
50247计数的点上，因此轴定义声明将写作：

#1->10000X –50247 。

**编码器偏移：**

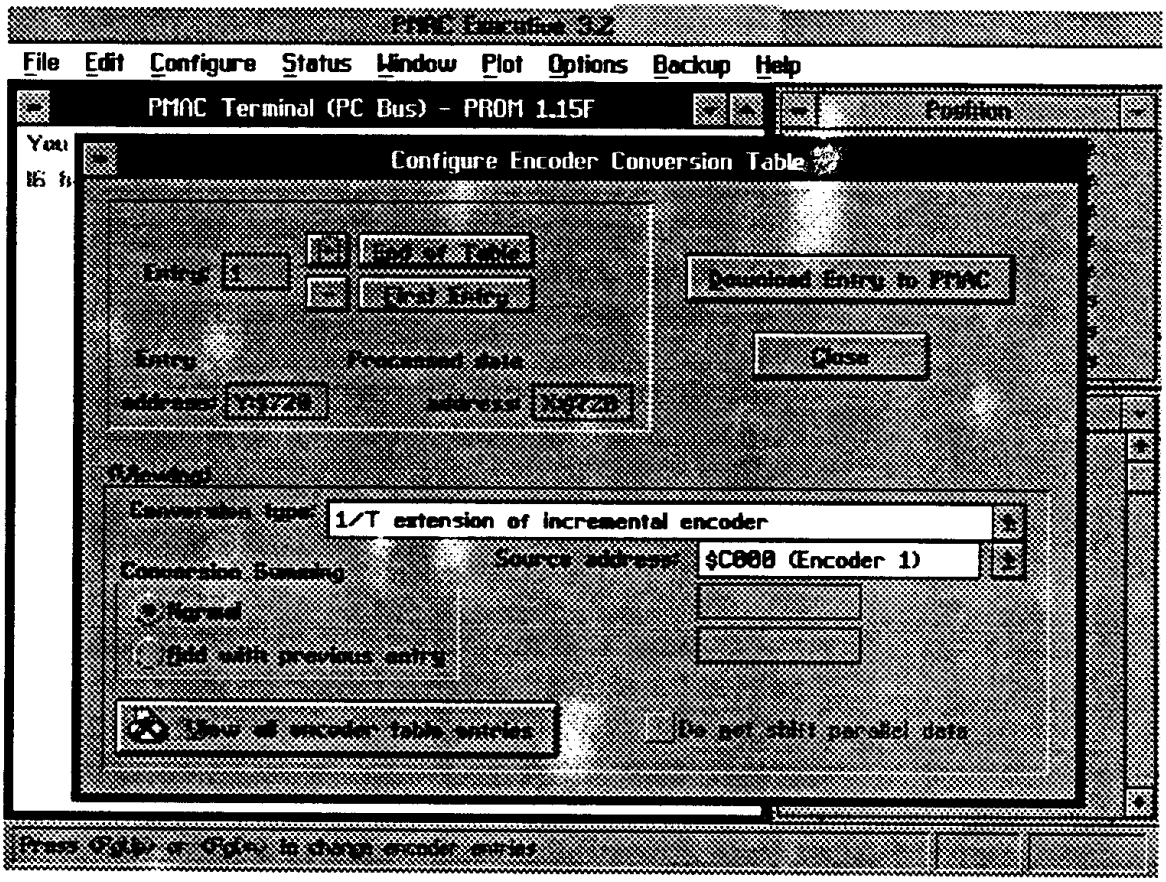
如果你为绝对启动位置信息而用旋转变压器，那么顺序位置信息将来自在启动时设置为零的编码计数器。在大多数用途，这对于用户来说都是明白的。但是当你希望直接使用编码器寄存器，通常是位置

捕获和比较功能，那么你必须明白编码计数器零位置和电机（旋转变压器）零位置之间的不同。

该值保存在电机编码器位置偏置寄存器[Y:$0815(电机1),Y:$08D5(电机2), 等等]中。关于该寄存器使用的一个示例参见《基本电机移动》中的《存储引导位置》部分。

PMAC用到一个多级的进程来和它的反馈，编码器转换表主位置信息，以及用以提供最大功率和柔性的外部时基资源一起来工作。对于大多数带有正交编码器的PMAC用户来说，这个进程实际上是很明白的，不必担心细节问题。然而某些用户将需要理解某些细节上的转换进程，以便为最优化系统而用到其它类型反馈，因此需要做必须的改变，或者执行特殊的功能。这一部分就是给那些用户的。

注意：给PC系列兼容机的PMAC执行程序有一个为转换表的特殊的编辑屏幕,使得观察和改变转换表都非常容易。这里的详细的介绍将会展示即使没有执行程序屏幕的帮助，怎样去观察和改变转换表。




## （PAGE 3－50 转换表界面图）

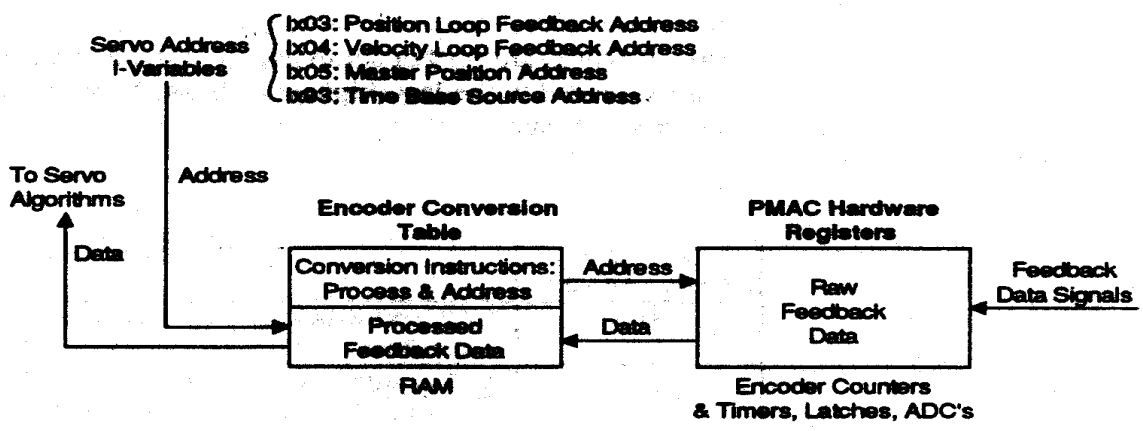
在位置和时基转换进程的第一步是和计时器，A/D寄存器，或者为了并行输入的附件板卡相联系的硬件编码计时器。这些都连续地工作，没有软件的直接行为干预（尽管它们也能够通过软件加以设置）。在这点之前，该进程是软件控制的。在每个伺服周期的开始，一个伺服中断信号将被送出以锁住所有的寄存器。

在这一点上，PMAC使用一个称为“编码器转换表”的软件结构来处理被锁寄存器里的信息。该表告诉PMAC处理哪一个寄存器，以及怎样去处理它们；它也保持中间进程数据。

PMAC**编码器转换表原理表图**

I×03：位置环反馈地址

I×04：速度环反馈地址
I×05：主位置地址
I×93：时基源地址




**转换表结构：**

编码器转换表有两“列”，一个是在处理器的X存储空间，另一个是在Y存储空间中。X列存储已转换的数据，而Y列保存源寄存器的地址。用户通过写Y列来建立该表，而PMAC在每一伺服周期用Y列数据来填写X列数据。

PMAC编码器转换表

| X存储器（结果） | | | Y存储器（设置） | |
|---|---|---|---|---|
| **1. 单线入口** | | | | |
| **Bits 5-23** | **Bits 0-4** | | **Bits 16-23** | **Bits 0-15** |
| **结果：整数** | **小数** | | **方式** | **源地址** |
| **2. 多线入口** | | | | |
| **Bits 5-23** | **Bits 0-4** | | **Bits 16-23** | **Bits 0-15** |
| **（中间结果）** | | | **方式** | **资源地址** |
| • • • • • • | | | **（转换系数）** | |
| **结果：整数** | **小数** | | • • • • • • | |

转换方式：

下面的表列出了可能的转换格式。要做一个转换，8位格式配上16位地址来填写转换表里的24位Y字。如果对于一个给定的转换类型，其多于一行，另外的Y字就是该转换的更深的建立参数。转换结果将放在最后（最高位地址）的X字中，别的X字将保存中间变量。

X＝0 给标准转换，无求和
X＝1 将上一入口和这一入口求和的转换
Y＝0 给标准转换，无求和
Y＝1 将上一入口和这一入口求和的转换
Y＝8 给一不带移位的转换，无求和
Y＝9 将上一入口和这一入口求和的不带移位的转换

| Y字 Bits 16-23 | 转换类型 | 行号 |
|---|---|---|
| $0x | 增量编码器的1/T扩展 | 1 |
| $1x | A/D寄存器-----不翻转 | 1 |
| $2y | 来自Y字未滤波的并行位置-------带翻转 | 2 |
| $3y | 来自Y字经滤波的并行位置-------带翻转 | 3 |
| $4x | 时基转换比例数字分化 | 2 |
| $50 | 积分模拟 | 2 |
| $6y | 来自X字未滤波的并行位置-------带翻转 | 2 |
| $7y | 来自X字经滤波的并行位置-------带浮动 | 3 |
| $8x | 带子计数扩展的增量编码器 | 1 |
| $90 | 被触发的时基(被冻结的) | 2 |
| $A0 | 被触发的时基(正在执行的) | 2 |
| $B0 | 被触发的时基(等待处理的) | 2 |
| $Cx | 无扩展的增量编码器 | 1 |
| $D0 | 指数滤波器 | 1 |
| $Ex | (保存给未来的使用) | 1 |
| $Fx | (保存给未来的使用) | |

入口值相加:

对于许多转换表入口值来说—那些在上表中带有x或y的第二位数的 ------ 将设置字的第16位设为1意味着转换的结果并不仅仅来自指定的资源。实际上，它是在表中的该入口值和其以上入口值的和。（如果传感器的极性或它们的计数器是相反的，这就提供传感器之间的差别。这对于多普勒类型的传感器是非常有用的，因为其参考波和移动频率被反馈给不同的计时器，一个递增计数，一个递减计数；把两个计时器的值相加就得到位置。）

注意：如果转换表在一行中有2个或更多的相加的入口值，只有第一个入口值将执行相加动作。别的入口将只会不相加的处理它们的资源数据。要将3个或更多的资源相加，必须在第二和第三资源入口之间使用一个中间的不相加的入口。这是通过用一个使用$68格式的入口去读第二入口的输出（第一个已被加的入口值）来完成的。该入口的值只要用标准技巧就可以被加到第三资源入口中去。

设置字示例：

```
WX:$720,$00C000          ;给编码器通道1的1/T入口
WX:$721,$01C004          ;给编码器通道2的1/T入口,和通道1
                         ;相加
WX:$722,$680721,$FFFFFF  ；为编码器通道1和2相加的
                         ；中间入口
WX:$724,00C008           ;给编码器通道3和中间入口相加的1/T入口
```

下面将讨论每一种类型的转换。

**增量编码器入口：**

增量编码器是用转换格式$0x,$8x,或$Cx中的一种来进行转换的。设置字的低16位指定了在X数据总线上的资源地址。对于增量编码器来说，资源地址必须是DSPGATE编码计时器中的一个，可从下表中选择：

| ENC1： | $C000 | ENC9: | $C020 |
|---|---|---|---|
| ENC2: | $C004 | ENC10: | $C024 |
| ENC3: | $C008 | ENC11: | $C028 |
| ENC4: | $C00C | ENC12: | $C02C |
| ENC5: | $C010 | ENC13: | $C030 |
| ENC6: | $C014 | ENC14: | $C034 |
| ENC7: | $C018 | ENC15: | $C038 |
| ENC8: | $C01C | ENC16: | $C03C |

给增量编码器转换的表的入口如下所示：

| X字<br>被转换的位置数据 | Y字源数据和数据处理 |
|---|---|
| Bits 0-4<br>小数位 | Bits 0-15<br>源数据的字地址 |
| Bits 5-23 | Bits 16-23 |

| | |
|---|---|
| 整数位 | 转换格式：<br>$00=1/T 插补<br>$80=并行位插补<br>$C0=无插补 |

源计数器已经反映了正交输入或脉冲一方向波形输入的译码方式，不论每周期是1，2，或4位计数。译码方式是通过给该编码器的编码器I一变量（I900、I905等等）决定的。在计数器里的一位是一个“计数”，不论它代表的是全波、半波或1/4波周期。

1/T插补：

大多数人都用1/T扩展转换方式（$0x）,该方式使用和每一个计数器都相联系的计数器来计算小数分辨率（见前面的说明）。给它的一个典型的设置字是$00C008,其提供了编码器3计数器的1/T扩展转换。

并行位插补：

那些设置使用并行子计数插补作为增量反馈的人将用$8x转换格式。在这里，资源地址必须是一个奇数编码器。给这种情况的典型设置是$80C010，它用到编码器6的标志提供给编码器5计数器并行扩展。

无插补：

对于没有任何子计数插补的转换，$Cx转换格式将被使用。给这种情况的典型设置字是$C0C00C，它给编码器4计数器提供了一个无插补的转换。

**附件28模数转换寄存器输入口：**

$1x转换格式从24位字的高16位采集数据。它还可以同在DSP一GATEs里的通过附件23（过时了）或附件28反馈的A/D转换寄存器联系起来工作。（当使用附件36 A/D转换卡时，将数据作为12位并行格式数据对待）。在Y存储空间中的资源地址如下所列：

| | | | |
|---|---|---|---|
| ADC1： | $C006 | ADC9： | $C026 |
| ADC2： | $C007 | ADC10： | $C027 |
| ADC3： | $C00E | ADC11： | $C02E |
| ADC4： | $C00F | ADC12： | $C02F |
| ADC5： | $C016 | ADC13： | $C036 |
| ADC6： | $C017 | ADC14： | $C037 |
| ADC7： | $C01E | ADC15： | $C03E |
| ADC8： | $C01F | ADC16： | $C03F |

给一个A/D寄存器的典型的设置字是$10C006，它提供了ADC1寄存器的转换。用A /D转换时，软件扩展没有被执行，因此翻转是不允许的。

结果被放在输入的X寄存器中，有19位整数（最高的3位仅是符号的扩展），和5位小数（小数位总是零）。

积分模拟：

使用转换表来对模拟输入和相当于模拟输入的积分，这是可能的。这是用转换格式$50来完成的，而不是用标准（不积分的）模拟转换$10。A /D源寄存器的地址就象格式$10那样来指定。将ADC1的输入积分的输入将是$50C006。

*移项：*

积分模拟格式要求第二个输入口来指定A/D的偏移。这是一个符号数，带有16位A/D转换器的最低位的1/256单元。例如，如果在5LSBits的16位ADC中有一个偏移，该条目将被设置为1280。如果不要求偏移，该条目将被置为零。该项可允许积分，甚至带一个模拟偏移。

*结果格式：*

积分的结果放在该入口第二行的X寄存器里，有19位整数和5位的小数（小数位一直是零）。因为输入数据有16位（24位字的高16位），在输入的最大范围，当每一个伺服周期积分动作被执行时，只有3位（8次）输入的扩展进入到积分寄存器。因此，不论使用该信息的什么用途，都必须至少在每8个伺服周期查看一次积分寄存器以处理潜在的位置翻转。对于使用该信息（主控制机和反馈）的自动伺服环是没有问题的，但对于后台任务来说就有问题了。

总结，对于一个积分模拟反馈，一个入口应该为：

| X字 | Y字 |
|---|---|
| 1．中间数据 | 1． 资源和处理：<br>Bits 0-15:<br>资源数据的Y地址<br>Bits 16-23:<br>=$50 |
| 2． 被转换的数据：<br>Bits 0-4:<br>小数位<br>Bits 5-23:<br>整数位 | 2.偏移项：<br>16位ADC的1/256位 |

积分的结果在启动/重置时将自动地置为零，并且在这之后积分功能将立即执行。用户在任何时候可向结果寄存器中写入任何值（通常是通过一个M一变量）；而确认此时任何事都不会因为该寄存器的值被改变而受到相反的影响，则是用户的责任。

*积分模拟的使用：*

这种格式有好几种可能的用法。第一种，一个模拟速度传感器（例如转速表）可以用来提供如同位置的信息给PMAC的伺服环（记住速度环希望位置信息）。例如，假定一个连着带转速表的电机的轴，一个线性编码器，和一个为得到高响应的电流环放大器。为位置环和速度环都用线性比例是很难获得稳定的，因为没有关于电机正在做什么的直接信息。转速表可以同附件28上的一个A/D转换器连接起来（例如，ADC1）。这样转换表入口就能将A/D的值积分成伺服环所要用的位置信息。资源和处理字将是$50C006;偏移项将依靠经验地设置为当电机静止时保持积分值不变。电机的Ix04将指出使用积分值的第二行的入口。

第二种，使得在PMAC内部做级联环成为可能。外环（可能是作用在标准位置环周围的力或张力环）将产生一个作为速度修正的命令值给内部位置环。位置环将把这修正作为来自主位置控制寄存器的位置信息来接受。转换表入口可以将外环的速度输出转换为内环的位置输入。外环通过将Ix02设置给一个内部未用的存储寄存器，从而直接将它的命令输出到该寄存器（例如，Y:$07F0）。在该转换表中的积分模拟入口将把该寄存器用作它的源地址（$5007F0）。不需要偏置。内环的Ix05指出了该入口的第二行，其中放着作为位置环修正的积分值。

无符号模拟:

如果模拟转换设置字的第19位被设置为1(对标准模拟是$18xxxx,对积分模拟是$58xxxx),PMAC将把在资源字的高16位里的A/D数作为范围是0到65,535的无符号数对待，而不是作为范围是-32,768到+32,767的符号数对待。无符号的转换要求使用更新的附件28B。而通常带符号的转换（第19位是0）只要求使用原来的附件28和附件28A。

**并行位置反馈转换：**

如果你将位置信息作为并行数据字提供给PMAC（例如，从一个绝对编码器，或来自一台激光干涉仪），那么你将使用$2x,$3x,$6x或$7x中的一种转换格式。格式$2x和$3x从Y存储空间的指定资源获取数据；$6x和$7x则从X存储空间的指定资源得到数据。通常该数据是通过Y存储空间里的附件14而引进的，所以$6x和$7x格式很少被用到。如果在转换格式中的x的值是0或1（格式字第19位是0），则结果数据将会左移5位，以便资源字的最低有效位能够作为一个“数”给伺服算法（要求有5位小数）。如果x的值是8或9（格式字第19位是1），结果数据将不会移动，并且LSB将给伺服算法显示1/32计数（参见下面的*无移动转换*）。如果x的值是1或9（格式字第16位是1），结果数据将和前面入口的结果

相加。如果x的值是0或8（格式字的第16位是0），不执行相加的动作。
附件14源寄存器：

当使用附件14来引入数据，下列资源地址将会被用到：

| | |
|---|---|
| 1st ACC-14 Port A (J7) : | $FFD0 |
| 1st ACC-14 Port B (J15): | $FFD1 |
| 2nd ACC-14 Port A (J7) : | $FFD8 |
| 2nd ACC-14 Port B (J15): | $FFD9 |
| 3rd ACC-14 Port A (J7) : | $FFE0 |
| 3rd ACC-14 Port B (J15): | $FFE1 |
| 4th ACC-14 Port A (J7) : | $FFE8 |
| 4th ACC-14 Port B (J15): | $FFE9 |
| 5th ACC-14 Port A (J7) : | $FFF0 |
| 5th ACC-14 Port B (J15): | $FFF1 |
| 6th ACC-14 Port A (J7) : | $FFF8 |
| 6th ACC-14 Port B (J15): | $FFF9 |

对于这种类型的反馈的典型设置字是$20FFD0,它为不滤波的转换提供了反馈给和PMAC连接的第一片附件14端口A的并行数据。

位有效屏蔽字：

并行反馈转换要求在转换表里有一个双重（不带滤波的）入口或三重（带滤波的）入口。不论是带滤波还是不带滤波，第二个入口指定了反馈字所用的大小。入口是一个24位的字，其中每一位实际上为并行反馈所使用的位是1；没有用到的位是0（并行反馈是从数据字的位0开始连接的）。对于一个12位的绝对编码器，入口将是$000FFF;对于14位而言，它将是$003FFF。

对于标准的移动转换，最大有效屏蔽字是$07FFFF，没有屏蔽低19位；任何在高5位的数据将被移出该字。对于不移动转换，最多到实际数据的24位都可被用作屏蔽字$FFFFFF。屏蔽字允许PMAC在资源数据里检测浮动，以便它能在软件中正确地扩展计数。

滤波器字：

如果转换格式是$3x或$7x，并行数据字将被滤波。滤波器将设置一个最大的量，在该量内时，在一个单独的伺服周期中数据字允许被改变。如果PMAC发现改变量超过了设置的最大量，则资源数据字将按照设置的最大量进行改变。

滤波的目的：

滤波允许在根本没有延迟合理变化时，提供抵抗高位数据线上的干扰变化的保护。这个最大量是给在转换表Y列的编码器的第三设置入口。它应当被设置得只比传感器所希望的最大实际速度稍微大一点。用每伺服周期的计数位来表示。

被转换的数据：

从并行字被转换的数据放在同给入口的最后（第二或第三个）的设置字相匹配的X数据字里。这是通过采集位置的电机I－变量（Ix03,Ix04,或Ix05）将要使用的地址。例如，如果在转换表中的第一设置入口值（地址
Y:$0720）是$30FFD0（滤波后的并行数据），大小入口值将在Y:$0721,并且最大改变量入口值将在Y:$0722。转换后的数据将被放在X:$0722
。如果这是给1#电机的位置反馈，Ix03将被设置为$0722(十进制值为1826
)。对于增量并行反馈，为了正确的回零搜索移动，Ix03的第16位将被设置为1。

对于未滤波的并行反馈，一个入口将是：

| X字 | Y字 |
|---|---|
| 1. 中间数据：<br>经符号扩展的最高有效字 | 1. 资源和处理：<br>Bits 0-15:<br>资源数据的地址<br>如果是$20转换，则Y字 |

| | |
|---|---|
| | 如果是$60转换，则X字<br>Bits 16-23:<br>对于Y字源，＝$20<br>对于X字源，＝$60 |
| 2． 转换的数据：<br>Bits 0-4： 小数位<br>Bits 5-23: 整数位 | 2.位有效屏蔽：<br>Bit=1,使用来自源字的对应位<br>Bit=0,不使用来自源字的对应位 |

对于经滤波的并行数据转换，一个入口将是：

| X字 | Y字 |
|---|---|
| 1． 中间数据：<br>原始数据读入 | 1． 源和处理：<br>Bits 0-15:<br>源数据的地址<br>如果是$30转换，则Y字<br>如果是$70转换，则X字<br>Bits 16-23:<br>对于Y字源，＝$30<br>对于X字源，＝$70 |
| 2． 中间数据：<br>带符号扩展的最高有效字 | 2.位有效屏蔽：<br>Bit=1,使用来自源字的对应位<br>Bit=0,不使用来自源字的对应位 |
| 3. 转换的数据：<br>Bits 0-4： 小数位<br>Bits 5-23: 整数位 | 4.滤波器值：<br><br>最大允许的改变（按计数位/伺服周期） |

不移位的转换：

对于一个并行数据转换的“资源和处理”字的第19位如果被置为1，那么转换后的数据将不包括小数位。这种格式的入口将使用转换格式（该字的16－23位)$28,$38,$68,或$78，和在转换后的数据中提供5位小数位的标准入口的格式$20,$30,$60,和$70是相反的。

*用途：*

不移位的格式可以用在高速度，很高分辨率的方面，例如用在激光干涉仪反馈上。当用标准的移位的格式时，PMAC的内部速度寄存器状态（计数/秒）最高到256M（268,435,456）。而用不移位格式时，这个限制将提高32倍：8G（8,589,934,592）。

使用不移位格式时，用户必须注意PMAC将反馈设备的最低位作为一个数1/32来对待，而不是作为一个数来对待。例如，你有一个传感器在电机上（X轴），分辨率是2.5纳米，而你想要按毫米来给该轴编程，那么你将把一个数作为80nm(2.5×32)对待,并且使你的轴的定义为#1->12500X (因为1,000,000/80）。

对于这种的转换表入口将由一个资源和处理字$38FFC2(Y:$FFC2的不移动并行转换），一个屏蔽字$00FF00（只使用24位字的中间8位），和一个滤波器值如＄000020（每一伺服周期最大速度32计数）。从属于该操纵轮的任何电机x的Ix05将被设置为该入口的第三根线的地址。对于1：1的随动比及Ix08的默认值是96，Ix07的值将被设置为12而不是通常的96，以便反映来自改附件的每一次计数其显示都比正常快8倍的情况。

右移并行转换：

如果给并行数据反馈的“资源和处理”字的第19和18位都被设置为1，那么资源地址中的行数据在被放入结果字里之前将被右移3位。这种形式的入口将有$2C,$3C,$6C,或$7C等转换格式（该字的16－23位）。这种转换格式适用于放在24位字（LSB是第8位）的高16位的数据，就象来自一个MACRO输入寄存器的反馈。

该格式的另一用途是用附件39手轮译码器。该板有一个HCTL－2000正交编码器IC，它能将来自手轮的正交信号转变为一个8位的并行字，该字可以通过JPAN控制板端口被传递进来。在PMAC－PC

、－Lite，和－VME上，该字节是通过寄存器Y:$FFC0的8－15位来显示的。一个标准的并行转换将把手轮计数器的第1位放到第13位，使得它比如果放在正常的第5位位置大了256倍。右移转换将第1位放到第5位，就象标准编码器一样。

当使用该右移格式时,位有效屏蔽字应该反映移位后被使用的位的位置。例如，如果所有的高16位（8到23位）都被使用了，那么位有效屏蔽字应为$1FFFC0，来标志移动后5到20位的使用。对于使用8-15位的附件39，屏蔽字应该是$001FC0，来标志移动后5到12位的使用。

**时基转换入口：**

一个“时基”转换基本上是一个比例数值求导。当源数据是一个计数频率计，其结果是每个伺服周期的一个频率值，那么表将计算给该周期的资源寄存器的值和给上一周期的值之间的差，并且将此差值乘以比例因子。

对于结果值的最普遍的用途是进行时基（进给速度修调）控制，这可以使得PMAC的执行速度能和外部频率相称（一般是一台主设备的速度）。参见《使PMAC和外部事件同步》一章，可以得到更详细的情况。

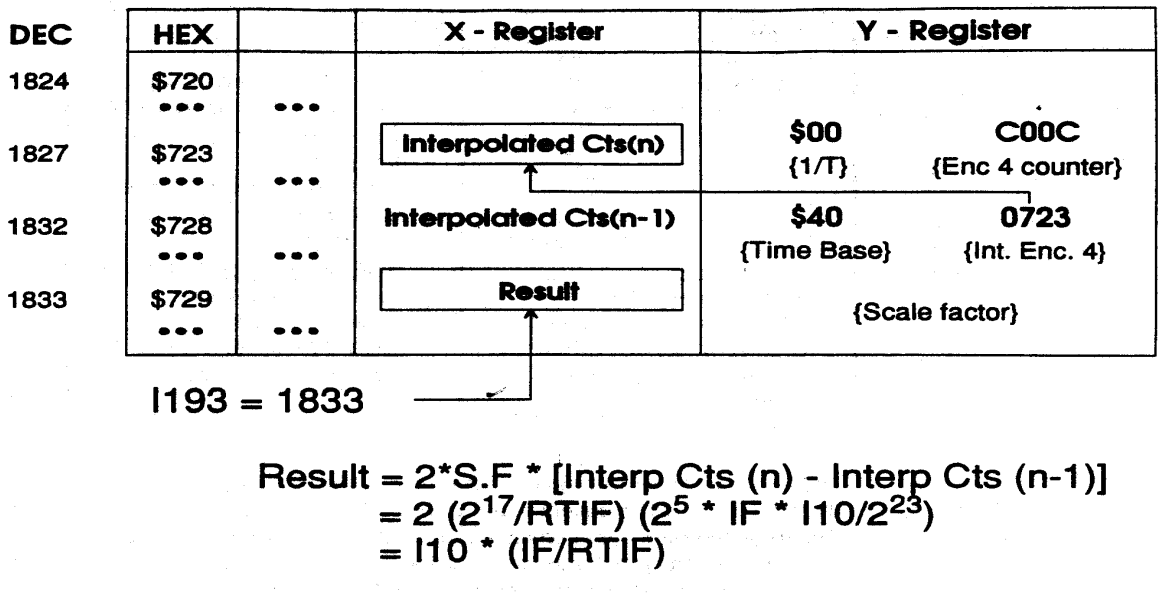
对于时基转换，在每一列有两个入口，其格式是：

| X字 | Y字 |
|---|---|
| 1. 上一周期的源数据：<br>Bits 0-4 : 小数位<br>Bits 5-23: 整数位 | 1. 源和处理：<br>Bits 0-15:<br>源数据的X地址（通常是一个被转换了的位置寄存器）<br>Bits 16-23:<br>对于时基转换＝$40 |
| 2. 实际的时基值：两次源值的差和比例因子的乘积 | 2. 时基比例因子（用户提供） |

例如，缺省转换表将从在编码4计数器里的数据建立一个时基值。在时基转换中要求资源数据带有子计数插补，通过减少由数值差值产生的量化误差，这将使处理过程变得更顺利。为了做到这些，资源寄存器也应来自转换表本身，而不是来自编码计数器。

在缺省的转换表中，来自编码器4的被转换了的数据放在X:$0723(十进制是1827)。因此给时基转换入口的第一个设置（Y）字是$400723,其中$40指定为时基转换，而$0723则指定了源地址。

**（PAGE 3－61时基入口的转换表例子图）**




例因子：

第二个设置（Y）字是比例因子，就是乘以当前源数据和上次资源数据之间的差值的值。设置它的值通常需要一些计算；关于这一主题，可参见“坐标系特性”的时基控制部分。

转换了的数据：

上一次的源数据字存放在表的入口的第一个X字中，而最终结果是放在第二个X字中。最终结果的值是2×比例因子×（新资源－旧资源）。如果你想用该值控制坐标系的时基，你将给坐标系键入作为Ix93的值的地址（时基资源地址）。

**触发的时基转换输入：**

对于那些必须同主编码器的位置确切同步的应用，当你在计算同步顺序的第一步移动时转换表提供了冻结时基的能力，然后当启动触发发生时，准确参考着发生的主位置让时基启动（通常是主编码器的索引通道）。这就提供给那些从属于主控设备的从件完全的位置锁；因此没有必要使后续调整确保它们“反相”，而对于标准（非触发）时基控制的情况这恰恰是需要的。

输入格式：

对于一个触发的时基转换，在每一列有两个输入，其格式为：

| X字 | Y字 |
|---|---|
| 1．上一周期的源数据：<br>Bits 0-4：小数位<br>Bits 5-23: 整数位 | 1．源和处理：<br>Bits 0-15:<br>源数据的X地址（通常是<br>一套DSPGATE编码寄存器）<br>Bits 16-23:<br>$90(被冻结的；为准备)<br>$B0(等待处理的；等待触发)<br>$A0(正执行的；后置触发器) |
| 2．实际的时基值：两次源值的差和比例<br>因子的乘积 | 3．时基比例因子：<br>用户提供；等于131,072/在cts/<br>毫秒里的实时输入频率 |

不同于标准（非触发）时基转换，源数据必须来自是在DSPGATE里的带有原始（未处理的）数据编码寄存器。触发的时基转换本身完成1/T插补功能。给触发的时基入口的有效地址和给增量编码器入口的有效地址是一样的：$C000给编码器1，$C004给编码器2，等等，直到$C03C给编码器16。

设置触发器状态：

一个单独的触发的时基输入的“进程”位（在转换表入口里的第一个Y字的16到23位）在使用的正常过程中将具有三个值。这通常是通过一个8位的M－变量来完成的。首先，当从动轴位于其启动位置时，在顺序运动程序做第一次移动的计算中进程位将被设置为$90。这将强制时基值为零，并使坐标系处于进给保持模式。

其次，另一个程序（通常是一个PLC程序）将这些位从$90改变到$B0。这使时基处于等待处理状态，以便它等待资源编码器上的位置捕获触发器，这是通过给该编码器的编码器/标志I－变量2来定义的。当触发时，PMAC自动将进程位从$B0改变到$A0。对于使用该格式的未触发用途，用户只能自己将进程位设置为$A0。

示例：

例如，我们将在标准转换表的末尾加上一个从编码器8工作的触发的时基输入，带有64cts/毫秒的实时输入频率。这些入口将位于寄存器$072A(1834)和$072B(1835)处。初始时我们将给Y:$072A写入一个值为$A0C01C(执行来自编码8寄存器的时基)，并给Y:$072B写入值$800(131,072/64=2048=$800)。我们将一个M－变量定义给该进程位，通过命令M199－>Y:$072A,16,8。当从动轴位于启动位置时，我们通过运动程序命令M199＝$90冻结该时基。如果Ix90还没有指向X寄存器$072B，那么现在完成它。这之后的运动程序命令将计算移动，但是时基值是零，运动的执行将放在起始点。然而一个PLC程序执行时将搜寻等于$90的M199，并将其改为$B0,等待触发。因为一个PLC程序在计算移动的运动

程序计算过程中中断，所以我们可以确信这个改变要直到计算全部完成之后才会发生。这个改变可以通过一个PLC程序中的三个程序行来完成：

```
IF （M199＝＄90）
   M199＝＄B0
ENDIF
```

一旦通过PLC程序准备好了触发器，当捕获触发器发生是，PMAC将自动开启时基并把进程位改变为$A0。

**指数滤波器入口：**

使用转换表在输入数据的字上创建一个指数滤波器是可能的。这对于位置跟随（电子齿轮）特别有用，特别是从动件对于主动件是“增速”时，它能使从动轴的运动更平稳。

每一伺服周期n的指数滤波器公式为：

$$Out(n)=Out(n-1)+(K/2^{23})\times[In(n)-Out(n-1)]$$

*If[Out(n)-Out(n-1)]>Max_change,Out(n)=Out(n-1)+Max_change*
*If[Out(n)-Out(n-1)]<-Max_change,Out(n)=Out(n-1)-Max_change*

In,Out和K都是带符号的24位的数（范围是：从-8,388,608到8,388,607)。差值*[In(n)-Out(n-1)]*被截断到24位，以便能正确地处理翻转。

在伺服周期中滤波器的时间常数是$2^{23}$/K 。K值越低，时间常数越大。

没有执行移动动作。任何操作（例如1/T插补）都应该在数据被准备好后才能被进行，以便给该滤波器的源寄存器将是上一次操作的结果寄存器。

指数滤波器的输出值放在转换表入口的第三行的X寄存器里。用到该值的操作会将该第三个寄存器选址；例如，Ix05对于位置跟随，或对于时基转换表入口的源地址（在时基中保持位置锁定，该滤波必须在时基求导之前被执行，而不是之后）。

入口格式：

对于一个指数转换，每列有三个入口，其格式为：

| X字 | Y字 |
|---|---|
| 1．中间数据 | 1． 资源和处理：<br>位 0-15: 资源的X地址（通常为一个转换了的位置寄存器）<br>位16-23 = $D0 |
| 2．中间数据 | 2.<br>输出值的最大允许改变：每一伺服周期中用LSBs中表达 |
| 滤波结果：没有从资源数据中移动 | 3. 指数增益K，滤波公式为<br>Out(n)=Out(n-1)+(K/$2^{23}$)×[In(n)<br>-Out(n-1)] |

示例:

用占据寄存器＄0720到＄0729位置的默认编码转换表启动，要求在表中加上一个新的入口，以便能对和PMAC上编码器5相连的手轮编码器进行滤波；滤波器将有一个8个伺服周期的时间常数，输出的最大速度是每伺服周期16计数位。

在缺省表里，给编码器5的1/T插补结果被放在寄存器X：＄0724中。这是给指数滤波器的源地址。既然8个伺服周期的时间常数等于8,388,608被滤波增益K除，那么K等于1,048,576。源寄存器单元（ISBS）是1/32计数，所以最大的变化量是32×16，或每伺服周期 512LSBs。

这些值可以被键入PMAC执行程序（V3.0或更高版本）的交互式菜单中，也可以用直接内存写命令直接写入：

WY：$072A，$D00724，512，1048576

Y：$072A是PMAC中该入口的起始位置；$D0指定了指数滤波器；512(或$200)指定了最大输出改变化率；1048576（或$100000）指定了滤波器增益。如果经过滤波的值将被用作一个主编码，Ix05将被设置为$072C。

**设立编码器转换表：**

编码器转换表是从PMAC内存地址$720（十进制为1824）开始的。它可以连续，甚至通过地址$73F（十进制为1855）。表的激活部分将由第一个全部为零的Y字结束。才出厂的编码器转换表能够转换在基本PMAC板位置$720到$727（1824到1831）的8个增量编码寄存器。位置$728和$729创建了来自转换了的编码器4寄存器的时基信息。Y：$72A是零时，结束表的激活部分。缺省表如下所示：

| 地址 | Y字 | 含义 |
|---|---|---|
| $720（1824） | $00C000 | 编码器1的1/T转换 |
| $721（1825） | $00C004 | 编码器2的1/T转换 |
| $722（1826） | $00C008 | 编码器3的1/T转换 |
| $723（1827） | $00C00C | 编码器4的1/T转换 |
| $724（1828） | $00C010 | 编码器5的1/T转换 |
| $725（1829） | $00C014 | 编码器6的1/T转换 |
| $726（1830） | $00C018 | 编码器7的1/T转换 |
| $727（1831） | $00C01C | 编码器8的1/T转换 |
| $728（1832） | $400723 | 来自转换的编码器4的时基 |
| $729（1833） | $000295 | 给上面的时基比例因子 |
| $72A（1834） | $000000 | 表结束信号 |

大多数用户可以不用改变的使用该表。注意缺省的电机反馈位置地址和主位置地址I一变量（I103到I105，I203到I205，等等）在该表中指向位置，并且假定表的默认设置。

然而这儿有几条理由，用户可能需要改变该表。第一，如果应用中使用了附件24轴扩展板，它将需要转换编码器9~16，因此给这些编码器的入口必须加到表里去。第二，用到外部时基频率资源的用户要改变比例因子，还有资源。第三，用户也许不想在位置反馈上使用1/T插值。第四，要求非常快地控制少数几根轴的用户可能想要缩减该表以节省计算时间，因为每一次转换都要耗费一定的时间。

示例：

一个用户控制两根轴，轴上有非常快的激光干涉仪正交反馈（进入到编码器1和3），带有并行的子计数插补，没有操纵轮或外部时基，他可以设立如下的一张表以得到最小的转换时间：

| 地址 | Y字 | 含义 |
|---|---|---|
| $720（1824） | $80C000 | 编码器1的子计数转换 |
| $721（1825） | $80C008 | 编码器3的子计数转换 |
| $722（1826） | $000000 | 表结束信号 |

示例：

一个用户，想要转换来自第一片附件14的两个16位绝对编码器，波后只允许每伺服周期最大8位的改变，4个带1/T插补的增量编码器（ENC1~4），4个A/D转换器（ADC1~4），和两个未进滤波的线性位置传感器（ENC9~10）。转换表的设置可以是：

| 地址 | Y字 | 含义 |
|---|---|---|
| $720（1824） | $00C000 | 编码器1的1/T转换 |
| $721（1825） | $00C004 | 编码器2的1/T转换 |
| $722（1826） | $00C008 | 编码器3的1/T转换 |
| $723（1827） | $00C00C | 编码器4的1/T转换 |

| | | |
|---|---|---|
| $724（1828） | $10C006 | ADC1的转换 |
| $725（1829） | $10C007 | ADC2的转换 |
| $726（1830） | $10C00E | ADC3的转换 |
| $727（1831） | $10C00F | ADC4的转换 |
| $728（1832） | $30FFD0 | 从第一片附件14的经滤波的并行 |
| $729（1833） | $00FFFF | 使用字的低16位 |
| $72A（1834） | $000100 | 最大改变量是256计数/周期 |
| $72B（1835） | $30FFD1 | 从第一片附件14的经滤波的并行 |
| $72C（1836） | $00FFFF | 使用字的低16位 |
| $72D（1837） | $000100 | 最大改变量是256计数/周期 |
| $72E（1838） | $20C020 | 来自编码器9计时器的并行 |
| $72F（1839） | $07FFFF | 使用低19位（最大允许的） |
| $730（1840） | $20C024 | 来自编码器10计时器的并行 |
| $731（1841） | $07FFFF | 使用低19位（最大允许的） |
| $732（1842） | $000000 | 表结束信号 |

如果你不希望使用PMAC执行程序里的转换表编辑器界面，你可以用一个Read-Hex（RH）命令来查看转换表当前的设置。例如，假定转换表就如同上面的例子所设置的那样，那么命令RHY：$720，24（以十六进制汇报自$720起的24个Y字）将会有以下的情况：

```
00C000 00C004 00C008 00C00C 10C006 10C007 10C00E 10C00F
30FFD0 00FFFF 000100 30FFD1 00FFFF 000100 20C020 07FFFF
20C024 07FFFF 000000 000000 000000 000000 000000 000000
```

如果你不喜欢用PMAC执行程序里的转换表编辑器，你可以通过一到几个Write（W）命令来改变表里的入口。例如，你想要把第三和第四入口改变到直接到表中的同一寄存器，你需要命令为：

```
WY：$722，$C0C008，$C0C00C
```

当然设立值也可以用十进制数指定，但你将发现用十六进制数指定更加容易些（除了比例因子）。

编码器转换表的设置通过用SAVE命令能够被存储到EAROM中。在启动或重置时最近存储的设置将从EAROM拷贝到RAM（激活的内存）中，所以假如你想保存对表的改变，那么你应该使用SAVE命令将改变保存到表中去。

## 更深入的位置处理：

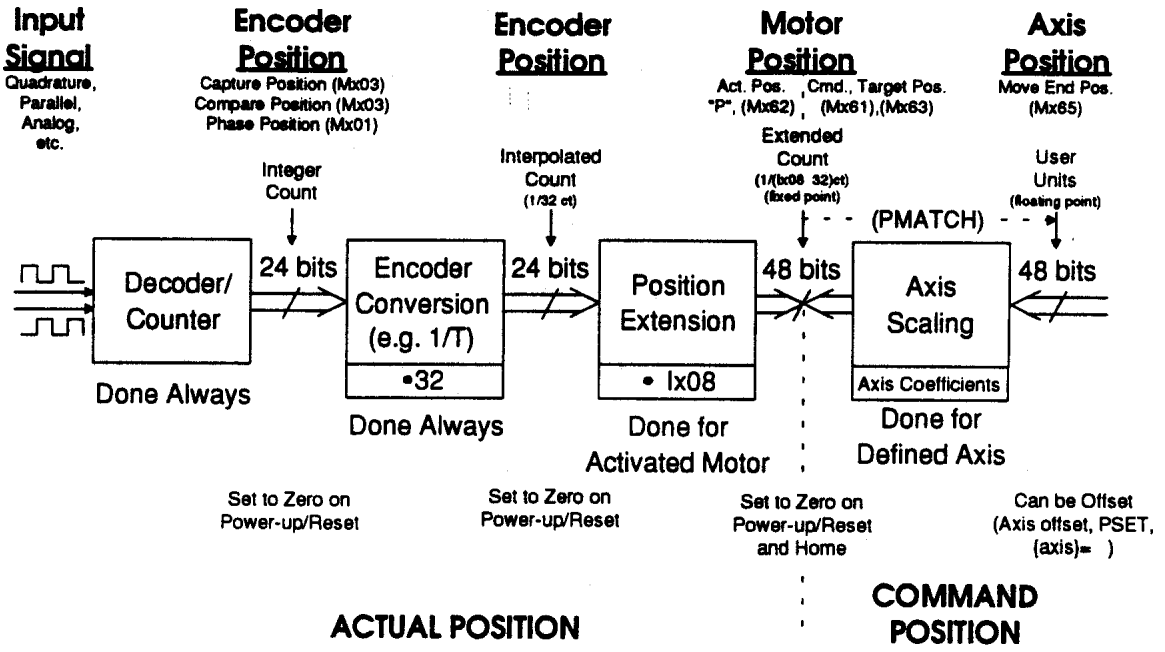
一旦位置反馈信号已经被编码器转换表处理了（这是在每一伺服周期开始时进行的），数据就准备被伺服环使用。

**软件位置扩展：**

对于每一台被激活的电机，PMAC将取得位置信息，放到由Ix03所指的24位寄存器，并且在软件上将它扩展为包含实际电机位置的48位寄存器。在这个扩展过程中，PMAC将该值同Ix08的位置比例因子相乘。既然转换表中的寄存器是以一个计数的1/32为一单元，那么实际电机位置寄存器就是以一个计数的1/(Ix08×32)为一单元。

这些扩展了的电机位置寄存器在启动/重置时被设为零（除非有一个绝对位置传感器），而且在寻零移动结束时又被设为零。编码器位置寄存器只在启动/重置时才被设为零。因此，在电机被回零之后，在电机零位置和编码器零位置之间有一偏置。

**（PAGE 3－67 PMAC位置处理图）**




需要担心这个偏置的唯一用户是那些需要直接使用编码器寄存器用户（例如，位置捕获和比较），和那些可能述及这些给电机位置的值的用户。这些用户需要发现并存储该偏置，当回零触发被发现时，该值就在位置捕获寄存器中。在示例部分的程序HOMOFFST.PMC将演示怎样做。如果他们想要移动超过＋/-
8兆计数，那么他们将不得不处理编码寄存器的翻转。取模（%）操作符对于这非常有用。参见《使PMAC和外部事件同步》一章，可以得到更多的细节。

**轴位置比例：**

电机位置通常是用计数的形式来保持的。当一台电机通过一条轴定义语句（见坐标系特性）被分配给一根轴，在语句中的比例因子就决定了轴的单位是什么（一般是英寸、毫米、度等等）。编程的移动被给了轴，并且PMAC通过使用来自轴定义语句中的比例因子，将此转换为电机的运动。该转换只是对*命令位置*而言，并且转换通常只有从轴到电机一条途经，认识到这一点是很重要的。PMAC从来不计算实际的轴的位置。

**螺旋补偿：**

PMAC能够执行通常所说的“丝杆补偿”的功能。这项技术通常也有着别的一些名称，对于一张修正表来说可作为电机位置功能进入到PMAC中。PMAC能够储存8个这样的补偿表。

每一台电机都有一个“属于”它自己的表。除非另外指定，该表将使用来自该电机（源数据）的位置信息决定在表中的位置，并且将它的修正加到该电机上（目标数据）。然而，不管是源电机，或者是源电机和目标电机两者，它们都可能被指定为那张修正表不属于它们的电机。（如果两电机是不同的，那么这张表“属于”一台电机的概念只对PMAC自己的资源管理有用。）

补偿是在伺服环（每一伺服周期）内被执行的，以便获得最大的速度和最高的精度。PMAC取得源电机的位置，并在表内找到匹配的位置。典型地，该位置是在表的两个入口之间，因此PMAC能在这两个入口之间线性地插补，从而得到给当前伺服环的修正。然后它将该修正加到目标电机的位置上去。在表上，修正的入口必须是整数值，并且带有目标电机的1/16计数单位（因此一个48的入口值代表3个计数）。

每台电机的复合表：
　一台电机可以提供给源数据以多达8张补偿表；目标电机也可以多达8个。

表的范围：

补偿直接定义了一个给源电机从在零计数位的起始位置（最近的回零位置或启动/重置位置）沿正方向运动的范围。该范围的大小是作为命令*DEFINE COMP*的最后一个变元而声明的。该变元有源电机的计数的单位。入口之间的间距是整个范围被入口的数目除而得到的（入口的数目是命令*DEFINE COMP*的第一个变元）。表里的第一个入口值定义了距离资源电机起始位置一个间距的修正，第二个入口值定义了距离资源电机起始位置两个间距的修正，等等。

浮动：

在范围之外，未修正的位置在补偿被做之前“浮动”到范围以内，实质上是一个取模的操作。这就允许旋转轴的补偿越过几转，并且允许给编码器偏心以简单的补偿。当然，如表能够大到覆盖整个资源电机的移动，那么浮动的特性就没有必要再用。

如果电机的移动范围是在零位置的负方向，而这时又要求补偿，那么这些入口将被设置地如同它们已经经过了电机范围的正端。例如，假定一台电机的移动范围是＋/-50,000计数，并且每500计数设一入口值（因此总共要200入口值），那么该表可用下面的一条命令来设置：*DEFINE COMP 200，100000*

。第一个100入口值将覆盖500到＋50,000计数范围，而后面的100入口值将覆盖－50,000到0计数范围。（通常，表都在资源电机的零位置有一个零修正，所以在表里的最后一个入口值应该是0。）实质上，－50,000到0的范围将被映射到+50,000到+100,000范围上。

示例：如果下面的简单的表被键入：

```
＃1
DEFINE COMP 8，4000  ；有8个入口值，覆盖4000cts的表
                    ；属于1＃电机
                    ；用1＃电机给资源和目标
                    ；因为没有别的电机被指定
－160               ；在4000/8（500）cts的修正是
                    ；－160/16＝－10
80                  ；在1000计数位的修正是5计数位
120                 ；在1500计数位的修正是7.5计数位
96                  ；在2000计数位的修正是6计数位
20                  ；在2500计数位的修正是1.25计数位
-56                 ；在3000计数位的修正是-4.5计数位
-12                 ；在3500计数位的修正是-0.75计数位
0                   ；在4000(和0)计数位的修正是0
```

并且轴的定义是＃1－〉1000X，一个要求到X1.3的移动将给出一个没有修正的电机位置（1300计数位）。所用到的修正将在表中被线性插值：

修正＝(7.5-5)×(1300-1000)÷500＋5＝＋6.5计数位在X8.4，PMAC将计算出未修正的电机位置是8400计数位；将此浮动到表的范围内：8400 mod 4000＝400计数位；所以从表中来的修正值将是：

修正＝(-10-0)×(400-0)÷500+0＝－8计数位

使能和禁能：

当I51被设为1时，所有的丝杆补偿表都被使能；当I51被设为0时，全部禁止。

十字轴补偿的使用：

对一张表拥有互相隔离的资源和目标电机的能力有下面几个用途。第一，是给不完全几何特征的传统补偿，就象一个弯曲的螺杆一样，例如，在X、Y平面，假如X轴丝杆弯曲，Y轴将接收作为X轴位置函数的校正数据，如果#1电机是X轴，而且#2电机是Y轴，校正表将把#1电机作为源电机，#2电机作为目标电机。

十字轴补偿的第二个用途是称作“电子凸轮”的应用。在这种情况下，目标电机的整个移动都是由补偿表的入口值引起的，并不仅仅是修正。通过PMAC的时基随动来创建电子凸轮操作的方式比另一种创建电子凸轮额方式有两个重要的优点：补偿表是双向的，因此主轴可以按正反两方向中任意一个方向旋转；而且它是绝对的，因此“反相”回零运动只是一个简单的事情。

从动电机的运动程序在其中定义了运动的时基方式保持了下面的两个优势，一是通过程序中的代数和

逻辑能在不工作时改变，二是能在两点间进行2阶或3阶插值，不象补偿表只能进行一阶插值。

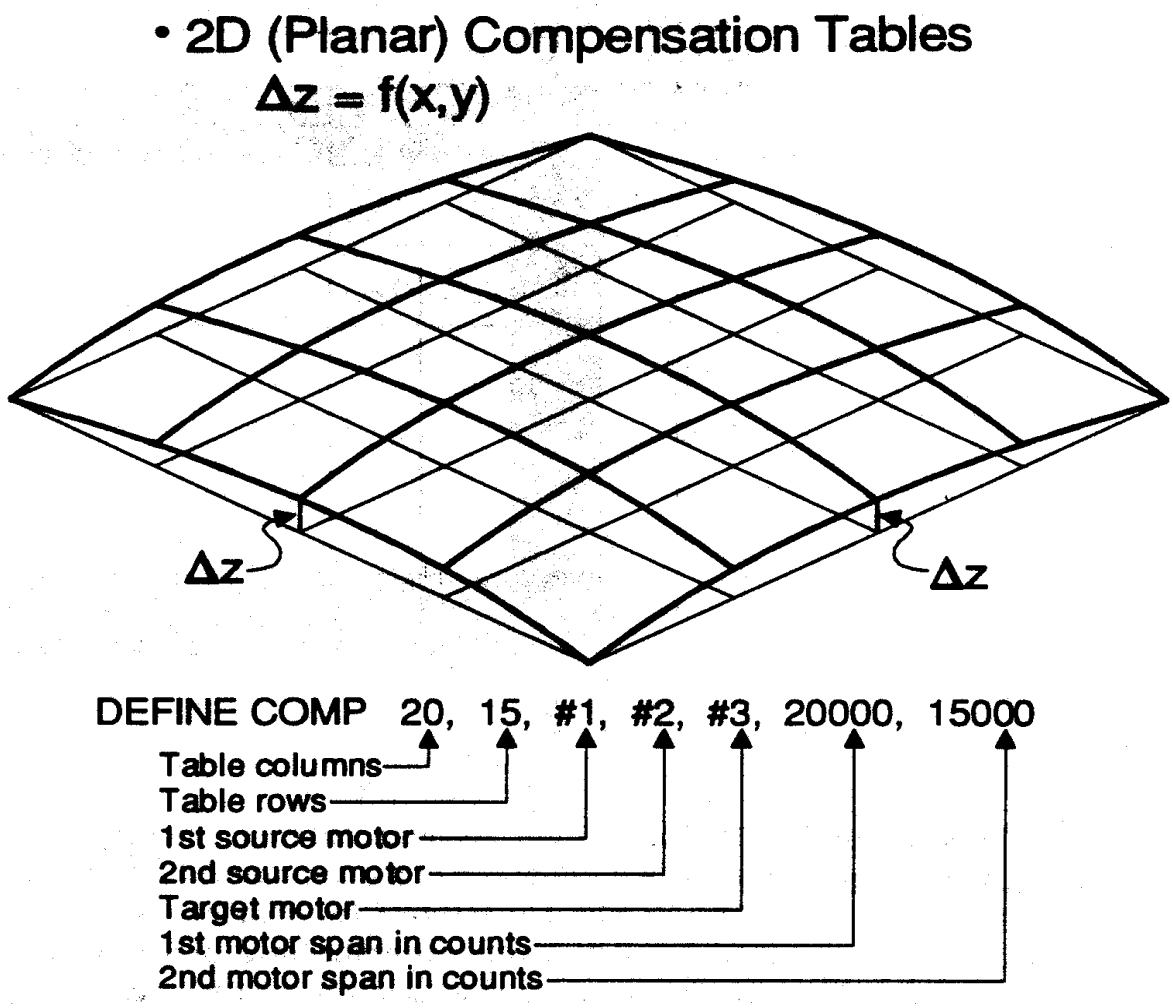
二维丝杆补偿：

在PMAC上建立二维的补偿表是可能的，该补偿是两台电机的位置的函数。这就使通过指定补偿点的栅格来建立平面补偿是可能的。一个2D补偿表有两台源电机和一台目标电机。目标电机也能够是源电机中的一台。

如果在建立补偿表的DEFINE COMP命令的大小参数有一个小数点，那么它是一张二维表，小数点前的值指定了“列”的数目，或者指向第一台源电机；小数点之后的值指定了“行”的数目，或者指向第二台源电机。

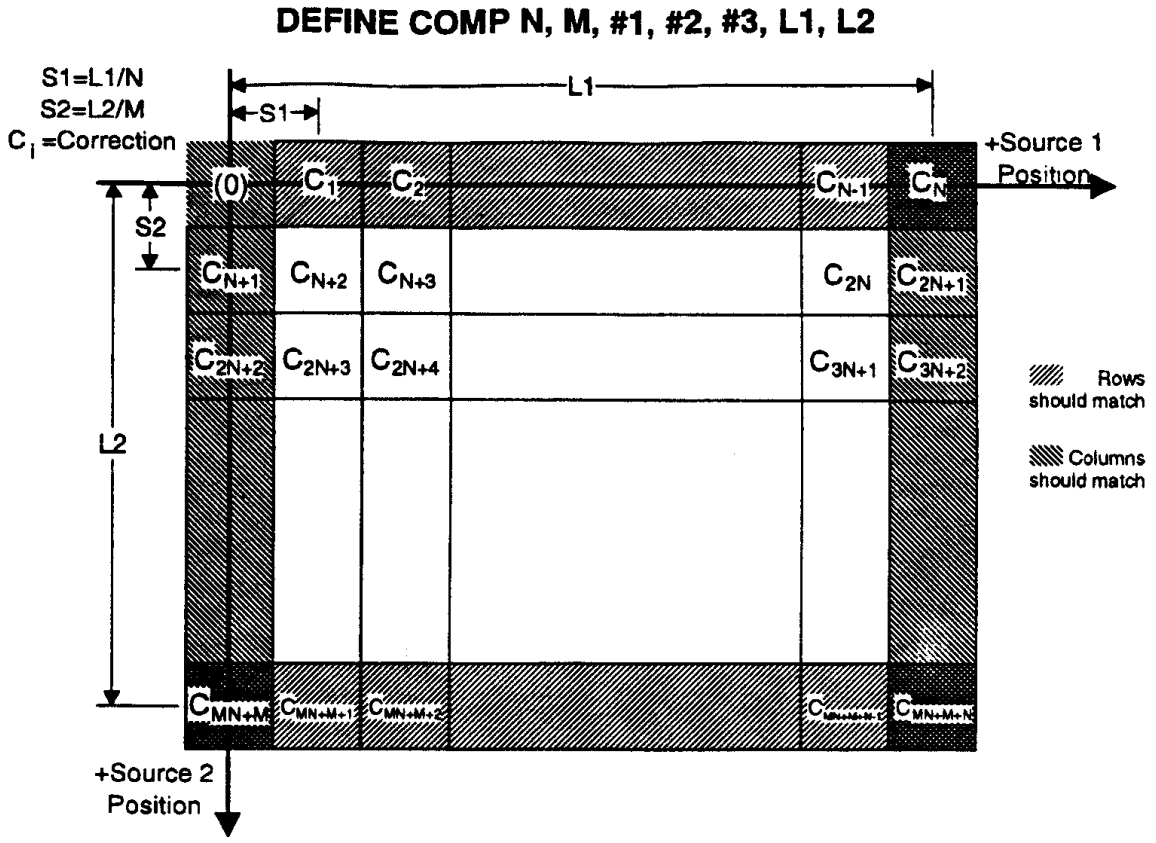
在操作中，PMAC将给在两台源电机的平面里的给定位置计算补偿，作为围绕该位置的4个指定的补偿值的加权平均。参见《软件参考》中关于2D DEFINE COMP命令的描述，抗议得到更详细的细节。

（PAGE 3－71 PMAC补偿表图－2D平面补偿表）



## （PAGE 3－72 二维补偿表，DEFINE COMP）

间隙补偿：




PMAC能够给所有电机执行复杂的间隙补偿。当按和要求的速度的方向反转时，一个可预处理的间隙就加了进来。该间隙在电机移动中可以是一个常数，也可以是电机位置的一个函数。间隙被引入和去除的比例是可编程的，就象要求间隙的反转的量级被引入和去除一样。

常数间隙:

给电机x的Ix86是常数间隙距离参数。当电机要求的移动的方向从正向变为负向时，该值就被引入激活的间隙补偿寄存器，该寄存器将减去名义上的要求的位置。当电机所要求的移动的方向从负向变为正向时，间隙补偿寄存器内的值将减到零。电机反转时，Ix86的正值加上一个附加的距离给电机的移动，从而补偿真正的物理上的间隙。Ix86的单位是一个计数的1/16，所以它的值将是要求的间隙补偿的计数位数目的16倍。

间隙张紧率：

Ix85控制反转时，给电机引入或去除间隙的比率。这允许用户优化出迅速而且平稳的间隙补偿。当检测到反向时，每一个后台周期（在每一次PLC扫描之间），一个等于Ix85的数被加到激活的间隙补偿寄存器中，或从该寄存器中被减去，直到该寄存器的值等于Ix86的值或等于0。一般地，都是用能产生平稳转换的最高的Ix85的值。

间隙滞后：

I99控制PMAC上所有电机在要求位置新的方向上的计数位的数，这些必须是在PMAC判定反向已经发生
并且间隙必须被改变之前就应该被查明的。如此，I99扮演了一个“滞后”的项目。如果是一个主编码器用来驱动一台电机，这就特别有用，在主编码器中的如此轻微的颤动调谐不会导致间隙的重复引入和去除。I99单位是1/16计数位，所以默认值64将提供4个滞后的计数位。

间隙表：
用DEFINE

BLCOMP命令创建的间隙补偿表能够被用来产生随着电机位置变化的间隙距离。这多数时候是和一个丝杆补偿表联合起来使用，以产生一个双向丝杆补偿表的效果。对于一个给定的电机位置，采自间隙表的间隙距离的值将被加到Ix86“常数”间隙参数上。在电机位置0（回零位置），来自改表的间隙距离被定义为零，所以如果间隙表被使用了，Ix86将包含在回零位置的间隙值。

给一台电机的间隙表只有当运动的最近的要求方向是负向时才被激活；当电机经过负方向移动到该位置，并在现在被要求保持不变时间隙表依然被激活。在操作中，该表读入当前的电机位置，并计算两个最近的表入口的加权平均，在两个点之间产生一个一阶插值。间隙补偿的定义是配合着电机位置的范围的，该电机位置的范围是从零计数位起始，沿正方向行进，并到达由命令DEFINE BLCOMP的最后一个参数声明的计数长度。入口之间的间距是该长度被入口的数目除（入口的数目是命令DEFINE BLCOMP的第一个参数声明的）。表中的第一个入口值定义了距电机零位置一个间距时的补偿，第二个入口值则是定义了距电机零位置两个间距时的补偿，如此等等。

