第 4 章 程序存储器

目录

本章包括下列主题：

4.1 程序存储器地址映射

PIC24F 器件有 4M x 24 位程序存储器地址空间，如图 4-2 所示。访问程序空间有三种方法可用。

1. 通过 23 位 PC。
2. 通过表读 （TBLRD）和表写 （TBLWT）指令。
3. 通过把程序存储器的 32 KB 段映射到数据存储器地址空间。

程序存储器映射被划分为用户程序空间和配置空间。用户程序空间 （000000h 至 7FFFFFh）包含复位向量、中断向量表和程序存储器。器件配置寄存器和器件 ID 单元映射在配置空间中。配置位和器件 ID 可从这些存储单元读取；但是，可通过对闪存配置字中的期望值编程来置 1 或清零配置位。片上程序存储器的头两个字是为配置信息保留的。器件复位时，该配置信息被复制到相应的配置寄存器中。有关配置位的更多信息，请参见**第 32 章 “器件配置”**。

4.1.1 程序存储器构成

程序存储空间由可字寻址的块 （block）构成。尽管程序存储空间被视为 24 位宽，但最好还是将它的每个地址看作低位和高位字，高位字的高位字节未实现。低位字总是偶地址，而高位字总是奇地址 （图 4-1）。

程序存储器地址在低位字上总是字对齐的，执行代码时地址增 2 或减 2。

图 4-1： 程序存储器构成

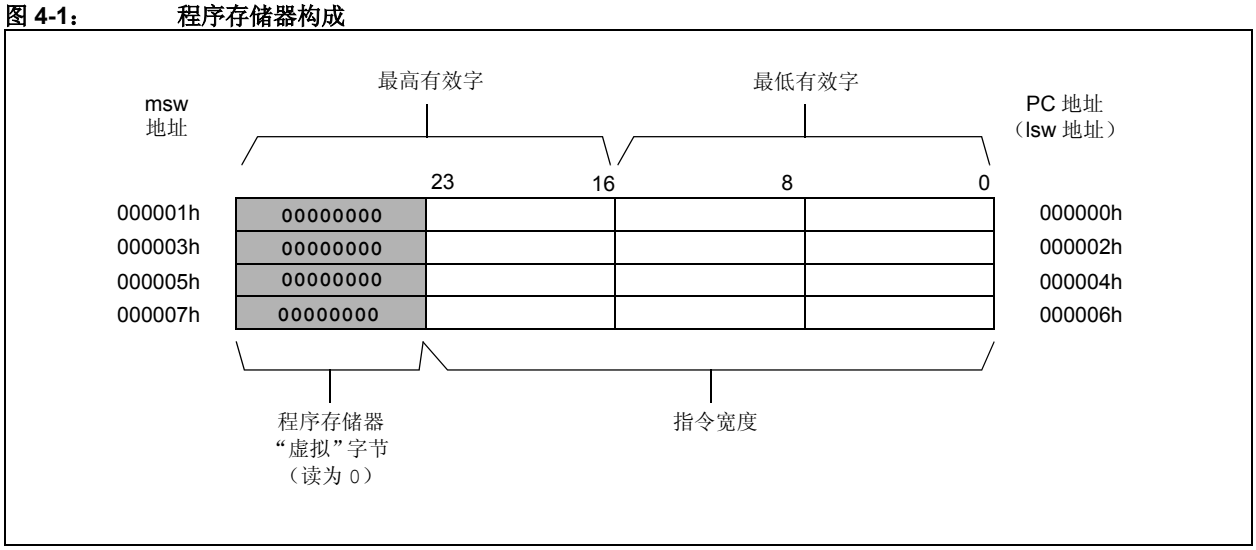
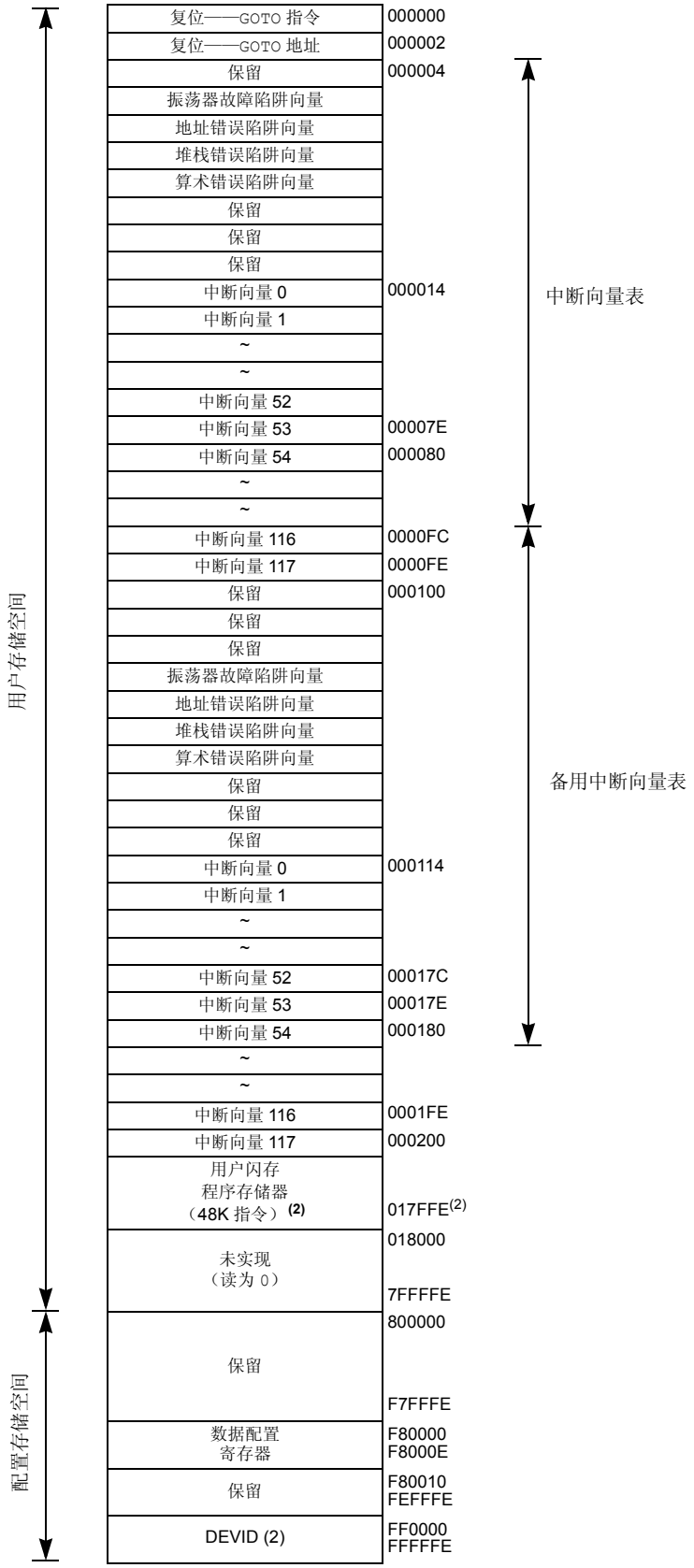


