注意 : この日本語版文書は参考資料としてご利用ください。最新情報は必ずオリジナルの英語版をご参照願います。

セクション 2. CPU

ハイライト

本セクションには以下の主要項目を記載しています。

2.1 はじめに

dsPIC33F は、拡張命令セットを備えた 16 ビット (データ) 改良型ハーバード アーキテクチャを採用し、デジタル信号処理を強力にサポートします。この CPU は 24 ビット命令ワードを使用し、可変長オペコード フィールドを備えます。24 ビット幅のプログラム カウンタ (PC) は、最大 4M x 24 ビットのユーザ プログラムメモリ領域をアドレッシングします。

シングルサイクル命令の先読み機能はスループットの維持を支援し、予測可能な動作を提供します。プログラムフローを変更する命令、2 ワード移動動命令 (MOV.D)、テーブル命令、PSV (Program Space Visibility) へアクセスする命令の実行には複数サイクルを要しますが、その他の命令は全てシングルサイクル命令です。任意位置で割り込みが可能な DO および REPEAT 命令を使用して、オーバーヘッドのないプログラムループを構成できます。

2.1.1 レジスタ

dsPIC33F は、プログラマモデル内に 16 個の 16 ビットワーキング レジスタを備えます。各ワーキング レジスタはデータ、アドレス、アドレスオフセット レジスタのいずれかとして使用できます。16 番目のワーキング レジスタ (W15) は、割り込みおよびコール用のソフトウェア スタックポインタ (SSP) として動作します。

2.1.2 命令セット

dsPIC33F の命令セットは下記の 2 クラスに分類されます。

- MCU クラス命令
- DSP クラス命令

これら 2 つの命令クラスはアーキテクチャへシームレスに統合され、単一実行ユニットから実行されます。命令セットは多数のアドレッシング モードを持ち、C 言語コードのコンパイル効率が最適になるように設計されています。

2.1.3 データ領域のアドレッシング

32K ワードまたは 64K バイトとしてアドレッシング可能なデータ領域は、2 つのブロック (X データメモリと Y データメモリ) に分割されます。各メモリブロックは、それぞれ個別のアドレス生成ユニット (AGU) を備えます。 MCU クラス命令は、X メモリ AGU のみを介して動作し、メモリマップ全体へ 1 つのデータ領域としてアクセスします。一部の DSP 命令は X および Y AGU を介して動作し、データアドレス領域を 2 つに分割する 2 オペランド読み出しをサポートします。X および Y データ領域の境界位置はデバイスによって異なります。

必要に応じて、データ領域メモリマップの上位 32K バイトをプログラム領域内へ割り当てる事ができます。この場合の 16K プログラムワード境界の位置は、PSVPAG (Program Space Visibility Page) レジスタにより定義します。プログラム - データ領域マッピング機能により、全ての命令はあたかもそれがデータ領域であるかのようにしてプログラム領域へアクセスできます。さらに、外部バスを用いてデバイス上のプログラムメモリ バスへ RAM を接続する事により、内蔵データ RAM を拡張できます。

オーバーヘッドなしのリングバッファ (モジュロ アドレッシング) は、X および Y アドレス領域の両方で使用できます。モジュロ アドレッシングは、DSP アルゴリズムのソフトウェア境界チェックによるオーバーヘッドを回避します。X AGU 循環アドレッシングは MCU クラス命令の全てに使用できます。X AGU はビット反転アドレッシングもサポートします。これにより、基数 2 の FFT アルゴリズムに対する入出力データの並べ換えを大幅に簡略化できます。

2.1.4 アドレッシング モード

本 CPU は下記のアドレッシング モードをサポートします。

- 命令内 (オペランドなし)
- 相対
- リテラル
- メモリ直接
- レジスタ直接
- レジスタ間接

各命令は、機能的要件に応じて、定義済みのアドレッシング モード グループへ関連付けられます。各命令に対して最大 6 つのアドレッシング モードをサポートします。

大部分の命令において、dsPIC33F は下記の一連の動作を単一命令サイクルで実行可能です。

- データ (またはプログラムデータ) メモリの読み出し
- ワーキング レジスタ (データ) の読み出し
- データメモリの書き込み
- プログラム (命令) メモリの読み出し

従って 3 オペランド命令に対応可能であるため、A ＋ B = C 演算を単一サイクルで実行できます。

2.1.5 DSP エンジンと命令

DSP エンジンは下記を備えます。

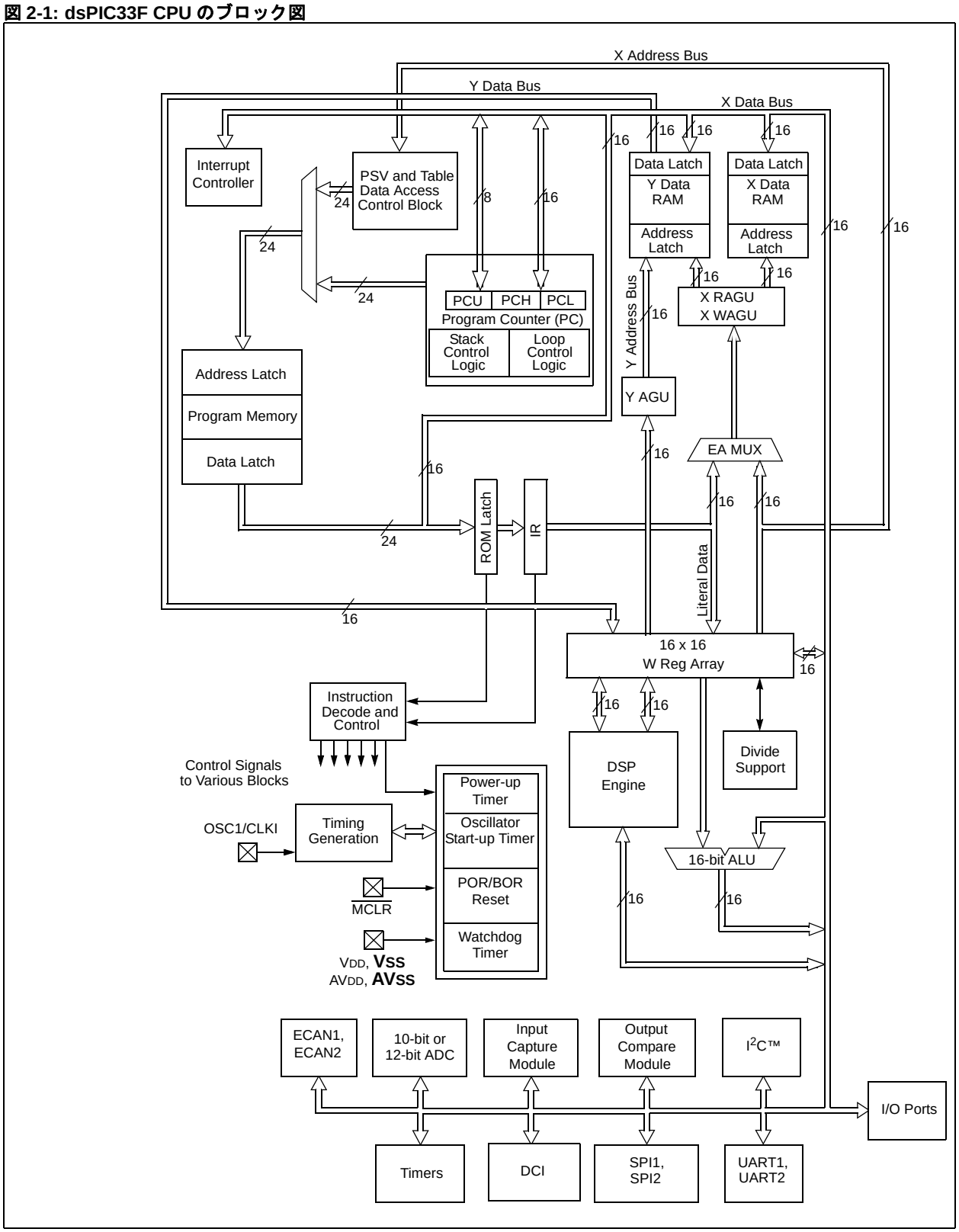
- 1 つの 17 ビット x 17 ビット高速乗算器 (符号付き 16 ビット乗算用)
- 1 つの 40 ビット算術論理演算ユニット (ALU)
- 2 つの 40 ビット飽和処理アキュムレータ
- 1 つの 40 ビット双方向バレルシフタ (単一サイクルで 40 ビット値を 16 ビットまで左右にシフト可能)

DSP 命令は他の命令とシームレスに動作します。また、最適なリアルタイム性能が得られるように設計されています。MAC 命令および他の関連命令は、2 つのワーキング (W) レジスタを乗算する際に、メモリから 2 つのデータオペランドを同時にフェッチ可能です。これらの命令に対してのみデータ領域を分割する必要があります。これは特定のワーキング レジスタを各アドレス領域へ割り当てる事によって、透過的かつ柔軟に行えます。

2.1.6 例外処理

dsPIC33F は、最大 8 つのノンマスカブル トラップと 118 個の割り込みソースを備えたベクタ方式の例外処理機構を採用しています。各割り込みソースには 7 段階の優先度を割り当てる事ができます。図 2-1 に CPU のブロック図を示します。

図 2-1: dsPIC33F CPU のブロック図



2.2 プログラマモデル

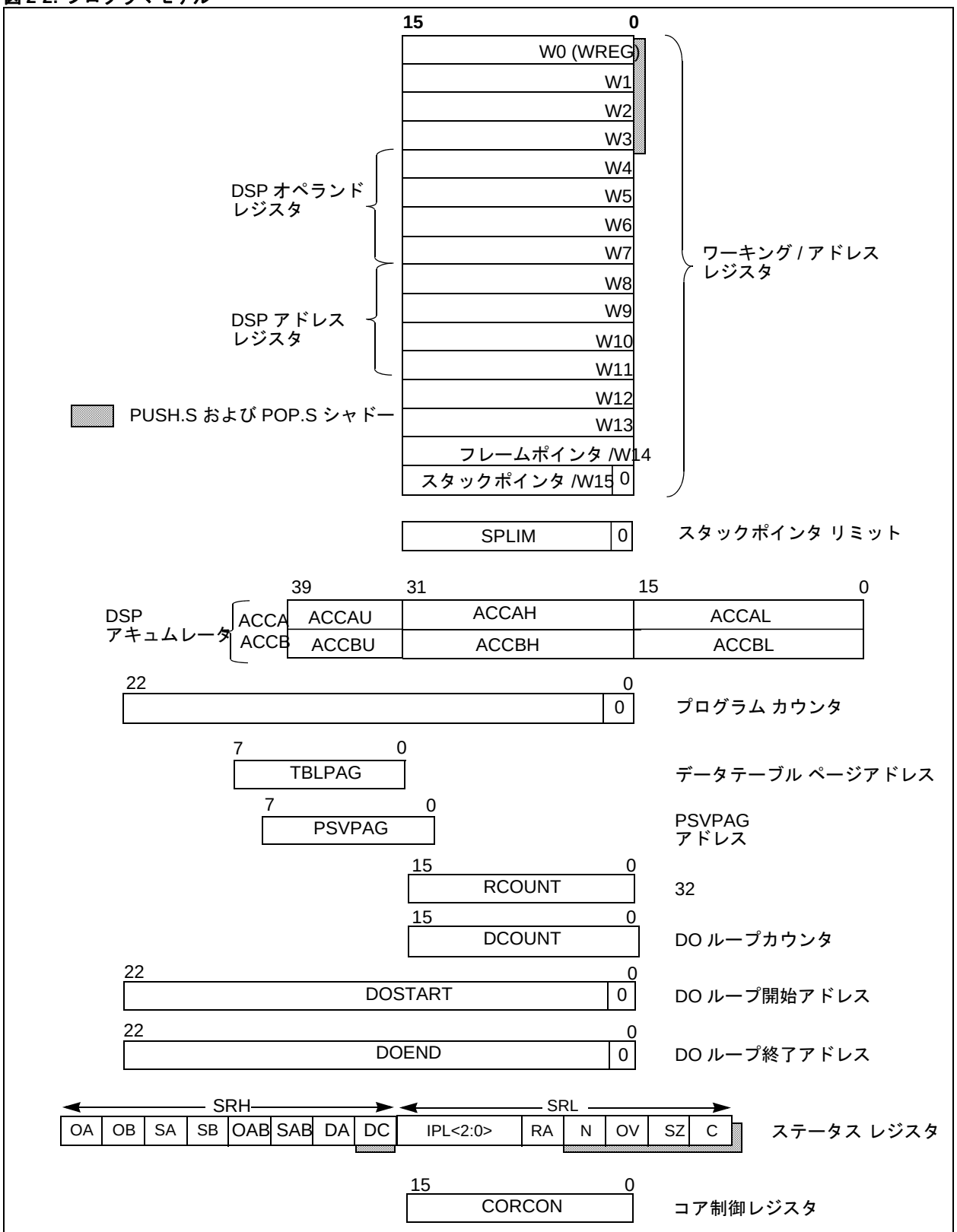
dsPIC33Fのプログラマモデルを図2-2に示します。プログラマモデル内の全てのレジスタはメモリに割り当てられ、命令を用いて直接操作可能です。表 2-1 に各レジスタの概要を示します。

表 2-1: プログラマモデル内のレジスタの概要

レジスタ名	内容
W0 ～ W15	ワーキング レジスタ配列
ACCA、ACCB	40 ビット DSP アキュムレータ
PC	23 ビット PC
SR	ALU および DSP エンジンのステータス レジスタ
SPLIM	スタックポインタのリミット値レジスタ
TBLPAG	テーブルメモリのページアドレス レジスタ
PSVPAG	PSVPAG アドレス レジスタ
RCOUNT	REPEAT ループのカウンタレジスタ
DCOUNT	DO ループのカウンタレジスタ
DOSTART	DO ループの開始アドレスレジスタ
DOEND	DO ループの終了アドレスレジスタ
CORCON	DSP エンジンと DO ループの制御ビットを格納

プログラムモデル内のレジスタに加えて、dsPIC33F はモジュロ アドレッシング用、ビット反転アドレッシング用、割り込み用の制御レジスタも備えます。これらのレジスタについては、本書で後述します。プログラマモデルに関連する全レジスタのメモリ割り当てを表 2-3 に示します。

図 2-2: プログラマモデル



Note: DCOUNT、DOSTART、DOEND は DO ループのネスティング用に 1 レベルのシャドーレジスタを備えています (図には示していません)。

2.2.1 ワーキング レジスタ配列

16 個のワーキング (W) レジスタはデータ、アドレス、アドレスオフセット レジスタのいずれかとして使用できます。ワーキング レジスタの機能は、そのレジスタにアクセスする命令のアドレッシング モードによって決まります。

dsPIC33F の命令セットは下記の 2 タイプに分類できます。

- レジスタ命令
- ファイルレジスタ命令

2.2.1.1 レジスタ命令

レジスタ命令は、各ワーキング レジスタをデータ値またはアドレス オフセット値として使用できます (例 2-1 参照)。

例 2-1: レジスタ命令

```
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.
```

2.2.1.2 ファイルレジスタ命令

ファイルレジスタ命令は、命令オペコードとレジスタ W0 に格納された特定メモリアドレス上で動作します。W0 はファイルレジスタ命令で使用す特殊なワーキング レジスタです。ファイルレジスタ命令では、ワーキング レジスタ W1 ～ W15 をターゲット レジスタとして指定する事はできません。

ファイルレジスタ命令は、1 つのワーキング レジスタしか持たない従来の PIC®MCU に対して下位互換性を保ちます。アセンブラ構文では、ファイルレジスタ命令内で W0 を指定するためにラベル「WREG」を使用します (例 2-2 参照)。