在该范围之外时，在补偿被做之前未修正的位置将浮动到该范围之内。该浮动的产生和螺旋补偿表的一样，可以参看螺旋补偿表对那一部分，以获得详细的细节。

常量间隙参数Ix86总是被激活的。如果I51被设为1，则间隙表被激活；如果I51被设为0，间隙表将不被激活。

示例：

假定依靠一台能够在两个方向工作的精确线性测量设备来执行轴的标定，线性设备给设置电机编码器的位置的读入如下（用电机编码器的单位表达）：

| 电机位置(cts) | 0 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 |
|---|---|---|---|---|---|---|---|---|
| 正向位置读入(cts) | 0* | 510 | 995 | 1492.5 | 1994 | 2497.5 | 3003.5 | 3500.5 |
| 负向位置读入(cts) | 5 | 516 | 998.5 | 1494 | 2000 | 2501 | 3010.5 | 3508.5 |

*参考点；定义为零

只有补偿表工作在正方向时，表里的入口值将会是正向变化的读入位置和电机位置的负的差值，用1/16计数位表达：

| 电机位置(cts) | 0 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 |
|---|---|---|---|---|---|---|---|---|
| 读入电机(cts) | 0* | +10 | -5 | -7.5 | -6 | -2.5 | +3.5 | ＋0.5 |
| 电机读入(1/16cts) | 0* | -160 | +80 | +120 | +96 | +40 | -56 | -8 |

*参考点：定义为零

产生这些修正的补偿表定义为：

DEFINE COMP 8，4000
－160 80 120 96 40 －56 －8 0

注意，第一个入口值是给在500计数位的修正的，并且最后一个入口值是零，这是给在4000计数位和零计数位的修正。

在电机零位置有一个5计数位的间隙，因此Ix86将被设置为5×16，或者是80。

间隙表将包含负向读入位置和正向读入位置的差，减去Ix86：

| 电机位置(cts) | 0 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 |
|---|---|---|---|---|---|---|---|---|
| Load(-)-load(+)cts | 5 | 6 | 3.5 | 1.5 | 6 | 3.5 | 7 | 8 |
| Load(-)-load(+)-Ix86 Cts | 0* | 1 | -1.5 | -3.5 | 1 | -1.5 | 2 | 3 |
| Load(-load(+)-Ix86 1/16cts | 0* | 16 | -24 | -56 | 16 | -24 | 32 | 48 |

*参考点：定义为零

产生这些修正的间隙表定义为：

DEFINE BLCOMP 8，4000
16 －24 －56 16 －24 32 48 0

注意，第一个入口值是给在500计数位的修正的，并且最后一个入口值是零，这是给在4000计数位和零计数位的修正。

注意：并不要求间隙表的范围和间距要和丝杆补偿表完全一样。即使是现在已有了给电机的丝杆补偿表，这并不要求给电机一个间隙表。

**力矩补偿表：**

PMAC提供了给伺服环输出产生一个作为电机位置函数的修正的表的能力。通常地，该功能将和伺服环一起用在力矩模式下（不论PMAC有没有正在执行电机换向），所以该功能被称为“力矩补偿表”。力矩补偿表的键入和操作很象提供位置修正的丝杆补偿表。但却没有十字轴或多轴力矩补偿表。属于一台电机的力矩补偿表提供给该电机作为电机位置的函数的力矩修正。

对于以访问的电机，给该电机的力矩补偿表是通过在线命令DEFINE TCOMP{entries},{count length}来声明的。{entries}定义了表里点的数目，而{count length}则定义了在电机计数内的表的范围。在表里入口之间的间距当然是{count length}/{entries}。表中的第一个入口值定义了距电机零位置一个间距时的补偿，第二个入口值则是定义了距电机零位置两个间距时的补偿，如此等等。电机零位置的修正是通过定义设为零的。

修正直接定义在电机位置从0到{count length}的范围上的。对于该范围之外的电机位置，在修正被提供之前将浮动到该范围以内。通过这种方法，象电机齿轮力矩这样的周期干扰就能够得到补偿。在表的结束的修正等于在零位置的修正；因为在零位置的修正被定义为零，所以任何表的最后一个入口值也将是零。

在表定义命令之后，送往PMAC的下一个{entries}常量将被放在表中作为表的入口。在表内该入口的单位是范围为-32,768到+32,767之间的16位DAC的单位，即使是一个不同分辨率的输出设备被用到。在表中入口之间的点的修正将是在表中来自最近的一个值的线性插值。

假如下列表被键入：

```
#1 DEFINE TCOMP 8，2000      ；给电机1的覆盖2000计
                             ；数位的8个入口的表
125   ；在2000/8＝250计数位修正是125 DAC bits
-50   ；在500计数位修正是-50 DAC bits
80    ；在750计数位修正是83 DAC bits
－97  ；在1000计数位修正是－97 DAC bits
60    ；在1250计数位修正是60 DAC bits
－43  ；在1500计数位修正是-43 DAC bits
129   ；在1750计数位修正是129 DAC bits
0     ；在2000计数位修正是0 DAC bits
```

在600计数位提供的修正将是：

修正＝－50+(600－500)÷(750-500)×(83-[-50])=3

# §3－5 设置PMAC换向：

## 介绍：

这一部分介绍了如果PMAC给一台电机执行换向时，如何设立换向方案。如果在你所用的任何一台电机上，都不需要PMAC来执行换向功能，那么你可以跳过这一部分。确认对于你的所有激活的电机，Ix01被设为0，以便PMAC能不给它们换向。

如果你使用PMAC给一台电机换向，你需要告诉PMAC怎样来执行换向。这是通过给电机的I－变量Ix70到Ix83以正确的设置来完成的。这一部分解释了怎样去设置这些I－变量。一旦设置完成，换向操作将自动进行，并且对于用户来说也是可见的。

PMAC对于直流无刷电机、磁阻可变电机、交流感应电机和步进电机来说，有着复杂的在板换向特性。这些算法允许PMAC直接驱动电机的相位，仅仅要求简单的给放大器的当前循环的电桥。作为它的换向反馈设备，PMAC能够利用和那些用在位置伺服反馈中的完全一样的反馈设备(例如一个编码器或一个旋转变压器。

**增量编码反馈要求：**

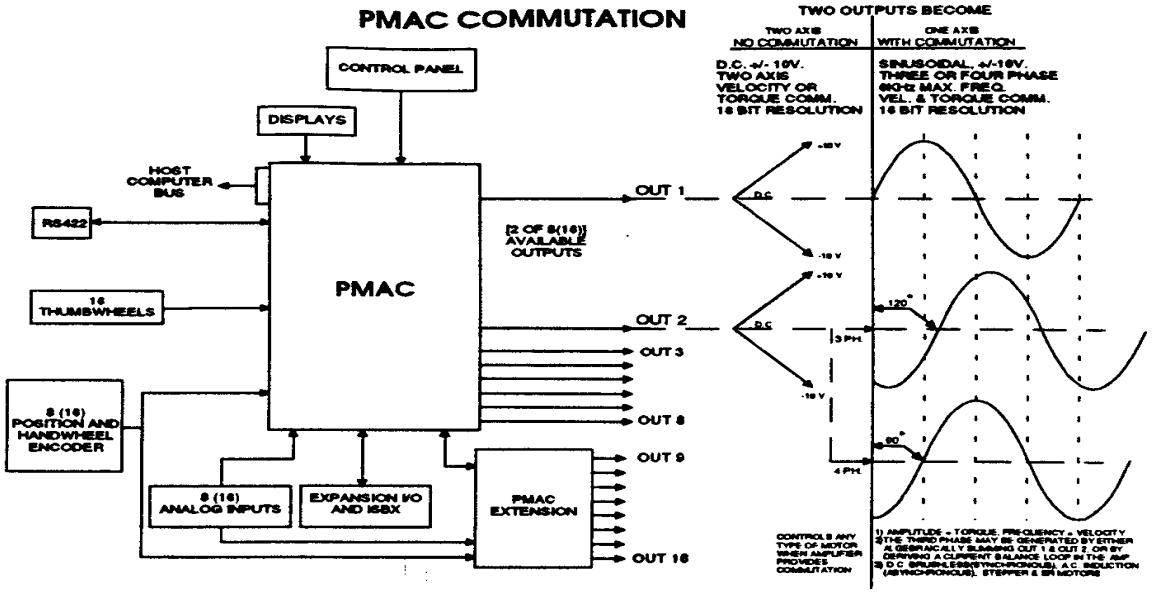
给一台电机正在进行的换向位置信息必须经过一个增量编码计数器。如果换向反馈设备是一个绝对编码器或旋转变压器，则启动信息能够直接从该设备中获得，但是一个增量信号必须采自为正在进行的换向的绝对位置信息。PMAC的附件8D
选件7(R/D转换器板)能够为旋转变压器自动地完成这一步，而附件14上的选项6将为绝对编码器完成这一步。

你必须用Ix83指定保存正在进行的换向位置反馈信息的寄存器的地址。除PMAC微步之外，这几乎总是给在DSPGATE里的编码器的“相位位置”寄存器（地址：X：＄C001，X：＄C005，X：＄C009，等等）。

**相位参考：**

当给一个同步电机进行换相时，例如永磁无刷电机或交换式磁阻电机，必须将相位周期参考电机的物理特性。和前面提到的许多情况恰恰相反，这并不要求一个绝对传感器。如果启动时一个确实可靠的相位搜寻被执行，那么只需用到一个增量传感器。PMAC能够执行这样的一个相位搜寻功能。如果使用了一个绝对传感器，则相位参考只需要在系统装配时执行一遍就可以了。两种相位参考的方式在这一部分都有叙述。

**（PAGE 3－78 PMAC换向示例图）**




**双模拟输出要求：**

如果PMAC正在给一台电机换向，每台电机将要求两个模拟输出通道；如果需要，还可以有第三，第四个；相位通过在放大器里的平衡环而产生。（记住，如果一台多相电机在放大器内部被换相，只要求有一个PMAC模拟输出。）

正如前面在*选择输出*中提到的那样，对于一台PMAC换相的电机，用户必须指定一对相连DAC寄存器

的低地址，该寄存器可能通过Ix02被用来输出相位命令。Ix02的合法值是＄C002（DAC1和2），＄C00A（DAC3和4），＄C012（DAC5和6），＄C01A（DAC7和7），＄C022（DAC9和10），＄C02A（DAC11和12），＄C032（DAC13和14），还有＄C03A（DAC15和16）。

**基本参数规定：**

不论PMAC给哪种类型的电机进行换向，一些参数需要被指定。给电机x的换向参数开始于Ix70。只有当Ix01＝1时，换相参数才能被使用。

每一换相周期的计数位：

首先被指定的是每一换向周期（或电子周期，或极对）的计数的数目。通过电机I一变量Ix70和Ix71即可，其中Ix71/Ix70就是每一周期计数的数目。Ix70和Ix71都必须是整数。译码之后还有编码器计数，所以如果x4译码被用到，则每一编码周期有4个计数位。通常Ix70是1，除非是一些特殊的情况（例如6极电机，它的每一极对不可能有一个整数的计数数目）。

相位之间的角度：

用Ix72可以设置相位之间合适的角度偏差，该偏差对于三相电机和四相电机是不同的。该参数也允许定相的反向，以便如果定相在装配时就错误了的时候，电机的导向柱不会彼此碰撞。Ix72的单位是一个换向周期的1/256，因此对于一台三相电机，可能的值是85（256的1/3）或171（256的2/3）。对于一台四相电机，可能的值是64（256的1/4）或192（256的3/4）。当交换两个电机接线柱时，对于一个给定的相位数字在两个值之间改变Ix72将有同样的影响效果。给每对值中的正确的设置的试验如下所示。

## 永磁无刷电机换向：

当给一台永磁无刷电机（通常称为直流无刷电机，有时也称为交流同步电机）换向时，除前面提到的基本的换向周期参数之外，基本上就没有什么必须详细说明的了。

**验证极性正确：**

为了正确地换向，需要由编码线路决定的反馈极性，需要由放大器、电机线路和Ix72决定的匹配输出极性的编码译码I一变量（I900、I905等等）。也就是说，当PMAC通过它的换向输出发出一个正向的命令时，它必须使编码计数器开始递增计数。在一台永磁无刷电机上如果极性被错误地匹配了，电机将会迅速地“锁紧”，并拒绝移动。

测试极性：

在电机设置的开始阶段的一个快速的测试能够验证极性是否正确。该测试用到输出偏置变量Ix29和Ix79来强制电流进入到一个特定的相位中并象一台步进电机那样来驱动电机。当不同的相位被驱动时，观测电机位置计数的方向，并且当已给出了现在的编码线路和译码变量，以及现在的电机线路和输出相位变量，我们就能够知道极性是否正确。该测试通过输入一些简单的命令，可以在执行程序的终端窗口很容易地被完成。下面举一个例子，其中用到电机1：

```
#1o 0          ；要求为零输出
I129=2000      ；在第一相位正向偏置2000 bits
P              ；要求位置（电机停稳之后）
382            ；PMAC位置响应
I179=2000      ；在第二相位正向偏置2000 bits
P              ；要求位置（电机停稳之后）
215            ；PMAC位置响应
```

极性规则：

I172等于64或85，当第二相位正向偏置加到第一相位正向偏置上（正如示例中所做的一样）时，电机将沿负方向移动。I172等于171或192，电机将会沿正方向移动。作为选择的规定，如果在测试中电机递减计数，I172将会设置成64或85；如果电机在测试中递增计数，I172将会被设置为171或192。

对于这个例子中的电机，我们可以得到结论：如果它是一台4相电机，则我们需要值64，如果是一台3相电机，我们需要值85。如果由于系统的原因，编码器方向被改变了，I172也必须改变，以便匹配。

**加电相位检索：**

如果换向中用到了一个非绝对的传感器，PMAC必须在每一次启动时执行一个搜寻移动，以便得到正确的相位参考（如果使用了绝对传感器，这只需在系统的组装时做一次就可以了）。做这个相位搜寻有好几种方法。PMAC有两种被固件自动执行的方式；其它方式或这些方式的增强可以用PLC程序来执行。

一个加电相位检索允许永磁无刷电机的换向，而不需要一个更昂贵并且精度还低一些的绝对传感器。然而，在某些用途中,却是不能依靠相位检索的；这时就需要一个绝对传感器了。

*警告：一个不可靠的相位检索可能导致一个失控条件。仔细测试你的相位检索方式，确认它在所有假定的条件下都能正常地工作。确认你的Ix11致命随动误差限制被激活了，而且尽可能的适当，以便电机在发生了严重的相位检索错误事件时能迅速地停车。*

两步推测相位检索：

PMAC的第一种自动相位检索方式是“两步推测”相位检索。因为，它将两个任意的假设作为相位位置，用每一个假设简明地提供了一个力矩命令，并观察电机对每一条命令的响应。在这两个响应的量级和方向的基础上，PMAC计算这两个响应，计算正确的参考点。然后在这参考的基础上开始换向，并闭环以保持位置。

两步推测相位检索非常快，几乎不需要什么移动。当外部载荷比较低时（例如重力和摩擦力等等），它能够很好地工作。但是，如果外部载荷很大时，它将不是一种可靠的相位检索方式（不可靠的相位检索方式将是十分危险的）；如果是这种情况，另外一种方式，例如下面将提到的步进电机方式将会被用到。

通过设置Ix80为0或1，可以选择两步推测方式检索。Ix80是0时，在启动/重置周期内相位检索将不会被自动执行；必须要用一个＄命令来执行相位检索。Ix80是1时，在启动/重置周期内相位检索将被自动执行；当然，它也可以用一个＄命令来顺序地被执行。

两个参数必须被指定以告诉PMAC怎样来做这个相位检索。Ix73指定了每一步假设中力矩命令的量级，所用单位是16位DAC的单位。该参数典型的值是2000到6000；4000（大约全范围的1/8）通常是起始点。Ix74设置了每个扭矩命令的持续时间和其响应的估算，用伺服周期的单位。该参数的典型的值是3到10；5（默认的伺服更新大约是2毫秒）通常是起始点。

步进电机相位检索：

同步电机的另一种自动相位检索方式是“步进电机”方式。这种方式强制电流通过特定的电机的相位，就象一台步进电机控制器那样，并且等待它稳定。通过适当的操作，这将是换向周期的一个已

知的位置。

步进电机相位检索比两步推测方式要求更多的移动和时间，但是，在存在很大的外部载荷时，它在搜寻相位的准确方面更加可靠。

通过设置Ix80为2或3，可以选择步进电机方式检索。Ix80是2时，在启动/重置周期内相位检索将不会被自动执行；必须要用一个＄命令来执行相位检索。Ix80是3时，在启动/重置周期内相位检索将被自动执行（我们不推荐用这种）；当然，它也可以用一个＄命令来顺序地被执行。

在这种方式里，Ix73控制通过相位的电流的量级，而32767代表的就是全范围。对于该参数，通常一个接近3000的值（全范围的大约1/10）将被用到，尽管实际的值是要依靠载荷的。

Ix74控制给搜寻中用到的两步的每一个的稳定时间。在这种模式下，Ix74的单位是伺服周期×256，如果是默认的伺服周期的话，这个单位大约是1/10秒。通常用到的稳定时间是1到2秒。

在步进电机相位检索中，PMAC首先强制电流将电机放到换相周期中的＋/－60°的点上，并且等待稳定时间。然后，它强制电流将电机放到换相周期中的0°的点上，并且再次等待稳定时间。它将检测以确认是否在两步之间有至少1/16周期（22.5°）移动。如果已经有了，它将强制相位位置寄存器为0，清除相位检索误差电机状态位，并闭环。如果检测到移动比22.5°少，它将设置相位检索误差位，并使伺服环无效。

如果步进电机相位检索在启动/重置周期之外被执行,那么当放大器出错或超越行程限制条件被检测到时,相位检索算法将会失败。PMAC将会设置相位检索错误位，并使伺服环无效。如果在启动/重置周期之内被执行，PMAC将不能自动地检测这些错误，而检索将可能由于缺乏移动而失败。

常见的相位检索方式：

写下一些常见的相位检索方式可能是必须的。通常这些算法是作为PMAC的PLC程序被执行的，但是它们也常常可以用在线命令来测试和调试。如果相位检索只在开发阶段为了给一个绝对传感器建立参考而被执行，那么在线命令就显得特别有用了。

大多数常见算法在步进电机相位检索方式上都有所改变。它们通过0命令使用相电流偏置值Ix29和Ix79，强制电流进入到特定的相位，以便电机在它的换相周期里将锁定在一定的物理位置。下面的表显示了对于一台三相电机在它的换相周期中通过Ix29和Ix79的不同的组合而产生的位置。通常非零值的量级是2000到3000：

| Ix29 | =0 | <0 | <0 | =0 | >0 | >0 |
|---|---|---|---|---|---|---|
| Ix79 | >0 | >0 | =0 | <0 | <0 | =0 |
| Pos (Ix72=85) | 0°e | 60°e | 120°e | ±180°e | －120°e | －60°e |
| Pos (Ix72=171) | ±180°e | －120°e | －60°e | 0°e | 60°e | 120°e |

例如，下列PMAC执行程序的在线命令的设置能够被用来强制一台电机在它的换相周期里到零位置，设置相位位置寄存器为零，并使电机有效。

```
#1O0        ；使电机有效，且有开环零量级
I129＝0     ；在相位A无偏置
I179＝3000  ；在相位B的正向偏置强制到0°（Ix72=85）
M171＝0     ；向相位位置寄存器写入零
I179＝0     ；在相位B无偏置
J/          ；闭上伺服环
```

键入命令之间的时间将给位置设置提供充分的时间。

下面的PLC程序对于在步进电机相位检索方式上的变量是一个好的开始点。该程序的扩展可用来同时给两个龙门结构的电机定相，或者“步进”直到越出位置限制。该示例用到Ix73和Ix74，就象它们在自动的步进电机相方式中被用到的那样。

```
；**********Set-up and Definitions**********
CLOSE                ;确认所有缓冲区都被关闭
M70->X:$0700,0,24,S  ;24位自动计时寄存器
M271->X:$007D,0,24,S ;2#电机相位位置寄存器

；**********Program to do phasing search**********
OPEN PLC 1 CLEAR
CMD“#200”        ；强制零量级开环
P229＝I229       ；保存实时的相位A偏置
P279＝I279       ；保存实时的相位B偏置
IF（I272〈128）
```

```
    I229＝－I273        ；强制反向偏置进入A
    I279＝I273          ；强制正向偏置进入B
ELSE                ；Ix72〉128
    I229＝I273          ；强制正向偏置进入A
    I279＝－I273        ；强制反向偏置进入B
ENDIF               ；这将强制到60°
M70＝I274×256         ；给向下计时器的起始值
WHILE（M70〉0）        ；等待法定时间
ENDWHILE
I229＝P229            ；为0°恢复实时偏置给A
M70＝I274×256         ；给递减计时器的起始值
WHILE（M70〉0）        ；等待法定时间
ENDWHILE
M271＝0             ；设置相位位置为零
I279＝P279            ；恢复实时偏置给B
CMD“＃2J/”           ；闭上伺服环
DISABLE PLC 1         ；阻止再次执行
CLOSE
```

**给绝对传感器的相位参考：**

如果在电机的至少一个换向周期中有一个位置环是绝对的，那么在电机的启动/重置时不用执行定相搜索就能正确地设置电机的相位是完全可能的。电机在它的换相周期内通过读绝对传感器可以决定它的位置。通过这一方法，相位参考只需要在初始化系统的过程中被执行一次。记住，这对绝对位置的读操作只在电机的重置时间内被执行；正在执行的相位位置总是通过一个编码计数器来读。

I－变量：

通过一个绝对传感器来执行相位检索，有两个I－变量需要正确地被设置。Ix81告诉PMAC绝对传感器的地址和格式。如果该参数比零大，PMAC将在启动/重置时从指定的地址以指定的格式读入绝对位置信息。Ix75则指定传感器的零位置和换相周期的零位置之间的差，所用到的单位是计数位×Ix70。在从绝对传感器读入启动/重置位置之后，PMAC将该值相加，并将结果写入相位位置寄存器。

设立相位检索：

为了设立一个绝对相位，我们首先要在没有用到绝对启动位置时在电机上做一个相位检索。为了用一个增量传感器设立换向的所有方向在这儿得以运用。为了达到最高的精度，这些任务应该用一台

未加载的电机来执行。确信你自己能够在一台使用“步进电机”方式（象手册中显示的PLC程序实施的那样）的电机上执行一个相位搜索，同时确信能够用小的开环命令（例如o 5,o –5）让电机在两个方向上转动。该定相搜索也能够通过如下所示的四条在线命令来执行。

一旦你确信用“步进电机”定相搜索的换向算法工作地很好，那我们就能够决定用绝对传感器定相所要求的定相偏置。首先，我们必须定义一个M－变量来读绝对传感器。对于一个旋转变压器则通过附件8D选件7上的R/D
转换器板来读，这是M－变量的TWR格式。对于一个象绝对编码器那样的传感器是通过并行位来读的，通常是通过一片附件14I/O板，这是M－变量的一个Y格式。例如：

```
M171－>TWR：0,2          ;在附件8D选件7板上多路复
                         ;器地址0和该地址的位置2
                         ;上的旋转变压器
M171－>Y：＄FFD0，0，16，U ;在第一片附件14端口A的16
                         ;位并行绝对传感器
```

接着我们将使用在线命令手动地执行“步进电机”相位检索。该顺序的点将强制电机进入到换相周期的零点。在这一点我们使用定义过的M－变量来读绝对传感器。下面的例子中我们将使用1＃电机。确切的顺序将要依靠在Ix72中的相位角的值。

```
I172＝64（四相）或85（三相）：
＃1 o 0              ；零量级的开环命令
I129＝－200 I179＝2000  ；强制电机到初步位置
I129＝0              ；现在强制电机到相位周期的零位置
                  ；如果你需要估算电流环的偏置（这
```

；种情况下Ix29可以被设成强制零
；位电流通过相位的值），Ix29可以
；被设成非零值。

I172＝192（四相）或171（三相）：
＃1 O 0　　；零量级的开环命令
I129＝2000 I179＝－2000　；强制电机到初步位置
I129＝0　　；现在强制电机到相位周期的零位置
；参见以上的相电流偏置的注释

在这一点，我们通过查询M－变量的值来读绝对位置传感器。

例如：
M171　　；询问M171的值
475　　；PMAC的响应

我们取得PMAC返回的值，将它反号，乘上Ix70的值，并将结果放到Ix75中。下面接着我们的例子，如果I170＝1，我们将发出这样的命令：

I175＝－475

如果I170＝2，我们将发出这样的命令：

I175＝－950

最后的准备：

为了完成我们给绝对相位读入的准备，我们还必须做一些更多的事：通过设置Ix79为零去掉残留的相位偏差；通过设置Ix73和Ix74为零来防止任意的相位检索移动，而且也可删去相位检索PLC程序；通过设置Ix81来定义给绝对相位读入的地址；通过设置Ix80来决定我们是否需要在启动/重置时让电机立即有效。接着上面的例子：

I179＝0　　；删除相位偏置（或给零位电流设置偏置）
I173＝0　　；使定相搜索无效
I174＝0　　；使定相搜索无效
I180＝1　　；使电机在启动重置时立即有效
I181＝＄020100　　；读多路复用器地址0（＄0100）位置2
；（＄02）的R/D转换器以初始化相位位置
or
I181＝＄10FFD0　　；从第一片附件14（＄FFD0）的端口A读
；16位（＄10）的并行数据给初始相位位置

测试绝对定相：

现在我们就能通过发出＄电机重置命令来测试我们的设置。如果相位工作良好，那么我们就能够用小的开环命令轻易地在两个方向上使电机运动起来。如果我们已经合理地调整好伺服环，我们应该也能够使电机在两个方向上运动起来。差的运动性能可能是由于伺服环没有被很好地调整，特别是当开环时工作良好，闭环则性能不行的情况。

保存设置好的值：

一旦我们确认通过＄命令我们能够重复地获得正确的定相，我们就应该用SAVE命令将我们的参数值保存到永久性存储器中去，并且用＄＄＄命令将整个卡重置。如果Ix80被保存为1,那么电机在重置后将会立即被使能并处于闭环位置控制中，即使我们还没有调整好伺服环，电机的刚性可能也会不太好。不管怎样，我们将能够从开环命令中获得良好的响应。

如果Ix80被保存为0，那么在重置期间绝对相位位置将被执行，但是电机将被保持在无效状态。通过开环命令或者jog命令，电机都可以被使能。一个＄命令也可以使电机使能（闭环），同时在进程中将会做另一次绝对位置读操作。

如果在这一点电机在两个方向上都能很好地执行开环命令，那么换向设置已经完成，电机准备进行伺服环调整。

**参考霍耳效应传感器的定相：**

PMAC能够在启动时使用霍耳效应换向传感器或其等效的器件来进行一个逼近的相位参考。该相位参考对于换向周期的±30°是很好的，能够不用任何相位检索就足以得到合适的扭矩和平稳度。当发现标

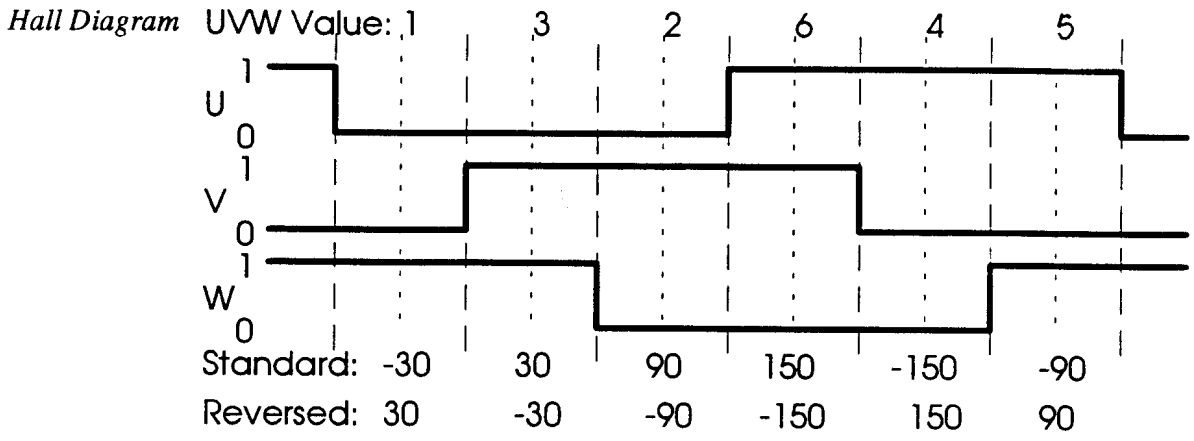
志脉冲时，才能做最后的相位参考。通常标志脉冲是回零位置触发的一部分，所以在寻零运动完成以后，相位位置将基于在开发期间被完成的测量的基础上进行调整。

设置Ix81的第23位为1来指定一个霍耳效应启动相位参考。在这种情况下，Ix81的地址段指定了一个PMAC
X地址，通常另一个未用的匹配第二个DAC的标志寄存器的这一部分输出给电机。例如，一台用到DAC的1和2作为换向的输出的电机将使用处于X：＄C000的标志1作为它的主标志，在X：＄C004的标志2作为它的霍耳效应输入。

PMAC希望在被指定的寄存器的第20、21和22位发现霍耳效应输入。在一个标志寄存器里，这些位分别匹配HMFLn、－LIMn和＋LIMn输入。霍耳效应输入通常标记为U、V和W。U输入是第22位（＋LIMn），V是第21位（－LIMn），而W是第20位（HMFLn）。
霍耳效应信号各自都必须有50％（180°e）的占空比。PMAC能够使用互相间隔120°
e的霍耳效应换向传感器。霍耳效应传感器没有关于方向传感或零参考点的工业标准，因此这必须用Ix81的软件设置来处理。

第22位如下图所示是控制霍耳效应传感器的方向传感的，其中第22位的值为0是“标准”，而值为1则是“反向”。

## （PAGE 3－86 霍耳效应的波形图）




该图显示了零偏置的霍耳效应波形，它是这样定义的，当U信号是低时（当零点在霍耳效应周期之内使定义的）V信号的阶跃代表了PMAC的换向周期里的零点。

如果霍耳效应传感器没有这个定向，Ix81的第16到21位能够被用来指定PMAC的零点和霍耳效应零点之间的偏置。这些位可以被赋予0到63的值，单位是一个换向周期的1/64（5.625°e）。该偏置可以通过执行一个建立没有霍耳效应传感器的相位参考的定相搜索移动来计算，然后在V信号阶跃而U信号为低（霍耳效应零点）时读24位相位位置寄存器（建议用－M变量Mx71给高24位）。这最好是这样来完成，当在示波器上观察U和V信号，或者将－M变量匹配给PMAC执行程序观察窗口且Mx71显示在观察窗口时，使电机无效并且用手来转动它。偏置值可以被计算为：

偏置＝（Mx71％ Ix71）×64÷Ix71

这里计算的偏置值应该被舍入为整数。

在《软件参考》中对Ix81的描述显示了对于所有情况所用到的偏置的通常的值，这些情况包括霍耳效应周期内的零点在0°、60°、120°、180°、－120°或－60°等等，而这些都是制造设备分配给各传器的。

在电机标志脉冲时发现精确的相位位置，执行一个启动定相搜索运动以建立精确的相位参考，然后在

标志脉冲时读入相位位置寄存器Mx71的值，同时通过一台示波器或在PMAC上匹配的M一变量来监测该脉冲。该值能够被存储在相位位置偏置变量Ix75中，该变量在这种模式下不会被自动固件所用到。实际操作中，在引导到标志脉冲后，该值能够被拷贝到相位位置寄存器中。整个的相位参考包括下列步骤：

1. 作为由Ix81指定，使用霍耳效应传感器执行一个粗糙的相位参考，不论是当Ix80=1在启动/重置时自动执行，还是当Ix81=0通过＄命令来执行。

2. 将标志脉冲用作引导触发的一部分，在电机上执行一个寻零移动。

3. 在标志位置使用电机正常位置状态位（建议的一M变量Mx40）等待电机设置“正常位置”（随动误差小于Ix27）。

4. 强制电机位置寄存器到这一点的预定值，通过使用类似Mx71=Ix75这样的命令。

**相位超前：**

一个速度相位超前增益项目（Ix76）允许定相顺序在运动的方向上得以提前，通过给速度指定一个合适的量来抵消计算延迟和在电机与放大器里的滞后。这个功能能够大大地增加电机的最高速度，

并且大大提高系统的能量使用效率。当脱离霍耳效应传感器换向时，这是不可能的。这个参数通常是通过高速开动电机并发现使电流牵引最小的设置来交互地设置的。

## 交换磁阻电机换向：

对于PMAC的换向算法来说，一台交换（可变的）磁阻电机可以和一台永磁直流无刷电机一样对待。差别只是在放大器中。因为一台SR电机的相位不是被直接驱动的，其功率级相比来说可以简单些。

然而模拟预驱动电路必须转换PMAC输出的双向作用特性。首先，附加的相位从独立的命令产生（不是从任何实际的电流信息，就象对直流无刷电机和交流感应电机所要求的那样）。然后每一个相位电流命令信号在被送到电流环之前都必须经过半波整流，因为在一台SR电机中，在换向周期的一半中，任何电流不论是在方向还是在相位上都是和你相抵抗的。

## 交流感应电机换向：

通过间接向量控制的技术，PMAC可以将一台标准交流感应电机作为位置伺服来驱动。PMAC连续地估计转子磁场的方向并给定子相位里的电流定向，从而感应转子电流并且产生扭矩。

其算法和给直流无刷电机的算法是一样的，但是还有两个附加的项目：一是磁化强度（感应）电流的量级（Ix77），它和被估计的转子磁场是保持并行的；差动增益率（Ix78），它决定了被估算的转子磁场角在响应每单位的定子扭矩电流时应前进多少（它和被估算的转子磁场保持垂直）。

**设置差动增益：**

给一台感应电机设置差动增益有一个简单的技巧。差动增益是差频和提供的扭矩之间的一个比例常数。如果我们知道了差频和扭矩之间的匹配值，我们就可以让前者被后者除，还有转换单位，就可以得到差动增益。我们可以从给电机、放大器和控制器的“铭牌”上得到所有我们所需要的信息。

电机信息：

从电机上，我们需要以下信息：

- 一额定（满载）速度，单位：转/分
- 一额定行频率（场频），单位：Hertz
- 一磁极的数目
- 一额定（满载）电流（RMS）

从这些信息，我们将可以计算出作为场频和转子频率之间的差值的电机的额定（满载）差频：

差频(Hz)＝场频(Hz)一额定速度(RPM)÷60(秒/分)×磁极数÷2

我们将在后面用到当前的信息。

放大器信息：

从放大器中，我们需要知道作为电机额定满载RMS电流的一个百分比的最大（超载）RMS电流。

控制器信息：

从PMAC中，我们需要知道：

- 进行换向计算的频率
- 给放大器命令使其输出最大RMS电流所要求的峰值输出（DAC）位的数目
-

涉及差频、提供的扭矩和差动增益的等式里的比例常数第一个信息允许我们将差频转换为每相位更新电子周期的单元。PMAC所用到的差动单位：

差频(周期数/更新)＝差频(Hz)÷相位更新比例（更新数/秒）

给PMAC的相位更新比例是由主时钟频率和跳线E29~E33和E98决定的。默认比例是9.04kHz，或者为9040更新数/秒。

第二条信息允许我们将额定提供的扭矩转换为DAC位的单位。PMAC所用的扭矩单位：

额定扭矩=最大扭矩×额定电机RMS电流(A)÷最大放大器RMS电流(A)

第三条信息允许我们用扭矩去除差频从而得到正确单位的差动增益。求差动增益的等式是：

$$\text{Slip gain}\frac{(2^{38}\,cycles/update)}{(DACbit)}=\frac{2^{38}*slipfreq(cycles/updat)}{Torque(DACbits)}$$

该值将被放入到Ix78中。

示例：

这个方法最好是通过一个示例来说明。假定一个带有下列参数的系统：

电机： 额定RMS电流：20A
额定满载速度：1755 RPM
额定线速度： 60 Hz
磁极数目： 4

放大器： 最大RMS电流： 40A

PMAC： 放大器最大电流的峰值输出：32,767DAC 位（10V）
相位更新比例：9.04kHz（9040更新数/秒）
比例常数：$2^{38}$

从这些数据，我们可以计算上述等式：

差频(Hz)=60－$\frac{1755*4}{60*2}$＝1.50Hz

差频(周期数/更新)=1.50(Hz)÷9040(更新数/秒)=$1.659\times10^{-4}$

额定扭矩＝32768DAC bits×(20A/40A)=16384DAC bits[$2^{14}$]

差动增益($2^{38}$周期数/更新)÷(DAC bit)＝$2^{38}\times1.659\times10^{-4}\div16384$＝2784

Ix78将被置为2784。

**设置磁化强度电流：**

注意：这一部分所描述的方法可能涉及到可致命的高电压测量，确保你与电源电压绝缘良好来测量电压。在你使用该方法之前，确认你已经正确细致地理解了测量交流电压量级的技术。

正确的磁化强度电流的确定最好是通过一个简单的实验方法来完成。该方法依赖于电机速度和back-EMF之间的线性关系，在这其中，磁化强度电流提供了比例常数（$K_E$，电压，或back-EMF常数，都直接和磁化强度电流成比例）。

考虑电机的空载速度（因为空载时只有零差动，故等于行频率）。对于一台60Hz的4相电机来说，空载

速度是1800RPM。在这个速度上，空载时，从back-EMF来的电压波形应该刚好在饱和点（例如380VRMS）。

一般给定一个为满载电流的5~10％的启动电流（Ix77=1638到3267）。根据上面的等式来设置差动增益（Ix78）。当它输出零扭矩时（使用开环命令**O**），我们将在空载速度的已知部分测量电机的back-EMF。或者从另一台电机来以一个已知的速度驱动一台电机，或者用小开环扭矩（例如，**O**5）来取得在该速度上的电机。然后命令**O**0，并且，象电机“惯性追踪”一样，通过要求的速度来测量。

如果测得的电压太低，则磁化强度电流太低。如果测得的电压太高，则磁化强度电流太高。该差值是成比例的：如果电压只有它应该有的75％，那么磁化强度电流也只有它应该有的75％。根据你的测量来调整Ix77，然后再使。在第二遍时你将基本正确了。

**感应电机参数的实验设置：**

如果你对用感应电机铭牌值来设置感应电机参数并不熟练，或者需要检查一下从铭牌值来的参数是否正确，那么下列的实验方式将会被用到。该方式应该用在一台空载的电机上，因为它使用了会引起大量难以控制旋转的开环命令。来自一台空载电机的实验设置即使在加载时也是有效的，因为感应电机参数是被独立地读入，是只和电机电子性质有关的功能。

注意:在该实验中所得到的值主要依赖于电机转子的温度，因为转子的阻抗，还有它的R/L时间常数，都随着温度的变化而变化。因此最好用转子的发热来优化设置。在这种情况下，由于转子变冷电机的运行效率将会变低，因为这将要求更多的电流来使电机升温。在你将要做最后的优化之前，你需要先以一定的电流（例如，o 30）将电机运行几分钟。

步骤1：给Ix77磁化强度电流任意选择一个值。如果你有一个从铭牌值算出来的数，那么就用它。如果你没有任何已计算好的值，那么3000（大约最大值的1/10）将是一个好的起始值。

步骤2：给Ix78差动增益任意选择一个值。如果你有一个从铭牌值算出来的数，那么就用它。如果你没有任何已计算好的值，那么4000将是一个好的起始值。

步骤3：当用一个开环命令加速电机时，从电机采集实际位置数据。用PMAC执行程序来选择实际电机位置的采集。可用下列顺序的在线命令来执行实际的采集。

```
GAT O10          ；开始数据采集，开环10％命令
ENDG             ；停止数据采集
O0               ；开环0％，使电机停止
```

不要直到电机已经停止加速时才发出ENDG命令。用F10键，将数据送给PC。

步骤4：在屏幕上绘出速度—时间图。从它的斜率计算出加速度。

步骤5：按10％的比率减小Ix78差动增益，并重复步骤6和7。如果该图离开零速度的加速度比第一张图的大，那么继续减小差动增益。如果比第一张图的小，那么从初始值按10％的比率增大Ix78，并重复步骤6和7。

步骤6：继续更新Ix78差动增益，直到你得到一个能提供最大加速度的值。当你闭环之后，给Ix78以越来越小的变化，直到在电机的响应里没有显著的变化为止。

步骤7：将Ix77和Ix78的值乘到一起。这个结果就是给你的电机的最优化值（至少是在它现在的温度下）。

步骤8：在能够提供最大加速度的那张图上标出最大速度。该加速度由于你的电机上back-EMF而停止，back-EMF和你电机上的同供应电压相匹配的Ix77磁化强度电流是成比例的。如果你想要获得比现在高的速度，你必须按反比关系减小你的Ix77的值，从而增大速度。如果你想要使你的速度加倍，你必须将Ix77的值减小为它现在值的50％ 。

如果你能够容忍一个比较低的最大速度值，并且你想在低速时得到较大的扭矩，那么你必须增大Ix77的值。如果你改变了Ix77的值，那么你必须按照反比关系改变Ix78的值，以便Ix77×Ix78的值能够保持为常数（新的Ix78＝理想值/新的Ix77）。

## 开环微步换向：

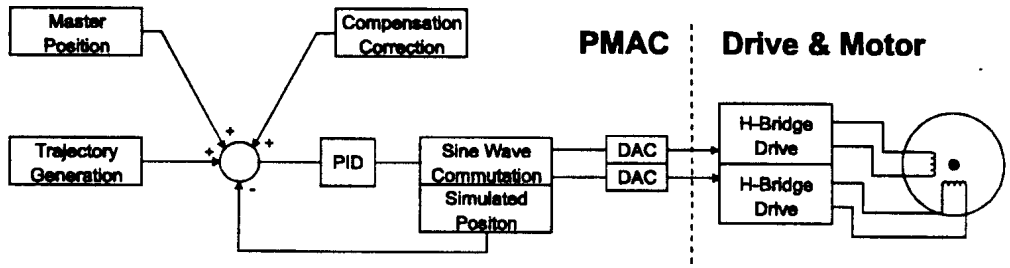
PMAC能够执行标准步进电机的开环微步（直接微步），并能给换向和伺服算法逐渐清除内部生成的假反馈。

该技术和使用一片带伏频转换器的PMAC来命令一个外部微步驱动是不同的，该技术根本就没有用到PMAC的换向算法。

当进行微步时，PMAC提供了用作给电机相位的电流命令的两路模拟输出。对于一台微步电机来说，这两个相位是电子上独立的，并且相位上相差90°。在这种情况下，这两路输出仅仅是给驱动每个相位的H－桥式放大器的双向电流命令。这些放大器可以是简单的扭矩模式（电流模式）直流无刷电机放大器。

PMAC的微步算法在一个电子周期内提供了256微步，其中是64微步/步。在一台典型的200－步/转的电机上，这个数目可以到12，800微步每转。使用默认的相位更新频率9KHz时，PMAC能以576，000微步/秒回转（9000全步每秒）。当用到的电机数目比较少，并且（或者）用速度比较快PMAC时，可以使用更高的相位更新频率。

**（PAGE 3－92 PMAC/PMAC2直接微步系统图）**




**设置I－变量：**

为微步建立一台电机只需要根据下列列表设置电机I－变量即可。因为没有反馈，所以也就没有调整的必要。

**a. Ix01**：设置为1使PMAC换向有效

**b.** **Ix02:**

设置第0到15位为你希望用到的DACs输出对的低地址（＄C002给DAC1和DAC2，＄C00A给DAC3和DAC4）。设置第16位为1，告诉PMAC该电机微步运行。例如 **I102＝＄1C002**。

**c.** **Ix03，Ix04:**

将这两个变量放到处理“相位位置”寄存器内数据的编码器转换表中的寄存器中。表中的这个入口应该设置如下：

第一列 Y-寄存器：＄600041（＄60是并行的X字资源；＄0041是电机1的“相位位置”）

Use $60007D --- 电机 2;
Use $6000B9 --- 电机 3;
Use $6000F5 --- 电机 4;
Use $600131 --- 电机 5;
Use $60016D --- 电机 6;
Use $6001A9 --- 电机 7;
Use $6001E5 --- 电机 8。

第二列Y-寄存器：＄0000FF（只用低8位）

记住将Ix03和Ix04指给该入口的第二行。例如：
WY720，＄600041，＄0000FF 设置了转换表入口；
I103＝＄721并且I104＝＄721指该转换表（＄721＝1825decimal）

**d. Ix08，Ix09:** 设置比例因子为32。

**e. Ix30:** 设置该比例增益项为8192。

**f. Ix31:** 设置微分项为0。

**g. Ix32:** 设置速度前馈项为65,536。

**h. Ix33:** 设置积分项为0。

**i. Ix35:**
设置加速度前馈项为65，536。如果你在手动中得到一个跟随误差（可能只在一台高编号的电机中会遇到），那么增加Ix35的值（通过65，536×相位更新时间/伺服更新时间），通常[65，536×1/4]，得到81，920。如果仍然又跟随误差，用同样的增量再次增加（通常产生98，304）。

**j. Ix69:** 设置DAC输出限制为524，287。

**k. Ix70，Ix71:**
设置Ix70为1，且设置Ix71为256，以提供256个计数位（微步）/电子周期（64微步/步）。

**l. Ix72:**
通常的两相微步电机的相位角参数设为64或192。在这个值之间的改变将会改变正向旋转的方向传感。如果你想要让一台三相电机微步运动，那么使用85或171。

**m. Ix77:**
设置“磁化强度电流”参数来控制在该相位中使用的电流的量。该变量的值保存了用来控制DAC输出（给放大器的电流命令）的DAC
bits的最大数目。例如，值16，384对应于每一个相位的＋/－5V的正弦输出。

**n. Ix78:**
设置“差动增益”参数等于4,194,304/N，其中N是每伺服周期中定相周期的数目，是通过E3~E6来设置的。其默认跳线设置N的值为4，所以在默认设置下Ix78将被 设置为1,048,576。

**o. Ix83:**
设置电机的相位地址参数，电机1为＄42，电机2为＄7E，电机3为＄BA，电机4为＄F6，电机5为＄132，电机6为＄16E，电机7为＄1AA，电机8为＄1E6。

**电机的使用：**

一旦你设置好了电机，你就可以象用任何别的PMAC电机一样来使用该电机。实际上，因为PMAC能虚拟内部反馈，你可以给这台“电机”编程或测试，而不需要加上任何实际物理上的电机。

## 用户写的换向算法：

对于那些有着不寻常或困难的换向需求的高级用户来说，PMAC提供了与常见的用户写的换向（定相）算法的接口。这些程序是用Motorola
56000汇编语言代码写的。通常是在一台PC机或兼容机上，并且能给56000进行交叉汇编。***Delta Tau***提供了以下信息：

到何处采集需要的信息，在哪里留下输出命令，还有在哪里存储算法本身。（注释：这些对于一个不熟练的用户是不要求的。因为，想要做到这些，用户必须对电机理论和汇编语言编程都非常熟悉。）用户写的换向算法是通过让电机x的变量Ix59置为2或3从而使之有效的（Ix59＝3也让用户写的伺服程序有效）。在启动/重置时，PMAC将只能在标准换向算法和用户所写的换向算法中选择一个，所以为了改变PMAC所用到的算法，Ix59的值必须被改变。而Ix59的值必须用SAVE命令存储到掉电保护的内存中去，并且整个卡必须被重置。

**存储空间，软件界面和程序节流：**

分配给用户写的换向算法的程序空间是：

- 程序代码起始地址 P：＄BB00

- 最大的连续程序长度是256个24位字（P：＄BB00到P：＄BBFF）。通过跳转,指令保留给用户使用的别的程序空间（P：＄8000~P：＄BAFF）也能够被使用。编译过的PLC代码，如果是当前正使用的，将从P：＄8000开始；用户写的伺服程序，如果也是正使用的，将从＄B800开始。在该范围内未用作这些目的的存储器将分配给用户所写的换向算法。

在用户所写的伺服算法中用到的变量所利用的数据空间是：

- 初始化为零值的寄存器 L：＄0770到L：＄077F
- 未初始化的用户寄存器 L：＄07F0到L：＄07FF。这些寄存器用电池保证的方式保存了在断电/重置之前的写入它们的最后值；然后在启动时将最后的值保存到PMAC的闪存中。
- 用DEFINE UBUFFER 命令保留下来的寄存器；从L:＄9FFFF起，地址递减到缓冲区声明的长度。

用户所写的换向算法必须直接进入到存储器和存储映射的I/O寄存器中。不象用户所写的伺服程序，用户所写的换向算法没有特殊的数据写入内部DSP寄存器，或从内部DSP寄存器中删除。在用户换向算法中用到的普通寄存器是：

- 伺服滤波器结果：X：＄0045,X：＄0081等等。
- 编码器相位位置寄存器：X：＄C001,X：＄C005,等等。

在用户换向算法中必须遵循下列限制条件：

- 代码必须用ORG P：＄BB00起始
- 不能假设入口上的DSP内部寄存器的状态
- 不能使用堆栈
- 如果使用R，M和N寄存器，它们必须在退出之前存储。
- 退出时B一累加寄存器必须被清除（只是PMAC1要求）
- 代码必须用一个RTS指令结束

# §3—6 闭合伺服环

PMAC自动闭合所有活动电机的数字伺服环。伺服环的目的是为了产生一个使电机的实际位置逼近所要求的位置的输出。它的效果依靠伺服环滤波器的调节--参数的设置—
和被控制的物理系统的动力学性能。

## 伺服更新率

伺服环以一个由主频，跳线E98，跳线E29-E33(对主频分频来产生相频率)，跳线E3-E6(对相频率分频来产生伺服频率)和参数Ix60(用软件扩展伺服频率)决定的频率闭合(更新)。在降低或升高某个电机的伺服频率时，可用Ix60;它还可用于快速地检测在一个较低的伺服频率下对所有的电机你是否能得到满意的性能;另外，它能使伺服频率降至1KHZ以下。但使用跳线降低所有电机的伺服频率是更有效的方法。

**增加伺服频率的原因**

在你的系统中伺服环应有多快的频率?对于大多数应用，缺省的配置是442毫秒。改变这个时间的基本原因有两个。第一，如果你得不到你想要的动态性能，你应该加大伺服更新率(降低更新时间)，在大多数系统里，一个更快的更新率意味着一个更具有刚性和响应性的伺服环闭合，它的误差和滞后较小。

**降低伺服频率的原因**

第二，如果你的优先级低于伺服环的例程执行的不够快，你可以考虑降低伺服更新率(增加更新时间)。要得到你需要的性能的更新率可能要低于你设置的。如果是这样，你正在为不需要的额外更新上浪费处理时间。例如，把伺服更新时间从442毫秒提高到885毫秒，增加一倍，实际上加倍了运动和PLC程序执行的可用时间，允许更快的运动处理速度和PLC扫描速度。

某些系统在低的伺服更新率下反而能得到更好的性能。通常这些系统带有分辨率相对低的编码器，一般仅在负载上的编码器由于微分增益不足以给出足够的阻尼，放大器的量化误差会导致不稳定的高频颤动。此时，降低更新率(增加更新时间)有助于给出足够的阻尼以防止过大的量化噪声。

**改变伺服频率的矛盾**

如果你改变伺服更新时间，许多已有的伺服增益Ix30到Ix39的作用将发生变化。为保持不变的伺服性能，你必须改变这些值。参考软件说明，I变量描述中的Ix30-
Ix35的细节。参考以下的陷波滤波器部分，重新计算陷波滤波器参数Ix36-Ix39。

如果你用跳线改变了伺服更新时间，你必须改变参数I10，以正确的速度得到轨迹。当改变Ix60时，不一定要改变I10来匹配这个变化。

## 放大器类型

PMAC可与多种类型的放大器连接。每种特殊电机使用的放大器在伺服环的调节上有重要的差别。下面将阐述每种通用的类型。

**速度模式放大器**

许多放大器接收从控制器送来的速度命令，和从电机传感器送来的速度反馈信号--
通常由一个转速计或解算器综合得到。使用这些放大器的电机由放大器闭合它们的速度环，而不需要使用PMAC速度环的微分增益，使速度环得到较好的调节。在这些系统中，PMAC的模拟输出表示一个速度命令。这些放大器还在内部闭合电流环，并且如果电机是无刷的，它们执行换向操作。

速度模式放大器的关键优点是在闭合一个模拟速度环时，它们不受量化误差和数字速度环采样频率的限制。这样它们常可获得较高的速度环增益，得到较高的硬度和好的抗干扰能力。由于这一点，它们被广泛地用于机床切削，以在很大的切削力下保持精度。因为这些高增益，PMAC的位置环比例增益要比其它类型的放大器低得多。

但是，大部分速度环的硬度来自于速度积分增益。积分增益会导致一个滞后，这会使它对外部的命令反应缓慢。因此，这些放大器不适用于要求启动和停止快的功能，如许多导引功能的使用。

随着DSP的处理速度的增加和时钟频率的提高，利于克服采样率的限制，以及应用1/T等方法改进数字速度预估技术，减小量化误差，速度模式的放大器将使用的越来越少。

在调节PMAC的位置环之前，调节带着速度模式驱动器驱动的负载的速度环是重要的。因为速度环的调节与负载有关，放大器制造商不可能完成最终的调节；机器制造者必须调节环路。步进的速度响应

不能有任何明显的超调或振荡:如果有，那么要想用PMAC来闭合一个性能好的位置环是不可能的。PMAC执行程序中有一个叫做"开环调节"的功能可用来为放大器给出单步的速度命令，并可观察屏幕上画出的响应。这使调节放大器或仅确定它是否得到较好地调节变得容易了。

**扭矩模式放大器**

另一种普遍的放大器类型是扭矩模式放大器，在这里，控制器给出的模拟电压表示一个给电机的扭矩命令(或一个给直线电机的力命令)。由于电机原理公式表明扭矩与电机电流成正比，它通常也被叫做电流模式或电流环放大器(必须闭合一个电流环以确定输出扭矩与输入电压成正比)。另一个少用的名称是“跨导”放大器，意味着一个电压输入导致一个与之成正比的电流输出。如果是无刷电机，这些放大器也能执行换向操作。

牛顿第二定律指出扭矩或力分别与转动或直线加速度成正比，所以输入这些放大器的命令实际上是加速度命令。在放大器内没有速度环闭合，所以要靠PMAC自己闭合速度环来得到一个稳定系统所需的足够阻尼。对于标准的PID滤波器，这是由微分增益完成的，所以这些系统要有稳定的响应，有足够的微分增益是重要的。

有许多原因使扭矩模式放大器被普遍使用。由于它们不需要一个转速计或模拟的速度环电路，所以比较简单，成本也低。因为电流环增益仅仅与电机的特性有关，而与负载无关，对于要使用的特定的电机，它们可被制造商预先调节好。当电机与负载连接后，机器制造者不需要进行再调节。

另外，扭矩放大器在那些要求加速和减速快的系统中工作良好。它们的性能与误差积分器没有太大关系。而后者会引入滞后和对速度变化的响应慢等缺点。正由于如此，即使没有成本上的优势，它们也比速度模式放大器应用要更广泛。

**电压模式放大器**

电压模式放大器是最简单和最便宜的放大器。一个电压命令输入导致给电机的一个成比例放大的电压输出。放大器本身没有任何种类的反馈。但由于一些原因，它们不适用于工业位置控制。

首先，一个送给电机的电压命令必须克服产生电流的电机的电气时间常数。这在电机响应中会导致延迟。其次，即使在延迟之后,导致产生电流的电机的电压命令仅仅是在电机返回的EMF以上的部分，因此产生的扭矩与电机的速度有关。由于以上的问题，它只适合与要求较低的数字位置环和速度环连接。

最后，直接控制电流是防止产生过电流的最好方法。大多数工业伺服电机在电源返回的EMF没有富裕时满载工作几个毫秒，就会烧毁它们。如果过电流切断保护需要电流传感器，它们也能用于以非常小的额外代价闭合电流环。

电压模式的放大器的性能介于速度模式放大器与扭矩模式放大器之间。在低载情况下，速度被电机的返回EMF所限制;在低速情况下，电流被电枢电阻所限制。从控制者的观点来看，电机通过它自身的EMF获得了一定的阻尼，但对于大多数位置控制应用来说是不够的，必须由控制者提供微分增益加以补充。

**正弦输入放大器**

一种适用于无刷电机--包括永磁式和感应式电机--
的放大器，自身不进行换向，而依靠控制器如PMAC来完成。它需要两路从控制器得到的模拟相电流命令。在常速和常载的情况下，这些命令是正弦波的形式，所以这种放大器有时被称作“正弦波输入”放大器。

这种放大器仍然闭合电流环，并且在有必要平衡环路时还可生成第三和第四个相位命令。对于PMAC的伺服环，这种类型的放大器可看作扭矩模式的放大器，正弦波的幅值与扭矩命令成正比。伺服算法产生一个单独的扭矩命令；但是，并不是直接将这个命令值作为一个模拟输出，而是PMAC通过换向算法进行处理，产生两个模拟输出。

这种类型的放大器有许多优点。首先，它允许使用PMAC高性能的换向算法，通常比放大器换向提供更好的性能。其次，由于PMAC具有相位搜寻功能，对于同步电机(永磁式和开关磁阻式电机)，它允许使用成本较低的位置增量式传感器。第三，由于仅仅是控制器需要反馈，而放大器不需要，它的连线减少了。

最后，在反馈失去时，它提供一个安全的失效模式。当一个伺服算法失去反馈时，它会输出一个导致失控的大扭矩命令。但是，当一个换向算法失去它的反馈时，它要么象一个步进电机那样被锁住(对于一个同步电机)，要么至少不会产生一个大扭矩(对于一个异步--感应--
电机)。如果伺服和换向算法使用连接在同一条线上的同一个反馈传感器，当一个失去反馈时，另一个也会失去，而产生一个更加良性的失效条件。

**脉冲和方向放大器**

大多数步进电机放大器以及一些“步进替换”伺服放大器接受脉冲和方向输入，此时，每个脉冲为一个位置增量。PMAC可通过一个电压-频率(V/F)转换器如附件8D option 2板把它的模拟命令电压转换成这种命令格式。在这些系统内，同任何速度环，电流环和电机的换向一样,真正的位置环在放大器内被闭合。PMAC必须使用作为反馈产生的脉冲列为这些电机闭合一个假的位置环。由于输出命令是频率，或位置改变速率，它实际上是一个速度命令，而且环路可象一个速度环放大器那样调节。附件8D Opt 2 V/F 板的手册有每种频率的优化增益设置列表。

**液压伺服放大器**

液压伺服阀门产生一个与控制节流开关的命令电压成正比的压力或力。所以，对于控制者的伺服环来说，它看起来象一个需要微分增益来稳定的扭矩模式放大器。一些制造商使用较低廉的液压阀门。这些阀门和伺服阀门比较有实际的交接死区。这些死区可以被PMAC的死区补偿Ix64和Ix65补偿至某些范围，但一个放大器的物理限制还是存在的。

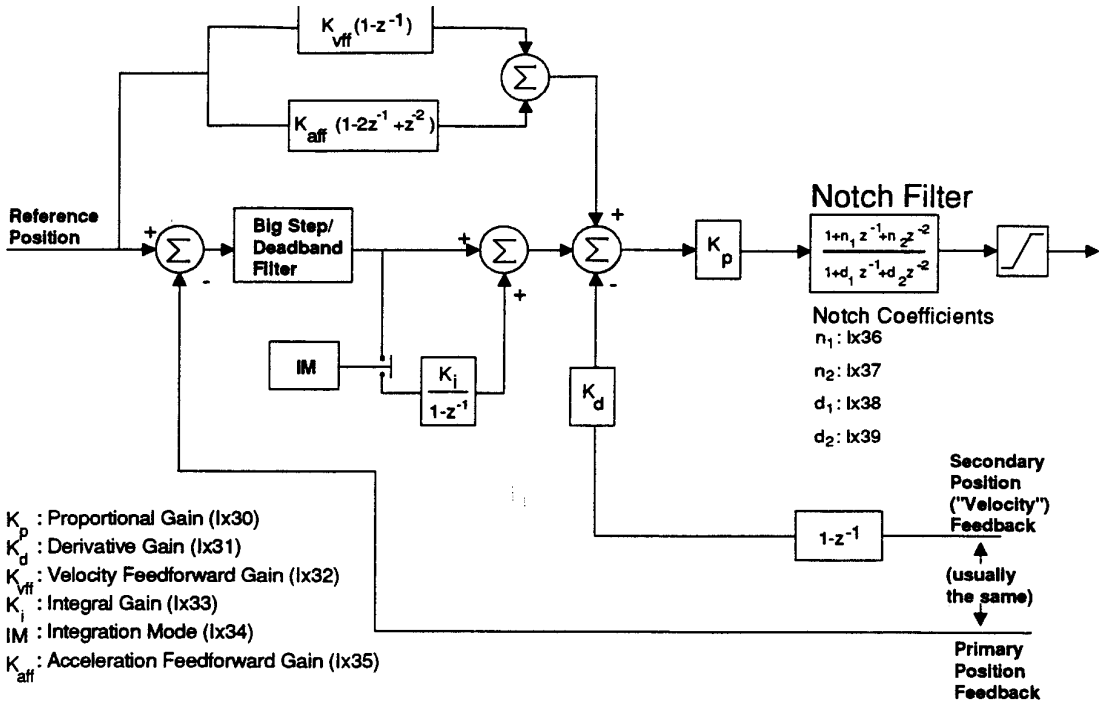
液压电机放大器可以是扭矩模式或速度模式，取决于它们是否使用一个速度传感器并自身闭合一个速度环。对于PMAC的伺服环，这些放大器看起来就象是电磁电机使用的同一种放大器。