图 4-2：　　程序空间存储器映射示例 (1)



注　**1：**存储器区域未按比例显示。
2：用户闪存程序存储器和闪存配置字的地址边界取决于所选择的 PIC24F 器件。更多详细信息，请参见相应器件的数据手册。

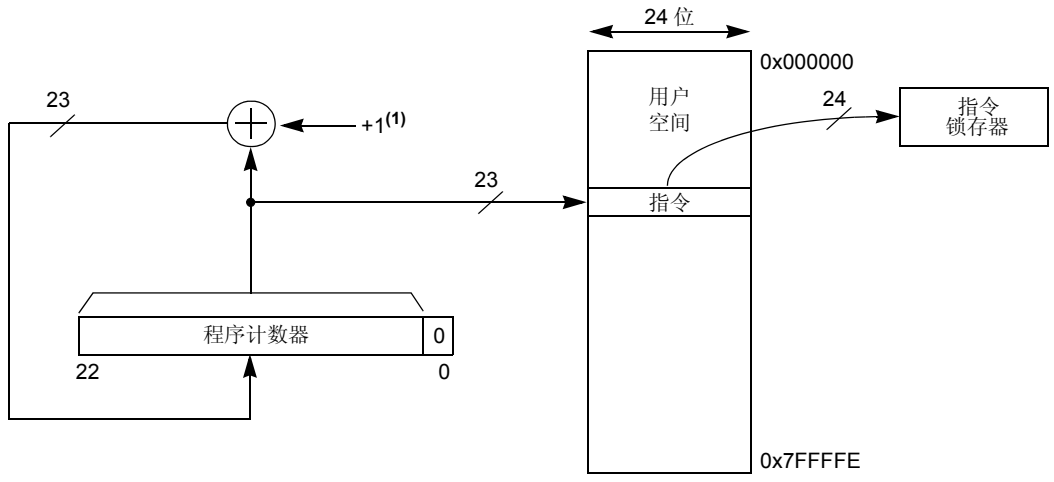
4.2 程序计数器

PC 以 2 为增量且 LSb 设置为 0 以使之与数据空间寻址相兼容。用 PC<22:1> 在 4M 程序存储空间中对连续指令字寻址。每个指令字为 24 位宽。

程序存储器地址的 LSb （PC<0>）保留为字节选择位，用于从使用程序空间可视性 （Program Space Visibility， PSV）或表指令的数据空间访问程序存储器。对于通过 PC 取指的情况，不需要该字节选择位。所以，此时 PC<0> 总是设置为 0。

取指示例如图 4-3 所示。注意 PC<22:1> 增 1 相当于 PC<22:0> 增 2。

图 4-3： 取指示例



注 1： PC<22:1> 增 1 相当于 PC<22:0> 增 2。

4.3 从程序存储器存取数据

可以使用两种方法在程序存储器和数据存储空间之间传送数据，即：可通过特殊的表指令或通过把 32 KB 程序空间页重新映射到数据空间的上半部分进行。TBLRDL 和 TBLWTL 指令提供了读或写程序空间内任何地址的最低有效字（least significant word， lsw）的直接方法（无需通过数据空间），很适合某些应用。 TBLRDH 和 TBLWTH 指令是可以把一个程序字的高 8 位作为数据存取的惟一方法。

4.3.1 表指令汇总

我们提供了一组表指令用于将字节或字大小的数据在程序空间和数据空间之间传送。表读指令用于把数据从程序存储空间读入数据存储空间。表写指令可以把数据从数据存储空间写入程序存储空间。

注： 使用表指令的详细代码示例，请参见**第 5 章 “闪存操作”**。

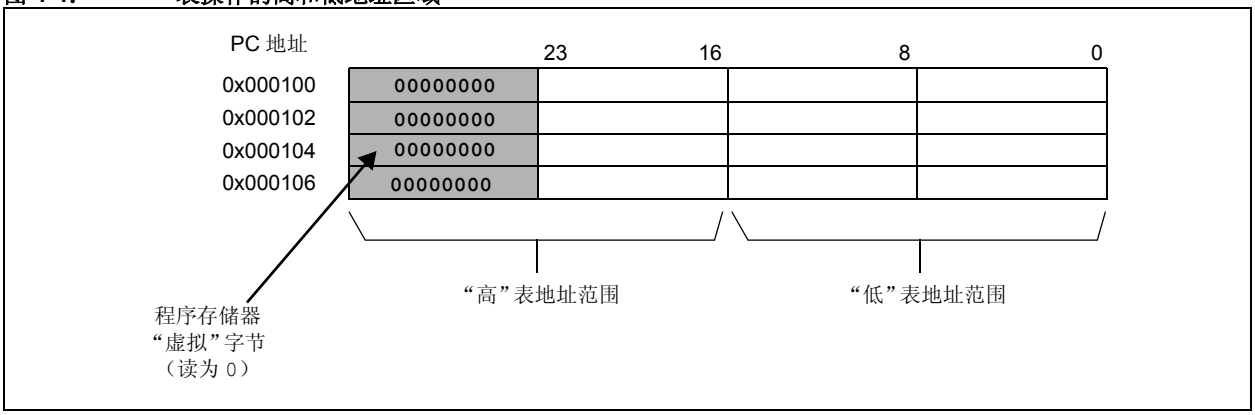
下面列出了 4 条可用的表指令：

- TBLRDL：表读低位字
- TBLWTL：表写低位字
- TBLRDH：表读高位字
- TBLWTH：表写高位字

对于表指令，程序存储器可以视作并排放置的两个 16 位字宽的地址空间，每个地址空间都有相同的地址范围，见图 4-4。

TBLRDL 和 TBLWTL 访问程序存储器的 lsw，而 TBLRDH 和 TBLWTH 访问高位字。由于程序存储器只有 24 位宽，所以后一个字空间的高字节不存在（虽然它是可寻址的）。因此称之为“虚拟（phantom）”字节。

