第 2 章 CPU

目录

本章包括下列主题：

2.1 简介

PIC24F CPU 模块采用 16 位（数据）改良的哈佛架构，并带有增强型指令集。CPU 具有 24 位指令字，指令字带有长度可变的操作码字段。程序计数器（Program Counter，PC）为 24 位宽，可以寻址高达 4M x 24 位的用户程序存储空间。单周期指令预取机制用来帮助维持吞吐量并提供可预测的指令执行过程。除了改变程序流的指令、双字移动（MOV.D）指令和表指令以外，所有指令都在单个周期内执行。模块使用 REPEAT 指令支持无开销的程序循环结构，该指令在任何时候都可被中断。

PIC24F 器件在编程模型中有 16 个 16 位工作寄存器。每个工作寄存器都可以充当数据、地址或地址偏移寄存器。第 16 个工作寄存器（W15）作为软件堆栈指针工作，用于中断和调用。第 15 个工作寄存器（W14）可用作堆栈帧指针，与 LNK 和 UNLK 指令配合使用。

可以选择将数据空间存储器映射的高 32 KB 映射到由 8 位程序空间可视性页（Program Space Visibility Page，PSVPAG）寄存器定义的任何 16K 字程序边界内的程序空间内。数据空间到程序空间的映射功能让任何指令都能像访问数据空间一样访问程序空间。关于程序空间可视性的更多信息，请参见**第 4.4 节 “来自数据空间的程序空间可视性”**。

指令集架构（Instruction Set Architecture，ISA）在 PIC18F 指令集架构的基础上有显著增强，但仍在可接受程度上保持了向后兼容性。所有 PIC18F 指令和寻址模式都能直接得到支持或通过简单的宏得到支持。许多 ISA 增强功能都是为了提高编译器效率而做出的。

内核支持固有（无操作数）、相对、立即数和存储器直接寻址模式，以及 3 组寻址模式（MODE1、MODE2 和 MODE3）。所有模式都支持寄存器直接和各种寄存器间接寻址模式。每组都提供最多 7 种寻址模式。指令根据其功能要求，与预定义的寻址模式相关联。

还有一种“有符号 10 位偏移寄存器间接”寻址模式，专用于两条特殊移动指令 LDWLO 和 STWLO。更多详细信息，请参见**第 32 章 “指令集”**。

对于大多数指令，内核能在每个指令周期内执行一次数据（或程序数据）存储器读操作、一次工作寄存器（数据）读操作、一次数据存储器写操作和一次程序（指令）存储器读操作。因此可以支持 3 个参数的指令，使 A + B = C 操作能在单个周期内执行。

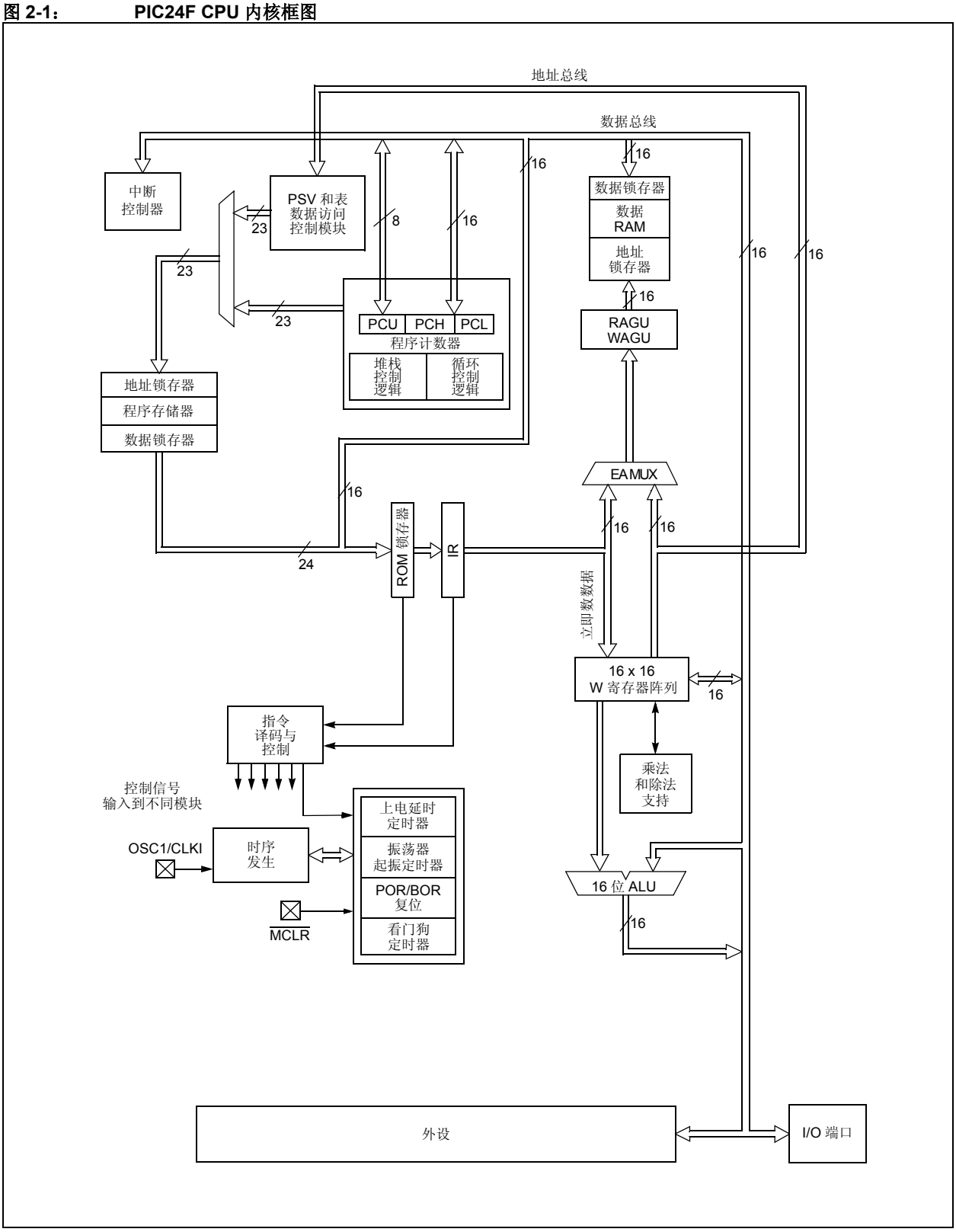
该模块具备一个高速 17 位 X17 位乘法器，显著提高了内核算术能力和吞吐量。乘法器支持有符号、无符号和混合模式的 16 位 X16 位或 8 位 X8 位整数相乘。所有乘法指令都在单个周期内执行。

支持迭代不恢复（Non-restoring）除法算法的整数除法支持硬件增强 16 位 ALU。它与 REPEAT 指令循环机制和一些迭代除法指令配合使用，支持 32 位（或 16 位）除以 16 位整数的有符号和无符号除法。所有除法操作都需要 19 个周期来完成，但可以在任何周期边界上中断。

