

---

---

## 第 4 章 プログラム メモリ

---

---

### ハイライト

本章では次のトピックについて説明します。

4.1	プログラム メモリのアドレス マップ .....	4-2
4.2	プログラム カウンタ .....	4-4
4.3	プログラム メモリからのデータ アクセス .....	4-4
4.4	データ空間からのプログラム空間可視化 .....	4-7
4.5	プログラム メモリへの書き込み .....	4-10
4.6	テーブル命令の動作 .....	4-10
4.7	フラッシュ プログラミング動作 .....	4-16
4.8	レジスタ マップ .....	4-19
4.9	関連するアプリケーション ノート .....	4-20
4.10	改版履歴 .....	4-21

## 4.1 プログラムメモリのアドレスマップ

PIC24F デバイスは、図 4-2 に示すような最大 4M x 24 ビットまでのプログラム メモリ アドレス空間を持つことができます。このプログラム空間をアクセスするためには 3 つの方法があります。

1. 23 ビットのプログラム カウンタ (PC) による方法
2. テーブルリード (TBLRD)) と テーブルライト (TBLWT) 命令による方法
3. プログラムメモリの任意の 32 K バイト セグメントをデータ メモリ アドレス空間にマッピングする方法

プログラム メモリ マップはユーザー プログラム空間とコンフィギュレーション空間に分けられます。ユーザー プログラム空間 (000000h ~ 7FFFFFFh) には、RESET ベクタ、割り込みベクタ テーブル、プログラム メモリが含まれます。デバイス コンフィギュレーションレジスタとデバイス ID 空間は、コンフィギュレーション空間にマッピングされています。コンフィギュレーション ビットとデバイス ID はこれらの場所から読み出すことができ、フラッシュ コンフィギュレーション ワードに望む値をプログラムすることで、コンフィギュレーション ビットをセットまたはクリアすることができます。チップに実装されたプログラム メモリの先頭の 2 ワードは、コンフィギュレーション情報として予約されています。デバイス リセット時に、コンフィギュレーション情報が対応するコンフィギュレーションレジスタにコピーされます。コンフィギュレーション ビットの詳細は、**32.2 項「デバイス コンフィギュレーション」**を参照して下さい。

### 4.1.1 プログラムメモリの構成

プログラム メモリ空間はワードアドレスのブロックとして構成されています。これは 24 ビット幅として扱われますが、各プログラム メモリのアドレスを上位ワードと下位ワードとし、上位ワードの上位バイトは実装されていないととして考えた方が適切です。下位ワードは常に偶数アドレスで、上位ワードは奇数アドレス (図 4-1) です。

プログラム メモリ アドレスは常に下位ワードのワードへ配置されていて、コード実行中にはアドレスは 2 ずつ増減します。

図 4-1: プログラムメモリの構成

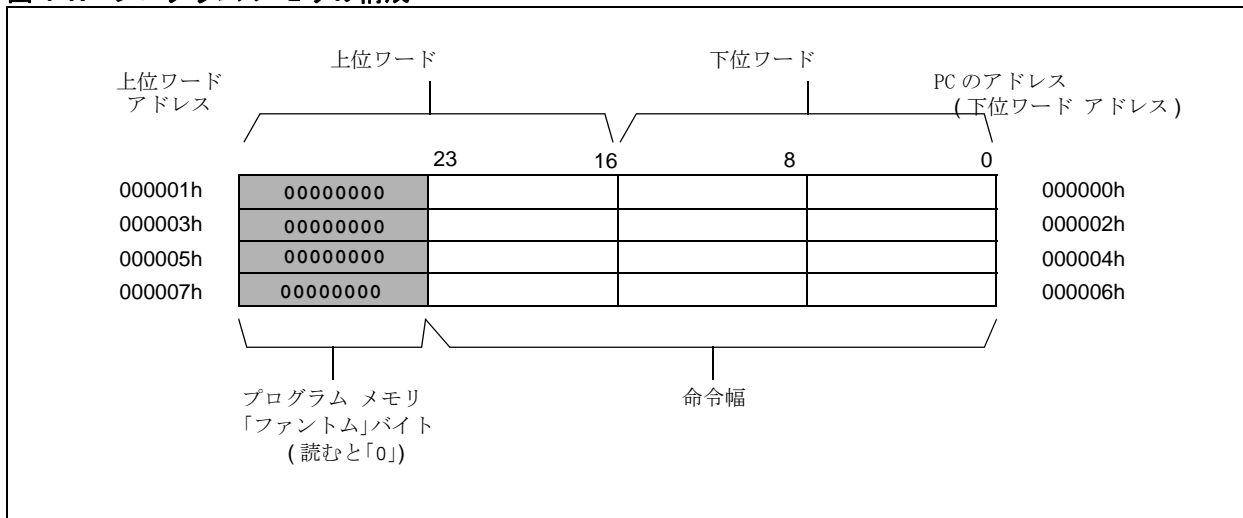
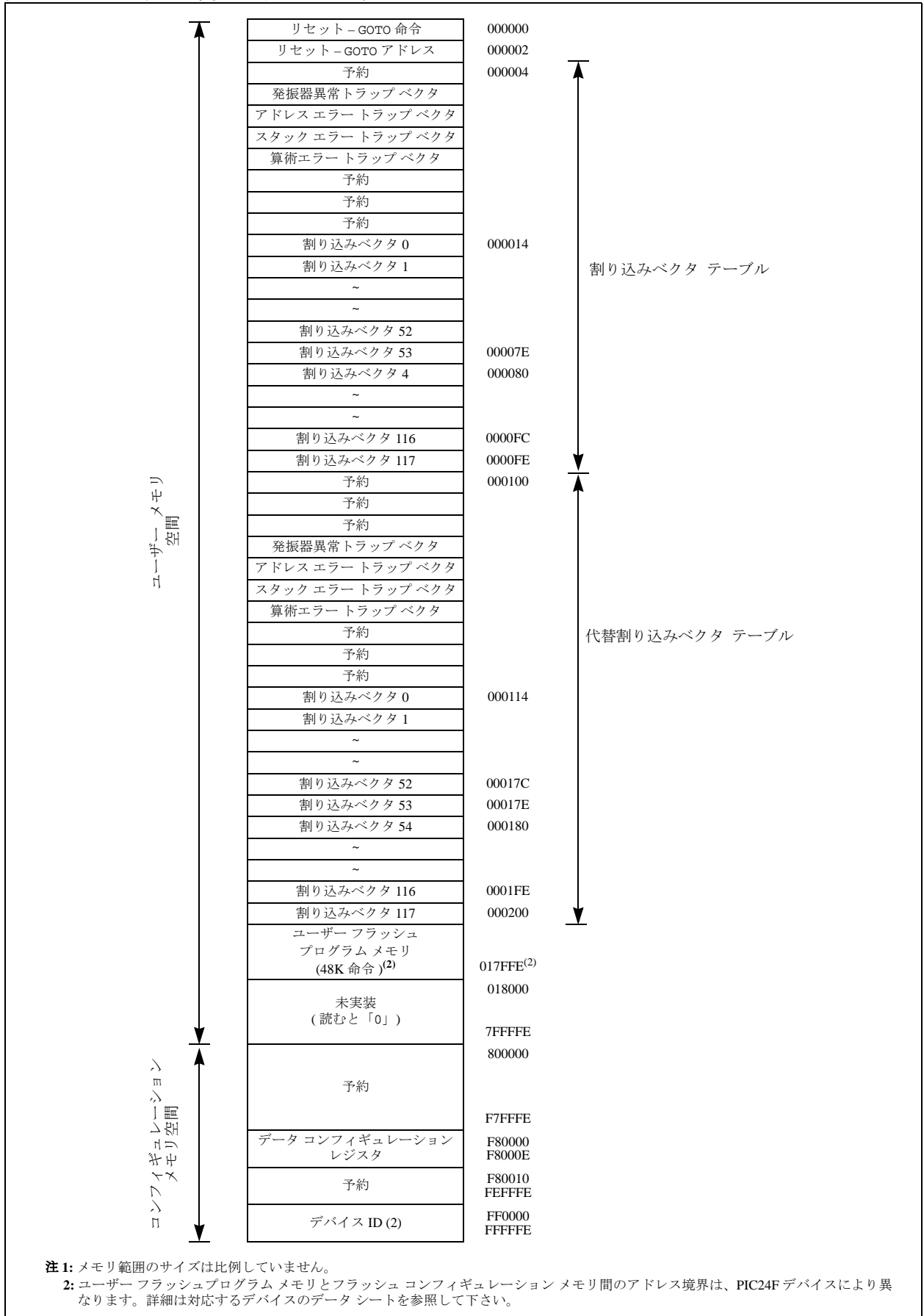