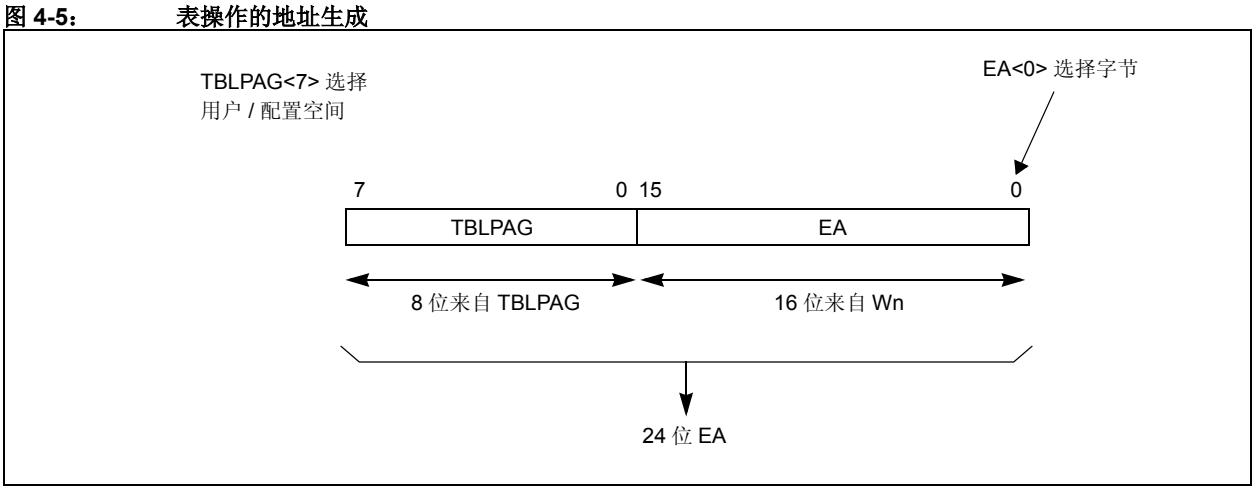
图 4-4：　表操作的高和低地址区域



4.3.2　表地址的生成

对于所有表指令，W 寄存器地址值与 8 位表页地址指针 TBLPAG 相连，形成一个 23 位有效的程序空间地址加上一个字节选择位，如图 4-5 所示。由于 W 寄存器提供了 15 位的程序空间地址，所以程序存储器中的数据表页的大小为 32K 字。

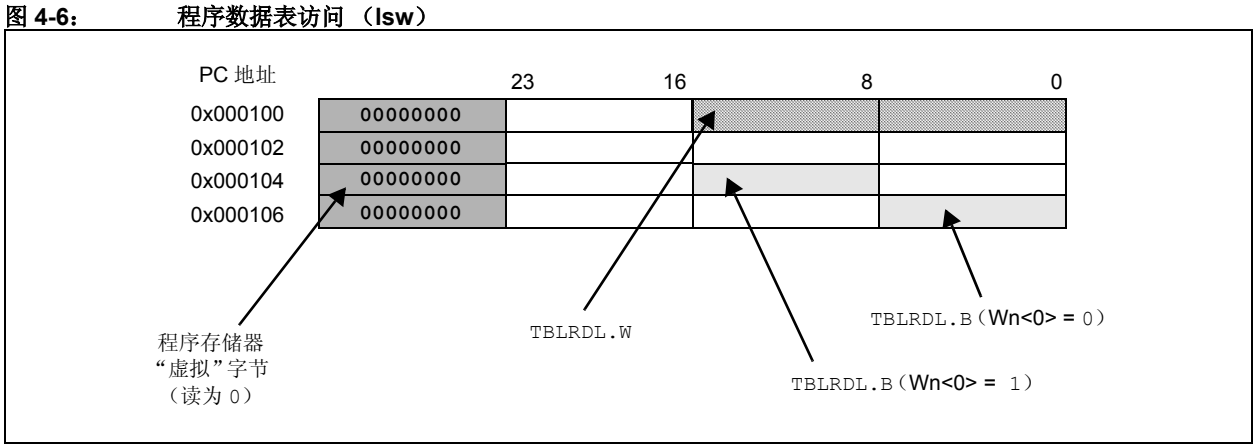
图 4-5：　表操作的地址生成



4.3.3 程序存储器低位字访问

TBLRDL 和 TBLWTL 指令用于访问程序存储器数据的低 16 位。对于以字宽为单位的表访问， W 寄存器地址的 LSb 被忽略。对于以字节宽为单位的访问， W 寄存器地址的 LSb 决定读哪一个字节。图 4-6 演示了用 TBLRDL 和 TBLWTL 指令访问的程序存储器数据区域。

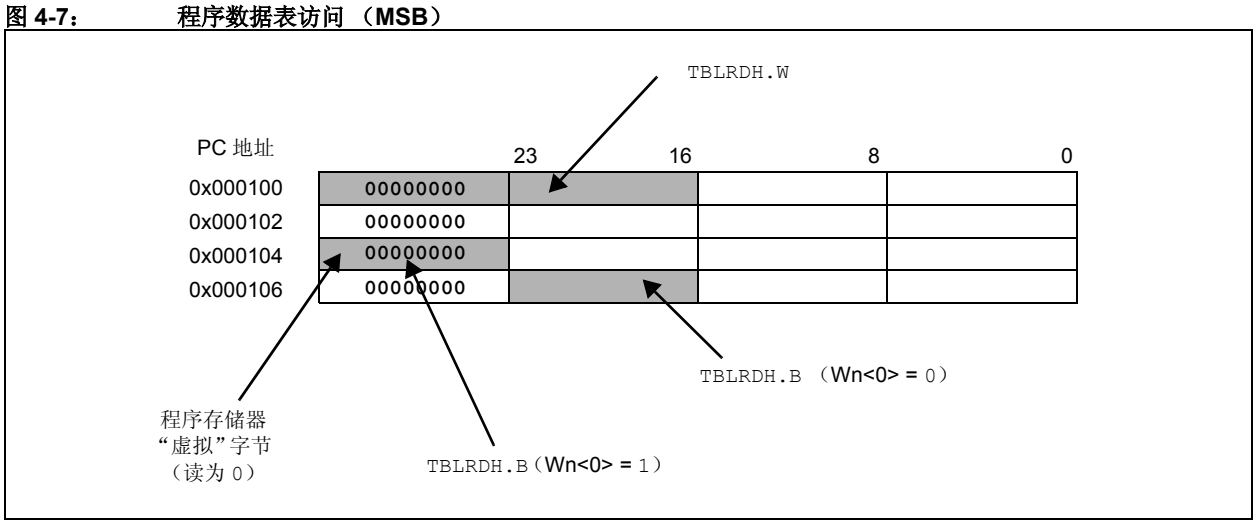
图 4-6： 程序数据表访问 （lsw）



4.3.4 程序存储器高位字访问

TBLRDH 和 TBLWTH 指令用于访问程序存储器数据的高 8 位。为了正交性，这些指令也支持字或字节访问模式，但是程序存储器数据的高字节将总是返回 0，如图 4-7 所示。