PIC24F 具有向量异常（Exception）机制，带有最多 8 个不可屏蔽陷阱源和中断源。可以为每个中断源分配 7 个优先级之一。

图 2-1 给出了 CPU 的框图。

图 2-1： PIC24F CPU 内核框图



2.2 编程模型

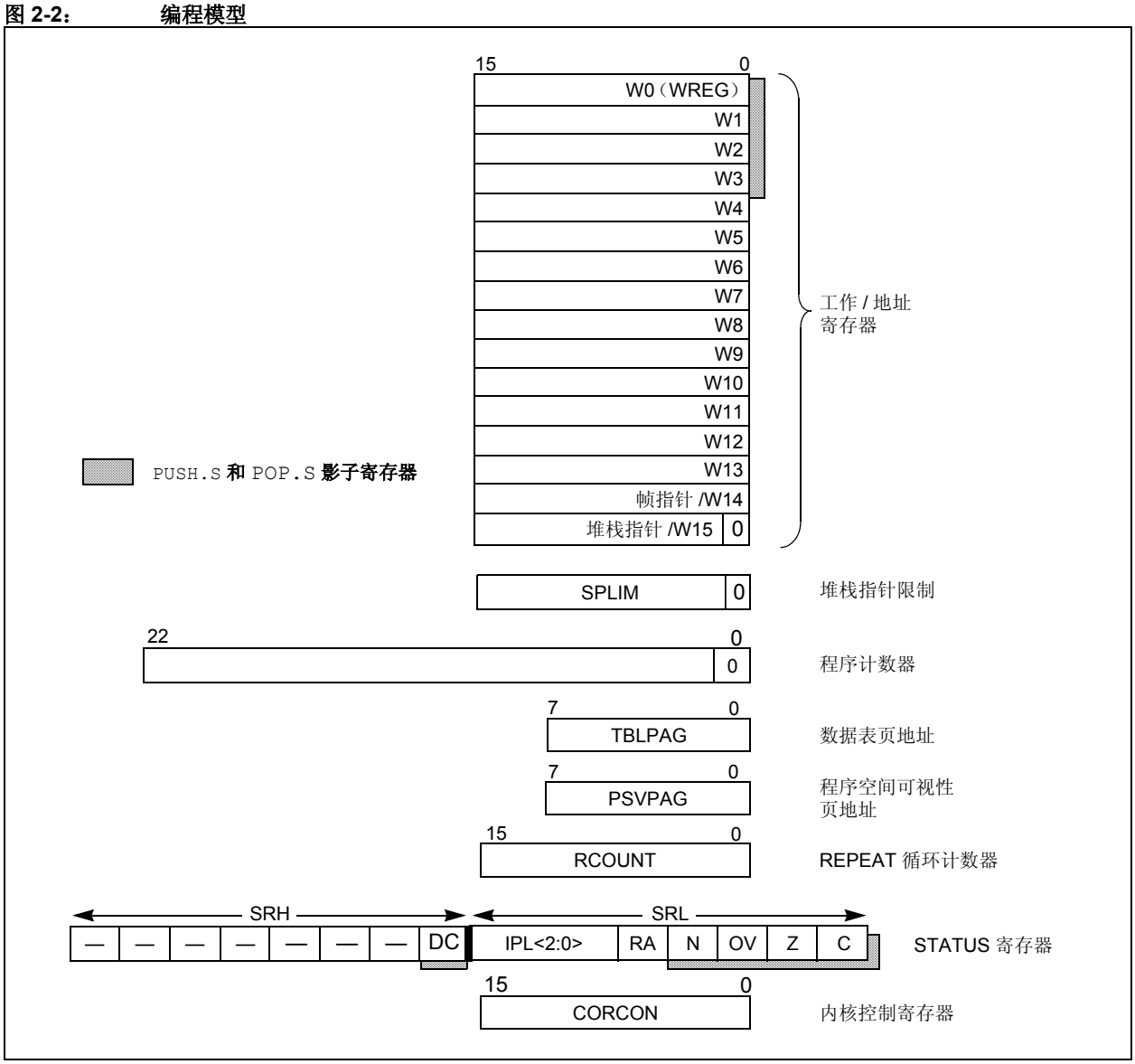
图 2-2 中所示为 PIC24F 的编程模型。编程模型中的所有寄存器都是存储器映射的，并且可以由指令直接控制。表 2-1 中提供了对每个寄存器的说明。

表 2-1： 编程模型的寄存器说明

寄存器名称	说明
W0 到 W15	工作寄存器阵列
PC	23 位程序计数器
SR	ALU STATUS 寄存器
SPLIM	堆栈指针极限值寄存器
TBLPAG	表存储器页地址寄存器
PSVPAG	程序空间可视性页地址寄存器
RCOUNT	REPEAT 循环计数寄存器
CORCON	CPU 控制寄存器

与编程模型相关的所有寄存器都是存储器映射的，如表 2-5 中所示。

图 2-2： 编程模型



2.2.1 工作寄存器阵列

16 个工作寄存器 （W）可以作为数据、地址或地址偏移寄存器。 W 寄存器的功能由访问它的指令的寻址模式决定。

PIC24F 指令集可被分成两种指令类型：寄存器和文件寄存器指令。寄存器指令可以把每个 W 寄存器用作数据值或地址偏移值。例如：

例 2-1： 寄存器指令

```
MOV     W0, W1          ; move contents of W0 to W1
MOV     W0, [W1]        ; move W0 to address contained in W1
ADD     W0, [W4], W5    ; add contents of W0 to contents pointed
                        ; to by W4. Place result in W5.
```

2.2.1.1 W0 和文件寄存器指令

W0 是一个特殊的工作寄存器，因为它是可在文件寄存器指令中使用的惟一的工作寄存器。文件寄存器指令对在指令操作码和 W0 中包含的特定存储器地址进行操作。在文件寄存器指令中，W1 到 W15 不可被指定为目标寄存器。

文件寄存器指令对只有一个 W 寄存器的现有 PIC® 器件提供向后兼容性。在汇编器语法中使用标号 “WREG”来表示文件寄存器指令中的 W0。例如：

例 2-2： 文件寄存器指令

```
MOV     WREG, 0x0100        ; move contents of W0 to address 0x0100
ADD     0x0100, WREG        ; add W0 to address 0x0100, store in W0
```

注： 关于寻址模式和指令语法的完整说明，请参见 “*dsPIC30F Programmer’s Reference Manual*”（DS70030）。

2.2.1.2 W 寄存器存储器映射

由于 W 寄存器是存储器映射的，因此可在文件寄存器指令中访问 W 寄存器，示例如下：

例 2-3： 在文件寄存器指令中访问 W 寄存器

```
    MOV     0x0004, W10         ; equivalent to MOV W2, W10
```

其中：

0x0004 是 W2 存储器中的地址

此外，还可执行一条试图将 W 寄存器同时用作地址指针和操作数目标的指令。例如：

例 2-4： W 寄存器用作地址指针和操作数目标

```
    MOV     W1, [W2++]
```

其中：

```
    W1 = 0x1234
    W2 = 0x0004             ;[W2] addresses W2
```

在例 2-4 中，W2 的内容是 0x0004。由于 W2 被用作地址指针，它指向存储器中的 0x0004 单元。W2 也映射到存储器中的该地址空间。虽然这是不太可能发生的事件，但不到运行时不可能检测到。 PIC24F 确保数据写操作将占据主导，使上面示例中 W2 的结果为 0x1234。

2.2.1.3 W 寄存器和字节模式指令

把W寄存器阵列当作目标寄存器的字节指令只影响目标寄存器的最低有效字节。因为工作寄存器是存储器映射的，所以可以通过对数据存储空间进行字节宽度的访问来控制工作寄存器的最低*和*最高有效字节。

2.2.2 影子寄存器

如表 2-5 中所示，许多寄存器都有相关的影子寄存器。影子寄存器用作临时的保持寄存器，可在发生某些事件时，在它与其主寄存器之间传送内容。影子寄存器都不能直接访问。 PUSH.S 和 POP.S 影子规则应用于传送进出影子的寄存器。

2.2.2.1 PUSH.S 和 POP.S 影子寄存器

在执行函数调用或中断服务程序 （Interrupt Service Routine， ISR）时， PUSH.S 和 POP.S 指令可用于快速的现场保存 / 恢复。 PUSH.S 指令会将下列寄存器的值传送到它们各自的影子寄存器中：

- W0...W3
- SR （仅 N、 OV、 Z、 C 和 DC 位）

POP.S指令会将值从影子寄存器恢复到这些寄存器单元。例2-5中显示了使用PUSH.S和POP.S指令的代码示例。

例 2-5： PUSH.S 和 POP.S 指令

```
MyFunction:
    PUSH.S                      ; Save W registers, MCU status
    MOV     #0x03, W0           ; load a literal value into W0
    ADD     RAM100              ; add W0 to contents of RAM100
    BTSC    SR, #Z              ; is the result 0?
    BSET    Flags, #IsZero      ; Yes, set a flag
    POP.S                       ; Restore W regs, MCU status
    RETURN
```

PUSH.S 指令会改写先前保存在影子寄存器中的内容。影子寄存器深度只有一级，所以如果多个软件任务使用影子寄存器，必须小心。

用户必须确保使任何使用影子寄存器的任务均不会被同样使用该影子寄存器且具有更高优先级的任务中断。如果允许较高优先级的任务中断较低优先级的任务，较低优先级任务保存在影子寄存器中的内容将被较高优先级任务改写。

2.2.3 未初始化的 W 寄存器的复位

W 寄存器阵列 （除 W15 之外）在发生所有复位时被清零，并且在被写入前视为未经初始化。试图把未初始化的寄存器用作地址指针将会复位器件。更多详细信息，请参见**第 7 章 “复位”**（请访问 Microchip 网站 www.microchip.com 查看是否提供）。