図 4-2: プログラム空間メモリ マップの例 (1)



注 1: メモリ範囲のサイズは比例していません。

注 2: ユーザーフラッシュプログラムメモリとフラッシュコンフィギュレーションメモリ間のアドレス境界は、PIC24F デバイスにより異なります。詳細は対応するデバイスのデータシートを参照して下さい。

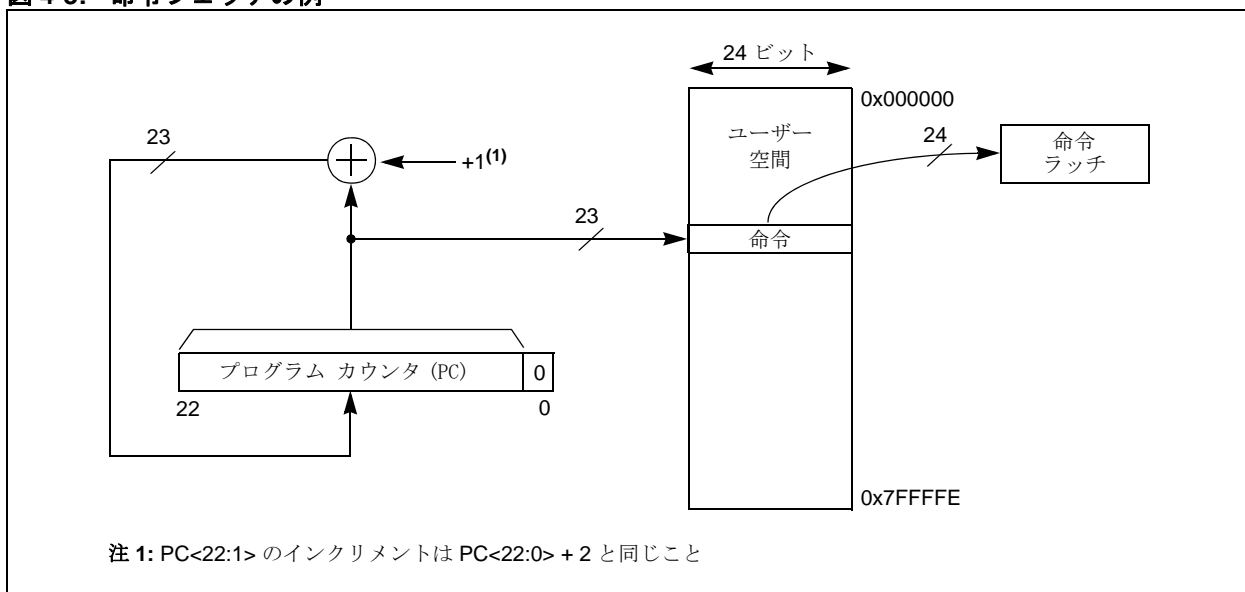
## 4.2 プログラムカウンタ

PC(プログラムカウンタ)は、データ空間アドレスとの整合性をとるために、LSbを「0」にセットし、2つつインクリメントします。連続する命令ワードは、PC<22:1>により4Mプログラムメモリ空間にアドレスされます。それぞれの命令ワードは24ビット幅です。

プログラムメモリアドレスのLSb(PC<0>)は、プログラムメモリをプログラム空間可視化でデータ空間としてアクセスする場合、あるいはテーブル命令でアクセスする場合のために、バイト選択ビットとして予約されています。PCによる命令フェッチのときには、バイト選択ビットは不要です。従って、PC<0>は常に「0」にセットされます。

命令フェッチの例を図4-3に示します。PC<22:1>を1つつインクリメントするのは、PC<22:0>に2を加えることと同等である点に注意してください。

図 4-3: 命令フェッチの例



## 4.3 プログラムメモリからのデータアクセス

プログラムメモリとデータメモリ空間の間のデータ転送に使用される方法として2つの方法があります。それは、特別なテーブル命令によるものと、データ空間の上位半分に32Kバイトのプログラム空間ページを再マッピングするものの2つです。TBLRDHとTBLWTH命令は、データ空間を経由することなく、プログラム空間内のアドレスの下位ワード(lsw)を直接読み書きできるので、アプリケーションによっては非常に有利です。TBLRDHとTBLWTH命令は、プログラムワードの上位8ビットをデータとしてアクセスする唯一の方法です。

### 4.3.1 テーブル命令のまとめ

テーブル命令セットは、プログラム空間とデータ空間の間で、バイトまたはワードサイズのデータを移動するために提供されます。テーブル読み出し命令は、プログラムメモリ空間からデータメモリへの読み出しに使用されます。テーブル書き込み命令により、データメモリをプログラムメモリ空間へ書き込むことができます。