图 4-7： 程序数据表访问 （MSB）



4.3.5 程序存储器中的数据存储

假设对于大多数应用，高字节 （P<23:16>）不用于存储数据，这使程序存储器有 16 位宽供数据存储。建议将程序数据的高字节编程为 NOP （0x00 或 0xFF）或作为非法操作码 （0x3F）值，以防止器件意外执行所存储的数据。TBLRDH 和 TBLWTH 指令主要用于阵列编程 / 校验和那些要求压缩数据存储的应用。

4.4 来自数据空间的程序空间可视性

可选择将 PIC24F 数据存储器地址空间的高 32 KB 映射到任何 16K 字程序空间页。这种操作模式被称为程序空间可视性 （PSV），它提供从数据空间对存储的常数数据的透明访问，而无需使用特殊指令 （即， TBLRD 和 TBLWT 指令）。

4.4.1 PSV 配置

通过将 PSV 位 （CORCON<2>）置 1 使能程序空间可视性。有关对 CORCON 寄存器的描述，请参见**第 2 章 “CPU”**。

当 PSV 使能时，在数据存储器映射空间上半部分 （地址较高的数据存储器）的每个数据空间地址将直接映射到一个程序地址（见图 4-8）。PSV 视窗允许访问该 24 位程序字的低 16 位。程序存储器数据的高 8 位应该编程，以强制对其的访问为非法指令或 NOP，从而维持器件的鲁棒性（robustness）。请注意表指令是读每个程序存储器字的高 8 位的惟一方法。

图 4-9 显示了如何生成 PSV 地址。PSV 地址的 15 个 LSb 由包含有效地址的 W 寄存器提供。W 寄存器的 MSb 不用于形成该地址，而是用于指定是从程序空间执行 PSV 访问还是从数据存储空间执行正常的访问。如果使用的 W 寄存器有效地址大于或等于 0x8000，使能 PSV 时，数据访问会从程序存储空间进行。当 W 寄存器的有效地址小于 0x8000 时，所有访问将从数据存储器进行。

余下的地址位由 PSVPAG 寄存器 （PSVPAG<7:0>）提供，如图 4-9 所示。 PSVPAG 位与 W 寄存器中保存有效地址的 15 个 LSb 相连形成一个 23 位的程序存储器地址。 PSV 只能用来访问程序存储空间中的值。必须用表指令来访问用户配置空间中的值。

W 寄存器值的 LSb 用作字节选择位，该位允许使用 PSV 的指令以字节或字模式运行。

图 4-8：　　程序空间可视性操作

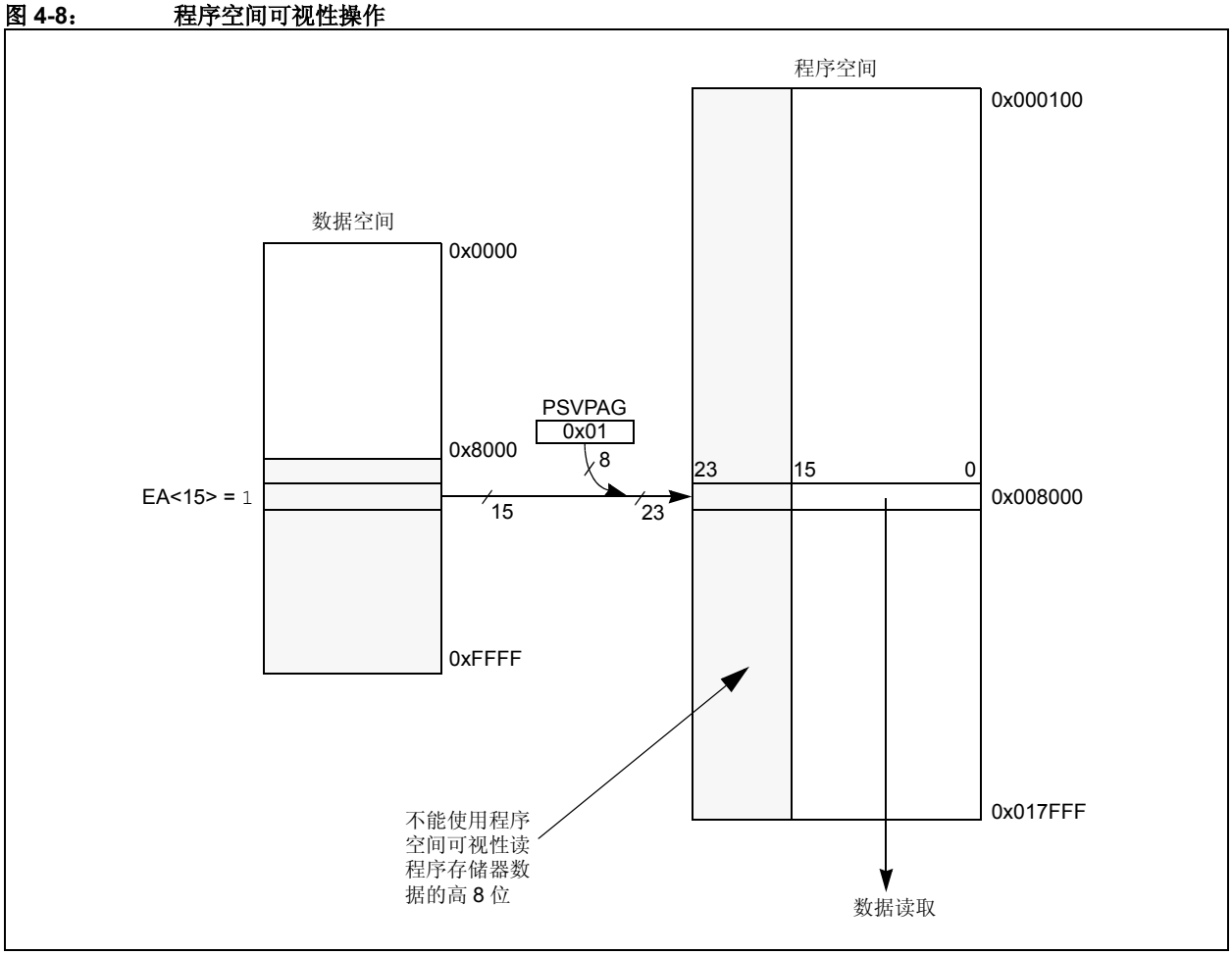
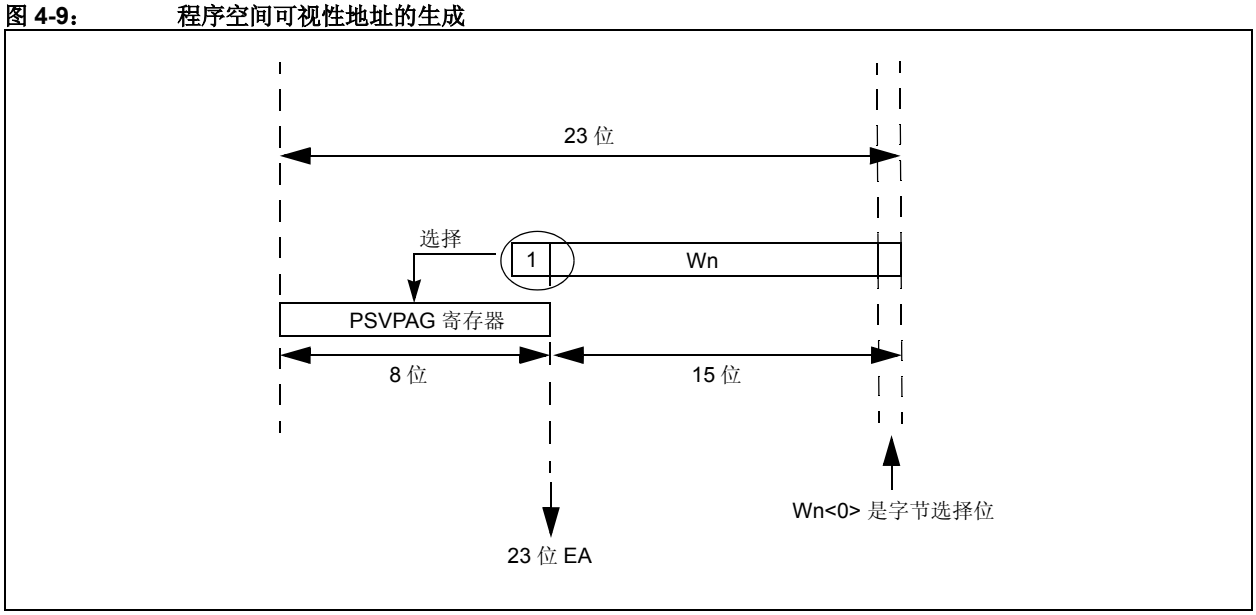