必须执行字写操作来初始化 W 寄存器。字节写操作不会影响初始化检测逻辑。

2.3 软件堆栈指针

W15 作为专用的软件堆栈指针并被异常处理、子程序调用和返回自动修改。但是， W15 可以被任何指令以与所有其他 W 寄存器相同的方式引用。这样就简化了对堆栈指针的读、写和控制（如创建堆栈帧）。

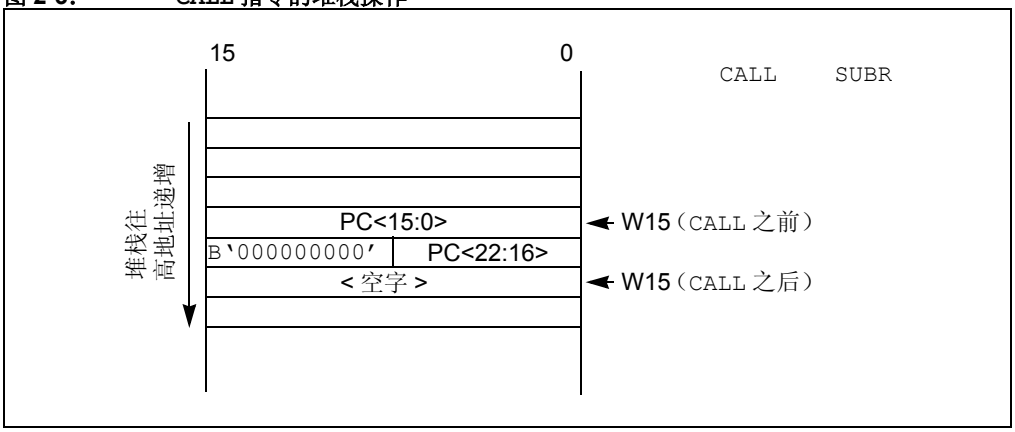
注： 为了防止偏离的堆栈访问， W15<0> 被硬件固定为 0。

所有复位均将 W15 初始化为 0x0800。此地址确保在所有 PIC24F 器件中，堆栈指针 （SP）将指向有效的 RAM 并允许不可屏蔽异常陷阱 （在 SP 被用户软件初始化前可能发生）使用堆栈。在初始化期间，用户可以将 SP 重新编程以指向数据空间内的任何单元。

堆栈指针总是指向第一个可用的空字并从低地址到高地址填充软件堆栈。堆栈出栈（读）时，堆栈指针先减；堆栈进栈 （写）时，堆栈指针后加，如图 2-3 中所示。

当 PC 压入堆栈时，PC<15:0> 位被压入第一个可用的堆栈字，然后 PC<22:16> 位被压入第二个可用的堆栈单元。对于任何 CALL 指令执行期间的 PC 进栈，进栈前 PC 的 MSB 是以零扩展的，如图 2-3 中所示。异常处理期间， PC 的 MSB 与 CPU STATUS 寄存器 SR 的低 8 位相连。这样就使 SRL 的内容在中断处理期间能被自动保存。

图 2-3： CALL 指令的堆栈操作



2.3.1 软件堆栈示例

使用 PUSH 和 POP 指令可控制软件堆栈。 PUSH 和 POP 指令相当于将 W15 用作目标指针的 MOV 指令。例如，要把 W0 的内容压入堆栈，可通过：

```
PUSH    W0
```

此语法相当于：

```
MOV     W0, [W15++]
```

要把栈顶 （TOS）的内容返回 W0，可通过：

```
POP     W0
```

此语法相当于：

```
MOV     [--W15], W0
```

图 2-4 到图 2-7 给出了如何使用软件堆栈的示例。图 2-4 所示为器件初始化时的软件堆栈。 W15 已经初始化为 0x0800。此外，此示例假设 0x5A5A 和 0x3636 这两个值已被分别写入 W0 和 W1。图 2-5 中堆栈第一次进栈，W0 中包含的值被复制到堆栈中。W15 自动更新以指向下一个可用的堆栈单元 （0x0802）。在图 2-6 中， W1 的内容被压入堆栈。在图 2-7 中，数据出栈并且栈顶值（先前从 W1 压入）被写入 W3。

图 2-4： 器件复位时的堆栈指针

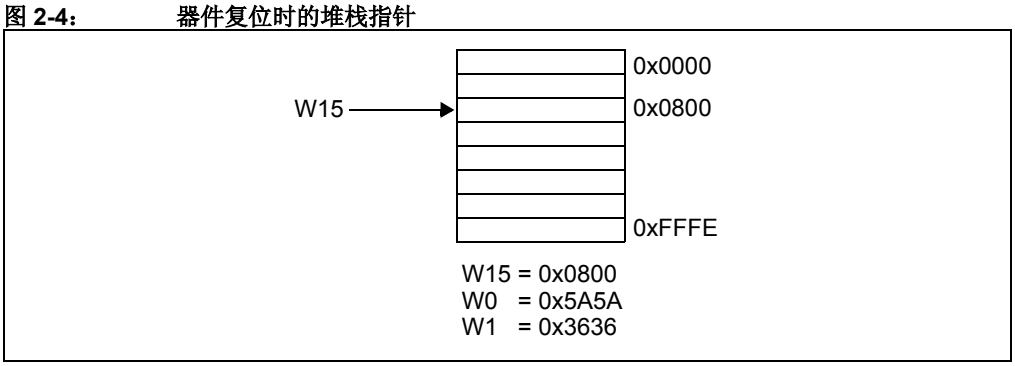


图 2-5： 第一次执行 PUSH 指令后的堆栈指针

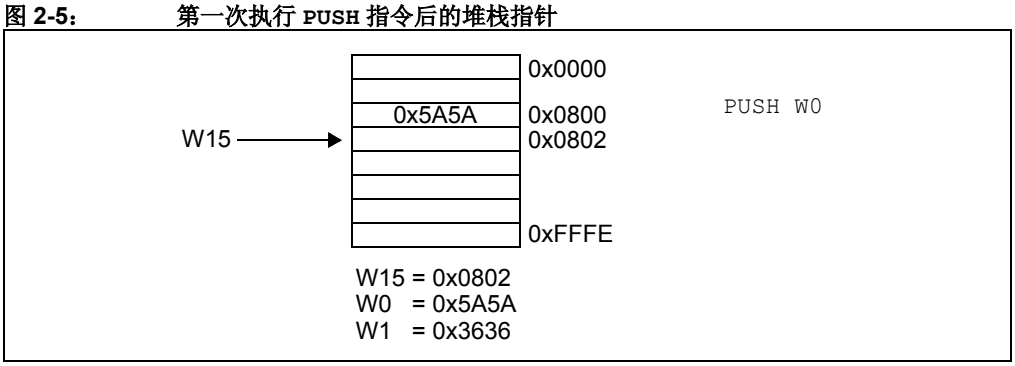


图 2-6： 第二次执行 PUSH 指令后的堆栈指针

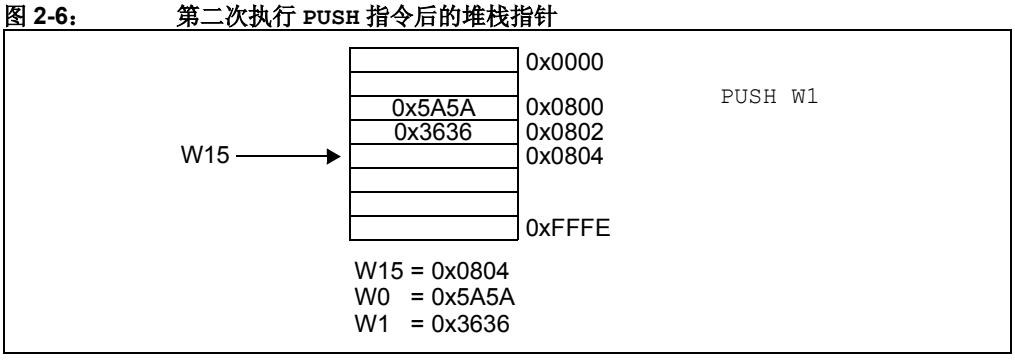
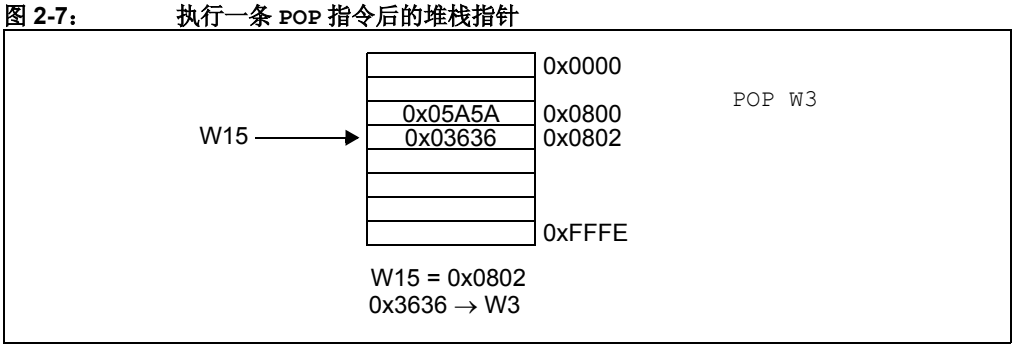