**注:** テーブル命令の詳細なコード例が第5章「フラッシュメモリ動作」にあります。

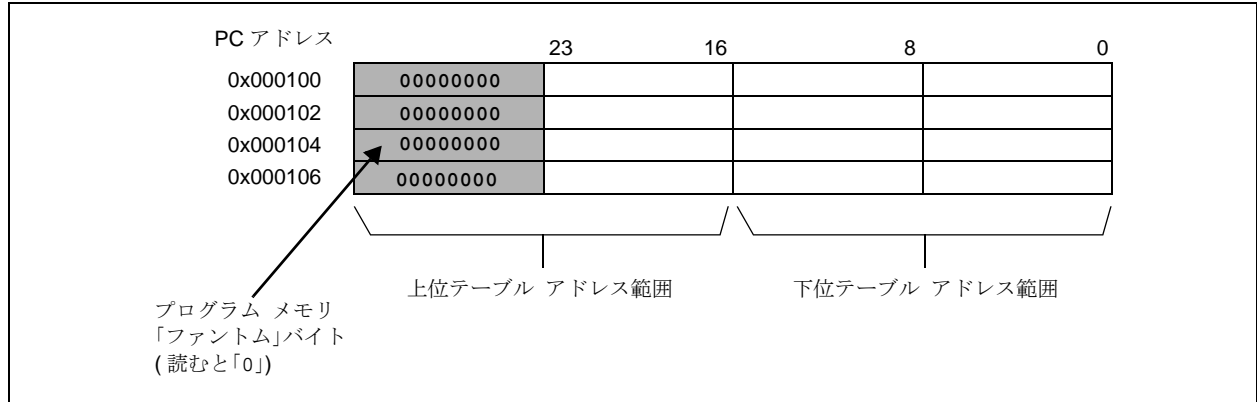
使用できる4種のテーブル命令は次の通りです。

- TBLRDH: テーブル読み出し上位
- TBLRDL: テーブル読み出し下位
- TBLWTH: テーブル書き込み上位
- TBLWTL: テーブル書き込み下位

テーブル命令では、プログラムメモリは、隣り合って存在する2つの16ビットワード幅のアドレス空間として見なされ、図4-4に示すように、それらは同じアドレス範囲となります。

TBLRDL と TBLWTL はプログラムメモリの下位ワード (lsw) をアクセスし、TBLRDH と TBLWTHは上位ワード (msw) をアクセスします。プログラムメモリは24ビット幅ですので、この後者の空間の上位のバイトは、アドレスは可能ですが存在しません。したがって、「ファントム」バイトと呼ばれます。

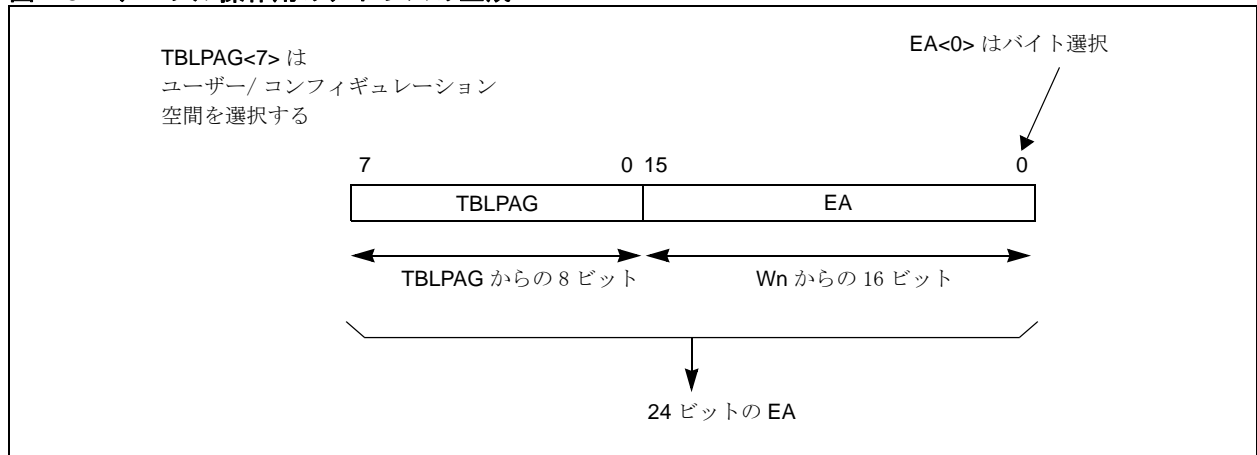
図 4-4: テーブル操作の上位、下位アドレス範囲



### 4.3.2 テーブルアドレスの生成

すべてのテーブル命令では、W レジスタ アドレス値は、8ビットのテーブルページアドレスポインタ TBLPAG と連結され、図4-5に示すように、23ビットの有効プログラム空間アドレスおよびバイト選択ビットを構成します。W レジスタから与えられるプログラム空間アドレスは15ビットありますので、プログラムメモリ内のデータテーブルページは32Kワードです。

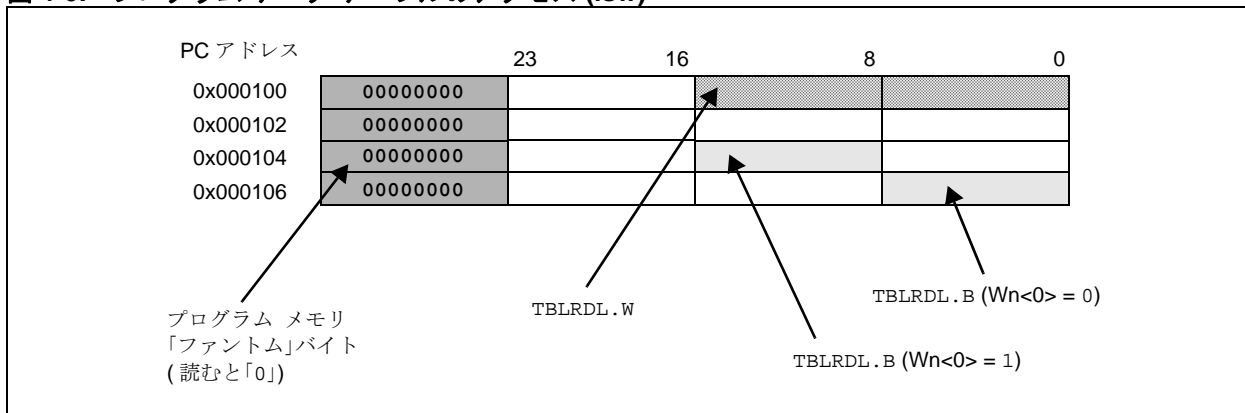
図 4-5: テーブル操作作用のアドレスの生成



## 4.3.3 プログラムメモリの下位ワードアクセス

TBLRDLとTBLWTL命令は、プログラムメモリデータの下位16ビットをアクセスするために使用されます。WレジスタアドレスのLSbは、ワード幅テーブルアクセス用としては無視されます。バイト幅アクセスでは、WレジスタアドレスのLSbは、どちらのバイトを読み出すかを決定します。図4-6に、TBLRDLとTBLWTL命令でアクセスされるプログラムメモリデータ領域を示します。