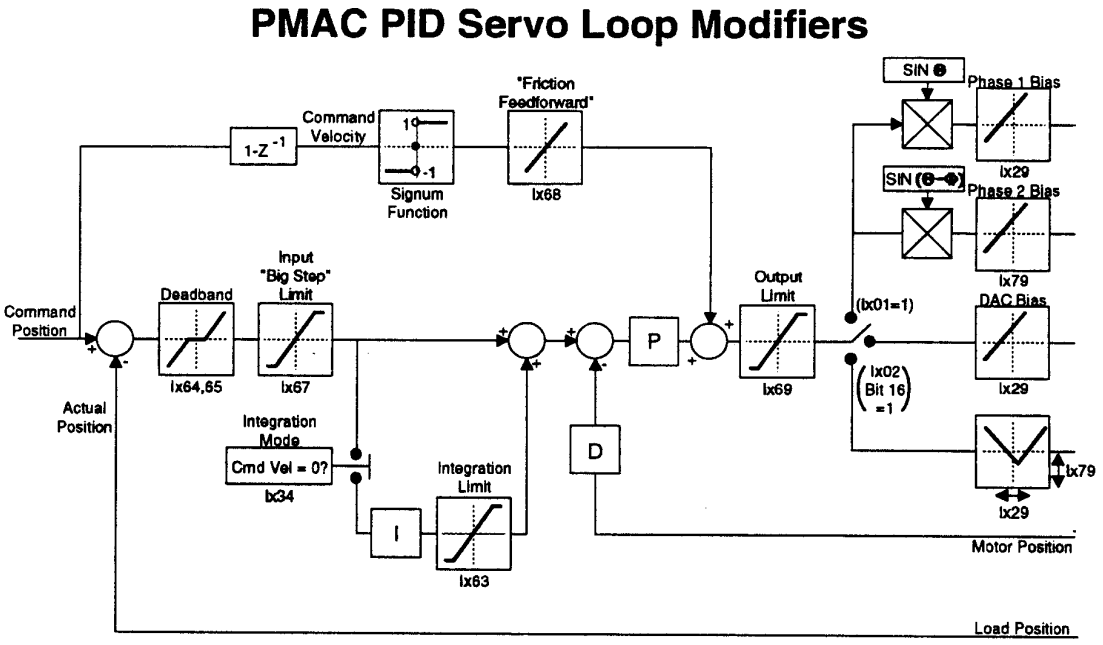
## PID伺服滤波器

PID滤波器工作原理

标准的PMAC控制器提供一个PID位置环伺服滤波器。大多数用户将发现该滤波器足够控制他们的系统，而且即使那些不是专家的人也易于理解。滤波器是通过设置每个电机的适当的I变量来调节的。比例增益(“P”--Ix30)提供系统的硬度;微分增益(“D”--Ix31)提供稳定需要的阻尼;积分增益(“I”--Ix33)消除稳态误差。Ix34决定积分增益是全程有效还是只在控制速度为0时才有效。另外，速度前馈增益(Ix32)减小由于阻尼(与速度成正比)引入的跟随误差，加速度前馈增益(Ix35)减小或消除由于系统惯性(与加速度成正比)带来的跟随误差。

**PMAC PID + 陷波伺服滤波器(图5)**




**调节PID滤波器**

适用于PC及其兼容机的PMAC执行程序提供了一个调节PID滤波器的容易的方法。它允许运行标准运动的简单命令，收集响应数据，把这些数据画在屏幕上，并为响应计算重要的统计数据这就允许没有经验的用户通过简单的规则(由PMAC提供)来作出判断，优化调节。在执行程序手册中有详细的指导和示例。

执行程序还有一个"自动调节"功能来激励电机，评估响应，并为目标响应计算所需的增益。该功能允许计算机为得到适当的增益作出所有的决定。

**实际的PID算法**

对于电机X使用的计算控制输出的PID算法的实际公式如下所示:

$$DACout(n) = 2^{-19} * Ix30 * [\{Ix08 * [FE(n) + (Ix32 * CV(n) + Ix35 * CA(n)) / 128 + Ix33 * IE(n) / 2^{23}]\}$$
$$-Ix31 * Ix09 * AV(n) / 128]$$

注释:

DACout(n)为16位的伺服周期输出命令(-32768到+32767)。它被转换成-10V到+10V的输出。DACout(n)的值由Ix69定义。

Ix08为电机X的一个内部位置放大系数(通常设为96)。

Ix09为电机X速度环的一个内部放大系数。

FE(n)是伺服周期n内所得的跟随误差，即为该周期内命令位置与实际位置的差值[CP(n)-AP(n)]。

AV(n)是伺服周期n内的实际速度，即为每个伺服周期最后两个实际位置的差值[AP(n)-AP(n-1)]。

CV(n)是伺服周期n内的指令速度，即为每个伺服周期最后两个指令位置的差值[CP(n)-CP(n-1)]。

CA(n)是伺服周期n内的指令加速度，每个伺服周期最后两个指令速度的差值[CV(n)-CV(n-1)]。

IE(n)是伺服周期n的跟随误差的积分，大小为:

$\sum_{j=0}^{n-1}[FE(j)]$(在所有的伺服周期内积分都起作用。当CV不等于0时，Ix34=1只关掉了积分器的输入，而没有关掉它的输出。)

## 陷波滤波器

PMAC可用来设置陷波滤波器。陷波滤波器是一个防谐振(带阻)滤波器，用于抵销共振的影响。如何设置陷波滤波器有很多不同的基本定律。我们推荐设置一个轻阻尼的带阻滤波器，其中心频率约为共振频率的90%，和一个大阻尼的带通滤波器，其通频带要高于共振频率(为了减小滤波器本身的高频增益)。

对于那些常见的控制理论(对滤波使用并不必要)，PMAC陷波滤波器系统的形式为

$$\frac{N(z)}{D(z)} = \frac{1 + N1 * z^{-1} + N2 * z^{-2}}{1 + D1 * z^{-1} + D2 * z^{-2}}$$

分子N(z)为带阻滤波器，分母D(z)为带通滤波器。陷波滤波器对PID滤波器本身的输出起作用。

PMAC使用四个变量指定整个陷波滤波器系统:两个(Ix36[N1]和Ix37[N2])是带阻滤波器参数，两个(Ix38[D1]和Ix39[D2])是带通滤波器参数。这些I变量代表陷波滤波器公式里使用的实际系数。它们的范围为-2。0到+2。0;它们是带一1个符号位，2个整数位和21个分数位的24位值。

当你在PID附加算法内执行陷波滤波器之前，你应当把PID参数调节至获得最简单的性能，即使这样会导致振荡难以控制。

**自动陷波指定**

PMAC执行程序允许你非常简单地设置一个陷波滤波器，而不需要了解一个陷波滤波器是如何工作的。最简单的方法是仅需要输入你希望控制的机械的共振频率。执行程序将自动计算所需的带通和带阻滤波器的特性，及它们的系数，并送给PMAC。或者，你也可以单独地指定带通和带阻滤波器需要的特性，PMAC将计算得到这些特性所需的系数，并下载它们。欲详细了解请参考执行程序手册。

**手动陷波指定**

一些用户希望自己来计算陷波滤波器的特性和系数。步骤很明了，但它需要许多计算，如下所示:

标识一个陷波滤波器

一个简单的带通或带阻滤波器可仅由两个参数来标识。但不同的分析途径使用的参数不同。通常使用的参数为:

—自然频率($\omega_n$):无阻尼时的中心频率;等于$\omega_d$ /sqrt(1-b$^2$)

—阻尼频率($\omega_d$):有阻尼时的中心频率;等于$\omega_n$ *sqrt(1-b$^2$)

—阻尼系数(b):范围由0(无阻尼)到1(临界阻尼);等于1/2Q

—Q因子(Q):范围由1(临界阻尼)到无穷(无阻尼);等于1/2b

计算S平面根:

一旦你决定了你的带通和带阻滤波器的参数，你必须计算滤波器S平面根的位置。可由以下两个公式得到:

$s_{real}$=-b*$\omega_n$=-b*$\omega_d$/sqrt(1- $b^2$)

$s_{image}=\omega_d=\omega_n$*sqrt(1- $b^2$)

等式中使用的频率的单位为弧度/秒;它可由HZ*2*Pi(6。283)得到。

计算Z平面根:

接着，我们必须使滤波器数字化，通过$Z=e^{sT}$得到Z平面的根，在这里，T为伺服采样时间。分别计算Z的虚部和实部:

$z_{real}$=exp($s_{real}$*T)*cos($s_{imag}$*T)

$z_{imag}$=exp($s_{real}$*T)*sin($s_{imag}$*T)

计算实际系数:

之后，我们计算数字陷波滤波器的系数。将两个根相乘(共轭复根):

(z- $z_{real}$- $z_{imag}$)*(z- $z_{real}$+$z_{imag}$)

=$z^2-2*z_{real}+(z_{real}^2+z_{imag}^2)$

=1-2*$z_{real}$*$z^{-1}$+($z_{real}^2+z_{imag}^2$)*$z^{-2}$

=1-2*$z_{real}$*$z^{-1}$+exp($s_{real}*T)^2$*$z^{-2}$

第一个系数为-2*$z_{real}$，第二个系数为exp($s_{real}*T)^2$或$z_{real}^2+z_{imag}^2$。

直流增益:

现在，计算该滤波器的直流增益。将z=1带入表达式:

DC gain=1-2* $z_{real}$+$z_{real}^2+z_{imag}^2$=1-2* $z_{real}$+ exp($s_{real}*T)^2$

(如果该滤波器的表达式在分母上--为D(z)--则直流增益为计算所得的倒数。)

这些直流增益值是很重要的，因为我们不希望加上陷波滤波器后改变滤波器的总的直流增益。我们将改变比例增益来进行补偿。

I变量值

现在，我们把计算所得的值赋给适当的I变量:N1(Ix36)或D1(Ix38)表示$z^{-1}$的系数;N2(Ix37)或D2(Ix39)表示$z^{-2}$的系数;新的比例增益(Ix30)等于陷波滤波器直流增益除以旧的比例增益。Ix36-Ix39为24位数字量，范围从-2.0到+2.0。已有的比例增益必须同陷波滤波器直流增益的倒数相乘以保持整个滤波器的硬度不变(以后你可能会增大它)。

*例子*

这个过程最好用例子来说明。假设我们的机械连接的共振频率为50HZ。为了消除它，我们决定加入一个小阻尼的带阻滤波器(阻尼系数为0.2，自然频率为50HZ，和一个大阻尼的带通滤波器(阻尼系数为0.8，自然频率为80HZ)来限制滤波器的高频增益。伺服更新时间为442毫秒。

带阻滤波器

首先我们计算带阻滤波器的分子，或抗共振部分。

S平面根

计算抗共振频率为50HZ的S平面根:

$s_{real}=-b*\omega_n=-0.2*[2*Pi*50]$

$=-62.8\sec^{-1}$

Z平面根

$\pm\omega_n*sqrt(1-b^2)=\pm2*Pi*50*sqrt(1-0.04)$

=±307.8$\sec^{-1}$

接着我们计算匹配的Z平面根:

$z_{real}=\exp(-62.8*0.000442)*\cos(307.8*0.000442)$

=0.9726*0.9907=0.964

$z_{imag}=\exp(-62.8*0.000442)*\sin(307.8*0.000442)$

=0.9726*0.1356=0.132

系数

现在我们计算Z的系数:

$N(z)=1-1.928z^{-1}+0.946z^{-2}$

滤波器的直流增益为1-1.928+0.946=0.018

带通滤波器

对80HZ带通滤波器重复以上计算将得出分母:

S平面根

$s_{real}$=-b*$\omega_n$ =-0.8*[2*Pi*80]

=-402.1sec$^{-1}$

$s_{imag}=\pm\omega_n * sqrt(1-b^2)=\pm$2*Pi*80*sqrt(1-0.64)

=±301.4sec$^{-1}$

Z平面根

$z_{real}=\exp(-402.1*0.000442)*\cos(301.4*0.000442)$

=0.8372*0.9911=0.830

$z_{imag}=\exp(-402.1*0.000442)*\sin(301.4*0.000442)$

=0.8372*0.1329=0.111

系数

$D(z)=1-1.660z^{-1}+0.701z^{-2}$

DC gain=1/(1-1.660+0.701)=1/0.041=24.48

陷波滤波器的净增益为0.018*24.48=0.441。我们必须用大小为1/0.441=2.27的因子加以补偿，以不影响整个滤波器的直流增益。

I变量

为构成抗共振(带阻)滤波器，我们将使用PID滤波器的“N”部分;为构成高频带通滤波器，我们将使用PID的“D”部分。N1(Ix36)和D1(Ix38)为各自滤波器的$z^{-1}$系数;N(2)和D2(Ix39)为$z^{-2}$的系数。

```
Ix36=-1.928     ;N1
Ix37=0.946      ;N2
Ix38=-1.660     ;D1
Ix39=0.701      ;D2
```

直流增益更新

最后，我们把旧的比例增益与陷波滤波器直流增益的倒数相乘。如果Ix30在设置陷波滤波器之前为100，000，为保持硬度不变，我们设置:

```
Ix30=227000     ;100，000*2.27
```

我们可以把Ix30设得更高一些来改变环路的硬度而不影响陷波滤波器的特性。事实上，引入陷波滤波器的一个重要原因就是能在不影响稳定性的情况下增加环路的硬度。

**陷波滤波器的其它用途**

陷波滤波器是一个真正广义的二次数字滤波器。除了产生滤波功能外，它还能用于其它用途。这使用户在定制伺服算法的性能上有很大的灵活性。

超前滞后校正

如果多项式的根的实部大于虚部，陷波滤波器可仅作为一个超前滞后滤波器使用。超前滞后滤波器的性能与一个PID滤波器很类似，它用在当滤波器的设置更多地由分析来决定，而不是经验时。当需要一个基本的超前滞后伺服滤波器时，所有的伺服增益Ix31-Ix35需被设为0;Ix30仍作为一般的增益项使用。

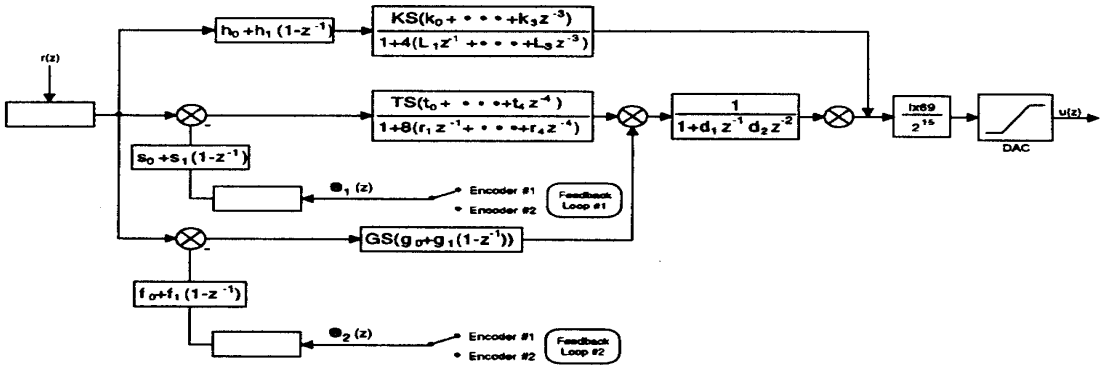
低通滤波器

当提高运行精度比提高系统带宽更重要时，它也可被当作低通滤波器使用。这可以通过把D1项(Ix38)设为0.0-1.0之间的一个正值来实现。这个值越大，滤波器的剪切频率越低。滤波器的直流增益为1/(1+D1)，而比例增益需乘以因子(1+D1)来保证整个环路的增益不变。

## 扩展(设置极点)伺服滤波器

如果系统具有很差的动态性能，例如复合的共振和低频共振，与PMAC Option 6一同购买的扩展伺服算法可替代PID滤波器。当购买了Option 6时，PMAC上的所有电机必须使用扩展算法来替代PID;它们不可以混合使用。当使用Option 6时，I变量Ix30-Ix69的意义与标准的PMAC不同;你需要参考扩展伺服算法的手册。

**PMAC:扩展控制算法方框图(图7)**




因为控制算法是以“零极点”的形式而不是增益的形式表达，它不象PID滤波器那样可通过直觉来调节。因此，我们几乎总要使用附件25“伺服评估软件包”来正确地调节这种滤波器。

### 用户决定的伺服滤波器

对一个有经验的用户，针对较少见或(和)动态性能很差的系统，PMAC提供了与定制的用户写的伺服算法的链接。这些例程通常在PC或其兼容机上由Motorola 56000汇编语言代码编写，由56000进行交叉汇编。Delta Tau提供了关于在哪儿找到所需的信息，在哪儿用输出命令以及在哪儿存储算法程序的信息。(注释:没有经验的用户不要使用这项功能。若要这样做，用户必须熟知伺服理论以及汇编语言的知识。)

### 如何定制算法

用户必须在主机上使用交叉汇编工具编写算法。Motorola为PC及其兼容机(SSP56000CLASa)，Macintosh II(SSP56000CLASb)，Sun-3 工作站(SSP56000CLASc)，以及DEC VAX(SSP56000CLASd)提供了56000交叉汇编程序。几乎所有的用户都在IBM-PC上工作，文件将转换成DOS格式。这些例程通常由一个简单的屏幕编辑器编写，然后通过交叉汇编程序转换成56000机器代码。这些机器代码文件必须以。LOD作为后缀。

### 下载及运行

第一步:将用户编写的算法编译成以。LOD为后缀的DOS文件。

第二步:执行由Delta Tau提供的IBM-PC环境下的转换程序”CODE。EXE”，将机器代码文件转换成PMAC可接受的格式。在DOS提示符下键入:

>CODE {filename} <ENTER>

在这里，{filename}指机器代码文件(不加。LOD后缀)。CODE将产生一个名为{filename}。PMC的文件，它可被送给PMAC。

第三步:执行PMAC执行程序并选择编辑器菜单(用鼠标，<ALT-E>或<F10>E)。选择“Download File to PMAC”选项，然后键入文件名(。PMC为缺省的后缀)。一旦你按下<ENTER>，执行程序将自动下载文件到正确的PMAC内存位置，它将被后备电池保存在RAM中的某一位置。

第四步:通过设置Ix59为1或3，可使电机使用用户编写的滤波器。如果Ix59被设为0或2(缺省)，它将使

用标准的伺服算法。

**内存位置，软件界面和程序限制**

用户编写的伺服程序在内存中位置为:

--程序代码开始地址:P:$B800(在V1.14或更早版本为P:$9C00)

--允许程序最大长度为1K的24位字(P:$B800到P:$BBFF; 在V1.14或更早版本为P:$9C00到P:$9FFF)

可用数据空间

用户编写伺服程序中变量可用的数据空间为:

--初始化为0的用户寄存器L:$0770到L:$077F

--

非初始化的用户寄存器L:$07F0到L:$07FF。对于使用后备电池的PMAC，在掉电/重启之前这些寄存器保持送给它们的最后一个值;对于使用闪存的PMAC，在开机时，它们为闪存内的最后一个值。

--为DEFINE UBUFFER保留的寄存器：从L:$9FFF以下，长度为被语句的缓冲区长度。

与其它固件的联系

PMAC的固件用以下格式与用户编写的滤波器通讯:

--在入口，“B”累加器包含一个48位的整形量，表示要求的位置(DPOS)，单位为1/(Ix08*32)步。

--在入口，“X”寄存器包含一个48位的整形量，表示实际的位置(APOS)，单位为1/(Ix08*32)步。

--

在入口，“A”累加器包含一个48位的整形量，表示要求的速度(DVEL)，单位为1/(Ix08*32)步/伺服周期。

--

在入口，“Y1”寄存器包含入口的Ix08值。“R1”寄存器包含伺服状态寄存器的地址，它决定哪个电机将被闭环($003D为电机1，$0079为电机2，etc。)。在使用多个伺服电机时，该信息对区分特定电机的寄存器是有用的。

--

在出口，“A1”的高18位包含电机的控制信息。如果电机不由PMAC换向(Ix01=0)，该值将直接被载入DAC寄存器。如果电机由PMAC换向，该值将被相位例程使用，来产生两个DAC输出。

--用户编写的滤波器算法必须以跳转到P:$0023的JMP指令结束。

限制

在用户编写的代码中必须遵守以下限制:

--只可使用一级堆栈。

--

不要往56000的地址寄存器R2，R3，R6，R7;变址寄存器M2，M3，M6，M7:或偏置寄存器N2，N3，N6，N7内写值。其它R，M，N寄存器可被写入，但在跳出用户编写的代码之前必须恢复。

--除了A，B，X和Y寄存器外，所有其它的56000的寄存器在跳出用户编写的代码之前必须恢复。

**用户编写伺服程序的其它用途**

这些计算不一定必须用来闭合电机的伺服环。它可被仅仅看作一个可在每个伺服周期内以非常快的速度执行的例程。由于这一点，它可被用作一个超快的PLC程序(速度上比PLC0快的多)。

在不被当作伺服程序使用时，它们最普遍的用途是用来设置以及清除基于位置的许多输出(当不仅仅需要硬件的比较位置输出)。与PLC程序比较，它有许多优点。首先，它由汇编程序编写，所以没有编译或解释的延迟。其次，它使用的定点算法要好于PLC(和运动)程序使用的浮点算法。第三，它保证以一个固定的速度运行;而即使PLC0也会被运动程序所延迟。

为实现这样一个功能，你需要激活另外一个电机(Ix00=1)并告诉它使用”用户编写的伺服算法”(Ix59=1)。为执行算法，该电机必须处在允许闭环的状态下;如果Ix80=1，它将被自动设置，或通过J/命令来使其进入这种状态。你不需要担心从哪儿得到和停止伺服信息，但你仍需小心不干扰任何其它东西。

简单的用户编写的伺服程序的例子:

```
        ADD B,A Y:SAMP_A,Y0      ;Counter=counter+1, Y0=samp. factor.
        SUB Y0,A A1,X:SAMP_A     ;Compare counter with factor & save count.
        JMI ESERVO               ;Skip if counter < samp.factor.
        MOVE A1,X:SAMP_A         ;Clear and save the counter.
        MOVE Y:DEMAND_A,B        ;Load demand position (assuming 24-bit)
        MOVE X1,A           ;Load Actual position (assuming 24-bit)
        SUB A,B             ;Form the 24-bit error(k)

; Now execute the low pass filter and
; exit with value of DAC in the upper 18 bits of A1 (integer).
;
        RND B X:COEF_A,X1            ;Round B adding 1 to bit 23 (of B0).
        MOVE X:OLD_UF_A,X0 B,Y1      ;X0=uf(k-1), Y1=error(k), X1=a1.
        MPY X1,Y1,B Y:COEF_A,Y0      ;B=a1*error(k), Y0=a2.
        MACR X0,Y0,B                 ;B=a1*error(k)+a2*uf(k-1).
        MOVE  B,X:OLD_UF_A           ;Save uf(k)
ESERVO  MOVE X:OLD_UF_A,A            ;LOAD SAVED DAC VALUE
        MOVE A0,A1
        REP #6
        ASL A

        JMP <$23                     ;RETURN TO SERVO
        END
```

```
/* .LOD TO .PMC CONVERSION PROGRAM 09/04/90 V1.0 */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

   FILE  *infile,*outfile;
   char  buf[81],bufr[32768];
   int      l,m,n,x,y;

void main (argc,argv)
   int      argc;
   char  *argv[];
{
   m = 0;
   x = 0;
   if (argc > 1) {
      infile = fopen(strcat(strcpy(buf,argv[1]),".lod"),"r");
      if (infile != NULL) {
         outfile = fopen(strcat(strcpy(buf,argv[1]),".pmc"),"wb");
         setvbuf(outfile,bufr,_IOFBF,32767);
         while(fgets(buf,80,infile)!=NULL&&strncmp(buf,"_DATA P",7)!=0);
         sscanf(buf+8,"%4x",&y);
         while(fgets(buf,80,infile)!=NULL&&strncmp(buf,"_END",4)!=0) {
            n = 0;
            l = strlen(buf);
            while (l > n+5) {
               switch (m) {
                  case  0: fprintf(outfile,"WP$%04X",y); y += 5;
```

```
                    default:
fprintf(outfile,",$%c%c%c%c%c%c",buf[n],buf[n+1],buf[n+2],buf[n+3],buf[n
+4],buf[n+5]);
                }
                n += 7;
                m += 1;
                if (m==5) {
                   m = 0;
                   fprintf(outfile,"\r\n");
                }
             }
          }
       if (m>0) fprintf(outfile,"\r\n");
       }
    }
```

## §3—7 确保你的应用安全：

重要注释：***Delta Tau***数据系统在PMAC控制器上提供了许多安全特征，而且投入了许多资源和手段来确保PMAC是一个安全的产品。然而使用PMAC的控制系统的安全的最终责任要依靠运用PMAC的安全特征和系统其它部分的安全措施的系统设计人员。

### 硬件超行程限制开关：

PMAC有和每一个编码器输入相联系的正向和反向硬件超行程开关输入。这些输入都用故障保护电路设计进行了光电隔离。这些输入必须被主动的保持为低电平，从而给PMAC的卡的源电流（例如，正常闭合的开关）将认为它自己还未进入限制。电流源既可以是供给模拟输出级能量的外部＋15V输入，如果被跳线到供给模拟输出级能量则是＋12V的总线动力线路（消除了模拟光电隔离），又可以是从JMAC H2
OPTO＋V脚引入的＋24V输入。E89和E90的跳线决定了选用哪一种资源。为了便于移动，它们必须正确地被设置。

每一台电机都必须监测这些输入信号，这是通过将I－变量Ix25（给x＃电机）设置给保存适当输入的寄存器的地址来完成的。当撞到一个行程开关时，PMAC将使该电机以用户编程定义的比率减速（查看I－变量Ix15）。如果此时电机是在一个正在执行一个运动程序的坐标系中，那么该坐标系中的所有电机都将以它们自己的Ix15中的比率减速到停止。其效果相当于发出了一个A（abort）命令。

注意：一旦PMAC检测到一个限位条件，它将接受响应的电机引入一个预定的轨迹。如果此时有显著的跟随误差，实际位置将在一个较长的时间中试图赶上要求的位置。当跟随误差足够大时，有可能要求的位置将超过行程限位并导致硬件停止。所以设置一个合理的致命跟随误差限制，并且允许有足够的空间在越过限位开关，吸收误差到跟随误差限制，这是十分重要的。

限位输入脚是方向传感的：－LIM脚停止正向移动（从负的一边撞到限位开关），而＋LIM脚停止负向移动（从正的一边撞到限位开关）。这使得发出一个命令从你撞入的行程开关离开成为可能。然而这实质上也意味着你必须将限位开关和输入正确地联系起来，否则它们将全然无用。

注意：限位开关的极性常是和大多数人凭直觉所想到的相反。行程正向限位端的开关是－LIMn输入，而行程负向限位端的开关则是＋LIMn输入。确认你已仔细地测试了你的限位开关的极性。

通过设置Ix25的第17位为1，可以使给电机的硬件限位功能无效。这对于回零进入一个限位开关（默认条件下，如果限位被使能，则其将不会工作），或对于硬件超行程限位开关不实用的系统（例如，旋转台面），将是十分有用的。如果在Ix25中的标志的地址是＄C004，那么使限位无效的Ix25的值将是＄2C004。

## 软件超行程限位：

PMAC对于每一台电机还有正向和反向的软件限位作为硬件限位的补充和替代。给这些限位的用户设置的值（对x＃电机，是Ix13和Ix14参数）不能被存储到电池保持的EAROM里。当撞到这些限位时PMAC的行为动作将和硬件限位的完全一样。在回零移动中，软件限位将自动无效，直到发现回零触发器为止。一旦发现了回零触发器，软件限位将被重新激活，并使用新的起始位置作为参

考。如果用到了超行程限位，那么限位开关应该距离起始位置足够远，以允许在发现触发器之后能减速和换向。

## 跟随误差限制：

PMAC给每一台电机都有三个跟随误差限制（跟随误差就是在任何时候命令要求的位置和实际位置之间的差值）。跟随误差限制对于严重的系统错误是一个重要的保护手段。如果没有这样的措施，一些严重的系统错误如反馈的丢失，将导致象全速失控那样的危险情况。

**致命跟随误差限制：**

这些限制中的一个（Ix11）是一个“致命”的限制，如果超出了该限制将导致整个系统的关断，例如，输出被命令为零，放大器失效，给定电机的运动和给定电机坐标系的程序都被取消。哪一个放大器将会失效，只是超越限制的电机，或是整个坐标系中的放大器，还是整个PMAC板上的放大器，是通过Ix25来决定的。该限制适用于某环节犯了严重错误的情况（例如，反馈或功率级的丢失），并且所有的操作都将会停止。

在由于致命跟随误差限制而导致一台电机或多台电机被停机之后，闭环控制将用J/命令（对单个电机），A命令（对坐标系），或〈CTRL－A〉命令（对整个卡），重新被建立起来。

**警告跟随误差：**

第二个限制（Ix12）是一个警告限制。当它被超越时，PMAC能通过可编程中断控制器（给PMAC－PC和PMAC－STD）设置电机和电机的坐标系的状态位，而且还能设置在控制板连接器上，在机器连接器上的输出线。这允许采取一定的特殊动作，既可以让PMAC自身通过一个PLC程序来做，也可以通过主机来完成（主机可以通过一个中断来发现，可以通过查询卡的情况来发现），还可以通过一个执行机构来完成，该执行机构可以通过一个外部信号来通知。

这些限制都可以通过设置参数为零而使无效，但是在一些具有潜在的杀伤人员的可能性的应用当中，即使是可能导致人员危险的应用当中，这都是应该绝对禁止的。使致命限制无效去除了对严重错误的保护措施，而这些严重的错误是可能导致环境失控，从而将系统带入任何人都来不及反应的全功率状态。

**积分跟随误差保护：**

作为对由Ix11
变量提供给每一台电机的正常跟随误差的保护的补充，PMAC在跟随误差的时间积分值超过预置值时将会关短电机。这个积分的误差功能将在那些被测的跟随误差的量级总不会是十分大的情况下起到保护作用，例如后面紧接着一个非常短的移动命令的反馈丢失的情况。

如果Ix63积分限制参数小于零,PMAC只执行积分跟随误差检查。当是这种情况时，Ix11的值被用来给标准的非积分跟随误差检查，但是附加的，PID积分器的值将和Ix63积分限制值进行比较。如果积分器的值在＋Ix63或-
Ix63饱和了（PID环的限制功能将不会使它超过该限制值），那么PMAC将由于一个积分跟随误差限制错误而使电机无效，就象它对待一个标准跟随误差错误那样。

为了使积分跟随误差限制有效，Ix33的积分增益必须比零大，并且应设置地尽可能地高。同时，Ix34积分模式参数必须被设置为0，以便在编程执行中积分器是打开的。

需要知道的是，如果命令输出在Ix69饱和，积分器将停止数值上的增长。Ix63的值应该足够的小，以便在输出饱和之前能安全断开。将导致积分器的输出在Ix69饱和的Ix63的值是：

$$|Ix63| = (Ix69 \times 2^{23}) \div （Ix08 \times Ix30）$$

为了让关断的功能有效，Ix63的值必须比该值小。记住输出的组成部分不只有来自积分器的输出，例如，还有来自比例增益的输出。用一台无配件电机来测试该限制能否可靠地使你的电机安全断开。
当由于积分跟随误差的缘故使得电机无效时，标准跟随误差错误电机状态位被设置。另外，一个单独的积分跟随误差错误电机状态位被设置。当电机重新有效时，这两个位都将被清除。

*注意：在用到选件6被扩展的伺服算法固件的PMAC板上，是没有积分跟随误差保护功能的。因为该算法在伺服中没有积分寄存器来和积分限制作比较。*

## 速度限制：

PMAC对于每一台电机都有一个可编程的速度限制（Ix16），它对于线性混合的编程运动都是有效的（圆，PVT，急速运动和主轴运动观察不到这个限制）。如果电机要求的速度超过了该电机的速度限制，那么移动将会变慢以便让速度不会超过该限制。在一个多轴可编程移动中，坐标系中的所有轴都将按比例地变慢，以便轨迹不会发生变化。速度和限制的比较是在假定没有进给速度修调的情况下进行的（100％的速度）；如果用到了速度进给修调（a.k.a.时基控制），速度限制将用该修调按比例决定。当PMAC自动分段移动时（Ix13>0），将观察不到Ix16速度限制。

## 加速度限制：

PMAC对每一台电机都有两个可编程的加速度限制，一个是给手动和回零移动的（Ix19），另一个是给

线性混合编程运动的（Ix17）。圆形运动、急速运动、PVT和主轴运动将观察不到这个限制。如果一台电机要求的加速度超过了给该电机的设置，加速将会变缓，以便加速度限制不会被超过。在一个多轴混合可编程运动中，坐标系中的所有的轴都按比例地使加速变缓，以便轨迹不会发生变化。加速度和加速度限制的比较是在假定没有进给速度修调的情况下进行的（100％的速度）；如果用到了速度进给修调（a.k.a.时基控制），加速度限制将用该修调按比例决定。当PMAC为了圆周插补运算，而自动分段移动时（Ix13>0），将观察不到Ix17加速度限制。

Ix17对于在系统开发前期阻止不合理的运动非常有效，而在系统开发前期是很容易犯这样的大错误的。在某些系统中，它也能够被用在一些实际应用中以确认加速度总是在最小的时间内产生。在这些应用中，TA和TS加速时间设置得非常小，以便Ix17能够一直被用到。

有时Ix17限制在你不需要的时候显得太“有效”了；而有时它又不够有效，能够允许一些轨迹运动超过该限制值。

要得到更多的细节，参考《编写一个运动程序》中的加速度限制部分。

## 命令输出限制：

PMAC对每一根轴（Ix69）都有一个可编程的输出限制（在PMAC送给放大器的命令上），它是作为给电流环放大器的扭矩限制，或给转速计放大器的实际速度进行限制来起作用的。如果该限制被用来改变伺服环的命令，当从该限制条件退出时，PMAC的anti-windup保护功能将激活以阻止振荡。另外，还有一个允许反馈滤波器查看的误差范围的限制（“Big Step”限制：Ix67），它可以用一种被控制的方式来使一个突然的运动减慢。

## 积分电流（$I^2T$）保护：

如果时间积分电流级超过了一定的限制，PMAC将告知一台电机出错。这可以保护放大器和（或）电机不会因为过热而产生故障。这类保护通常被称为$I^2T$（“eye-squared-tee”），因为它测量超时电流平方的积分（能量的散失是和电流的平方成比例的）。

一些放大器在它们内部有自己的$I^2T$保护，但是别的更多的放大器却没有。PMAC的$I^2T$保护可以用在两种情况的任何一种。它能读PMAC的命令电流寄存器以决定电流级别，所以它能够不用给PMAC引入实际的电流测量信号而被应用。它可以和PMAC计算电流命令的任何放大器联系起来使用，不论PMAC是否正在执行换向或（和）数字电流环功能。它不适用于PMAC输出一个速度命令的系统中，不论是模拟速度，还是脉冲频率。

注释：$I^2T$保护功能在用到选件6扩展伺服算法固件的PMAC板上是不具备的。Ix57和Ix58在该固件版本上有其它的功能。

对于每台电机，两个I一变量控制着给该电机的$I^2T$保护功能。Ix57是连续电流限制值。它有和Ix69瞬时输出限制一样的单位，一个16位DAC的字节（即使用到了其他的输出设备）。两者的范围都是0到32,767，而32,767是PMAC最大可能的输出值。通常，Ix57被设置为Ix69的值的1/4到1/2。

Ix58是电流积分限制参数。如果Ix58被设置为0，该功能无效。如果Ix58比0大，PMAC会把电流积分后的值和Ix58的值相比较。当电流积分后的值超过了Ix58的值时，PMAC将认为电机出错（就象一个放大器出错一样）。出错的电机将无效；如果该电机是在一个正在执行一个运动程序的坐标系中，那么该运动程序将被取消；依照Ix25的第21和22位的设置，其它的电机也将无效。

PMAC的$I^2T$保护功能依照下列等式来工作：(见下页)

$$\text{Sum=sum+}\left[\left(\frac{I_q}{32768}\right)^2+\left(\frac{I_d}{32768}\right)^2-\left(\frac{Ix57}{32768}\right)^2\right]\Delta t$$

其中:

$I_q$ （正交电流）一个16位DAC的单位。是PID滤波器要求扭矩结果的输出；

$I_d$ （直接电流）通过设置Ix77而要求的磁化强度电流；除当PMAC正在执行感应电机的矢量控制时,该值通常是零。

$\Delta t$ 伺服周期中自上次采样以来的时间。

如果其和超过了Ix58，一个$I^2T$错误将会产生。当要求的电流低于Ix57时，该和将减少，但是它绝对不会比零低。

示例：

用到命令输出限制Ix69＝32767（最大值），积分电流限制Ix57＝16384（最大值的一半），磁化强度电流Ix77=0，电机撞到一个限制，并且命令输出在32767饱和。在这段时间中，$I^2T$功能将计算：

$$\text{sum=sum+}\left[I^2+0^2-0.5^2\right]\Delta t=\text{sum+0.75}\Delta t$$

sum将以每伺服周期0.75的比率增加。在默认的2.25kHz的伺服周期更新率时，sum将以每秒2250×0.75＝1688的比率增加。

如果你希望电机在该条件的3秒之后安全脱开,你应该设置Ix58为1688×3 ＝5064。

当电机上发生一个$I^2T$错误时，PMAC将象对待一个放大器错误一样采取相应的动作。发生错误的电机将无效，如果设置了Ix25可能将使其它的一些电机也无效。PMAC设置放大器错误电机状态位。对于一个$I^2T$错误，PMAC将也会设置一个单独的$I^2T$错误电机状态位。这两个位将在电机重新有效时被清除。

注意：当PMAC给电机执行换向时，没有用到$I^2T$保护，你需要确认磁化强度电流参数Ix77仍然被设置为零。在这种设置中，Ix77将不会影响操作，但它将影响$I^2T$的计算。

## 放大器使能和报错线：

给每一台电机的放大器使能（AENA）输出和放大器报错（FAULTn）输入线的用途对于安全操作是十分重要的。不使用使能线，欲使一个放大器无效，将依靠在PMAC模拟输出和放大器模拟输入里的精确的零偏置。如果不用报错线，则当一个放大器关断时，PMAC将不会知道，并且也不会采取相应的动作。

注意：当使用放大器使能信号默认的驱动器，且用low-true使能极性（低电压－传导有效；高电压－不传导无效），则对能量供应提供了更好的故障安全保护。假如给PMAC计算部分的+5V能源，或者是+15V的模拟电源丢失了，那么放大器将会自动无效，因为输出三极管将进入它的非导通状态。如果你要求这种破损安全保护，但又不能将这种极性的信号直接连接到放大器上，那么你必须使用中间电路来转换信号的格式。使用一台交变电源的驱动器时，high-true极性将提供更好的故障安全保护。

参考3－27页上的*专用数字输出标志（JMACH，JEQU Ports）*和3－26页上的*专用数字输入标志，*以获得详细的细节。

## 看门狗计时器：

PMAC有一个在板的“dead-man”（或“watchdog”）监视时钟。这个子系统给软件故障提供了故障安全关断功能。阻止PMAC由于监视时钟而断开硬件电路,必须要有两个条件。第一，它必须检查到一比大约4.75V高的直流电压。如果电源电压比这个值低，电路的继电器将断开而且整

个卡将关断。这就避免了由于电压不足而导致的寄存器错误。第二个必须的条件是时钟必须检查到一个频率比大约25Hz高的方波输入（由PMAC软件提供）。在前台，实时中断程序设置了该信号，然后后台程序重新设置该信号。如果该卡由于硬件或软件的什么原因而不能以该频率或更高的频率来重复设置并清除该位，那么电路的继电器将断开而且整个卡将关断。

当时钟关断，在JPAN连接器线F2LD/，和（如果E28被连接2-3）JMACH连接器上的线FEFCO/将变低，DAC的输出将强制为零，而且线AENA将处于无效状态。另外，在PMAC－PC的可编程中断控制器(PIC)上的三级中断将被触发，而在PMAC－STD的PIC上的一个五级中断将被触发。板上CPU部分的红色发光二极管将被打开。

如果PMAC的监视时钟使电路断开，关掉你的电源电路是十分重要的。在JMACH1连接器上的FEFCO/输出对于该用途是十分有用的。它是一个集电极开路输出门输出，是参考当监视时钟关断电路时变低（100mA汇集能力）的AGND。它也可以由于缺乏＋5V电源而关断，如果该电压降压足够慢，以至于自欠压状态的监视时钟关断能在其之前完成。在你的工业系统中，该使用必须经过评估，以确定它是否可靠。

一旦监视时钟关断了电路，给PMAC的电源必须周期性的开关，或者在JPAN上的INIT/线必须先变低，然后变高，以便恢复正常的功能。

## 硬件停止命令输出：

PMAC有一种硬件输入，可以用用户设置的减速来停止一个运动或一个程序。“Abort”输入将使被选坐标系中的所有轴都停止运动，就象是由电机/系统选择输入所决定的那样，该命令立即执行，并且每一台电机用到的减速都是在Ix15中所设置的。“Hold”输入将执行同样的功能，除了各轴的减速是相互协调的，以使多轴的轨迹在减速中能保持不变。

这些专用的输入是在PMAC的控制板连接器上（JPAN；J2）。它们工作的坐标系是由四个low-ture输入线FPD0/（LSBit），FPD1/，FPD2/,和FPD3/(MSBit)产生的二进制数决定的。值0（所有的都高）将使该功能无效；值1到8将选择被编号的坐标系。

## 主机产生的停止命令：

这些功能和别的一些功能能够通过主机，用单字符命令或双字符命令来执行。例如，〈CTRL－A〉命令执行的功能和对所有被选择的坐标系执行“Abort”输入是一样的，而A命令只对软件选址的坐标系执行“Abort”功能。〈CTRL－O〉命令对所有的坐标系执行“Hold”功能，而H命令只是对软件选址的坐标系执行。还有，〈CTRL－Q〉命令使在预定移动结束的所有程序停止，而Q命令只停止被软件选址的坐标系的程序。〈CTRL－K〉命令使所有电机立即无效，而K命令只是使软件编址的电机无效（如果该电机是在一个正在执行一个运动程序的坐标系中，那么在发出K命令之前必须发出一个Abort命令）。

这里的任何一条命令都可以用COMMAND“{command}”或COMMAND^{letter}的语法，从程序里发出。当一个坐标系中运行的一个运动程序发出一个使当前坐标系中电机的电机无效的命令（K命令）时，该命令将被自动拒绝。

## 程序校验和：

### 固件校验和：

作为一个后台任务，PMAC将连续地计算它的内部程序（固件）的校验和。每一次它计算了校验和后，它都将该值同存储器中参考寄存器里（X：＄07B1）的值进行比较，该寄存器中是人工地将正确的值输入的。刚出厂的PMAC将预加载工厂中给该固件版本的正确的参考值。

如果PMAC检测到它所计算的校验和和参考的校验和之间存在错误的匹配，它将设置全程状态位（X：＄0003的第12和13位，可以用？？？命令来操作），并且停止执行任何校验操作。这使得被计算的

值冻结在正在执行的校验和寄存器X：$0794中。PMAC在固件错误事件中将有可能采取任何别的动作；究竟让PMAC采取何种的动作，这将由主机或者一个PMAC PLC程序来决定。

当PMAC通过在标准CPU部分的PROM的更新或者将新的固件加到可选CPU部分的闪存EEPROM IC中，从而使PMAC升级到新的固件时，那么原来的参考校验和的值对于新的固件来说将是错误的。得到正确的参考校验和的值的最容易的方法是用$$$***命令来重新初始化整个卡，这将自动地将新的校验和的值读到参考寄存器中去。然后，必须在关断电源或重置PMAC之前用SAVE命令保存该变化。

如果用户想要修改这个参考值而不用重新初始化整个卡，那么可以利用这样的一个情况，那就是当PMAC发现计算的校验和和参考值之间如果存在错误的匹配时，则该计算的值将被冻结在正在执行校验的寄存器X：$0794中。因此，给新版本固件的正确的校验值可以用命令RHX：$0794来读入，而且返回值将被写入参考寄存器X:$07B1中。

**用户可编程序校验和：**

作为一个后台任务，PMAC将连续地计算固定的用户编程缓冲区的校验和。每一次它计算了该校验和后，它都将这一次计算的值和上一次这些缓冲区之一被关闭时所计算的校验和的值相比较。

如果PMAC在这两个校验和之间检测到错误的匹配，它将设置全程状态位（X：$0003的第13位，可用？？？命令来操作），并且将停止执行任何程序校验操作（通信校验是独立于这些之外的）。它不会自动关断操作。将由主机或者一个PMAC PLC程序来决定当有一个校验错误时应该采取何种动作。

## 通信数字积分功能：

PMAC提供了许多技术来确保有效的数据通信，包括串行奇偶校验，帧间错误校验，串行全双工通信以及在串行和总线通信中都有的双向校验计算。参考《编写一个主机通信程序》部分，可以知道这些技术如何工作的详细情况。

# §3—8 基本的电机运动：

一旦你将电机定义好并使它简单工作起来，你将发一些命令让电机做一些基本的运动。手动命令将允许你让一台电机作独立于其它电机的简单运动，而且不需要编写运动程序。你可以在开发中，在诊断中，以及在调试中用到这些运动，但你也可以在你的实际应用当中用到这些运动。

另外一类简单电机运动是回零运动。它基本上是一个“触发微动”类型的命令。这时PMAC命令电机运动直到它发现预先定义好的触发器。然后它使电机停下来并返回触发时位置（可能会有一个偏置），并且设置电机位置为零。

回零运动是当你不知道起始位置在哪里时才用到的。如果你有一个增量传感器，并且在启动时你不知道位置；那么在这种系统中回零运动将是执行的第一个运动。但是，如果你知道起始位置，且只是希望返回该位置，那就不需要做回零运动；只需要命令其运动到零位置（例如，J=0或X0）。

手动和回零运动的轨迹实质上和线性混合编程运动是一样的。所不同的是手动和回零运动的运动参数必须通过I—变量来指定，并且运动本身是通过在线命令来启动的，而不是通过运动程序。还有，这些运动是直接指定给用编号定义的电机，而不是给用字母指定的轴的。这些运动是不用比例单位来描述的（都是基于编码器计数和毫秒的）。

## 手动运动控制：

### 手动加速：

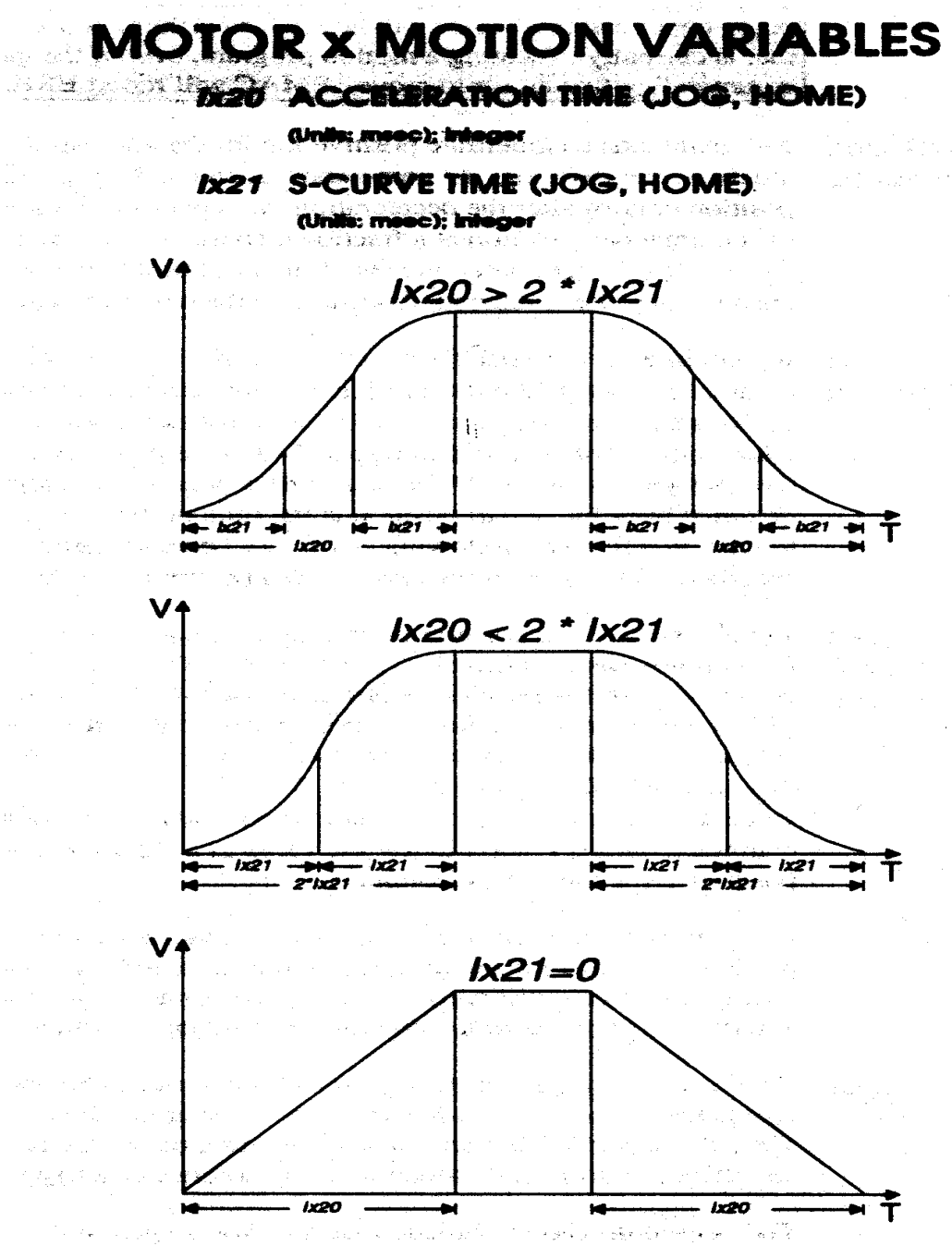
手动和回零加速时间是由给电机x的Ix20 来指定的，而S-curve时间是由Ix21来指定的。如果Ix20的值比Ix21的值小两倍，那么所用到的加速时间将是Ix21的两倍。手动和回零运动的加速限制是由Ix19来设置的（用计数/毫秒$^2$ 单位）。如果Ix20和Ix21是如此之小，以至于Ix19将会被超过，那么Ix19将会来控制加速时间（不会改变轮廓形状）。如果你希望用比例来指定加速度而不是时间，那么只需要将你的加速时间参数设置得足够小，以便限制加速比例参数。

即使你希望用比例设置你的加速度，但也不要将两个加速时间参数Ix20和Ix21都设置为零。这将导致运动计算中的“被零除”的错误，并产生不稳定的运动。最小的加速时间设置应该是Ix20=1以及Ix21=0。

### 手动速度：

手动速度是由Ix22来指定的，Ix22是一个速度量级，其单位是计数/毫秒。方向是由手动命令本身来指定的。

## （PAGE 3－123 电机 x运动变量图）




**手动命令：**

手动控制一台电机的命令是电机所特有的在线（立即被执行）命令；它们只在当前选址的电机上起作用。

如果当电机是在一个正在执行一个运动程序的坐标系中，即使运动程序现在没有命令电机运动，发给电机的手动命令也将被拒绝。如果I6被设置为1或3，PMAC将报错为ERR001。

不明确的手动控制命令：

J＋给被选址的电机发出一个模糊的正向手动命令；J－则给选址的电机发出一个模糊的反向手动命令；J/命令手动停止，在减速之后使电机处在位置控制之下。对于J/命令来说，将命令位置保留在小数

计数上是完全可能的，而这将引起相邻整数计数值之间的颤动调谐。如果这成为一个问题，J！命令能够用来强制命令位置到最近的整数计数值。

手动控制到一个指定的位置：

手动命令可以用来控制电机到一个指定的位置，或经过一段指定的距离。J＝发出一个到上一次微动位置的微动命令；J＝{常量}发出一个到命令中指定位置（非比例的）的微动命令；J＝＝{常量}发出一个到命令中指定位置（非比例的）的微动命令，并使该位置成为“上一次微动”位置；J^{常量}发出到距发出命令时刻实际位置指定距离的微动命令（J^0对于接收残留的跟随误差是很有用的）；J：{常量}发出到距发出命令时刻命令位置指定距离的微动命令。

手动控制变量指定的运动：

可以实现手动控制运动到一个指定的位置或经过一个指定的距离是可能的。每一台电机都有一个指定的寄存器（L:$082B给电机1，L:$08EB给电机2，等等），来保存给下一次手动控制运动的位置和距离。该寄存器可以用来容纳一个按比例编码计数位的浮点数值。可以用L格式的M－变量来使用它。命令J＝* 将使PMAC把该值用作一个目的地址。而J^*命令将使PMAC把该值用作在发出命令时距当时实际位置的一个距离。J：*
命令将使PMAC把该值用作在发出命令时距当时命令位置的一个距离。

每一次用到这些命令时，此时的加速度和速度参数控制着对命令的响应。如果你希望改变手动控制电机的加速度和速度参数，改变适当的参数，然后再发出另一个手动命令。

触发微动：

触发微动功能允许一个手动控制的运动被触发器中断并被此时和触发时位置相关的运动中止。这和回零运动非常相似，除了电机零位置没有被改变之外，还有在存在一个缺乏寄存器时目的地址。

给一台电机的“触发微动”功能是通过在给电机的通常的“定义”手动控制命令之后加上一个^{常量}区分符来实现的。其中，{常量}是在停止前和触发时位置相关的距离，按编码计数位单位。它不能和模糊的手动控制命令J＋和J－一起来使用。

这样，给一个触发微动的手动控制命令象以下形式：J＝10000^100,J=*^-50或J:50000^0。在^之前的值是目的位置或在缺乏触发器时将要移动的距离（依靠手动控制命令的类型）。如果该第一个值用符号*
来代替，PMAC将在预定义寄存器中查找距离或位置。第二个值是此时和触发时位置相关的将要移动的距离。不论何种类型的手动控制命令，该值总是被表达为一段距离。两个值都是用编码计数位单位来表达的。

给电机的触发条件和给回零运动的设置是一样的：

- Ix03的第17位指定了是否用输入标志来产生一个触发器，还是用警告跟随误差限制状态位作为一个触发器（“扭矩限制触发器”）：0，用标志；1，用误差状态位
- 如果用输入标志来产生一个触发器，Ix25指定了标志寄存器。
- 如果用输入标志来产生一个触发器，给该标志设置的编码器/标志I变量2和3指定了何种信号的哪一个边缘将导致触发。
- Ix03第16位指定了是用将硬件捕获计数器值作为触发器位置——适合于增量编码器信号，实时信号或模拟信号，还是用软件可读位置来作为触发位置——适合于其它类型的反馈（0，表示用硬件捕获位置；1，表示用软件可读位置）。但是如果跟随误差状态用作触发器，则必须用软件可读位置。

PMAC在给前置触发运动的命令的时候将会有效地使用手动控制参数Ix19到Ix22，而且这些参数的值在给后置触发运动的触发时刻将仍然有效。

在触发时，位置传感器的捕获值将存储在专用的寄存器中，以备后面使用时需要。其单位是计数位；

对于增量编码器，它们和启动/重置位置相关。

PMAC在触发微动的开始将设置电机的“进程中回零”状态位（对于一个？命令返回的第一个电机状态字的第10位）为真（1）。在发现了触发器时，或者在运动的结束时，这一位将被设置为假（0）。

PMAC在触发微动的开始也将设置电机的“触发运动”状态位（对于一个？命令返回的第二个电机状态字的第7位）为真，并且保持该状态直到运动结束。如果在运动中发现了一个触发器，那么在后置触发运动结束时这一位将被设置为假；可是，如果前置触发运动完成且没有发现触发器，那么在该运动的结束时这一位仍然将保留为真。所以，这一位能够被用来在运动结束时说明是成功地触发还是没有。电机的“理想速度零”状态位可以被用来决定运动的结束。

## 回零运动控制：

### 回零加速度：

回零运动的加速度是由同手动控制运动的参数一样的参数来控制的——Ix19,Ix20和Ix21。它们在上一部分有详细的描述。

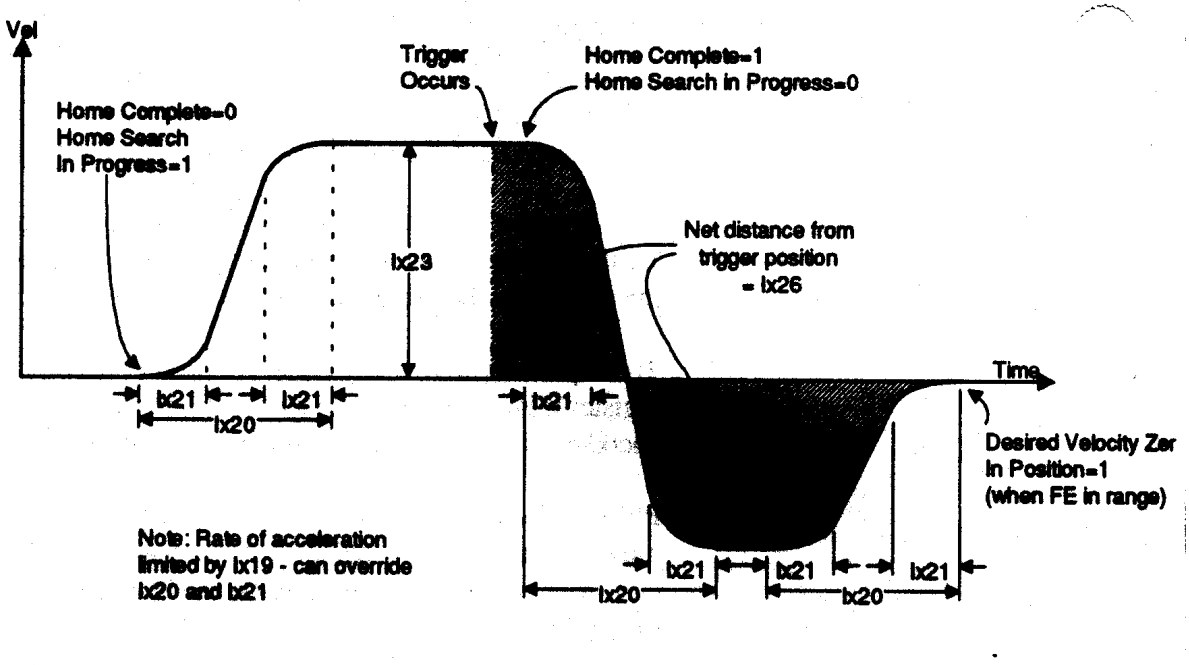
### 回零速度：

回零速度和方向是由Ix23来指定的。如果Ix23比零大，回零运动将会是正向的。如果它比零小，那么该运动将会是反向的。而Ix23的值将控制运动的速度（以计数位/秒的单位）。

### 回零触发条件：

PMAC的回零运动利用了内置在DSPGATE IC上的硬件位置捕获功能。因为对于实际的捕获来说，不要求软件的动作，因此它将是令人难以置信的迅捷和准确（延迟小于100纳秒）。这就意味着不论电机的速度如何，捕获总是十分准确的，因此让回零运动减慢从而获得一个准确的捕获的操作是没有必要的。

**（PAGE 3－126 回零运动轨迹图）**




指定标志设置：

在设置基本电机中，Ix25指定了电机所用的是何种标志设置（和编码计数器之一相联系）。为了利用好PMAC准确的硬件位置捕获功能，电机的位置编码器号和标志号相互匹配是非常重要的（例如，如果你将ENC1用作你的位置环反馈，那么你应该使用标志1——HMFL1、＋/－LIM1、FAULT1——作为你的标志，并且用CHC1作为你的编码器索引通道）。

软件捕获选项：

如果位置环没有使用正交编码器反馈，但仍然需要执行一个回零运动，那么你必须将位置环反馈地址参数Ix03的第16位设置为1，以告诉PMAC它将不能使用硬件捕获功能，因此它必须用到软件捕获技术。举例来说，如果给Ix03的地址是＄0724，那么为了回零位置的软件捕获，Ix03应该被设置为＄10724。

当用到软件捕获时，在实际的触发器和PMAC位置捕获之间存在潜在的几毫秒延迟。这可能导致位置捕获的不准确性；为了达到捕获的足够准确度，触发时电机的速度必须保持足够低。在这些类型的系统中，通常用到的是一个两步的过程，第一步是一个快速的，不准确的捕获，紧接着是一个慢速的，准确的捕获。

触发信号和边缘：

一旦你用Ix25指定了给电机的标志设置，你必须使用编码器/标志I－变量2（I902，I907，等等）来告诉PMAC，是用一个标志，或是用这索引通道，还是将两个都用作捕获触发器，还有就是标志和(或)索引通道将使用哪一个边缘。

接着你必须用编码器/标志I－变量（I903，I908，等等）来指定四个标志（HMFLn、＋LIMn、－LIMn、FAULTn）中的哪一个被用作该捕获动作。如果你的回零捕获用到一个限制或一个错误标志，那么你必须通过设置Ix25的最高位使正常的输入功能无效，至少在回零运动期间应该这么做。（参见下面的示例）

扭矩模式触发：

通常，回零运动、触发微动(如硬件停)运动和运动程序中的触发运动的触发条件是一个输入标志信号的阶跃。有时要求在遇到一个障碍时发生触发。为了支持这种类型的官能度，PMAC允许用

一个警告跟随误差条件代替输入标志作为一个触发条件。这有时也被称为“扭矩模式”触发，这是因为它是在一定的扭矩量级上才能有效地触发（速度模式的放大器除外），而这是由于输出扭矩是和跟随误差成比例的。它也可以被称为“扭矩限制”模式，因为它给产生在一定扭矩限制内的运动提供了一个方便的途径，并且该运动在达到扭矩限制时将会停止。

为了使“扭矩模式”触发有效，应该设置位置环反馈地址I一变量Ix03的第17位为1。Ix03的第16位也应该被设置为1，以便告诉PMAC在一个捕获中使用软件读入位置来代替硬件锁定位置，因为在这种模式下没有输入信号来锁定位置。第0到15位容纳了反馈的实际位置。例如，I103的默认值是＄0720，指定了编码器转换表第一个入口的地址，也指定了其作为信号基底触发。如果I103被改变为＄30720，使用一样的寄存器给反馈，但是现在扭矩模式被指定了。