图 2-7：　　执行一条 POP 指令后的堆栈指针



2.3.2　W14 软件堆栈帧指针

帧是堆栈中用户定义的存储器段，供单个子程序使用。 W14 是特殊工作寄存器，因为通过使用 LNK （link，连接）和 ULNK （unlink，不连接）指令可以把它用作堆栈帧指针。当不用作堆栈帧指针时， W14 可被指令当作普通的工作寄存器使用。

关于将 W14 用作堆栈帧指针的软件示例，请参见“*dsPIC30F Programmer's Reference Manual*”（DS70030）。

2.3.3　堆栈指针上溢

有一个与堆栈指针相关的堆栈指针极限寄存器（SPLIM），复位时为 0x0000。SPLIM 是一个 16 位寄存器，但 SPLIM<0> 被固定为 0，因为所有的堆栈操作都必须按字对齐。

直到一个字写入 SPLIM 后才使能堆栈上溢检查，在此之后，只能通过器件复位禁止堆栈上溢检查。所有将 W15 用作源或目标寄存器而产生的有效地址将与 SPLIM 中的值作比较。如果堆栈指针 （W15）的内容比 SPLIM 寄存器的内容大 2，并且执行了进栈操作，将不会产生堆栈错误陷阱 （Stack Error Trap）。堆栈错误陷阱将在随后的进栈操作时产生。所以，例如，如果想要在堆栈指针递增超出 RAM 中的地址 0x2000 时引起堆栈错误陷阱，可将 SPLIM 初始化为 0x1FFE。

注：	任何使用 W15 寄存器的内容来产生有效地址（Effective Address，EA）的指令均有可能产生堆栈错误陷阱。因此，如果 W15 的内容比 SPLIM 寄存器的内容大 2，并且执行了一条 CALL 指令或发生了中断，那么将产生堆栈错误陷阱。

如果已经使能了堆栈上溢检查，则当 W15 有效地址计算越过了数据空间的末尾 （0xFFFF）时，也将产生堆栈错误陷阱。

注：	对堆栈指针极限寄存器 SPLIM 的写操作后面不应紧跟一个使用 W15 的间接读操作。

关于堆栈错误陷阱的更多信息，请参见**第 8 章 “中断”**。

2.3.4　堆栈指针下溢

发生复位时，堆栈初始化为 0x0800。如果堆栈指针地址小于 0x0800，将产生堆栈错误陷阱。

注：	通常，数据空间中 0x0000 和 0x07FF 之间的单元预留给内核和外设的特殊功能寄存器。

2.4 CPU 寄存器说明

2.4.1 SR：CPU STATUS 寄存器

PIC24F 的 CPU 有一个 16 位 STATUS 寄存器（SR），它的低字节称为低 STATUS 寄存器（Lower STATUS Register， SRL）。 SR 的高字节称为 SRH。寄存器 2-1 中给出了对 SR 的详细说明。

SRL 包含了所有的 MCU ALU 操作状态标志，以及 CPU 中断优先级状态位 IPL<2:0> 和 REPEAT 循环有效状态位 RA（SR<4>）。异常处理期间， SRL 与 PC 的 MSB 相连形成一个完整的字值，然后将该字值压入堆栈。

SRH 只包含半进位位 DC （SR<8>）。

SR 位可读 / 写，但以下各位例外：

1. RA 位 （SR<4>）：RA 是只读位。
2. IPL<2:0>：寄存器被禁止 （NSTDIS = 1）时， IPL<2:0> 位变为只读位。

注： *“dsPIC30F Programmer’s Reference Manual”*（DS70030）中提供了对受各条指令影响的 SR 位的说明。

寄存器 2-1：　　SR：CPU STATUS 寄存器

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	DC
bit 15							bit 8

R/W-0(2)	R/W-0(2)	R/W-0(2)	R-0	R/W-0	R/W-0	R/W-0	R/W-0
IPL<2:0>			RA	N	OV	Z	C
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现，读为 0	
-n = POR 值	1 = 置 1	0 = 清零	x = 未知

bit 15-9　**未实现：**读为 0

bit 8　**DC：**MCU ALU 半进位 / 借位位
1 = 结果的第4个低位（对于字节大小的数据）或第8个低位（对于字大小的数据）发生了向高位的进位
0 = 结果的第4个低位（对于字节大小的数据）或第8个低位（对于字大小的数据）未发生向高位的进位

bit 7-5　**IPL<2:0>：**CPU 中断优先级状态位 (1)
111 = CPU 中断优先级是 7 （15），禁止用户中断
110 = CPU 中断优先级是 6 （14）
101 = CPU 中断优先级是 5 （13）
100 = CPU 中断优先级是 4 （12）
011 = CPU 中断优先级是 3 （11）
010 = CPU 中断优先级是 2 （10）
001 = CPU 中断优先级是 1 （9）
000 = CPU 中断优先级是 0 （8）

bit 4　**RA：**REPEAT 循环有效位
1 = 正在进行 REPEAT 循环
0 = 不在进行 REPEAT 循环

bit 3　**N：**MCU ALU 负标志位
1 = 结果为负
0 = 结果为非负 （零或正）

bit 2　**OV：**MCU ALU 溢出标志位
此位用于有符号的算术运算 （二进制补码）。它表明数量级的溢出而导致符号位改变状态。
1 = 有符号算术运算中发生溢出 （本次运算）
0 = 未发生溢出

bit 1　**Z：**MCU ALU 零标志位
1 = 上个操作结果为零
0 = 上个操作结果不为零

bit 0　**C：**MCU ALU 进位 / 借位位
1 = 结果的最高有效位发生了进位
0 = 结果的最高有效位未发生进位

注　1： IPL<2:0> 位与 IPL3 位 （CORCON<3>）相连以形成 CPU 中断优先级。括号中的值表示 IPL3 = 1 时的 IPL。当 IPL3 = 1 时，禁止用户中断。
2： 当 NSTDIS = 1 （INTCON1<15>）时， IPL<2:0> 状态位为只读位。

寄存器 2-2：　　CORCON：内核控制寄存器

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	U-0	R/C-0	R/W-0	U-0	U-0
—	—	—	—	IPL3(1)	PSV	—	—
bit 7							bit 0

图注：	C = 可清零位		
R = 可读位	W = 可写位	U = 未实现，读为 0	
-n = POR 值	1 = 置 1	0 = 清零	x = 未知

bit 15-4　**未实现：**读为 0

bit 3　**IPL3：**CPU 中断优先级状态位 (1)
1 = CPU 中断优先级高于 7
0 = CPU 中断优先级等于或低于 7

bit 2　**PSV：**数据空间中的程序空间可视性使能位
1 = 程序空间在数据空间中可视
0 = 程序空间在数据空间中不可视

bit 1-0　**未实现：**读为 0

注　1： 当 IPL3 = 1 时，禁止用户中断。

2.4.2 其他 PIC24F CPU 控制寄存器

以下所列的寄存器与 PIC24F CPU 内核相关，本手册的其他章节会对它们进行更详细的说明。

2.4.2.1 TBLPAG：表页地址指针

TBLPAG 寄存器用于在表读和表写操作过程中保存程序存储器地址的高 8 位。表指令用于在程序存储空间和数据存储空间之间传送数据。更多详细信息，请参见**第 4 章 “程序存储器”**（请访问 Microchip 网站 www.microchip.com 查看是否提供）。