例 2-2: ファイルレジスタ命令

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

Note:	アドレッシング モードと命令構文の詳細は「dsPIC30F/33F プログラマ リファレンス マニュアル」(DS70157) を参照してください。

2.2.1.3 ワーキング レジスタのメモリ割り当て

ワーキング レジスタはメモリへ割り当てられるため、ファイルレジスタ命令ではワーキング レジスタへアクセスする事ができます (例 2-3 参照)。

例 2-3: ファイルレジスタ命令におけるワーキング レジスタへのアクセス

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

where:

0x0004 is the address memory address in W2

さらに、1 つのワーキング レジスタをアドレスポインタとオペランド デスティネーションの両方として使用する命令の実行も可能です (例 2-4 参照)。

例 2-4: ワーキング レジスタをアドレスポインタとオペランド デスティネーションとして使用

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

where:

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

例2-4では、W2の内容は0x0004です。W2はアドレスポインタとして使用され、メモリ内のアドレス 0x0004 を指します。W2 自身もメモリ内のこのアドレスへ割り当てられます。これは起こりそうにない状況ですが、実行前に検出する事は不可能です。dsPIC33F はデータ書き込みを優先するため、この例の結果は W2 = 0x1234 となります。

2.2.1.4 ワーキング レジスタとバイトモード命令

ワーキング レジスタ配列に対するバイト命令は、ターゲット レジスタの最下位バイト (LSB) にのみ影響します。ワーキング レジスタはメモリに割り当てられるため、データメモリ領域へのバイト幅アクセスを介して LSB と MSB(最上位バイト) を操作できます。

2.2.2 シャドーレジスタ

プログラマモデル内のレジスタの多くは、対応するシャドーレジスタを備えます (図 2-2 参照)。これらのシャドーレジスタへ直接アクセスする事はできません。シャドーレジスタには下記の 2 タイプが存在します。

- PUSH.S および POP.S 命令が使用するシャドーレジスタ
- DO 命令が使用するシャドーレジスタ

2.2.2.1 PUSH.S および POP.S 命令が使用するシャドーレジスタ

PUSH.S および POP.S 命令は、関数コールまたは割り込みサービスルーチン (ISR) 発生時のコンテクスト保存 / 復元を高速に実行する上で有効です。PUSH.S 命令は、下記のレジスタ値を対応するシャドーレジスタへ転送します。

- W0 ～ W3
- SR (N、OV、Z、C、DC ビットのみ)

POP.S 命令は、値をシャドーレジスタから元のレジスタへ復元します。例 2-5 に、PUSH.S および POP.S 命令を使用したサンプルコードを示します。

例 2-5: PUSH.S および POP.S 命令

```
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命令はシャドーレジスタの既存の内容を上書きします。シャドーレジスタは1レベルの深さしか持ちません。従ってシャドーレジスタを複数ソフトウェア タスクで使用する場合には注意が必要です。

ユーザ割り当てアプリケーションでは、シャドーレジスタを使用するタスクの実行中に、同一シャドータスクを使用する別の高優先度タスクが割り込まないように配慮する必要があります。高優先度タスクの割り込みによって低優先度タスクが中断すると、低優先度タスク中に書き込んだシャドーレジスタの内容が高優先度タスクによって上書きされます。

2.2.2.2 DO ループのシャドーレジスタ

DO 命令の実行時に、下記レジスタの内容を自動的にシャドーレジスタへ保存します。

- DOSTART
- DOEND
- DCOUNT

DO シャドーレジスタは 1 レベルの深さしか持たず、2 重ループまで自動的なネスティングが可能です。DO ループのネスティングの詳細は **2.9.2.2「DO ループのネスティング」**を参照してください。

2.2.3 初期化されていないワーキング レジスタのリセット

ワーキング レジスタ配列 (W15 を除く) は、全てのリセット発生時にクリアされ、その後書き込みが行われるまで初期化されていないと見なされます。初期化されていないレジスタをアドレスポインタとして用いようと試みると、デバイスがリセットされます。

ワーキング レジスタを初期化するには、1 ワードの書き込みを実行する必要があります。1 バイトの書き込みでは初期化として認識されません。

2.3 ソフトウェア スタックポインタ (SSP)

W15 レジスタは SSP 専用として機能し、例外処理とサブルーチン コール / リターンによって自動的に変更されます。しかし W15 は、他のワーキング レジスタと同様の方法で全ての命令から参照可能です。これはスタックポインタ (SP)の読み書きと操作を簡略化します (例: スタックフレームの作成)。

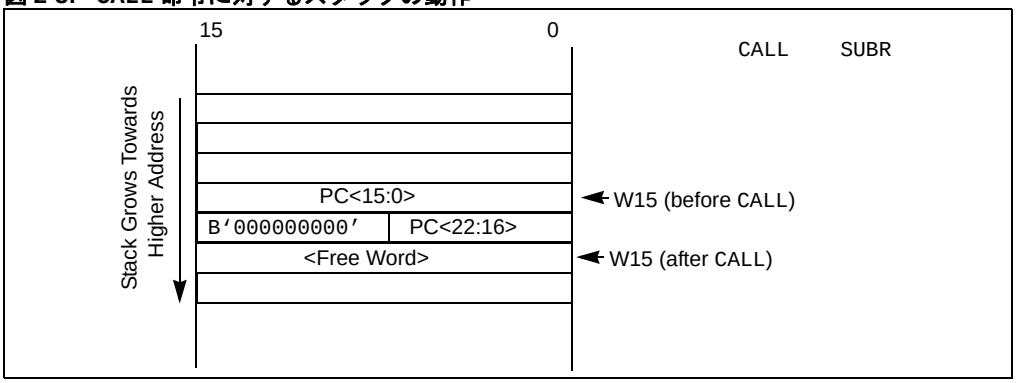
Note:	スタックアクセスの不整合を避けるために、ハードウェアは W15<0> を「0」に設定します。

W15 は全てのリセット発生時に 0x0800 へ初期化されます。このアドレスでは SP が全ての dsPIC33F デバイスの有効な RAM 領域を示す事が保証され、ユーザ ソフトウェアで SP を初期化する前に、例外トラップが発生してしまう事を避けられます。これらは、ユーザ割り当てアプリケーション ソフトウェアが SP を初期化する以前の状態です。ユーザは、初期化中に SP をデータ領域内の任意位置へ移動する事ができます。

SP は常に利用可能な先頭のワードを指し、下位から上位アドレスへ向かってソフトウェア スタックに書き込みます。図 2-3 に、SP によるスタックポップ (読み出し) 前のデクリメントと、スタックプッシュ (書き込み) 後のインクリメントを示します。

PC がスタックへプッシュされると、PC<15:0> ビットが先頭の利用可能スタックワードへプッシュされ、次いで PC<22:16> ビットが 2 番目の利用可能スタック位置へプッシュされます。CALL命令実行時のPCプッシュでは、プッシュ前にPCのMSBをゼロ拡張します(図2-3参照)。例外処理では、PC の MSB を CPU ステータスレジスタ (SR) の下位 8 ビットと連結します。これにより、割り込み処理中に SRL の内容を自動的に保持できます。

図 2-3: CALL 命令に対するスタックの動作



2.3.1 ソフトウェア スタックの例

ソフトウェア スタックの操作には PUSH および POP 命令を使用します。PUSH および POP 命令は、W15 をデスティネーションポインタとして使用する MOV 命令と同じです。W0 の内容をスタックへプッシュできます (例 2-6 参照)。

例 2-6: ソフトウェア スタックの例