在这种模式下，给电机的回零运动或者触发运动的触发是警告跟随误差错误状态位的真状态。给电机的警告跟随误差的量值是通过Ix12来指定的，所用的单位是一个计数的1/16。当PMAC检测到该阶跃时，它读入现在的位置作为触发位置，然后移动到和该位置相邻的位置。在一个回零运动中，一段相邻的距离是由Ix26来指定的，所用的单位是一个计数的1/16。在一个触发微动运动中，这段距离是由手动控制命令的第二个值指定的——在^箭头之后的值——单位是计数位。在一个运动程序的触发运动中，这段距离是由轴命令的第二个值来指定的——在^箭头之后的值——单位是用户轴的单位。

在许多情况下，这些类型的运动中要求将Ix69的命令输出设置为一个代表扭矩或力的限制的低值，以确保在运动中，不论触发前还是触发后，任何时候都不会被超过该限制。

注意，如果警告跟随误差状态位在运动开始时是“真”，那么触发将会立即发生。

双重触发器的优点

使用回零开关和索引通道的组合作为回零触发条件只是一个普通的实践。一个编码器的索引通道除了精确和可重复之外，在大多数的应用中并不是独立的，因为电机可能移动不止一转。回零开关是独立的，但是却并不是非常精确，也不是可重复的。通过使用二者的逻辑组合，你就能够从开关处得到独立性，而从索引通道中得到精确性和可重复性。在这种方案中，回零开关被有效地用来选择哪一个索引通道脉冲将用作回零触发器。

尽管在这种应用中回零开关不需要被设置得非常精确，在同样的两个索引通道脉冲之间触发边缘保持安全是十分重要的。而回零开关脉冲应该是足够宽以便总能容纳至少一个索引通道脉冲。

**触发后的动作:**

在回零运动中,一旦PMAC固件确认硬件触发已经发生,它将采取一系列相应的动作。它读入在捕获时刻的位置，通常是硬件捕获寄存器，并且用它和Ix26回零偏置参数来计算新的零位置。一旦执行，报告的位置将参考该新的零位置（在轴定义声明中加上或减去任何轴偏置；如果轴的定义是＃1一〉10000X＋3000，则回零位置将被报告为3000计数位）。

如果用到了软件超行程限制（Ix13，Ix14不等于零），那么在搜寻触发器之后它们将使能，而在搜寻触发器期间软件超行程限制是自动无效的。然后将计算新的零位置的轨迹，如果必要的话，还将包括减速和反向。注意，如果一个软件限制太接近于零，电机在它达到限制之前将不能停止和反向。电机将停止在位置控制状态下，并使它的命令位置等于回零位置。如果有一个跟随误差，那么实际位置将和命令位置之间将有一个跟随误差的差距。

**回零命令:**

回零运动既能够通过一个在线命令（可以从一个PLC程序中用COMMAND“”语法来给出），

也可以通过一个运动程序声明来执行。

在线命令:

一个回零运动可以用在线电机描述命令HOME（缩写为HM）来初始化。这是一个启动搜索运动的简单命令；PMAC不提供运动已完成的自动指示，除非你设置好确认“在位”（IPOS）中断。

*给完成的监视:*

如果你从主机或从一个PLC程序来监视电机是否完成了回零运动，你最好用？命令或M一变量来查看“home complete”和“desired velocity zero”电机状态字。在启动/重置时“home complete”位被设置为零；在一次回零运动的开始时它也将被设置为零，即使前一次回零运动成功地完

成了。一旦在电机停止之前，在回零运动当中发现了一次触发，它将被设置为1。

在运动中“home search in progress”位是“home complete”位的反：它在直到触发时都是1，在触发后立即变为零。因此监测也应该查看“desired velocity zero”位是否为1，从而表示运动是否结束。

*错误监视*

一个强大的监测算法也要搜寻在一个错误的条件下回零运动能够结束的可能性。通常这只是一直被执行的总的错误监测的一部分，包括寻找超行程限制，致命跟随误差，和放大器错误。如果在回零运动期间发生了一个错误，区分它是在触发之前发生的还是在触发之后发生是十分重要的。如果错误是在触发之后发生的，PMAC将知道哪里是回零位置，并且回零运动将不必再重复。一旦错误原因被更新，电机只用命令J＝0就可以移动到回零位置。

缓冲的程序命令：

回零运动也能够在一个运动程序中用HOMEn命令来发出命令，其中n是电机号。注意，该命令指定一台电机，而不象别的运动程序命令指定一个轴的运动。在一个运动程序中，PMAC自动给运动结束的顺序例程编程。当运动成功地被完成，程序将连续地执行下一条命令。

多重回零运动可以通过指定电机序号的一张列表或一个范围的命令（例如，HOME1，3或HOME2..6）来一起执行。进一步的程序执行将等待所有这些电机完成它们的回零运动之后。独立的回零命令，即使是在同一线上（例如，HOME1 HOME2）也将按顺序执行，在第二个运动开始之前，第一个运动必须已经结束。从一个单独的运动程序并行重叠的执行回零运动是不可能的。

需要仔细注意的是，在线命令和缓冲命令之间语法上的差别。在线命令只是简单的HOME或HM，并且它只对当前选址的电机起作用，因此在命令之前必须指定电机序号（例如，＃1HM）。在缓冲命令中，电机序号是命令的一部分，是紧接在HOME或HM之后的（例如，HM1）。

**从一个PLC程序中回零：**

PMAC PLC程序能够通过用COMMAND“”声明发出在线命令产生回零运动（例如，COMMAND“＃1HM”）。这些命令将启动回零运动；如果需要的话，代码必须写入监测器中。电机序号必须在描述命令字符串中加以指定，或者使用ADDRESS＃n声明；没有这个声明，在PLC程序中电机选址将不是模态的。

**对PLC程序回零运动：**

下面的表总结了使用运动程序和PMAC PLC程序进行回零之间的差别。

| 运动程序 | PLC程序 |
|---|---|
| 程序执行点将保留在容纳回零命令的那一行，直到回零运动完成。 | PLC将不会自动地监测回零运动的启动和结束。 |
| 回零命令可以用编程的轴运动来组合。 | 轴运动只能通过手动控制命令来执行。 |
| C.S.必须准备执行一个运动程序。 | C.S.不必准备执行一个运动程序。 |
| 只能回零由C.S.中执行的运动程序定义的电机。 | 能回零C.S.中执行的运动程序没有定义的任何电机。 |
| 电机可以被同时回零，或者一个接一个地回零，或者两者的任意组合。 | 电机可以按照任何顺序被回零。包括在一台电机的回零运动中间启动另一台电机的回零运动。 |
| 运动程序必须通过一个在线命令，一个PLC程序，或别的运动程序来启动。 | PLC能够通过一个在线命令，一个PLC程序，别的运动程序来启动，或在启动/重置时自动启动。 |

**零移动回零：**

如果你希望不用命令任何移动就声明你的当前位置就是回零位置，那么你可以用HOMEZ（在线）或者HOMEZn（运动程序）命令。这些命令和HOME命令很相似，除了它们立即将当前位置作为零位置

。使用HOMEZ命令时，不需用到Ix26偏置。

注意：如果当你给出HOMEZ命令时有一个跟随误差，那么在HOMEZ命令之后报告的实际位置将不是确切的零；它将等于跟随误差的反。

**回零到一个限位开关：**

将一个限位开关用作一个回零开关是可能的。但是，如果你希望运动能够正常地完成，你必须首先使限位开关的限位功能失效；如果你没有做这一步，限位功能将使回零运动作废。即使这样，回零位置也已经被设置了；命令J＝0可以被用来使电机运动到回零位置上。

注意：限位开关的极性和许多人们所期望的相反。－LIMn输入将和限位开关的行程的正端相连；而－LIMn输入将和行程限位开关的行程的反端相连。

为了使限位开关的限位功能失效，你必须设置给电机的变量Ix25的第17位为1。例如，假如I125通常是＄C000（默认的），指定了电机1的＋/－LIM1的使用，而设置它为＄2C000将使限位功能失效。

用回零偏置参数Ix26来使你的回零位置到限位开关之外将是一个好办法，这样你就可以在回零运动之后立即使限位重新有效。

下列示例显示了执行这种类型的回零的两种快捷的方式。一种是用到了运动程序，另一种是用到了PLC程序。用在线命令也可以执行同样的功能。

```
；**************运动程序设立变量（将被保存）*****************
CLOSE
I123＝－10              ；回零速度10cts/msec 反向
I125＝＄C000            ；用标志1给电机1（限位有效）
I126＝32000             ；回零偏置为2000cts
                  ；（足够你到限位之外）
I902＝3                 ；在启动标志和启动索引时捕获
I903＝2                 ；用＋LIM1作为标志（反向端开关）

；**************运动程序执行部分 ***************************
OPEN PROG 101 CLEAR
I125=$2C000             ；使＋/-LIM限位无效
HOME1                   ；回零＃1电机到限位中，并设偏置
I125=$C000              ；使＋/-LIM限位重新有效
CLOSE                   ；程序结束


；**************PLC设立变量（将被保存）**********************
CLOSE
I123＝－10              ；回零速度10cts/msec 反向
I125＝＄C000            ；用标志1给电机1（限位有效）
I126＝32000             ；回零偏置为2000cts
                  ；（足够你到限位之外）
I902＝3                 ；在启动标志和启动索引时捕获
I903＝2                 ；用＋LIM1作为标志（反向端开关）

M133－〉X：＄003D，13，1  ；Desired Velocity Zero bit
M145-〉Y:＄0814,10,1       ；Home complete bit

；************** PLC程序的执行部分 ************************
```

```
OPEN PLC 10 CLEAR
I125=$2C000          ；使＋/-LIM限位无效
CMD“＃1HM”           ；回零＃1电机到限位中，并设偏置
WHILE（M145＝1）      ；等待回零运动启动
ENDWHILE
WHILE（M133＝0）      ；等待回零运动完成
ENDWHILE
I125＝＄C000         ；使＋/-LIM限位重新有效
DIS PLC10            ；一旦发现回零位置，则使PLC失效
CLOSE                ；PLC结束
```

**多步回零进程：**

你可能会要求一个回零进程，该回零进程是不能用一个单独的PMAC回零运动来执行的。在这种情况下，你将会使用两个（或更多）回零运动，并且在两个运动中改变运动参数。尽管这用一系列的在线命令也能够被完成，但编写一个小的运动程序来执行它可能更容易些。

向哪一个方向回零：

在这些情况中最普遍的就是，当启动时不知道在启动触发器的哪一边。在这种情况下，必须移动到一个限位开关处，以确认自己是在行程的一端（这可以通过回零到限位开关处的步骤来完成，就象上面的例子那样）。然后你就可以沿另一个方向执行回零运动，直到一个实时的回零触发器。

一个示例的运动程序如下所示：

```
CLOSE OPEN PROG 102 CLEAR
I223＝10             ；回零速度10cts/msec 正向
I225＝＄2C004        ；使＋/-LIM2限位无效
I226＝0              ；没有回零偏置
I907＝2              ；在标志的上升沿捕获
I908＝1              ；使用HMFL2（回零标志）为触发标志
HOME2                ；执行实际的回零运动
CLOSE
```

一个PLC程序进程示例如下：

```
CLOSE
M233ー〉X：＄0079，13，1 ；Desired Vclocity Zero bit
M245ー〉Y：＄08D4，10，1 ；Home complete bit
OPEN PLC 11 CLEAR
I223＝10             ；回零速度10cts/msec 正向
I225＝＄2C004        ；使＋/-LIM2限位无效
I226＝0              ；没有回零偏置
I907＝2              ；在标志的上升沿捕获
I908＝1              ；使用ーLIM2标志（正端限位）
CMD“＃2HM”           ；回零到限制位
WHILE（M245＝1）      ；等待回零开始
ENDWHILE
WHILE（M233＝0）      ；等待回零运动完成
ENDWHILE
I233＝ー10            ；回零速度10cts/msec 反向
I225＝＄C004         ；使＋/-LIM2限位重新有效
I907＝11             ；在标志变低而索引通道变高时捕获
I908＝0              ；使用HMFL2(回零标志)作为触发标志
CMD“＃2HM”           ；执行实际的回零运动
WHILE（M245＝1）      ；等待回零运动开始
ENDWHILE
WHILE（M233＝0）      ；等待回零运动完成
ENDWHILE
DIS PLC11            ；一旦发现回零位置，使PLC无效
CLOSE                ；PLC结束
```

已经进入回零了吗？

还有一个可能产生的相似情况是，在启动时你怎样才能知道你是否已经触到了你的回零触发器了。这儿有一个简单的方法，编写一个评估该条件的程序；如果它在触发中，那么在执行实际的回零之前它将移开。

```
；*************运动程序设立变量（将被保存）*****************
CLOSE
M230—〉X：＄C008，20，1 ；给HMFL3输入的变量
I325＝＄C008          ；用标志3给电机3
；***************运动程序执行部分 ***************************
OPEN PROG 103 CLEAR
IF （M320＝1）          ；已经处于触发状态中？
  I323＝10          ；回零速度10cts/msec 正向
  I326＝1600          ；回零偏置＋100cts（确认已清除）
  I912＝11          ；在标志变低而索引通道变高时捕获
  I913＝0          ；使用HMFL3作为标志
  HOME3          ；“回零”离开开关
ENDIF
I323＝－10          ；回零速度10cts/msec 反向
I326＝0          ；没有回零偏置
I912＝3          ；在标志和索引通道都是上升沿时捕
                ；获
I913＝0          ；使用HMFL3作为标志
HOME3          ；执行实际的回零运动
CLOSE          ；程序结束

；**************PLC设立变量（将被保存）*********************
CLOSE
M320—〉X：＄C008，20，1 ；给HMFL3输入的变量
I325＝＄C008          ；用标志3给电机3
M333—〉X：＄00B5，13，1 ；Desired Velocity Zero bit
M345—〉Y：＄0994，10，1 ；Home complete bit
M350—〉D：＄009E        ；给HMFL3输入的变量
；************** PLC程序的执行部分 ***************************
OPEN PROG 12 CLEAR
IF （M320＝1）          ；已经处于触发状态中？
  I323＝10          ；回零速度10cts/msec 正向
  I326＝1600          ；回零偏置＋100cts（确认已清除）
  I912＝11          ；在标志变低而索引通道变高时捕获
  I913＝0          ；使用HMFL3作为标志
  CMD“＃3HM”          ；“回零”离开开关
  WHILE（M345＝1）        ；等待回零运动开始
  ENDWHILE
  WHILE（M333＝0）        ；等待回零运动完成
  ENDWHILE
ENDIF
I323＝－10          ；回零速度10cts/msec 反向
I326＝0          ；没有回零偏置
I912＝3          ；在标志和索引通道都是上升沿时捕
                ；获
I913＝0          ；使用HMFL3作为标志
CMD“＃3HM”          ；执行实际的回零运动
WHILE（M345＝1）          ；等待回零运动开始
 NDWHILE
WHILE（M333＝0）          ；等待回零运动完成
ENDWHILE
DIS PLC12          ；一旦发现回零位置，使PLC无效
CLOSE          ；PLC结束
```

**保存回零位置：**

PMAC将自动保存在给电机的前一次回零运动中捕获到的编码器位置。该值被保存在电机编码器位置偏置寄存器[Y:$0815(电机1)，Y:$08D5（电机2），等等]，该寄存器在电机没有绝对启动位置时，将在启动/重置时被设置为0。如果Ix10〉0，从而指定了一个从解算器读入的绝对启动位置，使得回零不成为必要，则该寄存器中将保存启动解算位置的反。在别的情况中，它将容纳编码计数器零位置（启动位置）和电机零位置（回零位置）之间的差。

注意：对于V1.14固件之前的版本，这个值能够通过使用PLC程序HOMOFFST.PMC来得到，该程序在用户手册的举例部分有叙述。从V1.14固件版本开始，PMAC将自动存储该值。

用途：

该寄存器有两个主要的用途。第一个，它给使用编码器位置捕获和位置比较寄存器提供了一个参考。这些寄存器都是参考编码器零位置的，该零位置是启动位置，不是回零位置（电机零位置）。该寄存器保存了两个位置之间的差。从编码器位置（通常从位置捕获得到）中减去该值就可以得到电机位置，或者在电机位置上加上该值，就可以得到编码器位置（通常用作位置比较）。

示例：

使一根轴运动直到触发时才停下来，然后将捕获的编码器位置转换成为电机位置，你可以用下列M一变量定义：

```
M103一〉X:$C003,24,S     ；编码器1位置捕获寄存器
M117一〉X:$C000,17       ；编码器1位置捕获标志
M125一〉Y:$0815,24,S     ；电机1编码器位置偏置寄存器
```

现在你可以使用如下所示的运动程序段：

```
INC                  ；增量模式的移动
TM10 TA10            ；移动段时间10 msec
WHILE （M117＝0）      ；当没有给捕获位置的触发时
  X20                ；命令下一个运动段
ENDWHILE
P103＝M103一M125       ；读捕获的位置；减去偏置以得到
                     ；触发时的电机位置
```

该寄存器的第二个用途是决定是否编码计数器已经丢失了任意计数位。这可以通过在一次操作之后执行第二个回零运动，并且将第二次回零运动之后寄存器中的值和第一次回零运动之后的值进行比较，来完成这一功能。

## 开环运动：

正如它们的名字所示，开环运动不执行闭环的位置控制。它们打开伺服环，并只在输出中给定指定的值。它们经常是用在诊断阶段，但是它们也可以用在实际的应用当中。

这些运动是通过使用电机描述在线命令**O**{常量}来执行的，其中{常量}代表了的输出的值，这个值是最大输出参数Ix69的百分比。该命令不能成为一个运动程序的一部分，也不能在一个坐标系正在执行一个运动程序时给电机发出该命令，即使此时电机并未移动。

如果该电机不是通过PMAC来换向，那么这条命令将在该电机的单独的DAC输出上产生一个恒定的直流电压。如果该电机是通过PMAC来换向，该命令将设置给电机的进行正弦交换的两个DAC输出的信号的量值。

为了执行一个可变的**O**命令，应给滤波结果寄存器（X：＄003A，等等）定义一个M一变量，给电机发出命令**O**0并将它放到开环模式中，然后给M一变量分配一个变量值。这个技术在由PMAC换向的电机上也是可以工作的。

PMAC可执行程序调试部分使用开环运动模式以便用户能够诊断和调试放大器的反应。

## §3—9 设置一个坐标系

一旦你设置，调节好了电机，并能控制它们进行微动和回零移动，你就需要为电机设置一个或多个坐标系来运行运动程序。

PMAC有几种协调复合运动的方法，它们或者由PMAC直接控制，或者不是。依靠用户的情况和需要，我们可按照以下所示来完成设置坐标系。

### 什么是坐标系?

PMAC中的坐标系指的是一个或一组为了同步运动的目的而组织起来的电机。一个坐标系(甚至只有一个电机)能运行一个运动程序；而一个电机不行。PMAC可拥有至多8个坐标系，分别是&1到&8，它们的形式非常灵活(例如，可以8个电机8个坐标系，也可8个电机1个坐标系，或每两个电机一个坐标系，共4个坐标系，etc。)。

一般情况下，如果你需要协调几个电机的运动，把它们放在一个坐标系内。如果你希望它们独立地运动，把它们放在不同的坐标系内。不同的坐标系可在不同的时间(或重叠的时间)内运行不同的程序，或在不同的时间内(或重叠的时间内)运行相同的程序。

一个坐标系首先要通过“轴定义语句”(见下)为电机分配轴来建立。一个坐标系至少有一个分配了该坐标系内轴的电机，否则它将不能运行运动程序。当为一个坐标系写下程序时，若希望在同时其它电机也有运动，只需将它们的运动命令写在同一行里，就可以协调它们的运动。

### 什么是轴?

轴是坐标系的一个元素。它类似于电机，但不是同一事物。一根轴由字母来指明。一根坐标系内至多有8根轴，从X，Y，Z，A，B，C，U，V和W中选择。通过将轴分配给电机来定义它，同时有两个参数，放大因子和一个偏置(X，Y和Z可被定义为三个电机的线性组合，U，V，W也可以)。和轴有关的变量为可放大的浮点值。

**一对一的匹配**

在大多数情况下，电机和轴之间为一对一的对应关系。就是说，在一个坐标系内，单个轴被分配给单个的电机。即使在这种情况下，匹配的轴和电机也不是完全同步的。轴由工程单位分度，并仅使用命令位置。除了在PMACH功能下，计算仅仅是从轴的命令位置到电机的命令位置。

**复合电机的轴**

在一个坐标系中，一根轴可被分配给不止一个电机。这在吊架系统中很常见，此时，希望在十字

接头的对立端的电机做同样的运动。通过把一根轴分配给不同的电机，程序中的单个轴上的运动将给复合的电机以完全相同的运动命令。这只是对于两个电机，但在PMAC中，多至八个电机都可使用这种方式控制。记住，电机的伺服环还是独立的，并且实际的电机位置并不一定要相同。

用这种方式协调并行的电机要好于使用主/从技术(可通过PMAC的位置跟随特性来完成，在同步PMAC与外部事件中有叙述)。在主/从技术中，主电机的实际位置是在译码器中被量测的，在带有所有的扰动和量化误差后又变成从电机的命令轨迹，这样它的实际轨迹的误差将更大。电机的命令轨迹的不精确导致难以甚至不可能适当地使用前馈，这将导致一个滞后。事实上，如果主电机得到一个扰动，从电机将观察到它，并试图与之匹配，但如果从电机得到一个扰动，主电机将无法观察到它。

必须注意有紧密机械连接的电机的启动和归位。一般地，电机在启动时并没有理想的直线校直。通常的步骤是让一个有从电机的电机做一个归位运动，接着再往后做一个足够远的偏置，来使第二个电机知道如何到达它的归位位置。然后，让第二个电机成为主电机，并让它做一个归位运动。这时，第一个电机成为它的从电机。这将使第一个电机稍微偏离它的归位位置;然后用J=0命令让它回到归位位置。之后关掉从电机，这样我们就可以使用共同的轴命令来完全相同地控制电机。

**模拟轴**

坐标系中的轴可以不与电机相连(一个“虚拟”轴)，此时，对那根轴的运动编程将不引起运动，即使这将影响其它轴和电机的运动。例如，如果希望一根轴完成一个正弦波的轮廓，最简单的办法是设第二根“虚拟”轴，使它循环地做插入运动。

**轴定义语句**

一个坐标系的建立是通过用轴定义部分来完成的。通过将一个电机(数字编号)与一个或多个轴(字母编号)匹配来完成轴的定义。

电机与轴的匹配

最简单的轴定义语句为#1->X。它仅仅将1#电机分配给当前坐标系的X轴。当在该坐标系中执行一个X轴的运动时，1#电机将完成这个运动。

放大和偏置

轴定义语句也定义了轴的用户单位。例如，#1->10000X也将1#电机匹配给X轴，但这个语句同时设置10,000个编码器步数为轴的一个用户单位(例如英寸或厘米)。这个放大特性几乎总被使用着。一旦在语句中定义了放大倍数，用户就可以在工程单位下编写轴的运动程序而无需再去关心放大倍数的问题。

语句#1->10000X+20000设置了轴的零点与电机零点(回零位置)的距离为20，000步(两个用户单位)。这种偏置较少被使用。进一步地，一个轴定义语句能将一个电机与一个笛卡尔坐标的线性轴组合匹配起来，来允许坐标系的旋转或正交更新。

**轴的类型**

一根轴有以下几种属性。注意，对于大多数轴的功能，它并不关心使用何种类型的轴以及赋给它什么字母。但是，对于某些特性，仅仅可以使用特定的轴的名字

笛卡尔轴

笛卡尔轴为一组两，三根轴中的一根，这组轴使得沿着这根轴的运动成为这两，三根轴运动的线性组合。X，Y和Z组成一套笛卡尔轴;U，V和W组成另一套。另外，有许多命令(NORMAL，周期运动)可独立地通过I，J和K向量来引用X，Y和Z轴。

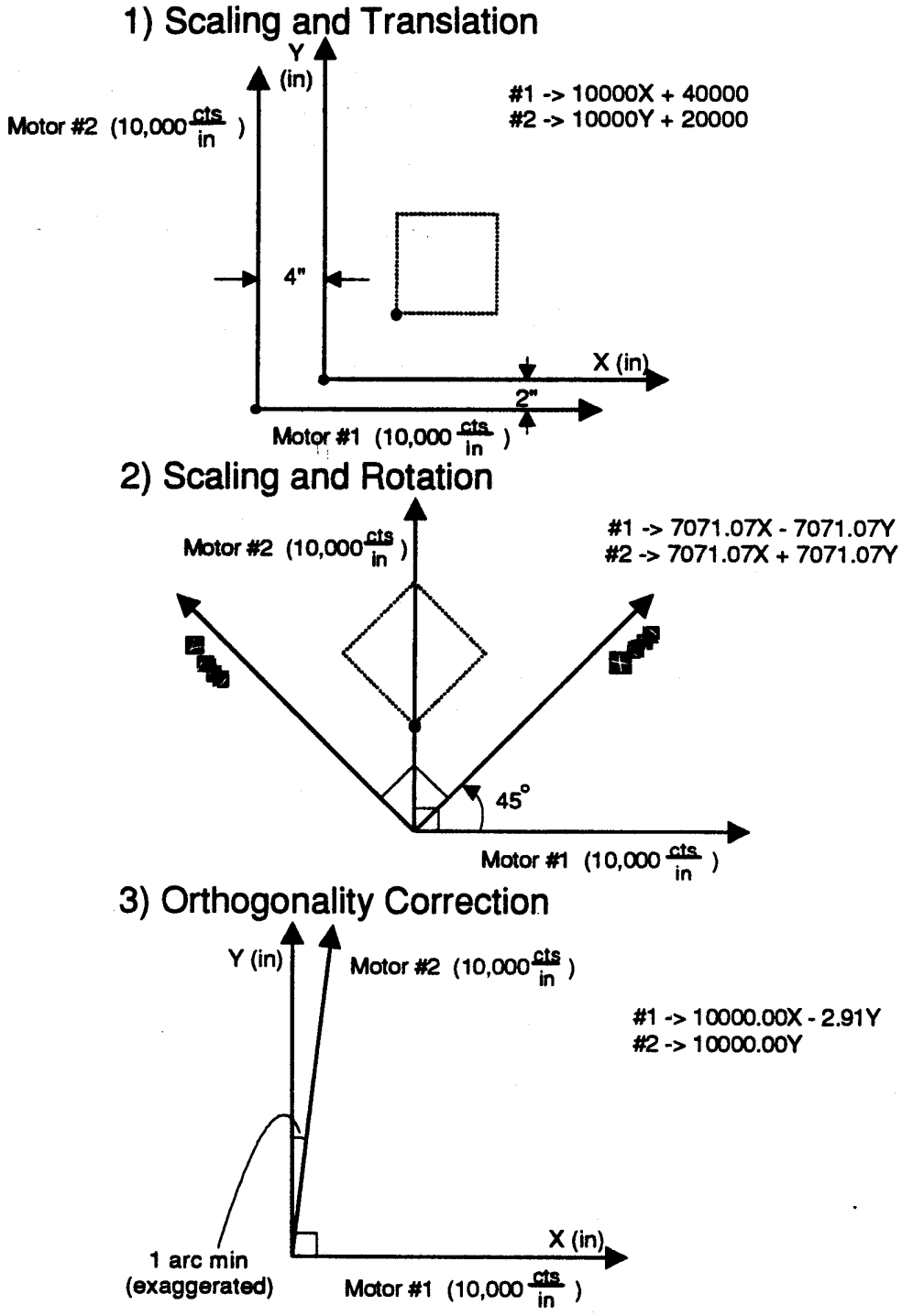
当你希望定义一根几个电机线性组合的笛卡尔轴时，可以通过轴定义语句的扩展形式来完成。例如，你可以通过以下的轴定义语句从你的电机得到一根30°的旋转轴:

```
#1->8660.25X-5000Y
#2->5000X+8660.25Y
```

此时，对于Y轴(或X轴)的运动将导致1#和2#电机同时运动。

## PMAC坐标系定义

**1)放大和平动**




仅仅X，Y和Z笛卡尔轴可被用于PMAC循环插入例程，刀具半径补偿例程和矩阵轴转换例程。如果你想对其它轴做循环插入，你可通过在子程序进行短的三角运动来完成。可见例程序CIRCTRY.PMC

**旋转轴**

旋转轴为允许转动的轴，但不能分配给组合的电机。一根旋转轴必须被命名为A，B或C转动，是一种电机功能，由Ix27来指定，但它只能在把旋转轴分配给电机后才能运行。当在程序中指定一个轴的绝对运动时，转动功能允许电机在旋转范围内采用最短的路径。

**进给轴**

进给轴在坐标系中可对指定进给的运动进行计算。指定进给的运动的时间由进给量除以距离而得。如果其它轴也要在这种状态下进行运动，它们需要在同样的时间内进行线性地插入运算。

缺省的进给轴为笛卡尔轴X，Y和Z。该设置可由命令FRAX(进给轴)进行改变。

**轴与电机位置的再匹配**

在PMAC把指令电机位置转换成命令轴位置的过程中只有一类计算。它在PMATCH(位置匹配)功能中进行。以下几种情况需要这种计算:

首先，当某个电机功能，例如，微动，开环运动，或在放弃或到达限制位置时停止，在最后一次轴的运动(或回零运动)时改变了电机的命令位置。换句话说，轴不知道电机往哪儿去。为了使下一次轴的运动程序能够正确的执行，必须告诉轴它要从哪儿开始。这就需要使用PMATCH功能。

其次，如果有一个绝对位置传感器。需要在第一个运动程序执行之前使用PMATCH功能，因为此时电机一般不会在启动时正处于零点位置(对于增量传感器是这样的)，而我们必须使对轴给出这个起始点。

PMATCH功能转变了在坐标系内轴定义语句中的等式的含义。用轴的命令位置代替了电机的命令位置。如果一根轴分配给几个电机(例如，#1->10000X，#2->10000X)，则编号小的电机使用PMATCH功能来计算。

如果变量I14=1，那么在每次运动程序执行(所有的R和S命令)之前都会自动地执行PMATCH功能。如果位置已经匹配，对PMATCH功能也不会有影响，所以绝大多数用户应让I14=1。只有那些需要不断地快速启动，以致于要避免PMATCH功能花费的1到2个毫秒的额外计算时间，用户可能要使I14=0。

**坐标系的时基是什么?**

每个坐标系都有自己的“时基”来帮助控制在那个坐标系中插入运动的速度。在每个伺服周期内PMAC的插入例程给一个“时间”寄存器以增量。当伺服周期的确切时间已由硬件设置(通过跳线E98，E29-E33，E3-E6)并不改变时，每个伺服周期“时间”寄存器的增量正是内存寄存器中的值。它不必与周期的真实物理时间匹配。

$2^{23}$ (8，388，608)个时基寄存器单位等于1毫秒。时基寄存器的缺省值等于I10的值。出厂时I10的缺省值3,713,707表示缺省的伺服周期物理时间为442微秒。

如果时基寄存器的值通过I10加以改变，插补运动的速度将与程序中给定的速度不同。许多人把这叫做“进给率超速”，注意，物理时间并未改变，所以伺服环的动态性没有改变。

每个坐标系有一个变量Ix93，它包含了坐标系使用的时基寄存器的地址。对于Ix93的缺省值，坐标系通过主机的%命令设置的寄存器中得到它的时基。一个%100命令把I10的值放入寄存器中，而%50命令则把I10/2的值放入寄存器中。

不考虑时基信息源，一个%的查询命令将使PMAC以一个I10百分数的形式返回当前时基的值。

时间基数的信息也可从其它来源得到。指令源时间基数最普遍的来源就是外部的频率源时基，该时间基数的值与主编码器的频率成正比。它提供了一个有力的位置跟随的机械性能，通常叫做”电

子凸轮”。

要使用外部时基，请参考“使PMAC与外部事件同步”指导。

# §3－10 计算特性

PMAC有很强的运算功能，可以分担主机的许多运算，甚至可通过以前不可能的方法进行独立的计算。许多数学，逻辑和超越函数的计算可以通过用户程序中的变量和常数进行。

## 运算优先级

作为一种多任务，实时的计算机，PMAC有一个详细的优先级排列来保证在需要时完成重要的任务，并使所有的任务尽快地完成。PMAC尽可能地使面对用户的优先级表显得简单，但同时也给那些有特殊需要的用户优化控制提供了弹性。优先级排列如下:

### 1. 单字符I/O

在字符输入和输出时，串行口和主机接口(PC或STD)有最高的优先级。每个字符占用时间仅为200纳秒，使它有这种高优先级的原因是为了保证PMAC在进行字符操作时不会失去主机的控制。在PMAC的总计算时间中它不会成为主要部分。注意，该任务不包括处理一整条命令;后者的优先级较低。(见下)

### 2. 换向更新

PMAC的换向（相位)更新的优先级为第二级。电机的换向操作每个更新周期(对于30MHZ的PMAC卡为2毫秒)占用3毫秒。该任务的频率由主频和跳线E98，E29-
E33决定。更新频率的缺省值为9KHZ(每周期110毫秒)。在该缺省值下，每个电机的换向操作将占用PMAC运算能力的近3%。

### 3. 伺服更新

伺服更新——计算新的指令位置，读入新的实际位置，并通过两者的差计算指令输出——
的优先级为第三级。对于PMAC上每个激活电机，该任务每个更新周期(对于30MHZ的PMAC卡为20毫秒)占用30毫秒，一般的伺服任务，例如编码器转换表，要加上30毫秒。该任务的频率由主频和跳线E98，E29-
E33，E3-
E6决定。修正频率的缺省值为2。26KHZ(每周期442毫秒)。在该缺省值下，每个电机的伺服更新将占用PMAC运算能力的近7%。有关优化这个更新率的讨论可见“闭环伺服环”部分。

### 4. 实时任务中断

实时中断（RTI）任务在PMAC中的优先级为第五级。它们在伺服更新任务之后立刻发生，其频率由参数I8控制（每I8+1个伺服更新周期）。在这一优先级下有两个主要的任务：PLC0和运动程序准备。

PLC程序0

PLC0是一个特殊的PLC程序，它执行的优先级要高于其它的PLC程序。这意味着它仅在那些执行频率要高于PLC程序的少数任务中使用。PLC0与前一个RTI完成之后的任务一同执行。从扰乱了任务执行的顺序这一点来说，PLC0可能是PMAC中最危险的任务。如果它太长了，它将使后台的程序得不到时间来执行。你首先会发现通讯和后台的PLC程序将变得缓慢起来。在最坏的情况下，将会触发监控时钟，关掉PMAC卡，因为后台的辅助处理程序将没有时间来更新它。

运动程序准备

运动程序准备在运动程序执行的路线上一直工作着，直到遇见下一个运动或停止命令，并为下一次运动计算移动公式。每当PMAC开始执行一个新的运动，它就设一个内部标志，示意在程序内为下一个运动进行准备。该准备发生在下一个RTI时。

### 5. VME邮箱处理

由VME邮箱寄存器读出或写入不超过16个字符的操作具有第四优先级。它发生的速率由主机控制。它在PMAC的运算能力中只占很小一部分。

### 6. 后台任务

在没有任何更高优先级的任务要执行时，PMAC将执行后台任务。共有三个基本的后台任务:命令处理，PLC程序1-
31和辅助处理程序。这些后台任务执行的频率由PMAC的运算大小决定:被执行任务的优先级越高，后台任务执行的越慢;后台任务越多，它们执行的越慢。

PLC程序1-31

PLC程序1-
31是在后台执行的。每个PLC程序在执行一遍扫描时(到结束或到一个ENDWHILE语句)，不会被任何其它的PLC程序打断(虽然它可被更高优先级的任务打断)。在每个PLC程序之间，如果需要的话，PMAC将执行辅助处

理程序或响应主机命令。

编译PLC程序1-31

编译PLC程序1-31是在后台执行的。所有有效的PLCC程序按照从低到高的顺序进行扫描时(到结束或到一个ENDWHILE语句)，不会被任何其它的PLC程序打断(虽然它可被更高优先级的任务打断)。在加电\重启PLCC程序在第一个PLC程序运行之后运行。

主机命令响应

从任何接口接受控制字符对PMAC是一个信号，它必须对命令作出响应。最普遍的控制字符为回车(<CR>)，它告诉PMAC将刚才所有的字母和数字字符作为一个命令行。另一些控制字符有它们自己的意义，而与接收的字母和数字字符无关。之后PMAC将执行适当的操作，如果接收了一个非法命令，它将向主机报错。

资源管理

在每次对每个后台PLC程序的扫描之间，PMAC将执行辅助处理程序来进行正确地更新。它们中最重要的是安全性检查(跟随错误，超行程，错误，监控时钟，等等。)。尽管它的优先级很低，但确保了一个最低频率，因为如果该频率太低，看门狗计时器将被触动，关掉PMAC卡。

优先级优化

PMAC通常具有足够的速度和计算能力来完成所要求的所有任务而无需用户担心。一些功能在确定的优先级下的要求很高，为了使PMAC更有效率地工作，有必要对一些优先级作出优化。

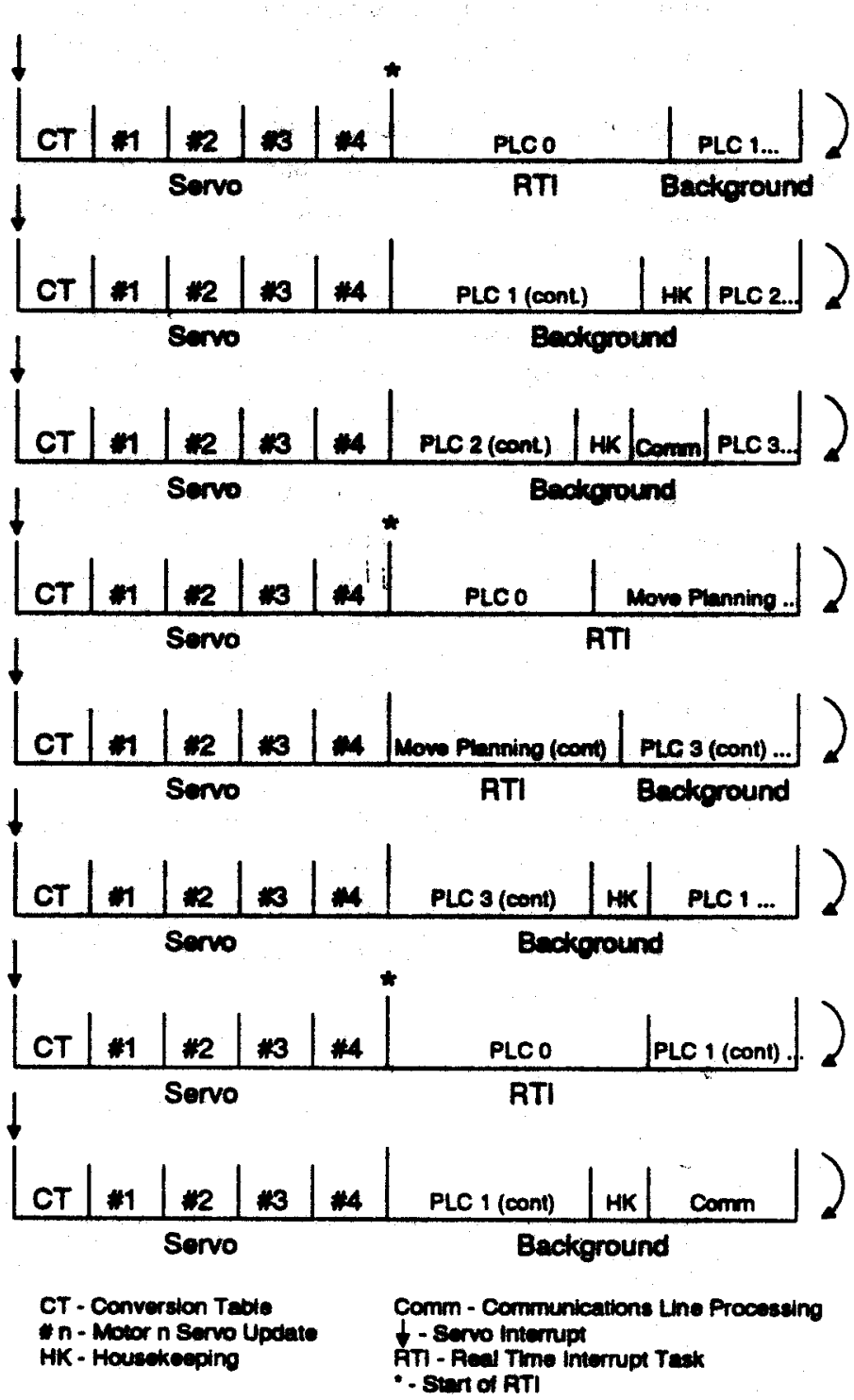
当PMAC的处理时间不够时，类似通讯迟缓，PLC/PLCC扫描速度变慢，运行时间错误，甚至触发看门狗计时器的问题就会发生。解决以上现象的办法将在手册的这一部分加以讨论。对于这些问题的解决办法分两步，首先，高优先级的任务可被减慢或移至较低的优先级。应对类似编码器转换表，PLC/PLCC0和实时中断(RTI)等任务加以分析，看看是否这些任务都是必须的或其中一些能不能移至较低的优先级或减慢。例如，一个5轴的任务可能不需要编码器转换表6到9，PLC0可能可作为PLCC0来完成或RTI可以每4或5个周期运行一次。

其次，可以调整任务的优先级，使它们变得不重要。大的PLC程序可被分成几个小的PLC程序。通过在PLC扫描中给出更多的中断来增加辅助处理和通讯的频率。运动程序中的WHILE (条件) WAIT语句如下所示:

```
WHILE (条件)
    DWELL20
ENDWHILE
```

这将给其它的RTI，例如运动准备和PLC/PLCC0以更多的时间。

PMAC多任务示例




## 数值量

PMAC可以许多方式来存储和处理数值量，包括定点和浮点值。PMAC的CPU，Motorola 56000 DSP是一个具有嵌入的24位和48位算术能力(加上一个56位的累加器)的定点处理器。但PMAC的固件有一整套浮点例程。

### 内部格式

内部伺服，插入和换向例程都由24位和48位的定点运算以最大的速度来完成。用户程序，运动程序和PL

C，以最大的范围和一般性使用浮点运算来完成。甚至在向定点寄存器读或(和)写时，中间格式都是浮点格式。该规则唯一的例外是在编译新的PLC程序时;在只包括“L变量”和整形常数的语句中，中间格式为带符号的24位整形值。欲了解细节可参考“编写PLC程序”的编译PLC部分。

一般的浮点格式为48位长，包括一个36位的尾数和一个12位的指数。提供的范围是$\pm 2^{2047}$或$\pm 3.233 \times 10^{616}$，这个范围对于PMAC任何可预见的用途都足够了。

**接收数值**

作为主机命令行的一部分，常量以ASCII码，十进制或十六进制的形式送给PMAC。十六进制值必须以$为前缀；它们必须是无符号的，而且不能包括分数部分。十进制值可以是正的或负的，而且可以包括分数部分。PMAC的数值解释器不支持指数表示法，它的值的范围为$\pm 2^{35}$或$\pm 3.43 \times 10^{10}$。超出该范围的值从最大或最小值处截断。

例子:
```
1234
3
03          (允许前导零)
-27.656
0.001
.001        (不需要前导零)
$FF00       (十六进制数)
```

**显示数值**

作为响应行的一部分，PMAC以ASCII码的形式在主机上显示数值量(如果I9=2或3，也可以十六进制的ASCII形式显示地址值)。显示值的范围为$\pm 2^{47}$或$\pm 1.41 \times 10^{14}$。
超出该范围的值将从最大或最小值处截断。

## 地址

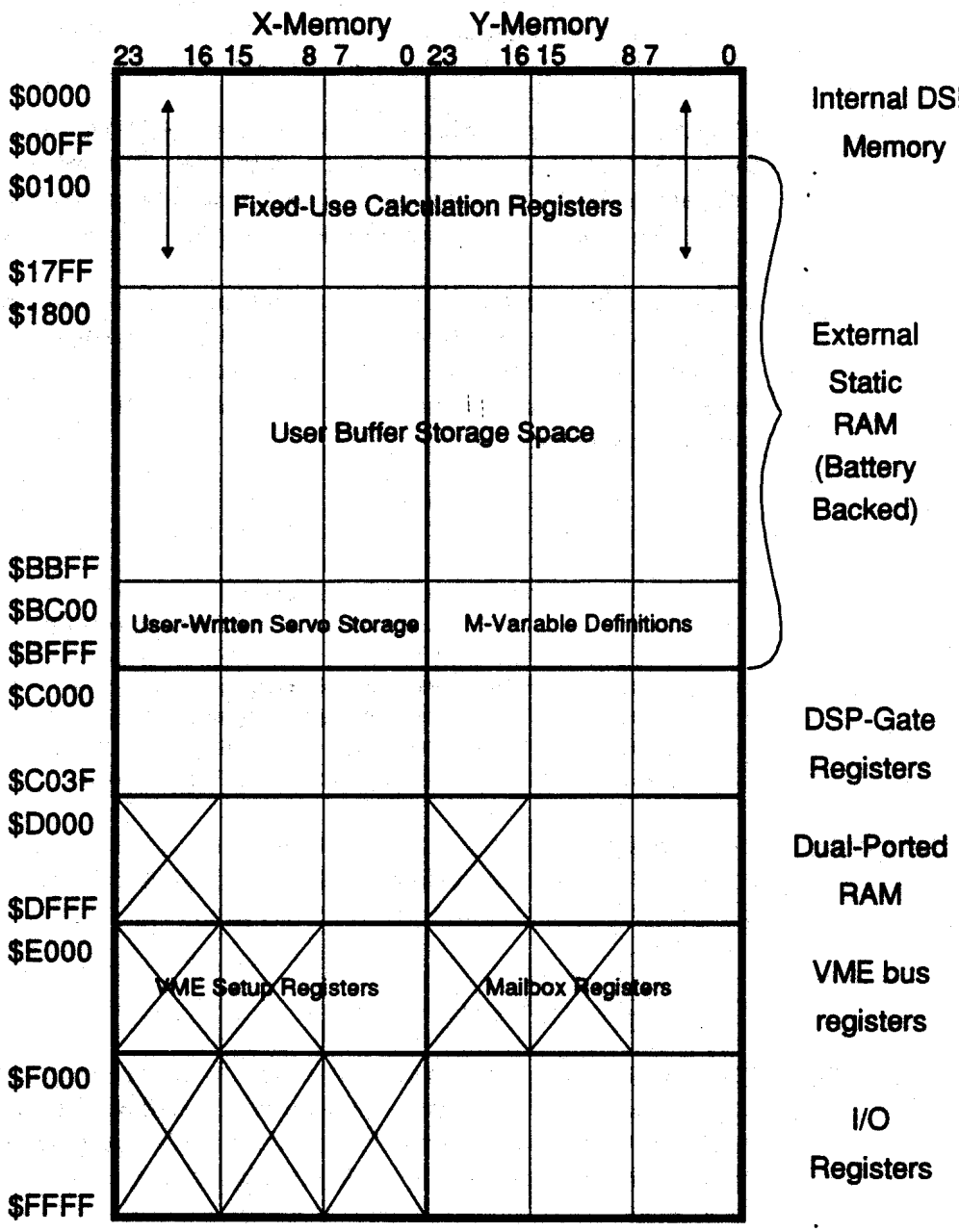
PMAC使用Motorola56001作为它的处理器。56001的内存和I/O有双16位地址空间。(注意，PMAC的I/O和内存在一起;它不象你的PC机那样有独立的I/O空间。)当指定一个地址时，你必须指出使用哪一半内存(X或Y)--还是作为一个48位字来使用(L)--
之后是一个可选的冒号，最后是地址。地址必须是在0-65536(十六进制的$0-$FFFF)之间的一个常量。

不要把PMAC的内存和I/O地址和主机的混淆起来。

下面是合法的地址:
```
Y:$FFC2(包含I/O地址的字)
X:1824      (插入编码器1的位置)
X$C003      (俘获编码器1的位置)
Y49155      (DAC1的输出值)
```

这种指定地址的格式用于M变量定义和直接读/写命令中。也有指定地址的I变量，但通常都预先指定了是X，还是Y部分。所以只需要数字值。数据采集地址I变量(I21-I44)在数字值之前使用额外的六位数指定哪半部分内存(可见I21的描述)

## PMAC内存分配图




## 变量

PMAC有许多种类型的变量。在PMAC中，一个变量是由一个跟随着0到1023的数字的单个字母(I，P，Q或M)指定的。每个字母代表不同类型的变量，每种类型有自己的特点。不同类型的变量有这样一个共同的特征，当在一个表达式内引用它们的名字时，将使用这些变量的当前值(读出);可通过等式给它们赋值(写入)。

在PMAC中，用户不可以定义他自己的变量名；虽然PMAC执行程序的编辑器有一选项(“macro”)允许程序用用户定义的变量名编写，但在下载过程中将把这些变量名转变成PMAC中合法的变量名。

**I变量**

I变量(初始化或设置变量)对PMAC卡功能特性进行设置。它们在内存中有固定的位置，并有预先定义好的意义。大多数为整形量，特定变量的值的变化范围是不一样的。共有1024个I变量，从I0到I1023，如下所示:

| | |
|---|---|
| I0--I79: | 一般的PMAC卡设置 |
| I80--I99: | 连接旋转变压器设置 |
| I185--I199 | 坐标系1设置 |
| I200--I284 | 2#电机设置 |
| I285--I299 | 坐标系2设置 |
| … … | |
| I800--I884 | 8#电机设置 |
| I885--I899 | 坐标系8设置 |

I900--I979　　1-16编码器设置
I980--I1023　　为将来的用途保留

赋值

可将表达式或常量的值赋给I变量。如果赋值时没有打开的缓冲区，或命令编程缓冲区是打开的，这个命令是在线命令(立即执行)。

*举例:*

```
I120=45
I120=(I120+P25*3)
```

取值范围

I变量有取值范围的限制，当试图赋给它一个超出范围的值时不会导致错误。通过取模操作(截断)，该值将自动“转换”至范围以内。例如，I3的取值范围为0到3(4个可能值)。命令I3=5实际上将把4除5的模数1赋给I3。

电源故障备份

对于有后备电池RAM的PMAC，大部分I变量值通过SAVE命令存在一个2Kx8 EEPROM中。即使在后备电池RAM发生故障时它们也是安全的，所以卡的基本设置不会丢失。在给某个I变量以新值后，应使用SAVE命令以使该值在发生电源故障或重启之后能保持下来。

没有存进EEPROM中的I变量将保存在后备电池RAM中。这些变量不需要SAVE命令来保存，电源故障或重启之后，以前的值将不能保存下来。这些变量是:I19-I44，Ix13，Ix14。

对于有后备闪存的PMAC(Option 4A，5A或5B)，所有的变量可通过SAVE命令存在闪存内。如果板上有一块EEPROM，它不会被使用。在给某个I变量以新值后，应使用SAVE命令以使该值在电源发生故障或重启之后能保持下来。

缺省值

所有I变量的缺省值都保存在厂家提供的固件内。它们可通过I{constant}=*命令单独使用或通过I{constant}…{constant}=*命令使用某一个范围内的缺省值。在非缺省设置时可通过$$$***命令或跳线E51进行重新初始化，所有的缺省设置将被从固件拷入内存。最后存入的值并没有丢失；只是它们不被使用了。

**P变量**

P变量是全局用户变量。它们在内存中有固定的位置，为48位的浮点变量，但没有预先定义用途。共有1024个P变量，从P0到P1023。一个给定的P变量与其它P变量是一样的；所有坐标系都可以读写所有的P变量(与Q变量对比，它只能被给定的坐标系读写)。这就允许在不同的坐标系间传递有用的信息。P变量在程序中可任意使用:位置，距离，速度，时间，模式，角度，中间计算，等等。

数组

*数组读入*

可以把一些P变量当作数组使用。这样做使得从变量读取值变得很容易；只需用圆括号内的表达式代替指定变量的常数--用P({expression})代替P{constant}。PMAC将把它作为一个功能调用，象SIN({表达式})一样。

举例

如果你想顺序地把P101到P200作为位置输入，你可使用下面的这段语句。

```
F10
P1=101              ; 数组检索变量
WHILE (P1<201)      ; 开始循环
    X(P(P1))        ; 随着P1改变，目标位置也会改变
    DWELL100
    P1=P1+1         ; 给检索变量增量
ENDWHILE
```

*数组写入*

向作为一个数组的一组变量赋值要复杂一些;它要通过间接的选址技术来完成。首先，定义一个指向P0的M变量(例如，M0->L:$1000)。接着，定义第二个指向在Y:$BC00中的第一个M变量定义字的低12位的M变量(例如，M10->Y:$BC00，0，12，定义M10为指向M0定义字的低12位的M变量)。如果我们想指向M1的定义字，应使用Y:$BC01;指向M2，使用Y:$BC02;指向M100，使用Y:$BC64(十六进制的64即为十进制的100);指向M1023，使用Y:$BFFF。

现在，通过给第二个M变量赋值，你可改变第一个M变量指向哪一个P变量。在我们的例子里，命令M10=

5使M0指向P5。

一旦第一个M变量已经指向了一个特定的P变量，给该M变量赋值将会把该值写入它指向的那个P变量。继续我们的例子，命令M0=73将把73写入P5。

举例

如果我们想用P0到P359建立一个每一度一个值的正弦表，你可使用下面的这段语句。(这里假定M0和M10已由上面的语句设置):

```
P1000=0                 ; 数组检索变量起始值
WHILE (P1000<360)       ; 开始循环
    M10=P1000           ; 将M0指向适当的P变量
    M0=SIN(P1000)       ; 给该P变量赋正弦值
    P1000=P1000+1       ; 给检索变量增量
ENDWHILE
```

P变量的特殊用途

如果一个只包含常量值的命令送给PMAC，PMAC将把该值赋给P0。(除非定义了一个特殊的缓冲区表，例如一个补偿值表或激励输入表，并没有被填满--
在这种情况下，该常量将被填入表中。)例如，如果你发出一个342<CR>命令给PMAC，它将解释为P0=342<CR>。

这一功能是为了便利简单运算符的终端界面。它意味着最好不要把P0用于其它用途，因为它很容易被偶然地改变。

**Q变量**

Q变量，象P变量一样，是通用的用户变量:在内存中有固定的位置，为48位的浮点变量，但没有预先定义用途。但是一个给定的Q变量的意义(以及它包含的值)和使用它的坐标系有关。这允许几个坐标系使用同一个程序(例如，都包括直线X(Q1+25) Y(Q2))，但它们自己的Q变量的值不同(在这里，意味着不同的目标点)。

分配Q变量

共有1024个Q变量。如果你只使用一个坐标系(坐标系1--
指定为&1)，你可使用所有Q变量:Q0到Q1023。坐标系2(&2)的Q变量是重叠的：&2的Q0是&1的Q512，&2的Q511是&1的Q1023。(Q变量的缓冲区实际上是环形的，所以&2的Q512就是&1的Q0，&2的Q1023是&1的Q511。)所以，两个坐标系都有512个独立的Q变量:Q0到Q511。

对于覆盖其它坐标系的Q变量PMAC没有提供保护。用户必须保证Q变量在正确的范围之内。

&3的Q0就是&1的Q256；&4的Q0是&2的Q256，&1的Q768；&5的Q0是&1的Q128；&6的Q0是&2的Q128，&1的Q640；&7的Q0是&3的Q128，&1的Q384；&8的Q0是&4的Q128，&1的Q896。可见下表。

PMAC Q变量内存分配

| 内存位置 | 坐标系1 | 坐标系2 | 坐标系3 | 坐标系4 | 坐标系5 | 坐标系6 | 坐标系7 | 坐标系8 |
|---|---|---|---|---|---|---|---|---|
| $1400 | 0 | 512 | 768 | 256 | 896 | 384 | 640 | 128 |
| …… | …… | …… | …… | …… | …… | ….. | …… | …… |
| $147F | 127 | 639 | 895 | 383 | 1023 | 511 | 767 | 255 |
| $1480 | 128 | 640 | 896 | 384 | 0 | 512 | 768 | 256 |
| …… | …… | …… | ……. | …… | …… | …… | …… | ……. |
| $14FF | 255 | 767 | 1023 | 511 | 127 | 639 | 895 | 383 |
| $1500 | 256 | 768 | 0 | 512 | 128 | 640 | 896 | 384 |
| …… | …… | …… | …… | …… | …… | …… | …… | …… |
| $157F | 383 | 895 | 127 | 639 | 255 | 767 | 1023 | 511 |
| $1580 | 384 | 896 | 128 | 640 | 256 | 768 | 0 | 512 |
| …… | …… | …… | …… | …… | …… | ……. | …… | …… |
| $15FF | 511 | 1023 | 255 | 767 | 383 | 895 | 127 | 639 |
| $1600 | 512 | 0 | 256 | 768 | 384 | 896 | 128 | 640 |
| …… | … … | … … | … … | … … | … … | … … | … … | … … |
| $167F | 639 | 127 | 383 | 895 | 511 | 1023 | 255 | 767 |
| $1680 | 640 | 128 | 384 | 896 | 512 | 0 | 256 | 768 |
| … … | … … | … … | … … | … … | … … | … … | … … | … … |
| $16FF | 767 | 255 | 511 | 1023 | 639 | 127 | 383 | 895 |
| $1700 | 768 | 256 | 512 | 0 | 640 | 128 | 384 | 896 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ... ... | ... ... | ... ... | ... ... | ... ... | ... ... | ... ... | ... ... | ... ... |
| $177F | 895 | 383 | 639 | 127 | 767 | 255 | 511 | 1023 |
| $1780 | 896 | 384 | 640 | 128 | 768 | 256 | 512 | 0 |
| ... ... | ... ... | ... ... | ... ... | ... ... | ... ... | ... ... | ... ... | ... ... |
| $17FF | 1023 | 511 | 767 | 255 | 895 | 383 | 639 | 127 |

黑体数字表示该坐标系可以使用的，不与其它坐标系使用的Q变量发生重叠的，Q变量编号的最大值。

Q变量的选址

你如何知道你你正在使用哪一组Q变量呢?这由命令的类型决定。当你通过一个在线(立即执行)命令存取Q变量时，你使用的是当前由主机选定的坐标系的Q变量(用&n命令)。

当你通过一个运动程序来存取一个Q变量时，该变量将属于运行该程序的坐标系。如果不同的坐标系运行相同的运动程序，它将使用不同的Q变量。

当你通过一个PLC程序存取Q变量时，你将使用由PLC程序中的ADDRESS命令指定的坐标系的Q变量。每个PLC程序都可以选定一个特定的坐标系，它与其它的PLC程序或主机的选定是独立的。如果PLC程序中没有使用ADDRESS命令，程序将使用坐标系1的Q变量。

数组

*数组读入*

可以把一些Q变量当作数组使用。这样做使得从变量读取值变得很容易:只需用圆括号内的表达式代替指定变量的常数--用Q({表达式})代替Q{常数}。PMAC将把它作为一个功能调用，象SIN({表达式})一样。

举例

如果你想顺序地把Q51到Q100作为位置输入，你可使用下面的这段语句。

```
F10
P1=51              ; 数组检索变量
WHILE (P1<101)     ; 开始循环
    X(Q(P1))       ; 随着P1改变，目标位置也会改变
    DWELL100
    P1=P1+1        ; 给检索变量增量
ENDWHILE
```

*数组写入*

向作为一个数组的一组变量赋值要复杂一些;它要通过间接的选址技术来完成。首先，定义一个指向坐标系1的Q0的M变量(例如，M60->L:$1400)。接着，定义第二个指向在Y:$BC3C中的第一个M变量定义字的低12位的M变量(例如，M70->Y:$BC3C，0，12，定义M70为指向M60定义字的低12位的M变量)。如果我们想指向M0的定义字，应使用Y:$BC00;指向M1，使用Y:$BC01;指向M50，使用Y:$BC32(十六进制的32即为十进制的50);指向M1023，使用Y:$BFFF。

现在，通过给第二个M变量赋值，你可改变第一个M变量指向哪一个P变量。在我们的例子里，命令M70=1024+17使M0指向坐标系1的Q17。

*Q指针偏置*

注意坐标系1的偏置1024;每个坐标系都需要有它自己的偏置来使M变量定义的数值与那个坐标系的Q变量编号相匹配。这些偏置是:

```
C.S.1:  1024     C.S.5   1152
C.S.2:  1536     C.S.6   1664
C.S.3:  1280     C.S.7   1408
C.S.4   1792     C.S.8   1920
```

一旦第一个M变量已经指向了一个特定的Q变量，给该M变量赋值将会把该值写入它指向的那个Q变量。继续我们的例子，命令M60=3.14将把3.14写入坐标系1的Q17。

*举例*

如果我们想用坐标系2的Q0到Q99建立一个从0.0到9.9的平方根表，你可使用下面的这段语句。(这里假定M60和M70已由上面的语句设置):

```
Q100=0                    ; 数组检索变量起始值
WHILE (Q100<100)          ; 开始循环
    M70=1536+Q100         ; 将M60指向适当的Q变量
    M60=SQRT(Q100/10)     ; 给该P变量赋正弦值
    Q100=Q100+1           ; 给检索变量增量
ENDWHILE
```

Q变量的特殊用途

有些Q变量有你需要注意的特殊用途。ATAN2功能(两个参数的反正切函数)自动使用Q0作为它的第二个参数(“cosine”参数)。READ命令把它读入的值在前头加上字母A到Z分别放在Q101到Q126内，同时一个表征码指出哪一个变量被读入了Q100。运动程序中的S表达式（主轴）将跟在它后面的值放在Q127内。

**M变量**

为了使用户存取PMAC的内存和I/O空间更容易，提供M变量。一般只需要用一个在线命令做一次定义。对于有后备电池的PMAC，定义自动被保存下来。对于有后备闪存的PMAC，必须使用SAVE命令以保证在电源故障或重启后仍能保存定义。用户通过给M变量分配一块内存来定义它，并且要指定这块内存的大小和存放值的格式。一个M变量可以是1位，1个半字节(4位)，1个字节(8位)，1又1/2个字节(12位)，1个双字节(16位)，2又1/2个字节(20位)，1个24位的字，1个48位的定点双字，1个48位的浮点双字，或双口RAM及多路拨码开关的特殊格式。