图 4-9：　　程序空间可视性地址的生成



4.4.2 PSV 时序

除了所有 MOV 指令（包括 MOV.D 指令）只需要一个额外周期即可执行完毕以外，其他使用 PSV 的指令需要两个额外的指令周期才能执行完毕。

额外的指令周期用于取程序存储器总线上的 PSV 数据。

4.4.2.1 在 REPEAT 循环中使用 PSV

在一个 REPEAT 循环内使用 PSV 的指令消除了访问程序存储器内数据所需的额外指令周期，从而不会产生执行时间上的开销。然而，以下 REPEAT 循环迭代将产生两个指令周期的开销才能执行完毕：

- 第一次迭代。
- 最后一次迭代。
- 由于中断，在退出该循环前的指令执行。
- 中断处理后，重新进入该循环的指令执行。

4.4.2.2 PSV 和指令停顿

更多有关使用 PSV 的指令停顿 （stall）的信息，请参见**第 2 章 “CPU”**。

4.5 写程序存储器

本节描述闪存程序存储器的编程技术。 PIC24F 系列器件包含内部可编程闪存存储器，用于执行用户代码。用户可用两种方法对该存储器编程：

- 运行时自编程 （Run-Time Self-Programming， RTSP）
- 在线串行编程 （In-Circuit Serial Programming， ICSP）
- 增强型在线串行编程 （Enhanced In-Circuit Serial Programming， EICSP）
- JTAG 编程

RTSP 是由用户软件执行的。 ICSP 和 EICSP 是通过使用器件的串行数据连接执行的，编程比 RTSP 快得多。本节将描述 RTSP 技术。

“PIC24FJXXXGA0XX Flash Programming Specification”（DS39768）中定义了 ICSP 和 EICSP 协议，可从 Microchip 网站（www.microchip.com）下载。 IEEE 1149.1-2001 的编程章节，*“IEEE Standard Test Access Port and Boundary Scan Architecture”*中定义了 JTAG 编程。

4.6 表指令操作

表指令提供了一种在 PIC24F 器件的程序存储空间和数据存储空间之间传送数据的方法。本节提供了对闪存程序存储器编程期间使用的表指令汇总。有 4 条基本的表指令：

- TBLRDL：表读低位字
- TBLRDH：表读高位字
- TBLWTL：表写低位字
- TBLWTH：表写高位字

TBLRDL 和 TBLWTL 指令用于读或写程序存储空间的 <15:0> 位。TBLRDL 和 TBLWTL 能以字或字节模式访问程序存储器。

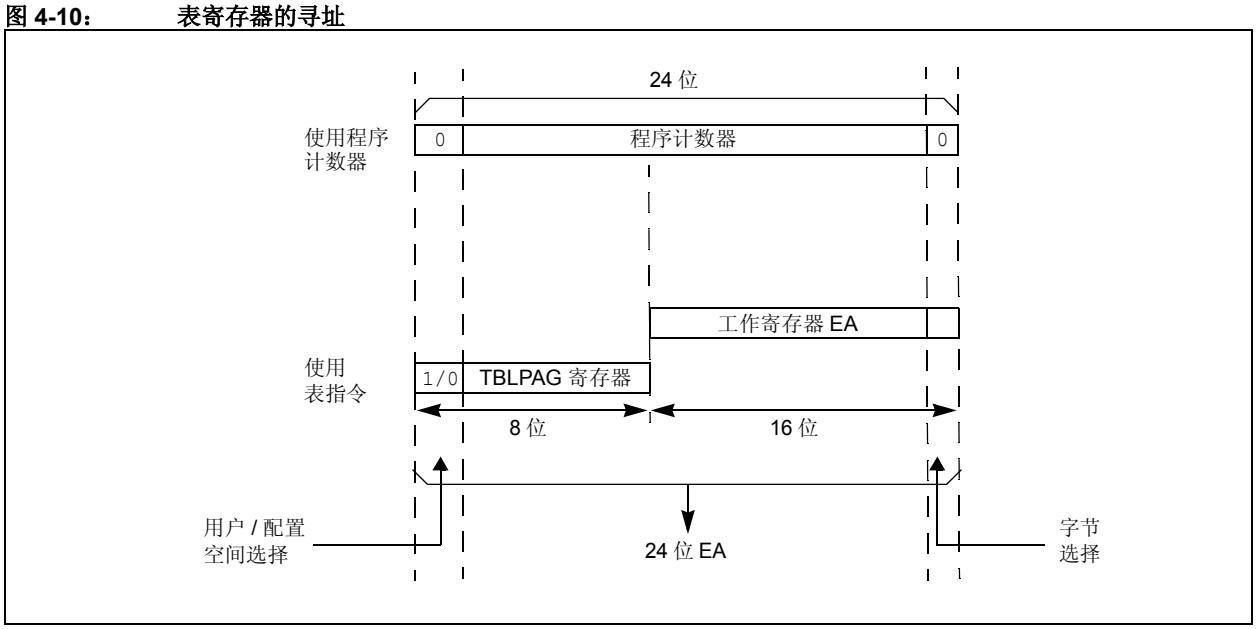
TBLRDH 和 TBLWTH 指令用于读或写程序存储空间的 <23:16> 位。TBLRDH 和 TBLWTH 能以字或字节模式访问程序存储器。因为程序存储器只有 24 位宽，所以 TBLRDH 和 TBLWTH 指令能寻址程序存储器中并不存在的高位字节。该字节被称为“虚拟字节”。读取虚拟字节将总是返回0x00，而对其进行写操作则不起作用。