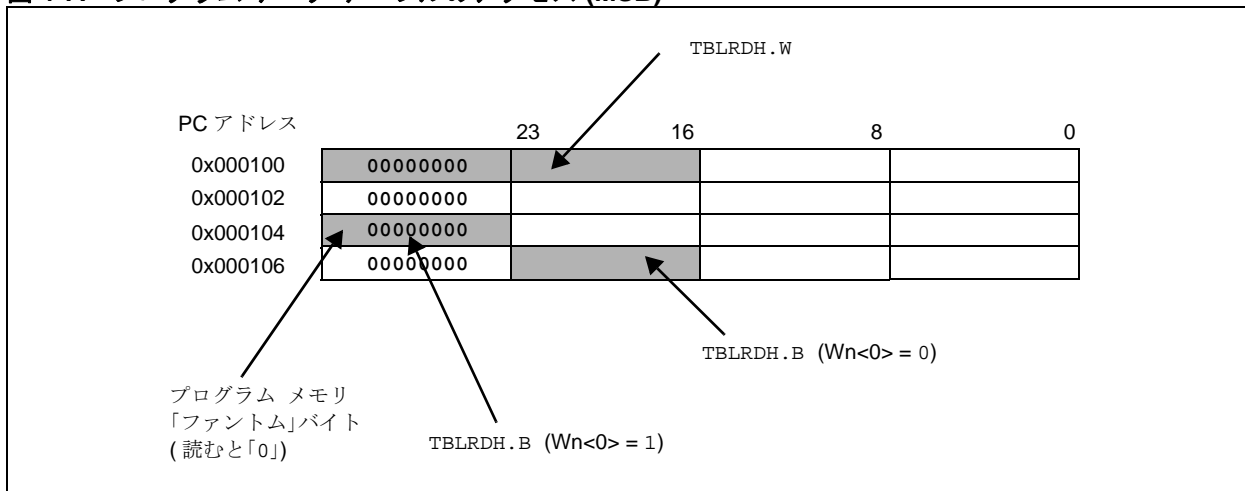
図 4-6: プログラムデータテーブルのアクセス (lsw)



## 4.3.4 プログラムメモリの上位ワードアクセス

TBLRDHとTBLWTH命令は、プログラムメモリデータの上位8ビットをアクセスするために使用されます。これらの命令はまた、ワードまたはバイトアクセスモードの直交性をサポートしますが、図4-7に示すように、プログラムメモリデータの上位バイトは、常に「0」を返します。

図 4-7: プログラムデータテーブルのアクセス (MSB)



## 4.3.5 プログラムメモリへのデータ保存

アプリケーションの多くは、データ保存時にはプログラムメモリが16ビット幅のデータメモリとして見えるようにし、上位バイト(P<23:16>)はデータとしては使われないと仮定しています。プログラムデータの上位バイトはNOP(0x00または0xFF)または不正命令コード(0x3F)として、デバイスが偶発的に格納データを実行してしまうことを防止するようにします。TBLRDHとTBLWTH命令は、データを圧縮して格納する必要があるアプリケーションのため、主にアレイのプログラム/ベリファイのために提供されています。

## 4.4 データ空間からのプログラム空間可視化

PIC24F データ メモリ アドレス空間の上位 32K バイトは、オプションで 16K ワード プログラム空間ページにマッピングすることができます。この動作モードはプログラム空間可視化 (PSV) と呼び、特別な命令 (例えば、TBLRD、TBLWT 命令) を使用することなく、データ空間から、格納された定数データに透明性のあるアクセスをすることができます。

### 4.4.1 PSV 構成

プログラム空間可視化は、PSV ビット (CORCON<2>) をセットすることで有効になります。CORCON レジスタの説明は第 2 章項「CPU」にあります。

PSV が有効になると、データメモリ マップの上位半分 (上位側アドレスのデータメモリ) にあるそれぞれのデータ空間アドレスは、プログラムアドレスに直接マッピングされます (図 4-8 参照) が、PSV ウィンドウで 24 ビット プログラムワードの下位 16 ビットにアクセスできます。プログラムメモリ データの上位 8 ビットは、装置の堅牢性を維持するために、構成的に無効命令または NOP になるようにプログラムする必要があります。テーブル命令だけがそれぞれのプログラムメモリワードの上位 8 ビットを読み出す方法を提供します。

図 4-9: 項「プログラム空間可視化のアドレス生成方法」は PSV アドレスがどのように生成されるかを示しています。PSV アドレスの下位 15 ビットは有効アドレスを含む W レジスタから提供されます。W レジスタの最上位ビットは有効アドレスを形成するためには使用されません。その代わりに、最上位ビットは、プログラム空間からの PSV アクセスを実行するか、データメモリ空間からの通常アクセスを実行するかを特定します。0x8000 以上の W レジスタ有効アドレスが使用されると、PSV が有効な時は、データアクセスはプログラムメモリ空間から行われます。W レジスタ実効アドレスが 0x8000 より小さい場合は、すべてのアクセスがデータメモリから行われます。

残りのアドレスビットは、図 4-9: 項「プログラム空間可視化のアドレス生成方法」に示すように PSVPAG レジスタ (PSVPAG<7:0>) により与えられます。PSVPAG ビットは、W レジスタの下位 15 ビットと結合され、23 ビットのプログラムメモリアドレスを形成するための有効アドレスを保持します。PSV はプログラムメモリ空間内の値にアクセスするときのみを使用されます。ユーザーコンフィギュレーション内の値にアクセスするためには、テーブル命令を使わなければなりません。