共有1024个M变量(M0到M1023)，象其它类型的变量一样，当读取时，M变量的编号必须由常数或表达式指定：M576或M(P1+20)；在写入时，编号必须由常数指定。

M变量定义

M变量的定义通过由减号和大于号组成的“定义箭头”(->)来完成。一般的，只需要用在线命令进行一次定义，因为它将被存在后备电池RAM或闪存内。被定义的M变量可以重复使用。

M变量通过定义中地址前缀的指定，可以是以下类型中的一种:

| | |
|---|---|
| X: | X内存中1-24位的定点值 |
| Y: | Y内存中1-24位的定点值 |
| D: | 同时占用X和Y内存的48位定点值 |
| L: | 同时占用X和Y内存的48位浮点值 |
| DP: | 32位定点值(X，Y内存内为低16位)(双口RAM使用) |
| F: | 32位定点值(X，Y内存内为低16位)(双口RAM使用) |
| TWD: | 多路拨码开关BCD码 |
| TWB: | 多路拨码开关二进制码 |
| TWS: | 多路拨码开关串行I/O码 |
| *: | 没有地址定义；使用定义字的部分作为一般用途变量 |

如果指定了M变量的X，Y类型，你必须同时指定使用的开始位，位数和格式(代码类型)。

典型的M变量定义语句如下:

```
M1->Y:$FFC2，8，1
M102->Y:49155，8，16，s
M103->X:$C003，0，24，s
M161->D:$002B
M191->L:$0822
M50->DP:$D201
M51->F:$D7FF
M100->TWD:4，0.8.3，U
```

手册的在线命令部分有每种类型M变量定义的指导。许多推荐的M变量定义在例程序SETUP。PMC内。

把所有的M变量定义做成单个的文件是个好主意，并在文件头上加上命令M0…1023->*。这将删除所有已有的M变量定义，有助于防止“混乱”的M变量定义造成的奇怪的问题。

M变量定义被作为24位代码存在PMAC的Y:$BC00(M0)到Y:$BCFF(M1023)内。除了多路拨码开关M变量，该代码的低16位包含了M变量指向的寄存器(高8位指出使用哪一部分地址以及如何解释它)。如果另一个M变量指向定义的这一部分，它可用来改变目标寄存器。这一技术主要用来生成P和Q变量的数组，如同上面解释的那样，在这两个变量的描述部分有例子说明。它也可用来在双端口RAM或用户缓冲区内生成数组(可见在线命令，定义缓冲区部分)。

取值范围

M变量的取值范围要比PMAC的整个运算范围小。当试图把一个超出M变量取值范围的值赋给那个M变量，PMAC将自动“浮动”该值到范围以内，而并不报错。

例如，对一个单个位的M变量，所有赋给该变量的奇数将被当作”1”，所有偶数被当作“0”。当试图把一个非整形量赋给一个整形的M变量时，PMAC将自动取舍为最接近的整数。

使用M变量

一旦定义，就可以在程序里象使用任何其它变量和表达式那样使用M变量。在计算表达式时，PMAC读指定的内存区域，通过定义的大小和格式计算出值，并在表达式中使用该值。

在表达式中使用M变量要小心。如果某个M变量可被一个在后台运行的优先级较高的伺服例程所改变(例如瞬时指令位置)，那么在计算表达式时就不能保证计算过程中该值不会发生改变。例如，如果在表达式(M16-M17)*(M16+M17)中M变量是瞬时伺服变量，用户就无法肯定表达式中不同位置的M16或M17有相同的值，或M16和M17的值来自同一个伺服周期。第一个问题可以通过设置P1=M16和P2=M17来解决，但第二个问题却没有一个普遍的解决办法。

## 运算符

PMAC的运算符和其它计算机语言的作用是一样的：它们组合数值来产生新值。

**数学运算符**

PMAC使用四种标准的数学运算符:+，-，*和/。使用标准的数学优先级准则:乘和除在加和减之前执行，相同优先级的运算由左向右执行，而括号内的运算优先执行。

取模运算

PMAC也有取模运算符“%”，在该运算符之前的值被它后面的值除时，它产生余数。运算对象必须是整形或浮点值。该运算符对于处理浮动的计数器和计时器特别有用。

当取模运算的操作对象为一个正值X时，所得结果的范围为0到X(不包括X本身)。当取模运算的操作对象为一个负值-X时，所得结果的范围为-X到X(不包括X本身)。当一个寄存器能在另一个方向浮动时，这种负的取模运算是非常有用的。

**逻辑运算符**

PMAC有三种对”位”进行操作的逻辑运算符:&(按位与)，|(按位或)和^(按位异或)。如果对浮点值进行操作，运算符对整数和分数位都起作用。&的优先级与*和/一样;|的优先级与+和-一样。可使用括号来超越这些缺省优先级的限制。

注意，这些按位操作的逻辑运算符与复合条件(q..)中使用的布尔运算符AND和OR不同

## 函数

函数可对常量或表达式进行操作来产生新值。一般格式为:

{函数名} ({表达式})

可用的函数为SIN，COS，TAN，ASIN，ACOS，ATAN，ATAN2，SQRT，LN，EXP，ABS和INT。

注意，三角函数使用的单位是度还是弧度，由全局I变量I15控制。

| 函数 | 标准三角正弦函数 |
|---|---|
| 语法 | SIN({表达式}) |
| 定义域 | 所有实数 |
| 定义单位 | 度/弧度 |
| 值域 | -1.0--1.0 |
| 函数值单位 | 无 |
| 出错 | 无 |

| | |
|---|---|
| 函数 | 标准三角余弦函数 |
| 语法 | COS({表达式}) |
| 定义域 | 所有实数 |
| 定义单位 | 度/弧度 |
| 值域 | -1.0--1.0 |
| 函数值单位 | 无 |
| 出错 | 无 |
| 函数 | 标准三角正切函数 |
| 语法 | TAN({表达式}) |
| 定义域 | 除+/-Pi/2，3Pi/2，5Pi/2以外的所有实数 |
| 自变量单位 | 度/弧度 |
| 值域 | 所有实数 |
| 函数值单位 | 无 |
| 出错 | 在非法定义域内被0除(返回最大的实数值) |
| 函数 | 反正弦函数(arc-sin) |
| 语法 | ASIN({表达式}) |
| 定义域 | -1.0--1.0 |
| 自变量单位 | 无 |
| 值域 | -Pi/2--Pi/2弧度(-90--90度) |
| 函数值单位 | 度/弧度 |
| 出错 | 非法自变量域 |
| 函数 | 反余弦函数(arc-cos) |
| 语法 | ACOS({表达式}) |
| 定义域 | -1.0--1.0 |
| 自变量单位 | 无 |
| 值域 | 0--Pi弧度(0--180度) |
| 函数值单位 | 度/弧度 |
| 出错 | 非法定义域 |
| 函数 | 标准反正弦函数 |
| 语法 | ATAN({表达式}) |
| 定义域 | 所有实数 |
| 定义单位 | 无 |
| 值域 | -Pi/2--Pi/2弧度(-90--90度) |
| 函数值单位 | 度/弧度 |
| 出错 | 无 |
| 函数 | 扩展的反正弦函数。它以表达式的值为正弦值，以坐标系的Q0值为余弦值，计算得角度值。如果是在PLC程序中进行计算，应确保在该PLC程序中已用ADDRESS命令指定了正确的坐标系。(事实上，它仅仅是这两个值的绝对值的比，它们的符号，与函数的取值有关)。由于使用了两个参数。它与标准的ATAN函数不同。该函数的优点在于它的取值范围为360度，比单参数的ATAN函数的180度要好。 |
| 语法 | ATAN2({表达式}) |
| 定义域 | 所有实数 |
| 自变量单位 | 无 |
| 值域 | -Pi--Pi弧度(-180--180度) |
| 函数值单位 | 度/弧度 |
| 出错 | 无 |
| 函数 | 自然对数函数(底数为e) |
| 语法 | LN({表达式}) |
| 定义域 | 所有正实数 |
| 自变量单位 | 无 |
| 值域 | 所有实数 |
| 函数值单位 | 无 |
| 出错 | 非法定义域 |
| 函数 | 指数函数($e^x$). |

| | |
|---|---|
| | 注意，可用$e^{x\ln(y)}$来实现$y^x$函数。PMAC表达式EXP(P2*LN(P1))将实现函数$P1^{P2}$ |
| 语法 | LN({表达式}) |
| 定义域 | 所有实数 |
| 自变量单位 | 无 |
| 值域 | 所有正实数 |
| 函数值单位 | 无 |
| 出错 | 无 |
| 函数 | 取平方根 |
| 语法 | SQRT({表达式}) |
| 定义域 | 所有非负实数 |
| 自变量单位 | 任意 |
| 值域 | 所有非负实数 |
| 函数值单位 | 任意 |
| 出错 | 非法自变量域 |
| 函数 | 绝对值函数 |
| 语法 | ABS({表达式}) |
| 定义域 | 所有实数 |
| 自变量单位 | 任意 |
| 值域 | 所有非负实数 |
| 函数值单位 | 任意 |
| 出错 | 无 |
| 函数 | 截断函数，将返回小于或等于参数的最大的整数(INT(2。5)=2，INT(-2。5)=-3)。 |
| 语法 | INT({表达式}) |
| 定义域 | 所有实数 |
| 自变量单位 | 任意 |
| 值域 | 整数 |
| 函数值单位 | 任意 |
| 出错 | 无 |

## 表达式

PMAC的表达式是一个由运算符连接的常量，变量以及函数组成的一个数学结构。表达式可用来给变量赋值，可决定运动程序的参数或作为条件的一部分。一个常量也可作为表达式，所以在一个表达式语法里，一个常量可能与一个更复杂的表达式一起使用--
与定义{data}时不一样，非常量的表达式不需要额外的括号。表达式的例子如下:

```
512
P1
P1-Q18
1000*cos(Q25*3.14159/180)
I100*ABS(M347)/ATAN(P(Q3+1)/6.28)+5
```

## 数据

如果PMAC的命令需要{data}，用户可以或者使用一个不被括号包围的常量，或者使用一个被括号包围的表达式。(由于一个常量也可被当作表达式，把常量放在括号内也是允许的，但这样花费了更多的存储空间和计算时间。)

例如，如果列表命令语法为T{data}，那么T100的合法使用为:

```
T(P1+250*P2)          T(100)   (合法但不必要)
```

## 变量值赋值语句

这类语句计算并给一个变量赋一个值。当一个赋值语句送给PMAC时，如果一个程序缓冲区是空的，该语句将加入这个缓冲区内。如果不是，它将立即执行。标准的赋值语法为:

{变量名}={表达式}

{变量名}指定要使用哪一个变量，{表达式}表示要赋给该变量的值。

**给I变量赋缺省值**

语句I{data}=*将指定变量的厂家缺省值赋给该变量(而不是用户的EEPROM中的存储值)。

**同步的M变量赋值**

在一个运动程序内，当PMAC在进行混合或样条运动时，计算必须在实际运动点之前进行。为了能够进行完全的加速，也为了做成合理的速度和加速度限制，这是必要的。加速时的计算可能发生在实际运动的一，二或三步之前，这由运动的模式决定。

为什么需要这样做?

如果赋值是计算的一部分，在考虑实际运动的执行时，变量会在程序中它们所处的位置之前得到它们的新值。对于P和Q变量，这一般不成问题，因为它们仅在进一步的运动计算时才存在。但对于M变量，特别是输出时，这就成为一个问题了，因为对于一个普通的变量值赋值语句，它要比预期更早地发生。

例如，程序段

```
X10          ;把X轴移到10
M1=1         ;打开输出1
X20          ;把X轴移到20
```

你可能希望在X轴到达位置10时才打开输出1。但由于PMAC的计算在前，在开始向位置10移动时，它已经完成了程序下一步的计算，包括打开输出的语句M1=1。所以，使用这种技术，打开输出将比我们期望的要快。

它们怎样工作?

同步的M变量赋值语句是解决这个问题的方法。当在程序中遇上这一类语句时，它并不是被马上执行;而是被放进堆栈里，在程序下一步的实际计算开始时才执行。这就使打开输出的操作与运动的操作同步了。

修改过的程序段

```
X10          ;把X轴移到10
M1==1        ;同步打开输出1
X20          ;把X轴移到20
```

语句M1==1(双等号指出为同步赋值)在移动到X10的一开始就被处理，但它要到程序的下一步(X20)开始时才被执行。

注意:对于同步的赋值，实际的赋值操作将在对新的运动进行加速时执行，它通常在编程点之前。在LINEAR和CIRCLE模式运动中，加速发生在距离指定中间点V*TA/2远的地方，V为指令速度，TA为加速时间。

同时应注意赋值是与指令位置同步的，而不是实际位置。使指令位置和实际位置接近是伺服环要完成的。

在PMAC执行分段运动的应用中(I13>0)，同步的M变量赋值是在程序运动加速开始之后的第一个I13样条段开始时执行的。

注意:当程序终止或暂时停止时，程序中的最后一个运动或DWELL之后的同步M变量赋值并不执行。用DWELL作为程序的最后一条语句来执行这些语句。

语法

M变量同步赋值语句有四种形式:

```
M{常数}=={表达式}        ; 直接等号赋值
M{常数}&={表达式}        ; AND-等号赋值
M{常数}|={表达式}        ; OR-等号赋值
M{常数}^={表达式}        ; XOR-等号赋值
```

在所有这些形式中，当运动执行之前，程序遇上这一行时，将计算语句右边表达式的值。该值，变量编号和运算符将放在堆栈内，在适当的时间加以执行。

执行

当相关的运动开始实际执行时，这些项被从堆栈中取出并执行。对于==的语法，表达式的值被简单地赋给变量。对于其它形式(&=，|=和^=)，变量的值被读出，与表达式的值进行按位的布尔操作(分别进行AND，OR和XOR)，并把结果写回变量。

特殊的布尔特性

这些布尔赋值运算符与==有细微的差别。假设它们对一个8位的M变量进行操作，把所有的奇数位设为1，保留所有的偶数位:

```
M50==M50&$AA
M50&=$AA
```

这两个语句的差别在于运算时对M50的读取。在第一种情况下，当语句第一次被处理时读取M变量。在第二种情况下，当运算被从堆栈中取出，在回写变量之前读取M变量。对于这种情况，M变量的值不可能被有些其它的任务所改变。

限制

对于这些函数，有几个用户必须注意的要求

*合理的形式*

首先，这些语句不可以使用所有的多路拨码开关M变量形式(TWB，TWD，TWR或TWS)。布尔赋值符不能使用任何两倍宽度的M变量形式(D，L或F)。

*堆栈限制*

其次，保存这些操作的堆栈空间被限制在32个字以内(每个坐标系)。每个赋值占用2-3个字(见下)，并有一个额外的字来标志一步操作的结束。当PMAC在前头执行了“n”步时，每一步分配32/(n+1)个字总是安全的。经验法则给了PMAC足够的堆栈空间存放它的栈操作并给大多数用户以足够的同步M变量。当PMAC在前头执行了1步时，经验法则给了每一步16个字的堆栈空间(至少有5个加上一个结束字的赋值)；当PMAC在前头执行了2步时，经验法则给了每一步10个字的堆栈空间(至少有3个加上一个结束字的赋值)；只要在这些限制内，你的操作总是安全的。

如果你知道只有单个的运动，你就可以加入更多的赋值，例如，下一步运动没有同步的赋值，或某些赋值在堆栈中占用少于3个字的空间。单个运动的赋值不可以占用超过32个字的内存。

下表显示了每种类型的赋值需占用的堆栈空间:

| 函数 | 表达式的值 | 1-到20-位的M变量 | *或24-位的M变量 | 32-或48-位 的M变量 |
|---|---|---|---|---|
| == | 都为1 | 2 | 2 | 3 |
| == | 都为0 | 2 | 2 | 3 |
| == | 其它 | 3 | 2 | 3 |
| &= | 都为1 | 0(无操作) | 0(无操作) | 非法 |
| &= | 都为0 | 2 | 2 | 非法 |
| &= | 其它 | 2 | 2 | 非法 |
| \|= | 都为1 | 2 | 2 | 非法 |
| \|= | 都为0 | 0(无操作) | 0(无操作) | 非法 |
| \|= | 其它 | 2 | 2 | 非法 |
| ^= | 都为1 | 2 | 2 | 非法 |
| ^= | 都为0 | 0(无操作) | 0(无操作) | 非法 |
| ^= | 其它 | 2 | 2 | 非法 |

## 比较符

比较符计算两值(常量或表达式)之间的关系。它被用来决定运动或PLC程序中条件的真假。PMAC的合法比较符为:

```
=     (相等)
!=    (不等)
>     (大于)
!>    (不大于；小于或等于)
<     (小于)
!<    (不小于；大于或等于)
```

~ (约等于——差在1以内)
!~ (非约等于——差至少为1)

注意:<=和>=不是PMAC的合法比较符。它们分别被!>和!<所取代。

## 条件

条件用来控制运动或PLC程序的程序流程。它的值可以是真或假。它能被用在IF分支语句或WHILE循环语句中。PMAC支持简单和组合条件。

命令行——IF或WHILE——中的条件必须用括号括上。

**简单条件**

一个简单条件由三个部分组成:

{表达式} {比较符} {表达式}

如果比较符两边的表达式的关系是合理的，条件为真;否则，条件为假。命令中的简单条件举例如下:

```
WHILE (1<2)            (总为真)
IF (P1>5000)
WHILE (SIN(P2-P1)!>P300/1000)
```

注意，条件需要用括号括上。

不象有些高级语言，PMAC的条件不能仅仅是由0或非0来区别的一个值(例如，IF (P1)是非法的)。它必须由两个表达式和一个比较符组成。

**组合条件**

组合条件是由逻辑运算符AND和OR连接起来的一组简单条件。组合条件的值通过对简单条件的值进行布尔运算来决定。在PMAC中，AND的优先级比OR要高(就是说，OR对由AND连接起来的简单条件进行运算)。在发现一个假的简单条件后PMAC将停止对复合的AND条件进行计算。命令行中的复合条件举例如下:

```
IF (P1>-20 AND P1<20)
WHILE (P80=0 OR I120>300 AND I120<400)
IF (Q16!<Q17 AND Q16!>Q18 OR M13<256 AND M137<256)
```

在单个命令行内复合条件包含的简单条件不可以被括号分开。例如，IF((P1>-20) AND (P1<20))是一个非法条件，并且将被当作非法语法而拒绝。

单行条件语句

在PMAC的运动程序中(但不包括PLC程序)，条件为真时执行的动作可以和条件放在同一行里。在这时，不需要使用ENDIF或ENDWHILE来表示条件句的结束，也不允许使用；该行的结束将自动作为条件句的结束。例如：

```
IF (P1<0) P1=0
WHILE (M11=0) DWELL 10
```

在PMAC的环形程序缓冲区内只允许单行的条件语句。不允许多行的条件语句，因为这时不能保证将要跳转的语句在该环形缓冲区内。

多行条件语句

在PMAC的运动程序中(但不包括PLC程序)，允许多行的复合条件语句。条件语句的第一行必须以IF或WHILE开头；后面的各行必须以AND或OR开头。某一程序行内的简单或复合条件总是在各行的语句组合之前被处理。多行条件之间，AND的优先级要高于OR。在发现某一行条件为假后，PMAC将停止处理多行的AND条件。例如:

```
IF (M11=1 OR M12=1)
AND(M13=1 OR M14=1)
```

# 计时器

PMAC有四个计时寄存器--它们分别在地址X:$0700，Y:$0700，X:$0701和Y:$0701。这四个24位寄存器是用户程序使用的一般用途计时器。每个伺服周期PMAC给它们减量。用户可以在需要的时候为它们赋值。读写操作通常通过M变量完成。通常用户写入的值等于有伺服周期标度的需要等待的时间。然后程序等待寄存器小于0。寄存器将继续计数直到它们到达$-2^{23}$ (-8，388，608)。它们不会翻转回正值。

由于这些计时器的单位为伺服周期，而大多数用户希望以毫秒为单位工作，必须做一个转换。要把毫秒转换成伺服周期，乘以$2^{23}$ (8，388，608)并除以I10的值。

要得到这些寄存器的过零点，必须把它们当作有符号的变量(补码)。任何M变量的定义应有“S”(符号格式)。例如:M90->Y:$0700，0，24，S。

**举例**

在一个PLC程序中，在一段以毫秒为单位的时间内打开一个输出：

```
M1=1                    ; 打开输出1
M90=125*8388608/I10     ; 设置计时器(M90)为125msec
WHILE (M90>0)           ; 等待计时器计时到0
ENDWHILE
M1=0                    ; 关闭输出1
```

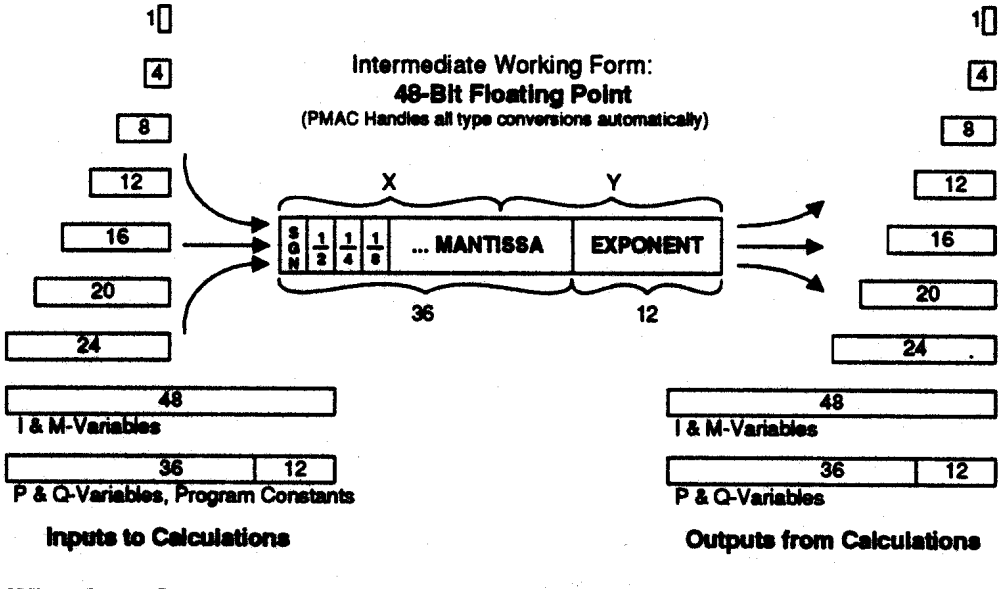
**计算特性的其它考虑**

当PMAC在PLC程序，运动程序或在线命令中做计算时，它使用它的48位浮点格式作为计算的中间形式。这给PMAC提供了在不同数值格式间自动转换的能力，并使它能对P和Q变量按位操作，尽管它们都是浮点值。

把一个数字转换成48位格式的处理是非常快的，在大多数PMAC应用中根本无法察觉它。
对于PMAC对计算有要求的应用来说，跳过转换这一步能提高处理的速度和效率。对于这一类应用，使用跳过转换的P，Q和L格式M变量(它们已经是48位格式)要比使用其它类型的变量计算的快。

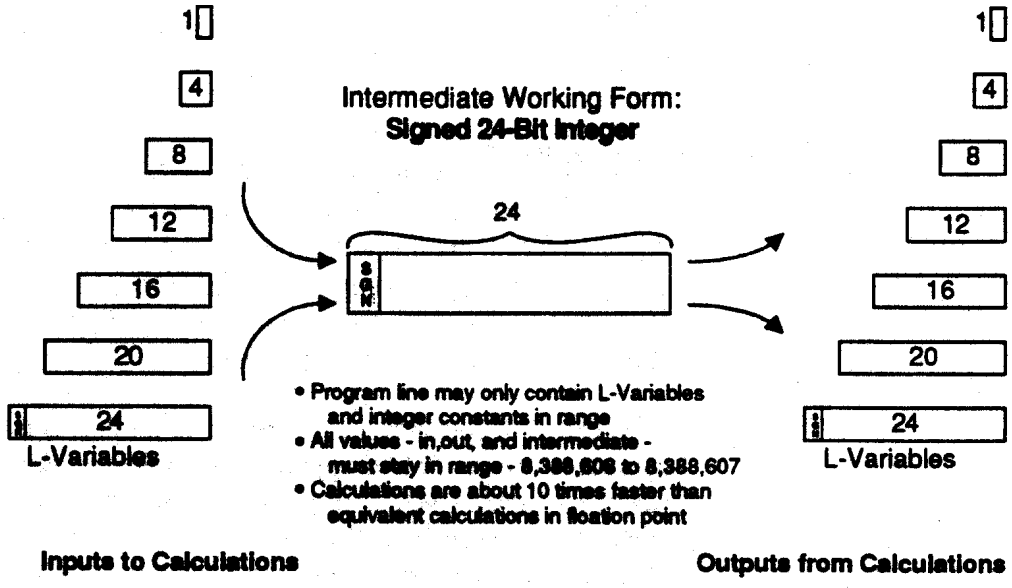
**PMAC程序数学计算**

包括所有的运动和PLC程序　除了定点的编译PLC程序




当PMAC在一个编译的PLC程序中(PLCC)用L变量作计算时，它使用24位定点格式作为计算的中间形式。这使得PMAC可以以非常快的速度进行计算。L变量计算要比相同的 浮点计算快10倍左右。

**PMAC程序数学计算**




# §3—11 编写一个运动程序：

**简介：**

PMAC能够支持多达256个运动程序。任意坐标系在任何时候都可以执行这些程序中的任意一个，即使另外的坐标系正在执行同样的程序。PMAC能够同时执行和该卡上定义的坐标系的数目（最多到8）一样多的运动程序。一个运动程序能够将任何一个别的运动程序调用作子程序，可以带变元，也可以不带变量。

作为在高级计算机语言如Basic或者Pascal和“G-代码”(RS-274)和机器加工语言的交互使用PMAC运动语言也许是一种最好的描述。实际上，PMAC能够直接接受“G代码”程序（要提供这项功能必须正确得设置）。它有一个计算机语言的计算和逻辑结构，而运行复本结构时又非常象机器工具语言。程序中的数值能够被指定为常数或表达式。

**流程控制：**

在一个运动程序中，PMAC有WHILE循环和IF..ELSE分支来控制程序流程。这些结构可以被嵌套而不需定义。此外，还有GOTO语句，可带有常数或变量（可用GOTO能执行和CASE语句一样的功能）。GOSUB语句（恒定或可变的标号）允许在程序中执行子程序。CALL语句允许别的程序被用作子程序。子程序的入口不必在程序的开始处——例如语句CALL 20.15000将使程序20的入口在行N15000处。GOSUBs和CALLs只能被嵌套15层。

**G代码：**

为了处理机床语言G代码，PMAC将Gnn作为CALL1000.nn000来对待。这一行随后的值（例如，X1000）将作为参数对待，就象对一个回定循环，或者一个没有变量的可执行的子程序那样，然后返回，执行该行的余下部分（就象对一个模态G代码一样）。机床语言设计者编写了程序1000以便象他希望的那样实施G代码，并允许定制和增强。***Delta Tau***提供了实施所有标准G代码的样本文件。M，S，T，和D代码的编写与G代码相似。

**模态命令：**

PMAC运动程序中的许多指令在本质上是模态的。这些指令中就包括运动模式，它指定了一个运动命令将产生怎样类型的轨迹；这类中包括LINEAR，RAPID，CIRCLE，PVT，和SPLINE。运动可以用INC和ABS命令来指定为增量式的（距离）或绝对式的（位置），可以每根轴地单独选择。运动时间（TA，TS和TM）和（或）速度（F），也可在模态命令中得到实施。模态命令将领先于它们将影响到的运动命令，或者它们将作为这些运动命令中的第一位而处在同一行上。

**运动命令：**

运动命令本身是由一个单字母轴定义符和后面跟随的一位或两位数值（常数或表达式）组成的。当执行一行时，在同一行指定的所有轴将协调一致地同时运动；串连的行将顺序地执行（在之间有还是没有停顿，将由其模式来决定）。依靠有效的模式，指定值可能意味着目的，距离，和（或）速度（参见轨迹特征部分）。

## 运动程序轨迹：

PMAC的轨迹生成是它众多的优异性能中算法最具能力和灵活性的一个。这些算法允许大量不同变量的操作得到执行，并且使用户在易用和控制等级两方面得到一致。要记住这些轨迹只是一系
列的命令位置。如果试图使实际位置和命令位置相匹配，那么每根轴都要使用伺服环。在这一部分中所讨论的时间，速度，距离和图形，如无另外注释，都是我们所推荐的。

## 线性混合运动：

想要完成的最容易的一类运动就是线性混合运动类。在这类运动中，一个轴以指定的速度移向指定位置，在控制方式下，从该速度上加速或减速。如果有不止一个运动按顺序被指定，且之间没有停顿，第一个运动将以同一类型的加速减速控制方式混合到第二个运动当中去。

线性混合运动模式是给运动程序的默认模式。如果是在别的运动模式下，程序可以通过用LINEAR而被放到线性混合模式中来。当然，程序也可以通过用别的运动模式（例如，CIRCLE1，CIRCLE2，RAPID，PVT，SPLINE）跳出线性混合模式。在每一个程序中都是LINEAR模式，而不依靠默认，这是一个好的编程习惯。LINEAR指令等效于RS－274G代码的G01。

**加速度参数：**

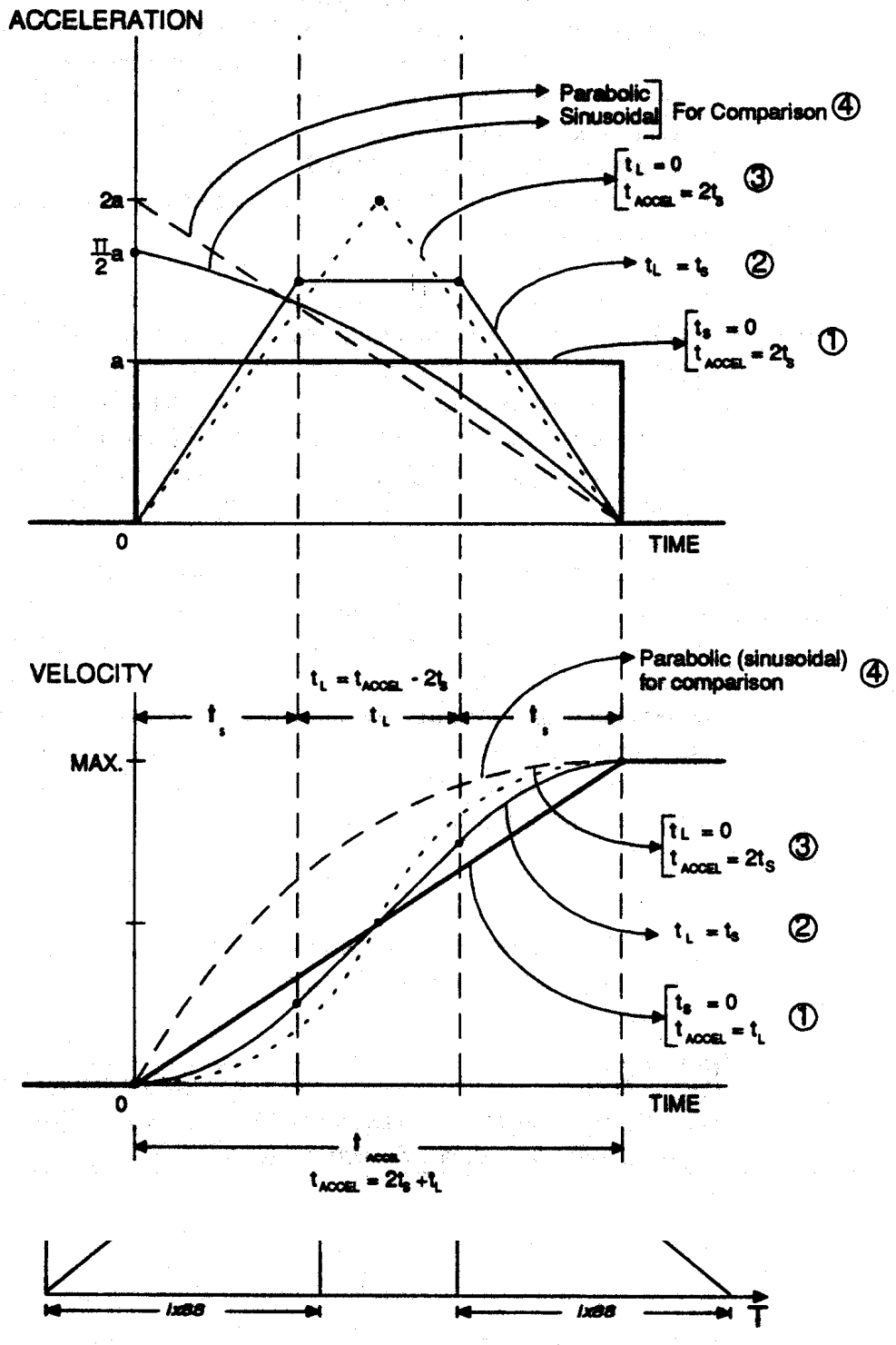
加速达到一个速度，或从该速度减速的加速度应该是常数，提供梯形的速度轨迹，它也可以线性地变化，得到S－曲线速度轨迹，或者它还可以是两者的组合。用户仅需要指定全加速的时间（TA，默认参数是坐标系I－变量Ix87），和每半个“S”曲线的时间（TS，默认参数是Ix89）。如果指定的TA时间比指定的TS时间的两倍要小，那么用到的TA时间将是TS时间的两倍（只想要纯粹的S－曲线加速的用户需要设置TA为0）。PMAC对于TA和TS只能用整数值。如果指定的不是整数值，PMAC在轨迹运算中用到该值之前将会把这个值取整为距该值最近的整数。

**加速度限制：**

注意：如果PMAC在运动分段模式下工作（I13>0），该模式对于圆弧插补是需要的，Ix17加速度限制将无效。

如果在一个运动中对于任何电机指定的加速度超过了可编程的最大加速度（由Ix17指定，单位是counts/msec$^2$），那么该运动中所有的电机都将按比例的减速，以便不超过该限制。通过空间的轨迹不会改变，给每一台电机的速度图形也不会改变。如果用户想要直接指定加速率，TA和TS应该被设置得非常小，以便不超过限制条件，这时加速度是通过电机I－变量Ix17来控制。

（PAGE 3－172坐标系变量图）




（PAGE 3－173自动“S”曲线加速度图）

注意：不要同时将TA和TS（Ix87和Ix88）时间都设置为零，即使是你想依靠加速度限制的作用也不要这么做。这将导致被零除的错误，并可能产生不稳定的运行。

**何时会更有效：**

当将多个线性运动混合到一起时，Ix17对于减速到停止后再混入下一个运动时的中间加速度也是强制起作用的。当PMAC计算混合序列中的每一个运动时，它都假定这是序列中的最后一个运动，而且它将确认减速的加速度在限制以内。这将平致减速的时间比编程指定的运动时间（既可以直接通过TM指定，也可以通过F反馈率的距离间接指定）要长，也将导致运动执行的速度比编程的速度要慢。当运动被分成很小的部分混合到一起

时，这就会显得非常有局限性了。为了不限制速度，Ix17

设置应该比用最高速度除最小程序段时间的值大。这将使得加速度控制更为有效。

**何时将无效*：***

为了执行加速度限制，PMAC将提前运算实际运动执行的前两个运动，并且重新核算这两个运动以便保证加速度在Ix17的限制之内。然而，在多于两个运动的情况下，为了保证加速度在限制范围内，要计算多于两个的运动。在这些情况下，PMAC将尽可能地限制加速度，但是因为较早的运动已经被执行过了，它们不能被废弃，因此加速度限制将可能被超过。

**速率或移动时间指定：**

用户既可以用一个F命令来给运动指定目标速度（速率），也可以用TM命令来指定运动时间。如果F被指定了，那么运动时间将被计算出来，同样如果TM被指定了，则速率将被计算出来。这两个值之间的关系是在给定距离的前提下的倒数关系。不论在哪一种指定情况下，用户必须记住在每一个运动或运动序列的结束都有一个附加的TA时间用来减速到停止。也就是说，一个指定了TM的单独的运动从开始到结束所花的时间将是TM＋TA。一个N段的运动将会用

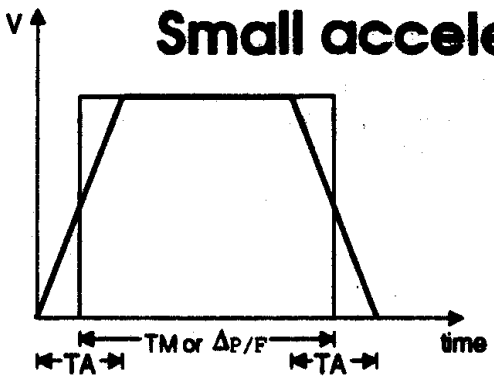
$$SUM_{(I=1ton)} = \{TM_I\} + TA_1/2 + TA_N/2$$

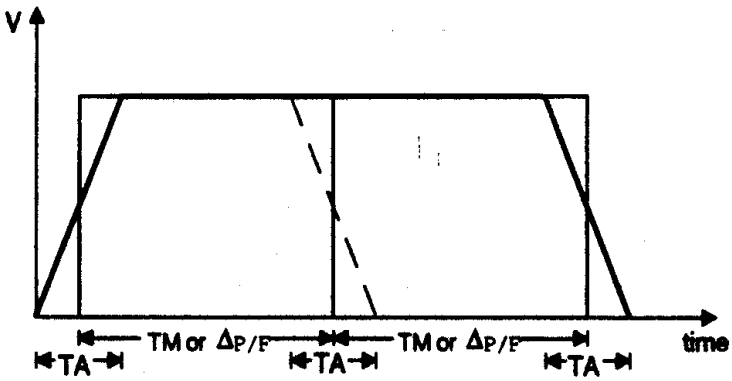
的时间来完成。对于指定速率的运动也是一样，除了每段的TM是用距离除速率来计算的，而不是直接指定的。

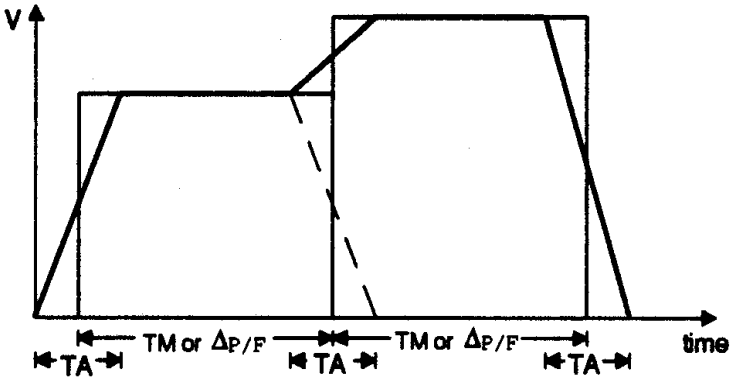
注意：速率是一个数值，因此应当总是一个正数。负的速率将导致和坐标系定义中正的定义完全相反的运动。

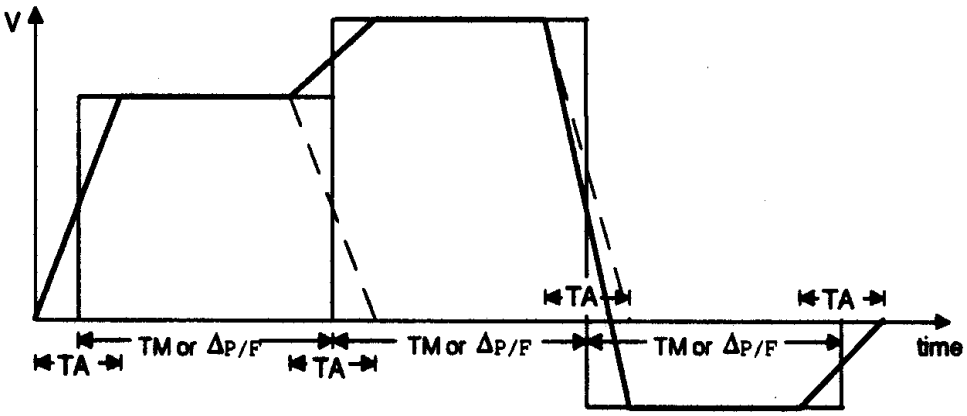
## （PAGE 3－175 线性模式轨迹图）

# Small acceleration time



**短程运动：**

如果一个速率指定运动部分在距离上是如此之短，以至于它不能够达到目标速度，它将把它的整个时间花在加速上（将产生一三角的，而不是一个梯形的图形）。给这样的一个运动的最短的时间将是指定的加速时间（TA或2TS）。对于一个单独的运动要记住加上为停止而附加的用于减速的加速度时间。

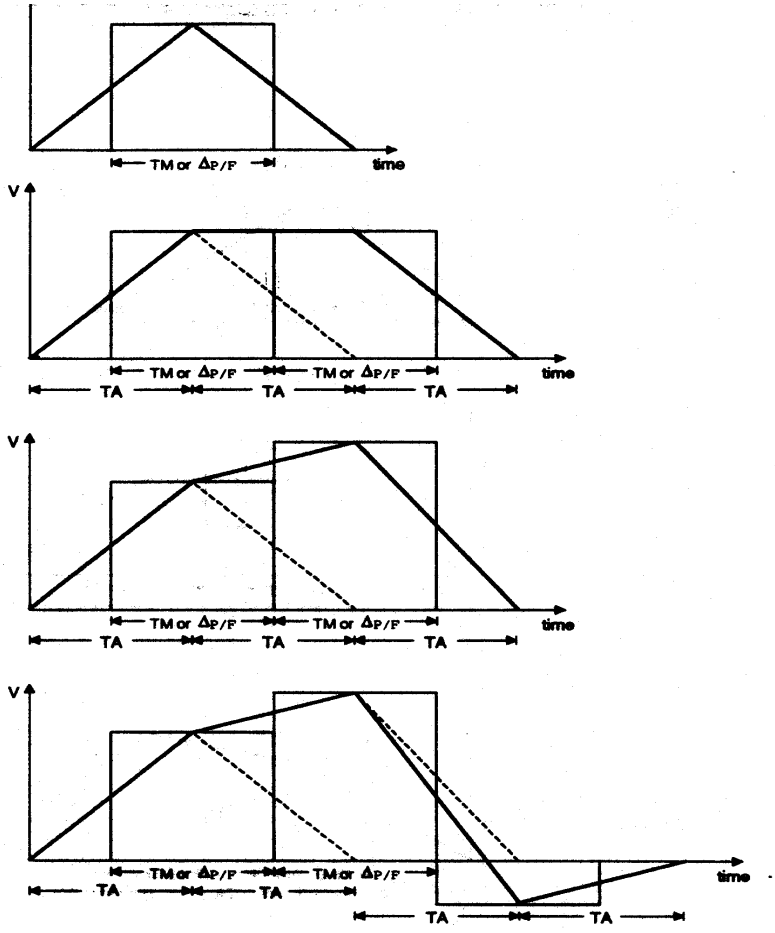
在一个指定时间的运动段中，如果TM小于加速时间，该部分将在加速时间中被执行，而不是TM时间里。

换言之，对于一个单独的混合运动或混合的运动部分，加速时间将是最短的时间。这是在为防止运动时间太短以至于PMAC不能计算出它们的实际时间的保护中的。如果你用太短的运动段工作，并且你的运动序列将比你所希望的执行要慢的多，加速时间限制将可能引起问题。

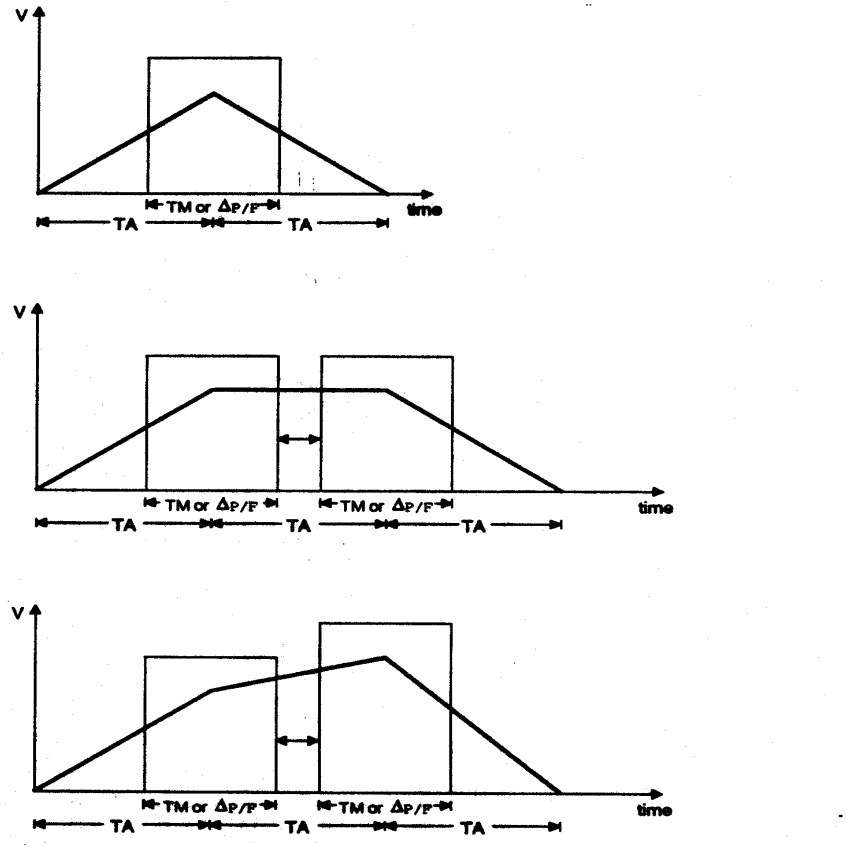
**远程运动：**

对于一个编程运动的最大的时间是$2^{23}-1$（8,388,607）毫秒，大约2小时20分钟。这是PMAC用TM命令能够接受的最大的值。这也是PMAC给一个速率指定的运动进行计算的最大的值。如果给该运动的矢量距离被速率除产生了一个比8,388,607毫秒大的时间，PMAC将使用8,388,607毫秒作为移动时间，并且速度将比编程指定的速度要高。

**（PAGE 3—177 匹配运动时间的线性模式轨迹加速时间图）**




**（PAGE 3—178 线性模式轨迹最大（速度限制）加速时间图）**




**进给率轴：**

如果一个多轴运动是通过速率指定的（不是通过时间），用户使用FRAX命令在线指令或编程指令，将在轴控制的矢量速率方面拥有更多的灵活性，速度被分配在这些轴中以便它们的矢量组合（平方和的平方根）是指定的速度，PMAC将用速率轴的矢量距离除编程的来计算出运动时间。这就使用户不必再因为每次运动的不同角度而不得不单独计算每根轴的速度。如果一个同步运动要求一个非进给轴，该运动将在给速率轴计算的一样的时间内完成。对于一个坐标系来说，默认的进给轴是X，Y，和Z轴。

注意：如果一个速率指定的运动只要求非无进给轴，PMAC将计算其矢量距离为零（因为没有一根进给轴移动），产生一个零移动时间。因此，运动时间将由加速时间和（或者）电机速度限制来控制。

**矢量速率计算示例**

INC　　Dist=SQRT($3^2+4^2$)=5
FRAX（X，Y）　　Move Time=5/10=0.5
X3 Y4 F10　　Vx=3/0.5=6
Vy=4/0.5=8

INC　　Dist=SQRT($3^2+4^2$)=5
FRAX（X，Y）　　Move Time=5/10=0.5
X3 Y4 Z12 F10　　Vx=3/0.5=6
Vy=4/0.5=8
Vz=12/0.5=24

INC　　Dist=SQRT($3^2+4^2+12^2$)=13
FRAX（X，Y，Z）　　Move Time=13/10=1.3
X3 Y4 Z12 F10　　Vx=3/1.3=2.31

Vy=4/1.3=3.08
Vz=12/1.3=9.23

INC Dist=0
FRAX（X，Y，Z） Move Time=0/10=0(<TA)
C10 F10 加速度限制运动

**速度限制:**

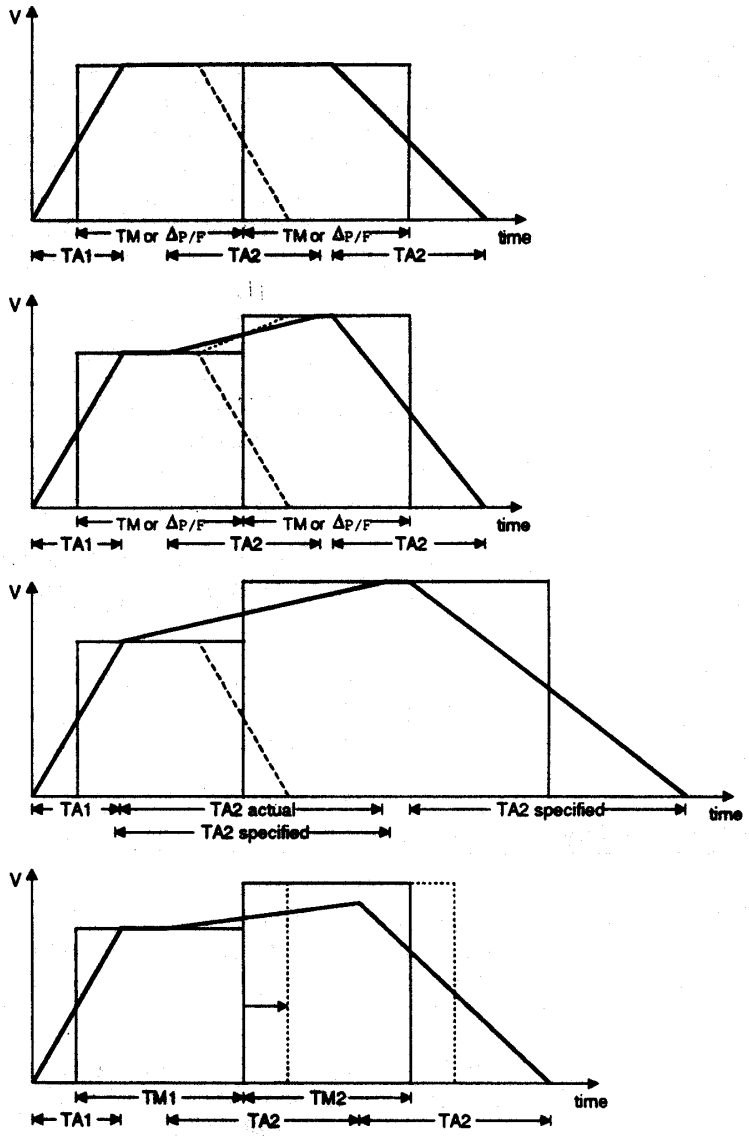
每一台的电机的速度都受到电机速度限制（Ix16）的制约。如果有一台电机超过了该限制，该运动中所包括的所有电机都将按比例变慢，以便没有电机超过该限制，但是空间的轨迹仍然不变。

注意：如果PMAC是工作在圆弧插补所要求的分段运动模式下（I13>0），将不能观察到Ix16速度限制。

**混合功能:**

如果有不止一个运动连续地被指定，其中没有插入任何延迟的命令，那么每一台电机将根据此时的加速度和有效的S一曲线值，从第一个运动的速度到第二个运动的速度平滑地混合运行。这个在速度上的变化（也可能是一个零变化）开始于第一个运动为了停在指定位置而开始减速的那个点，不是在第一个运动本身结束时的端点。（然而，如果Ix92被设置为1，在那个坐标系中的“混合”运动总在下一个运动开始之前就结束了。）

（**PAGE 3一181线性模式轨迹** 改变加速时间图）




加速度参数TA和TS可以在每一次运动之间改变。如果你想要最后到停止的减速度用到不同于前面序列中混合加速时间TA或TS，你必须在序列中最后的运动命令之后，但在DWELL或别的使连续序列停止的功能之前指定新的TA或TS。

G01，线性插补模式：

该代码在PMAC中是通过用LINEAR命令来实现的。它的最简单的实现是N01000 LINEAR RET。如果要求速率修调，而它应该在RAPID模式里是无效的，子例程将设置时基资源地址变量给包含外部信息的寄存器（例如，I193＝1833）。

G02，平面顺时针圆弧模式：

该代码在PMAC中是通过用CIRCLE1命令来实现的。最简单的实现是N02000 CIRCLE1 RET。如果要求速率修调，而它本应该在RAPID模式里是无效的，子例程将设置时基资源地址变量给容纳外部信息的寄存器（例如，I193＝1833）。

G03，平面反时针圆弧模式：

该代码在PMAC中是通过用CIRCLE2命令来实现的。最简单的实现是N02000 CIRCLE2 RET。如果要求馈送率修调，而它本应该在RAPID模式里是无效的，子例程将设置时基资源地址变量给容纳外部信息的寄存器（例如，I193＝1833）。

G04，DWELL命令：

该代码要求使用READ命令。不同的G代码在一个P之后或在一个X之后有一个保持时间。PMAC相应地使用一个READ（P）或READ（X）命令就可以分别进行处理；P值将放在Q116里，而X值将放在Q124中。时间的单位也必须被考虑。PMAC的保持时间的单位是毫秒。如果G04的单位是秒，传送的值必须乘以1000。一个典型的实现是N04000 READ（P） DWELL（Q116×1000）RET。

G09，确切的停止：

在一些类型的G代码中，该代码将在两个运动中导致停止，以便在运动之间没有执行角度拐角混合。在PMAC中，通过执行一个短暂的保持可以实现该代码。一个典型的实现为N09000 DWELL10 RET。

G17，G18，G19，用于选择平面：

这些代码将选择执行圆弧插补和刀具半径补偿的平面。G17选择XY平面，G18选择ZX平面，G19选择YZ平面。在PMAC中，这是通过执行NORMAL命令来完成的，NORMAL命令指定了垂直于该平面的矢量（并且并不限制这些选择）。这些代码的标准的PMAC实现将是：

```
N17000 NORMAL K－1
RET
N18000 NORMAL J－1
RET
N19000 NORMAL I－1
RET
```

在这里RET命令处于单独的一行是很重要的；否则，当PMAC返回到调用子例程的那一行时，NORMAL命令将试图从该行中“取得”更多的变量。你可能也希望在这些程序中设置一些变量以标记指定了哪一个平面，如果你需要在其它程序中使用该信息的话（例如，G68旋转）。PMAC的圆弧插补和半径补偿程序不需要这样的一个变量。

G40，G41，G42，刀具半径补偿：

刀具半径补偿可以很容易地用CC0，CC1和CC2命令来打开和关断，这些命令分别对应G40，G41和G42。实现这些的程序将是：

```
N40000 CC0 RET ；关断刀具半径补偿
N41000 CC1 RET ；打开左刀具半径补偿
N42000 CC0 RET ；打开右刀具半径补偿
```

G90，绝对运动模式：

该代码在PMAC中是通过用ABS命令来实现的。没有轴跟随的ABS命令将使坐标系中的所有的轴都处于绝对运动模式下。一般是通过G90000 ABS RET语句实现。如果一种G代码的格式中使G90让圆周运动中心矢量也是绝对的（这不是标准的！），那么在这个程序中将加上一个ABS（R）命令。

G91，增量运动模式：

该代码在PMAC中是通过用INC命令来实现的。没有轴跟随的INC命令将使坐标系中的所有的轴都处于增量运动模式下。一般是通过G91000 INC RET语句实现。如果一种G代码的格式中使G90和G91都影响圆周运动中心矢量的模式（这不是标准的！），那么在这个程序中将加上一个INC（R）命令。

G92，位置设置（预载）命令：

如果该代码只是用来设置轴的位置，那么实现将是非常简单：G92000 PSET RET。使用同一行中的返回语句，程序将跳回调用行，并使用那儿的值（例如，X10 Y20）作为给PSET命令的变量。然而，假如该代码用给别的事物，例如设置主轴的最大速度，那么子例程将需要变得更长一些并且在程序内部完成这些设置。

例如，如果G92被用来在X，Y，和Z轴上预置位置，设置最大主轴速度（S变量），并且定义从刀具顶端到主轴中心的距离（R变元），子例程应该是：

```
N92000 READ（X，Y，Z，S，R）
IF（Q100 & 8388608 > 0）PSET X（Q124）；X轴预置
IF（Q100 & 16777216 > 0）PSET Y（Q125）；Y轴预置
IF（Q100 & 33554432 > 0）PSET Z（Q126）；Z轴预置
IF（Q100 & 262144 > 0）P92＝Q119　　；存储S值
IF（Q100 & 131072 > 0）P98＝M165－Q118 ；存储R值
RET
```

在每一行中设置条件的目的是查看在子例程调用中的该变量是否已经被实际传送给该子例程。如果没有，那么将不会用这些参数执行任何功能（参见上面的《传送变量》部分）。在用到S变量的情况下，该值只是被简单地存储起来，以便别的程序后面使用，因此这里一个命令的主轴速度将不会超过指定的限制。在用到R变元的情况下，程序将计算当前要求的X轴位置（M165）和被声明的径向位置（R变量：Q118）之间的差值，以求得一个偏置值（P98）。该偏置值可以被主轴程序用来计算实时径向位置。

G94，每分钟英寸(毫米)模式：

该代码设立程序以便F值（速率）为每分钟的长度单位（英寸或毫米）。在PMAC里，F值表示速度（每单位时间的长度），其中长度单位是通过轴定义语句来设置，而时间单位是通过坐标系变量Ix90来设置。因为Ix90的单位是毫秒，该程序将设置Ix90到60，000。同时，由于G94通常被用来撤销G95（通过将主轴编码器作为外部时基资源，从而将F值解释为对应主轴每次旋转的英寸（毫米），该程序将把坐标系返回给内部时基。一个典型的程序是：

```
N94000 I190＝60000　；馈送率是每分钟
I193＝2054　　；使用内部时基

RET
```

G95，每转英寸（毫米）模式：

该代码设立程序以便F值（馈送率）为主轴每转的长度单位（英寸或毫米）。在PMAC中，这要求该坐标系的“时基”应该被主轴编码器所控制。速率仍然被解释为每单位时间的长度，但是通过使用外部时基，时间被解释为输入频率的比例，因此也就相当于和主轴旋转速度是成比例的，所以将使得馈送率将表示为每转的有效长度。

子例程要想实现G95必须使程序能够从主轴编码器得到时基，并能得到正确的比例常数。（实际上，这些常量中的一些甚至是全部都可以在时基之前被设定好。）该外部时基功能是通过PMAC的一项软件功能——编码转换表来完成的，编码转换表在《用户手册》反馈功能部分有详细的叙述。而关于怎样建立一个外部时基在《用户手册》的《使PMAC与外部事件同步》的部分中有详细的介绍。

简单来说，时间和频率之间的比例因子必须在定义“实时”输入频率（RTIF）的转换表中加以设定。然后运动程序才能象它已经得到该频率那样被编写。在我们的例子中，我们将采取接近或超过最大值的实时主轴速度。

例如，我们将6000rpm（100rev/sec）用作我们的实时主轴速度。在“实时”中，主轴转一转将花去10毫秒，所以我们想要我们的速率是单位长度每10毫秒，那么只需要设置Ix90（速率时间单位）为10。假如我们主轴一转有4096计数位（解码之后），那么我们的RTIF将是4096×100＝409,600cts/sec＝409.6cts/msec。给转换表比例因子（SF）的公式是：

SF＝131,072/RTIF(cts/msec)＝131,072/409.6=320

该值必须是一个真正的整数，没有任何的舍入误差。如果主轴编码器的分辨率是一个2的幂，那么将是很容易的。如果不是，那么你的单位是rps的实时主轴速度应该是2的幂，而Ix90将不再是整数了（这样处理比较好）。

该比例因子应该被写到转换表里相应的寄存器中。通常，不必在每次执行G95时都要执行这一步；它可以作为系统设置的一部分。给G95的典型程序将包含给坐标系的Ix93和Ix90：

```
N95000 I190＝10      ；PMAC F是长度/10毫秒
I193＝1833           ；时基资源是外部的
RET
```

G96，恒定表面速度模式有效：

该代码在程序中是使主轴处在恒定表面速度（CSS）模式中。在这种模式里，主轴的实时角速度是变化的，以便经过刀头的表面速度能够保持恒定。实质上，这意味着主轴的角速度是和刀头到主轴中心的距离成反比的。该距离通常是X轴位置，这意味着X轴0零位置通常是主轴的中心。一些G代码允许局部程序用G92R（q.v.）来产生一个X轴偏置，它定义了在当前X轴命令位置的径向距离。

模式建议的方式是使主轴单独处在和其它轴不同PMAC坐标系中。这就允许主轴程序可以按来自主程序的不同比率被执行或反应，最终仍然被局部程序通过变量和标志加以控制。这种类型的主轴程序将在下面加以详细解释。

一个G96代码将按照单位英尺/分钟或米/分钟来记忆主轴表面速度S代码。该值将被放在给主轴程序的一个变量中准备被读取。标志也应该被设置以标明主轴所处的是一个怎样的模式。要记住，主轴模式和速度可以被独立地设置为主轴开/关状态和方向（参见M03，M04，M05）。

一个典型的G96程序是：

```
N96000 READ（S）     ；将主轴表面速度读入Q119
P96＝Q119            ；存储主轴速度
M96＝1               ；标志CSS模式
RETURN
```

G97，恒定表面速度无效：

该代码撤销了主轴恒定表面速度模式，并将主轴放到恒定角速度模式中。在这个模式下，主轴速度独立于刀具的径向位置。由于主轴处在一个单独的坐标系中，执行该代码的子例程将简单地设置一个变量和一个标志以便程序查看。通常，G97代码将按RPM来记忆一个主轴速度S代码。如果这样执行了，程序将读取它并将它放到一个变量中。如果没有这样执行，程序将允许主轴程序保持最近的在G96下从表面速度和径向距离计算的RPM。

一个典型的G97程序是：

```
N97000 READ（S）                ；将主轴RPM读入Q119
IF（M100 & 262144 > 0）P97＝Q119 ；给主轴程序存储
M96＝0                          ；取消CSS模式
```