2.4.2.2 PSVPAG：程序存储器可视性页地址指针

程序空间可视性允许用户将程序存储空间的 32 KB 区域映射到数据地址空间的高 32 KB。此特性允许通过在数据存储器上操作的 PIC24F 指令对常数数据进行透明访问。PSVPAG 寄存器选择映射到数据地址空间的程序存储空间的 32 KB 区域。关于 PSVPAG 寄存器的更多信息，请参见**第 4 章 “程序存储器”**（请访问 Microchip 网站 www.microchip.com 查看是否提供）。

2.4.2.3 DISICNT：禁止中断计数寄存器

DISI 指令使用 DISICNT 寄存器将优先级为 1-6 的中断在指定的几个周期内禁止。更多信息，请参见**第 8 章 “中断”**。

2.5 算术逻辑单元（ALU）

PIC24F ALU 为 16 位宽，能进行加、减、单位移位和逻辑运算。除非另外声明，算术运算一般是以二进制补码方式进行的。根据不同的操作，ALU 可能会影响 SR 寄存器中的进位 / 借位（C）、零（Z）、负（N）、溢出（OV）和半进位 / 借位（DC）状态位的值。在减法操作中， C 和 DC 状态位分别作为借位和半借位位。

根据所使用的指令模式，ALU 可以执行 8 位或 16 位操作。根据指令的寻址模式，ALU 操作的数据可以来自 W 寄存器阵列或数据存储器。同样，ALU 的输出数据可以被写入 W 寄存器阵列或数据存储单元。

关于受到指令、寻址模式和 8 位 /16 位指令模式影响的 SR 位的信息，请参见 *“dsPIC30F Programmer’s Reference Manual”*（DS70030）。

注 1： 字节操作使用 16 位 ALU，并可以产生再加 8 位的结果。但是，如果要保持和 PIC 器件的向后兼容性，所有字节操作的 ALU 结果必须被写回为一个字节（即，不修改 MSB），并且只根据结果 LSB 的状态更新 CPU STATUS 寄存器 SR。

2： 字节模式中执行的所有寄存器指令只会影响 W寄存器的 LSB。可以使用访问 W 寄存器的存储器映射内容的文件寄存器指令修改任何 W 寄存器的 MSB。

2.6 乘法和除法支持

2.6.1 概述

PIC24F 内核包含一个 17 位 x17 位的乘法器，可以用以下乘法模式进行无符号、有符号或混合符号运算：

1. 16 位 x16 位有符号
2. 16 位 x16 位无符号
3. 16 位有符号 x5 位 （立即数）无符号
4. 16 位无符号 x16 位无符号
5. 16 位无符号 x5 位 （立即数）无符号
6. 16 位无符号 x16 位有符号
7. 8 位无符号 x8 位无符号

除法模块能够支持以下数据大小的 32 位 /16 位和 16 位 /16 位有符号和无符号整数除法运算：

1. 32 位有符号 /16 位有符号除法
2. 32 位无符号 /16 位无符号除法
3. 16 位有符号 /16 位有符号除法
4. 16 位无符号 /16 位无符号除法

2.6.2 乘法器

图 2-8 给出了乘法器的框图。它用于支持包含 16 位有符号、无符号和混合符号整数乘法的乘法指令，包括 PIC18F 无符号乘法 MULWF （MUL.w 和 MUL.b）。所有乘法指令都只支持结果的寄存器直接寻址模式。 32 位结果 （来自 MULWF 以外的乘法）被写入任何两个对齐的连续 W 寄存器对，但 W15:W14 因不允许写入而除外。

MULWF指令可选择使用字节或字大小的操作数。目标总是W阵列中的W3:W2寄存器对。字节被乘数会把 16 位结果置于 W2 （W3 不变），字被乘数会把 32 位结果置于 W3:W2。

注：	乘法指令的目标寄存器对必须是 “对齐” 的 （即奇：偶），其中 “奇” 包含结果字的最高有效位，“偶” 包含结果字的最低有效位。例如， W3:W2 是可接受的，而 W4:W3 则不行。

所有乘法指令 （除了 MULWF 这个特例之外）的被乘数都来自 W 阵列 （第 1 个字）和数据空间 （第 2 个字）。 MULWF 的被乘数使用零扩展的 13 位绝对地址，来自 W2 （第 1 个字或字节）和数据空间 （第 2 个字或字节）。

提供了其他数据路径以使这些指令可将结果写回到 W 阵列和数据总线 （通过 W 阵列）中，如图 2-8 中所示。

2.6.2.1 单一和混合模式整数

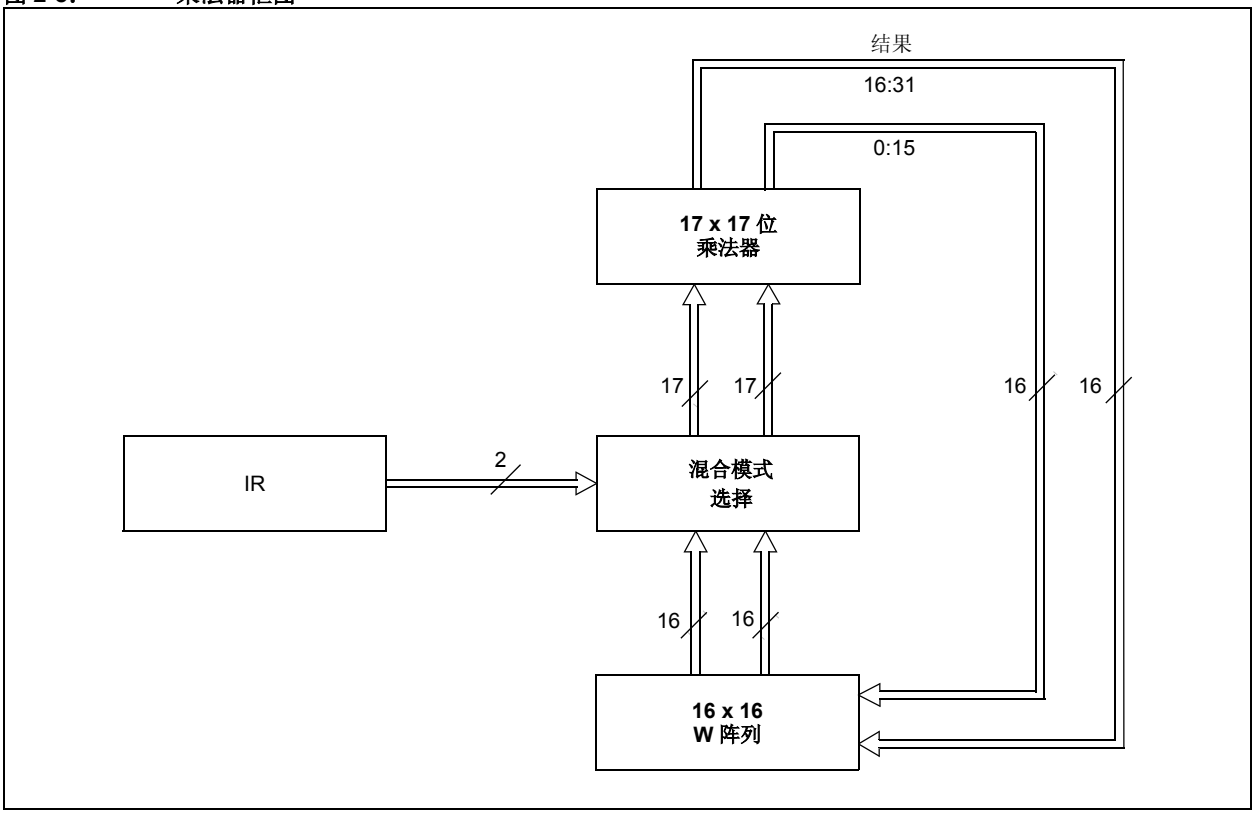
简单数据预处理逻辑将所有操作数零扩展或符号扩展至 17 位，因此无符号、有符号或混合符号乘法都可作为有符号值执行。所有无符号操作数总是零扩展到乘法器输入值的第 17 位。所有有符号操作数总是零扩展到乘法器输入值的第 17 位。

对于 16 位无符号乘法，乘法器产生 32 位无符号结果。

对于 16 位有符号乘法，乘法器产生 30 位数据和 2 位符号。

对于 16 位混合模式（有符号 / 无符号）乘法，乘法器产生 31 位数据和 1 位符号。

图 2-8： 乘法器框图



2.6.3 除法器

PIC24F 同时具有 32 位 /16 位和 16 位 /16 位有符号和无符号整数除法运算功能，这是通过单一指令迭代除法实现的。

所有除法指令的商都放在 W0 中，余数放在 W1 中。16 位有符号和无符号 DIV 指令可为 16 位除数指定任何 W 寄存器（Wn），也可为 32 位被除数指定任何（对齐的）W 寄存器对（W(m + 1):Wm）。除法算法需要用一个周期处理除数的每一位，因此 32 位 /16 位和 16 位 /16 位指令执行所需的周期数是相同的。