```
PUSH W0
This syntax is equivalent to:
          MOV W0,[W15++]
The contents of the top-of-stack can be returned to W0 by:
          POP W0
This syntax is equivalent to:
```

図 2-4 ～図 2-7 にソフトウェア スタックの使用例を示します。図 2-4 はデバイス初期化時のソフトウェア スタックの状態です。W15 は 0x0800 へ初期化済みです。この例では、W0 と W1 にそれぞれ値 0x5A5A と 0x3636 を既に書き込んだ状態を想定します。まず図 2-5 の状態にスタックをプッシュし、W0 内の値をスタックへコピーします。W15 が示すスタック位置は、自動的に次に利用可能なスタック位置 (0x0802) へ更新されます。図 2-6 では、W1 の内容をスタックへプッシュします。図 2-7 ではスタックをポップして、スタック先頭 (TOP) 値 (W1 からプッシュされた値) を W3 へ書き込みます。

図 2-4: デバイスリセット時のスタックポインタ

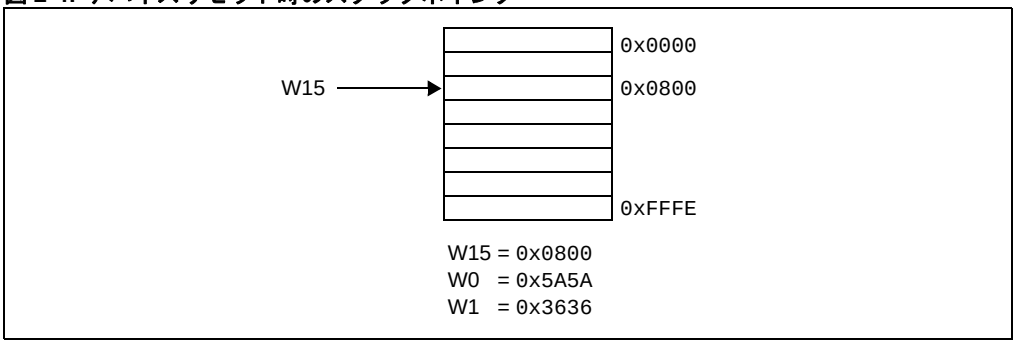


図 2-5: 初回の PUSH 命令後のスタックポインタ

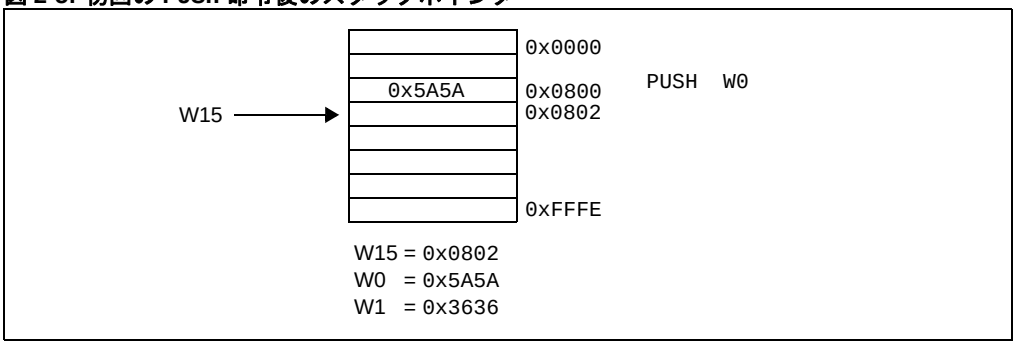


図 2-6: 2 回目の PUSH 命令後のスタックポインタ

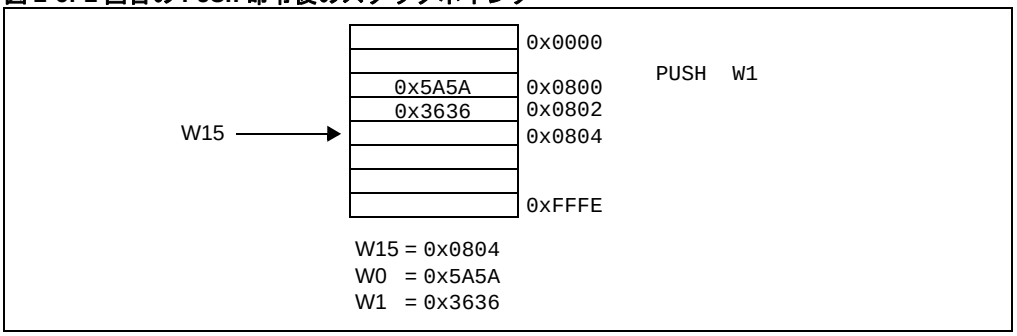


図 2-7: POP 命令後のスタックポインタ

2.3.2 W14 ソフトウェア スタック フレームポインタ

フレームはスタック内のユーザ定義メモリセクションであり、単一のサブルーチンによって使用されます。LNK (リンク) および ULNK (リンク解除) 命令では、ワーキング レジスタ W14 をスタック フレームポインタとして使用できます。W14をフレームポインタとして使用しない場合には、W14 を通常のワーキング レジスタとして使用できます。

W14 をスタック フレームポインタとして使用するソフトウェア例については、「*dsPIC30F/33F プログラマ リファレンス マニュアル*」(DS70157) を参照してください。

2.3.3 スタックポインタのオーバーフロー

スタックポインタ リミット (SPLIM) レジスタは、スタックバッファのサイズを指定します。SPLIM は 16 ビットレジスタですが、全てのスタックはワード単位で処理する必要があるため、SPLIM<0> は「0」に設定されます。

SPLIM レジスタへ 1 ワードを書き込むまで、スタック オーバーフロー チェックは有効になりません。このチェックが一度有効になると、デバイスリセットだけがこれを無効にできます。W15 を用いて生成したソースまたはデスティネーションとして有効な全てのアドレスは、SPLIM 内の値と比較されます。スタックポインタ (W15) の内容が SPLIM レジスタの内容を 2 上回る状態でプッシュ操作を行うと、スタックエラートラップが次のプッシュ操作で発生します。例えば、スタックが RAM 内のアドレス 0x2000 を超えた時にスタックエラー トラップを発生させたい場合は、SPLIM の値を 0x1FFE に初期化します。

Note:	W15 レジスタの内容を使用して有効アドレス (EA) を生成する全ての命令では、スタックエラー トラップが発生する可能性があります。従って、W15 の内容が SPLIM レジスタの内容を 2 上回る場合に CALL 命令を実行するか、あるいは割り込みが発生すると、スタックエラー トラップが発生します。

スタック オーバーフロー チェックが有効な場合、W15 の有効アドレスの計算がデータ領域の終端 (0xFFFF) に達して先頭へ戻る際にも、スタックエラー トラップが発生します。

Note:	SPLIM レジスタへの書き込みの後にW15 を用いた間接読み出しを行ってはなりません。

スタックエラー トラップの詳細は**セクション 6.「割り込み」** (DS70184) を参照してください。

2.3.4 スタックポインタのアンダーフロー

スタックはリセット時に 0x0800 へ初期化されます。スタックポインタのアドレスが 0x0800 より低い場合にスタックエラー トラップが発生します。

Note:	0x0000 ～ 0x07FF のデータ領域は、一般的に、コアと周辺モジュールの特殊機能レジスタ (SFR) 向けに予約済みです。

2.4 CPU レジスタの説明

2.4.1 SR: CPU ステータス レジスタ

dsPIC33F は 16 ビットのステータス レジスタ (SR) を備えます。CPU ステータス レジスタの詳細をレジスタ 2-1 に示します。このレジスタの LSB は、SRL(ステータス レジスタ、下位バイト) と呼びます。同様に MSB は SRH(ステータス レジスタ、上位バイト) と呼びます。

SRL レジスタは、MCU ALU 動作ステータスフラグ、CPU 割り込み優先度ステータスビット (IPL<2:0>)、REPEAT ループのアクティブ ステータスビット RA (SR<4>) の全てを格納します。例外処理中に、SRL を PC の MSB と連結して完全な 1 ワード値を構成し、これをスタックします。

SRH は下記を格納します。

- DSP 加算器 / 減算器のステータスビット
- DO ループのアクティブビット、DA (SR<9>)
- デジットキャリー ビット、DC (SR<8>)

下記を除く SR ビットは読み書き可能です。

- DA ビット (SR<8>): 読み出しのみ可能
- RA ビット (SR<4>): 読み出しのみ可能
- OA、OB (SR<15:14>)、OAB (SR<11>) ビット : 読み出しのみ可能、DSP エンジン ハードウェアのみが変更可能
- SA、SB (SR<13:12>)、SAB (SR<10>) ビット : 読み出しおよびクリア可能、DSP エンジン ハードウェアのみがセット可能 - このビットを一度セットすると、その後の DSP 動作の結果に関係なく、ユーザ割り当てアプリケーションがクリアするまでセット状態を維持します。

Note 1: SAB ビットをクリアすると、SA ビットと SB ビットもクリアされます。

2: 各命令が SR ビットに与える影響については「dsPIC30F/33F プログラマ リファレンス マニュアル」(DS70157) を参照してください。

2.4.2 CORCON: コア制御レジスタ

CORCON レジスタは、DSP 乗算器と DO ループ ハードウェアの動作を制御するためのビットを格納します。CORCON レジスタは IPL3 ステータスビットも格納します。このステータスビットは IPL<2:0> (SR<7:5>) と連結して CPU 割り込み優先度 (IPL) を構成します。

レジスタ 2-1: SR: CPU ステータス レジスタ

R-0	R-0	R/C-0	R/C-0	R-0	R/C-0	R -0	R/W-0
OA	OB	SA(1)	SB(2)	OAB	SAB(3)	DA	DC
bit 15							bit 8

R/W-0(4,5)	R/W-0(4,5)	R/W-0(4,5)	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

凡例 :		C = クリア可能ビット	
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR の値	1 = ビットをセット	0 = ビットをクリア	x = ビットは未知

bit 15 **OA:** アキュムレータ A のオーバーフロー ステータスビット
1 = アキュムレータ A はオーバーフローした
0 = アキュムレータ A はオーバーフローしていない

bit 14 **OB:** アキュムレータ B のオーバーフロー ステータスビット
1 = アキュムレータ B はオーバーフローした
0 = アキュムレータ B はオーバーフローしていない

bit 13 **SA:** アキュムレータ A の飽和ステータスビット (1)
1 = アキュムレータ A は現在飽和している、または、過去に飽和した
0 = アキュムレータ A は飽和していない

bit 12 **SB:** アキュムレータ B の飽和ステータスビット (2)
1 = アキュムレータ B は現在飽和している、または、過去に飽和が発生した
0 = アキュムレータ B は飽和していない

bit 11 **OAB:** OA || OB 複合アキュムレータ オーバーフロー ステータスビット
1 = アキュムレータ A または B でオーバーフローが発生した
0 = アキュムレータ A および B のいずれもオーバーフローしていない

bit 10 **SAB:** SA || SB 複合アキュムレータ飽和「Sticky」ステータスビット (3)
1 = アキュムレータ A または B が現在飽和している、または、過去に飽和した
0 = アキュムレータ A および B のいずれも飽和していない

bit 9 **DA:** DO ループのアクティブビット
1 = DO ループは実行中
0 = DO ループは実行中ではない

bit 8 **DC:** MCU ALU のハーフ キャリー / $\overline{\text{ボロー}}$ ビット
1 = 下位から4桁目のビット(バイトサイズ データの場合)または下位から8桁目のビット(ワードサイズ データの場合) からのキャリーが結果に発生した
0 = 結果に上記のキャリーは発生しなかった

Note 1: SA ビットに対しては、読み出しまたはクリアが可能です (セットは不可)。

2: SB ビットに対しては、読み出しまたはクリアが可能です (セットは不可)。

3: SAB ビットに対しては、読み出しまたはクリアが可能です (セットは不可)。このビットをクリアすると SA および SB ビットもクリアされます。

4: IPL<2:0> ビットは IPL<3> ビット (CORCON<3>) と連結して CPU 割り込み優先度を構成します。括弧内の値は IPL<3> = 1 の場合の値です。IPL<3> = 1 の場合ユーザ割り込みは無効です。

5: IPL<2:0> ステータスビットは、NSTDIS = 1 (INTCON1<15>) の場合にのみ読み出されます。

レジスタ 2-1: SR: CPU ステータス レジスタ (続き)

bit 7-5 **IPL<2:0>**: CPU 割り込み優先度ステータスビット (4,5)
111 = CPU 割り込み優先度は 7(15) 、ユーザ割り込みは無効
110 = CPU 割り込み優先度は 6(14)
101 = CPU 割り込み優先度は 5(13)
100 = CPU 割り込み優先度は 4(12)
011 = CPU 割り込み優先度は 3(11)
010 = CPU 割り込み優先度は 2(10)
001 = CPU 割り込み優先度は 1(9)
000 = CPU 割り込み優先度は 0(8)

bit 4 **RA:** REPEAT ループのアクティブビット
1 = REPEAT ループを実行中
0 = REPEAT ループを実行中ではない

bit 3 **N:** MCU ALU の負数結果ビット
1 = 結果は負数
0 = 結果は負数ではない (ゼロまたは正数)

bit 2 **OV:** MCU ALU のオーバーフロー ビット
このビットは符号付き算術演算 (2 の補数) に使用します。このビットは符号ビットに変化を引き起こすオーバーフローの発生を示します。
1 = 今回の符号付き算術演算でオーバーフローが発生
0 = オーバーフローの発生なし

bit 1 **Z:** MCU ALU のゼロビット
1 = Z ビットに影響する動作が過去に Z ビットをセットした後、Z ビットはクリアされていない
0 = Z ビットに影響する直前の動作が Z ビットをクリアした (すなわち非ゼロ結果)

bit 0 **C:**MCU ALU のキャリー / $\overline{\text{ボロー}}$ビット
1 = 結果の最上位ビット (MSb) からのキャリーが発生した
0 = 結果の最上位ビット (MSb) からのキャリーは発生しなかった

Note 1: SA ビットに対しては、読み出しまたはクリアが可能です (セットは不可)。

2: SB ビットに対しては、読み出しまたはクリアが可能です (セットは不可)。

3: SAB ビットに対しては、読み出しまたはクリアが可能です (セットは不可)。このビットをクリアすると SA および SB ビットもクリアされます。

4: IPL<2:0> ビットは IPL<3> ビット (CORCON<3>) と連結して CPU 割り込み優先度を構成します。括弧内の値は IPL<3> = 1 の場合の値です。IPL<3> = 1 の場合ユーザ割り込みは無効です。

5: IPL<2:0> ステータスビットは、NSTDIS = 1 (INTCON1<15>) の場合にのみ読み出されます。

レジスタ 2-2: CORCON: コア制御レジスタ

U-0	U-0	U-0	R/W-0	R/W-0	R-0	R-0	R-0
—	—	—	US	EDT(1)	DL<2:0>		
bit 15							bit 8

R/W-0	R/W-0	R/W-1	R/W-0	R/C-0	R/W-0	R/W-0	R/W-0
SATA	SATB	SATDW	ACCSAT	IPL3(2)	PSV	RND	IF
bit 7							bit 0

凡例 :	C = クリア可能ビット		
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR での値	1 = ビットをセット	0 = ビットをクリア	x = ビットは未知

bit 15-13 **未実装 :** 「0」として読み出し

bit 12 **US:** DSP 乗算の符号なし / 符号付き制御ビット
1 = DSP エンジン乗算は符号なし
0 = DSP エンジン乗算は符号付き