**主轴编程：**

在PMAC中控制主轴可以通过许多方法来完成，当然这根据主轴需要做什么来决定。最简单的主轴操作就是要求主轴在一段充分的时间内按一个方向或另一个方向以恒定的速度运动。在这种情况下，不必编写主轴运动程序；PMAC只要输出一个正比于速度的电压即可（因此主轴处在开环状态下直到PMAC被考虑），或者主轴电机被受到控制（在PMAC闭环控制下）。

主轴手动控制：

手动控制的主轴电机不必处在任何坐标系中（它禁止处在和别的轴同样的坐标系中，否则当一个局部程序在运行时，它是不能被手动控制的），但是将该电机放在一个不同的坐标系中将是一个好主意，因为不处在任何坐标系中的电机将使用第一时基控制的**坐标系**（速率修调）。

主轴速度值将被进行比例计算，并被放到手动控制速度I一变量中（Ix22），而主轴开/关功能将命令手动控制起始和停止（参见M03，M04，和M05）。

主轴开环控制：

如果你将主轴开环使用，你可以通过使用一个M一变量直接写入另外的一个未用的DAC输出寄存器。举例来说，语句M425一〉Y:$C00A,8,16,S将变量M425和DAC4输出寄存器匹配起来。给该M一变量的任何输入值都将导致在DAC4输出线上的相应的电压。通过这种方式,一个主轴启动命令（参见M03，M04）可以是这样,M425＝P10或M425＝一P10，其中P10已经通过一个S代码预先进行了设置。主轴关断命令（参见M05）可以是M425＝0。

在主轴和定位之间切换：

在有些情况下主轴电机时而被象普通轴那样使用，执行的是位移运动而不是恒定的速度运动，时而被用作主轴运动。在这种情况下，示轴电机将被作为放在主坐标系中的轴，以便能够执行协调的运动。当要求实时的主轴操作时，将通过设置电机的比例增益为零并且写输出偏置寄存器（Ix29）来产生一个假开环模式。在这种模式中，Ix29将被作为上面所述的M425来对待。当然，对于这种模式的操作将要求一个速度环（转速计）放大器。参见示例OPENCLOS.PMC可以得到更多的细节。

主轴的恒定表面速度：

假如你希望主轴能够执行恒定表面速度（CSS）模式，你必须编写一个运动程序，因为速度作

为另外一根轴的位置的功能必须是变化的。在示例SPLINE.PMC显示中，推荐的方式是将运动分解为小的时间段，给每一时间段的命令距离依靠此时的系统条件而定，包括命令速度和刀具径向位置。

如果主轴要在CSS模式中被开环控制，那么最好用一个PLC程序作为刀具径向位置功能修正输出命令（Mx25或Ix29）。该PLC程序的结构和示例SPINDLE.PMC中的闭环运动程序很相似，处理并不需要实际的运动命令；一旦经过数学处理，该值将被分配给相应的变量。

**标准M代码：**

下面的部分将详细地讨论在实施标准M一代码中所遇到的问题。认识到程序中M代码和M一变量的区别是很重要的。它们看起来可能是一样的，但是如果被作为一个M一变量来解释，它必须只被用在一个等式或表达式中。举例来说，M01＝1参考M一变量序号1（通常这设置了机器输出1），其中M01本身是M代码序号1。M代码被作为运动程序1001中相应行标的子函数调用来对待。

M00，编程停止：

执行该代码的程序要包括STOP命令。该代码寻找PROG 1001的行标N0，任何程序的起始总是被隐含定义为N0，所以这必定是在PROG 1001的最顶端。实现该代码的文件的一部分应该是：

```
CLOSE
OPEN PROG 1001     ；缓冲区控制命令
CLEAR              ；当送入新值是擦去旧的值
STOP               ；实际程序的第一行
RET                ；重起时将跳回
```

M01，选项停止：

假如操纵者的面板上设置了“可选停止”开关，那么该代码将被用来执行停止功能。假定该开关连接到PMAC的机床输入1，并且变量M11被分配给该输入（这是默认的），那么执行该代码的程序应该是：

```
N01000 IF （M11＝1） STOP
```

```
RET
```

M02，程序结束：

由于PMAC自动地辨认一个程序的结束，并且将程序指针重置到程序的顶端，给该代码的程序将是空的（只是RET语句）。所以，在许多系统中，大多数变量和模式被设置为默认值。一个典型的程序结束的程序将是：

```
N02000 M55＝0      ；关掉主轴
M7＝0              ；关掉冷却剂
M2＝0              ；关掉转换表
LINEAR             ；确认不是在圆弧模式下
RET
```

M03，按顺时针方向的主轴：M4，按反时针方向的主轴：M05，主轴停止：
假如主轴只是执行恒定速度运动，程序将简单发出手动控制命令。例如：

```
N03000 CMD “＃4J＋”

RET
N04000 CMD “＃4J－”
RET
N05000 CMD “＃4J/”
RET
```

当然，这是假定了PMAC上的4＃电机是主轴电机，并且字增计数方向是顺时针方向。主轴速度在另外的程序中通过设置I422（4＃电机手动控制命令）已经被决定了。如果PMAC通过一个开环电压来控制主轴，程序将通过写DAC寄存器来在一个未用的模拟输出上输出一个电压。例如：

```
N03000 M402＝P97×P9
RET
N04000 M402＝－P97×P9
RET
N05000 M402＝0
```

该示例假定M402被分配给DAC4寄存器（Y：＄C00A，8，16，S），P97是RAM里要求的主轴速度，而P9是联系RPM和DAC位的比例因子（3,276.7DAC位/伏特）。参见《主轴编程》部分可以得到更多的详情 。

假如要求象恒定表面速度那样的任务，那么将需要给主轴一个单独的运动程序，就象前面的例子中所表现的那样。如果这些M－代码要和该示例联系起来，那将是：

```
N03000 M55＝1      ；主轴顺时针标志
CMD “&2B1010R”     ；启动主轴程序
RET
N04000 M55＝－1    ；主轴逆时针标志
CMD “&2B1010R”     ；启动主轴程序
RET
N05000 M55＝0      ；无主轴标志
RET
```

M07，低能级冷却液打开（喷雾）
M08，高能级冷却液打开（大量）
M09，冷却液关断

这些M－代码的实际应用很大程度上取决于机器，但是却非常简单。例如，如果冷却液的开/关控制连

到PMAC的机器输出7上，而冷却液大/小控制连到PMAC的机器输出8上，那么该程序将是：

```
N07000 M7＝1    ；设置机器输出7：冷却液开
M8＝0           ；清除机器输出8：低能级
RET
N08000 M7＝1    ；设置机器输出7：冷却液开
M8＝1           ；清除机器输出8：高能级
N09000 M7＝0    ；清除机器输出7：冷却液关
RET
```

如果需要为发生变化提供一些时间，DWELL语句能够被加到输出设置的前面和（或者）后面。

M12，切屑传送带打开；M13，切屑传送带关闭

这些M一代码的实际应用很大程度上取决于机器，但是却非常简单。例如，如果传送带的开/关控制线连到PMAC的机器输出2上，那么其程序将是：

```
N12000 M2＝1      ；设置机器输出2：传送带打开
RET
N13000 M2＝0      ；清除机器输出2：传送带关闭
RET
```

M30，带循环的结束程序

参见M02描述。在大多数系统中M30等效于M02，但是它将使程序返回到程序的开头。

**默认条件：**

通常，一台执行G代码程序的机器除了在启动/重置周期中PMAC自动设置的条件之外，还要求许多默认值和默认模式。为了设置这些默认条件，最好使用PLC1程序，它将在自动启动/重置周期后第一个被执行（这有效地扩展了该周期中被执行的内容）。程序中的最后一行将是DISABLE PLC 1，这可以避免重复执行该程序。一个给该程序的简单示例如下：

```
CLOSE
OPEN PLC 1
CLEAR
M55＝0          ；主轴关闭
P92＝3000       ；使主轴速度最大（RPM）
P95＝1000       ；使主轴加速度最大（RPM/sec）
M70＝0          ；英制测量单位
DISABLE PLC 1   ；只执行一次
CLOSE
```

## 循环运动程序缓冲区：

循环运动程序缓冲区允许在程序执行期间对程序行进行下载，并覆盖已经被执行的程序行。这就能够连续执行比PMAC的存储空间大的程序，并且实时下载程序行（等效于SMCC的MDI模式）。

**定义一个循环缓冲区：**

每一个坐标系都有一个循环程序缓冲区。为了给一个坐标系创建一个循环缓冲区，对该坐标系选址（&n），并发出DEFINE ROT {常量}命令，其中{常量}是存储器中缓冲区的大小。程序中的每一个值（例如，X1250）都占去内存中的一个字。缓冲区应该被定义的在对你希望加载的程序执行点之前的距离有足够大的空间。既然大多数利用循环缓冲区的应用并不限制PMAC的存储要求，加大缓冲区将是一个好办法。

例如，假如在一个四轴应用中你希望在程序执行点之前读入100程序行（你用常量来表示该执行点的位置，例如，X1000 Y1200 Z1400

A1600），那么在缓冲区中你至少需要400字的存储空间，所以给该循环缓冲区分配500或600字的空间是一个比较好的办法（例如，DEFINE ROT 600）。

给定义要求的缓冲区状态：

对于PMAC来说，想要为循环缓冲区保留空间，就将没有数据采集缓冲区，而且也没有给一个在发出DEFINE ROT命令时的更高编号的坐标系的循环程序缓冲区。因此，你首先必须删除所

有的数据采集缓冲区，并且从高的编号到低的编号来定义你的循环程序缓冲区：例如：

```
DELETE GATHER
&3 DEFINE ROT 200
&2 DEFINE ROT 1000
&1 DEFINE ROT 20
```

**准备执行：**

为了准备在一个坐标系中执行一个循环程序，当选址坐标系时使用命令B0（到循环程序0的起始处）。这只有在没有缓冲区被打开时才能被执行，否则它将被解释为一个B轴命令。一旦准备这样好了，用一个R命令就可以开始执行程序了。该命令在缓冲区打开或关闭时都可以被给出。如果该命令被给到一个空的循环程序缓冲区，那么该缓冲区将等待发给它一个命令，接到该命令后将立即执行它。

**打开缓冲区以便输入：**

OPEN ROT命令将打开所有已经定义的循环程序缓冲区。紧接着该命令的程序行将被送到主机选址（&n）的坐标系的缓冲区中。大多数循环缓冲区的用户都只有一个坐标系，所以这和这些用户无关。但是当一次用到几个循环缓冲区时，就可能需要在坐标系之间切换了。在OPEN ROT命令之后，PMAC将尽可能多的命令作为缓冲的命令来对待，即使它立即执行了这些命令（一些命令作为在线命令时是一样的，但是作为缓冲命令就是不同的了），认识到这一点是非常重要的。举例来说，命令I100当缓冲区关闭是时要求I—变量100的值，但是当运动程序缓冲区打开时，它是命令执行一个半径是100单位的整圆周（I值是径向矢量的X轴分量；如果没有给定轴位置，它们都被假定为和起始点一样）！

**处在被执行的行之前：**

处理循环程序缓冲区的关键是要知道在你之前有多少行；也就是说，在PMAC执行的程序行之前，你究竟需要读入多少程序行。通常，你需要读入一定数目的行，然后等待，直到执行的程序追到更小数目的行时再开始重新执行。一个实时应用将在执行该行之前在线工作；一个周期性下载大文件的应用在前面将会有1000行，当程序到500行之内时，程序将再次开始执行。

PR命令：

告知在你之前有多远有好几种方法。第一个是PR（程序保持）命令，它将返回前面的行的数目。这就提供了一个非常简单的查询方法，但是它对于紧密的实时应用可能并不太好。

BREQ中断：

对紧密耦合的应用，将有硬件线来处理给循环缓冲区的信号交换输入输出，以及控制这些线的转换点的变量。当被选址的坐标系的循环缓冲区需要更多的程序行时，BREQ（BUFFER REQUEST）线变高，当该循环缓冲区不需要更多的程序行时，它将变低。该线被连接到PMAC-PC的可编程中断控制器上，所以它可以被用来给主机产生一个中断。（参见下面的“使用PMAC中断主机”部分。）作为补充，在PAN插座上提供了BREQ/。对于每一个坐标系都有一个“缓冲区满”的状态位（BREQ/）。

I17停止中断：

变量I17控制着主机能够读入前面多少行，并让BREQ为“真”。如果你给一个循环缓冲区送出一个程序行，BREQ变低，至少是暂时地变低。如果在被执行的行之前的行的数目仍然少于I17指定的行的数目，BREQ又将变高，这将产生一个中断。如果你使该行数等于或是多于I17所指定的行数，BREQ将保持为低。当你用OPEN ROT命令来进入一个循环程序缓冲区或者改变被选址的坐标系，BREQ将变低，然后如果缓冲区中在

执行点之前的行的数目少于I17所指定的数目，BREQ

将被设为高。

I16重新启动中断：

变量I16控制着当在循环缓冲区中的正在执行的程序追上最后一个被读入的行时，BREQ在哪儿重新被设置。如果在一行执行完之后，在循环缓冲区中前面比I16指定的行数少，BREQ将被设为高。这可以被用来提醒主机需要送出更多的程序行。

通过使用这两个变量和给中断的BREQ线，你能创建一个非常快而有效的系统以实时地从PC下载程序。

如果缓冲区被用完时怎么办：

假如程序计算追上了循环缓冲区的读入点，这将不会有错误发生；程序操作将被悬挂起来直到更多的行输入到循环缓冲区中。技术上，程序仍然在执行；为了真正地使程序执行停止，需要发出一个Q或A命令。

假如PMAC是处在分段模式下（I13>0）并且正在执行循环缓冲区中的最后一行，只要一个新的程序行在开始减速到停止之前被输入，那么PMAC将进入到新的运动中而不会停止。

**关闭和删除缓冲区：**

CLOSE命令将关闭循环缓冲区，正如它对于其它类型的缓冲区所做的那样。关闭循环缓冲区将不会影响到缓冲区程序的执行；它只是阻止新的缓冲命令被输入到缓冲区中，直到这些缓冲区重新被打开。DELETE ROT给已选址的坐标系擦去循环缓冲区，并去除曾经为它保留的存储空间。

## PMAC怎样执行一个运动程序：

知道PMAC怎样执行一个运动程序是很重要的。一个运动程序和一个典型的高级计算机程序是有着基本的不同点的。在一个典型的高级计算机程序中，有着“占用时间”的语句（运动，DWELLS，和DELAYS）。在计算时间和执行时间之间有着重要的区别。

一个PMAC程序将数据传送给轨迹生成程序以计算给电机每伺服周期的一系列的命令位置。运动程序必须工作在实际要求的运动之前以便保持数据能够传送给轨迹生成器。如果一个程序没有能够保持超前，而且给下一次运动的时间在没有正确的数据送给轨迹生成器的情况下来到，PMAC将停止执行程序，并使坐标系中的所有电机停止。

**超前计算：**

PMAC将在一个或两个运动之前处理程序行（包括DWELLs和DELAYs）。为了能够将运动混合到一起，超前一个运动是必须的；如果要执行正确的加速度和速度限制，或者想要计算一个三点样条（SPLINE模式），那么超前两个运动计算将是必须的。对于I13等于零的线性混合运动（I13是运动分段时间，当等于零时无效），PMAC将超前两个运动进行计算，因为这时速度和加速度限制将有效。在所有其它的情况下，PMAC将超前一个运动来计算。

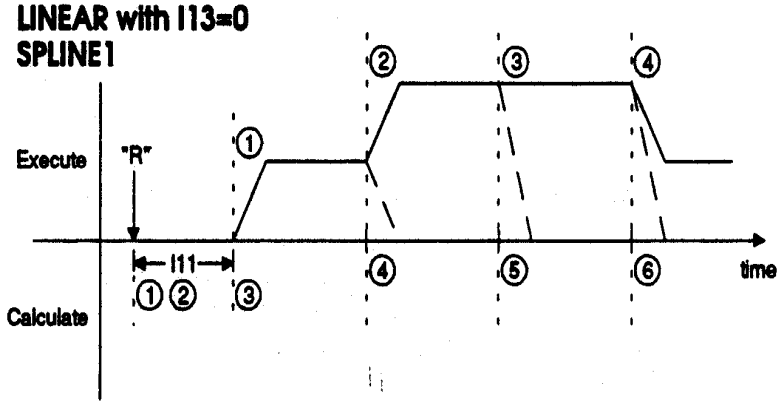
注意：没有速度限制或加速度限制的动态运行将是十分安全的。一个动态算法超前查看的运动越多，那么它就越可能在所有情况下成功执行，但是对于某一些，在时间之前必须估算整个运动顺序。

**开始计算：**

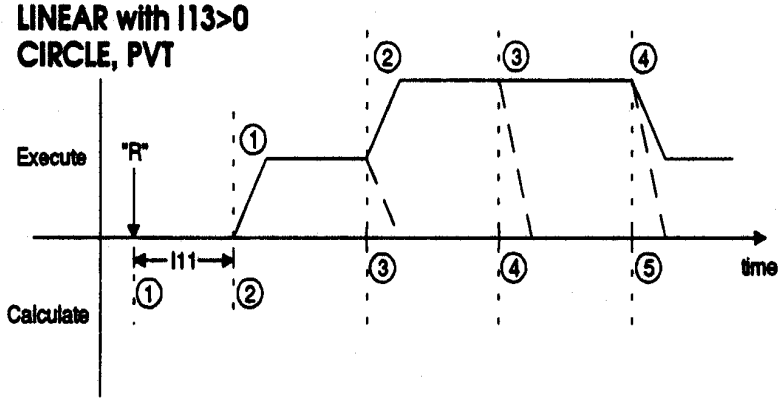
在开始执行程序之前，PMAC将顺次计算程序语句，包括第一和第二个运动语句，这将依靠运动的模式和I13的设置。这能够包括多重模态语句，计算语句，和逻辑控制语句。

# PMAC Motion Program Precalculation

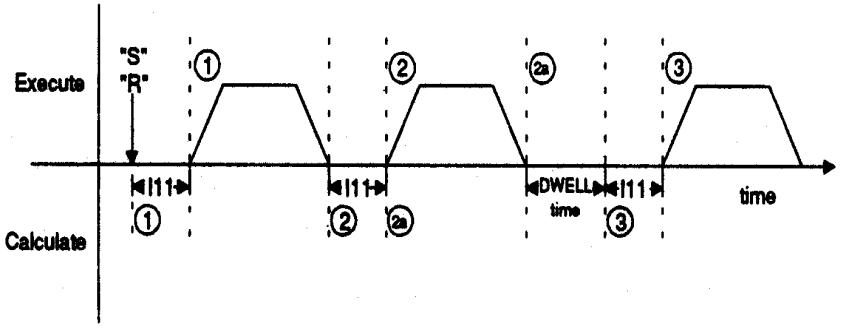
**A. Two moves ahead**
**LINEAR with I13=0**
**SPLINE1**




**B. One move ahead**
**LINEAR with I13>0**
**CIRCLE, PVT**




**C. No moves ahead**
**Ix92=1, RAPID,**
**HOME, DWELL,"S"**



可编程运动直到I11毫秒已经被传送才会实际开始执行，即使计算已经完成。这就允许在卡之间的同步，以便在其它卡没有开始之前不会有卡开始执行。如果I11被设为零，第一个运动将在计算被完成之后立即开始。

**顺序运动的计算：**

通过轨迹生成程序执行一个运动一开始，一个标志就将设置给PMAC以便计算所有下面连续的程序语句，也包括下一个运动语句，并将计算好的数据放到给轨迹生成器的队列中。然后程序计算将暂停，直到轨迹生成器开始下一个运动，并且PMAC执行另一个任务（PLC程序，通信等等）。

计算时间不够时：

假如在一个运动的执行开始时PMAC还不能完成对该运动的轨迹计算，PMAC将取消该程序，并在它的状态字中显示一个执行时间错误。这通常发生在运动时间被设置的非常短（几个毫秒），并且（或者）在两个运动命令之间有大量的计算的时候。关于该“运动块”计算率的限制将视具体的应用而定，但是通常是每秒几百个块。

如果你关心该运动计算限制，那么在开发期间你应该让你的计算时间连续的变短直到PMAC失效为止。然后对于你最后的应用，你可以确认你的最小运动时间比该值大，通常的安全边界应该是比最小值还大25％。

**当没有提前计算时：**

在一个运动程序中可能会有一些条件将中止混合运动并停止提前运算。在这些情况下，PMAC在它开始下一个或下两个运动的计算之前，将等待该操作完成。在任何这些中断期间，PMAC将使用I11计算时间以延迟下一个运动的开始。

DWELL命令：

运动程序中的一个DWELL命令将终止运动的混合，以便在DWELL命令期间PMAC不会进行计算。PMAC将不会开始计算后面的运动直到DWELL命令时间完成之后。相反的，DELAY命令实质上是一个指定时间的零距离运动命令；PMAC在DELAY命令期间仍然执行计算。

回零，快速运动：

如果在一个程序中命令一个回零运动（HOMEn）或者一个快速模式的运动，该运动将不会和其它的运动相混合。PMAC将不会开始计算后面的运动直到所有电机完成它们要求的这些类型的运动。

PSET命令：

如果在一个运动程序中用到一个PSET命令来重新定义轴位置，PMAC将不会把PSET命令之前的运动和该命令之后的运动互相混合起来。PMAC将不会开始后面运动的计算直到前面要求的运动已经被完成，并且PSET命令已经被执行了。

两次回跳规则：

如果在计算下一个运动的流程中，PMAC检测到在程序的逻辑中的两次回跳，PMAC将不会把上一次已经计算过的运动和下一个即将开始的运动混合起来。这些回跳既可能由ENDWHILE语句，也可以由GOTO语句引起；GOSUB，CALL，和RETURN的跳转在此不计入此类。该规则的目的是阻止如果PMAC在一个短的运动上需要重复多次，由于计算时间不足而导致程序被取消的现象发生。

*混合停止：*

PMAC将允许先前的运动停止，并且将在下一个实时中断（参见I18的描述）开始再次计算程

序，这将持续下去直到它发现下一个运动语句，或者多于两个的回跳（在这种情况下，处理过程是重复的）。这允许不确定的等待循环，这种等待循环不会由于计算时间的不足而导致PMAC取消该运动程序。

*嵌套循环：*

该“两次回跳”规则能够导致编程者在当他们在嵌套WHILE循环中计算运动时，由于疏忽而使得混合停止。认真考虑下列的示例，该例子试图在内环产生一个连续的混合的正弦运动，用外环来标志该正弦的范围：

```
SPLINE1 TA20
P1＝0
WHILE（P1<10）
    P2＝0
    WHILE（P2<360）
        X（P1×SIN（P2））
        P2＝P2＋1
    ENDWHILE
    P1＝P1＋1
ENDWHILE
```

第一个360份将通过PMAC在内环执行程序而被混合到一起。但是当PMAC使P2增加到360时，它就触及了第一个ENDWHILE并且跳回到内环的WHILE条件处，此时该条件是“假”，所以程序执行将跳到下面的一句，增加P1，直到触及第二个ENDWHILE，然后跳回外环的WHILE条件处，而这些将不会遇到一个运动命令。

此时，PMAC将调用“两次回跳”规则，并使前一次的编程运动停止。PMAC这么做是为了防止可能由于在没有运动的循环中被不明确的“真”的设置所捕获，而导致对下一个运动等式来不及准备。当这次运动完成之后PMAC将恢复计算，并将启动内环中的下一个顺序运动。

但是，如果你希望将所有这些运动都连续地混合到一起，那将得到什么呢？仅仅是将内环的最后一个运动放到内环的外面。通过这种办法，两个ENDWHILE语句在两个运动命令之间将绝对不会遇到一起：

```
SPLINE1 TA20
P1＝0
WHILE（P1<10）
    P2＝0
    WHILE（P2<359）       ；让循环早些结束
        X（P1×SIN（P2））
        P2＝P2＋1
    ENDWHILE
    X（P1×SIN（P2））      ；最后的运动到内环外
    P1＝P1＋1
ENDWHILE
```

循环等待：

有好几种方法可用来保持程序的执行，以等待一定的条件发生。通常这总是和一个WHILE循环一起来执行的，但是在循环的内部，其效果将和读入的计算有关。

最快的执行是WHILE({条件})
WAIT循环。一旦遇到WAIT命令，运动程序计算将被悬挂起来直到下一个实时中断，在那时，将重新评价条件。运动程序将更象一个单线PLC程序。如果下一个RTI已经发生了，PMAC将重新输入中断服务程序，并重新评价条件。如果这重复发生，后台程序将会因此而缺乏时间，将使PLCs和通信变慢，在更坏的情况下，将触及监视时钟。通常这只发生在多重坐标系在同步的WHILE・・・WAIT循环中。

具有相似速度的还有空的WHILE・・・ENDWHILE循环，或者其中有一个非运动命令。每一次实时中断，它将执行两次，然后被两次回跳规则停止。在下一次实时中断计算将被恢复，或者假如实时中断已经发生，这将立即恢复计算，并使后台程序由于缺乏时间而产生同样的结果。

使用WHILE({条件})DWELL单线命令有助于更好地控制循环比率，给后台程序以时间。条件在每次D

WELL命令之后只评价一次。

**提前计算的含义：**

在一个连续的顺序运动期间，提前计算运动程序的需要意味着非运动动作（特别是一些输出的设置）将在你认为它们被执行之前就将被程序执行，或者被一或两个运动所执行。对于只在程序中用到的变量，这没有问题，因为所有事情都在程序中顺次发生。

将这些非运动动作在程序中向后移动一到两个运动以便让它们在要求时再发生，这是完全可能的。然而这将使程序按照正常的操作顺序会非常难读。

同步的M－变量分配：

同步的M－变量分配语句被用来处理该问题。这类语句用双等符号（＝＝）来代替单等符号。这对于PMAC来说是一个标志，使得PMAC将拖延语句的实际执行直到运动的开始就紧接在后面，所以实际的动作将符合实际的运动。

同步的M－变量分配语句在《用户指南》的计算特征部分有详细的介绍，语法介绍是M{常量}＝＝{表达式}，在《程序命令描述》部分。

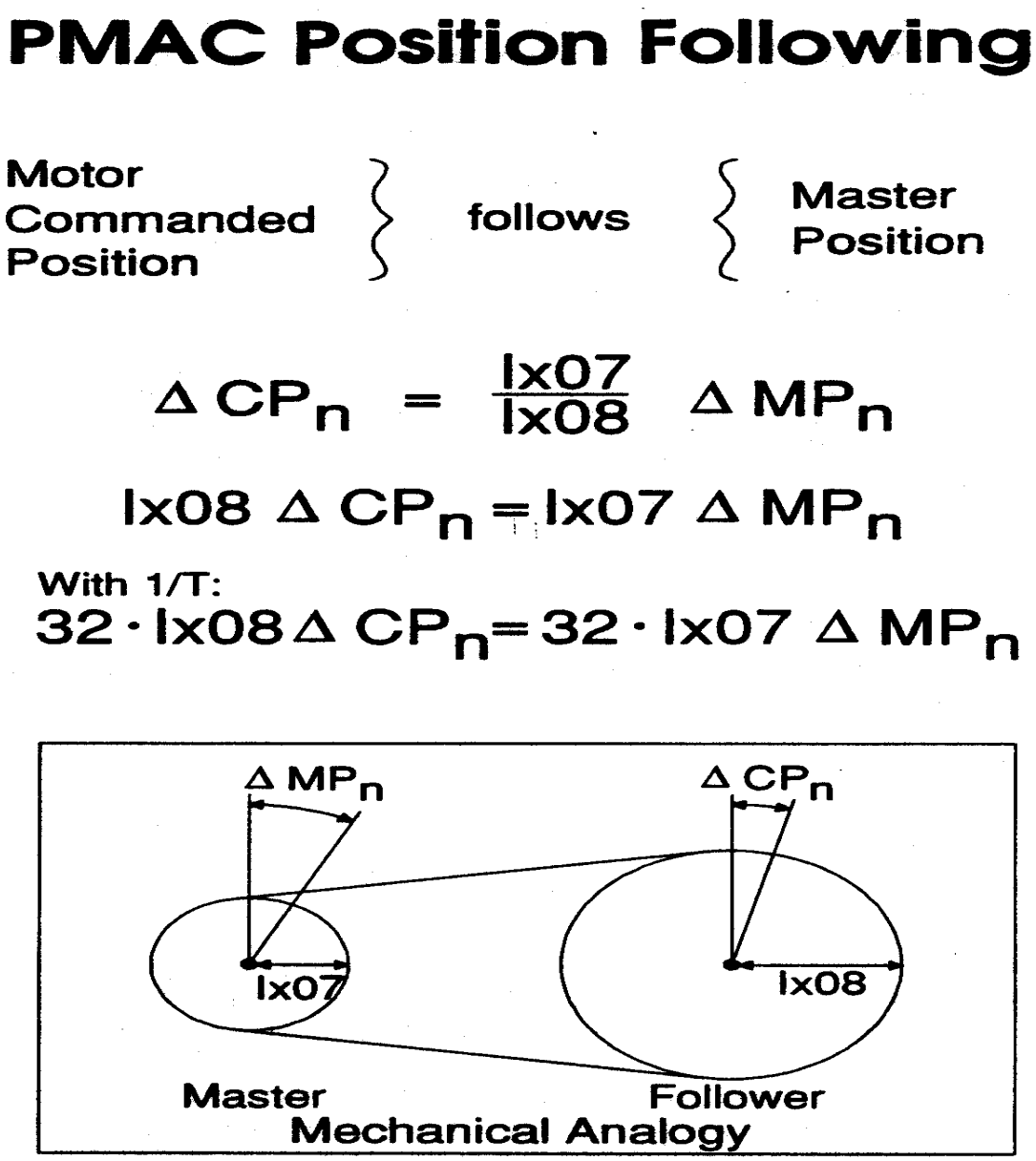
# §3－12 使PMAC与外部事件同步

PMAC有许多有强大的特性来帮助它控制下的运动与外部事件同步。这些包括位置跟随，通常叫做电子齿轮；时基控制，通常叫做电子凸轮；位置捕捉，在定位操作中很有用；位置比较，可用于精确的扫描和测量功能。以下将对每一部分进行描述。

## 位置跟随(电子齿轮)

PMAC有几种办法来协调在它控制下和不在它控制下的轴。最简单的办法就是基本的位置跟随。这是一个电机对电机的功能，而不是象时基跟随那样的坐标系功能(见下)。一个来自主轴(不由PMAC控制的轴)的编码器信号被送给PMAC的编码器输入。该主信号通常从一个开环驱动器或一个手轮按钮得到。

**PMAC位置跟随**

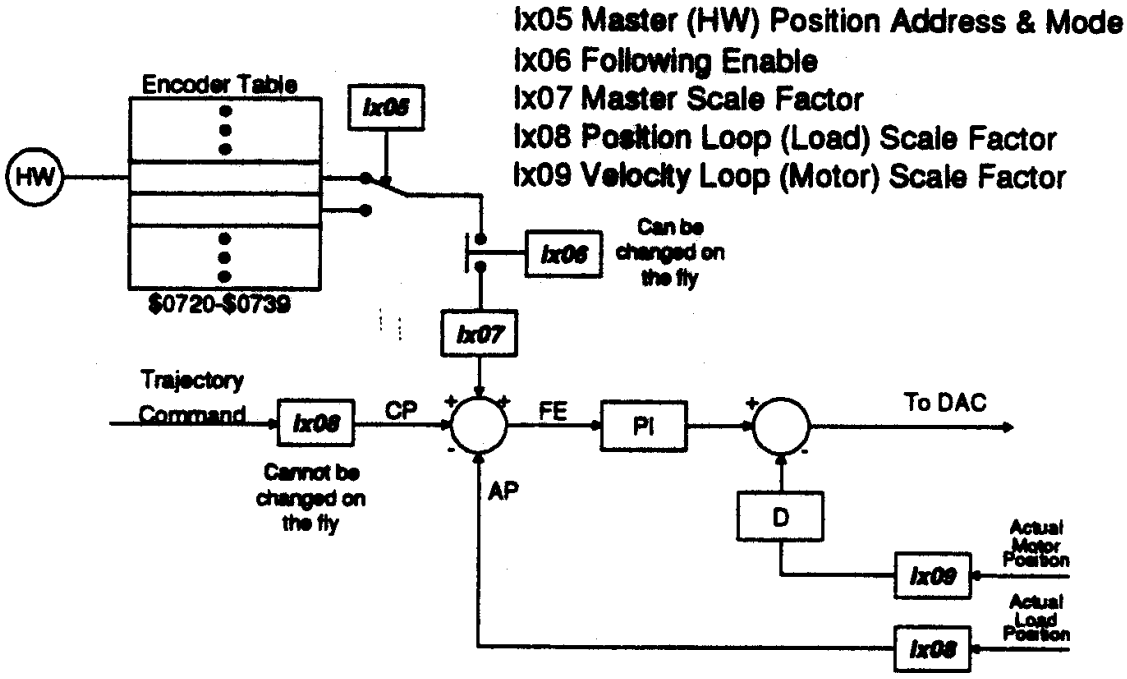



**位置跟随I变量**

一个或多个PMAC的电机被告知该编码器寄存器是主寄存器(用Ix05)。Ix05是一个地址I变量；就是说，它的值包含了保存主位置信息寄存器的地址。该寄存器通常是编码器转换表内一些处理过的位置

数据。在对主编码器的采样中，它有助于完成1/T插入或减小量化误差(这在缺省设置中是自动完成的)。

每个电机的跟随比由Ix07和Ix08设置，它们可以被看作机械传动系统中主从齿轮的齿数。随动功能由Ix06来看作开关。当它打开时，来自主轴的输入流就象一个轨迹生成器，产生跟随轴要到达的一系列指令位置。




**PMAC跟随参数**

**运行中改变跟随比**

如果你想在运行时改变你的跟随比，你可单独改变Ix07。由于Ix08与伺服反馈的计算有关，所以不能在运行时加以改变。

应该在分辨率和系统的伺服性能间找到平衡。放大系数Ix08越大，分辨率越高。但同时Ix08越大，比例增益Ix30就越小以防止产生内部饱和，而且最大速度也越低以防止那些寄存器的内部饱和。一般地，Ix08应保持在1000以下。

**可编程运动的叠加跟随**

另外，该跟随功能可在计算轨迹时叠加。这允许，例如，在对一个运动的金属薄条进行成型加工时，可以在不考虑薄条运动的条件下写出成型程序，而一个由薄条送来的主信号将对运动做成补偿。

Ix05的第16位决定跟随功能是否处于“偏置”模式下，即跟随电机的位置并不反映由于跟随产生的变化，或是“正常”模式下，即与前者相反。在用户使用跟随叠加和进行可计算的运动时，通常会使用“偏置”模式，这时第16位为1(如果主位置信号从X:$0721得到，参数值可以是，例如，I105=$10721)。

## 时基控制(电子凸轮)

一个与外部轴协调的更复杂的方法是时间基数控制，即用输入信号的频率来控制运动和程序的执行速度。时基控制在整个的坐标系中进行。用户必须指定哪一个编码器寄存器接收输入频率，以及输入频率与程序执行速度间的关系。这不仅仅使运动速度与输入频率成比例(频率始终在接近零)，而且保持所有位置的同步。它允许多线螺纹加工等操作。

**什么是时基控制?**

PMAC的运动语言把位置轨迹表述成时间的函数。无论运动是由时间直接指定，还是通过速度，最终轨迹是被定义为位置和时间的函数。

这对于大多数应用是好的。但是，在一些应用中，我们希望PMAC的轴从动于不在PMAC控制下的一根外部轴(或有时是在PMAC控制下的另一个坐标系的独立轴)。在这些应用中，我们希望把PMAC的轨迹定义成主位置的函数，而不是时间。

实时输入频率

PMAC完成这一功能的方法是使“时间”同主轴通过的距离成比例，而不是通过语言表达成“时间”的函数。这是通过定义一个从主轴位置传感器得到的“实时输入频率”(RTIF)来完成的，它的单位是步每毫秒。举例来说，我们定义一根RTIF为32cts/msec。那么，在时间基数模式下，当程序中为1 msec时，它其实是指主编码器进行32步计算，而无论物理距离是多少。如果我们在从动程序中编制一个需耗时2秒的运动程序，其实需要主编码器进行64,000步计算来完成它。

选择RTIF的限制

如果PMAC在它的时基计算中有无限的分辨率和无限的动态范围，那么实时输入频率的选择将是完全可以假定的--

你可选择任何你想要的RTIF，并用这个RTIF来编写你的运动程序。但是，PMAC是在整数范围内进行时基的计算，这限制了它的分辨率，而且用的是24位的寄存器，这限制了它的动态范围。

这些极限导致了RTIF选择的三个限制:

1.)

从RTIF得到的时基放大系数(TBSF)必须是整数。PMAC进行计算需要的值不是以cts/msec为单位的频率，而是以msec/cts为单位的频率的倒数。为了使该值在整数范围内，规定把频率的倒数乘以$2^{17}$ (131,072)。

如果选择100cts/msec为RTIF，那么TBSF将是131,072/100=131.072，它不是整数。PMAC只能接受整数部分131，这就产生了偏移。

选择以2为底的实时输入频率(例如，32,64,128)就总能得到一个整数的TBSF。也可以选择RTIF为一个以2为底的数与一整数相除得到的值，例如，204.8cts/msec(=2048/10，=$2^{10}$/10)将使TBSF为$2^{17}/2^{10}*10=640$。

2.)

当输入频率(IF)与RTIF的比值(IF/RTIF)等于伺服更新频率(以KHZ为单位)时时基计算将会发生饱和。在缺省的伺服修正频率2.25KHZ和32cts/msec的RTIF的条件下，不发生饱和的最大可接受输入频率为32*2.25=72cts/msec如果系统以100cts/msec工作，则RTIF=32cts/msec是不可接受的，而RTIF=64cts/msec则是可接受的(100/64=1.5625<2.25)

选择RTIF大于最大输入频率总是可接受的。

3.)

如果使用PVT或SPLINE运动模式，RTIF的分段时间必须是整数。这意味着必须选择RTIF使总的循环时间为整数。

有时做到这一点是不可能的，除非主编码器的分辨率是2的指数倍。因此，我们希望选择2的指数倍作为主编码器的分辨率(例如，1024线/转，而不是1000线/转)

举一个例子，对于一个有1000线/转编码器的主轴电机，RTIF为200cts/msec对应速度为50转/秒(3000转/分钟)，即20毫秒/转。但是，得到的TBSF为655.36，不是一个整数。

对于这种编码器，若从RTIF能得到一个整数的TBSF—

例如，256cts/msec得到TBSF为512,204。8cts/msec得到TBSF为640—

但同时每一转的时间就不是一个整数。

但是，用一个1024线/转(4096步/转)的编码器，RTIF为204.8cts/msec，得到的速度为50rps(3000rpm)，即每转20毫秒。在这种情况下，时间基数功能将被锁定，可写入一个SPLINE或PVT的顺序而得到一个确定数值的主轴转数。

它如何工作?

时基控制通过给每个伺服周期都会发生的指令位置修正公式提供假的从上一个伺服周期以来的时间来工作。(比例I10包含实际时间的数值)注意，伺服周期的实际时间并没有改变，伺服环的动态性能也没有改变。仅仅是指令轨迹的速率随外部频率而变化，而由于坐标系内所有轨迹一起变化，所以空间的路径并没有改变。

**使用外部时基信号的指令**

使用一个外部时基信号需要好几步设置。但一旦完成了设置，时基控制对于用户和程序来说是透明的—它们将自动执行。设置的细节如下所示:

第一步:信号解码

输入给PMAC的信号是某一个增量编码器的输入(通道A或B)。信号必须或是一个正交信号(同输出一样)，或是一个脉冲和方向信号(脉冲进入A，方向进入B)。对于使用的编码器(编码器1到16的一个)，编码器I变量0(编码器1为I900，编码器2为I905，etc。)控制解码方法，并定义哪一种"步数"。例如，对于输入编码器4的正交信号，I915=3或7定义每个编码器周期为4步，而I915=2或6定义每个编码器周期仅为2步。3或7和2或6的区别在于解码器用哪一个信号传感器来加计数。

| 你必须确定你对主信号进行向上计数, 向下计数将导致一个负的时基，PMAC将无法接受。 |
|---|

*频率的模拟源*

PMAC上有一个电压到频率(V到F)转换器，允许把一个电压输出连到JPAN接口的WIPER线上来控制时间基数。0到+10V的输入被转换成名义上的0到250KHZ的频率(25KHZ/V)。跳线E72和E73ON时把该信号连到编码器4的解码器计数器上(这儿无法选择用哪个编码器)。确定跳线E24的1，2针被连在一起(缺省)。控制该信号解码的I变量I915应设为4(脉冲和方向，对该信号向上计数)。

通过这样做，时基控制可被看作就好象它们来自一个外部的频率源一样。注意，缺省的转换表被设置成从这个编码器计数器来处理时间基数信息。参考连接PMAC与主机中控制面板I/O下的图。

第二步:插补

一旦解码和计数后，每个伺服周期信号的值都被输入编码器转换表，就好象一个伺服反馈信号一样。建议使用1/T转换法，因为这种方法提供了非常好的信号子脉冲插补(使用与计数器相连的计时器)，可以显著提高时基信息的平滑度。你必须确定转换表被设置成以这种方法来处理输入信号的计数器。编码器的转换表在出厂时被设置成对编码器计数器1到8进行1/T转换。欲知细节可参考编码器转换表的描述。

第三步:时基计算

一个独立的编码器转换表入口项接受由上一步得到的插补"位置"信息，并减去上一个伺服周期得到的插补"位置"信息，并把结果与一个放大因子相乘来产生伺服周期的时基值。(该时基值然后成为位置更新计算的一个因子，所以修正的大小与上一个伺服周期里从时基信号得到的步数成比例。)

这一步里的两个设置项为信息源(插补"位置"寄存器)和比例系数。它们都是编码器转换表的入口项。要了解如何输入它们的细节可参考转换表的描述。

时间基数转换的公式为:

$$\%value=(100.0*SCALE\_FACTOR*INPUT\_FREQ)/2^{17}$$

%value(也被看作进给率超调值)用来控制位置更新率—
当它等于100.0时，程序和运动将以"真实时间"(例如程序中指定的时间和速度)执行。

SCALE_FACTOR是一个整数值，必须决定它的大小以正确的完成时间基数的以下设置。INPUT_FREQ为计数率(由信号和编码器I变量0决定)，单位是步/毫秒。$2^{17}$ 为131,072

可通过"实时"输入计数频率——此时你的程序和运动以指定的速率运行—
来设置你的放大因子。因为这时%value值应为100.0，我们可简单地解出放大因子:

$$SCALE\_FACTOR=131,072/(REAL\_TIME\_INPUT\_FREQ)$$

由于放大因子必须是整数，而131,072为2的指数倍，你可能想使你的以步/毫秒为单位的实时输入频率也成为2的指数倍。例如，如果你的有一个全速输入计数频率为60,000步/秒的系统，定义你的实时输入频率为64步/毫秒。这将使你的比例系数为131,072/64=2,048。

到现在为止，我们得到的是与主频成比例的寄存器中的值。现在我们必须使用这个值来控制我们

的运动程序。

第四步:使用时基计算

时基值只在某个坐标系内起作用。每个坐标系都有一个I变量来告诉它到哪里寻找它的时基信息。对于坐标系X，该变量为Ix93。Ix93的缺省值是由软件控制的寄存器的地址，而不是由一个外部频率控制。所以对于某个坐标系，如果你希望由外部的时基控制，你必须设置Ix93为由以上所得的放大的时基的地址。例如，在缺省的转换表内，该值在地址$729(十进制的1833)处，所以如果想用这个频率来控制坐标系1，I193应设置为1833(它总是一个位于X内存的字，所以不需要指定“X”)。

一旦设置了这个I变量，对于编程和非编程的运动所有分配给这个坐标系的电机都将由外部频率来控制。

I变量Ix94控制坐标系X的时基值的最大变化率。当从主机给出时基值时(用一个%n命令)，你可能希望它低一些，以使时基向新值的转换要平稳一些。但是，如果你希望PMAC与一个外部信号如时基源保持良好的同步，该值应被设的尽可能大(最大值为8,388,607)，这样使时基的变化和信号源的变化一样快。将该值设的低一些可提高跟随的平滑度，但代价是跟随时出现滑移。如果Ix94总用于对外部时基的限制，将不能保证与主轴的位置同步。

第五步:编写程序

当你编写由外部时基控制的程序时，就好象输入信号总在“实时”频率时一样。运行时，程序将以一个与输入频率成比例的速度执行。对于你指定的运动时间和进给率你有完全的浮点分辨率。

记住，DWELL命令总是在实时下执行，而与输入频率无关。如果你想在受输入频率控制的程序中暂停，用DELAY命令，而不要用DWELL。

**时基举例**

你有一个以额定速度50英寸/秒运动着的卷筒。在卷筒上有一个分辨率为500线/英寸的编码器。你有一个在PMAC控制下的横切轴。当卷筒以额定速度运动时，你希望作一个0.75秒的切削运动，然后在2.50秒之后开始另一个运动。卷筒上的编码器与编码器2的输入线连在一起。

第一步:信号解码

由于卷筒上的编码器是编码器2，所以I905控制解码。为得到最大的分辨率，我们想设置I905为3或7，得到四倍的解码。首先试试3。在手册的M变量列表中，我们看到该编码器的“编码器位置”M变量为M201。我们对M变量进行定义，然后在卷筒将要运行的方向上调节它，不断地查询它的值(可以使用PMAC执行程序的观察窗口)。如果在调节卷筒时，该值增加，那么I905的设置是正确的。如果它减小，我们就重新设置I905为7。(如果它不变化，就需要检查连线是否正确。)

第二步:插补

接着我们来看看编码器转换表的设置。完成它的最简单的办法是通过PMAC执行程序的设置菜单。如果这不可行，我们可以使用RHY:$720,16命令，它使PMAC返回地址Y:$720到Y:$72F的内容——转换表的设置数据。我们可能看到下面这些值:

```
00C000  00C004  00C008  00C00C  00C010  00C014  00C018
00C01C  400723  000295   000000  000000   000000  000000
000000   000000
```

(这儿的值是表的缺省值。)返回的第二个值(地址Y:$721)显示编码器2使用1/T转换法，占用的寄存器为$C004到$C007(49156到49159)。它给我们提供了一个理想的另一个计数数据，可以提高平滑度。我们不需要改变这儿的任何值。但如果读出的是C0C004，我们可用命令WY:$721，$00C004来改变它。

第三步:时基计算

现在我们想设置表的入口来把插入位置转换成时间基数的格式。观察上面给出的值，我们看到第九个值(地址Y:$728)，400723，是一个时间基数转换。但它的源地址$723，是来自编码器4的插入位置，而不是我们想要的编码器2。我们用命令WY:$728，$400721来改变它。

现在我们必须计算比例系数。我们看到，额定速度为50英寸/秒，分辨率为500线/英寸，并且为4倍的解码，计算得:

50英寸/秒*500周期/英寸*4步/周期
=100，000步/秒
=100步/毫秒

由于如果该数为2的指数倍时，计算会更容易一些，我们让我们的“实时”计数率为128步/毫秒。然后计算放大因子为131,072/128=1024。我们通过命令WY:$729,1024将该值写入(注意，我们可以省略$符，以十进制的形式将该值输入)。

第四步:使用时基计算

由于在坐标系1内工作，我们把$729(十进制的1833)赋给I193来指向这个时基值。我们把I194设置为最大值(8,388,607)，这样在快速变化时也能保持同步。

第五步:编写程序

在编写程序时，我们必须工作在“实时”输入频率下，它与我们开始时的名义速度不同——在这种情况下，它要快28%。所以，所有编程中的速度都将提高28%，而所有编程中的时间都将减少28%。我们将额定切削时间750毫秒(0.75秒)乘以100/128，得到585.9375毫秒。类似的，2500毫秒也被缩小到1953.25毫秒。(如果这些数在你的程序中不能确切的给出，你可以在程序中直接计算；PMAC用48位浮点数的精度执行这种计算。)我们可有一个如下的主程序循环:

```
WHILE  (M11=1)          ; 当输入为真时切削
    TM  585.9375    ; 切削运动时间
    X100000             ; 实际切削运动
    DELAY  500          ; 暂停
    TM  1953.125    ; 返回时间
    X0                  ; 实际返回运动
    DELAY  500          ; 暂停
ENDWHILE
```

# 触发时基功能

时基技术可以使从动坐标系与主坐标系保持良好的同步，但它们不能提供一种与主坐标系上的特殊点同步的方法。因此，主电机和从动电机可能会发生“异相”，必须使用一些特殊的技术，通常涉及到从一个定位标志得到位置信息，来使它们达到同相。

许多时基的应用不需要主从动电机同相(比如，分割白纸的长度要大于被打印的页)，而其它的由于拉伸，滑移或不均匀的间距的要求，需要不断地进行再定位。这一类的应用可以使用标准的时基功能。

但是，那些需要与主电机保持“同相”的，以及那些定位操作很困难，甚至不可能的应用，可以使用转换表的触发时基特性。通过在触发之前冻结时基，然后以被触发器捕获的那个位置为参考启动时基，这种技术允许与触发器捕获的主轴位置保持良好的同步。

转换表中的触发时基的入口与标准时基的入口类似。它是一个双线入口，第一线指定主编码器数据的处理过程和源地址，第二线指定了时基比例系数。触发时基的入口与标准时基的入口有两个重要差别。首先，指定处理过程的值不同，在触发的处理过程中它被改变了($90，$A0和$B0，标准时基为$B0)。第二，源地址是实际主编码器计数器的寄存器，而不是转换表中处理过的编码器数据。比例系数与标准时基功能是相同的。如何对该入口进行操作将在转换表的指导中加以详细讨论。

**触发时基功能的指导**

使用触发时基特性包括正确的设置I变量值，M变量定义，转换表的入口(这可在之前做)，编写运动程序，以及编写PLC程序。每一部分都将在以下讨论，先做一般解释，然后举一个特定的例子。

第一步:信号解码设置。

主信号的解码与标准的时基功能一致:正交或脉冲和方向信号必须被解码以使计数器加计数。这是编码器I变量0(I900，I905，etc。)设置的。

第二步:插补和时基设置

编码器转换表中的触发时基转换同时处理通过插补值进行的1/T插补和时基的计算。在初始化设置中，转换表中生成的触发时基入口通常处于运行(而不是冻结或等待触发)状态。时基比例系数也在这

儿输入；它的计算方法与标准时基功能的计算方法是一样的。

第三步:编写运动程序

在编写使用触发时基的运动程序时，所有的轴必须停止在它们等待触发的那一点上。如果这不是运动的起始位置，那通过DWELL命令，这一部分将放在最前面的位置上。

在开始由触发器触发的运动的计算时，时基必须被“冻结”，以防止开始运动。这最好通过使用已被赋给了转换表中触发时基入口的”处理”位的M变量来完成。如果前一个运动是通过另一个时基源来完成的，那坐标系的时基地址—Ix93—应该改变为触发时基的入口。

这些运动程序中的命令应该跟在由触发器触发的第一个运动的计算和命令之后。当时基被“冻结”时，PMAC将完成所有的计算，但并不开始运动的实际执行。变量I11(计算延迟)应被设为0，这样时基功能一开始，PMAC就可以执行运动了。

第四步:准备触发器

执行运动计算的运动程序自己不能准备触发器，这样在计算未进行完之前不可能产生触发。如果是这样，程序将落在需要的同步之后。所以，为了可靠的执行，触发器应由一个在所有的运动计算已经完成后才执行的任务，通常是一个PLC程序来准备。在PLC程序中，准备触发器只需要一个简单的条件分支；它只是判断是否时基被“冻结”，如果是，PLC程序准备触发器。因为PLC程序不能中断运动程序，这就保证了它在运动程序已经完成了所有的运动计算之后发生。

第五步:开始触发

一旦触发器被准备完毕，PMAC就等待主编码器中的位置捕获触发发生。编码器/标志变量2和3决定哪一个信号的哪一个边沿导致触发。当PMAC看到触发发生，它开始时间基数功能，使用被捕获的主轴位置作为时基功能的起始点。

**触发时基举例**

电机#1是坐标系1的A轴。它是一根有2500线/转分辨率编码器的旋转轴，并且它到负载的减速比为3:1。它要从动与PMAC上的编码器4。主编码器的分辨率为4096线/转，额定转速为600rpm。在接收命令运行之前，X轴必须等待一个检索脉冲并落后它45度。在接下来主编码器的36度内，它必须加速，然后在主编码器的144度内匀速运行，最后在主编码器的36度内减速至0。该运动必须在A轴的一整转内完成。

我们将使用触发时基功能，由主编码器的检索脉冲触发。选择600rpm作为主轴的“实时”速度，我们以步/毫秒为单位计算出我们的实时输入频率(RTIF)。

600转/分钟*(分钟/60秒)*(4096线/转)*(4步/线)*(秒/1000毫秒)
=163.84步/毫秒

时基比例系数(SF)为

SF=131,072/RTIF=131,072/163.84=800

在实时速度为600rpm(10rps)的情况下，主轴每转需时100毫秒；所以主轴的45度需时12。5毫秒，依此类推。

*设置与定义*

```
I915=3                      ; 编码器4,4倍解码
                            ; 在运动方向上设为加计数
I917=1                      ; 编码器4在检索脉冲的上升沿
                            ; 触发
WY:$072A,$A0C00C,800        ; 在缺省转换表的末尾添加触发
                            ; 时基入口
                            ; $A0为触发时基

        ; 运行(登记触发器);
                            ; $C00C指向编码器4的寄存器
                            ; 比例系数为800(十进制)
M199->Y:$072A,16,8          ; 转换表入口的处理位
&1                          ; 指定坐标系1
#1->83.33333333A      ; 登记为为坐标系1的A轴;
```

```
                                    ; 3X2500X4步/转(360度/转)
```

*运动程序*

```
CLOSE
OPEN PROG 12 CLEAR
I193=$072B              ; 触发时基源地址
DWELL0                  ; 表中的时基数转换(第二行)
M199=$90                ; 冻结时基(前一行应为DWELL)
LINEAR                  ; 线性运动模式
INC                     ; 增量运动指定
TA10                    ; 主轴36度为10毫秒
TS0                     ; 无S曲线
DELAY12.5               ; 主轴45度为12.5毫秒
TM50                    ; 主轴的36+144度为50毫秒
A360                    ; 从动轴的一整转
CLOSE
```

*PLC程序*

```
CLOSE
OPEN PLC 10 CLEAR
IF (M199=$90)           ; 时基被冻结了吗?
    M199=$B0            ; 那么准备触发器
ENDIF
CLOSE
```

# 使PMAC与其它PMAC同步

当多片PMAC一起使用时，各卡的同步是通过将伺服时钟信号从第一个卡传到其它卡来保持的。仔细地编写程序，就能使不同的PMAC的轴保持完全的协调。

PMAC提供了在单个应用中使用多卡的能力。为了使这些卡正确的工作，必须考虑以下因素:

1.主机通讯地址
2.时钟同步
3.运动程序同步

主机通讯地址在与PMAC联系和编写主机通讯程序内有描述。有关同步的事项将在下面阐述。

**时钟同步**

PMAC使用一个晶振作为它基本的时钟发生器。每个PMAC都有它自己的晶振。由于这些晶

振的精度都很高(标准的为50ppm，Option 8为10ppm)，它们很难准确地保持一致。如果它们使用各自的主时钟，就会导致在处理长的运动程序时PMAC间不能保持同步。一般的，如果连续运动超过10分钟时，才能观察到这个现象。例如，在最坏的情形下，两卡的偏差达到100ppm，经过10分钟的连续运动之后，误差为60毫秒。

共享时钟信号

解决这个问题的一个办法是使所有的卡共享一个公共的时钟信号。对于PMAC，这是通过串行接口上的备用线完成的。@0卡输出它的时钟信号；所有其它的卡将其作为输入接收。真正被共享的是相频和由跳线E29-E33和E3-E6决定的，通过对主频分配得到的伺服时钟信号。当共享时钟信号时，仅有@0卡上的这些跳线设置起作用。其它板上的伺服周期参数I10应与@0卡上的跳线设置匹配。

当多个卡使用同一个串行口进行通讯时，它们必须共享一个公共的设置信号。这是因为控制软件地址的相同的跳线或开关(E40-E43，或SW1-1-SW1-4)也控制时钟信号是输入还是输出，所以仅有一个卡使用它自己的时钟信号；其它卡必须从串行口接口上接收它们。

当多个卡在同一个总线上使用独立的硬件地址进行通讯，或进行独立的操作时，是否共享同一个时钟信号是可选的。如果你希望它们共享一个公共的时钟信号，使用跳线或开关将一个卡设置为@0，将其它的以数字编号，并把所有卡上串行口的时钟信号连在一起。这些跳线设置不会影响总线通讯的协议。

如果在应用总线时，你不想共享时钟信号，每块卡都必须被设为@0。

连接

只要把PMAC上的相同管脚连在一起就可以实现共享。在总线或串行通讯应用中(当然也包括实际的串行通讯)，可以用附件3D或3L导线连接PMAC的接口(每个PMAC有一个附件3E)。在独立的或总线通讯应用中，不需要主机进行参预。对于通讯线，不可以把时钟线从PMAC-PC的RS-442口上接到PMAC-Lite的RS-232口上。如果PMAC-Lite(Opt .9L)有RS-422口，到PMAC-PC的连接是可能的，但接口的输出脚有差别。

如果没有使用串行通讯，但串行数据线连着时钟信号，可能需要关掉串行口以防止噪声产生给PMAC的输出命令字符。对于PMAC-PC，PMAC-Lite和PMAC-VME，可通过使跳线E44-E47全部为ON来实现；对于PMAC-STD，则通过把DIP开关SW1-5到SW1-8设为全OFF来实现。

外部时间基准

如果几个卡的轴都接在一个外部频率时间基准上，并希望实现同步，那所有卡的编码器计数器都应输入同一个频率信号。如果在使用外部时间基准时不需要完全的同步，那就不需把PMAC的时钟源接在一起，因为外部频率已有效地提供了公共时钟。

**运动程序同步**

无论各卡是否共享一个时钟信号，各卡运动的同步将受每个卡的运动程序中的指定时间的控制。如果一个卡被要求做3秒长的运动，而另一个做4秒长的运动，它们同时开始，显然它们不能一起结束。所以，认真编写将要在多个卡上一起运行的运动程序以使它们耗费同样的运动时间是及其重要的。

初始计算延迟

在接收到一个Run或Step命令后，在PAMC开始第一个运动之前，它需要一些初始的计算时间--通常是几个毫秒。如果几个PMAC被告知同时开始一个程序，通常它们不会耗费同样的时间

来计算它们的第一个运动。如果每个卡都在计算结束后立即开始运动，那么卡与卡之间将失去同步。PMAC的参数I11(运动程序计算延迟)就是为了解决这个问题。它决定了在接收到Run或Step命令后，和开始第一个运动之间的毫秒数。在要求同步时，对于所有的卡，I11必须被设定为相同的值；实际上，缺省值10(=10毫秒延迟)可被用于所有的功能上。(如果I11被设为0，那在计算结束后就会立即开始第一个运动。对于单卡的应用是没有问题的，但不适用于多卡的应用。)

运动的指定时间

一般的，在这些程序的运动必须指定运动时间(TM，TA和TS)，而不是进给率(F)。指定进给率的运动的时间是通过将所有进给轴(FRAX)的距离向量用进给率除得到的。保证不同卡的这种运动耗费相同的时间是很困难的。

DWELL是一个非同步命令，在编写多卡应用的运动程序时，不能用这个命令。用DELAY命令来保持程序的同步性。

无偏移条件

如果认真地编写运动程序，使用指定时间的运动，并且各卡共享公共的时钟信号，那么各卡就能够完全无偏移地运行。即使同步的命令，当它们开始它们的运动程序时，各卡之间仍存在一个不超过几个毫秒的初始偏移，但如果准确地编写程序，并共享时钟信号，该偏移值就不会增大。下面的部分解释了如何使该偏移值最小(或消失)。

最小化初始偏移

如果没有PLC程序被置为有效，被告知同时开始一个程序的PMAC卡通常在同一个伺服周期内执行程序。不能在同一个伺服周期内开始的程序将在下一个实时中断时开始。这将是在((I8+1)*伺服周期长度)微秒后。如果PLC程序被置为有效，各卡开始时的偏移可能与需要编译和运行的最长的PLC程序耗费的时间一样长。一个消除初始执行偏移的好办法如下所示:

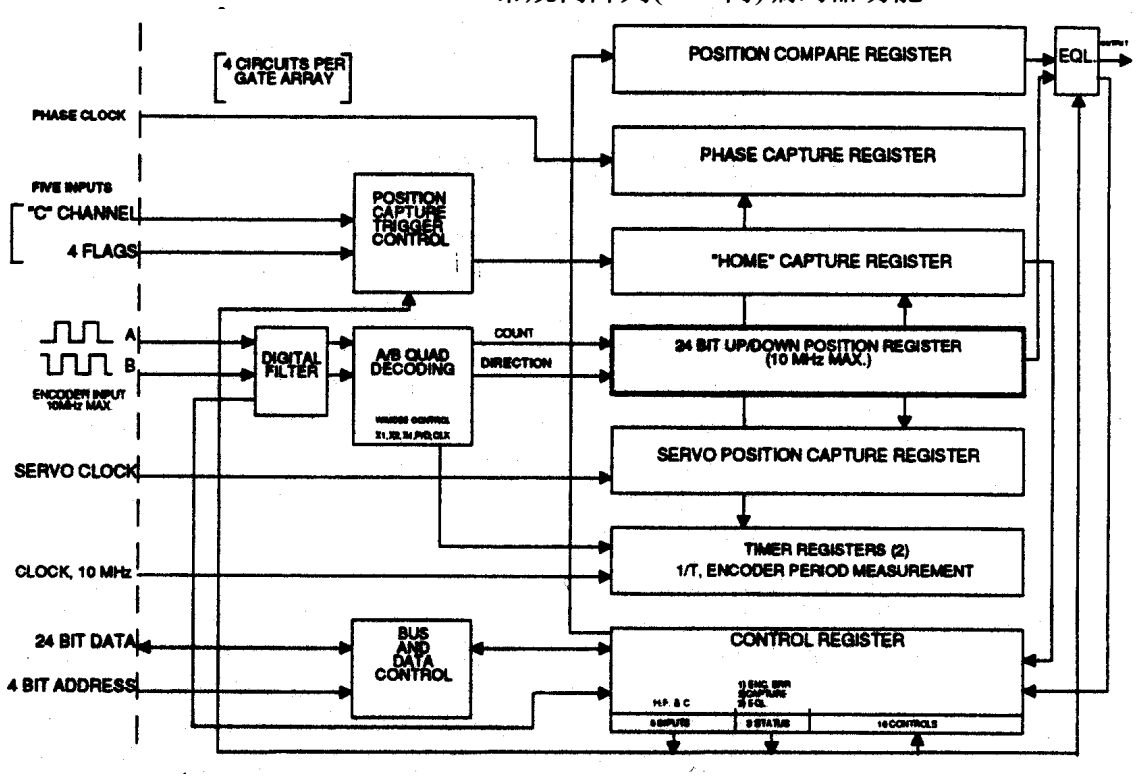
- 初始化所有PMAC卡上的所有程序计数器。当各卡上的相同程序具有相同的程序名时，只需输入