W レジスタ値の最下位ビットはバイト選択ビットとして使用され、PSV を使用した命令がバイトまたはワードモード動作できるようにします。

図 4-8: プログラム空間可視化の動作

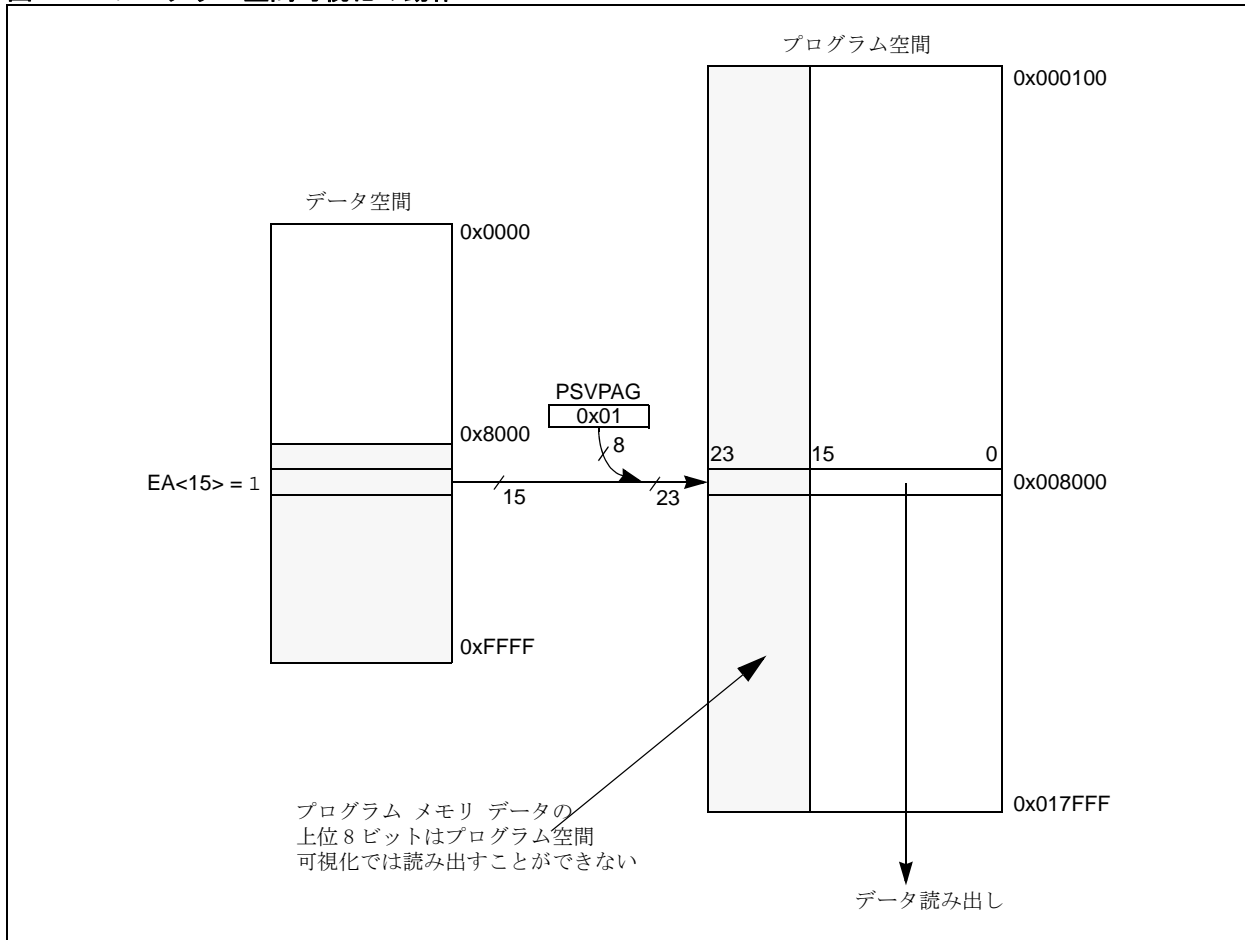
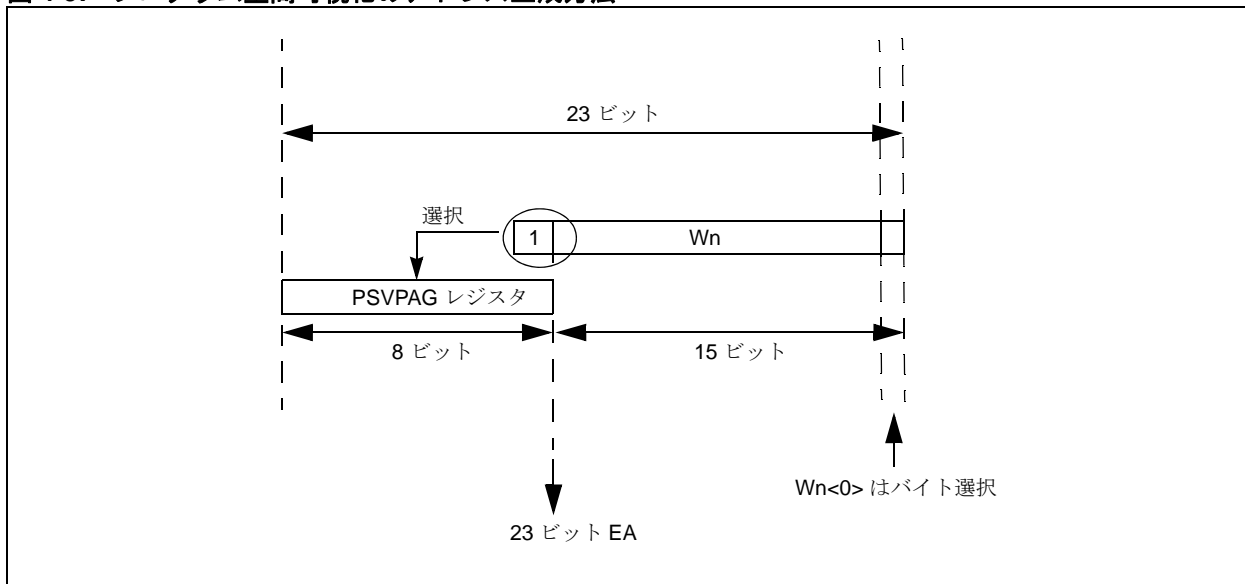


図 4-9: プログラム空間可視化のアドレス生成方法



### 4.4.2 PSV タイミング

PSV を使用する命令は、命令実行に2つの追加命令サイクルを必要とします。ただし、MOV 命令 (MOV.D も含む) は、命令実行に1つだけの追加命令サイクルとなります。

この追加サイクルはプログラムメモリバス上のPSVデータをフェッチするために使用されます。

#### 4.4.2.1 REPEAT ループ内での PSV 使用

REPEAT ループ内で PSV を使用する命令では、プログラムメモリアクセスからのデータアクセスのための追加命令サイクルを無くすことができ、その結果実行時間のオーバーヘッドがありません。ただし、次のような REPEAT ループ内での命令の繰り返しでは、実行完了に2命令サイクルのオーバーヘッドを招きます。

- 最初の繰り返し
- 最後の繰り返し
- 割り込みによりループを抜け出す前の命令実行
- 割り込み処理後にループに戻るときの命令実行

#### 4.4.2.2 PSV と命令ストール

PSV を使用したときの命令ストールについての詳細は第2章「CPU」を参照して下さい。

## 4.5 プログラムメモリへの書き込み

この項では、フラッシュプログラムメモリへのプログラミングテクニックについて説明します。PIC24F デバイスは、ユーザーコードを実行するための内蔵プログラマブルフラッシュメモリを持っています。このメモリに書き込むには、4つの方法があります。

- 実行時自己プログラミング (RTSP)
- インサーキットシリアルプログラミング (ICSP™)
- 改良インサーキットシリアルプログラミング (EICSP)
- JTAG プログラミング

RTSP はユーザーソフトウェアで実行されます。ICSP と EICSP は、デバイスとのシリアルデータ接続で実行され、RTSP よりかなり高速のプログラミングができます。RTSP テクニックは、本項で説明しています。

ICSP と EICSP プロトコルの詳細は、『PIC24FJXXXGA0XX Flash Programming Specification』(DS39768) を参照してください。マイクロチップのウェブサイト ([www.microchip.com](http://www.microchip.com)) からダウンロードできます。JTAG プログラミングについては、IEEE 1149.1-2001, の『IEEE Standard Test Access Port and Boundary Scan Architecture』のプログラミングの章に定義されています。

## 4.6 テーブル命令の動作

テーブル命令は、PIC24F デバイスのプログラム メモリ空間とデータ メモリ空間の間のひとつのデータ転送手段を提供します。フラッシュ プログラム メモリのプログラミング中に使用されるテーブル命令の概略については本項の中に記述されています。4つの基本的なテーブル命令があります。

TBLRDL: テーブル読み出し下位

TBLRDH: テーブル読み出し上位

TBLWTL: テーブル書き込み下位

TBLWTH: テーブル書き込み上位

TBLRDL と TBLWTL 命令は、プログラム メモリ空間のビット <15:0> に対しての読み書きに使用され、TBLRDL と TBLWTL はワードもしくはバイト モードでプログラム メモリにアクセスできます。