bit 11 **EDT:** DO ループの早期終了制御ビット (1)
1 = 現在実行中のループ繰り返しが終了した時点で DO ループの実行を停止
0 = 効果なし

bit 10-8 **DL<2:0>:** DO ループ ネスティング レベルのステータスビット
111 = 7 レベルの DO ループがアクティブ
•
•
•
•
001 = 1 レベルの DO ループがアクティブ
000 = アクティブな DO ループなし

bit 7 **SATA:** ACCA 飽和処理のイネーブルビット
1 = アキュムレータ A の飽和処理は有効
0 = アキュムレータ A の飽和処理は無効

bit 6 **SATB:** ACCB 飽和処理のイネーブルビット
1 = アキュムレータ B の飽和処理は有効
0 = アキュムレータ B の飽和処理は無効

bit 5 **SATDW:** DSP エンジンからのデータ領域書き込み飽和処理のイネーブルビット
1 = データ領域書き込み飽和処理は有効
0 = データ領域書き込み飽和処理は無効

bit 4 **ACCSAT:** アキュムレータ飽和処理モードの選択ビット
1 = 9.31 飽和処理 (超飽和)
0 = 1.31 飽和処理 (通常飽和)

bit 3 **IPL3:** CPU 割り込み優先度のステータスビット 3 (2)
1 = CPU 優先度は 7 より大きい
0 = CPU 優先度は 7 以下

bit 2 **PSV:** データ領域内 PSV (Program Space Visibility) のイネーブルビット
1 = データ領域内でプログラム領域は可視
0 = データ領域内でプログラム領域は不可視

Note 1: EDT ビットは常に「0」として読み出されます。

2: IPL3 ビットは IPL<2:0> ビット (SR<7:5>) と連結して CPU 割り込み優先度を構成します。

レジスタ 2-2: CORCON: コア制御レジスタ (続き)

bit 1 **RND:** 丸め処理モードの選択ビット
1 = バイアス (通常) 丸め処理が有効
0 = 符号なし (収束) 丸め処理が有効

bit 0 **IF:** 乗算器の整数 / 小数モードの選択ビット
1 = DSP 乗算の整数モードが有効
0 = DSP 乗算の小数モードが有効

Note 1: EDT ビットは常に「0」として読み出されます。

2: IPL3 ビットは IPL<2:0> ビット (SR<7:5>) と連結して CPU 割り込み優先度を構成します。

2.4.3 dsPIC33F のその他の制御レジスタ

以下では、dsPIC33F が備えるその他のレジスタについて簡単に説明します。これらのレジスタに関する詳細な説明は「*dsPIC33F ファミリ リファレンス マニュアル*」の他のセクションに記載しています。

- **TBLPAG: テーブルページ レジスタ**
  TBLPAG レジスタは、テーブルの読み書き動作中プログラムメモリ アドレスの上位 8 ビットを保持します。テーブル命令は、プログラムメモリ領域とデータメモリ領域間のデータ転送に使用します。詳細は**セクション 4.「プログラムメモリ」**(DS70203) を参照してください。
- **PSVPAG: PSV (Program Space Visibility) ページレジスタ**
  PSVPAG により、ユーザ割り当てアプリケーションはプログラムメモリ領域の 32K バイトセクションをデータアドレス領域の上位 32K バイトへ割り当てる事ができます。この機能を使用すると、データメモリ上で動作する dsPIC33F 命令を用いて (プログラムメモリ上の) 定数データへ透過的にアクセスできます。PSVPAG レジスタは、データアドレス領域へ割り当てられたプログラムメモリ領域の 32K バイトセクションを選択します。PSVPAG レジスタの詳細は**セクション 4.「プログラムメモリ」**(DS70203) を参照してください。
- **MODCON: モジュロ制御レジスタ**
  MODCON レジスタは、モジュロ アドレッシング (リングバッファ) の有効化と設定に使用します。モジュロ アドレッシングの詳細は**セクション 3.「データメモリ」**(DS70202) を参照してください。
- **XMODSRT、XMODEND: X モジュロの開始 / 終了アドレスレジスタ**
  XMODSRT および XMODEND レジスタは、X データメモリ アドレス領域内のモジュロ (リング) バッファの開始および終了アドレスを保持します。モジュロ アレッシングの詳細は**セクション 3.「データメモリ」**(DS70202) を参照してください。
- **YMODSRT、YMODEND: Y モジュロの開始 / 終了アドレスレジスタ**
  YMODSRT および YMODEND レジスタは、Y データメモリ アドレス領域内のモジュロ (リング) バッファの開始および終了アドレスを保持します。モジュロ アレッシングの詳細は**セクション 3.「データメモリ」**(DS70202) を参照してください。
- **XBREV: X モジュロのビット反転レジスタ**
  XBREV レジスタは、ビット反転アドレッシングに使用するバッファサイズを設定します。モジュロ アドレッシングの詳細は**セクション 3.「データメモリ」**(DS70202) を参照してください。
- **DISICNT: 割り込み無効化カウントレジスタ**
  DISICNT レジスタは、指定数のサイクルで優先度 1 ～ 6 の割り込みを無効にするために `DISI` 命令で使用します。モジュロ アドレッシングの詳細は**セクション 6「割り込み」**(DS70184) を参照してください。

2.5 算術論理演算ユニット (ALU)

dsPIC33F の ALU は 16 ビット幅を持ち、加減算、1 ビットシフト、論理演算の能力を備えます。特に明記がない限り、算術演算は 2 の補数に基づきます。演算によっては、ALU は SR レジスタ内の下記ステータスビットの値に影響を与える事ができます。

- Carry (C)
- Zero (Z)
- Negative (N)
- Overflow (OV)
- Digit Carry (DC)

減算では、C および DC ステータスビットはそれぞれボローおよびデジットボロー ビットとして動作します。

ALU は命令のモードに応じて 8 ビットまたは 16 ビット演算を行います。ALU 演算に使用するデータは、命令のアドレッシング モードに応じて、ワーキング レジスタ配列またはデータメモリから読み出す事ができます。同様に、ALU からの出力データは、ワーキング レジスタ配列またはデータメモリ領域へ書き込む事ができます。

各命令、アドレッシング モード、命令の 8/16 ビットモードが SR ビットに与える影響の詳細については、*「dsPIC30F/33F プログラマ リファレンス マニュアル」* (DS70157) を参照してください。

Note 1: バイト演算は 16 ビット ALU を使用し、8 ビットを超える結果を生成可能です。ただし、他の PCI MCU との下位互換性を保つために、バイト演算から得られた ALU 結果は 1 バイトとして書き戻され (すなわち MSB は変更されない)、SR レジスタは結果の LSB ステートのみに基づいて更新されます。

2: バイトモードで実行する全てのレジスタ命令は、ワーキング レジスタの LSB にのみ影響します。 全てのワーキング レジスタの MSB は、メモリへ割り当てられたワーキング レジスタの内容へアクセスするファイルレジスタ命令を用いて変更できます。

2.5.1 バイトからワードへの変換

dsPIC33F は、8 ビットと 16 ビットの ALU 演算を混用する際に便利な下記の 2 つの命令を備えています。

- 符号拡張 (SE) 命令は、ワーキング レジスタまたはデータメモリ内の 1 バイト値を使用して、ワーキング レジスタに保存する符号拡張ワード値を生成します。
- ゼロ拡張 (ZE) 命令は、ワーキング レジスタまたはデータメモリ内の 1 ワード値の 8 MSb をクリアし、その結果をデスティネーションのワーキング レジスタへ書き込みます。

2.6 DSP エンジン

DSP エンジンは複数部品を含むハードウェアであり、ワーキング レジスタ配列からのデータを使用しますが、独自の結果レジスタも備えています。MCU ALU に指示を出す命令デコーダが DSP エンジンも駆動します。加えて、全てのオペランド有効アドレス (EA) がワーキング レジスタ配列内に生成されます。MCU 命令フローとの同時実行はできませんが、命令セット内の全ての命令は MCU ALU と DSP エンジンのリソースを共有できます。

DSP エンジンは下記の部品で構成されます。

- 17 x 17 ビット高速乗算器
- バレルシフタ
- 40 ビット加算器 / 減算器
- 2 つのターゲット アキュムレータ レジスタ
- モード選択が可能な丸め処理ロジック
- モード選択が可能な飽和処理ロジック

DSP エンジンへのデータ入力は、下記のいずれかのソースから行えます。

- 2 ソースオペランド DSP 命令の場合、ワーキング レジスタ配列 (W4、W5、W6、W7 のいずれか) から直接データ入力 (X および Y メモリのデータバス経由で W4、W5、W6、W7 レジスタのデータ値をプリフェッチ)
- その他の DSP 命令の場合、X メモリデータバスからデータを入力

DSP エンジンからのデータ出力は、下記デスティネーションのいずれかへ書き込めます。

- 実行中の DSP 命令が指定するターゲット アキュムレータ
- X メモリ データバスからデータメモリ アドレス領域内の任意位置へ

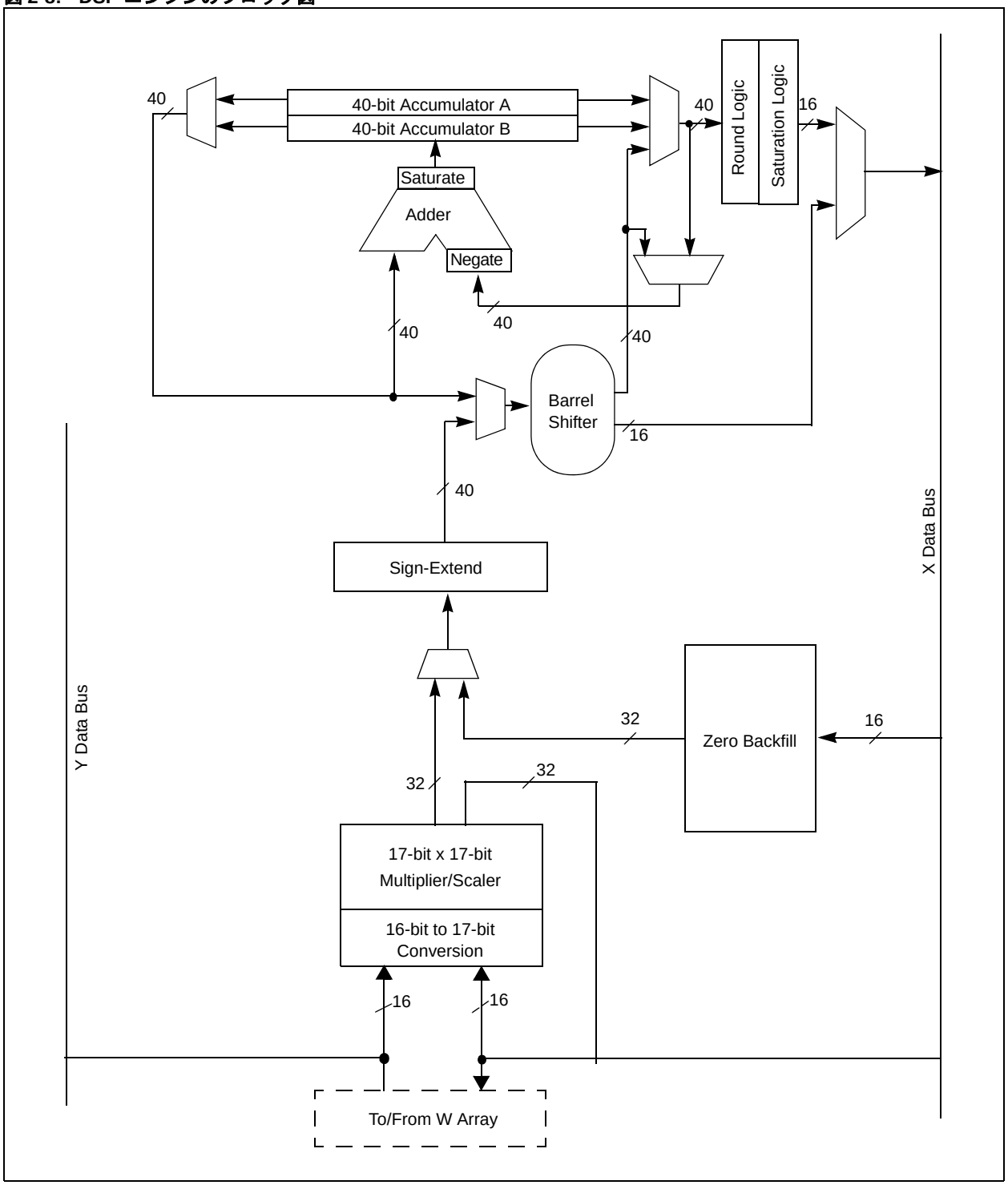
DSP エンジンは、追加データが必要ないアキュムレータ操作には、アキュムレータを指定する事ができます。

MCU シフトおよび乗算命令は、結果を得るために DSP エンジン ハードウェアを使用します。これらの演算におけるデータの読み書きには X メモリ データバスを使用します。

図 2-8 に DSP エンジンのブロック図を示します。

Note:	本セクション関連のサンプルコードと命令構文は「dsPIC30F/33F プログラマ リファレンス マニュアル」(DS70157) を参照してください。

図 2-8: DSP エンジンのブロック図



2.6.1 データ アキュムレータ

2 つの 40 ビット データ アキュムレータ (ACCA と ACCB) は、表 2-3 内の DSP 命令向けの結果レジスタです。各アキュムレータは、下記の 3 つのレジスタへ割り当てられます (「x」はアキュムレータを指定)。

- ACCxL: ACCx<15:0>
- ACCxH: ACCx<31:16>
- ACCxU: ACCx<39:32>

アキュムレータを使用する小数演算では、基数点をビット 31 の右側に置きます。各アキュムレータに保存可能な小数値の範囲は -256.0 ～ (256.0 – 2^{-31}) です。
アキュムレータを使用する整数演算では、基数点をビット 0 の右側に置きます。各アキュムレータに保存可能な整数値の範囲は -549,755,813,888 ～ 549,755,813,887 です。

2.6.2 乗算器

dsPIC33F は 17 x 17 ビット乗算器 (16 ビット符号付き乗算用) を備えます。MCU ALU と DSP エンジンはこの乗算器を共有します。この乗算器は符号付きまたは符号なし演算に対応し、1.31 小数 (Q.31) または 32 ビット整数結果のいずれかをサポートします。

この乗算器は 16 ビットの入力データを 17 ビットへ変換します。乗算器への符号付き入力オペランドは符号拡張され、符号なし入力オペランドはゼロ拡張されます。乗算器内部ではデータを 17 ビットで表現するため、符号付きと符号なしが混在する 16 x 16 ビット乗算を正確に実行できます。

整数モードおよび小数モードにおける乗算器ハードウェア内部のデータ表現は下記の通りです。

- 整数データは基本的に符号付き 2 の補数値として表現し、最上位ビットを符号ビットとして定義します。一般的に N ビットの 2 の補数整数の範囲は -2^{N-1} ～ $2^{N-1} – 1$ です。
- 小数データは 2 の補数小数として表現し、最上位ビット (MSb) を符号ビットとして定義します。基数点は暗黙的に符号ビットの直後に置かれます (Q.X フォーマット)。この暗黙的基数点を持つ N ビットの 2 の補数小数の範囲は -1.0 ～ (1 – 2^{1-N}) です。

整数および小数モードのデータ範囲を表 2-2、図 2-9、図 2-10 を示します。これらの図表から、乗算器ハードウェアがどのように整数および小数モードのデータを解釈するのかが分かります。

乗算器モード選択 (IF) ビット (CORCON<0>) は、表 2-3 内の命令の演算モード (整数 / 小数) を決定します。表 2-4 内の MCU 乗算命令は常に整数モードを使用するため、IF ビットはこれらの命令に影響しません。小数モードでは、乗算器は結果を左へ 1 ビットシフトし、結果の最下位ビット (LSb) を常にクリアします。デバイスリセットは DSP 演算の乗算器モードを既定値の小数モードに設定します。

表 2-2: dsPIC33F のデータ範囲

レジスタ サイズ	整数範囲	小数範囲	小数 分解能
16 ビット	-32768 ～ 32767	-1.0 ～ (1.0 – 2^{-15}) (Q.15 フォーマット)	3.052 x 10^{-5}
32 ビット	-2,147,483,648 ～ 2,147,483,647	-1.0 ～ (1.0 – 2^{-31}) (Q.31 フォーマット)	4.657 x 10^{-10}
40 ビット	-549,755,813,888 ～ 549,755,813,887	-256.0 ～ (256.0 – 2^{-31}) (8 ガードビットの Q.31 フォーマット)	4.657 x 10^{-10}