24 位程序存储器可以视作并排放置的两个 16 位空间，每个空间都有相同的地址范围。因此，TBLRDL 和 TBLWTL 指令可访问 “低”程序存储空间 （PM<15:0>）。 TBLRDH 和 TBLWTH 指令可访问 “高”程序存储空间 （PM<31:16>）。读取或写入 PM<31:24> 可以访问虚拟 （未实现）字节。当在字节模式下使用任何表指令时，表地址的 LSb 将被用作字节选择位。LSb 决定将访问高或低程序存储空间的哪个字节。

图 4-10 显示了如何使用表指令寻址程序存储器。 24 位程序存储器地址由 TBLPAG<7:0> 位以及表指令指定的 W 寄存器中的有效地址 （Effective Address， EA）组成。图 4-10 中给出了 24 位程序计数器以供参考。EA 的高 23 位用于选择程序存储单元。对于字节模式的表指令，W 寄存器 EA 的 LSb 用于选择 16 位程序储存字中要寻址的字节。“1”选择 <15:8> 位，“0”选择 <7:0> 位。 W 寄存器 EA 的 LSb 在字模式下的表指令中会被忽略。

除了指定程序存储器地址外，表指令还指定作为要写入的程序存储器来源或要读取的程序存储器目标的 W 寄存器 （或指向存储单元的 W 寄存器指针）。对于字节模式下的表写操作，源工作寄存器的 <15:8> 位会被忽略。

图 4-10： 表寄存器的寻址



4.6.1 使用表读指令

表读需要两个步骤。首先，使用 TBLPAG 寄存器和一个 W 寄存器建立一个地址指针。然后就可以读取地址单元的程序存储器内容了。

4.6.1.1 使用表指令读程序存储器

以下代码示例显示了如何使用字 / 字节模式下的表指令读程序存储器中的一个字：

例 4-1： 字模式读取

```
; Setup the address pointer to program space
    MOV         #tblpage(PROG_ADDR),W0     ; get table page value
    MOV         W0,TBLPAG                  ; load TBLPAG register
    MOV         #tbloffset(PROG_ADDR),W0   ; load address LS word
; Read the program memory location
    TBLRDH      [W0],W3                    ; Read high byte to W3
    TBLRDL      [W0],W4                    ; Read low word to W4
```

等价的 C 代码

```
int addrOffset;
int VarWord;
int VarWord1;

{
:
:
    TBLPAG = ((PROG_ADDR & 0x7F0000)>>16);
    addrOffset = (PROG_ADDR & 0x00FFFF);
    asm("tblrdh.w [%1], %0" :"=r"(VarWord1)      :"r"(addrOffset));
    asm("tblrdl.w [%1], %0" :"=r"(VarWord)       :"r"(addrOffset));

:
:
}
```

注： 使用之前请保存所有工作寄存器。

例 4-2： 字节模式读取

```
; Setup the address pointer to program space
    MOV         #tblpage(PROG_ADDR),W0     ; get table page value
    MOV         W0,TBLPAG                  ; load TBLPAG register
    MOV         #tbloffset(PROG_ADDR),W0   ; load address LS word
; Read the program memory location
    TBLRDH.B    [W0],W3                    ; Read high byte to W3
    TBLRDL.B    [W0++],W4                  ; Read low byte to W4
    TBLRDL.B    [W0++],W5                  ; Read middle byte to W5
```

等价的 C 代码

```
int  addrOffset;
char VarByte1;
char VarByte2;
char VarByte3;

{
:
:
TBLPAG = ((PROG_ADDR & 0x7F0000)>>16);
addr = (PROG_ADDR & 0x00FFFF);

asm("tblrdl.b [%1], %0" :"=r"(LocalVarByte1) :"r"(addrOffset))      ; // Read low byte
asm("tblrdl.b [%1], %0" :"=r"(LocalVarByte2) :"r"(addrOffset +1))   ;//Read middle byte
asm("tblrdh.b [%1], %0" :"=r"(LocalVarByte3) :"r"(addrOffset))      ; // Read high byte
:
:
}
```

注： 使用之前请保存所有工作寄存器。

在例 4-1 和例 4-2 中，读低字节时的后增操作符会导致工作寄存器中的地址增 1。这会将 EA<0> 设置为 1，以访问第三条写指令中的中间字节。最后的后递增操作将 W0 设置回一个偶数地址，指向下一个程序存储单元。

注： Microchip 的 PIC24F 汇编器提供了 tblpage() 和 tbloffset() 伪指令。这些伪指令会从程序存储器地址值中为表指令选择适当的 TBLPAG 和 W 寄存器值。更多详细信息，请参见 *"MPLAB® ASM 30, MPLAB® LINK30 and Utilities User's Guide"*（DS51317）。

4.6.2 使用表写指令

4.6.2.1 表写保持锁存器

表写指令不会直接写非易失性程序存储器，而会装入用来存储写入数据的保持锁存器。保持锁存器不是存储器映射的，并且只能使用表写指令访问。当所有保持锁存器都被装入后，通过执行一个特殊的指令序列即可开始实际的存储器编程操作。