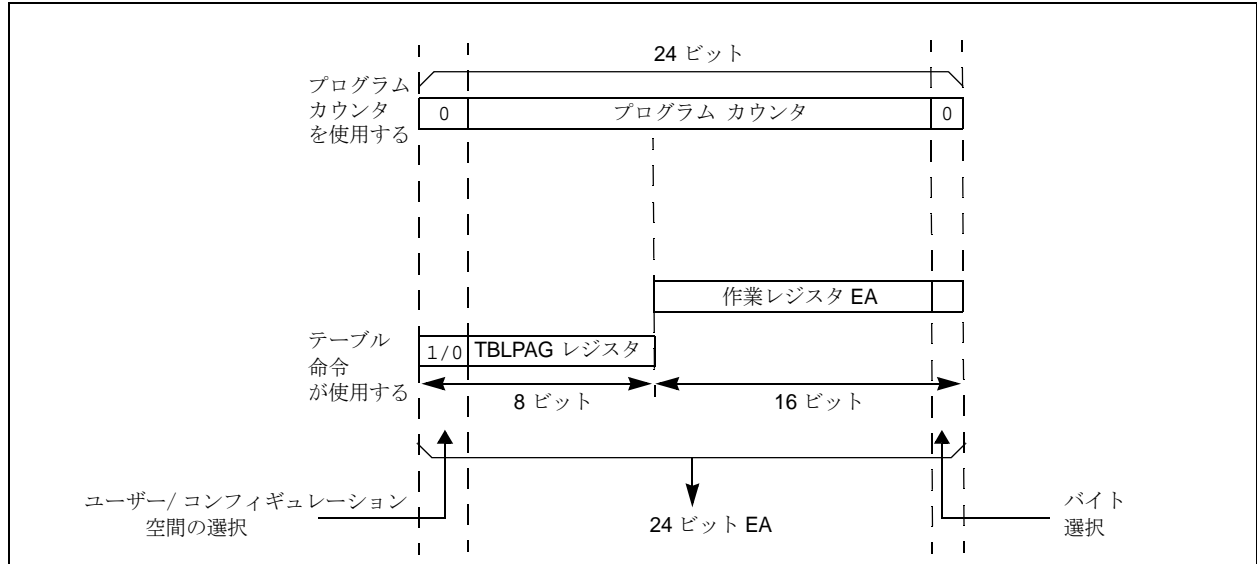
TBLRDH と TBLWTH 命令は、プログラム メモリ空間のビット <23:16> に対しての読み書きをするために使用され、TBLRDH と TBLWTH はワードもしくはバイト モードでプログラム メモリにアクセスできます。プログラム メモリは 24 ビット幅しかありませんので、TBLRDH と TBLWTH 命令は、プログラム メモリの、存在しない上位バイトをアドレッシングすることができてしまいます。このバイトは「ファントムバイト」と呼ばれます。ファントムバイトを読んでも 0x00 が戻るだけであり、ファントムバイトに書き込んでも影響はありません。

24 ビットのプログラム メモリは、並列した 2 つの 16 ビットの空間と見なされ、両空間は同じアドレス範囲を共有します。従って、TBLRDL と TBLWTL 命令は「下位」のプログラム メモリ空間 (PM<15:0>) をアクセスし、TBLRDH と TBLWTH 命令は「上位」のプログラム メモリ空間 (PM<31:16>) をアクセスします。PM<31:24> への読み書きは、ファントム (未実装) バイトへのアクセスになります。バイト モードにおいて任意のテーブル命令が使用される時には、テーブルアドレスの最下位ビットはバイト選択ビットとして使用されます。最下位 ビットは、プログラム メモリ空間の上位もしくは下位のどちらのバイトにアクセスするかを決定します。

図 4-10 にテーブル命令を使用して、プログラム メモリがどのようにアドレッシングされるかを示します。24 ビットのプログラム メモリ アドレスは、TBLPAG<7:0> ビットとテーブル命令で指定される W レジスタの有効アドレス (EA) で構成されます。図 4-10 には、24 ビットのプログラム カウンタも参考として示します。EA の上位 23 ビットはプログラム メモリ位置の選択に使用されます。バイト モードのテーブル命令では、W レジスタの EA の最下位ビットは、16 ビットプログラム メモリ ワードのどちらのバイトにアドレスするかを選ぶために使用されます。「1」はビット <15:8> を、「0」はビット <7:0> を選択します。W レジスタの EA の最下位ビットは、ワードモードのテーブル命令では無視されます。

プログラム メモリ アドレスに加えて、テーブル命令は W レジスタ (またはメモリ位置への W ポインタ) の指定も行います。それは、書き込まれるべきプログラム メモリ データのソースとなったり、もしくはプログラム メモリ読み出し用の読み出し先となります。バイト モードでのテーブル書き込み動作では、作業ソースレジスタのビット <15:8> は無視されます。

図 4-10: テーブルレジスタのアドレッシング



## 4.6.1 テーブル読み出し命令の使い方

テーブル読み出しには2つのステップが必要です。第一に、TBLPAGレジスタとWレジスタの1つを使用してアドレスポインタを設定します。それからそのアドレス位置にあるプログラムメモリの内容を読み出します。

### 4.6.1.1 テーブル命令を使ったプログラムメモリの読み出し

次のコード例は、テーブル命令のワード/バイトモードを使ってどのようにプログラムメモリからワードを読み出すかを示しています。

#### 例 4-1: ワードモードの読み出し

```

; 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
    
```

#### Equivalent C Code

```

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: すべての作業レジスタは使用する前に保存すること。

## 例 4-2: バイトモードの読み出し

```
; 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
```

### Equivalent C Code

```
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: すべての作業レジスタは使用する前に保存すること。

例 4-1 と例 4-2 のコード例では、下位バイトの読み出しの後置インクリメントにより、作業レジスタ内のアドレスが+1 されます。これにより EA<0> は「1」 となり、3 番目の書き込み命令で真中のバイトにアクセスします。最後の後置インクリメントにより、W0 は偶数アドレスにもどり、次のプログラムメモリ位置を指します

**注:** PIC24F 用のマイクロチップアセンブラには、tblpage() と tblockoffset() 擬似命令が提供されています。これらの擬似命令は、プログラムメモリアドレス値からテーブル命令用の TBLPAG と W レジスタの適切な値を選択します。詳しくは『MPLAB® ASM 30, MPLAB® LINK30 and Utilities User's Guide』(DS51317) を参照してください。

## 4.6.2 テーブル書き込み命令の使い方

### 4.6.2.1 テーブル書き込み保持ラッチ

テーブル書き込み命令は、不揮発性プログラムメモリに直接書き込みはしません。代わりに、テーブル書き込み命令は、書き込みデータを保持ラッチにロードします。保持ラッチはメモリ空間上にマッピングされておらず、テーブル命令だけがアクセスできます。すべての保持ラッチにデータがロードされたら、特別な命令シーケンスを実行することにより、実際のメモリプログラミング動作が開始されます。さらに詳細な情報については個別デバイスのデータシートを参照して下さい。