図 2-9: 0x4001 の整数表現と小数表現

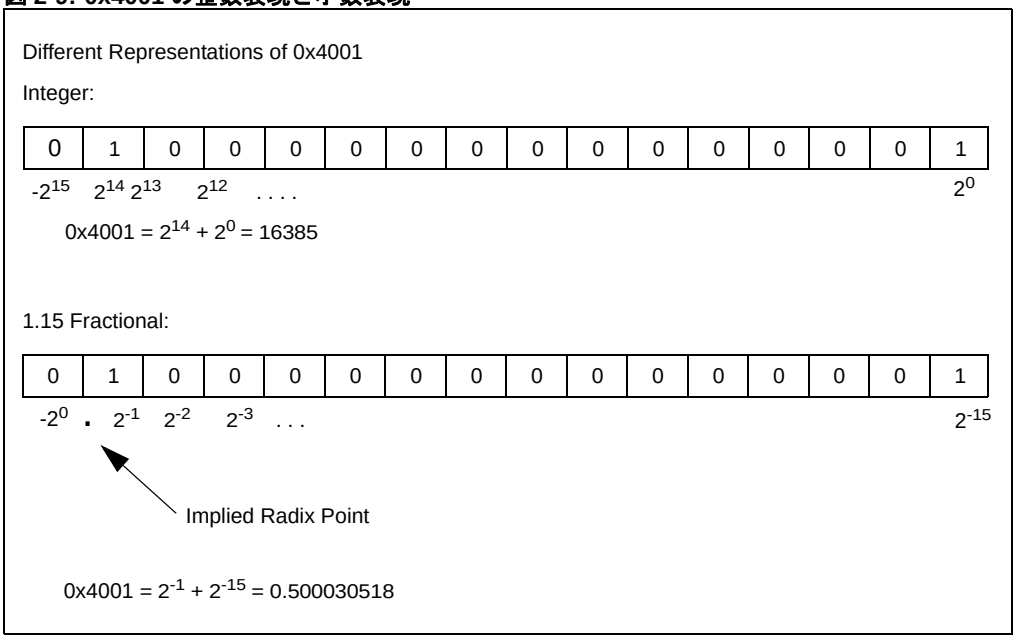
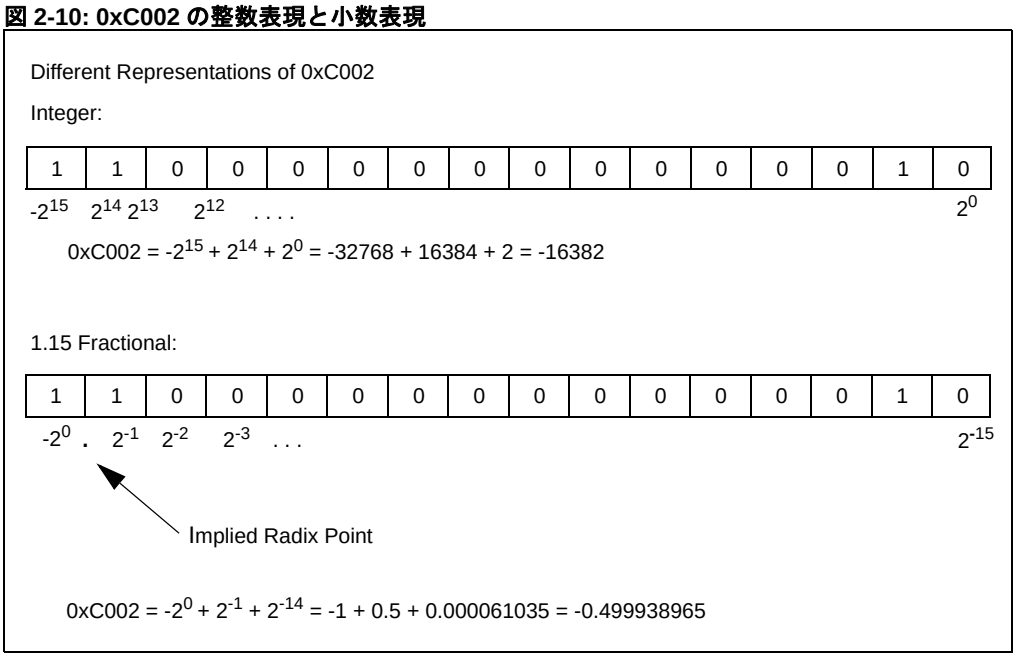


図 2-10: 0xC002 の整数表現と小数表現



2.6.2.1 DSP 乗算命令

乗算器を使用する DSP 命令を表 2-3 に要約します。

表 2-3: 乗算器を使用する DSP 命令

DSP 命令	内容	代数式
MAC	乗算結果をアキュムレータへ加算、または、2 乗結果をアキュムレータへ加算	a = a + b * c a = a + b^2
MSC	乗算結果をアキュムレータから減算	a = a + b * c
MPY	乗算	a = b * c
MPY.N	乗算結果の符号を反転	a = -b * c
ED	差の 2 乗	a = $(b – c)^2$
EDAC	差の 2 乗結果をアキュムレータへ加算	a = a + $(b – c)^2$

Note: 乗算器を使用する DSP 命令は、小数モード (1.15) または整数モードで演算可能です。

DSP 乗算器の符号なし / 符号付き制御 (US) ビット (CORCON<12>) は、DSP 乗算命令の符号付き (既定値)/ 符号なしを決定します。US ビットは MCU 乗算命令には影響しません。MCU 乗算では、符号付きおよび符号なし演算向けにそれぞれ個別の命令を使用します。US ビットをセットすると、表 2-3 内の命令に対する入力オペランドは符号なし値と見なされ、常に乗算値の 17 ビット目へゼロ拡張されます。

2.6.2.2 MCU 乗算命令

この乗算器は、MCU 乗算命令 (16 ビット整数の符号付き同士、符号なし同士、符号付きと符号なし間の乗算) もサポートします (表 2-4 参照)。全ての乗算には整数結果を出力する MUL 命令を使用します。MUL 命令では、オペランドのサイズ (バイトまたはワード) を指定できます。バイト入力オペランドは 16 ビット結果、ワード入力オペランドは 32 ビット結果をワーキング レジスタ配列内の指定レジスタへ出力します。

表 2-4: 乗算器を使用する MCU 命令

MCU 命令	内容
MUL/MUL.UU	2 つの符号なし整数同士を乗算
MUL.SS	2 つの符号付き整数同士を乗算
MUL.SU/MUL.US	符号付き整数と符号なし整数を乗算

Note 1: 乗算器を使用する MCU 命令は、整数モードでのみ演算を行います。

2: MCU 乗算の結果は 32 ビット長を持ち、一対のワーキング レジスタに格納されます。

表 2-5: 乗算オプション

命令クラス	符号付き x 符号付き	符号なし x 符号なし	符号なし x 符号付き	符号付き x 符号なし
MAC (DSP 乗算)	あり (整数または小数)	あり (整数または小数)	なし	なし
MUL (MCU 乗算)	あり (整数のみ)	あり (整数のみ)	あり (整数のみ)	あり (整数のみ)

2.6.3 データ アキュムレータの加算器 / 減算器

データ アキュムレータは、符号付き乗算結果に対する自動的な符号拡張ロジックを持つ40ビット加算器 / 減算器を備えています。アキュムレート前のソースおよびアキュムレート後のデスティネーションとして、2 つのアキュムレータ (A と B) のいずれかを選択できます。ADD (アキュムレータ) および LAC 命令では、必要に応じてアキュムレート実行前にバレルシフタを介してアキュムレートまたはロードするデータをスケーリングできます。

40 ビット加算器 / 減算器はオペランド入力の 1 つの符号を反転する事ができます。これにより、オペランドそのものは変更せずに結果の符号を反転できます。この符号反転は、MSC (a = a - b * c) または MPY.N (a = -b * c) 演算で使用します。

40 ビット乗算器 / 減算器は、アキュムレータのデータ飽和を制御するための飽和処理ブロックを備えています (使用する場合は有効にする必要があります)。

2.6.3.1 アキュムレータのステータスビット

CPU STATUS レジスタ (SR) 内の 6 つの STATUS レジスタビット (表 2-6 参照) は、飽和処理とオーバーフローをサポートします。

表 2-6: アクチュエータ オーバーフロー / 飽和のステータスビット

ステータス ビット	位置	内容
OA	SR<15>	アキュムレータ A がガードビット (ACCA<39:32>) へオーバーフロー
OB	SR<14>	アキュムレータ B がガードビット (ACCB<39:32>) へオーバーフロー
SA	SR<13>	ACCA が飽和 (ビット 31 がオーバーフロー / 飽和) または ACCA がガードビットへオーバーフローして飽和 (ビット 39 がオーバーフロー / 飽和)
SB	SR<12>	ACCB が飽和 (ビット 31 がオーバーフロー / 飽和) または ACCB がガードビットへオーバーフローして飽和 (ビット 39 がオーバーフロー / 飽和)
OAB	SR<11>	OA と OB の論理 OR 結合
SAB	SR<10>	SA と SB の論理 OR 結合 SAB をクリアすると SA と SB もクリアされます。

OA および OB ビットは読み出し専用です。これらのビットは、データがアキュムレータ加算 / 減算ロジックを通過するたびに毎回変更されます。これらのビットがセットされた場合、直前の演算がアキュムレータのガードビット (ビット 32 ～ 39) へオーバーフローしたという事を意味します。この場合ガードビットはアキュムレータ データを保持するため、このタイプのオーバーフローは致命的ではありません。OAB ステータスビットは OA と OB 間の論理 OR 値です。

必要に応じて、OA および OB ビットがセットされた時に算術エラートラップを発生させる事ができます。このトラップを有効にするには、割り込みコントローラ内で割り込み制御レジスタ 1 (INTCON1<10> または <9>) のオーバーフロー トラップフラグ イネーブルビット (OVATE または OVBTE) をセットします。トラップイベントにより、ユーザは即座に必要な修正措置を実施できます。

SA および SB ビットは、データがアキュムレータの飽和処理ロジックを通過するたびに毎回セット可能です。一度セットされると、これらのビットはユーザ割り当てアプリケーションがクリアするまでセット状態を維持します。SAB ステータスビットは SA と SB 間の論理 OR 値を示します。SAB をクリアすると SA および SB ビットもクリアされます。これらのビットがセットされた場合、アキュムレータが最大範囲をオーバーフロー (ビット 31: 32 ビット飽和、ビット 39: 40 ビット飽和) して飽和処理された事を意味します (飽和処理を有効にしておく必要があります)。

飽和処理を有効にしていない場合、SA および SB ビットは致命的なオーバーフロー (アキュムレータの符号破損) の発生を示します。致命的オーバーフロー トラップ イネーブル (COVTE) ビット (INTCON1<8>) をセットした場合には、SA および SB ビットは算術エラートラップを発生します (飽和処理が無効な場合)。

Note 1: 算術エラートラップの詳細は**セクション 6.「割り込み」**(DS70184) を参照してください。

2: SA、SB、SAB ステータスビットの意味は、アキュムレータ飽和処理が有効かどうかによって異なります。アキュムレータ飽和処理のモードは CORCON レジスタで制御します。

2.6.3.2 飽和 / オーバーフロー処理モード

dsPIC33F は 3 つの飽和 / オーバーフロー処理モード (アキュムレータの 39 ビット飽和処理、アキュムレータの 31 ビット飽和処理、アキュムレータの致命的オーバーフロー) をサポートします。

- **アキュムレータの 39 ビット飽和処理**
  このモードでは、飽和処理ロジックは 9.31 フォーマットの正の最大値 (0x7FFFFFFFFF) または負の最大値 (0x8000000000) をターゲット アキュムレータへ転送します。一度セットされた SA または SB ビットは、その後ユーザ割り当てアプリケーションがクリアするまでセット状態を維持します。この飽和処理モードは、アキュムレータのダイナミックレンジの拡張に有効です。

  この飽和処理モードを有効にするには、アキュムレータ飽和処理モード選択(ACCSAT)ビット (CORCON<4>) をセットします。加えて、ACCA 飽和処理イネーブル (SATA) ビット (CORCON<7>) および / または ACCB 飽和処理イネーブル (SATB) ビット (CORCON< 6>) をセットして、アキュムレータ飽和処理を有効にする必要があります。

- **アキュムレータの 31 ビット飽和処理**
  このモードでは、飽和処理ロジックは 1.31 フォーマットの正の最大値 (0x007FFFFFFF) または負の最大値 (0xFF80000000) をアキュムレータへ読み込みます。一度セットされた SA または SB ビットは、ユーザがクリアするまでセット状態を維持します。この飽和処理モードを有効にした場合、アキュムレータ値の符号拡張以外にはガードビット 32 ～ 39 を使用しません。このため SR 内の OA、OB、OAB ビットがセットされる事はありません。

  このオーバーフロー / 飽和処理モードを有効にするには、ACCSAT (CORCON<4>) ビットをクリアします。加えて、SATA (CORCON<7>) および / または SATB (CORCON<6>) ビットをセットして、アキュムレータ飽和処理を有効にする必要があります。

- **アキュムレータの致命的オーバーフロー**
  SATA (CORCON<7>) および / または SATB (CORCON<6>) ビットをセットしないとアキュムレータ飽和処理は行われず、従ってアキュムレータはビット 39 まで完全にオーバーフロー (符号を破壊) する可能性があります。この場合、致命的オーバーフロー トラップ イネーブル (COVTE) ビット (割り込みコントローラ内の INTCON1<8>) をセットする事によって、致命的オーバーフロー発生時に算術エラートラップを発動させる事ができます。

アキュムレータの飽和 / オーバーフロー検出は、2 つのアキュムレータのいずれかを 40 ビット DSP ALU を介して変更する DSP 命令の実行によってのみ発生します。MCU クラス命令を使用してアキュムレータへメモリ割り当てレジスタとしてアクセスする場合には、飽和 / オーバーフロー検出は行われません。また、表 2-6 内のアキュムレータ ステータスビットは変更されません。ただし、MCU ステータスビット (Z、N、C、OV、DC) は、アキュムレータへアクセスする MCU 命令に応じて変更されます。

Note: 算術エラートラップの詳細は**セクション 6.「割り込み」(DS70184)** を参照してください。

2.6.3.3 データ領域書き込みの飽和処理

加算器 / 減算器の飽和処理に加えて、データ領域への書き込みではソース アキュムレータの内容に影響を与えない飽和処理が可能です。この機能を使用すると、演算の中間段階でアキュムレータのダイナミックレンジを犠牲にする事なくデータを制限できます。データ領域書き込み飽和処理を有効にするには、DSP エンジンからのデータ領域書き込み飽和処理イネーブル (SATDW) 制御ビット (CORCON<5>) をセットします。データ領域書き込み飽和処理は、デバイスリセット時に既定値として有効になります。

データ領域書き込み飽和処理機能は、SAC および SAC.R 命令で動作します。これらの命令の実行中は、アキュムレータ内のホールド値は変更されません。ハードウェアは下記の手順を実行して飽和書き込み結果を得ます。

1. 命令内で指定された算術シフト値に基づいて読み出しデータをスケーリングする。
2. スケーリングしたデータを丸め処理する (SAC.R のみ)。
3. スケーリング / 丸め処理後の値をガードビットの値に基づいて 16 ビット結果へ飽和させる (データ値が 0x007FFF よりも大きい場合、メモリへ書き込むデータを 1.15 フォーマットの正の最大値 0x7FFF へ飽和させる。データ値が 0xFF8000 よりも小さい場合、メモリへ書き込むデータを 1.15 フォーマットの負の最大値 0x8000 へ飽和させる)。

2.6.3.4 アキュムレータ「書き戻し」

MAC および MSC 命令は、必要に応じて実行中の演算のターゲットではない方のアキュムレータの丸め処理済みの内容をデータ領域メモリへ書き込む事ができます。書き込みは、X バスを経由して、X および Y アドレス領域を結合した範囲に対して実行されます。このアキュムレータ書き戻し機能は、特定の FFT および LMS アルゴリズムに有効です。

アキュムレータ書き戻しハードウェアは下記のアドレッシング モードをサポートします。

- W13、レジスタ直接 : ターゲットではない方のアキュムレータの丸め処理済みの内容を W13 へ 1.15 小数結果として書き込みます。
- [W13]+=2、レジスタ間接、事後インクリメント : ターゲットではない方のアキュムレータの丸め処理済みの内容を W13 が指すアドレスへ 1.15 小数として書き込みます。その後 W13 を 2 つインクリメントします。

2.6.4 丸め処理ロジック

丸め処理ロジックは、アキュムレータ書き込み (保存) 中に、標準 (バイアスあり) または収束 (バイアスなし) 丸め処理を実行可能です。丸め処理モードは、丸め処理モード選択 (RND) ビット (CORCON<1>) によって決まります。丸め処理ロジックは 16 ビットの 1.15 フォーマット データ値を生成し、このデータはデータ領域書き込み飽和処理ロジックへ渡されます。命令で丸め処理モードを指定しない場合は、切り捨て処理した 1.15 フォーマット データを保存します。

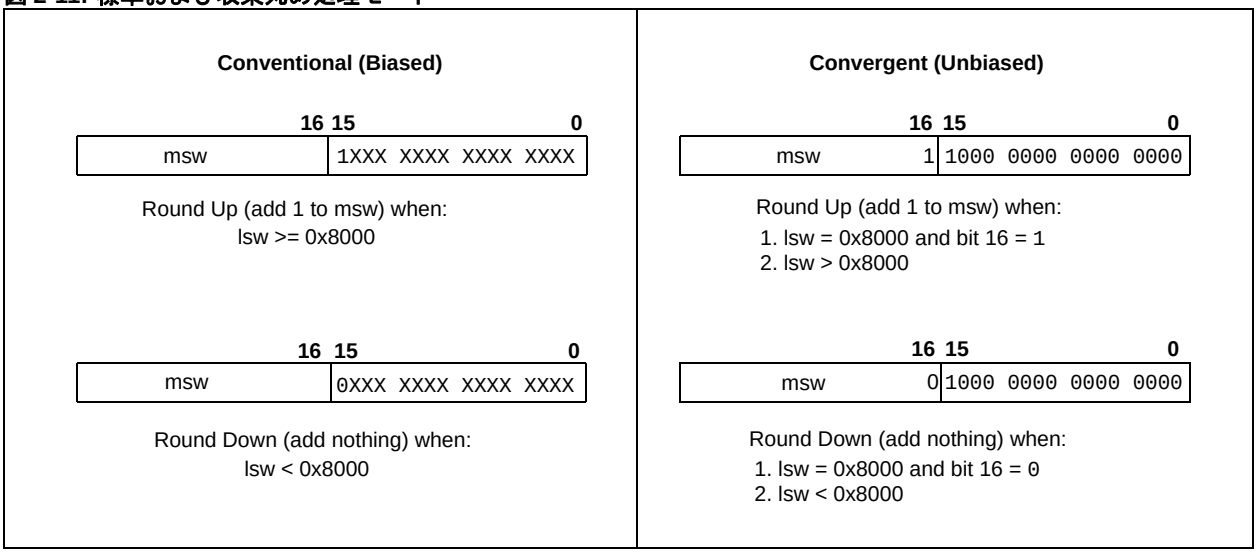