更多详细信息，请参见具体器件的数据手册。

4.6.2.2 在字 / 字节模式下写入一个程序存储器锁存单元

可以使用以下序列以字模式写入一个程序存储器锁存单元：

例 4-3： 字模式写入

```
; Setup the address pointer to program space
    MOV     #tblpage(PROG_ADDR),W0     ; get table page value
    MOV     W0,TBLPAG                  ; load TBLPAG register
    MOV     #tbloffset(PROG_ADDR),W0   ; load address LS word
; Load write data into W registers
    MOV     #PROG_LOW_WORD,W2
    MOV     #PROG_HI_BYTE,W3
; Perform the table writes to load the latch
    TBLWTL  W2,[W0]
    TBLWTH  W3,[W0++]
```

等价的 C 代码

```
int VarWord1 = 0xXXXX;
int VarWord2 = 0xXXXX;
int addrOffset;
{
:
:

TBLPAG =  ((PROG_ADDR & 0x7F0000)>>16);
addrOffset = (PROG_ADDR & 0x00FFFF);
asm("tblwtl %1, [%0]" :"=r"(addrOffset) :"d"(VarWord))      ;
asm("tblwth %1, [%0]" :"=r"(addrOffset) :"d"(VarWord1))     ;
:
:
}
```

注： 使用之前保存所有工作寄存器。

在此示例中， W3 高字节的内容无关紧要，因为该数据将会被写入到虚拟字节单元中。在第二条 TBLWTH 指令后， W0 后增 2，以准备写入下一个程序存储单元。

要以字节模式写入一个程序存储器锁存单元，可以使用以下代码序列：

例 4-4：　　字节模式写入

```
; Setup the address pointer to program space
    MOV         #tblpage(PROG_ADDR),W0     ; get table page value
    MOV         W0,TBLPAG                  ; load TBLPAG register
    MOV         #tbloffset(PROG_ADDR),W0   ; load address LS word
; Load data into working registers
    MOV         #LOW_BYTE,W2
    MOV         #MID_BYTE,W3
    MOV         #HIGH_BYTE,W4
; Write data to the latch
    TBLWTH.B    W4,[W0]                    ; write high byte
    TBLWTL.B    W2,[W0++]                  ; write low byte
    TBLWTL.B    W3,[W0++]                  ; write middle byte
```

等价的 C 代码

```
char VarByte1 = 0xXX;
char VarByte2 = 0xXX;
char VarByte3 = 0xXX;

{
:
:

TBLPAG =  ((PROG_ADDR & 0x7F0000)>>16);
addr = (PROG_ADDR & 0x00FFFF);
asm("tblwtl.b %1, [%0]" :"=r"(addr) :"d"(VarByte1)) ;//Low Byte
asm("tblwth.b %1, [%0]" :"=r"(addr) :"d"(VarByte3)) ;//Upper Byte
addr++;
asm("tblwtl.b %1, [%0]" :"=r"(addr) :"d"(VarByte2)) ;//Middle Byte
:
}
```

注：　　使用之前保存所有工作寄存器。

在上面的代码示例中，在写入低字节时的后增操作符会导致 W0 中的地址增 1。这会将 EA<0> 设置为 1，以访问第三条写指令中的中间字节。最后的后递增操作将 W0 设置回一个偶数地址，指向下一个程序存储单元。

4.6.3 运行时自编程 （RTSP）

RTSP 允许用户代码修改闪存程序存储器的内容。 RTSP 是通过使用 TBLRD （表读）、 TBLWT （表写）指令和 NVM 控制寄存器实现的。通过 RTSP，用户可以一次在程序存储器中擦除 8 行（row）（64x8 = 512 条指令），也可以在程序存储器中一次写入 1 行 （64 条指令）。

4.6.3.1 RTSP 工作原理

PIC24F 闪存程序存储器阵列是由多行组成的，每行 64 条指令或 192 字节。RTSP 可以让用户一次擦除 8 行 （512 条指令），也可以一次编程 64 条指令。从程序存储器开始处，8 行擦除块和 1 行写块分别在 1536 字节和 192 字节的边界上边缘对齐。

程序存储器实现了保持缓冲器，可包含 64 条指令的编程数据。实际编程操作前，必须将要写入的数据连续装入缓冲器。装入的指令字必须来自 64 个边界的组合。

RTSP 编程的基本步骤是先建立一个表指针，然后执行一系列 TBLWT 指令以装入缓冲器。编程是通过将 NVMCON 寄存器中的控制位置 1 执行的。指令的装入总共需要 64 条 TBLWTL 和 TBLWTH 指令。

所有表写操作都是单字写入 （2 个指令周期），因为只写入缓冲器。每行的编程都需要一个编程周期。

注：	行、块和保持锁存器的数目可能随器件不同而有所不同；实际数目请参见具体器件的数据手册。

4.6.4 控制寄存器

有两个 SFR 用于读写程序闪存存储器，它们是：NVMCON 和 NVMKEY。 NVMCON 寄存器（寄存器 4-1）控制擦除哪些块、对哪种类型的存储器编程，以及编程周期的开始。

NVMKEY 是用于写保护的只写寄存器。要开始编程或擦除序列，用户必须连续地将 55h 和 AAh 写入 NVMKEY 寄存器。

4.6.4.1 NVMCON 寄存器

NVMCON 寄存器是闪存和 EEPROM 编程 / 擦除操作的主控制寄存器。该寄存器选择执行的将是擦除还是编程操作，并用于开始编程或擦除周期。

寄存器 4-1 所示为 NVMCON 寄存器。 NVMCOM 的低字节用于配置将执行的 NVM 操作类型。

寄存器 4-1： NVMCOM：非易失性闪存存储器控制寄存器

R/SO-0(1)	R/W-0(1)	R/W-0(1)	U-0	U-0	U-0	U-0	U-0
WR	WREN	WRERR	—	—	—	—	—
bit 15							bit 8

U-0	R/W-0(1)	U-0	U-0	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)
—	ERASE	—	—	NVMOP3(2)	NVMOP2(2)	NVMOP1(2)	NVMOP0(2)
bit 7							bit 0

图注：	SO = 只可置 1 的位		
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = 复位时的值	1 = 置 1	0 = 清零	x = 未知

bit 15 **WR：**写控制位 (1)

1 = 启动闪存存储器编程或擦除操作。操作是自定时的，一旦该操作完成，该位即由硬件清零。
0 = 编程或擦除操作已完成且无效

bit 14 **WREN：**写使能位 (1)

1 = 使能闪存编程 / 擦除操作
0 = 禁止闪存编程 / 擦除操作

bit 13 **WRERR：**写序列错误标志位 (1)

1 = 不合法的编程或擦除序列尝试，或者发生终止 （试图将 WR 位置 1 时自动置 1 该位）
0 = 编程或擦除操作正常完成

bit 12-7 **未实现：**读为 0

bit 6 **擦除：**擦除 / 编程使能位 (1)