### 4.6.2.2 ワード/バイトモードでの1個のプログラムメモリラッチへの書き込み方

ワードモードで1個のプログラムメモリラッチに書き込むには、次のコードシーケンスが使用できます。

#### 例 4-3: ワードモード書き込み

```

; Setup the address pointer to program space
MOV    #tblpage(PROG_ADDR),W0    ; get table page value
MOV    W0,TBLPAG                ; load TBLPAG register
MOV    #tblockoffset(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++]
    
```

#### Equivalent C Code

```

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

TBLPAG = ((PROG_ADDR & 0x7F0000)>>16);
addrOffset = (PROG_ADDR & 0x00FFFF);
asm("tblwtl %1, [%0]" : "=r"(addrOffset) : "d"(VarWord2)    ;
asm("tblwth %1, [%0]" : "=r"(addrOffset) : "d"(VarWord1)    ;
:
:
}
    
```

**注:** すべての作業レジスタは使用する前に保存すること。

この例では、W3 の上位バイトの内容は、ファントムバイト位置に書き込まれるので、関係ありません。W0 は、2 番目の TBLWTH 命令の後で2 だけ後置インクリメントされ、次のプログラムメモリ位置に書き込む準備をします。

バイトモードで1個のプログラムメモリラッチに書き込むには、次のコードシーケンスが使用できます。

## 例 4-4: バイトモードの書き込み

```
; 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
```

### Equivalent C Code

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

{
:
:

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:** すべての作業レジスタは使用する前に保存すること。

上記コード例では、下位バイトに書き込みを行う際の後置インクリメントにより、W0 のアドレスが + 1 されます。これにより、3 番目の書き込み命令で真中のバイトにアクセスするため EA<0>=1 が設定されます。最後の後置インクリメントにより、W0 は偶数アドレスに戻り、次のプログラムメモリ位置を示します。

### 4.6.3 実行時自己プログラミング (RTSP)

RTSP によりユーザー コードでフラッシュ プログラム メモリの内容を変更できます。RTSP は、TBLRD (テーブル読み出し) と TBLWT (テーブル書き込み) 命令、および NVM コントロール レジスタを使用して遂行することができます。RTSP により、ユーザーはプログラムメモリの 8 行 (64x8 = 512 命令) を一度に消去でき、プログラムメモリデータの 1 行 (64 命令) を一度にプログラムできます。

#### 4.6.3.1 RTSP の動作

PIC24F フラッシュメモリ アレイは、64 命令つまり 192 バイトの行で構成されています。RTSP によりユーザーは一度に 8 行のブロック (512 命令) を消去でき、一度に 64 命令をプログラムできます。8 行の消去ブロックと 1 行の書き込みブロックは、プログラムメモリのはじめから端を揃えて配置され、それぞれ 1536 バイトと 192 バイトが境目となります。

プログラムメモリは、64 命令のプログラミングデータを保持できる保持バッファを内蔵しています。実際のプログラミング動作前に、書き込みデータを順番にバッファにロードします。命令ワードは常に 64 境界のグループでロードします。

RTSP プログラミングの基本のシーケンスは、テーブルポインタを設定してから、TBLWT 命令で連続してバッファにロードします。プログラミングは、NVMCON レジスタの制御ビットをセットすることで実行されます。全部で 64 個の TBLWTL と TBLWTH 命令が命令のロードに必要とされます。

すべてのテーブル書き込み動作は、バッファに書くだけなので 1 ワード書き込み (2 命令サイクル) です。各行のプログラミングには、プログラミングサイクルが必要です。

**注:** 行、ブロック、保持ラッチの数は、デバイスごとに異なります。実際の数については、そのデバイス個別のデータシートを参照して下さい。

### 4.6.4 制御レジスタ

プログラムフラッシュメモリの読み書きには、NVMCON と NVMKEY という 2 個の SFR が使用されます。NVMCON レジスタ (レジスタ 4-1) では、どのブロックを消去するか、どのメモリタイプの書き込みか、そしてプログラミングサイクルの開始を制御します。

NVMKEY は、書き込み専用レジスタで、書き込み保護に使われます。プログラムや消去シーケンスを開始するには、ユーザーは NVMKEY レジスタに 55h と AAh を連続して書き込む必要があります。

#### 4.6.4.1 NVMCON レジスタ

NVMCON レジスタは、フラッシュと EEPROM のプログラム / 消去動作の主制御レジスタです。このレジスタは、消去動作あるいはプログラム動作のどちらを実行するかを選択したり、プログラムや消去サイクルを開始するために使われます。

NVMCON レジスタをレジスタ 4-1 に示します。NVMCON の下位バイトは、実行される NVM 動作のタイプを構成します。

# PIC24F ファミリ リファレンス マニュアル

## レジスタ 4-1: NVMCOM: 不揮発性フラッシュ メモリ制御レジスタ

R/SO-0 <sup>(1)</sup>	R/W-0 <sup>(1)</sup>	R/W-0 <sup>(1)</sup>	U-0	U-0	U-0	U-0	U-0
WR	WREN	WRERR	—	—	—	—	—
ビット 15							ビット 8

U-0	R/W-0 <sup>(1)</sup>	U-0	U-0	R/W-0 <sup>(1)</sup>	R/W-0 <sup>(1)</sup>	R/W-0 <sup>(1)</sup>	R/W-0 <sup>(1)</sup>
—	ERASE	—	—	NVMOP3 <sup>(2)</sup>	NVMOP2 <sup>(2)</sup>	NVMOP1 <sup>(2)</sup>	NVMOP0 <sup>(2)</sup>
ビット 7							ビット 0

<b>凡例:</b>		SO = 設定のみ	
R = 読み出し可	W = 書き込み可	U = 未実装 (読むと「0」)	
-n = POR 後の値	「1」= セット	「0」= クリア	x = 不定

- bit 15      **WR:** 書き込み制御ビット <sup>(1)</sup>  
 1 = フラッシュ メモリのプログラムまたは消去を開始する。動作は自己のタイミングで行われ、動作完了するとハードウェアでクリアされる  
 0 = プログラムまたは消去が完了し、実行中ではない
- bit 14      **WREN:** 書き込み有効化ビット <sup>(1)</sup>  
 1 = フラッシュのプログラム / 消去動作を有効とする  
 0 = フラッシュのプログラム / 消去動作を禁止する
- bit 13      **WRERR:** 書き込みシーケンス エラー フラグ ビット <sup>(1)</sup>  
 1 = 不適切なプログラムまたは消去シーケンスが行われたか中断された (ビットは WR ビットをセットしようとするとき常に自動的にセットされる)  
 0 = プログラムまたは消去動作が正常に完了した
- bit 12-7    **未実装:** 読むと「0」
- bit 6      **ERASE:** 消去 / プログラム有効化ビット <sup>(1)</sup>  
 1 = NVMOP3:NVMOP0 で指定された消去動作を次の WR コマンドで実行する  
 0 = NVMOP3:NVMOP0 で指定されたプログラム動作を次の WR コマンドで実行する
- bit 5-4    **未実装:** 読むと「0」
- bit 3-0    **NVMOP3:NVMOP0:** NVM 動作選択ビット <sup>(2)</sup>  
 1111 = メモリ バルク 消去動作 (ERASE = 1) または動作なし (ERASE = 0)<sup>(3)</sup>  
 0011 = メモリ ワード プログラム動作 (ERASE = 0) または動作なし (ERASE = 1)  
 0010 = メモリ ページ消去動作 (ERASE = 1) または動作なし (ERASE = 0)  
 0001 = メモリ 行プログラム動作 (ERASE = 0) または動作なし (ERASE = 1)