除法指令必须在一个 REPEAT 循环内执行完。任何其他执行方式 （例如一系列不连续的除法指令）都无法正确运行，因为指令流功能取决于 RCOUNT。除法流不会自动设置 REPEAT，因此后者必须用正确的操作数值明确执行，如表 2-2 中所示 （REPEAT 将执行目标指令 { 操作数值 + 1} 次）。

表 2-2： 除法执行时间

指令	说明	迭代	REPEAT 操作数值	总执行时间（包括 REPEAT）
DIV.SD	有符号除法： W(m + 1):Wm/Wn → W0; Rem → W1	18	17	19
DIV.SW	有符号除法： Wm/Wn → W0; Rem → W1	18	17	19
DIV.UD	无符号除法： W(m + 1):Wm/Wn → W0; Rem → W1	18	17	19
DIV.UW	无符号除法： Wm/Wn → W0; Rem → W1	18	17	19

每次迭代后，所有中间数据都保存在 W1:W0 中。 N、 C 和 Z 状态标志用于在迭代之间传递控制信息。因此，尽管除法指令列为 19 个周期的操作，但除法迭代过程是可以中断的，这与任何其他 REPEAT 循环相同。

被零除会产生算术错误陷阱。在除法指令的第一个周期内取出除数，因此第一个周期在陷阱的异常处理开始前就执行了。更多详细信息，请参见**第 8 章 “中断”**。

2.7 编译器友好架构

内核架构的设计可生成高效 （代码长度和速度）的 C 编译器。

- 对于大多数指令，内核能在每个指令周期内执行一次数据 （或程序数据）存储器读操作、一次工作寄存器 （数据）读操作、一次数据存储器写操作和一次程序 （指令）存储器读操作。因此可以支持 3 个参数的指令，使 A + B = C 操作能在单个周期内执行。
- 指令寻址模式很灵活，非常符合编译器的需求。
- 有 16 个 16 x 16 位的工作寄存器阵列，每个都可以充当数据、地址或地址偏移寄存器。1 个工作寄存器 （W15）作为软件堆栈工作，用于中断和调用。
- 支持整个数据空间的线性间接访问，加上存储器直接寻址范围扩展到了 8 KB，还有 16 位直接地址装入和存储指令。
- 通过新的表读和表写指令使用任何工作寄存器，支持线性间接访问程序空间 （用户空间和测试空间）内的 32K 字 （64 KB）。
- 部分数据空间可映射到程序空间内，访问常数数据时就像它在使用 PSV 模式的数据空间中一样。

2.8 多位移位支持

PIC24F 内核支持使用移位器模块进行单周期、多位算术和逻辑移位。它还支持通过 ALU 进行单位移位。多位移位器能够在单个周期内执行最多 15 位的算术右移或最多 15 位的左移。

下表 2-3 中提供了使用移位操作的指令的完整汇总。

表 2-3： 使用单位和多位移位操作的指令

指令	说明
ASR	将源寄存器算术右移 1 位或多位。
SL	将源寄存器左移 1 位或多位。
LSR	将源寄存器逻辑右移 1 位或多位。

所有多位移位指令都只支持操作数源和结果目标的寄存器直接寻址模式。

2.9 指令流类型

PIC24F 架构中的大多数指令占用程序存储器的一个字并在单个周期内执行。指令预取机制方便了单周期（1 TCY）执行。但是，某些指令的执行需要 2 或 3 个指令周期。因此，PIC24F 架构中有 6 种不同类型的指令流。下面对它们进行了说明：

1. 1 个指令字，1 个指令周期：

执行这些指令需要一个指令周期，如例 2-6 中所示。

例 2-6： 单字单周期指令流

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1. MOV #350, W0	取指 1	执行 1				
2. INC W0, W2		取指 2	执行 2			
3. INC [W0++], W2			取指 3	执行 3		

2. 1 个指令字，2 个指令周期：

这些指令包括相对转移、相对调用、跳过和返回指令。当指令改变 PC（除了递增以外）时，流水线执行的预取指数据必须被丢弃。这使指令执行需要两个有效周期，如例 2-7 中所示。

例 2-7： 单字双周期指令流

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5	TCY6
1. MOV #55, W0	取指 1	执行 1					
2. BTSC.b PORTA, #3		取指 2	执行 2 执行跳过				
3. ADD.b, PORTB			取指 3	强制 NOP			
4. BRA SUB_1				取指 4	执行 4		
5. ADD.b W0, [W1], [W2]					取指 5	强制 NOP	
| | |							
6. Instruction @ address SUB_1						取指 SUB_1	执行 SUB_1

3. 1 个指令字，2 个指令周期（双操作）：

这一类型的指令只有 LDDW 和 STDW 指令（装入和存储双字）。因为数据访问必须是顺序的，完成这些指令需要两个周期，如例 2-8 中所示。

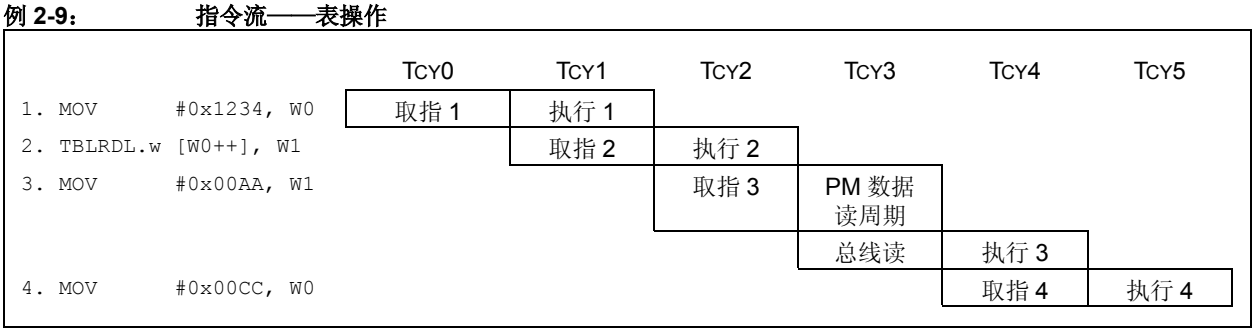
例 2-8： 单字双周期指令流

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1. MOV.w #1234, W2	取指 1	执行 1				
2. MOV.D [W2++], W4		取指 2	执行 2 R/W 周期 1	执行 2 R/W 周期 2		
3. MOV.w 0x0AA, W0			取指 3	不取指	执行 3	
4. MOV.b, PORTA					取指 4	执行 4

4. 1 个指令字， 2 个指令周期表操作：

这些指令将暂停取指令，向程序存储器中插入读或写周期。执行表操作时取得的指令将被保存一个周期，在紧接该表操作的下一个周期执行，如例 2-9 中所示。

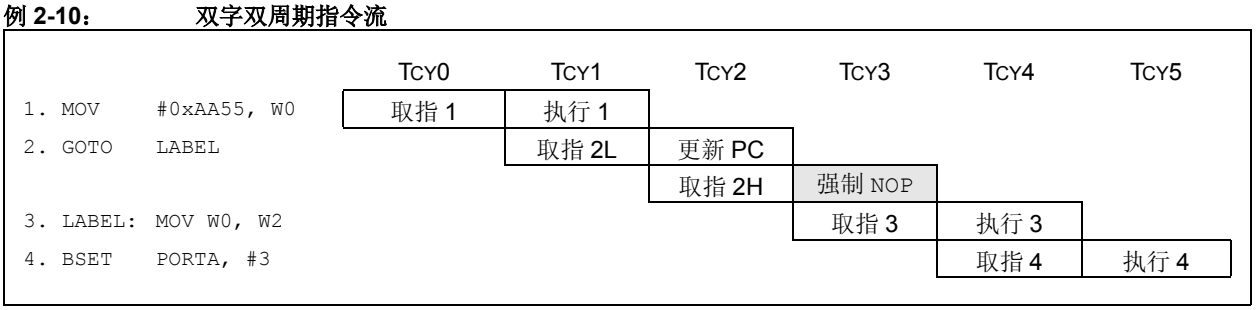
例 2-9： 指令流——表操作



5. 2 个指令字， 2 个指令周期 （GOTO 和 CALL）：

在这些指令中，指令后的取指包含数据。如例 2-10 中所示，这会产生一个双周期指令。如果 CPU 只取了双字指令的第 2 个字而未取第 1 个字，则第 2 个字将被编码以作为 NOP 执行。这在双字指令被跳过指令跳过时尤其重要 （见例 2-13）。