@@B1<CR>。对于复杂一些的情况可能需输入@0B1@1&3B2<CR>

- 用CTRL-D命令使所有的PLC程序无效。这将使PMAC对一个命令的响应达到最快
- 把I8设为0，这样每个伺服周期都会有一个实时中断。如果你不在运行PLC0，你可以使它一直等于0
- 在简单的情况下，如果@@已被执行，用R<CR>，或使用全局的运行命令<CTRL-F>来开始你的程序
- 如果你需要运行一个PLC0程序，把I8设回它的原始值，通常是2。让I8等于0可能会导致PLC0占用后台任务需要的处理器时间，导致通讯无法进行，甚至监控时钟出现错误。
- 使你需要的PLC出现有效。你必须在每个运动程序的第一行使每个PMAC卡的PLC程序有效。例如:

```
OPEN PROG 1 CLEAR
ENA PLC 1…31
TM 1000
```

**PMAC PC/VME 常规门阵列(DSP门)编码器功能**




# 位置捕捉功能

位置捕捉功能在一个外部事件进入一个特殊的寄存器时锁住当前的编码器位置。它完全在硬件中执行，而不需要软件的干预(虽然它是由软件来设置和服务的)。这意味着唯一的延迟是硬件门的延迟(在任何机械系统中都可忽视)，所以它提供了难以置信的精确的位置捕捉功能。

**设置触发条件**

可将位置捕捉寄存器设置成“自动”使用，即由固件内的常规例程直接处理寄存器，或“手动”使用，由用户程序处理寄存器的信息。在不考虑模式的情况下，导致位置捕捉发生的事件由编码器I变量2和3(对于编码器1为I902和I903)决定。编码器I变量2定义了编码器第三通道和触发位置捕捉(也可由软件触发)的编码器标志的转变过程的组合方式。如果它定义要使用标志，编码器I变量3决定使用哪一个标志(一般总设为0来指定归位标志)。

**回零时的使用**

当使用这项特性来归位一个电机时，电机标志地址I变量(对于电机X为Ix25)必须指向标志的正确设置(这已经通过正确地指定限位标志的地址来完成)。例如，I125的缺省值是49152($C000)，指向第一个标志。编码器标志I变量2(e.g.I902)和编码器标志I变量3(e.g.I903)导致位置捕捉操作的编码器标志的变化。一旦这些被正确地设置，回零功能将自动使用位置捕捉特性。

**用户程序中的使用**

如果你在你自己的程序中使用位置捕捉功能，这两个I变量仍控制着捕捉的操作。你可通过一个M变量存取被捕捉的位置(建议用M103作为编码器1捕捉寄存器的M变量:M103->X:$C00，17，1.
当触发条件变为真时，该位也变为真；当捕捉寄存器被读取后，它变为假(当在表达式中使用M103时)。该位一变为真，捕捉功能就被置为无效；你必须读取捕捉寄存器使捕捉功能再次有效。例子程序MOVTRIG。PMC指示在精确的定位过程中如何使用该功能。

**电机位置的偏置**

注意:位置捕捉功能得到编码器位置，而不是电机或轴的位置。编码器位置指向最近一次加电或重启时的位置，而不考虑之后执行的任何回零运动或偏置命令。为了使这个编码器位置与电机位置相联系，你必须知道编码器零点和回零位置间的偏差。幸好，PMAC为了将来的使用将回零运动中捕捉的位置存在寄存器Y:$0815(#1)，Y:$08D5(#2)，etc。

# 位置比较功能

位置比较功能实质上与位置回零功能正相反。不是在外部改变时将计数器中的位置存起来，而是当计数器到达一个指定的位置时改变一个外部信号。用这种方法，你可以在系统的某个实际位置触发一个事件。因为这种触发是一个纯粹的硬件功能(虽然用软件设置)，它是非常快和精确的。你可使用信号去触发主机，插置设备，或PMAC本身的一个动作。

**需要的M变量**

为了使用这个特性，我们必须存取位置比较寄存器，和几个状态和控制位。对于编码器1，我们定义相应M变量:

```
M103->X:$C003，0，24，S      ；24位位置比较寄存器
M111->X:$C000，11，1           ；比较标志锁存控制位
M112->X:$C000，12，1           ；比较输出使能位
M113->X:$C000，13，1           ；比较输出转换控制位
M116->X:$C000，16，1           ；比较相等标志
```

**预置比较位置**

为预置一个比较位置，只需给M103赋值，例如，M103=1250。该值在-8,388,608和+8,388,607之间。(记住你不可以读回该值；从同一个地址读出的是位置捕捉寄存器。)这个命令可以从一个PMAC的运动程序，一个PMAC的PLC程序，或主机给出。这是编码器的位置；如果你想指向电机的零点位置，你必须知道回零偏置(见上)。

**比较控制位**

共有三个控制位来设置“相等”信号的格式。标志锁存控制位(在我们的例子中为M111)控制比较相等的信号是暂时的--只在位置确实相等时为真，还是锁存的--
直到重启时一直为真。如果该控制位为0时信号是暂时的，为1时是锁存的。为清除一个锁存的标志，将位置为0，然后设回1。

该比较相等信号总被拷入在PMAC内部使用的比较相等标志(在我们的例子中为M116)。如果你在内部使用这个标志，确定这个信号是锁存的(M111=1)，否则你可能会丢失它。若要中断主机(边沿触发)，你可能要使该信号为暂时的。

输出使能信号(这儿为M112)决定比较相等信号是否在EQU线上输出(1为使能)。如果你想用该信号中断你的主机或直接触发一个外部事件，就必须设置它。输出转换位(这儿为M113)决定EQUn输出是高为真，还是低为真(为1时，低为真)。若要用于主机中断，你应设成高为真。

**用比较相等信号中断主机**

如果你使用EQUn线中断主机，你必须把它接在PMAC-PC的8259可编程中断控制器上(PIC)。为达到该目的的跳线为E54到E65，它们位于PMAC-PC板的底边上。
PIC的输出必须接在使用跳线E76到E84的一根PC中断线上。要了解详情，可以参考通讯功能的跳线表和使用PMAC-PC中断主机部分。

**直接触发外部动作**

如果你想使用EQU线触发来自PMAC-PC的一个外部动作，你应在E点(E53到E65)放置一个接口，把这些信号接在中断控制器上(一个IDC26针的接口就可很好地工作)。没有其它接口将这些信号引出。这些信号必须被缓冲；PMAC-PC的TTL驱动器的驱动能力很低。

对于PMAC-Lite，PMAC-VME和PMAC-STD，有一个JEQU接口引出比较相等输出。这些输出为开路集电极输出，额定值为24V，100mA。用户可以用源驱动IC(UDN2981A)代替已有的驱动IC。

举例

例子部分中的程序COMPPULS。PMC向你显示如果使用这一特性在一段距离上产生非常快的一系列“相等”脉冲。一旦PMAC检测到已经到达上一个比较位置，它就清除标志，载入下一个比较位置，然后进行位置计算。

**电机位置偏置**

注意:位置比较功能得到编码器位置，而不是电机或轴的位置。编码器位置指向最近一次加电或重启时的位置，而不考虑之后执行的任何回零运动或偏置命令。为了使这个编码器位置与电机位置相联系，你必须知道编码器零点和归位零点位置间的偏差。幸好，PMAC为了将来的使用将归位运动中捕捉的位置存在寄存器Y:$0815(#1)，Y:$08D5(#2)，etc。

# 同步M变量赋值

同步M变量赋值语句允许设置输出，并可在运动程序的下一个指令运动开始时进行清除。注意，这个输出是与指令位置同步的，而不一定是实际位置。由于跟随误差，这两者不一定相同。这些语句在用户手册的“计算特性”部分有详细的描述。

# §3－13 编写一个PLC程 序

## PLC程序

除了可顺序和同步执行的运动程序外—
任何在接着执行的程序行之前用以执行数量特定的运动命令——
PMAC有64个非同步执行的PLC程序(32个已编译PLC程序和32个未编译PLC程序)，它们可以以很快的速度重复执行。它们被称作PLC程序，因为它们执行很多与硬件的可编程逻辑控制器相同的功能。和运动程序一样，PLC程序有许多相同的逻辑构造，但是没有运动类型的语句。

如果未编译的PLC程序是不完全相同的，大多数已编译PLC程序很类似。实际上，在编译一个PLC程序之前，它会被作为一个未编译的PLC程序加以测试和调试。这两种类型的PLC程序的不同点在于缓冲区的控制，L变量，一些命令结构和编译器的使用。由于它们的相同点，有关未编译PLC程序的部分也适用于已编译的PLC程序。关于已编译PLC程序的信息在“已编译的PLC程序”部分可以找到。

**何时使用**

PLC程序用于执行与运动不同步的操作。如果你想进行的操作与编程的运动同步(e。g。每个运动一次)，那么使用一个运动程序来完成就可以了。

**一般用途**

PLC程序在监视模拟和数字输入，设置输出，发送信息，监视运动参数，象从主机一样执行命令，改变增益，以及开始和停止运动时是特别有用的。由于它们可完全地存取PMAC的变量和I/O及它们的非同步的性质，它们成为运动控制程序的非常有力的附件。

**32个PLC程序**

PLC的编译和未编译程序都从0到31编号。这意味着你可同时拥有存在PMAC中的一个已编译的和一个未编译的PLCn程序。PLC0是在伺服中断周期结束时以I8指定的频率(每I8+1个伺服周期)运行的一个很快的程序。该程序用于几个临界任务，它必须比较小，因为它的不断执行将占用其它任务的时间。

**警告:**
如果PLC0太大，占用了大量的后台任务的时间来执行，可能会导致无法预料的后果，甚至触发PMAC的看门狗计时器。

PLC程序1到31在时间允许的范围内在后台不断的运行，实际上是处于一个无限的循环中。它们会被电机定相，伺服环闭合，运动准备和PLC0等优先级高的任务所中断。

## 输入一个PLC程序

PLC程序的语句的输入和向PMAC中输入命令行是一样的。作为准备，使用一个"CLOSE"命令来确定没有其它被打开的缓冲区是必要的。也有必要用DELETE，GATHER和DELETE TRACE命令来确定内存没有被数据采集或程序轨迹缓冲区所占用。

**打开缓冲区**

对每个PLC程序，下一步是用OPEN PLC n命令来打开缓冲区的入口，n是缓冲区的编号。接着，如果缓冲区内有不需要保存的任何东西，用CLEAR命令来清除它（PLC的缓冲区不能被PMAC自己来编辑；它们必须被清除，然后再输入）。如果缓冲区没有被清除，新的语句将被加在缓冲区
的末尾。

**下载程序**

通常在编写程序时，编辑是在基于主机的文本编辑器上完成的，而每次新的版本被下载到PMAC上时，老的缓冲区就会被清空。当你完成时，用CLOSE关闭缓冲区。打开一个PLC程序缓冲区会自动停止程序的执行。当关闭缓冲区后，程序仍是无效的，可以用ENABLE PLC n命令来使它再次有效，n为缓冲区编号(0—31)。要执行PLC程序，还必须正确的设置I5

注意:

因为在PMAC内存里的所有PLC程序在加电/重启时都被设为有效了，所以在编辑PLC程序时最好把PMAC的变量I5设为0。这允许你重启PMAC，而且没有PLC程序在运行，当PLC程序出现错误时也更容易检查。

PLC程序的一般形式为:

```
CLOSE
DELETE GATHER
DELETE TRACE
OPEN PLC n
CLEAR
{PLC  statements}
CLOSE
ENABLE PLC n
```

**关闭缓冲区**

在关闭时，PMAC将检查是否所有的IF分支和WHILE循环都正确地结束了。如果不是，它将报错，并使缓冲区不能工作。你应在主机上改正PLC程序的错误并且再次输入它(当然必须清除程序中的错误部分)。对于你将使用的所有缓冲区都会重复这个过程。

**删除程序**

为了删除一个未编译的PLC程序，你必须打开缓冲区，清除里面的内容，然后再次关闭缓冲区。这可以通过一个命令行的3个命令来完成:

```
OPEN PLC 5 CLEAR CLOSE
```

**举例**

一个简单的PLC程序的内容如下所示:

```
OPEN PLC 2
CLEAR
IF (M11=1)
    I130=10000
ELSE
    I130=8000
ENDIF
CLOSE
ENABLE PLC 2
```

**PLC程序结构**

在编写PLC程序时必须记住的一件重要的事情是每个PLC程序实际上都在一个无限的循环里；它将不停地执行直到命令它停止。(它们被叫做PLC的原因就是因为它们与硬件的可编程逻辑控制器——按一定的顺序对一系列的操作或可能的操作重复地扫描——很类似。)

**计算语句**

PLC执行的大部分动作都是通过变量的赋值语句来完成的:{变量}={表达式}。变量的类型可以是I，P，Q或M。这些动作能够影响到PMAC卡内部和外部的许多事情。

象手册一开始提到的那样，最简单的PLC程序是由一行组成的:

```
P1=P1+1
```

每次PLC程序执行，通常每秒几百次，P1将增加1。

当然，这些语句可以更复杂些。语句:

```
P2=M162/(I108*32*10000)*COS(M262/(I208*32*100))
```

能把半径(M162)和角度(M262)位置转换成水平位置数据，并同时放大。由于它更新的非常快，无论谁需要存取该信息(例如 主机，运算符，运动程序)，都能保证得到的是当前的数据。

**条件语句**

PLC程序的大部分动作是依靠PMAC变量状态，如输入，输出，计数器，等的条件语句。你可能希望通过电平或边沿触发；这些都可以做到，但使用的方法不同。

电平触发条件:

一个由电平触发条件控制的分支是很容易实现的。让一个输入变量M11来控制变量的增量，我们可用:

```
IF (M11=1)
    P1=P1+1
ENDIF
```

如果输入为真，P1将每秒增量几百次，当输入变为假时，P1将停止增加。

边沿触发条件:

假设你只希望在每次M11变为真的时候才给P1增量一次(M11的上升沿触发也叫做”一次触发”或“锁定”)。要这样做，可能会复杂一些。我们需要一个复合条件来触发动作，作为条件的一部分，我们设置其中一个条件为假，这样在下一个PIC扫描时该动作就不会发生。这样做最简单的办法就是使用一个“影子变量”，它将跟随输入变量值的变化。只有在影子变量与输入变量不匹配时动作才会发生。我们的代码变为:

```
IF (M11=1)
    IF (P11=0)
        P1=P1+1
        P11=1
ENDIF
  ELSE
    P11=0
  ENDIF
```

注意，我们必须保证P11能随M11的变化而变化。在电平触发模式下我们把P11设为0；我们也可以使用边沿触发，就我们关心的程序最后输出来说，这是没有影响的，它与计算时间有关，并且它节省了程序行。

任何SEND，COMMAND或DISPLAY 语句仅仅在一个边沿触发条件中才能执行，因为PLC

程序的循环要比这些操作处理它们的数据要快，并且如果在PLC的串行扫描下执行这些程序，通讯通道可能会无法工作。在第九节中有如何使用这些语句的更多的例子。

```
IF (M11=1)                  ; 输入为ON
    IF (P11=0)              ; 上一次输入不是ON
        COMMAND “#1J+”      ; 微动电机
        P11=1               ; 设置P11
    ENDIF
ELSE
    P11=0                   ; 重置P11
ENDIF
```

WHILE循环:

一般的一个PLC程序在一次扫描中从头到尾执行一遍。但如果程序中有一个为真的WHILE条件时就不是这样了。在这种情况下，该程序将执行到ENDWHILE语句并退出这个PLC程序。当执行完所有其它的PLC程序后，它将在WHILE条件语句处再次进入这个PLC程序，而不是在一开始。只要条件为真，该过程将重复进行下去。当WHILE条件变为假时，PLC程序将跳过到ENDWHILE部分的语句，而继续执行余下的PLC程序。

如果你想只要输入为真就给我们的计数器增量，并不执行余下的PLC程序，可以这样做:

```
WHILE (M11=1)
    P1=P1+1
ENDWHILE
```

这种结构使得在PLC程序的某一部分“截断”PLC的操作变得更容易，当该条件为真时，程序中的其它分支不需要额外的条件就能不加以执行。将其与IF条件做比较(见上)。

一些COMMAND语句后面必须跟着一个WHILE条件来确保它们在执行程序余下的部分之前就产生了作用。如果后面跟着的第二个COMMAND语句需要第一个COMMAND语句结束后才能执行时，该条件总是为真的。(记住，COMMAND语句仅在后台周期的通讯部分才能进行处理。)假设你希望用一个输入来停止一个坐标系内的所有运动并开始运动程序10。可以使用以下的PLC程序:

```
IF (M11=1)                    ; 输入为ON
    IF (P11=0)                ; 上一次输入不是ON
        P11=1                 ; 设置P11
        COMMAND “&1A”         ; ABORT所有运动
        WHILE (M187=0)        ; 等待运动停止
        ENDW
        COMMAND“&1B10R”       ; 开始程序10
    ENDIF
ELSE
    P11=0                     ; 重置P11
ENDIF
```

注意，M187为例程序1中定义的坐标系In-Position位。

精确定时

由于PLC1到31在PMAC中的运算优先级最低，循环时间不能被精确地决定。如果你希望在一段非常精确的时间内拦截一个操作时该怎样做？你仍旧可使用WHILE循环，但不是为变量增值，你要使用一个板上的计时器。PMAC有四个你可写入的24位计时器，而且可以每个伺服周期向下计数。这些计时器分别是寄存器X:$0700，Y:$0700，X:$0701和Y:$0701。通常可以把一个有符号的M变量赋给计时器，以伺服周期为单位，把你需要的时间写入这个变量(将毫秒数乘以8，388，608/I10)；然后PLC将等待直到M变量小于0。M70定义为M70->X:$0700，24，S:

```
M70=P1*8388608/I10        ; 将计时器设为500msec
WHILE (M70>0)             ; 循环直至计数到零
ENDWHILE                  ; 当条件为真时退出PLC程序
```

如果你需要更多的计时器，最好使用内存地址为X:0的寄存器。这个24位的寄存器每个伺服周期向上计数。我们将为它存入一个初始值，然后在每次扫描中从当前值中减去这个初始值并把它与我们要等待的时间做比较。通过把结果存入另一个24位的寄存器，我们可以很轻松地处理X:0的翻转。

首先，用在线命令定义以下的M变量:

```
M0->X:$0，24          ; 伺服计时器寄存器
M85->X:$07F0，24      ; 24位寄存器
M86->X:$07F1，24      ; 24位寄存器
```

然后，写下以下的PLC程序:

```
M85=M0                ; 计时器的初始值
M86=M0-M85            ; 已经过的时间
WHILE (M86<P86)       ; 小于指定的时间?
    M86=M0-M85        ; 已经过的时间
ENDWHILE              ; 当条件为真时退出PLC程序
```

## 编译PLC程序

通过编译PMAC

PLC程序可以使它运行得更快。编译之后的PLC程序运行加快主要有两个因素:首先，节省了解释的时间。第二，来自于编译后的PLC程序具有处理整数计算的能力。编译的PLC程序的浮点运算要比解释的PLC程序快2到3倍；整数(包括布尔运算)要快20到30倍。

PMAC不能自己执行PLC程序的编译。编译工作是通过PC完成的；然后把机器码下载到PMAC上。

PMAC可存储和执行32个编译的PLC程序和32个解释(未编译)的PLC程序，共64个PLC程序。PMAC内存中的15K24位字是为编译的PLC程序保留的；如果有用户编写的伺服程序时则为14K。其它的任务不能使用这些内存，而编译的PLC程序也不会使用其它的内存。

注意:
这儿提到的编译代码的大小指的是实际的编译代码在PMAC的内存占用的空间。而不是编译器的输出文件在PC磁盘驱动器中的大小。

在PMAC中一个编译的PLC程序是由PLCC n（PLC Compiled #n）来编号的。它与以PLC n编号的解释的PLC程序不同。以同样数字编号的解释和编译的PLC程序间没有什么特殊的关系。

**编译PLC程序的执行**

在32个编译的PLC程序中(PLCC0到PLCC31)只有PLCC0可由实时中断(RTI)触发，在前台执行。PLCC1到PLCC31都是后台的任务。

在每个实时中断时，PMAC将检查是否有多个用户程序需要运行。实时中断每(I8+1)个伺服周期发生一次。PMAC按下面的顺序进行检查:

1) 运动程序运动准备:PMAC检查在每个坐标系中是否需要对程序的下一步进行计算。

2) 解释PLC0:PMAC检查是否I5=1或3及PLC0是否被置为有效。如果是这样，它将对PLC0进行一遍扫描。

3) 编译PLC0:PMAC检查是否I5=1或3及PLCC0是否被置为有效。如果是这样，它将对PLCC0进行一遍扫描。

保证PLCC0和PLC0的扫描执行时间少于一次实时中断周期是非常重要的。否则，它们的重复执行将占用后台时间，并有可能触发看门狗计时器。

在后台，PMAC对一个后台的解释PLC程序进行扫描时不会被任何其它任务中断(但高优先级的任务将产生中断)。在对每个独立的后台解释PLC程序的每次扫描之间，PMAC将对后台所有激活的编译PLC程序进行一遍扫描。这意味着后台编译PLC程序将比后台解释PLC程序以更高的扫描率执行。例如，如果有7个后台解释PLC程序，那么对其进行了一遍扫描时，每个后台编译PLC程序已经被扫描了7遍。

## 准备编译PLC程序

编译PLC程序有多个步骤。基本的步骤如下所示:

1.) 以解释的形式编写并调试PLC程序。
2.) 将所有要被编译的PLC程序的编号从PLC n改为PLCC n。
3.) 对于整数运算，定义L变量，并用它们代替程序中的那些老的变量名。
4.) 将所有要编译的PLC程序在PC上组合成一个文件，用“宏”名代替PMAC中的代码。
5.) 打开执行程序的”下载时编译”的特性，使用下载功能把文件下载到PMAC上。
6.) 执行编译的PLC程序。如果结果不正确，回到1或2。

以上的每一步都将在下面详细地加以讨论。

**1.) 初步调试**

建议以解释的形式完成PLC程序代码的大部分调试。由于使用PMAC执行程序和PMAC使重复编辑，下载和执行变得很简便，这样很快也很容易就能进行多次试验。一旦在解释的形式下已基本完成了调试，要把它变成编译的形式就很容易了。

**2.) 改变PLC程序的编号**

所有需要编译的程序的编号都要从PLC变成PLCC。每个要编译的PLC程序必须以一个OPEN PLCC n命令开头。这是无论告诉编辑器以下的部分—直到CLOSE命令—将要被编译。所有指向要编译的PLC程序的ENABLE PLC和DISABLE PLC命令都需被变成ENABLE PLCC和DISABLE PLCC，而无论这些命令是否在要编译的PLC程序内。

### 3.) 执行整数运算

编译的PLC程序有以24位符号整数的形式运行算术和逻辑运算的能力。而非编译的PLC程序则以48位浮点数的形式进行所有的算术和逻辑运算，即使是对整数变量进行操作。短整数的数学计算至少要比浮点的运算快10倍。加上不需要编译节省下的2-3倍时间，它的运行速度要比未编译时的浮点运算快了20-30倍。

一个编译的PLC程序可以用整形数进行一些语句的计算，而用浮点数进行其它语句的计算。但是，编译的PLC中一个给定的语句要么完全用整形数计算，要么用浮点数计算(即使它用整形的寄存器工作)。

使用L变量

在PLC程序的语句中使用L变量可通知编译器将要执行的语句用整形数而不是浮点数进行运算。

生成L变量

为了在编译的PLC程序中实现用整形数运算，用户必须定义任何被使用的L变量，并用它们代替那些在解释的形式下程序中使用的变量(通常是M变量)。编译器将把编译PLC程序中那些仅包含L变量(必须正确定义)和整形常量的语句解释为用整形数进行运算的操作。

L变量的定义同X或Y格式的M变量一样，但它们仅存在于编译器内。PMAC不能辨别L变量或L变量的定义，而且PMAC将拒绝发送给它的任何未编译的包含L变量的命令。所有L变量的定义必须放在要编译的PLC文件的最前面。

对于编辑器的合法的L变量定义由字母L和跟在它后面的一个整数组成，整数的范围为0-1023。总共有1024个可能的L变量(L0到L1023)。

你可以把所有的L变量定义放在一个独立的文件里，然后在PLC的主文件里用#include语句来表示引用这个文件(例如 #include lvardef。pmc)。当你在解释模式下调试程序时，可用分号使这一行成为注释。记住，如果你使用一个独立的编辑器，在编辑器运行之前，该定义文件必须和主文件放在一个文件内。当编辑器作为CNC执行程序或PMAC执行程序的一部分运行时，它能自动把一个主文件和主文件内用“#include”引用的所有文件链接起来。

由于变量指向PMAC内存和I/O空间的固定部分，L变量可以简单地替换M变量，也可以象定义M变量那样定义L变量。而且，同时保留M保留的定义也是完全可以的。你可能希望保留M变量的定义以进行调试，因为PMAC不接受对一个L变量的值或定义进行查询的命令。

举例来说，JOPTO口上的机器输出1和机器输入1在未编译的程序中通常使用以下的定义:

```
M1->Y:$FFC2，8          ；机器输出1
M11->Y:$FFC2，0      ；机器输入1
```

对于编译的PLC程序，你可定义相同的L变量:

```
L1->Y:$FFC2，8          ；机器输出1
L11->Y:$FFC2，0         ；机器输入1
```

一个使机器输出1跟随机器输入1变化的编译的PLC小程序为:

```
IF (L11=1)
    L1=1

ELSE
    L1=0
ENDIF
```

在程序的语句中用一个L变量存取一个寄存器，以及在另一个程序中用一个整形的M或I变量存取

同一个寄存器，甚至寄存器的同一位，是完全合法的。

混合L变量和P，Q变量对一个P或Q变量寄存器的存取将会导致无意义的结果，因为P和Q变量总是把寄存器当作一个浮点数来存取。

对于一般用途的整形变量，L变量可以替代程序中的P变量的作用。因为L变量必须被定义为一个特定的地址，所以在PMAC的内存中找到一块打开的区域来保存这些变量是很重要的。为了使这变得容易点，你可以在PMAC中用DEFINE UBUFFER {size}命令指定一个用户缓冲区，在这里，{size}代表PMAC内存中为该缓冲区保留的48位字的数目。缓冲区从地址$9FFF开始，向下延伸直到包含指定的字的数目。例如，命令DEFINE UBUFFER 256为用户使用保留，包括L变量，从$9F00到$9FFFF的地址，包括X和Y寄存器。

如果PMAC中已有用一个DEFINE命令(旋转程序，螺旋，等。)生成的任何缓冲区，那么就不允许你生成一个用户缓冲区。你会发现在用一个$$$***命令重新初始化卡后立刻生成用户缓冲区是很容易的。

你可能希望用作为一般用途变量分配给用户缓冲区寄存器的整形M变量对这些程序进行调试，而不是P变量。这会使你对整形的编译更加容易。

如果你只有几个一般用途的L变量，你可以使用打开的内存区$0770到$077F，和$07F0到$07FF(包括X和Y寄存器)。

记住，因为L变量的定义仅在编译的时候才使用，所以使用通过M变量在运行时改变定义的数组检索技术是不可能的。

在哪儿使用

L变量可在编译的PLC程序中的两种类型的语句中使用:变量值分配语句和条件语句(IF，WHILE，AND，OR)。

变量值赋值语句

PMAC用变量值赋值语句向任何寄存器写值，或是一个输出，内存中的一个一般用途变量，为一个特殊的用途使用的内存寄存器，例如一个增益，或一个硬件寄存器，例如一个标志控制寄存器。许多这样的语句可以用整形运算来执行。

*有效值*

在一个给定的变量值分配语句({变量}={表达式})中，L变量不能与其它任何变量组合在一起；这就是说，如果在语句里有L变量，那么等号两边的所有变量都必须是L变量。语句中的任何常量都必须是-8，388，608到+8，388，608(-$2^{23}$到$2^{23}$-1)之间的整形数。

在编译的PLC程序中，任何包含一个L变量的变量值分配语句，如果同时还包含其它类型的变量，一个非整形的常量，或一个超出-$2^{23}$到$2^{23}$-1范围的整形常量，将会被编译器拒绝。编译器将会提示错误和行号。

*合法运算符*

所有能够在未编译的PLC程序中用于浮点运算的数学运算符(+，-，*，/%)和按位的布尔运算符(&，|，^)也能用于编译的PLC程序的整形运算。这些运算符的优先级和浮点运算是相同的。

*整数的除运算*

整数的结果被向最接近的整数取整，而不是象PC的Intel 80x86那样被截断。如果小数部分正好为0。5，它会被取成下一个更大的整数(例如 -7.5到-7，7.5到8)。在PMAC的浮点运算时，所有的中间值都是浮点形式；如果最后的结果被存进一个整数寄存器，例如一个I或M变量，以上的取整规则也是适用的。

下表阐明了除法取整规则的效果:

| 平台 | 语句 | 结果 |
|---|---|---|
| PC | x=10*2/3 | 6 |
| PMAC | L1=10*2/3 | 7 |
| PMAC | M1=10*2/3 | 7 |
| PC | x=10*(2/3) | 0 |

| | | |
|---|---|---|
| PMAC | L1=10*(2/3) | 10 |
| PMAC | M1=10*(2/3) | 7 |

*位取反*

逻辑的“NOT”运算可通过把变量与一个所有位为1的常量进行^(异或)操作来完成。一个单个位L变量可通过与1异或来转换(例如，L1=L1^1)。一个24位变量的所有位可通过与$FFFFFF异或来转换(例如，L356=L355^$FFFFFF)。

*无函数*

在整数值赋值语句中使用函数是不合法的(例如，L1=SIN(L2))。

*中间值*

所有这些整形计算的中间值是24位的有符号值。用户必须确定计算中没有中间值越出这个范围。例如，语句L500=1000000*2000000/4000000将不能被正确执行，因为头两个数的结果超出了24位的长度。

变量值赋值语句中计算的结果必须写给一个L变量。该值可能小于24位的宽度。对于一个N位长的L变量来说，仅有计算值的低N位被写进寄存器；其它位被舍去了。注意，这N位不一定是寄存器的最低位。

举例来说，对于定义L102->Y:$C003，8，16，S(DAC1寄存器)，使用了24位字的高16位。语句L102=1000自动从Y:$C003的第八位写入值。

举例

合法的整形语句的例子:

```
L100=1
L0=L1-L2*L3+(L4/1024)
L5=L5%1000
L6=L1^$FF
```

非法的整形语句的例子:

```
L10=P10
P10=L10

L11=L11+P11
L13=16777216/L12
L253=L14*ATAN(L16)
```

条件语句

在一个条件语句中，任何仅包含L变量和在$-2^{23}$到$2^{23}-1$范围内的整形常量的简单条件({表达式}{比较符}{表达式})都使用较快的整数运算来处理。在编译的PLC程序中任何包含L变量和其它类型的变量，一个非整形的常量，或一个超出$-2^{23}$到$2^{23}-1$范围的整形常量的简单条件将会被编译器拒绝。编译器将会返回错误和行号。

整形的简单条件总是把两个24位的有符号数互相进行比较，并且在进行条件运算时所有的中间值都会被当作24位的有符号数保存，所以所有的值都必须被限制在$-2^{23}$到$2^{23}-1$之内。

整形变量值赋值语句中允许的数学计算在整形条件语句中也是允许的。以及所有的规则和速度/内存基准(见下)都是适用的。

由被逻辑运算符AND或OR分隔开的简单条件组合成的复合条件可以同时包含使用整形运算的简单条件和使用浮点运算的简单条件。

所有在PMAC浮点程序中可使用的比较符，除了~(约等于)和!~(非约等于)外，在编译PLC使用整形的条件中也可使用。这些合法的比较符是=，!=，>，!>，<和!<。

举例

合法的条件语句的例子:

```
IF (L50=1)
WHILE (L75<L76)
IF (L1&L2 | (L4+L5) > 0 AND P1=Q2 OR L6!=0) OR (L3&L5=$11 AND P1=P2 OR L1=0)
```

非法的条件语句的例子:

```
IF (L50=P1)
WHILE (L75<10000000)
IF (L1+P2=0)
```

对速度和内存的优化

24位有符号值的L变量的读写速度是最快的。没有偏置的无符号的1到20位的变量是第二快的，而有符号的1到20位的变量和那些在0位上有偏置的是最慢的。运算速度越慢，PLC占用的程序内存越多。但是，24位的L变量占用的数据内存要比宽度小的占用的多。由于在大多数编译PLC程序的应用中，速度要比数据内存更加受到注意，通常所有的不需要指向某个字的特定位的变量(例如在用户缓冲区内的所有“一般用途”的L变量)都是24位宽的，即使它们并不需要。

对有符号的24位的L变量的一个读或写操作需要3个DSP的指令周期来执行，和2个程序内存单元来储存。

对一个小于24位(0到20位)的有符号的L变量的读操作需要6到8个DSP指令周期和5到7个程序内存单元。对一个小于24位(0到20位)的无符号的L变量的读操作需要7到9个DSP

指令周期和6到8个程序内存单元。

对一个小于24位(0到20位)的有符号或无符号的L变量的写操作需要12到14个DSP指令周期和10到12个程序内存单元。

&，^，|，+，-，*运算需要1或2个指令周期。

除法(/)运算名义上需要82个指令周期。

取模(%)运算名义上需要76个指令周期。

**4.) 集成PLC文件**

在使用独立的编辑器(PLCC。EXE)之前，所有需要编译的PLC程序必须被组合成一个文件。如果主文件使用#include命令包括所有其它的文件，CNC执行程序和新的PMAC执行程序V3,x可以使用多个文件进行编译。

独立的编译器只能对一个文件进行编译，假设它包括了所有需要编译的PLC程序。在文件中保留一些PLC程序和其它命令不进行编译是可以的。编译器会跳过这些部分而不做改变。

独立的编译器不能使用#define宏指令和PMAC执行程序编辑器和下载例程使用的#include文件。对PMAC代码的替换必须在编译前完成。程序PREPROC。EXE可用来完成这些替换，并把多个文件集成为单个的文件。

如果你没有预处理器程序PREPROC。EXE，或编译器程序PLCC。EXE(见下)，它们可以通过Delta Tau’s BBS得到，电话号码为(818)407-4859。

要执行预处理器程序，在DOS提示符下键入PREPROC -F{filename}，这里，{filename}是主文件的全名，包括扩展名。该程序产生一个与输入文件同名的输出文件，但扩展名为。SRC。它同时还生成一个包含宏名和PMAC代码的相互对照的同名的。MAP文件。

链接地址文件

在包含交叉编译器的同一个子目录下还有一个名叫LISTLINK。TXT的文件，编译器在运行时会参考它。该文件包含了PMAC在LIST
LINK命令下返回的ASCII码。它包含了编译的PLC程序代码与之链接的关键子例程的PMAC地址。对于不同版本的PMAC固件该文件是不相同的。PMAC固件的任何改变，甚至是子版本的改变都要求新的LISTLINK。TXT文件，重新编译PLC程序，重新下载编译的代码。

**5.) 运行编译器**

在DOS提示符下键入PLCC -F{文件名}可执行独立的编译器，在这里，{文件名}是输入文件的全名，包括扩展名。编译器将开始执行直到编译完成或发现错误时为止。

在PMAC执行程序中，当“下载时编译”选项被激活后，编译器就和下载程序被集成在一起。

如果编译器能成功编译整个程序，它将生成一个包含PMAC机器代码的输出文件，可以直接被下载到PMAC上。它的文件名与输入文件的文件名是一样的，但它的扩展名是。56K。在屏幕上你可以看到以下信息:

```
***  PMAC PLC Compile V1.2 04/28/94  ***
***  PLC compile complete，no errors  ***
***  Downloadable PLCC code in file MYPLCC.56K  ***
***  PLCC program memory use: 7343 of 15360 words  ***
```

如果编译器发现了错误，将不生成输出文件。编译器停在发现的第一个错误处。在屏幕上你可以看到以下信息:

```
***  PMAC PLC Compile V1.2 04/28/94  ***
***  Compile aborted due to error  ***
***  Compile error #: 35  ***
***  PLC source file line: 27  ***
```

编译错误

编译器可以发现以下错误

| 编号 | 错误类型 |
|---|---|
| 33 | 无法装入浮点数 |
| 34 | 无法把字符串转换成浮点数 |
| 35 | 非法的字符串命令或格式 |
| 36 | 整数值溢出 |
| 37 | 括号不匹配 |
| 38 | 非法的ELSE命令 |
| 39 | 非法的ENDIF命令 |
| 40 | 非法的ENDWHILE命令 |
| 41 | PLCC输出文件错 |
| 42 | PLCC输入文件错 |
| 43 | 不闭合的IF或WHILE命令 |
| 44 | 超出PLCC最大内存 |
| 45 | 超出PLCC转换堆栈 |
| 46 | 第一次符号传送超出最大值 |
| 47 | 远程堆分配错 |
| 48 | 字符串长度必须<255个字符 |
| 49 | L变量地址未定义 |
| 50 | 两个L变量有相同地址或已被定义 |

编译处理

编译器按照以下规则解释输入文件:

1. 注释——从一个分号(；)到行尾的所有字符--被忽略，并且它们不会被传至输出文件。

2. 编译器接受并使用L命令定义，但不把它们传至输出文件。

3. 编译器试图把OPEN PLCC n和CLOSE之间，不包括注释的所有语句编译成DSP机器代码。在OPEN PLCC n命令之后不需要CLEAR命令，因为在下载新的PLC编译程序时自动删除已有的版本。但是，也没有必要删除编译器的CLEAR命令。

4. 对于每个编译的PLC n，编译器会自动在输出文件里加上OPEN PLC n CLEAR CLOSE命令。当发送给PMAC时，它自动删除该编号的未编译的PLC程序。这样做的原因是为了防止同一PLC程序的编译和未编译版本在PMAC中同时执行。

如果你想得到一个与编译的PLC程序同名，但内容不同的未编译的PLC程序，你可把它包含在输入文件内，在需要编译的PLC程序之后送给编译器。编译器将把它传送至输出文件(见下)。之后每次它都会被下载至PMAC上。否则你需要在每次编译/下载周期之后再为它作另一次下载。

5. 所有其它命令将被编译器无变化地传送至输出文件。

编译器将告诉你编译代码占用的PMAC内存；如果编译代码占用的内存超出了最大的允许值

15360个字，它会报错。如果在PMAC中你有一个用户编写的伺服程序，那么编译代码不可以超出14336个字。

**6.) 下载编译代码**

由编译器得到的单个输出文件可被任何能把文件传送给PMAC的程序或例程下载到PMAC上，例如PMAC执行程序的"下载文件到PMAC"菜单项。在CNC执行程序中，每次编译后会自动下载。

下载任何新的编译PLC程序时会自动删除PMAC激活内存中所有的编译PLC程序。不需要其它的命令来这样做。

在有电池备份内存的PMAC中，编译PLC程序在断电时会被电池自动保存起来，直到它们被DELETE PLCC命令完全删除掉，或下载了一个新的PLC编译程序。

在有后备闪存的PMAC中，下载后必须使用SAVE命令在断电或重启时保存PLC编译程序。SAVE命令把程序从内存中拷到稳定的闪存中。在加电/重启时，闪存的内容被拷入内存。

**7.) 运行PLC编译程序**

I变量I5可同时控制未编译和编译的PLC程序。PLC程序可分别地被ENABLE PLC，ENABLE PLCC，DISABLE PLC和DISABLE PLCC命令设为有效或无效。它们可由在线命令，或在运动程序，未编译的PLC程序或PLC编译程序中作为缓冲区命令给出。一个PLC程序还可以把它自己置为无效。唯一的限制是你在PLC0或PLCC0内不可以使用DISABLE PLCC命令；在这里它们不能保证会正常工作。

你可以把I5的两位看作屋内的主电闸，而把独立的程序使能控制看作屋内的电灯开关。只有电闸和电灯开关都被设为ON时电灯才能工作。电闸和电灯开关可以独立控制。

当PLC编译程序被下载到PMAC上时，它们全都被设为有效。如果I5允许，它们可立刻开始运行。

在加电/重启时，所有已有的PLC程序，编译或未编译的，都被设为有效。如果这时I5的值允许一个给定的PLC程序运行，在加电后它就可以执行了。PLC1将是第一个在加电/重启后执行的PLC程序(甚至在PLCC之前)。许多人把它作为"重置PLC程序"，执行一次后把自己置为无效以防止再次执行。通过使用DISABLE PLC和DISABLE PLCC命令这个程序可用来防止在加电后其它PLC程序立即执行。用这个办法，你可以在启动时选择你想要的PLC程序来执行。

<CONTROL-D>是使所有的PLC程序-编译或解释的——无效的快速方法。

| 注意:<br>最好不要在启动时就运行PLC0和PLCC0程序。所以，你不应把I5存为1或3。相反，把I5存为2；然后在你的PLC1"重置PLC程序"中使用下面的命令: |
|---|

```
DISABLE PLCC 0
DISABLE PLC 0
I5=3
```

PLC0和/或PLCC0可在需要时被置为有效。

# §3－14 编写一个主机通讯程序

如果你希望在实际应用中让PMAC与主机进行通讯，你就需要编写一个主机通讯程序。你在开发中使用的PMAC执行程序并不是作为一个实际应用的主机程序来使用的；它仅仅被设计成为一个开发工具。

从根本上说，你编写的主机通讯程序向PMAC发送ASCII码的字符串或从PMAC接收字符串。你应该编写一些底层的例程来发送和接收单独的字符和文本行；在指定你要读写的不同的文本串之后，它们会被反复地调用。

在用户手册的一开始，与PMAC通讯的部分阐述了通讯的基本概念。在学习这一章之前，你应复习那一部分。

与PMAC的通讯可以通过PMAC的三个通讯口之一来完成(每个版本的PMAC有两个端口)。每种PMAC都有一个可以与主机的“COM”口连接的串行口。PMAC-PC，-Lite和-STD有一个与主机进行总线通讯的“主机口”。该口在所有三个版本上的应用是一样的。PMAC-VME有一个与主机通过VME总线进行通讯的“邮箱寄存器”。每个通讯口的使用方法都将在以下的章节加以讨论。

注意:Delta Tau提供了许多软件库，这使得主机通讯程序的开发变得更加容易。最值得注意的是，附件9P，“PCOMM”库可处理所有在这一部分描述的底层操作。如果你使用这些通讯库，那么在这一部分的许多讨论是不需要的。

这个部分包括了通讯的文本命令和文本响应。使用Option 2双口RAM和在主机及PMAC上编写的程序，可实现完全的二进制通讯。使用这种技术的方法完全由用户决定。

## 查询和基于中断的通讯

不考虑使用的通讯口，完成字符的传送有两种基本的方法。第一种方法，主机可不断地查询PMAC，看看它是否已做好传送或接收下一个字符或文本行的准备。第二种方法，在PMAC准备好发送或接收下一个字符或文本行时中断主机。

查询通讯方式易于编写程序，但效率低，因为程序的时间花在等待PMAC准备上了。基于中断的通讯更有效率，因为当PMAC未准备好时主机可以处理其它事情，但它编写程序较困难。你必须初始化向量并打开中断，编写中断服务程序，决定一个程序哪一些其它中断被屏蔽，而且在结束时你必须恢复原来的中断。

可参考附录内的程序PMACPOLL。C和PMACINT。C。

## 串行口通讯

当通过PMAC的串行口与PMAC进行通讯时，通常你应使用主机的一个COM口。对于IBM-PC及其兼容机，它们通常是内装的COM1和COM2 的RS-232口，但也可能在扩展卡上。大多数COM口，甚至对于非IBM兼容机，使用相同的集成电路，所以它们通常在主机内有相同的寄存器。

**设置界面**

每次启动系统时，主机的串行口必须通过PMAC的设置才能与PMAC正确的连接。这通常是通过向主机的I/O地址写字节的简单命令来完成的。

基址

首先你必须知道主机I/O中COM口的基址。在IBM-PC中，COM1口的基址为03F8H(十进制为1016)，COM2口为02F8H(十进制为760)。

波特率

你必须设置主机的波特率计数器来与由主频和跳线E40-E43决定的PMAC的波特率匹配。波特率计数器的值应设为115,200除以波特率(e.g.若波特率为9600，该

值为115，200/9600=12)。下面的程序段可完成这一工作:

```
outportb(combase+3,131);          /*将COM口设为设置模式*/
baud_count=115200/baud;           /*计算计数器值*/
outportb(combase，baud_count);        /*向计数器写低字节*/
outportb(combase+1，baud_count);    /*向计数器写高字节*/
outportb(combase+3,3);              /*将COM口设为普通模式*/
                                    /*共8位，有一个停止位*/
```

命令outportb是一个向端口写字节的命令；combase为基址；baud为波特率，单位为位/秒。

在初始设置时计算一个与波特率和主机速度有关的"时间限制"值是一个好办法。当主机查询PMAC是否准备好通讯时，给一个计数器增量；如果计数器超出了"时间限制"值，主机将会放弃与PMAC通讯。由设置决定它或在不久后再试一次(当等待某些不同步的通讯时还是),假设出现了错误。计算时间限制值的公式一般为:

```
timeout=7*speed*(baud_count+100);
```

对于PC-XT机，速度值为1或2，286为3或4，386为5或6，486为7或8。

发送一个字符

在查询通讯时，主机必须等待串行口寄存器的两个状态位("写准备"位)变为1后才能向串行口输出写一个字符。这两位是{基址+5}的第5位和{基址+6}的第4位。完成这个功能的C语言程序为:

```
i=0;                                        /*计数器置为0*/
while (i++<timeout && (inportb(combase+5)&32)==0);   /*循环直到该位为1*/
while (i++<timeout && (inportb(combase+6)&16)==0);   /*循环直到该位为1*/
if (i<timeout outportb(combase，outchar);            /*送字符，除非超出时间限制*/
```

送一整行只需重复调用该程序，而每次的输出字符不同。

读入一个字符

要从串行口读入一个字符，主机需要把端口设为读(对于读一整行也可这样)，然后查询串行口寄存器的状态位。当它变为1时，就可读入字符。完成这个功能的C语言程序为:

```
I=0;                                        /*计数器置为0*/
outportb(combase+4，2);                     /*将端口设为读*/
while (i++<timeout && (inportb(combase+5)==0);   /*循环直到该位为1*/
if (I<timeout ) inchar=inportb(combase);         /*读字符,除非超出时间限制*/
disable();                                  /*关中断*/
outportb(combase+4,0);                      /*将端口设为写*/
enable();                                   /*再次开中断*/
```

你可以通过这一个程序完成一整行的读入，只要在行首和行尾转换端口状态即可。

# 主机端口总线通讯（ＰＣ或ＳＴＤ总线）

## 主机端口结构

PC(ISA)和STD总线通讯使用的与PMAC连接的主机端口占用了16个地址区域的11个连续地址(它不是内存地址表)。对于主机，这些寄存器可以通过读写命令存取，例如outportb，inportb，outp和intp。对于PMAC，作为主机命令的响应，它只管理对这些寄存器的直接存取。

基址选择

这11个在主机I/O空间内的寄存器的定位是由PMAC-PC和PMAC-Lite的跳线E91-E92，E66-E71或PMAC-STD的跳线W11到W22的设置来选择的。

实际的设置信息可参考手册的跳线描述部分。这些寄存器的地址范围为基址到基址+10。

寄存器功能

每个寄存器在主机通讯中都有它自己的功能，尽管只有几个是频繁使用的，而另一些根本不使用

。这些寄存器的功能是:

| | |
|---|---|
| 基址+0: | 中断控制寄存器 |
| 基址+1: | 命令向量寄存器 |
| 基址+2: | 中断状态寄存器 |
| 基址+3: | 中断向量寄存器 |
| 基址+4: | (未使用) |
| 基址+5: | 送出或接收的高位数据 |
| 基址+6: | 送出或接收的中间位数据 |
| 基址+7: | 送出或接收的低位数据 |
| 基址+8: | 中断控制命令字0 |
| 基址+9: | 中断控制命令字1 |
| 基址+10: | 中断确认字 |

简单查询通讯的寄存器

仅使用这些地址中的两个就可完成基本的查询通讯。{基址+7}保存送往或来自PMAC的每个字节。读写寄存器是独立的，所以当你在相反方向发送时无需担心会发生覆盖。

{基址+2}保存通讯时的状态位；尽管它被叫做中断状态寄存器，它也能在与主机进行查询通讯时使用。当PMAC准备好由PC向它写入一个字符时，写准备位(第1位)为1，当PMAC准备好由PC读出一个字符时，读准备位(第0位)为1。

设置端口

尽管建议向高位寄存器和中间位寄存器内写入0以清除它们，其实主机的端口是不需要什么真正的设置的。这可由下面的C程序完成:

```
outportb (combase+5，0);          /*清除高位寄存器*/
outportb (combase+6，0);          /*清除中间位寄存器*/
```

在初始设置时计算一个与主机速度有关的“时间限制”值是必要的。当主机查询PMAC是否准备好通讯时，给一个计数器增量；如果计数器超出了“时间限制”值，主机将会放弃与PMAC通讯。由设置决定它在不久后再试一次(当等待某些不同步的通讯时)，还是假设出现了错误，由设置决定。计算时间限制值的公式一般为:

```
timeout=7*speed*100;
```

对于PC-XT机，速度值为1或2，286为3或4，386为5或6，486为7或8。

发送一个字符

为发送一个字符，主机仅需要等待写准备位变为1，然后向输出口写出字符。这可由下面的C语言程序完成:

```
i=0;                                          /*把计数器置为0*/
while (i++<timeout && !(inportb(combase+2) & 2)); /*循环直到该位为1*/
if (i<timeout) outportb(combase+7，outchar);      /*写字符*/
```

读取一个字符

为读取一个字符，主机等待读准备位变为1，然后从输入口读入字符。这可由下面的C语言程序完成:

```
i=0;                                          /*把计数器置为0*/
while (i++<timeout && !(inportb(combase+2) & 1)); /*循环直到该位为1*/
if (i<timeout) inchar=inportb(combase+7);         /*读字符*/
```

**使用PMAC-PC/STD中断主机**

PMAC-PC，PMAC-Lite和PMAC-STD运动控制卡可有许多原因来中断主机。这个功能可给用户的系统提供额外的速度，权限和复杂性，但在PC中正确地使用中断需要编写好的程序。它要求用户有大量的编程经验和一定的耐心。一旦做到这些，它可以极大地提高程序的效率。

这些PMAC有一个Intel 8259可编程控制集成电路(PIC)。该集成电路有8个输入，可以给PC送出中断信号。通过硬件和软件的组合，用户可以选择哪个信号导致PC的中断。

可以使用的信号

这8个给PIC的输入分别编号为IR0到IR7。IR0的优先级最高，IR7最低。PMAC可以给这些输入提供多种不同的信号；对于某些输入，用户可以通过跳线选择什么信号被送给输入。

下表显示了在PMAC-PC和PMAC-Lite中与每个输入匹配的信号。那些标有*号的信号在PMAC-Lite中是不可用的:

| PLC输入 | PMAC信号 |
|---|---|
| IR0 | IPOS |
| IR1 | BREQ |
| IR2 | EROR |
| IR3 | F1ER |
| IR4 | HREQ |
| IR5 | EQU1(thru E65) |
| | EQU5(thru E64)* |
| | AXEXP1(thru E63) |
| | MI1(thru E62) |
| IR6 | EQU2(thru E61) |
| | EQU6(thru E60)* |
| | AXEXP0(thru E59) |
| | MI2(thru E58) |
| IR7 | EQU3(thru E57) |
| | EQU7(thru E56)* |
| | EQU4(thru E55) |
| | EQU8(thru E54)* |

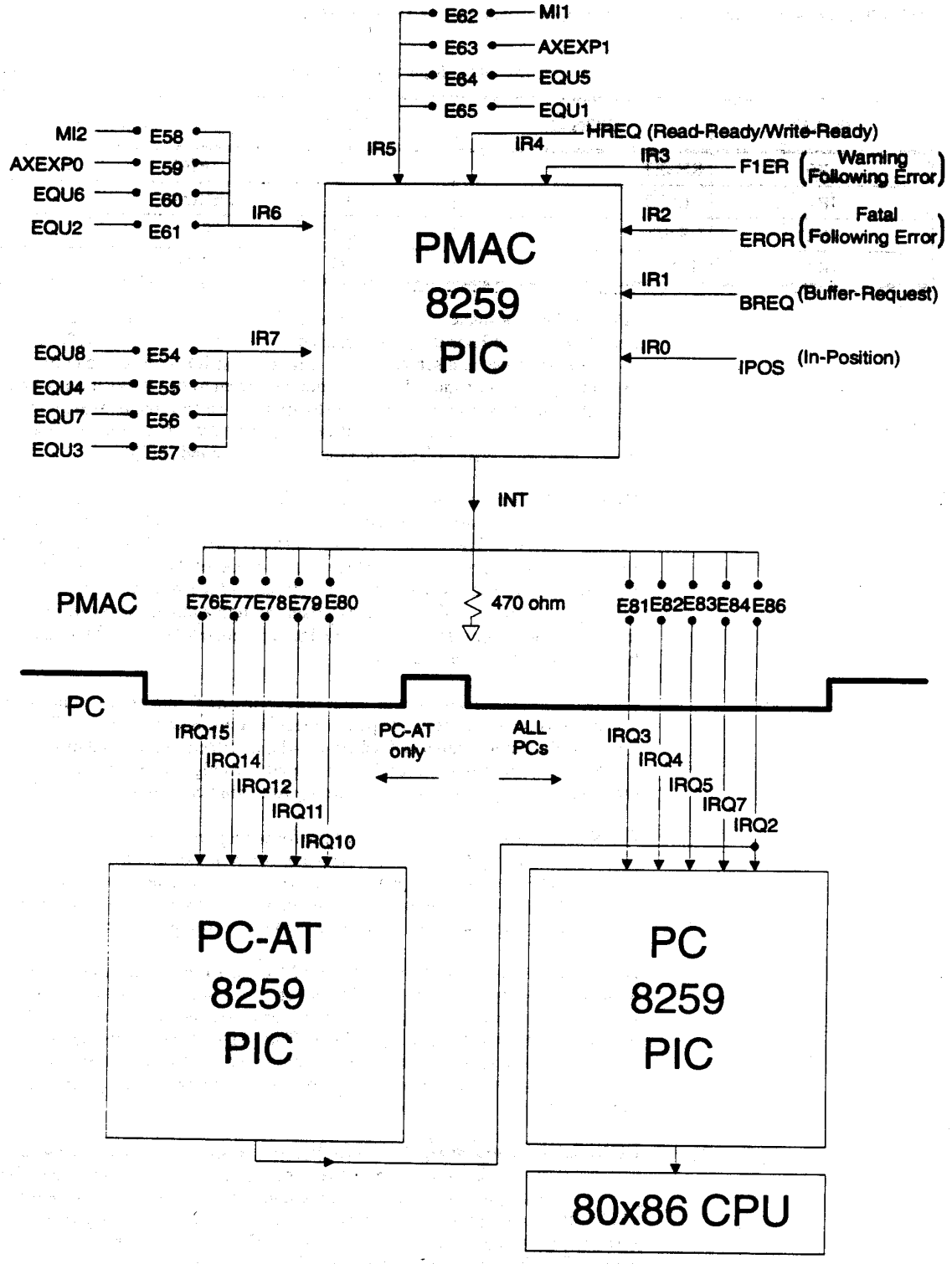
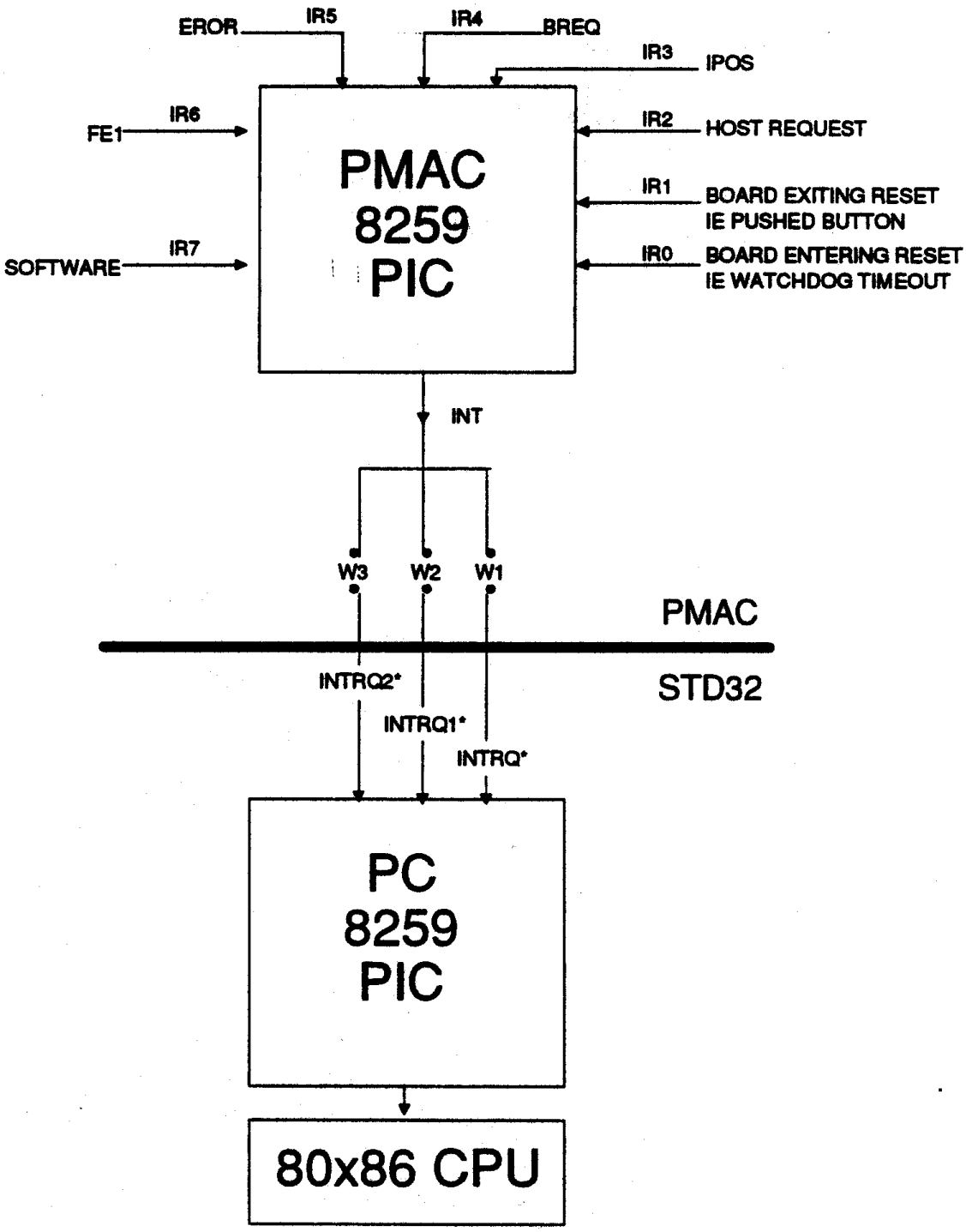



*Figure 8-19. PMAC-PC/PMAC-Lite Interrupt Structure*

下表显示了在PMAC-STD中与每个输入匹配的信号:

| 输入 | 信号 | 输入 | 信号 |
|---|---|---|---|
| IR0 | RESET | IR4 | BREQ |
| IR1 | RESET/ | IR5 | EROR |
| IR2 | HREQ | IR6 | FE1 |
| IR3 | IPOS | IR7 | (Software) |

PMAC中断结构




**IPOS**是坐标系的“到达位置”信号。如果控制面板有效(I2=0)，它反映了面板选择的坐标系(通过FDPn/线)。如果控制面板无效(I2=1)，它反映了主机指定的坐标系(通过&n命令)。若要使一个坐标系处在“到达位置”状态，必须满足三个条件：所有轴的期望速度必须为0；没有激活任何运动计时器（任何运动，DWELL或DELAY命令）；所有电机的跟随误差小于到达位置范围(Ix28)。

**BREQ**是“缓冲区请求”信号。如果一个常规缓冲区仍有空间输入更多行，它为高(你可用I18定义剩下多少内存时意味着还有足够空间)。当一个旋转缓冲区被打开时，当你在当前执行行之前载入的行数小于一个预先指定的数目时(由I16或I17设置)它为高。当缓冲区被关闭时它为低。当你把一行程序输入一个打开的缓冲区时，它总被设为低，而当以上的条件满足时它再次变为高。上升沿可作为一个中断，通知主机发送下一程序行。

**EROR**是坐标系“致命跟随误差”线。
如果控制面板有效(I2=0)，它反映了面板选择的坐标系(通过FDPn/线)。如果控制面板无效(I2=1)，它反映了主机指定的坐标系(通过&n命令)。如果坐标系内的任何电机超出了Ix11指定的致命跟随误差限，这个信号变为高。