注 1: これらのビットは POR でのみリセットされる。

2: NVMOP3:NVMOP0 の他の組み合わせはすべて未実装。

3: この動作は ICSP™ モードのみで有効。

## 4.6.4.2 NVMKEY レジスタ

NVMKEY は書き込み専用レジスタで、偶発的にフラッシュメモリが書き込まれたり消去されたりすることを防止するために使用されます。プログラムまたは消去シーケンスを開始するには、次のステップを示した順序で正確に実行する必要があります。

1. NVMKEY に 0x55 を書き込む
2. NVMKEY に 0xAA を書き込む
3. 2つの NOP 命令を実行する

このシーケンス後にNVMCONレジスタへの書き込みが1命令サイクルだけ許可されます。ほとんどの場合ユーザーは、プログラムまたは書き込みサイクルを始めるためにはNVMCONレジスタのWRビットを単純にセットして下さい。割り込みはこのアンロックシーケンスの間は禁止して下さい。次のコード例は、アンロックシーケンスがどのように実行されるかを示したものです。

## 例 4-5: アンロック シーケンスの実行

```

; 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

```

## Equivalent C Code

```

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

```

注: すべての作業レジスタは使用する前に保存すること。

そのほかのプログラミング例については 4.7 項「フラッシュプログラミング動作」を参照して下さい。

## 4.7 フラッシュ プログラミング動作

RTSP モードで内蔵フラッシュをプログラミングあるいは消去するためには、完全なプログラミング シーケンスを実行する必要があります。プログラミング動作は標準で 4 ms<sup>(1)</sup> かかり、この動作が完了するまでの間プロセッサはストール (待ち状態) となります。WR ビット (NVMCON<15>) をセットすると動作を開始し、WR ビットは動作完了で自動的にクリアされます。

**注 1:** プログラミング時間はデバイスごとに異なります。正確な値についてはそのデバイスの個別データ シートを参照して下さい。

フラッシュプログラミング動作は、次の不揮発性メモリ (NVM) 制御レジスタで行います。

- NVMCON
- NVMKEY

### 4.7.1 フラッシュ プログラム メモリ プログラミング アルゴリズム

ユーザーは、プログラム フラッシュ メモリを行単位でプログラムできます。このためには、対応する行を含む 8 行単位の消去ブロックを消去する必要があります。一般的な手順は次の通りです。

1. プログラム メモリの 8 行 (512 命令) を読み出してデータ RAM に保存する
2. RAM 内のプログラム データを新データで更新する
3. ブロックを消去する
  - a) ブロック消去のコンフィギュレーションをするため NVMOP ビット (NVMCOM<3:0>) を「0010」にセットする。ERASE (NVMCOM<6>) ビットと WREN (NVMCOM<14>) ビットをセットする
  - b) 消去するブロックの開始アドレスを TBLPAG と W レジスタにセットする
  - c) 55h を NVMKEY に書く
  - d) AAh を NVMKEY に書く
  - e) WR ビット (NVMCOM<15>) をセットする。消去サイクルが開始され、CPU は消去サイクルの間ストールする。消去が完了すると、WR ビットが自動的にクリアされる
4. データ RAM 内の最初の 64 命令をプログラム メモリ バッファに書き込む (4.5 項「プログラム メモリへの書き込み」参照)
5. プログラム ブロックをフラッシュ メモリに書き込む
  - a) 行プログラミングのコンフィギュレーションをするために NVMOP ビットを「0001」にセットする。ERASE ビットをクリアし WREN ビットをセットする。
  - b) 55h を NVMKEY に書く
  - c) AAh を NVMKEY に書く
  - d) WR ビットをセット。プログラミングサイクルが開始され、CPU は書き込みサイクルの間ストールする。フラッシュへの書き込み完了で、WR ビットが自動的にクリアされる
6. TBLPAG の値をインクリメントしてデータ RAM 内ブロックの次の 64 命令を、ステップ 4 と 5 を繰り返して書き込み、512 命令すべてがフラッシュ メモリに書き戻されるまで繰り返す

偶発的な動作を防止するため、いかなる消去あるいはプログラム動作に対しても NVMKEY への書き込み起動シーケンスを使用しなければなりません。プログラミング コマンドを実行後は、プログラミング動作が完了するまで待つ必要があります。プログラミング シーケンスを開始した後の 2 命令サイクルは 4.6.4.2 項「NVMKEY レジスタ」に示したように NOP とする必要があります。

**注 1:** フラッシュ メモリ プログラミングのより詳しいリファレンス コードは、そのデバイスの個別のデータ シートを参照して下さい。

**2:** 行、ブロック、保持ラッチの数は、デバイスごとに異なります。実際の数については、そのデバイスの個別のデータ シートを参照して下さい。

## 4.8 レジスタ マップ

PIC24F のプログラム メモリに関連する特殊機能レジスタのまとめを表 4-1 に示します。

表 4-1: プログラム メモリに関連する特殊機能レジスタ (1)

ファイル名	ビット 15	ビット 14	ビット 13	ビット 12	ビット 11	ビット 10	ビット 9	ビット 8	ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0	リセット後 <sup>(2)</sup>
TBLPAG	—	—	—	—	—	—	—	—	テーブル ページ アドレス ポインタ				—	—	—	0000	
NVMCON	WR	WREN	WRERR	—	—	—	—	—	—	ERASE	—	—	NVMOP3	NVMOP2	NVMOP1	NVMOP0	0000
NVMKEY	—	—	—	—	—	—	—	—	NVMKEY<7:0>								0000

凡例: — = 未実装で読むと「0」。リセット後の値は 16 進数で示す。

注 1: メモリ マップの詳細はそのデバイスの個別のデータ シートを参照して下さい。

注 2: リセット後の値は、POR のみを示す。リセット後の値は、リセット時のメモリ書き込みまたは消去動作状態に依存します。

## 4.9 関連するアプリケーションノート

この項では、マニュアルのこの章に関連するアプリケーションノートをリストアップします。これらのアプリケーションノートは、特に PIC24F デバイス ファミリー用に書かれているわけではありませんが、その概念は適切であり、変更あるいは制限事項も考慮に入れて使用可能です。現状、プログラム メモリに関連するアプリケーションノートは次の通りです。

タイトル	アプリケーションノート#
現在関連するアプリケーション ノートはありません。	

**注:** PIC24F ファミリ デバイスに関するその他のアプリケーションノートやコード例についてはマイクロチップ ウェブサイト ([www.microchip.com](http://www.microchip.com)) をご覧下さい。

### 4.10 改版履歴

#### リビジョン A (2007 年 1 月)

本文書の初版リリース。