2 つの丸め処理モードを図 2-11 に示します。標準丸め処理モードは、アキュムレータのビット 15 をゼロ拡張し、これを最上位ワード (msw) へ加算します (ガードビットまたはオーバフロービット (ビット 16 ～ 31) を除く)。アキュムレータの最下位ワード (lsw) が 0x8000 ～ 0xFFFF の範囲内 (0x8000 を含む) にある場合は、msw をインクリメントします。アキュムレータの lsw が 0x0000 ～ 0x7FFF の範囲内にある場合には、msw は変化しません。このアルゴズムでは、ランダムな丸め処理誤差が累積して値が大きく正側へバイアスする傾向があります。

収束 (バイアスなし) 丸め処理ロジックは、lsw が 0x8000 の場合にのみ標準丸め処理と異なります。この場合、msw の LSb (アキュムレータのビット 16) を評価します。このビットが「1」であれば msw をインクリメントします。このビットが「0」であれば msw を変更しません。このスキームは、ビット 16 が事実上ランダムであるとの仮定の下に、丸め処理によるバイアスを除去し、その累積を防ぎます。

SACおよびSAC.R命令は、ターゲット アキュムレータの切り捨て(SAC)または丸め(SAC.R)処理された内容を、Xバス経由でデータメモリへ保存します(データ飽和処理を適用)。詳細は**2.6.3.3「データ領域書き込みの飽和処理」**を参照してください。

MAC クラス命令では、アキュムレータ書き戻しデータパスは常に丸め処理されます。

図 2-11: 標準および収束丸め処理モード



2.6.5 バレルシフタ

バレルシフタは、最大 16 ビットの算術右シフトまたは左シフトを 1 サイクルで実行します。DSP または MSU 命令では、バレルシフタを使用して複数ビットのシフトを行えます。

バレルシフタには、符号付きバイナリ値を使用してシフト量 (ビット数) とシフト方向を指定する必要があります。

- 正の値は右方へのシフトを意味します。
- 負の値は左方へのシフトを意味します。
- この値が「0」の場合には、オペランドを変更しません。

バレルシフタは 40 ビット幅を持ち、アキュムレータのビット幅に対応します。DSP シフト操作には 40 ビット結果、MCU シフト操作には 16 ビット結果を提供します。

表 2-7 にバレルシフタを使用する命令の概要を示します。

表 2-7: DSP エンジン バレルシフタを使用する命令

命令	内容
ASR	データメモリ位置の算術複数ビット右シフト
LSR	データメモリ位置の論理複数ビット右シフト
SL	データメモリ位置の複数ビット左シフト
SAC	任意のシフトで DSP アキュムレータを保存
SFTAC	DSP アキュムレータをシフト

2.6.6 DSP エンジンモードの選択

上記の各セクションに記載した DSP エンジンの各種動作モード (下記) は、CPU コア設定レジスタ (CORCON) で選択できます。

- 小数または整数乗算
- 標準または収束丸め処理
- ACCA 自動飽和処理の ON/OFF
- ACCB 自動飽和処理の ON/OFF
- データメモリ書き込み自動飽和処理の ON/OFF
- アキュムレータ飽和処理モードの選択

2.6.7 DSP エンジンのトラップイベント

DSP エンジンの例外処理には下記の算術エラートラップを使用できます。これらは割り込み制御レジスタ (INTCON1) で選択します。

- ACCA オーバーフローのトラップ、OVATE (INTCON1<10>) で有効化
- ACCB オーバーフローのトラップ、OVBTE (INTCON1<9>) で有効化
- ACCA および / または ACCB の致命的オーバーフローのトラップ、COVTE (INTCON1<8>) で有効化

下記のエラー ステータスビットがトラップの発生を示します。

- OVAERR (INTCON1<14>)
- OVBERR (INTCON1<13>)
- COVAERR (INTCON1<12>)
- COVBERR (INTCON1<11>)

ユーザ割り当てアプリケーションが SFTAC を使用して最大許容範囲 (± 16 ビット) を超える値のシフトを試みた場合にも、算術エラートラップが発生します。このトラップソースは無効化できません。このステータスは、シフト アキュムレータ ステータス (SFTACERR) ビット (割り込みコントローラ内の INTCON1<7>) によって示されます。この場合、命令を実行してもシフト結果はターゲット アキュムレータへ書き込まれません。INTCON1 レジスタ内の各種ビットおよび算術エラートラップの詳細は**セクション 6.「割り込み」**(DS70184) を参照してください。

2.7 除算サポート

dsPIC33F は下記タイプの除算をサポートします。

- DIVF: 16/16 符号付き小数除算
- DIV.SD: 32/16 符号付き除算
- DIV.UD: 32/16 符号なし除算
- DIV.SW: 16/16 符号付き除算
- DIV.UW: 16/16 符号なし除算

全ての除算結果の整数部 (商の整数部) はワーキング レジスタ W0 に格納します。商の余り (剰余) は W1 に格納します。16 ビット除数は任意のワーキング レジスタに格納できます。16 ビット被除数も任意のワーキング レジスタに格納できます。32 ビット被除数は連続する 2 つのワーキング レジスタに格納する必要があります。

全ての除算命令は、REPEAT ループ内で 18 回の繰り返し演算を行う必要があります。REPEAT 命令のプログラミングはユーザの責任で行ってください。1 回の除算を完了するには 19 回の命令サイクルを要します。

除算フローは一般的な REPEAT ループと同様に割り込み可能です。ループの毎回の繰り返しの後に、全てのデータは対応するデータレジスタへ復元されます。従ってユーザ割り当てアプリケーションは、ISR 内で適切なワーキング レジスタを保存する必要があります。ワーキング レジスタ内の演算途中の値は除算ハードウェアにとっては重要ですが、ユーザ割り当てアプリケーションにとっては意味を持ちません。正しい除算結果を得るには、除算命令を REPEAT ループ内で 18 回実行する必要があります。

ゼロ除算エラーは算術エラートラップを発生します。このエラートラップの発生は、算術エラーステータス (DIV0ERR) ビット (割り込みコントローラの INTCON1<6>) で示されます。除算命令の詳細とプログラミング例は「*dsPIC30F/33F プログラマ リファレンス マニュアル*」(DS70157) を参照してください。

2.8 命令フローのタイプ

dsPIC33F アーキテクチャ内の大部分の命令は、プログラムメモリの単一ワードを使用して単一サイクル内で実行します。命令のプリフェッチ処理がシングルサイクル (1 x TCY) 実行を容易にします。ただし、一部の命令の実行には 2 または 3 命令サイクルを要します。dsPIC®DSC アーキテクチャでは、下記 7 タイプの命令フローを使用します。本セクションではこれらの各タイプについて説明します。

- 1 命令ワード、1 命令サイクル
- 1 命令ワード、2 命令サイクル
- 1 命令ワード、2 または 3 命令サイクル (プログラムフローが変化)
- 1 命令ワード、3 命令サイクル (RETFIE、RETURN、RETLW)
- テーブル読み書き命令
- 2 命令ワード、2 命令サイクル
- アドレスレジスタ依存性

2.8.1 1 命令ワード、1 命令サイクル

このタイプの命令の実行には 1 サイクルしか要しません (図 2-12 参照)。命令の大部分は 1 ワード /1 サイクル命令です。

図 2-12: 命令フロー : 1 ワード、1 サイクル

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1. MOV #0x55AA,W0	Fetch 1	Execute 1				
2. MOV W0,PORTA		Fetch 2	Execute 2			
3. MOV W0,PORTB			Fetch 3	Execute 3		

2.8.2 1 命令ワード、2 命令サイクル

このタイプの命令にはプリフェッチ フラッシュは存在しません。各種 MOV.D 命令 (ダブルワードの読み込み / 保存) だけがこのタイプに属します。これらの命令の実行には 2 サイクルを要します (図 2-13 参照)。

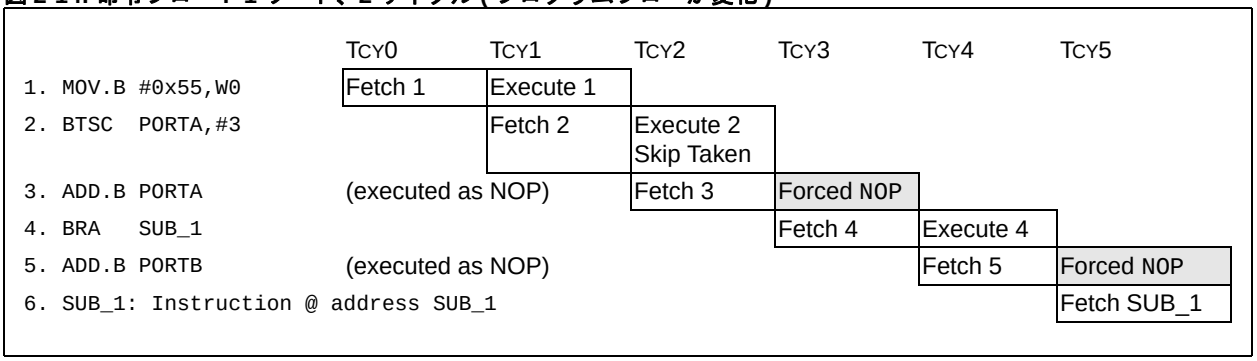
図 2-13: Instruction Flow: 1 ワード、2 サイクル (MOV.D 演算)

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1. MOV #0x1234,W0	Fetch 1	Execute 1				
2. MOV.D [W0++],W1		Fetch 2	Execute 2 R/W Cycle 1			
3. MOV #0x00AA,W1			Fetch 3	Execute 2 R/W Cycle2		
				No Fetch	Execute 3	
4. MOV #0x00CC,W0					Fetch 4	Execute 4

2.8.3 1 命令ワード、2 または 3 命令サイクル (プログラムフローが変化)

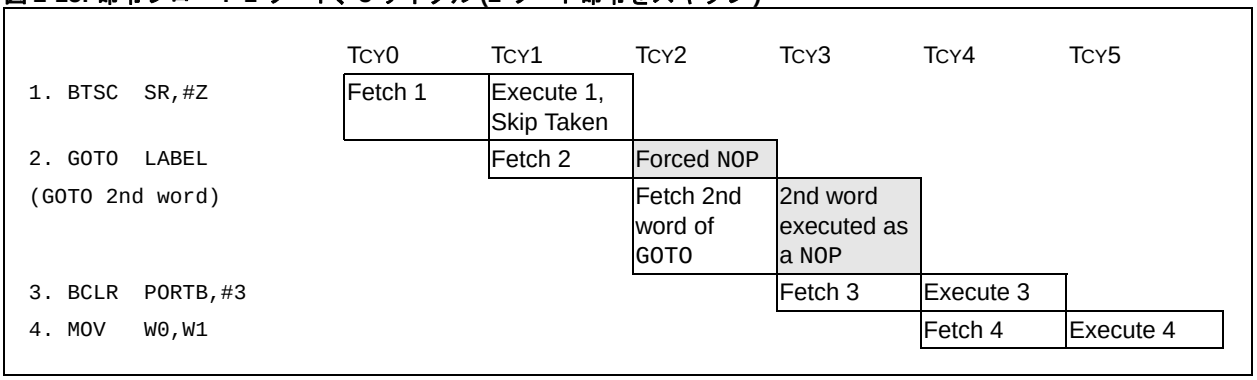
このタイプの命令には相対コール、分岐命令、スキップ命令が含まれます。命令が PC を変更した場合 (単純なインクリメントを除く)、プログラムメモリのプリフェッチ データを破棄する必要があります。命令はこれを行うにために 2 サイクルを要します (図 2-14 参照)。

図 2-14: 命令フロー : 1 ワード、2 サイクル (プログラムフローが変化)



2 ワード命令をスキップするには 3 サイクルが必要です。この場合プログラムメモリのプリフェッチ データを破棄し、2 ワード命令の第 2 ワードをフェッチします。図 2-15 に、命令の第 2 ワードを NOP として実行する様子を示します。

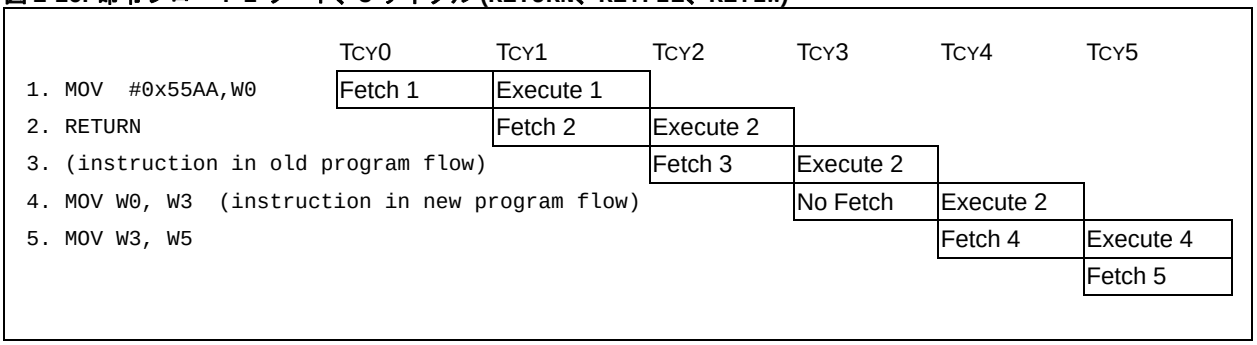
図 2-15: 命令フロー : 1 ワード、3 サイクル (2 ワード命令をスキップ)



2.8.4 1 命令ワード、3 命令サイクル (RETFIE、RETURN、RETLW)

図 2-16 に、RETFIE、RETURN、RETLW 命令を使用したサブルーチン コールまたは ISR からのリターンを示します。この実行には 3 命令サイクルを要します。

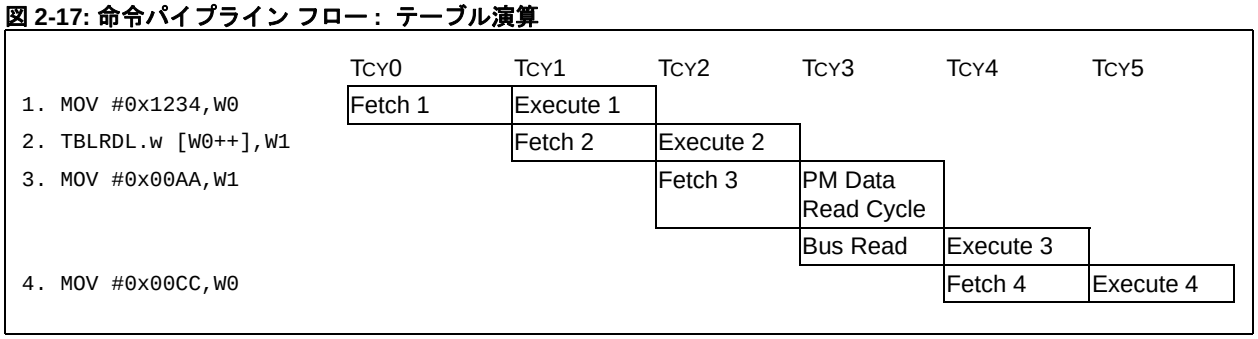
図 2-16: 命令フロー : 1 ワード、3 サイクル (RETURN、RETFIE、RETLW)



2.8.5 テーブル読み書き命令

このタイプの命令は、フェッチをサスペンドして 1 つの「読み出し」または「書き込み」サイクルをプログラムメモリへ挿入します。図 2-17 に、テーブル演算の実行中に命令をフェッチして 1 サイクル間保存し、これをテーブル演算直後のサイクルで実行する様子を示します。

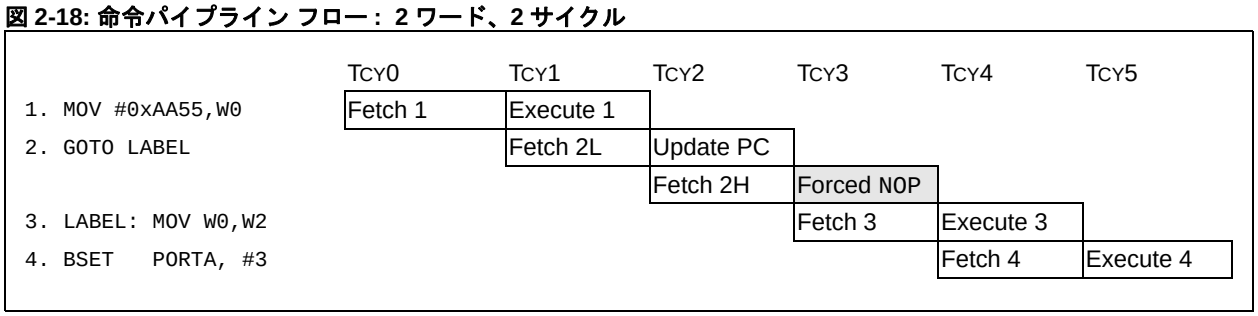
図 2-17: 命令パイプライン フロー : テーブル演算



2.8.6 2 命令ワード、2 命令サイクル

このタイプの命令では、命令後のフェッチがデータを格納します。このため、これらの命令は 2 サイクル命令となります (図 2-18 参照)。CPU が 2 ワード命令の第 1 ワードを最初にフェッチせずに第 2 ワードをフェッチする場合、第 2 ワードは NOP として実行するようにエンコードされます。これはスキップ命令を用いて 2 ワード命令をスキップする場合に重要です (図 2-15 参照)。

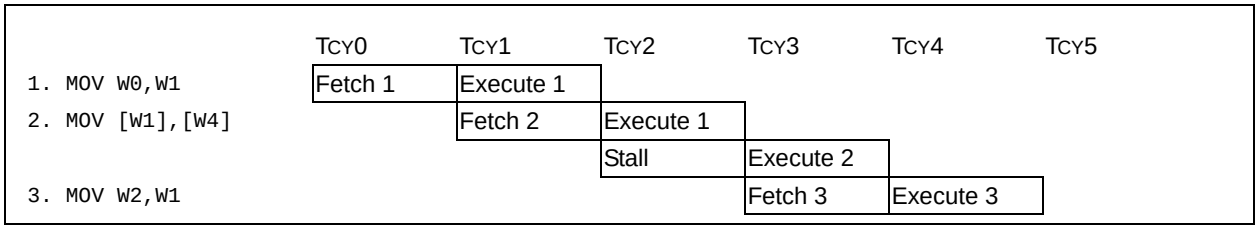
図 2-18: 命令パイプライン フロー : 2 ワード、2 サイクル



2.8.7 アドレスレジスタ依存性

このタイプの命令は、X データ領域の読み書き動作間のデータアドレス依存性のために、ストールを発生する可能性があります。ストールはリソースの衝突を解決するために追加の 1 サイクルを挿入します。詳細は **2.10「アドレスレジスタ依存性」**を参照してください。

図 2-19: 命令パイプライン フロー : 1 ワード、1 サイクル (命令ストールあり)



Note: RETURN 命令をプログラムメモリの終端に置くと、ランタイム中にデバイスが不正アドレス エラートラップを発生します。これはプリフェッチ動作がメモリの終端位置から次に続く 2 つの命令 (実際には存在しない) のプリロードを試みるためです。この問題を解消するには、RETURN 命令の後に 2 命令ワード分の領域を残して、コンパイラが NOP および RESET 命令をプログラムメモリの終端に格納できるようにする必要があります。

2.9 ループ構造

dsPIC33F は REPEAT および DO 命令構造の両方をサポートし、自動的な無条件ループ制御を提供します。REPEAT 命令は 1 つの命令だけを含むプログラムループを実行します。DO 命令は複数命令を含むプログラムループを実行します。両命令は、CPU ステータス レジスタ (SR) 内の制御ビットを使用して CPU 動作フローを一時的に変更します。

2.9.1 REPEAT ループ構造

REPEAT 命令は、後続の 1 つの命令を指定回数繰り返し実行します。REPEAT カウント値の指定には、命令内のリテラル値またはいずれかのワーキング レジスタ内の値を使用します。ワーキング レジスタを使用する場合には、ループカウントをソフトウェア変数として扱う事ができます。

REPEAT ループ内の命令は少なくとも 1 回実行されます。REPEAT ループの繰り返し数は、14 ビットリテラル値 + 1 または対応するワーキング レジスタの値 +1 です。

上記に対応する 2 つの REPEAT 命令構文を下に示します。

例 2-7: REPEAT 命令の構文