**FIER**是坐标系“警告跟随误差”线。
如果控制面板有效(I2=0)，它反映了面板选择的坐标系(通过FDPn/线)。如果控制面板无效(I2=1)，它反映了主机指定的坐标系(通过&n命令)。
如果坐标系内的任何电机超出了Ix12指定的警告跟随误差极限，这个信号变为高。

**HREQ**是PMAC处理器的“主机请求”线。该线可用来与PMAC进行单字符的跳通讯。该线可作为“读准备”和/或“写准备”，这由PC写往PMAC的基址寄存器，DSP的中断控制寄存器的字节的值来决定(0意味着都不产生，1意味着“主机读准备”产生一个中断，2意味着“主机写准备”产生一个中断，3意味着都产生一个中断)。

**EQUn**是PMAC编码器n的“比较相等”位。当编码器的位置与预先载入的位置比较寄存器的值匹配时它变为真。你可在一个运动程序或PLC程序中通过一个M变量来使用该位(通过用“EQU输出转换有效”位来改变它的极性。该位为DSPGATE状态/控制字的第13位，M变量定义为Mx13)，这样就允许PMAC的软件产生一个PC中断。

**AXEXP0**和 **AXEXP1**从PMAC的Accesory-24轴扩展板给EQUn线输入。附件24上的跳线控制它给哪一根EQU线输入。这也可通过PMAC的软件来设置。

**MI1**和**MI2**是PMAC的机器输入1和2，通常来自PMAC控制下的系统。
(**软件:**)在把一个M变量分配给Y寄存器$FFED的第7位后(例如M10->Y:$FFED，7，1)，PMAC-STD的这条线可通过PMAC程序来触发。将变量设为1时触发中断；将它设为0时清除它。

*选择主机中断线(PMAC-PC或-Lite)*
用户应使用PMAC-PC或PMAC-Lite的跳线E76到E84和E86中的一个(仅用一个)来选择哪一根PC的中断输入线(IRQn0接收PMAC的PIC产生的信号。跳线E81到E84和E86连接PC-XT接口的IRQ线；跳线E76到E80连接PC-AT接口的IRQ线，它们对于PC-XT以及相同的机型是不可用的。

这些线被送入PC的一个8259PIC；每条都是一个独立的输入(“中断源”)。通过编程使PC辨别这个特定的中断源(该中断源必须是“未屏蔽”和“定向”的)并作出正确的反应(通过一个中断服务程序)。这将在以下加以解释。

*选择主机中断线(PMAC-STD)*
用户应使用PMAC-STD的CPU板上的跳线W1，W2和W3中的一个(仅用一个)来分别地在STD总线的中断线INTRQ*，INTRQ1*和INTRQ2*中选择。

中断功能
这些中断的最普遍的用途是用于基本的通讯。通过输入PIC的HREQ线，PMAC可在它需要给主机发送信息和(或)做好准备接收字符的任何时刻中断主机。这就不需要主机不停地查询PMAC是否做好通讯的准备。可参考附录或执行查询盘上的PC演示程序。

在PMACINT.C中要特别注意设置(清除屏蔽和正确地给PC中断向量)，中断服务程序和在程序末尾旧的中断设置的恢复。

注意: PMAC也可与PC通过串行数据口(PC上的RS-232)进行基于中断的通讯。这种中断能力是PC的RS-232口固有的，并不依赖于PMAC的PIC，但对PC来说，它对于这种中断的处理很类似于总线通讯中断。我们为那些对这种技术感兴趣的用户提供了文件。

中断还可用于其它用途。BREQ线的使用允许在程序行下载时进行由中断驱动的逐行通讯。这在与旋转程序缓冲区进行快速实时通讯时是很有用的。IPOS线可通知主机系统已到达正确位置并可以执行其它的动作。FIER线能通知主机某根轴不能很好地跟随并需要采取补救措施。

EQUn线通知主机轴的实际位置已到达一个指定点，因而可执行适当的动作。

另外，在程序运行时用一个PMAC运动程序或PLC程序中断主机是最好的办法。

设置

中断结构的正确设置，包括硬件和软件，对于正确地执行中断是至关重要的。某些任务在应用中只需执行一次；而其它的在系统每次加电时都需要执行。

*找到一个可用的中断线*

需要做的第一件事情是在PC上选择一条可用的中断线。重要的是要保证在PC使用该中断与PMAC进行工作时它不被用于其它的用途。对于所有的PC，PMAC可使用IRQ2，IRQ3，IRQ4，IRQ5和IRQ7中断，而对于PC-
AT机，还可使用IRQ10，IRQ11，IRQ12，IRQ14和IRQ15。不同版本的“兼容机”使用不同的中断以实现不同的目的；哪种功能使用了哪个中断是机型来决定的。下表是这些中断线的“标准”用法:

| 中断线 | 中断号 | PC使用 | PC-AT使用 |
|---|---|---|---|
| IRQ2 | 0AH | LPT2 | IRQ8-15 |
| IRQ3 | 0BH | COM2 | COM2 |
| IRQ4 | 0CH | COM1 | COM1 |
| IRQ5 | 0DH | 硬盘 | LPT2 |
| IRQ7 | 0DH | LPT1 | LPT1 |
| IRQ10 | 72H | xxx | 可用 |
| IRQ11 | 73H | xxx | 可用* |
| IRQ12 | 74H | xxx | 可用* |
| IRQ14 | 76H | xxx | 硬盘 |
| IRQ15 | 77H | xxx | 可用 |

* -- IBM作为未实现的功能保留

如果某个COM或LPT口没有使用，可以“借用”它的中断。

*硬件考虑*

一旦选定了中断线，就必须考虑它的电气特性。对于“兼容机”通常并不是完全兼容的。某些不需要底板上的上拉电阻，而那些使用上拉电阻的又使用不同的值。PMAC-PC出厂时带有一个470

欧姆的下拉电阻(连接D3和R25，就在AT接口上)。对于大多数PC机它足够产生一个适当的“低”状态，而对于那些有低值上拉电阻的PC，需要一个更低的下拉电阻来得到一个正确的低状态。显然地，需要对中断线进行测试以验证正确的操作--对此，扩展卡可提供很大的帮助。

*初始化PC的PIC*

PC的8259PIC必须通过软件设置(每次运行PMAC应用程序时)来对PMAC的中断作出正确响应。该设置由两个部分组成：向量和清除屏蔽。向量告诉PC当它接收到中断时从哪里执行(要执行的指令的地址)。清除屏蔽使中断打开。TurbC2.0为这个步骤提供了有效的工具，并在这作为例子使用；当然还有很多其它的方法。

*向量*

第一步通常是保存旧的向量，这样在退出程序时就可以恢复它。如果你是借用了中断，这样做是很重要的。在TurboC中，可使用“getvect”函数:

```
oldvect=getvect(0x0d);
```

该语句在(长)变量“oldvect”中保存了中断号0d--IRQ5--的已有中断向量。

下一步是置你自己的中断向量。TurboC允许你用“setvect”函数来这样做:

```
setvect(0x0d，pmac_comm);
```

用这个函数的好处是你无需指定中断向量例程的地址(事先你可能并不知道)，而只需程序名--此例中为"pmac_comm"。

清除屏蔽将在PMAC设置完毕后进行--我们会在以后讨论它。

*设置主机请求函数*

为设置PMAC，首先向DSP的中断控制寄存器内写一个值。该寄存器在PC口地址的PMAC基址上。写入的值将决定哪个按字符的通讯中断被使用。参考以上的HREQ信号部分，找到正确的值。Turbo C命令为:

```
outportb(base，value)；
```

(类似的Microsoft C命令为“outp(base，value)”。)上述命令，或其它语言中的类似命令，也可用来执行以下的步骤。

*初始化PMAC的PIC*

接着，向PMAC PIC的初始化命令字(ICWs)写值来正确地设置PIC。虽然该IC在PMAC上，它仍然被作为两个寄存器映射入PC的口地址空间，地址为PMAC的基址+8和+9。

执行以下步骤:

1. 向[PMAC基址+8]写入17H，设置了ICW1。为边沿触发中断。
2. 向[PMAC基址+9]写入08H，设置了ICW2。
3. 向[PMAC基址+9]写入03H，设置了ICW4。为8086模式。
4. 向[PMAC基址+9]写入FFH。这是一个屏蔽所有进入PMAC PIC的8个中断的运行控制字。

*清除中断屏蔽*

当你做好准备从PMAC接收中断时，清除你需要的进入PMAC PIC的中断屏蔽。写出一个1字节的自变量，其中1代表被屏蔽的中断；0代表未屏蔽的中断。例如，要只清除IR4的中断屏蔽，自变量应为EFH；要只清除IR5的中断屏蔽，自变量应为DFH。程序把该自变量写入PMAC PIC的运行控制字1(OCW1)。不需要驱动，用以下命令:

```
outportb(base+9，0xef);              /*仅清除IR4的屏蔽*/
```

你可以清除你使用的PC PIC上的中断。首先，关闭PC的中断(TurboC命令:disable()；)。接着，读入I/O口地址21H的当前屏蔽字:

```
ch=inportb(0x21);
```

然后清除你需要使用的新中断的屏蔽，这可通过对当前屏蔽字和你想使你的中断有效的屏蔽字-若是IRQ4，为EFH，是IRQ3，为FEH-执行一个位与(AND)的操作。C语言命令为:

```
ch=ch & 0xef;
```

把得出的新的屏蔽字写回I/O口地址21H:

```
outportb(0x21，ch);
```

最后，重新打开PC的中断(TurboC命令:enable()；)。完成所有设置步骤。

使用中断

在实际使用中响应中断，你需要编写一个中断服务例程。TurboC有一个特殊类型的例程可以使之变得容易些；你可以指定它为一个中断例程。该例程的第一行为:

```
static void interrupt far pmac_comm(void)
```

在你的语句中可能不需要那么多:第一个“void”指出该程序没有返回值；“interrupt”指定它为一个中断服务程序；“far”表明地址(setvect中指定)为一个远指针；“pmac_comm”是程序名(由setvect调用)；第二个“void”表示没有自变量(也不可能有)。

中断例程应首先关闭PC的中断(使之有足够的时间完成所有必要的操作)。然后应向PC

PIC(I/O口地址20H)写入一个中断结束字节。该字节的高四位为6，低四位为IRQ号。例如，如果使用IRQ4，TurboC语句为:

```
outportb(0x20，0x64)；
```

尽可能早地再次打开PC的中断。什么时候可以这样做取决于你的程序结构。

现在你也必须向PMAC PIC写入一个中断结束字节。与上一个相同(高四位为6，低四位为硬件输入(IR)线号)。对于PMAC PIC，该字节被写入PC的I/O口地址中的卡的基址+8。这可通过一个outportb命令完成。

当然，在这里你需要放入在接收到中断时要采取的动作，它可以仅仅是读取一个字符，发送一个命令行，或找到一个错误条件。

恢复原先的向量

在你退出应用程序前，应再次关闭PC中断，恢复旧的向量，并关闭PMAC PIC的中断:

```
setvect(0x0c，oldvect)；          /*恢复旧的IRQ4向量*/
disable()；                       /*关闭PC中断*/

ch=inportb(0x21)；            /*获得PC屏蔽字*/
ch=ch|0x10；                  /*使第4位为1*/
outportb(0x21，ch)；          /*送出新的PC屏蔽字*/
enable()；                    /*再次打开PC中断*/
outportb(base+9，0xff)；      /*关闭所有的PMAC中断*/
```

# VME总线通讯

**设置PMAC-VME的基址**

PMAC-VME通过16个8位的邮箱寄存器--实际上是16个字节的双口寄存器(不要与Option 2V:8Kx16双端口RAM混淆)--与作为一个从属设备的VME总线进行通讯。邮箱寄存器占用VME总线的一个地址空间，从基址+1开始。

在用VME总线与PMAC-VME通讯之前，必须首先设置邮箱寄存器的基址。必须决定在VME系统中那些内存空间是可用的，这样PMAC-VME就不会与其它已有的设备冲突。在选择了地址位置后，下一步要做的是告诉PMAC它将驻留的VME总线地址。

这不是通过跳线设置，而是通过软件编程来完成的。这样需要一台运行PMAC执行程序(或一些合适的终端软件)的IBM-PC兼容机，与PMAC通过RS232/422口通讯(使用一根RS232/422导线把你计算机的COM口与PMAC在前遮光板上的J4接口连接起来)。

在PMAC的内存里，有10个寄存器或内存区域包含了VME总线的基址信息，地址变址和“忽略位”的信息，中断等级和向量号，和使用双端口RAM的VME总线的基址(如果使用)。这个内存区域从PMAC的X列内存X:$0783开始，一直到X:$078C。

你需要使用PMAC写内存命令(W)，或使用PMAC执行软件内的VME总线地址设置菜单项通过RS232/422口向这些寄存器中写值，这些值由地址和为PMAC-VME选择的工作方式决定。下表描述了需写入值的那些寄存器。

| PMAC内存地址 | 缺省值 | 描述 |
|---|---|---|
| X:$0783 | $39 | 地址变址(AM)(见表X.2) |
| X:$0784 | $04 | 地址变址“忽略位”。(在这个例子里，地址变址第2位是一个“忽略位”，允许AM=$29或AM=$2D。) |
| X:$0785 | $FF | PMAC基址位A31-A24(当使用A16或A24地址总线时不使用) |
| X:$0786 | $7F | PMAC基址位A23-A16(当使用A16地址总线时不使用) |
| X:$0787 | $A0 | PMAC基址位A15- |

| | | |
|---|---|---|
| | | A08。可用任何为奇数位宽度的地址总线选择基址(例如地址位A08必须为0) |
| X:$0788 | $02 | 中断等级 |
| X:$0789 | $A1 | 中断向量号，为向量对的上一个编号(在这个例子里，向量对为A0-A1)。 |
| X:$078A | $00 | 双口RAM起始地址位A23-A20。 |
| X:$078B | $60 | 有DPRAM时用$E0<br>没有DPRAM时用$60 |
| X:$078C | $10 | 地址总线宽度选择:<br>$30选择A16地址总线*<br>$10选择A24地址总线*<br>$00选择A32地址总线*<br>*当使用DPRAM时给这些值加上$80。 |

PMAC-VME设置寄存器缺省值

在举例之前，先介绍一下上述寄存器的细节。

*地址变址*

地址变址(AM)是每次读写操作时由主机(或主设备)发送到总线上的一个5位代码。它告诉辅助设备(VME总线上与主设备连接的卡)使用哪种类型的地址，是短的(16位)，标准的(24位)还是扩展的(32位)地址(要获得更多信息可查阅有关VME总线的手册。你必须告诉PMAC(辅助设备)使用哪种AM。该寄存器的缺省值为$39，指定标准的24位地址。有很多可能的地址变址，但PMAC-VME通常只使用三种:

| 地址变址 | 功能 |
|---|---|
| $29 | A16 16位地址 |
| $39 | A24 24位地址 |
| $09 | A32 32位地址 |

*地址变址“忽略位”*

该寄存器(X:$0784)仅指定地址变址的哪一位为“忽略位”。换句话说，它指出忽略AM的哪一位。缺省值$04告诉PMAC忽略AM的第2位(值为4)。举例来说，它会把$39和$3D都当作合法的24位AM。这个缺省值不需要改变。

*PMAC基址位*

PMAC-VME的基址被分做3个值，用3个寄存器(X:$0785-$0787)来保存它们。第一个寄存器保存地址位A31-A24，第二个保存地址位A23-A16，最后一个保存位A15-A8(A8位必须为0)。基址的A7-A0位不指定，可都被置为0。如果你以十六进制写出基址，你可很容易地计算出哪个地址位为A31-A24，等。可参考例2。0。

*中断等级*

当PMAC-VME确认接收到一个命令(合法或非法的)或/和主机(或主设备)要读入数据，它通常在VME总线上产生一个中断。该寄存器告诉我们PMAC使用哪一个中断等级(从1到7)。缺省值为$02，即中断等级为2。所以，当主机检测到一个等级为2的中断，即是PMAC产生的。注意，在PMAC收回它的中断申请之前，中断必须被主机使用或确认。

*中断向量号*

当PMAC-VME产生一个中断后，主机也会读入一个与中断一起送来的一个中断向量。当每个中断产生时，PMAC会发送2个中断向量中的1个，指出一个特定的条件。

缺省值为$A1，意味着PMAC每次产生中断时将发送$A0或$A1中断向量。但什么时候发送$A0向量，什么时候又发送$A1向量呢？我们知道每次PMAC有数据发送时它就在VME总线上产生一个中断，不管它是一个字符(例如一个换行<LF>符)还是多个字符(例如电机的当前位置)。在这个例子中(使用缺省值)，每次PMAC接收到从邮箱寄存器送来的一套字符时，不管它是命令行的一部分还是整个命令行，就会送出一个$A0向量。这是PMAC确认它接收到一个命令行，例如:I100=1<CR>或#1J+<CR>，的方法。如果有数据等待主机读入，就会送出一个$A1向量。例如，如果你要求PMAC显示一个P变量的值

或列出PMAC内存中的一个程序，PMAC将产生一个与$A1一同送出的中断，指出在它的邮箱寄存器内有数据需要读出(下面给出了一个从邮箱寄存器中写入和读出数据的例子)。

*双端口RAM基址*

如果使用双端口RAM，它的基址与邮箱寄存器的基址是不同的。该寄存器(X:$078A)中为双端口RAM地址位的A23到A20。前4位指定了基址的A23到A20。后4位为0。例如，如果DPRAM的基址为$780000，那么该寄存器将被设为$70，该字节的高4位“7”代表基址的23-20位为‘7’。

如果你使用32 A32，DPRAM的地址位A24-A31与邮箱寄存器的基址是相同的，由X:$0785指定。

DPRAM的A19-A14位基址必须由主机在每次系统加电或重启时指定。主机通过向地址为邮箱寄存器基址+$121的邮箱寄存器IC写入一字节来这样做。该字节的低6位代表DPRAM基址的A19-A14位。

例如，邮箱寄存器的基址为$7FA000，DPRAM的基址为$780000，DPRAM基址的19到14位可由二进制的100000，或十六进制的$20表示，所以主机将向总线地址$7FA121写入$20。

(使用VME控制器上的IC可允许DPRAM各“页”间的动态“切换”。PMAC-VME上的DPRAM IC仅有一“页”，但该页可通过“切换”不停地进行动态选择。)

*DPRAM有效*

该寄存器(X:$078B)打开或关闭DPRAM。如果你想使用DPRAM，向该寄存器写入$60；如果你不想使用，写入$E0。

*地址宽度总线*

该寄存器(X:$078C)指出你使用的地址总线的宽度，以及是否安装了DPRAM。从下表中选择适当的值。

| 总线宽度 | 无DPRAM | 有DPRAM |
|---|---|---|
| 16位 | $30 | $B0 |
| 24位 | $10 | $90 |
| 32位 | $00 | $80 |

*保存设置值*

一旦完成了这些设置，必须使用SAVE命令把它们存入稳定的内存中。然后必须重启PMAC($$$命令)，因为PMAC仅在加电/重启时才将这些值拷入工作寄存器。

例子

假设为PMAC-VME选择了基址$7FA000。地址宽度为24位，不使用DPRAM。同时你选择AM为$39，忽略位的值为$04，中断等级为2，使用中断向量$A0和$A1(这些都是出厂时的缺省值)。唯一要做的就是把基址分成4部分。最好把地址用二进制重写，并标出地址位，从最右边的位开始。

| 地址位 | A31 | A24 | A23 | A16 | A15 | A8 | A7 | A0 |
|---|---|---|---|---|---|---|---|---|
| 二进制 | 0000 | 0000 | 0111 | 1111 | 1010 | 0000 | 0000 | 0000 |
| 十六进制 | 00 | | 7F | | A0 | | 00 | |

很明显,地址位A31到A24的值为$00，A23到A16的值为$7F，A15到A8的值为$A0，A7到A0的值为$00。我们所需做的就是用PMAC执行软件或一个合适的终端通讯程序把这些值写入适当的PMAC寄存器:

| PMAC地址 | 值 |
|---|---|
| X:$0783 | $39 |
| X:$0784 | $04 |
| X:$0785 | $00 |
| X:$0786 | $7F |
| X:$0787 | $A0 |
| X:$0788 | $02 |
| X:$0789 | $A1 |

| | |
|---|---|
| X:$078A | $00 |
| X:$078B | $60 |
| X:$078C | $10 |

在一个简单的写命令后加上一个SAVE命令可把这些值放入寄存器内并保存它们:

```
WX$0783,$39,$4,$0,$7F,$A0,$02,$A1,$0,$60,$10
SAVE
```

在这些新值起作用之前，记住保存这些值，然后重启PMAC(用$$$命令，使INIT/输入线变低，或重新上电)。如果你使用PMAC执行程序，通过配置/VME通讯菜单你可以更容易地设置这些寄存器。到目前为止,已做好准备通过VME总线与PMAC-VME进行通讯了!

**设置VME双端口RAM(Option 2V)**

如果在PMAC-VME上安装了Option 2V，那么需要配置它的起始地址。该设置完全可以通过软件完成，与PMAC-VME的邮箱寄存器的设置过程很类似。这里不需要进行硬件设置或PMAC-VME，PMAC2-VME上2V DPRAM的连接.它们都已经安装好了。

起始地址

注意:大多数用户只要使用PMAC总线程序中的配置/VME通讯窗口就可为DPRAM设置地址。以下部分解释了在没有执行程序时如何完成这一设置。

在选择DPRAM起始地址之前，你必须决定在你的VME系统中哪一块内存空间是可用的(这样PMAC的DPRAM就不会与VME总线上的已有RAM或其它设备冲突)。就象设置PMAC-VME的基址那样，DPRAM的起始地址是通过软件设置的，但方法有一些不同。描述如何设置DPRAM的最好方法就是给出一个例子。

举例

假设你已选择了DPRAM的起始地址为$1FC000。就象在设置PMAC-VME的基址那样，最好把地址用二进制重写，并标出地址位，从最右边的位开始。

| 地址位 | A31 A24 | | A23 A16 | | A15 A8 | | A7 A0 | |
|---|---|---|---|---|---|---|---|---|
| 二进制 | 0000 | 0000 | 0001 | 1111 | 1100 | 0000 | 0000 | 0000 |
| 十六进制 | 00 | | 1F | | C0 | | 00 | |

很清楚，地址位A31到A24的值为$00，A23到A16的值为$7F，A15到A8的值为$A0，A7到A0的值为$00。要告诉PMAC希望DPRAM从哪儿开始，需要把这个起始地址分成两个部分:

1. 第一部分代表从A23到A20的地址位。
2. 第二部分代表从A19到A14的地址位。

注意:如果你使用A32，32位地址，双口DPRAM的地址位A24-A31与邮箱寄存器的基址是相同的，由X:$0785指定。

首先我们把地址位A23到A20的值写入PMAC内存X:$078A的第7位到第4位(高4位)，也就是左边的十六进制位上(最重要的位)是地址位A23到A20的值，其它位(右边的位及最不重要的位)被设为0。在这个例子里，地址位A23到A20的值为$1，所以要把$10写入X:$078A。现在再把地址位A19到A14的值写入PMAC的基址+$121(记住，与上一个例子一样，PMAC的基址为$7FA000)。每次PMAC加电或重启时都需要这样做(或者用硬件的重启线或使用$$$命令)。在这个例子里，地址位A19到A14的值为$3F:

| 地址位 | A19 | A18 | A17 | A16 | A15 | A14 |
|---|---|---|---|---|---|---|
| 二进制 | 1 | 1 | 1 | 1 | 1 | 1 |
| 十六进制 | 3 | | F | | | |

所以在PMAC加电或重启后，我们从VME主机(主设备)把$3F写入VME总线地址$7FA121。

注意:在工作中，我们可以改变基址+$121的值，这种”动态”的寻址方法提供以16K字节块寻址DPR

AM的1M以上字节的能力。但PMAC-
VME通常只使用一个16K字节块(8Kx16)所以基址+$121只在每次PMAC加电或重启时被写入一次。

到这时，DPRAM的起始地址已完全指定了。但还需要检查PMAC内存中的另两个寄存器是否有适当的值。地址X:$078B的值应为$E0以使你的PMAC-
VME上安装的DPRAM芯片有效，我们还需要把地址X:$078C(地址宽度寄存器)中的已有值加上$80。对于这个例子，PMAC寄存器的值应为:

| PMAC地址 | 值 |
|---|---|
| X:$0783 | $39 |
| X:$0784 | $04 |
| X:$0785 | $00 |
| X:$0786 | $7F |
| X:$0787 | $A0 |
| X:$0788 | $02 |
| X:$0789 | $A1 |
| X:$078A | $10 |
| X:$078B | $E0 |
| X:$078C | $90 |

上表中加阴影的寄存器包含了我们改变的值(例子2。0)，以使DPRAM有效。在一个简单的写命令后加上一个SAVE命令可把这些值放入寄存器内并保存它们:

**WX$0783，$39，$4，$0，$7F，$A0，$02，$A1，$10，$E0，$90**
**SAVE**

在这些新值起作用之前，记住保存这些值，然后重启PMAC(用$$$命令，使INIT/输入线变低，重新上电)。在将$3F写入$7FA121(基址+$121)后，我们就可以开始使用双端口RAM了。可参见用户手册的PMAC双端口RAM部分以获得更多细节。

**通过邮箱寄存器与PMAC-VME联系**

注意: 几乎所有的用户都购买了Option 2 DPRAM，并更多地使用DPRAM的ASCII码通讯特性，而不是本节所描述的ASCII码邮箱通讯。ASCII码DPRAM通讯要更容易，也更快些。可参见Option 2 DPRAM以获得细节。

使用VME总线与PMAC通讯同使用RS232/422口是不一样的。当通过VME总线进行读写时，你必须使用16个邮箱寄存器。邮箱寄存器是从PMAC-
VME的基址+1处开始的16个8位的寄存器。就是说，如果PMAC基址被选为$7FA000，那第一个邮箱寄存器(邮箱寄存器#0)的地址为$7FA001。第二个邮箱寄存器(邮箱寄存器#1)地址为$7FA003，第三个在$7FA005，依此类推，直到第十六个位于$7FA01F。你可以发现，邮箱寄存器的地址从PMAC的基址+1后开始都是奇数。我们首先通过例子讨论如何通过邮箱寄存器把命令送给PMAC。

通过邮箱寄存器把命令送给PMAC-VME

你可能已经猜到了，当把命令送给PMAC时，向这些邮箱寄存器内写值。这相对比较简单，只需遵守以下两个规则:

1.
当向PMAC发送命令时，永远不要向邮箱寄存器#1(在我们的例子中，它位于$7FA003)内写值。该邮箱寄存器有一个特殊的用途，待一会将会提到。明白这一点后，命令的第二个字符就应被写入邮箱寄存器#2(位于$7FA005的第三个邮箱寄存器)，依此类推。

2.
把信息的第一个字符(或15个字符一组的长信息)最后写入；也就是说，首先写入命令中的其它所有字符(从邮箱寄存器#2开始)，然后把第一个字符写入邮箱寄存器#0。这样做的原因是当向邮箱寄存器#0写值时，PMAC会立即读所有的邮箱寄存器并开始执行接收的命令行。

注意:不要忘记用一个回车<CR>结束你的ASCII码信息(命令)。不需要<CR>的控制字符命令将直接被写入邮箱寄存器#0。

*举例*

假设你想选择#1电机并使它微动。可达到此目的的两个命令可被组合在一行里:#1J+<CR>。这儿有5个ASCII字符，所以我们将写入5个邮箱寄存器内。为了发送这个命令，我们将执行5个VME写命令。PMAC的基址与前面的例子一样。下表是当我们这样做时邮箱寄存器中的内容:

| 地址 | $7FA001 | $7FA002 | $7FA005 | $7FA007 | $7FA009 | $7FA00B |
|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | 4 | 5 |
| 字符 | | --- | 1 | | | |

| 地址 | $7FA001 | $7FA002 | $7FA005 | $7FA007 | $7FA009 | $7FA00B |
|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | 4 | 5 |
| 字符 | | --- | 1 | J | | |

| 地址 | $7FA001 | $7FA002 | $7FA005 | $7FA007 | $7FA009 | $7FA00B |
|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | 4 | 5 |
| 字符 | | --- | 1 | J | + | |

| 地址 | $7FA001 | $7FA002 | $7FA005 | $7FA007 | $7FA009 | $7FA00B |
|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | 4 | 5 |
| 字符 | | --- | 1 | J | + | <CR> |

| 地址 | $7FA001 | $7FA002 | $7FA005 | $7FA007 | $7FA009 | $7FA00B |
|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | 4 | 5 |
| 字符 | # | --- | 1 | J | + | <CR> |

如同所看到的，首先向$7FA005写入一个ASCII码1，然后向$7FA007写入一个J，然后向$7FA009写入一个+，然后向$7FA00B写入一个回车(ASCII码为13)，最后，向$7FA001写入一个#。

*举例*

当一个命令行包含的字符等于或少与15个字符时(包括为结束这一行加上的<CR>)，以上的例子是可以很好地工作的。当如果命令行包含的字符多于15个字符时该怎么办呢?要知道我们只有15个邮箱寄存器可以使用(别忘记邮箱#1在向PMAC发送数据时不能使用!)。

需要做的就是输入头15个字符(不要发送<CR>)，接着连续输入余下的字符直到写完所有的字符。在送完最后一个字符后，我们再送一个<CR>，告诉PMAC执行命令。现在假设你下载了一个运动程序，其中的一条语句是: IF (P1=1) DISPLAY “DELTA TAU” <CR>。我们有27个字符需要输入，即要执行27个VME写命令。下表再次显示了邮箱寄存器的内容。在写完第一组的14个字符后(从字符F到Y)，你的邮箱寄存器的内容如下表所示:

| 地址 | $7FA001 | $7FA003 | $7FA005 | $7FA007 | … | $7FA01D | $7FA01F |
|---|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | … | 14 | 15 |
| 字符 | | --- | F | ( | … | A | Y |

现在，你写入第一个字符I

| 地址 | $7FA001 | $7FA003 | $7FA005 | $7FA007 | … | $7FA01D | $7FA01F |
|---|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | … | 14 | 15 |
| 字符 | I | --- | F | ( | … | A | Y |

这时，PMAC把这些字符送入它的命令队列中，但由于没有遇上<CR>，它不做任何事情。它声明一个中断等级(缺省为2)并为这个命令提供必须通过确认的接收中断向量(缺省为$A0)，现在我们发送剩下的11个字符(从D到跟着一个<CR>的”):

| 地址 | $7FA001 | $7FA003 | $7FA005 | $7FA007 | … | $7FA01D | $7FA01F |
|---|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | … | 14 | 15 |
| 字符 | | --- | D | E | … | “ | <CR> |

最后，你送入第二(最后)组字符的第一个字符，”:

| 地址 | $7FA001 | $7FA003 | $7FA005 | $7FA007 | … | $7FA01D | $7FA01F |
|---|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | … | 14 | 15 |
| 字符 | “ | --- | D | E | … | “ | <CR> |

PMAC再次声明一个中断等级2，并为这个命令提供一个接收中断向量。由于你送入了一个<CR>，PMAC知道命令行全部输入了。现在它把这一行送入刚才打开的程序缓冲区(记住，在这个例子里，我们正在把一个运动程序下载到PMAC上)。

通过邮箱寄存器从PMAC-VME读取数据

我们已经可以用邮箱寄存器发送数据给PMAC-VME了。现在决定如何从PMAC-VME读入数据。读入数据需要使用中断和PMAC-VME通过VME总线产生的中断向量。在下面的例子里，PMAC的基址为$7FA000，I变量I3被设为2(编写主机通讯程序的最好设置)。

通过邮箱寄存器从PMAC读取数据的关键在于向邮箱寄存器#1中写值以允许PMAC有要输出数据时将它的数据放在邮箱寄存器内。这实际上是事先通过"预使能"PMAC的响应来完成的。。。这是在下面的例子里使用的方法。

如果没有"预使能"，那么仅在期望立刻有一个响应时才向邮箱寄存器#1写值，这通常是在确认$A0中断之后(可见例子)。如果不这样做，PMAC将不会用$A1向量中断你。(没有"预使能"的唯一实际的好处是你可以打断一个长的PMAC响应来执行一个命令)。注意，如果使用"预使能"的方法，必须在加电或重启之后就进行预使能。在阅读完以下例子后可参考表3。1的流程图。

*例子*

假设将下面的命令行送给PMAC:#1J+<CR>。你可能知道，这个命令行不需要任何数据响应，所以PMAC除了一个确认<ACK>，表示收到一个合法命令(如果是一个非法命令，将送出一个<BELL>字符取代<ACK>)之外将没有任何数据响应。在这种情况下，PMAC将产生一个中断，并同时发出一个中断向量$A0(在PMAC寄存器X:$0789中定义)。在看到这个中断和伴随它的中断向量后，(VME主设备或主机)必须对该中断进行服务或确认，使PMAC收回它的中断申请。(一般的，当响应任何VME中断时，中断向量将是可用的。)

PMAC之后将再次中断，这次用的是中断向量$A1，指示在邮箱寄存器内有数据需要读出。现在，如果希望的话，可以读邮箱寄存器#0($7FA001)，"取出"由PMAC放在那儿的<ACK>字符，但这仅仅在需要确认刚才发出的那个命令行已被PMAC作为合法命令接收。最后，需要向邮箱寄存器#1($7FA003)写入$00，允许PMAC在需要的时候向邮箱寄存器内写入新数据(请阅读下一个例子以更好地理解这一点)。下一个例子演示如何从邮箱寄存器内读出由PMAC写入的数据。

*例子*

假设送入命令，查询电机1的位置:#1P<CR>。PMAC当然会用包含电机1的位置信息的数据来响应。如果电机1此时的位置在19.2。现在想读该信息的邮箱寄存器。首先要做的事情是送入这个命令行并使PMAC产生中断(使用中断向量$A0)。

在PMAC已处理了命令并把数据放入了邮箱寄存器内后，PMAC用中断向量$A1再次中断。记住，第二个中断产生的原因是因为PMAC已经把数据放入了邮箱寄存器内等待读取了。我们检查第二个中断并注意到中断向量为$A1，即PMAC通知我们可以读取邮箱寄存器内的数据了。

实际上，可以以任何顺序读取这些寄存器，但最好还是从第一个邮箱寄存器开始读取这些字符直到遇见一个<CR>(指示一行的结束)或一个<ACK>(确认接收到合法命令)或一个<BELL>(确认接收到非法命令)或已读取所有的邮箱寄存器(从$7FA001到$7FA01F)。

可以任意多地重新读取这些寄存器，因为PAMC不会向这些寄存器中写入新的数据(如果PMAC有更多的数据需要发送)直到向邮箱寄存器#1(在这个例子里为$7FA003)内写入$00。

在这个例子里，有6个字符等待被读出:19.2<CR><ACK>。(我们假设I变量I3被设为2。)寄存器中的数据如下所示:

| 地址 | $7FA001 | $7FA001 | $7FA001 | $7FA001 | $7FA001 |
|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | 4 |
| 字符 | 1 | 9 | . | 2 | <CR> |

从$7FA001，寄存器0开始读取字符。在邮箱寄存器4中我们遇见了<CR>，然后停止读取，并向邮箱寄存器1中写入$00来通知PMAC可以送入新的数据。由于PMAC仍要送出最后的<ACK>，它再次产生中断，并且我们发现在邮箱寄存器中:

| 地址 | $7FA001 | $7FA001 | $7FA001 | $7FA001 | $7FA001 |
|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | 4 |
| 字符 | <ACK> | 9 | . | 2 | <CR> |

现在我们从邮箱寄存器0开始并在其上结束，因为它包含一个<ACK>。现在，仅仅需要从邮箱寄存器#0，即$7FA001开始读入这些字符。回忆起我们说过当向PMAC-VME发送数据时永远不要向邮箱寄存器#1中写数据。这是因为一旦我们向邮箱寄存器#1中写值，就将允许PMAC向邮箱寄存器中写入新的数据。(附带地，它并不在意向邮箱寄存器#1中写入什么值，而只需要向这个寄存器写值这样一个事实。但是，最好还是向邮箱寄存器#1中写入$00，原因将在以后给出。)在向邮箱寄存器#1中写入$00之后，我们从PMAC处再次获得中断，这取决于PMAC是否还有更多的数据需要读取。

*例子*

假设送入命令，查询内存X:$1000到X:$1002的内容:RHX$1000，3<CR>。又假设这三处内存的数据分别为$123456，$789012，$345678。我们希望从邮箱寄存器中读入数据。送入上面的命令，并检查PMAC产生的中断(使用的中断向量为$A0)。

在PMAC已处理了命令并把数据放入了邮箱寄存器内，PMAC用中断向量$A1再次中断。记住，第二个中断产生的原因是因为PMAC已经把数据放入了邮箱寄存器内等待读取了。检查第二个中断并注意到中断向量为$A1，即PMAC通知我们可以读取邮箱寄存器内的数据了。在这个例子里，PMAC有22个字符需要被读取:123456 789012 345678<CR><ACK>，前16个在邮箱寄存器中。(我们再次假设I变量I3被设为2。)寄存器内的数据为:

| 地址 | $7FA001 | $7FA003 | $7FA005 | $7FA007 | … | $7FA01D | $7FA01F |
|---|---|---|---|---|---|---|---|
| 邮箱号 | 0 | 1 | 2 | 3 | … | 14 | 15 |
| 字符 | 1 | 2 | 3 | 4 | … | 3 | 4 |

从第一个寄存器开始读入数据直到遇见一个<CR>，<ACK>，<BELL>或读完所有的16个寄存器。在这个例子中，头16个字符中不包含<CR>，<ACK>或<BELL>。所以我们读完所有16个寄存器得到PMAC响应的前16个字符，然后向邮箱寄存器#1中写入$00(在这个例子里，为$7FA003)来允许PMAC在邮箱寄存器中放入余下的数据。PMA再次用$A1产生中断，此时寄存器中数据为:

| 地址 | $7FA001 | … | $7FA009 | $7FA00B | … | $7FA01D | $7FA01F |
|---|---|---|---|---|---|---|---|
| 邮箱号 | 0 | … | 4 | 5 | … | 14 | 15 |
| 字符 | 5 | … | <CR> | 2 | … | 3 | 4 |

现在我们再次读取邮箱寄存器，寻找<CR>，<ACK>或<BELL>。我们从邮箱寄存器#4($7FA009)中读入的第五个字符包含一个<CR>，所以我们停止读取并向邮箱寄存器#1写入$00。
由于PMAC仍要送出最后的<ACK>，它再次产生中断，并且发现在邮箱寄存器中:

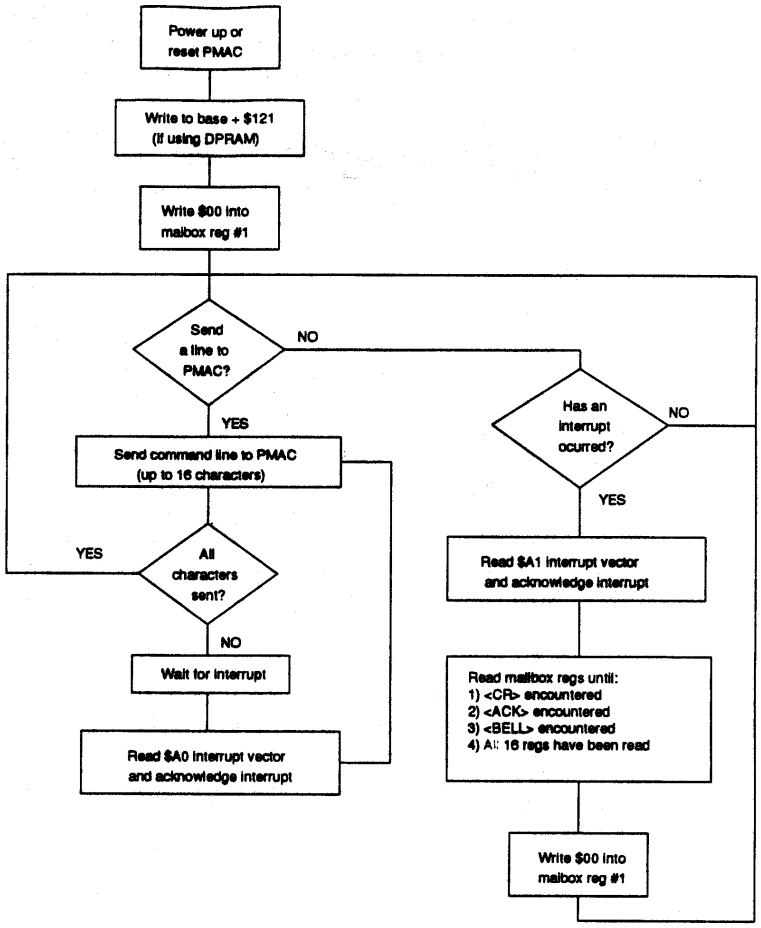
| 地址 | $7FA001 | … | $7FA009 | $7FA00B | … | $7FA01D | $7FA01F |
|---|---|---|---|---|---|---|---|
| 邮箱号 | 0 | … | 4 | 5 | … | 14 | 15 |
| 字符 | <ACK> | … | <CR> | 2 | … | 3 | 4 |

现在在第一个字符，作为字符传送结束的<ACK>处停止，并再次向邮箱寄存器#1中写入$00。因为PMAC现在没有更多的数据需要被读取，将不会获得连另一个中断(直到送入另一个命令或执行一个通过CMD或SEND命令输出数据的PLC或运动程序)。

下一页的图表概述了使用邮箱寄存器通过VME总线进行通讯的过程。

**PMAC通讯流程图**

当我们用命令LIST PROG 10或LIST PLC




0要求PMAC列出一个运动或PLC程序时，PMAC将有多行数据需要被读取。我们所要做的仅仅是等待中断发生，读取邮箱寄存器，向邮箱寄存器#1中写入$00，并等待再次中断，重复这个过程直到PMAC送出的所有数据都被读取。上图就是显示这个过程的流程图。

## 双端口RAM通讯

PMAC的Option 2提供了一个8Kx16位的RAM，允许PMAC和它主机共享一块快速内存。对于PMAC-PC和PMAC-Lite，Option 2是一块位于PC总线上，并与PMAC连接在一起的独立的板。对于PMAC-VME，Option 2(V)加在主板上的IC组成。Option 2对于PMAC-STD是不可用的。双端口RAM用来与PMAC进行快速的数据和命令通讯。

**DPRAM的使用**

在向PMAC写数据时通常用于在实时状态下快速的位置数据和/或旋转程序信息的重复下载。在从PMAC读数据时通常用于重复快速地获得状态信息。

类似电机状态，位置，速度，跟随误差，等。的数据可不停地更新并被PLC程序或被PMAC自动地写入DPRAM。如果不使用DPRAM，这些数据必须用PMAC在线命令，如?，P，V和F，通过VME邮箱寄存器或PC总线来存取。由于通过DPRAM存取不需要经过通讯口发送命令和等待响应的时间，所以要快得多。

为使用双端口RAM(DPRAM)在主机和PMAC间传送数据，PMAC提供了许多设备。这些设备包括以下的功能:

1. DPRAM控制面板功能(从主机到PAMC)
2. DPRAM伺服数据报告功能(从PMAC到主机)
3. DPRAM后台定点数据报告功能(从PMAC到主机)

4. DPRAM后台变量数据报告功能(从PMAC到主机)
5. DPRAM ASCII 通讯缓冲区(双向)
6. DPRAM二进制旋转程序缓冲区(从主机到PMAC)
7. DPRAM数据采集缓冲区(PMAC到主机--已有)
8. DPRAM<CONTROL-W> ASCII 命令功能(从主机到PMAC--已有)

除了这些“自动”功能外，用户还可以用PMAC的M变量和主机的指针变量来存取DPRAM中其它未用的寄存器，在PMAC和主机间传送数据。M变量的格式为X:，Y:(1到16位)，DP:(32位定点)和F:(32位浮点)。在将数据送回主机时，也可以使用PMAC的数据采集功能，直接送到双端口RAM中，而不是常规的RAM(由I45控制)。可查阅PMAC DPRAM 用户指导(Option手册)以获得细节。

# 在VME总线上使用多个PMAC-VME卡

可以在VME总线上安装多个PMAC-VEM卡。一个主机可以控制多至16个PMAC-VME卡。每个卡必须有它们自己的不同的基址和DPRAM开始地址(如果安装)，这样卡与卡之间在内存上就不会发生重叠(或与VME总线上的其它设备)。你同时还必须为每个卡设置具有不同中断向量号的不同中断等级。每个PMAC-VME卡占用512个VME总线内存位置(不包括由DPRAM占用的8K字节)。这就是说，如果第一个PMAC-VME的基址为$7FA000，第二个PMAC卡的基址必须在$7FA200。假设在VME总线上安装了8个PMAC-VME卡，第一个卡的基址为$7FA000。那么每个卡的邮箱寄存器(MB)可通过下表来设置。

| • | CARD 0 | CARD 1 | CARD 2 | CARD 3 | CARD 4 | CARD 5 | CARD 6 | CARD 7 |
|---|---|---|---|---|---|---|---|---|
| 基址 | 7FA000 | 7FA200 | 7FA400 | 7FA600 | 7FA800 | 7FAA00 | 7FAC00 | 7FAE00 |
| MB#0 | 7FA001 | 7FA201 | 7FA0401 | 7FA601 | 7FA801 | 7FAA01 | 7FAC01 | 7FAE01 |
| MB#1 | 7FA003 | 7FA203 | 7FA0403 | 7FA603 | 7FA803 | 7FAA03 | 7FAC03 | 7FAE03 |
| MB#2 | 7FA005 | 7FA205 | 7FA405 | 7FA605 | 7FA805 | 7FAA05 | 7FAC05 | 7FAE05 |
| MB#3 | 7FA007 | 7FA207 | 7FA407 | 7FA607 | 7FA807 | 7FAA07 | 7FAC07 | 7FAE07 |
| MB#4 | 7FA009 | 7FA209 | 7FA409 | 7FA609 | 7FA809 | 7FAA09 | 7FAC09 | 7FAE09 |
| MB#5 | 7FA00B | 7FA20B | 7FA40B | 7FA60B | 7FA80B | 7FAA0B | 7FAC0B | 7FAE0B |
| MB#6 | 7FA00D | 7FA20D | 7FA40D | 7FA60D | 7FA80D | 7FAA0D | 7FAC0D | 7FAE0D |
| MB#7 | 7FA00F | 7FA20F | 7FA40F | 7FA60F | 7FA80F | 7FAA0F | 7FAC0F | 7FAE0F |
| MB#8 | 7FA011 | 7FA211 | 7FA411 | 7FA611 | 7FA811 | 7FAA11 | 7FAC11 | 7FAE11 |
| MB#9 | 7FA013 | 7FA213 | 7FA413 | 7FA613 | 7FA813 | 7FAA13 | 7FAC13 | 7FAE13 |
| MB#10 | 7FA015 | 7FA215 | 7FA415 | 7FA615 | 7FA815 | 7FAA15 | 7FAC15 | 7FAE15 |
| MB#11 | 7FA017 | 7FA217 | 7FA417 | 7FA617 | 7FA817 | 7FAA17 | 7FAC17 | 7FAE17 |
| MB#12 | 7FA019 | 7FA219 | 7FA419 | 7FA619 | 7FA819 | 7FAA19 | 7FAC19 | 7FAE19 |
| MB#13 | 7FA01B | 7FA21B | 7FA41B | 7FA61B | 7FA81B | 7FAA1B | 7FAC1B | 7FAE1B |
| MB#14 | 7FA01D | 7FA21D | 7FA41D | 7FA61D | 7FA81D | 7FAA1D | 7FAC1D | 7FAE1D |
| MB#15 | 7FA01F | 7FA21F | 7FA41F | 7FA61F | 7FA81F | 7FAA1F | 7FAC1F | 7FAE1F |

除了有不同的基址，每个PMAC-VME必须还有不同的中断向量号。但是，所有的PMAC可以使用一样的中断等级(在我们前一个例子中为中断等级2)。下面的表5。2显示了每个PMAC-VME卡的建议中断向量分配。

| PMAC-VME卡编号 | 中断向量对 |
|---|---|
| 0 | $A0，$A1 |
| 1 | $A2，$A3 |
| 2 | $A4，$A5 |
| 3 | $A6，$A7 |
| 4 | $A8，$A9 |
| 5 | $AA，$AC |
| 6 | $AC，$AD |
| 7 | $AE，$AF |

通过串行口上附加的线分享时钟信号，从而使所有的PMAC完全同步是可能的。要这样做，只需在所有的PMAC-

VME的J4接口间建立一个菊花链。如果使用这个方法，必须通过跳线E40到E43的设置使某个卡成为#0卡(因为#0卡输出同步时钟)。而所有其它的PMAC卡必须通过E40到E43设置为更高的编号(#1，#2，等。接受同步时钟输入)。

如果PMAC不共享一个公共时钟信号，所有的卡必须都通过E40到E43设置为#0卡。没有公共时钟信号，不同卡上的动作将发生偏移，因为每个卡都参考来自于它自己的石英振荡器的时间。但是，由于振荡器的偏差很小，只有在连续运动10到15分钟后才会出现可察觉的偏移。

# 数据完整性检验

### 串行奇偶校验

在串行通讯时PMAC可以进行奇偶校验。如果跳线E49断开，PMAC检查它通过串行口从主机接收的每个字符是否有一个奇校验位，并且在它通过串行口向主机发送每个字符时也会加上一个

奇校验位

如果PMAC在命令的任何字符中检测到一个奇偶错，它将设置一个标志，当PMAC接收到<CR>后，整条命令将被作为有语法错误而被拒绝执行。如果I4=0或1，PMAC还会马上向主机发送一个<BELL>字符来通知它。

如果I4=2或3，主机将检查在行末尾是否有一个<BELL>字符。但是，没有直接的方法来判断是出于奇偶错还是一个真的语法错误。而且，如果奇偶错发生在<CR>字符自身上，PMAC将根本不做出响应，因为它没有见到行的末尾。在这种情况下，主机必须准备进行超时检验，并要么再次发送<CR>，要么发送一个<CTRL-X>来清除整个命令行。

当I4=0或1时，主机将要么检验在发出每个字符后是否有<BELL>字符，要么等待行结束。无论哪种情况，在接收到<BELL>字符后，主机将发送一个<CTRL-X>命令来清除PMAC在两个方向上的通讯缓冲区。接着主机将用相当于传送3个字符的时间来检查送入的字符，放弃所有它接收到的字符，以确认传送的所有字符都被清除了。然后它再次发送命令行。

如果跳线E49为ON，两个方向的奇偶检验都被关闭。PMAC将不检查是否有校验位，而且如果发送了一个校验位将会产生一个帧错误(见下)。PMAC将不会在它发送的字符上加上校验位。

### 串行帧错误检查

PMAC可以检查在串行口上发送给它的命令的帧错误。该检查总是使能的。如果PMAC在命令的任何字符中发现一个帧错误，它的反应与发现一个奇偶校验错一样，包括用I4来控制响应(见上)。

### 串行双工控制

PMAC可以通过立即响应它从串行口收到的每个字符来确认与它的通讯。该模式，被称作全双工，可以通过<CTRL-T>命令来打开或关闭。PMAC启动时为半双工模式，不回应接收的字符，所以主机必须给PMAC送一个<CTRL-T>命令来打开全双工。

为执行回应，PMAC接收字符，把它写入内存中的输入队列，然后从内存中把它拷入通讯口。这样，对于命令周期就有完整的检查。如果主机接收到的回应字符与它发送的不同，它将送一个<CTRL-X>命令来清除命令行，然后重新发送命令行。

### 通讯校验和

PMAC可对它与主机之间发送的通讯行进行校验和计算。当I4=1时该模式被打开，当I4=0时被关闭。它可在串行，PC总线，STD总线或VME总线通讯时执行。

PMAC对它与主机之间发送的通讯行的每一个字节(字符)进行校验和计算。它把每个字符的ASCII码值加在一起。在适当的时候(见下)，它发送校验和--至少是低位字节--给主机。主机将对PMAC计算的校验和与它自己计算的校验和进行比较。主机不会发送一个校验和字节给PMAC。

校验和工作的方法与主机-PMAC和PMAC-主机的通讯有一点不同。下面我们将加以解释:

主机-PMAC校验和

在一个完整的命令行，包括结束的回车(<CR>)字节，被送给PMAC后，PMAC将分析命令行并判断出它要执行什么操作。当它分析命令行时，它计算命令行中所有字符的校验和。校验和不包括任何

控制字符，例如那些被插入在行中间的，或是最后的<CR>字符。

在送出确认通讯的字符之后，PMAC将立即把校验和送回主机。(当I3=0，将没有确认字符，当I3=1，PMAC使用<LF>，I3=2或3，PMAC使用<ACK>作为确认字符。)如果命令需要一个数据响应，确认字符和校验和将在命令的数据响应后(以及数据响应的校验和!)被送回主机。如果PMAC在常规的语法检查中发现一个错误，它将用<BELL>字符进行响应(无论I3为何值)，但不会有一个校验和字节--整个命令行都被放弃了。

PMAC-主机校验和

PMAC将计算它送往主机的任何通讯行的校验和。该校验和包括与那一行一起发送的控制字符，包括<LF>字符和最后的<CR>字符。校验和将在<CR>字符后被立即送出。对于多线的通讯，每一行的校验和字符都将在那一行的<CR>字符后被送出。

如果PMAC-主机的通讯是由一个主机命令引起的，在完成通讯之后(可以是多线)，PMAC将送出确认通讯字符，后面跟着主机命令的校验和字符。如果通讯是由PMAC程序中的一个SEND或CMD语句引起的，将不会有确认通讯字符或命令的校验和(但有数据响应的校验和)。

校验和格式

一般的，某一行的校验和要大于一字节--由于所有的字节值相加，校验和会有两个字节。当使用串行口或VME总线口时，PMAC只发送低位字节。该字节的值为整个校验和对256取模。当使用PC总线或VME总线，PMAC把低位字节送给普通的通讯寄存器(基址+7)，而同时也把高位字节送给相邻的寄存器(基址+6)。用户可以选择是读取低位字节，还是整个字节。

*例子*

I3=3，I4=1，假设P100=35，Q10=0，Q11=1，而Q12=2:

主机发送: J+<CR>
PMAC发送: <ACK><117dec> (117=74[J]+43[+])

主机发送: P100<CR>
PMAC发送: <LF>35<CR><127dec> (127=10+51+53+13)
<ACK><225dec> (225=80+49+48+48)

主机发送: Q10..12
PMAC发送: <LF>0<CR><71dec> (71=10+48+13)
<LF>1<CR><72dec> (72=10+49+13)
<LF>2<CR><73dec> (73=10+50+13)
<ACK><369or113dec> (369=81+49+48+46+46+49+50)
(113=369modulo256)

# 数据采集

PMAC有一个数据采集功能，可在工作时不断存储实时数据。在这个功能中，PMAC可在不超出伺服中断频率的指定时间段内使用多达24个单元来存储数据。这些数据存在一个打开的PMAC内存缓冲区内，之后将被送给主机。该特性在滤波器调节和解决运动问题时非常有用。

**执行程序数据采集**

大多数用户用PC上的PMAC执行程序来使用该特性，因为它可自动处理该功能的细节。可参考手册的执行程序部分。可以编写(虽然不普遍)一个常规的主机程序来使用这个特性。

**采集I变量**

用户可用I变量I21到I44来指定多达24个源地址。这些变量的低16位表示地址。高2位控制是使用X字，还是Y字，还是两者都使用(定点或浮点格式)来采集数据。可在I20中设置一个"屏蔽"来指定选中这些地址中的哪一个，并可用I19来定义伺服中断周期中的采集时间。

**采集命令**

PMAC中的内存缓冲区可用在线命令DEFINE GATHER [{常数}]来设置。如果不指定值，那么PMAC所有打开的内存都被保留做这个缓冲区。(这意味着一旦该空间被保留下来，就不可以再给PMAC加入新的运动或PLC程序了)。实际的数据采集功能是由GAT

HER命令启动的。当输入该命令后，PMAC将以指定的速度把指定的数据送入采集缓冲区直到用END G告诉它停止，或直到空间不够了。

存储的数据可用LIST GATHER命令送给主机。数据将以十六进制ASCII码的形式被送往主机，对于单个(X或Y)字，为每项6个字符；对于双字(L或D)，为每项12个字节。数据将以12个字节为1组的形式提供。如果采样数据的最后一组只有6个字符，该组将用伺服周期计数器寄存器的内容来填写。

主机将对这些数据进行解码和处理，用于绘图，存储，分析或其它用途。

为数据采集缓冲区保留的空间可用DELETE GATHER命令来释放。

**在线数据采集**

<CTRLOL-E>命令是一种单击数据采集方式。在接收到这个命令后，PMAC将把由I21到I44指定的寄存器的内容送到主机。这里，数据的内容是以二进制，而不是ASCII码的形式送出的，并且没有任何握手协议字符。每个短字为3个字节，每个长字为6个字节。该命令用于快速的状态和位置查询。

**通过双端口RAM的实时数据采集**

使用双端口RAM，可以实现PMAC的数据采集功能并把采集的数据送给主机。(标准的数据采集功能—PMAC执行程序用来绘图--实时执行数据采集，把数据存在PMAC打开的常规内存里，然后送往主机。)这种实时上载要求PMAC与主机的通讯要稳定，才能保证数据可以可靠和高效地传送。

设置

DPRAM数据采集功能的设置和标准的数据采集功能设置是一样的，用I变量I19到I44来控制采集的数据，及采集频率。为实现指定数据采集到DPRAM，I45应被设为3(在标准采集时它被设为0)。

临时存放采集数据的缓冲区可用DEFINE GATHER {常数}命令来建立，{常数}是以字为单位的缓冲区的大小(在DPRAM中每个PMAC字为32位)。缓冲区总是从PMAC字地址$D200开始，对主机来说为DPRAM基址+$0800(2048)。每个短的数据源占用缓冲区内的一个字，每个长的数据源(定点或浮点)占用两个字。用户必须根据数据源的长度和数目，以及在最坏的情况下主机从DPRAM读取数据落后的采集周期数来设置缓冲区的大小。对于8Kx16的DPRAM，不能大于3500。通常取20到100个字。

DPRAM数据采集功能的启动和停止同标准数据采集功能是相同:要么用GATHER和ENDGATHER(ENDG)命令，要么直接通过M变量设置和清除数据采集控制位。

获取数据

一旦采集开始，主机必须监视DPRAM中包含送入DPRAM的数据的指针的寄存器。有两个关键的寄存器，而仅有一个需要重复地读入。在DPRAM基址+$07FE(2046)处是指向缓冲区结束的指针。该值由DEFINE GATHER命令指定，对于一个给定的功能它是固定的。

在DPRAM基址+$07FC(2044)处是指向DPRAM中将要存入采集数据的下一个地址的指针。就是这个寄存器，主机需要不停地监视，看它是否改变--意味着有新的数据送入DPRAM--以及如果变化，有多少个数据已被输入。

这些寄存器都包含一个PAMC内存字地址--实际上是从采集缓冲区开始($D200)的偏置。要把它翻译成主机的内存地址，可以使用下面这个公式:

Host_address=(DPRAM_base_address+$0800)+$*(Point_value)

当存储器指针的值(在$D200)等于或大于缓冲区结束的指针时，它将会翻转为0。没有数据项存在由缓冲区结束指针显示的PMAC字地址开始的DPRAM内，即使如果一个长的数据项从前一个DPRAM字处开始存储，后一半将会被放在实际的缓冲区结束的字上。

数据格式

数据以32位带符号扩展的格式存在缓冲区内。就是说，每个PMAC采集的短字(24位)是带符号扩展的，并存在32位的DPRAM内(低位在前)。符号位由第23位扩展而来，全部为1或0。每个长字(48位)被当作两个24位字对待，然后将每个短字符号扩展至32位。主机必须重新组合这些字而得到一个值。

要组合一个长的定点字，首先取出低位的32位字，并屏蔽掉符号位(高8位)。在C语言中，可用一

个位与AND:(LSW&16777215)来完成这一步。把这个结果当作一个无符号整数。接着，取出高位字，并将它乘以16,777,216。最后，把这两个中间值相加。

要组合一个长的浮点值，我们首先要把它分成两部分。低地址32位字的0到11位构成指数。这个字的12到23位构成36位尾数的低12位。另一个32位字的0到23位构成尾数的高24位；24到31位为符号扩展位。下面的代码可以完成这些工作(但不是最有效):

```
exp=first_word&0x00000FFFL;                 //选择低12位为指数
low_mantissa=(first_word>>12)&0x00FFFL;  //选择接下来的12位为尾数
                                            //右移并屏蔽
high_mantissa=second_word;
```

浮点数值可以如下计算:

```
mantissa=high_mantissa*4096.0+low_mantissa;
value=mantissa*pow(2.0，exp-2047-35);
```

从指数中减去2047，将它从无符号数转换成有符号数；减去35是将尾数放在0。5到1。0之间。下面的C程序稳定而有效地完成了转换:

```
double DPRFLoat (long first_word ， long second word){
    //return mantissa /2^35*3^(exp-2047)
    double mantissa;
long int exp;
     m=double)(second_word)*4096.0+(double)((first_word>>12)&0x00FFL);
    if (m==0.0) return (0.0);
    exp=(first_word&0x00000FFFL)-2082;          //2082=2048+35
    return (mantissa *pow(2.0*(double)exp));
}
```

要组合一个长的浮点字，低32位字的处理与定点字的例子一样。取出高32位字的低12位(MSW&4095)，乘以16,777,216并加在屏蔽后的低32位字上。构成浮点值的尾数。取出接下来的12位(MSW&16773120)。这是以2为低的指数，它与尾数一起组成了完整的数值。

more significant word (MSW & 4095), multiply by 16,777,216 and add to the masked less significant word. This forms the mantissa of the floating-point value. Now take the next 12 bits (MSW & 16773120) of the more significant word. This is the exponent to the power of two, which can be combined with the mantissa to form the complete value.