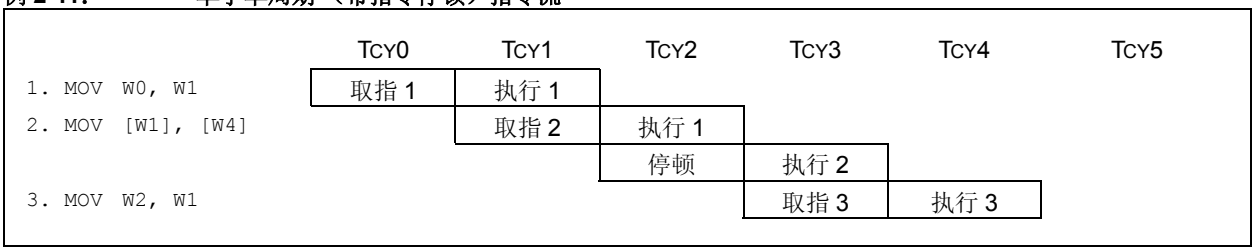
例 2-10： 双字双周期指令流



6. 地址寄存器相依性：

由于数据空间的读写操作之间存在数据地址的相依性，这些指令会遭遇停顿 （stall）。为了解决这种资源冲突，插入了一个额外的周期（见**第 2.11 节“地址寄存器相依性”**中的说明）。

例 2-11： 单字单周期 （带指令停顿）指令流



2.10 程序流循环控制

PIC24F 支持 REPEAT 指令结构，以提供无条件自动程序循环控制。 REPEAT 指令用于实现单指令程序循环。该指令使用 CPU STATUS 寄存器 SR 中的控制位来临时修改 CPU 操作。

2.10.1 REPEAT 循环

REPEAT 指令会使紧随其后的一条指令重复一定次数。可以使用指令中的立即数值或某个 W 寄存器中的值来指定 REPEAT 计数值。使用 W 寄存器的值时可使循环计数为软件变量。

REPEAT 循环中的指令至少执行一次。 REPEAT 循环的迭代次数是 14 位立即数值 + 1 或 Wn + 1。

下面列出了两种 REPEAT 指令的语法形式：

例 2-12： REPEAT 指令语法

```
REPEAT  #lit14          ; RCOUNT <-- lit14
（有效目标指令）
或
REPEAT  Wn              ; RCOUNT <-- Wn
（有效目标指令）
```

2.10.1.1 REPEAT 操作

REPEAT操作的循环计数保存在14位RCOUNT寄存器中，该寄存器是存储器映射的。RCOUNT由REPEAT指令初始化。如果RCOUNT为非零值，REPEAT指令将REPEAT有效状态位RA（SR<4>）置为 1。

RA 是只读位，不能用软件修改。REPEAT 循环计数值大于 0 时， PC 不会递增。PC 递增被禁止直到 RCOUNT = 0。 REPEAT 循环的指令流示例请参见例 2-13。

对于等于 0 的循环计数值， REPEAT 的作用相当于 NOP，并且 RA（SR<4>）位不置 1。REPEAT 循环在开始前实际上是被禁止的，这样可以在预取后续指令时（即在正常的执行流程中）让目标指令只执行一次。

注：	紧随 REPEAT 指令之后的指令 （即目标指令）总是至少执行一次。此指令的执行次数总是会比 14 位立即数或 W 寄存器操作数的指定值多一次。

例 2-13： REPEAT 指令流水线流程

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1.REPEAT #0x2	取指 1	执行 1				
2.MOV [W0++], [W1++]		取指 2	执行 2			
			不取指	执行 2		
				不取指	执行 2	
3.BSET PORTA, #3					取指 3	执行 3

PC （指令结束时）	PC	PC + 2	PC + 2	PC + 2	PC + 4	PC + 6
RCOUNT （指令结束时）	x	2	1	0	0	0
RA （指令结束时）	0	1	1	0	0	0

2.10.1.2 中断 REPEAT 循环

REPEAT 指令循环可在任何时候被中断。

在异常处理期间，RA 状态保留在堆栈中以便让用户在任何数量的中断嵌套中进一步执行 REPEAT 循环。 SRL 存入堆栈后， RA 状态位被清零以便从 ISR 内部恢复正常执行流程。

注： 如果 REPEAT 循环被中断且正在处理 ISR，则用户在 ISR 内部执行另一条 REPEAT 指令前，必须先将 RCOUNT （REPEAT 循环计数器）寄存器存入堆栈。

注： 如果在 ISR 内部使用 REPEAT，用户在执行 RETFIE 前必须先将 RCOUNT 的值出栈。

使用 RETFIE 从 ISR 返回 REPEAT 循环不需要任何特殊处理。中断会在 RETFIE 的第 3 个周期预取要重复的指令。当 SRL 寄存器被弹出堆栈时，堆栈的 RA 位将会恢复。此时如果它置 1，那么中断的 REPEAT 循环将会恢复。

注： 如果重复的指令 （REPEAT 循环中的目标指令）要使用程序空间可视性 （PSV）访问程序空间 （PS）中的数据，从异常处理程序中返回后第一次执行该指令将需要两个指令周期。类似于循环中的第一次迭代，时序限制将不允许第一条指令在单个指令周期内访问驻留在 PS 中的数据。

2.10.1.2.1 REPEAT 循环的提前终止

通过用软件将 RCOUNT 寄存器清零，可以在 ISR 中比正常情况提前终止中断的 REPEAT 循环。

2.10.1.3 REPEAT 指令的限制

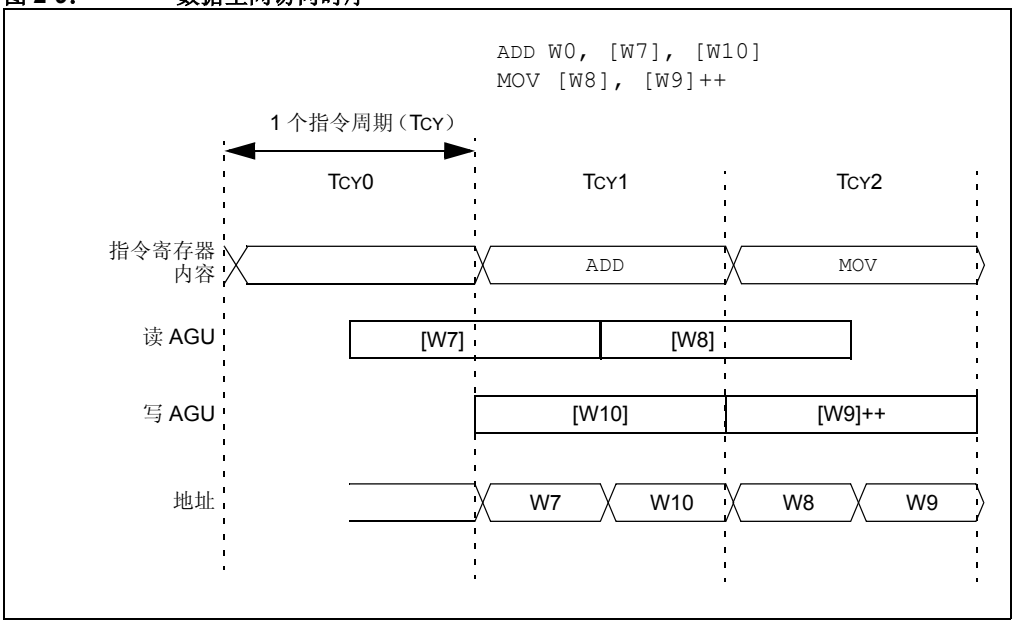
除了以下指令外，其他指令均可紧随 REPEAT 之后：

1. 程序流控制指令 （任何转移、比较和跳过、子程序调用或返回等指令）。
2. 另一条 REPEAT 指令。
3. DISI、 ULNK、 LNK、 PWRSAV 或 RESET 指令。
4. MOV.D 指令。

2.11 地址寄存器相依性

PIC24F 架构支持大多数指令的数据空间读取 （源）和数据空间写入 （目标）操作。地址生成单元（Address Generator Unit，AGU）进行的有效地址（EA）计算和后续数据空间读或写，每个都需要一个指令周期来完成。如图2-9中所示，此时序导致每条指令的数据空间读和写操作部分重叠。由于这一重叠，“先写后读”（Read-After-Write，RAW）数据相依性可能会在指令边界处发生。 RAW 数据相依性在运行时由 PIC24F CPU 检测并处理。

图 2-9： 数据空间访问时序



2.11.1 先写后读相依性规则

如果工作寄存器 Wn 在当前指令中被用作写操作目标，并且预取指令读取的也是同一个工作寄存器 Wn，则适用以下规则：

1. 如果对目标的写入 （当前指令）不修改 Wn 的内容，则不发生停顿；
   或
2. 如果对源的读取 （预取指令）不使用 Wn 计算 EA，也不会发生停顿。

在每个指令周期中，PIC24F 硬件会自动检查以确定是否将发生 RAW 数据相依性。如果不满足上述条件， CPU 会在执行预取指令前自动增加一个指令周期的延时。指令停顿使目标 W 寄存器有足够的时间先执行写入，再让下一条 （预取的）指令使用写入的数据。

表 2-4：　　先写后读相依性汇总

使用 Wn 的 目标寻址模式	使用 Wn 的 源寻址模式	状态	示例 （Wn = W2）
直接寻址	直接寻址	允许	`ADD.w  W0, W1, W2` `MOV.w  W2, W3`
直接寻址	间接寻址	停顿	`ADD.w  W0, W1, W2` `MOV.w  [W2], W3`
直接寻址	带有修改的 间接寻址	停顿	`ADD.w  W0, W1, W2` `MOV.w  [W2++], W3`
间接寻址	直接寻址	允许	`ADD.w  W0, W1, [W2]` `MOV.w  W2, W3`
间接寻址	间接寻址	允许	`ADD.w  W0, W1, [W2]` `MOV.w  [W2], W3`
间接寻址	带有修改的间接寻址	允许	`ADD.w  W0, W1, [W2]` `MOV.w  [W2++], W3`
带有修改的间接寻址	直接寻址	允许	`ADD.w  W0, W1, [W2++]` `MOV.w  W2, W3`

间接寻址	间接寻址	停顿	`ADD.w  W0, W1, [W2]`
			`MOV.w  [W2], W3`
			`; W2=0x0004 (mapped W2)`

间接寻址	带有修改的间接寻址	停顿	`ADD.w  W0, W1, [W2]`
			`MOV.w  [W2++], W3`
			`; W2=0x0004 (mapped W2)`

带有修改的间接寻址	间接寻址	停顿	`ADD.w  W0, W1, [W2++]` `MOV.w  [W2], W3`
带有修改的间接寻址	带有修改的间接寻址	停顿	`ADD.w  W0, W1, [W2++]` `MOV.w  [W2++], W3`

2.11.2　指令停顿周期

指令停顿实际上是附加在指令读阶段前的一个指令周期的等待时间，以便让前面的写操作先完成再发生下一个读操作。为了达到中断延时的目的，应该注意停顿周期与检测到它的指令后的那条指令是相关的 （即停顿周期总是在指令执行周期之前）。

如果检测到了 RAW 数据相依性，PIC24F 将开始指令停顿周期。在指令停顿期间，会发生以下事件：

1. 正在进行的 （上一条指令的）写操作可以正常完成。
2. 在指令停顿周期结束前不会寻址数据空间。
3. 在指令停顿周期结束前禁止 PC 递增。
4. 在指令停顿周期结束前禁止再次取指。

2.11.2.1　指令停顿周期和中断

当会造成指令停顿的两个相邻指令与中断事件同时发生时，可能会产生以下两个结果之一：

1. 中断可能会在第一条指令执行时发生。在这种情况下，允许第一条指令完成，而第二条指令则将在 ISR 完成后执行。这样，由于异常过程为第一条指令提供了完成写阶段的时间，停顿周期将在第二条指令中被消除。
2. 中断可能会在第二条指令执行时发生。在这种情况下，允许第二条指令和附加的停顿周期在 ISR 前执行。这样，与第二条指令关联的停顿周期会正常执行。但是，停顿周期实际上会被嵌入到异常过程时序内。如果一个正常的双周期指令被中断，异常过程将会继续。

2.11.2.2 指令停顿周期和流程更改指令

CALL和RCALL指令使用W15写入堆栈，并且如果下一条指令读取的源使用W15，可能会因此在下一条指令前强制执行指令停顿。

RETFIE和RETURN指令永远不能在下一条指令前强制执行指令停顿，因为这些指令都只能执行读操作。但是，用户应该注意RETLW指令能强制执行停顿，因为它会在最后一个周期写入W寄存器。

由于 GOTO 和转移指令不执行写操作，因此永远不能强制执行指令停顿。

2.11.2.3 指令停顿和 REPEAT 循环

除了增加指令停顿周期外， RAW 数据相依性不会影响 REPEAT 循环的操作。

REPEAT 循环中预取的指令在循环完成或发生异常前不会改变。虽然寄存器相依性检查会跨指令边界进行，在 REPEAT 循环中 PIC24F 实际上会比较同一条指令的源和目标地址。

2.11.2.4 指令停顿和程序空间可视性 （PSV）

程序空间可视性（PSV）使能并且有效地址（EA）处于可视的 PSV 窗口中时，读或写周期会被重新定向到程序空间中的地址。从程序空间访问数据最多需要 3 个指令周期。

PSV 地址空间的指令操作与任何其他指令一样，会受到指令停顿的影响。虽然指令停顿和 PSV 周期都在指令开始时发生，但它们不可能组合起来。如果停顿和 PSV 周期碰巧同时发生，停顿周期将被强制占先，然后是 PSV 周期，最后是指令周期。

看以下代码段：

```
ADD    W0, [W1], [W2++]         ; PSV = 1, W1=0x8000, PSVPAG=0xAA
MOV    [W2], [W3]
```

此指令序列将需要 5 个指令周期来执行。增加的 2 个指令周期用于通过 W1 执行 PSV 访问。此外，为了解决 W2 造成的 RAW 数据相依性，插入了一个指令停顿周期。

指令停顿期间， ROM 锁存值在第一个周期的 Q1 上升沿被传送到 IR，读取的闪存数据在指令第二个周期的 Q3 上升沿被传送到 ROM 锁存器，如图 2-9 中所示。

2.12 寄存器映射

表 2-5 中提供了与 PIC24F CPU 内核相关的寄存器汇总。

表 2-5： 内核 SFR 存储器映射 （用户模式）

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
W0	工作寄存器 0																0000
W1	工作寄存器 1																0000
W2	工作寄存器 2																0000
W3	工作寄存器 3																0000
W4	工作寄存器 4																0000
W5	工作寄存器 5																0000
W6	工作寄存器 6																0000
W7	工作寄存器 7																0000
W8	工作寄存器 8																0000
W9	工作寄存器 9																0000
W10	工作寄存器 10																0000
W11	工作寄存器 11																0000
W12	工作寄存器 12																0000
W13	工作寄存器 13																0000
W14	工作寄存器 14																0000
W15	工作寄存器 15																0800
SPLIM	堆栈指针限制																xxxx
PCL	程序计数器，低字节																0000
PCH	—	—	—	—	—	—	—	—	程序计数器，高字节								0000
TBLPAG	—	—	—	—	—	—	—	—	表页地址指针								0000
PSVPAG	—	—	—	—	—	—	—	—	程序存储器可视性页地址指针								0000
RCOUNT	Repeat 循环计数器																xxxx
SR	—	—	—	—	—	—	—	DC	IPL2	IPL1	IPL0	RA	N	OV	Z	C	0000
CORCON	—	—	—	—	—	—	—	—	—	—	—	—	IPL3	PSV	—	—	0000
DISICNT	—	—	禁止中断计数器														xxxx

图注： x = 复位时的未知值， — = 未实现，读为 0。所示复位值为十六进制。

注 1： 关于特定内核寄存器映射的详细信息，请参见器件数据手册。

2.13 相关应用笔记

本节列出了与手册的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，但是概念是相关的，通过适当修改即可使用，但在使用中可能会受到一定限制。当前与 CPU 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）。

2.14 版本历史

版本 A （2006 年 4 月）

这是本文档的初始发行版。

注：