```
REPEAT  #lit14          ; RCOUNT <-- lit14
(Valid target Instruction)
or
REPEAT  Wn              ; RCOUNT <-- Wn
(Valid target Instruction)
```

2.9.1.1 REPEAT 動作

REPEAT動作のループカウンタは、メモリへ割り当てられた14ビットのREPEATループカウンタ (RCOUNT) レジスタが保持します。RCOUNT は REPEAT 命令によって初期化されます。RCOUNT の値がゼロではない場合、REPEAT 命令は REPEAT ループ アクティブ (RA) ステータスビット (SR<4>) を「1」にセットします。

RA は読み出し専用ビットであり、ソフトウェアから変更する事はできません。REPEAT ループのカウント値が「0」より大きい場合には、PC をインクリメントしません (PC は RCOUNT = 0 になるまでインクリメントされません)。REPEAT ループ命令フローの例を図 2-20 に示します。

ループカウント値が「0」である場合、REPEAT 命令は NOP 命令と同じ効果を持ち、RA (SR<4>) ビットをセットしません。REPEAT ループは開始前には実質的に無効化されます。このためターゲット命令は、後続命令のプリフェッチ中に 1 度だけ実行されます (すなわち通常の実行フロー)。

Note:	REPEAT 命令直後の命令 (すなわちターゲット命令) は必ず 1 回以上実行され、その実行回数は 14 ビットリテラル値またはワーキング レジスタのオペランドで指定した値より 1 回多くなります。

図 2-20: REPEAT 命令のパイプライン フロー

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1.REPEAT #0x2	Fetch 1	Execute 1				
2.MAC W4*W5,A,[W8]+=2,W4		Fetch 2	Execute 2			
			No Fetch	Execute 2		
				No Fetch	Execute 2	
3.BSET PORTA,#3					Fetch 3	Execute 3

PC (at end of instruction)	PC	PC+2	PC+2	PC+2	PC+4	PC+6
RCOUNT (at end of instruction)	X	2	1	0	0	0
RA (at end of instruction)	0	1	1	0	0	0

2.9.1.2 REPEAT ループの中断

REPEAT 命令ループには常時割り込み可能です。RA ビットのステートは例外処理中にスタック上で保存されます。このためユーザ割り当てアプリケーションは、任意数にネスティングされた割り込み内から別の REPEAT ループを実行できます。SRL レジスタがスタックされると、RA ステータスビットをクリアして ISR 内で通常実行フローを復元します。

Note 1: REPEAT ループに割り込みが発生して ISR が処理中である場合、ユーザ割り当てアプリケーションは ISR 内で別の REPEAT 命令を実行する前に REPEAT カウント (RCOUNT) レジスタをスタックする必要があります。

2: REPEAT 命令を ISR 内で使用する場合、ユーザ割り当てアプリケーションは RETFIE 命令を実行する前に RCOUNT レジスタをアンスタックする必要があります。

RETFIE 命令を使用して ISR から REPEAT ループへ戻るには特別な操作は不要です。割り込みは、RETFIE 命令の 3 サイクル目の実行中に、繰り返し実行する命令をプリフェッチします。スタックされた RA ビットは、SRL レジスタがポップされた時に復元します。この RA ビットがセットされた状態であれば、中断していた REPEAT ループを再開します。

Note: 繰り返し実行する命令 (すなわち REPEAT ループ内のターゲット命令) は、PSV を使用してプログラム領域 (PS) 内のデータにアクセスします。例外処理から復帰した後の初回の実行では、この動作に 2 命令サイクルを要します。ループの初回の実行と同様に、復帰後の初回の命令も 1 命令サイクルでプログラム領域内のデータへアクセスする事はできません (タイミング上の制限による)。

2.9.1.2.1 REPEAT ループの早期終了

中断中の REPEAT ループは、ソフトウェアで RCOUNT をクリアする事によって、ISR 内で通常より早期に終了できます。

2.9.1.3 REPEAT 命令に関する制限事項

REPEAT の直後に下記の命令を実行する事はできません。

- プログラムフロー制御命令 (分岐、比較 / スキップ、サブルーチン コール、リターン等)
- 別の REPEAT または DO 命令
- DISI、ULNK、LNK,PWRSAV、RESET
- MOV.D 命令

Note: ループ内で実行可能な命令またはアドレスモードもありますが、反復しても意味のないものがあります。

2.9.2 DO ループ構造

DO 命令は、複数の命令をソフトウェア オーバーヘッドなしで指定回数実行します。終了アドレスまでの (終了アドレスを含む) 一連の命令を繰り返し実行します。DO 命令の DO 繰り返しカウント値は、14 ビットのリテラル値または命令内で宣言したワーキング レジスタの内容を使用して指定します。14 ビットリテラル値を使用する合の DO 命令構文を例 2-8 に示します。

例 2-8: 14 ビット リテラル値を使用する場合の DO 命令構文