1 = 在下一条 WR 命令时执行 NVMOP3:NVMOP0 指定的擦除操作
0 = 在下一条 WR 命令时执行 NVMOP3:NVMOP0 指定的编程操作

bit 5-4 **未实现：**读为 0

bit 3-0 **NVMOP3:NVMOP0：**NVM 操作选择位 (2)

1111 = 存储器批量擦除操作 （ERASE = 1）或无操作 （ERASE = 0） (3)
0011 = 存储器字编程操作 （ERASE = 0）或无操作 （ERASE = 1）
0010 = 存储器页擦除操作 （ERASE = 1）或无操作 （ERASE = 0）
0001 = 存储器行编程操作 （ERASE = 0）或无操作 （ERASE = 1）

注 1：这些位只能在 POR 时复位。
2：NVMOP3:NVMOP0 的所有其他组合均未实现。
3：此操作仅在 ICSP™ 模式下可用。

4.6.4.2 NVMKEY 寄存器

NVMKEY 是一个只写寄存器，用于防止闪存存储器的误写 / 误擦除。要开始编程或擦除序列，必须严格按照如下顺序执行下列步骤：

1. 将 0x55 写入 NVMKEY。
2. 将 0xAA 写入 NVMKEY。
3. 执行两条 NOP 指令。

在此序列后，就可以在一个指令周期中写入 NVMCON 寄存器。在多数情况下，用户只需要将 NVMCON 寄存器中的 WR 位置 1 就可以开始编程或擦除周期。在解锁序列中应禁止中断。下面的代码示例显示了解锁序列是如何执行的：

例 4-5： 执行解锁序列

```
    PUSH    SR              ; Disable interrupts, if enabled
    MOV     #0x00E0,W0
    IOR     SR

    MOV     #0x55,W0
    MOV     #0xAA,W0
    MOV     W0,NVMKEY
    MOV     W0,NVMKEY       ; NOP not required
    BSET    NVMCON,#WR      ; Start the program/erase cycle
    NOP
    NOP
    POP     SR              ; Re-enable interrupts
```

等价的 C 代码

```
    NVMKEY = 0x55;
    NVMKEY = 0xAA;
    NVMCONbits.WR=1;
    Nop();
    Nop();
```

注： 使用之前保存所有工作寄存器。

更多编程示例，请参见**第 4.7 节 “闪存编程操作”**。

4.7 闪存编程操作

在 RTSP 模式下对内部闪存进行编程或擦除操作必须使用完整的编程序列。编程操作持续时间的标称值为 4 ms(1)，处理器将暂停 （等待）直到编程操作完成。将 WR 位 （NVMCON<15>）置 1 将开始操作，当操作完成时 WR 位会自动清零。

注 1：编程时间可能随器件不同而有所不同；精确值请参见具体器件的数据手册。

闪存编程操作使用以下非易失性存储器 （Nonvolatile Memory， NVM）控制寄存器进行控制：

- NVMCON
- NVMKEY

4.7.1 闪存程序存储器编程算法

用户可以一次对闪存存储器的一行进行编程。要这样编程，必须擦除包含要编程的行的 8 行擦除块。一般过程如下：

1. 读取程序存储器的 8 行 （512 条指令），并存储在数据 RAM 中。
2. 用所需的新数据更新 RAM 中的程序数据。
3. 擦除该区块：
   a） 将 NVMOP 位 （NVMCOM<3:0>）设置为 0010 以配置块擦除。将 ERASE （NVMCOM<6>）和 WREN （NVMCOM<14>）位置 1。
   b） 将要擦除的块的起始地址写入 TBLPAG 和 W 寄存器。
   c） 将 55h 写入 NVMKEY。
   d） 将 AAh 写入 NVMKEY。
   e） 将 WR 位置 1 （NVMCOM <15>）。擦除周期开始，在擦除周期中 CPU 会暂停。当擦除结束时， WR 位会自动清零。
4. 将数据 RAM 中的前 64 条指令写入程序存储缓冲器 （见**第 4.5 节 “写程序存储器”**）。
5. 将程序块写入闪存存储器：
   a） 将 NVMOP 位设置为 0001 以配置行编程。将 ERASE 位清零，将 WREN 位置 1。
   b） 将 55h 写入 NVMKEY。
   c） 将 AAh 写入 NVMKEY。
   d） 将 WR 位置 1。编程周期开始，在写周期中 CPU 会暂停。写入闪存存储器结束时，WR 位会自动清零。
6. 通过递增 TBLPAG 中的值，使用数据 RAM 中的下一批可用的 64 条指令重复步骤 4 和 5，直到全部 512 条指令写回闪存存储器中。

为避免意外操作，任何擦除或编程操作都必须使用 NVMKEY 的写操作启动序列。执行编程命令后，用户必须等待编程时间直到编程完成。编程序列开始后的两条指令应为 NOP，如**第 4.6.4.2 节 “NVMKEY 寄存器”**中所示。

注 1：闪存存储器编程的完整参考代码，请参见具体器件的数据手册。
2：行、块和保持锁存器的数目可能随器件不同而有所不同；实际数目请参见具体器件的数据手册。

4.8 寄存器映射

表 4-1 中提供了与 PIC24F 程序存储器相关的特殊功能寄存器汇总。

表 4-1： 与程序存储器相关的特殊功能寄存器 (1)

寄存器名称	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	所有复位时 (2)
TBLPAG	—	—	—	—	—	—	—	—	表页地址指针					—	—	—	0000
NVMCON	WR	WREN	WRERR	—	—	—	—	—	—	ERASE	—	—	NVMOP3	NVMOP2	NVMOP1	NVMOP0	0000
NVMKEY	—	—	—	—	—	—	—	—	NVMKEY<7:0>								0000

图注： — = 未实现，读为 0。复位值以十六进制显示。

注 1： 有关具体存储器映射的详细信息，请参见器件数据手册。

2： 所示复位值仅适用于 POR。其他复位状态下的值取决于复位时存储器写操作或擦除操作的状态。

4.9 相关应用笔记

本节列出了与手册本章内容相关的应用笔记。这些应用笔记可能并不是专为 PIC24F 器件系列而编写的，但其概念是相关的，通过适当修改即可使用，但在使用中可能会受到一定的限制。当前与程序存储器相关的应用笔记有：

标题	应用笔记编号

目前没有相关的应用笔记。

注： 如需获取更多 PIC24F 系列器件的应用笔记和代码示例，请访问 Microchip 网站（www.microchip.com）。

4.10 版本历史

版本 A （2007 年 1 月）

这是本文档的初始发行版。