```
            DO      #lit14,LOOP_END        ; DCOUNT <-- lit14
            Instruction1
            Instruction2
            :
            :
LOOP_END:   Instruction n
```

ワーキング レジスタを宣言する場合の DO 命令構文を例 2-9 に示します。

例 2-9: ワーキング レジスタを宣言する場合の DO 命令構文

```
            DO      Wn,LOOP_END            ; DCOUNT <-- Wn<13:0>
            Instruction1
            Instruction2
            :
            :
LOOP_END:   Instruction n
```

DO ループ構造は下記の特徴を備えます。

- ワーキング レジスタによるループカウントの指定 (ランタイム中にループカウントを定義可能)
- 繰り返し命令が順番に並んでいる必要はない (分岐、サブルーチン コールが間に存在しても構わない)
- ループの終了アドレスは開始アドレスより小さくても構わない

2.9.2.1 DO ループのレジスタと動作

DO ループの繰り返し数は、14 ビットリテラル値 + 1 またはワーキング レジスタ 値 + 1 です。ワーキング レジスタを使用して繰り返し数を指定する場合、最上位の 2 ビットはループカウントの指定に使用しません。DO ループの動作は C 言語の DO-WHILE 構造に類似し、ループ内の命令を必ず 1 回以上実行します。

dsPIC33F は、DO ループに関連する下記 3 つのレジスタを備えます。

- DO ループ開始アドレス (DOSTART) レジスタ : DO ループの開始アドレスを保持する 22 ビットレジスタ
- DO ループ終了アドレス (DOEND) レジスタ : DO ループの終了アドレスを保持する 16 ビットレジスタ
- DO ループカウンタ (DCOUNT) レジスタ : ループの実行回数を保持する 16 ビットレジスタ

これらのレジスタはメモリへ割り当てられ、DO 命令実行時にハードウェアが自動的に読み込みます。これらのレジスタの MSb と LSb は「0」に設定されます。PC<0> は常に強制的に「0」に設定されるため、LSb はこれらのレジスタに保存されません。

DOループ アクティブ(DA)ステータスビット(SR<9>)は、1つのDOループ(またはネスティングされた DO ループ) がアクティブである事を示します。DO 命令を実行すると、DA ビットがセットされ、以降の各命令サイクルで PC アドレスと DOEND レジスタの比較が可能になります。PC が DOEND 内の値に一致すると、DCOUNT をデクリメントします。

DCOUNT レジスタがゼロではない場合、PC に DOSTART レジスタ内のアドレスを読み込んで DO ループの次の繰り返しを開始します。DCOUNT がゼロになると DO ループは終了します。

ネスティングされた他の DO ループが実行中でなければ、DA ビットもクリアされます。

Note: DO ループ構造内の一連の命令は必ず 1 回以上実行されます。DO ループの実行回数は、リテラル値またはワーキング レジスタ オペランドで指定した値よりも 1 回多くなります。

2.9.2.2 DO ループのネスティング

DOSTART、DOEND、DCOUNT レジスタは、それぞれ 1 つのシャドーレジスタを備えているため、DO ループ ハードウェアは 1 レベルの自動的なネスティングをサポートします。ユーザは DOSTART、DOEND、DCOUNT へアクセスできます。これらのレジスタを手動で保存する事により、必要に応じてネスティングを追加する事ができます。

DOループ ネスティング レベル(DL<2:0>)ステータスビット(CORCON<10:8>)は、現在実行中の DO ループのネスティング レベルを示します。最初の DO 命令を実行すると、DL<2:0> が B「001」へセットされ、1 レベルの DO ループが実行中である事を示します。DO ループのアクティブ (DA) ビット (SR<9>) もセットされます。

最初の DO ループ内で別の DO 命令を実行すると、DOSTART、DOEND、DCOUNT レジスタの値をシャドーレジスタへ保存してから、それらのレジスタの値を新たなループ値へ変更します。DL<2:0> ビットは B「010」へセットされ、ネスティングされた第 2 の DO ループが実行中である事を示します。DA (SR<9>) ビットもセット状態を維持します。

アプリケーションが複数レベルの DO ループ ネスティングを必要とする場合でも、特別な配慮は不要です。ただし、ユーザ割り当てアプリケーションが複数レベルの DO ループ ネスティングを必要とする場合には、新たな DO 命令を実行する前に DOSTART、DOEND、DCOUNT レジスタを保存する必要があります。これらのレジスタは、DL<2:0> が B「010」以上の場合には常に保存する必要があります。

DL<2:0> = B「010」の時に DO ループが終了すると、DOSTART、DOEND、DCOUNT レジスタは自動的に各シャドーレジスタから復元されます。

Note: DL<2:0> (CORCON<10:8>) の各ビットは、論理 OR で結合されて DA (SR<9>) ビットを形成します。ネスティングされた DO ループが実行中である場合、DA ビットは最も外側のループに対応するループカウントが終了するまでクリアされません。

2.9.2.3 DO ループの中断

DO ループへは常時割り込み可能です。ISR 中に別の DO ループを実行する場合、ユーザ割り当てアプリケーションは DL<2:0> ステータスビットをチェックし、必要に応じて DOSTART、DOEND、DCOUNT レジスタを保存する必要があります。

ユーザ割り当てアプリケーションが下記において DO ループが 1 レベルでのみ実行される事を保証できるのであれば、特別な操作は不要です。

- バックグラウンドといずれか 1 つの ISR ハンドラの両方 (割り込みネスティング有効の場合)、または、
- バックグラウンドと任意の ISR (割り込みネスティング無効の場合)

あるいはバックグラウンドもしくは下記のいずれかでは、最大 2 つの (ネスティングされた) DO ループを実行可能です。

- 1 つの ISR ハンドラ (割り込みネスティング有効の場合)、または、
- 任意の ISR(割り込みネスティング無効の場合)

トラップハンドラ内で DO ループを使用しない事を前提とします。

RETFIE 命令を使用して ISR から DO ループへ戻るには、特別な操作は不要です。

2.9.2.4 DO ループの早期終了

DO ループを通常より早期に終了させるには、下記の 2 つの方法があります。

- 早期 DO ループ終了制御 (EDT) ビット (CORCON<11>) を使用すると、指定回数のループを終了する前にユーザ割り当てアプリケーションから DO ループを終了させる事ができます。EDT ビットへ「1」を書き込むと、実行中の繰り返しが完了してからループが終了します。ループ内の最後から 2 番目あるいは 1 番最後の命令を実行中に EDT をセットすると、ループはその繰り返しでは終了せず、次の繰り返しを実行してから終了します。EDT は常に「0」として読まれ、クリアしても効果はありません。EDT ビットをセットした後に、ユーザは必要に応じて DO ループから外部へ分岐きます。
- あるいは、コードは最後の命令を除く任意の位置でループから外部へ分岐できます (最後の命令をフロー制御命令にする事はできません)。DA (SR<9>) ビットは DO ループ ハードウェアを有効にしますが、命令プリフェッチの間に、最後から 2 番目の命令のアドレスに遭遇しない限り効果はありません。これは DO ループの終了方法として推奨しません。

Note:	EDTを使用せずにDOループを終了する事は推奨しません(ハードウェアがDOENDアンドレスをチェックし続けるため)。

2.9.2.5 DO ループに関する制限事項

DO ループの使用には下記の制限が課せられます。

- DOEND レジスタが読み出し可能である事
- 一部の命令をループ内の最後の命令として使用しない事
- ループ長が短かすぎない事 (表 2-8 参照 - ループ長とは、ループ内で繰り返し実行する命令ブロックのサイズを指します)

2.9.2.5.1 DOEND レジスタに関する制限事項

DO ループは複数の命令を含む必要があります。これは最後から 2 番目の命令でループ終了チェックを行う必要があるためです。単一命令のループには REPEAT を使用する必要があります。

ユーザ ソフトウェアは、DO 命令または DOEND SFR へのファイルレジスタ書き込み動作の直後の命令で SFR と DOEND を読み出す事はできません。

DO ループ内の最後から 2 番目の命令の前の命令は、下記を変更してはなりません。

- CPU IPL ステータスビット (SR<7:5>) に従う CPU 優先度
- 割り込みイネーブル制御レジスタ (IEC0、IEC1、IEC2) に従う周辺モジュール割り込みイネーブルビット
- 割り込み優先度制御レジスタ (IPC0 ～ IPC11) に従う周辺モジュール割り込み優先度ビット

これらの制限事項を守らないと、DO ループが正常に動作しない可能性があります。

2.9.2.5.2 最後の命令に関する制限事項

DO ループ内の最後の命令に下記の命令を使用する事はできません。

- フロー制御命令 (例 : 分岐、比較 / スキップ、GOTO、CALL、RCALL、TRAP)
- 別の REPEAT または DO 命令
- REPEAT ループ内のターゲット命令 (この制限は、最後から 2 番目の命令にも REPEAT を使用できない事を意味します)
- プログラム領域内で 2 ワードを使用する全ての命令
- DISI 命令

RETURN、RETFIE、RETLW は、DO ループの最後の命令として正常に動作します。ただしユーザ割り当てアプリケーションは、ループへ戻ってこれらの命令を完了する必要があります。

2.9.2.5.3 ループ長に関する制限事項

ループ長は、DO ループ内の先頭の命令から最後の命令までの符号付きオフセットとして定義されます。ループ内の先頭命令のアドレスにループ長を加算すると、最後の命令のアドレスが得られます。表 2-8 に、回避すべきループ長の一覧を記載します。

表 2-8: 回避すべきループ長

ループ長	理由

-2	ループの実行は、ループ内の先頭命令 (PC アドレス) から始まり、ループ終了アドレス (この場合 [PC-4]) がプリフェッチされるまで続きます。これは DO 命令の先頭ワードであるため、DCOUNT の再初期化と [PC] のプリフェッチを行って DO 命令を再び実行します。これはループ終了アドレス [PC-4] がプリフェッチされ続ける限り永遠に終了しません。このループ長は無限ループを形成する能性があります (ウォッチドッグ タイマリセットを発生)。
	例 :
	`end_loop: DO #33, end_loop ;DO is a two-word instruction`
	`NOP ;2nd word of DO executes as a NOP`
	`ADD W2,W3,W4 ;First instruction in DO loop([PC])`

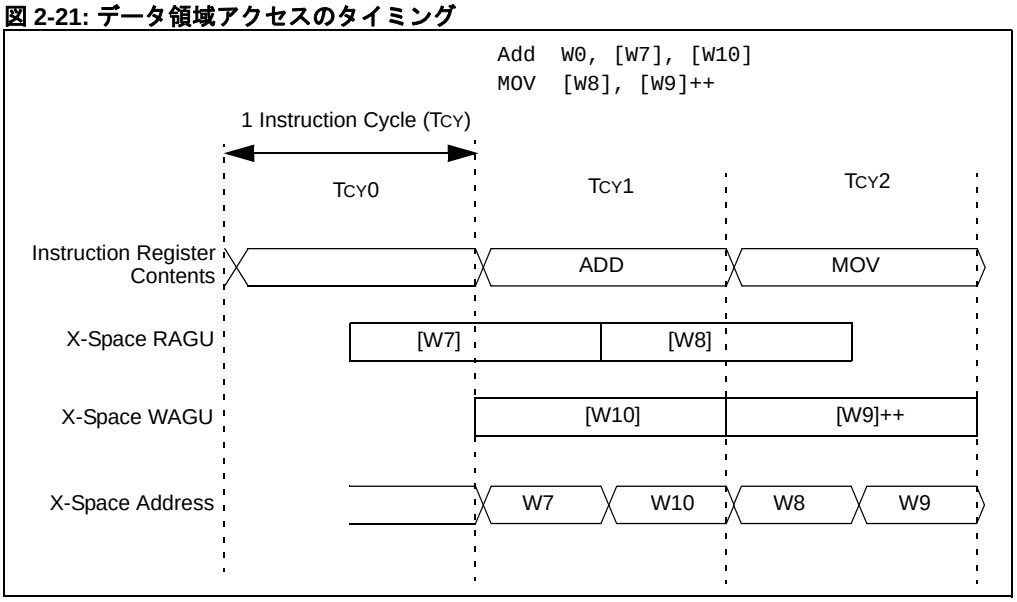
-1	ループの実行は、ループ内の先頭命令 ([PC] アドレス) から始まり、ループ終了アドレス (この場合 [PC-2]) がプリフェッチされるまで続きます。ループ終了アドレスは DO 命令の第 2 ワードであるため、NOP として実行されますが、依然として [PC] をプリフェッチします。そしてループを再び実行します。これはループ終了アドレス [PC-2] がプリフェッチされる限り続くため、ループは終了しません。DCOUNT レジスタ内の値がゼロになり、次のデクリメントでボローが発生すると、ループは終了します。ただしこの場合、ループから出た後の最初の命令は再びループの先頭命令となります。
	例 :
	`DO #33, end_loop ;DO is a two-word instruction`
	`end_loop: NOP ;2nd word of DO executes as a NOP`
	`ADD W2,W3,W4 ;First instruction in DO loop([PC])`

0	ループの実行は、ループ内の先頭命令 ([PC] アドレス) から始まり、ループ終了アドレス ([PC]) がプリフェッチされるまで続きます。ループが続行する場合、このプリフェッチによって、DO ループハードウェアが次のフェッチのために DOEND アドレス ([PC]) を PC へ読み込むという事態が発生します。初回のループを正常に実行した後、ループカウンタがアンダーフローしてループが終了するまで、ループ内の先頭命令が繰り返し実行されます。この状況では、ループから出た後の最初の命令は [PC] 後の命令となります。
	例 :
	`DO #33, end_loop ;DO is a two-word instruction`
	`NOP ;2nd word of DO executes as a NOP`
	`end_loop: ADD W2,W3,W4 ;First instruction in DO loop([PC])`

2.10 アドレスレジスタ依存性

dsPIC33F アーキテクチャは、大部分の MCU クラス命令に対して、データ領域の読み出し (ソース) と書き込み (デスティネーション) をサポートします。AGU による有効アドレス (EA) 計算と、これに続くデータ領域の読み書きの実行には、各々 1 命令サイクルを要します。このようなタイミングにより、各命令におけるデータ領域の読み書き動作が部分的にオーバーラップする状態が発生します (図 2-21 参照)。オーバーラップが発生すると、書き込み後読み出し (RAW) データ依然性が命令と命令の間で発生する可能性があります。dsPIC33F はランタイム中に RAW データ依存性を検出して対処します。

図 2-21: データ領域アクセスのタイミング



2.10.1 読み出し後書き込み (RAW) 依存性に関する動作

実行中の命令内でワーキング レジスタを書き込み動作のデスティネーションとして使用し、プリフェッチした命令が同じワーキング レジスタを読み出す場合の動作は下記に従います。

- デスティネーションへの書き込み (実行中の命令) がワーキング レジスタの内容を変更しない場合、ストールは発生しない
- ソースの読み出し (プリフェッチした命令) がワーキング レジスタを使用して EA の計算を行わない場合、ストールは発生しない

dsPIC33F ハードウェアは、各命令の実行中に、RAW データ依存性が発生する可能性を自動的にチェックします。上記の条件が満たされない場合、CPU はプリフェッチした命令を実行する前に、1 命令サイクルの遅延を自動的に挿入します。このような命令ストールにより、次の (プリフェッチした) 命令が読み出しを行う前に、デスティネーション ワーキング レジスタへの書き込みに十分な時間を確保する事が可能になります。表2-9にRAW依然性の概要を記載します。

表 2-9: 書き込み後読み出し依然性の概要

ワーキング レジスタを使用するデスティネーション アドレッシング モード	ワーキング レジスタを使用するソース アドレッシング モード	ステータス	例 (Wn = W2)
直接	直接	可能	`ADD.w W0, W1, W2` `MOV.w W2, W3`
直接	間接	ストール	`ADD.w W0, W1, W2` `MOV.w [W2], W3`
直接	間接、変更あり	ストール	`ADD.w W0, W1, W2` `MOV.w [W2++], W3`
間接	直接	可能	`ADD.w W0, W1, [W2]` `MOV.w W2, W3`
間接	間接	可能	`ADD.w W0, W1, [W2]` `MOV.w [W2], W3`
間接	間接、変更あり	可能	`ADD.w W0, W1, [W2]` `MOV.w [W2++], W3`
間接、変更あり	直接	可能	`ADD.w W0, W1, [W2++]` `MOV.w W2, W3`

間接	間接	ストール	`ADD.w W0, W1, [W2]`
			`MOV.w [W2], W3`
			`; W2=0x0004 (mapped W2)`

間接	間接、変更あり	ストール	`ADD.w W0, W1, [W2]`
			`MOV.w [W2++], W3`
			`; W2=0x0004 (mapped W2)`

間接、変更あり	間接	ストール	`ADD.w W0, W1, [W2++]` `MOV.w [W2], W3`
間接、変更あり	間接、変更あり	ストール	`ADD.w W0, W1, [W2++]` `MOV.w [W2++], W3`

2.10.2 命令ストールサイクル

命令ストールは基本的に、命令読み出しフェイズの前に 1 命令サイクルのウェイト期間を追加する事によって、次の動作の前に書き込みを完全に行えるようにします。割り込みレイテンシを目的とする場合、ストールサイクルは割り込みが検出された命令の次の命令に関連付けられます (すなわちストールサイクルは常に命令実行サイクルに先行します)。

dsPIC33F は RAW データ依存性を検出して命令ストールを開始します。命令ストール中の動作は下記の通りです。

- 直前の命令による書き込み動作を完全に実行
- 命令ストールが終了するまでデータ領域をアドレスしない
- 命令ストールが終了するまで PC インクリメントを行わない
- 命令ストールが終了するまで、以降の命令フェッチを行わない

2.10.2.1 命令ストールサイクルと割り込み

命令ストールを発生する 2 つの連続した命令と割り込みイベントが同時に生じた場合の動作は、下記のいずれかに従います。

- 割り込みが最初の命令と同時に発生した場合、その命令を完全に実行し、その後 ISR を完了してから次の命令を実行します。この場合、例外処理は最初の命令の書き込みフェイズを完了するために十分な時間を提供するため、ストールサイクルは省略されます。
- 割り込みが 2 番目の命令と同時に発生した場合、ISR を実行する前にストールサイクルと 2 番目の命令を実行します。この場合、ストールサイクルは通常通り 2 番目の命令の実行に関連付けられます。ただし、ストールサイクルは効率的に例外処理タイミングへ取り込まれます。例外処理は、あたかも通常の 1 サイクル命令または 2 サイクル命令が割り込まれたかのように動作します。

2.10.2.2　命令ストールサイクルとフロー変更命令

CALL および RCALL 命令は、ワーキング レジスタ W15 を使用してスタックへの書き込みを行います。従ってこれらの命令は、次の命令のソース読み出しが W15 を使用する場合に、次の命令の前に命令ストールを強制する事ができます。

RETFIE および RETURN 命令は読み出し動作しか行わないため、次の命令の前に命令ストールを強制する事はできません。ただし、RETLW 命令は最後のサイクルでワーキング レジスタへの書き込みを行うため、ストールを強制する事ができます。

GOTO および分岐命令は書き込み動作を行わないため、命令ストールを強制する事はできません。

2.10.2.3　命令ストールと DO および REPEAT ループ

RAW データ依存性は、命令ストールを追加する事以外には、DO または REPEAT ループに影響しません。

REPEAT ループ内でプリフェッチした命令は、ループが終了するか例外が発生するまで変化しません。命令と命令の間でレジスタ依存性チェックが発生しますが、dsPIC33F は REPEAT ループ中に同一命令のソースとデスティネーションを効果的に比較します。

DO ループの最後の命令は、ループ開始アドレスの命令または次の命令 (ループ外部の命令) のいずれかをプリフェッチします。この際、命令ストールを行うかどうかは、ループ内の最後の命令とプリフェッチした命令の内容に基づいて判定します。

2.10.2.4　命令ストールと PSV (PROGRAM SPACE VISIBILITY)

PSV (CORCON<2>) ビットを有効にしてプログラム領域 (PS) をデータ領域へ割り当て、かつ X 領域 EA が可視プログラム領域ウィンドウの範囲内に入る場合、読み出しサイクルはプログラム領域内のアドレスにリダイレクトされます。プログラム領域からのデータへのアクセスには、最大 3 命令サイクルを要します。

PSV アドレス領域内で動作する命令は、他の命令と同様に、RAW データ依存性 (結果として命令ストール) を生じる可能性があります。例 2-10 のサンプルコードを参照してください。

例 2-10: PSV 上で動作するサンプルコード

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

この一連の命令の実行には 5 命令サイクルを要します。W1 経由で PSV アクセスを実行するために 2 命令サイクルが追加され、さらに、W2 によって生じる RAW データ依存性を解消するために 1 命令ストールサイクルが追加されます。

2.11 レジスタマップ

dsPIC33F のレジスタマップの概要を表 2-10 に記載します。

表 2-10: CPU レジスタマップ

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	全リセット
W0	W0 (WREG)																0000 0000 0000 0000
W1	W1																0000 0000 0000 0000
W2	W2																0000 0000 0000 0000
W3	W3																0000 0000 0000 0000
W4	W4																0000 0000 0000 0000
W5	W5																0000 0000 0000 0000
W6	W6																0000 0000 0000 0000
W7	W7																0000 0000 0000 0000
W8	W8																0000 0000 0000 0000
W9	W9																0000 0000 0000 0000
W10	W10																0000 0000 0000 0000
W11	W11																0000 0000 0000 0000
W12	W12																0000 0000 0000 0000
W13	W13																0000 0000 0000 0000
W14	W14																0000 0000 0000 0000
W15	W15																0000 0000 0000 0000
SPLIM	SPLIM																0000 0000 0000 0000
ACCAL	ACCAL																0000 0000 0000 0000
ACCAH	ACCAH																0000 0000 0000 0000
ACCAU	ACCA<39> の符号拡張								ACCAU								0000 0000 0000 0000
ACCBL	ACCBL																0000 0000 0000 0000
ACCBH	ACCBH																0000 0000 0000 0000
ACCBU	ACCB<39> の符号拡張								ACCBU								0000 0000 0000 0000
PCL	PCL																0000 0000 0000 0000
PCH	—	—	—	—	—	—	—	—	—	PCH						0	0000 0000 0000 0000
TBLPAG	—	—	—	—	—	—	—	—	TBLPAG								0000 0000 0000 0000
PSVPAG	—	—	—	—	—	—	—	—	PSVPAG								0000 0000 0000 0000
RCOUNT	RCOUNT																xxxx xxxx xxxx xxxx
DCOUNT	DCOUNT																xxxx xxxx xxxx xxxx
DOSTARTL	DOSTARTL															0	xxxx xxxx xxxx xxx0
DOSTARTH	—	—	—	—	—	—	—	—	—	—	DOSTARTH						0000 0000 00xx xxxx
DOENDL	DOENDL															0	xxxx xxxx xxxx xxx0

凡例： x = リセット時に不明の値、— = 未実装、「0」として読み出し、リセット値は 16 進数で表記

Note: 各コアレジスタマップの詳細はデバイスのデータシートを参照してください。

表 2-10: CPU レジスタマップ (続き)

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	全リセット
DOENDH	—	—	—	—	—	—	—	—	—	—	DOENDH						0000 0000 00xx xxxx
SR	OA	OB	SA	SB	OAB	SAB	DA	DC	IPL2	IPL1	IPL0	RA	N	OV	Z	C	0000 0000 0000 0000
CORCON	—	—	—	US	EDT	DL<2:0>			SATA	SATB	SATDW	ACCSAT	IPL3	PSV	RND	IF	0000 0000 0010 0000
MODCON	XMODEN	YMODEN	—	—	BWM<3:0>				YWM<3:0>				XWM<3:0>				0000 0000 0000 0000
XMODSRT	XMODSRT<15:0>															0	xxxx xxxx xxxx xxx0
XMODEND	XMODEND<15:0>															1	xxxx xxxx xxxx xxx1
YMODSRT	YMODSRT<15:0>															0	xxxx xxxx xxxx xxx0
YMODEND	YMODEND<15:0>															1	xxxx xxxx xxxx xxx1
XBREV	BREN	XBREV<14:0>															xxxx xxxx xxxx xxxx
DISICNT	—	—	DISICNT<13:0>														0000 0000 0000 0000

凡例 : x = リセット時に不明の値、— = 未実装、「0」として読み出し、リセット値は 16 進数で表記

Note: 各コアレジスタマップの詳細はデバイスのデータシートを参照してください。

2.12 関連アプリケーション ノート

本セクションに関連するアプリケーション ノートの一覧を下に記載します。一部のアプリケーション ノートは dsPIC33F 製品ファミリ向けではありません。ただし概念は共通しており、変更が必要であったり制限事項が存在するものの利用が可能です。本CPUモジュールに関連する最新のアプリケーション ノートは以下の通りです。

タイトル	アプリケーション ノート番号

現在、関連するアプリケーション ノートはありません。

Note: dsPIC33F デバイス ファミリ向けのその他のアプリケーション ノートとサンプル コードについては、マイクロチップ社のウェブサイト (www.microchip.com) をご覧ください。

2.13 改訂履歴

リビジョン A (2007 年 4 月)

本書の初版

リビジョン B (2009 年 9 月)

このリビジョンでの変更内容は次の通りです。

- Note:
  - **2.8.7「アドレスレジスタ依存性」**にプログラムメモリ エラーに関する注釈を追加
- レジスタ :
  - **2.11「レジスタマップ」**内の表 2-10 の「アドレス」列を削除
- セクション :
  - **2.1「はじめに」**内の記述を右記に変更 : プログラムフローを変更する命令、2 ワード移動命令 (MOV.D)、テーブル命令、PSV (Program Space Visibility) へアクセスする命令の実行には複数サイクルを要しますが、その他の命令は全てシングルサイクル命令です。
  - **2.9.2「DO ループ構造」**内の「REPEAT カウント値」を「DO 繰り返しカウント値」に変更
  - **2.10.2.1「命令ストールサイクルと割り込み」**内の例外処理に関する記述を右記に変更 : 例外処理は、あたかも通常の 1 サイクル命令または 2 サイクル命令が割り込まれたかのように動作します。
  - **2.10.2.4「命令ストールと PSV (Program Space Visibility)」**内の「読み出しまたは書き込みサイクル」を「読み出しサイクル」に変更
- テーブル :
  - **2.6.2.2「MCU 乗算命令」**内の乗算オプションに関するテーブル (表 2-5) を追加
- 外部文書への参照の誤りを訂正
- 表現および体裁の変更等、本書全体の細部を修正

ISBN: 978-1-60932-448-3

NOTES: