TEXAS INSTRUMENTS

参 考 資 料

TMS320C55x DSP CPU

リファレンス・ガイド

2004

Micro Controller

SPRU371 翻訳版

JAJU024 WAS SPRU826

TMS320C55x DSP CPU
リファレンス・ガイド

TEXAS INSTRUMENTS

ご注意

日本テキサス・インスツルメンツ株式会社及びTexas Instruments Incorporated（以下TIといいます）は、TI所定の手続きに従い、あるいはお客様とTIとの間に取引契約が締結されている場合は当該契約条件に従い、その製品を変更し、もしくは製品の製造中止またはサービスの提供を中止することがありますので、お客様は、発注される前に、これから参照しようとする情報が最新かつ完全なものであることを確実なものとするため、最新版の情報を取得するようお勧めします。全ての製品は、お客様とTIとの間に取引契約が締結されている場合は、当該契約条件に基づき、また取引契約が締結されていない場合は、ご注文の受諾の際に提示される保証、特許侵害、責任制限に関する条項を含むTIの標準販売契約約款に従って販売されます。

TIは、その製品が、TIの標準保証条件に従い販売時の仕様書に対応した性能を有していること、またはお客様とTIとの間で合意された保証条件に従い合意された仕様書に対応した性能を有していることを保証します。検査およびその他の品質管理技法は、TIが当該保証を支援するのに必要とみなす範囲で行なわれております。各デバイスの全てのパラメーターに関する固有の検査は、政府がそれ等の実行を義務づけている場合を除き、必ずしも行なわれておりません。

TI製部品を使用しているお客様の製品についてはお客様が責任を負っています。

そのようなお客様の製品について想定されうる危険を最小のものとするため、製品固有の障害発生要因もしくは組み合せによる障害発生要因を減らすための、適切な設計上および操作上の安全対策は、必ずお客様にてお取り下さい。

TIは、製品の使用用途に関する支援もしくはお客様の製品の設計について責任を負うことはありません。TIは、その製品もしくはサービスが使用されうる、もしくは使用されている組み合せ、機械装置、もしくは方法をカバーしている、もしくはそれ等に関連している特許権、著作権、回路配置利用権、その他の知的財産権に基づいて何らかのライセンスを許諾するということは明示的にも黙示的にも保証も表示もしておりません。TIが第三者の製品もしくはサービスについて情報を提供しているということは、TIが当該製品もしくはサービスを承認、ライセンス、保証もしくは支持するということを意味しません。

TIのデータ・ブックもしくはデータ・シートの中にある情報を複製することは、その情報に一切の変更を加えること無く、且つその情報と結び付られた全ての保証、条件、制限及び通知と共に複製がなされる限りにおいて許されるものとします。当該情報に変更を加え、あるいはその一部のみ、表示もしくは複製することは当該情報に係るTI製品もしくはサービスに対して提供された全ての保証を無効にし、かつ不公正で誤認を生じさせる行為であり、TIは、そのような使用については如何なる義務ないし責任も負うものではありません。

TIの製品もしくはサービスについてTIにより示された数値、特性、条件その他と異なる、あるいは、それを超えてなされた説明で当該TI製品もしくはサービスを再販売することは、当該TI製品もしくはサービスに対する全ての明示的保証、及び何らかの黙示的保証を無効にし、且つ不公正で誤認を生じさせる行為であり、TIは、そのような使用については如何なる義務ないし責任も負うものではありません。

なお、日本テキサス・インスツルメンツ株式会社半導体集積回路製品販売用標準契約約款もご覧下さい。
http://www.tij.co.jp/jsc/docs/stdterms.htm

弊社半導体製品の取り扱い・保管について

半導体製品は、取り扱い、保管 輸送環境、基板実装条件によっては、お客様での実装前後に破壊 /劣化、または故障を起こすことがあります。
弊社半導体製品のお取り扱い、ご使用にあたっては下記の点を遵守して下さい。

1．静電気
- 素手で半導体製品単体を触らないこと。どうしても触る必要がある場合は、リストストラップ等で人体からアースをとり、導電性手袋等をして取り扱うこと。
- 弊社出荷梱包単位（外装から取り出された内装及び個装）又は製品単品で取り扱いを行う場合は、接地された導電性のテーブル上で（導電性マットにアースをとったもの等）、アースをした作業者が行うこと。また、コンテナ等も、導電性のものを使うこと。
- マウンタやはんだ付け設備等、半導体の実装に関わる全ての装置類は、静電気の帯電を防止する措置を施すこと。
- 前記のリストストラップ・導電性手袋・テーブル表面及び実装装置類の接地等の静電気帯電防止措置は、常に管理されその機能が確認されていること。

2．温 湿度環境
- 温度： 0～40℃、相対湿度： 40～85%で保管・輸送及び取り扱いを行うこと。（但し、結露しないこと。）
- 直射日光があたる状態で保管・輸送しないこと。

3．防湿梱包
- 防湿梱包品は、開封後は個別推奨保管環境及び期間に従い基板実装すること。

4．機械的衝撃
- 梱包品（外装、内装、個装）及び製品単品を落下させたり、衝撃を与えないこと。

5．熱衝撃
- はんだ付け時は、最低限 260℃以上の高温状態に、10秒以上さらさないこと。（個別推奨条件がある時はそれに従うこと。）

6．汚染
- はんだ付け性を損なう、又はアルミ配線腐食の原因となるような汚染物質（硫黄、塩素等ハロゲン）のある環境で保管・輸送しないこと。
- はんだ付け後は十分にフラックスの洗浄を行うこと。（不純物含有率が一定以下に保証された無洗浄タイプのフラックスは除く。）

以上

2001.11

最初にお読みください

本書について

本書では、TMS320C55x™（C55x™）固定小数点デジタル・シグナル・プロセッサ（DSP）の中央演算処理ユニット（CPU）について説明します。また、そのアーキテクチャ、レジスタ、および演算についても説明します。

表記規則

本書では、次の表記規則を使用しています。

- 信号またはピンがローアクティブの場合は、上線が付いています。 たとえば、$\overline{\text{RESET}}$ 信号はローアクティブです。
- 多くの場合、16 進数は末尾に h が付いて表されています。たとえば、次の数字は 16 進数の 40（10 進数 64）です。

  40h

  同様に、2 進数は通常、末尾に b が付いて表されています。たとえば、次の数字は 2 進数の 0100 (10 進数の 4) です。

  0100b

- ビットと信号は、場合によっては、次のように表記しています。

表記	説明	例
レジスタ (n-m)	レジスタのビット n ～ m	AC0(15-0) は、レジスタ AC0 のビット 15 ～ 0 を表します。
バス [n:m]	バスの信号 n ～ m	A[21:1] は、バス A の信号 21 ～ 1 を表します。

❑ 次の用語は、データの部分を表します。

用語	説明	例
LSB	最下位ビット	AC0(15-0) の場合、ビット 0 が LSB です。
MSB	最上位ビット	AC0(15-0) の場合、ビット 15 が MSB です。
LSByte	最下位バイト	AC0(15-0) の場合、ビット 7 ～ 0 が LSByte です。
MSByte	最上位バイト	AC0(15-0) の場合、ビット 15 ～ 8 が MSByte です。
LSW	最下位ワード	AC0(31-0) の場合、ビット 15 ～ 0 が LSW です。
MSW	最上位ワード	AC0(31-0) の場合、ビット 31 ～ 16 が MSW です。

関連資料

C55x デバイスおよびそのサポート・ツールなどを解説した関連資料は、次のとおりです。関連資料は、www.ti.com から入手可能です。www.ti.com にアクセスして、検索ボックスに文献番号を入力してください。

TMS320C55x Technical Overview（文献番号 SPRU393）では、TMS320C55x DSP、および TMS320C5000™ DSP プラットフォームにおける固定小数点 DSP の最新版について説明しています。以前の製品と同様に、このプロセッサは、高性能で低消費電力での動作に最適です。この資料では、CPU のアーキテクチャ、低消費電力拡張機能、および組み込みエミュレーション機能について説明しています。

TMS320C55x DSP Peripherals Overview Reference Guide（文献番号 SPRU317）では、TMS320C55x DSP で使用可能なペリフェラル、インターフェイス、および関連するハードウェアについて説明しています。

TMS320C55x DSP Algebraic Instruction Set Reference Guide（文献番号 SPRU375）では、TMS320C55x DSP の各代数表記命令について説明しています。また、命令セットの要約、命令オペコードの一覧、およびニーモニック命令セットへの相互参照も記述しています。

TMS320C55x DSP Mnemonic Instruction Set Reference Guide（文献番号 SPRU374）では、TMS320C55x DSP の各ニーモニック命令について説明しています。また、命令セットの要約、命令オペコードの一覧、および代数表記命令セットへの相互参照も記述しています。

TMS320C55x Optimizing C/C++ Compiler User's Guide（文献番号 SPRU281）では、TMS320C55x の C/C++ コンパイラについて説明しています。この C/C++ コンパイラは、ISO 標準の C および C++ ソース・コードに対応し、TMS320C55x デバイス用のアセンブリ言語ソース・コードを作成します。

TMS320C55x Assembly Language Tools User's Guide（文献番号 SPRU280）では、TMS320C55x デバイス用のアセンブリ言語ツール（アセンブリ言語コードの開発に使用するアセンブラやリンカなどのツール）、アセンブラ・ディレクティブ、マクロ、共通オブジェクト・ファイル・フォーマット、およびシンボリック・デバッグのディレクティブについて説明しています。

TMS320C55x DSP Programmer's Guide（文献番号 SPRU376）では、TMS320C55x DSP の C とアセンブリのコードを最適化する方法、また DSP の特殊な機能と命令を使用するコードの書き方について説明しています。

商標

TMS320C5000、TMS320C54x、C54x、TMS320C55x、および C55x は、Texas Instruments の商標です。

商標

目次

図

図

表

第 1 章

CPU アーキテクチャ

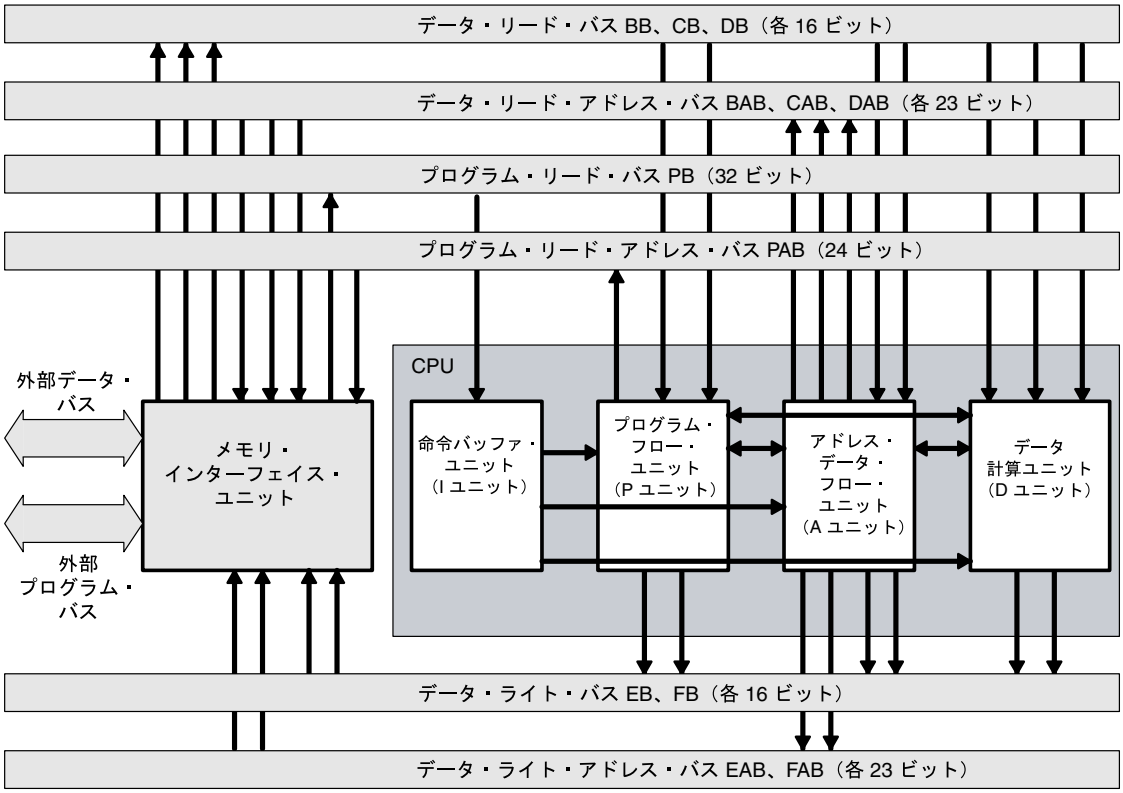
この章では、TMS320C55x™（C55x™）DSP の CPU アーキテクチャについて説明します。CPU の 4 つの機能ユニット、および命令とデータを伝送するバスの概念について詳しく説明します。また、命令パイプラインの並列フェーズとパイプライン保護メカニズム（リードとライトが意図しない順序で行われるのを防止するメカニズム）についても説明します。

項目 **ページ**

1.1 CPU アーキテクチャの概要

CPU の概念ブロック図を図 1-1 に示します。この図に示されているバスとユニットについて 1.1.1 ～ 1.1.6 の各項で説明します。

図 1-1. CPU の図



1.1.1 内部データおよびアドレス・バス

図 1-1 に示されているバスは、次のとおりです。

- **データ・リード・バス（BB、CB、DB）：**これらの 3 つのバスは、16 ビット・データをデータ空間または I/O 空間から CPU の機能ユニットに伝送します。

  BB は、内部メモリから D ユニットへのデータだけを伝送します（主に、デュアル積和演算（MAC）ユニットへの伝送）。BB は、外部メモリには接続されていません。特定の命令を使用すると、BB、CB、DB を使用して 3 つのオペランドを同時にリードすることができます。

> **注：**
>
> **BB と BAB は、両方とも外部メモリに接続されていません。BB と BAB を使用して、ある命令がオペランドをフェッチする場合、そのオペランドは、内部メモリに存在する必要があります。外部メモリ・アドレスを不適切に使用すると、バス・エラー割り込みが発生します。**

  CB と DB は、P ユニット、A ユニット、および D ユニットにデータを送ります。2 つのオペランドを同時にリードする命令は、CB と DB の両方を使用します。単一リード演算を実行する命令は、DB を使用します。

- **データ・リード・アドレス・バス（BAB、CAB、DAB）：**これらの 3 つのバスは、23 ビットのワード・データ・アドレスをメモリ・インターフェイス・ユニットに伝送し、その後、このユニットがメモリからデータをフェッチし、要求された値をデータ・リード・バスに転送します。すべてのデータ空間アドレスは、A ユニットで生成されます。

  BAB は、BB で内部メモリから CPU に伝送されるデータのアドレスを伝送します。

  CAB は、CB で CPU に伝送されるデータのアドレスを伝送します。

  DAB は、DB のみ、または CB と DB の両方で CPU に伝送されるデータのアドレスを伝送します。

- **プログラム・リード・バス（PB）：**PB は、プログラム・コードの 32 ビット（4 バイト）を I ユニットに一度に伝送します。命令はその I ユニットでデコードされます。

- **プログラム・リード・アドレス・バス（PAB）：**PAB は、PB が CPU に伝送するプログラム・コードの 24 ビットのバイト・プログラム・アドレスを伝送します。

- **データ・ライト・バス（EB、FB）：**これらの 2 つのバスは、16 ビット・データを CPU の機能ユニットからデータ空間または I/O 空間に伝送します。

  EB と FB は、P ユニット、A ユニット、および D ユニットからデータを受け取ります。2 つの 16 ビット値をメモリに同時にライトする命令は、EB と FB の両方を使用します。単一ライト演算を実行する命令は、EB を使用します。

- **データ・ライト・アドレス・バス（EAB、FAB）：**これらの 2 つのバスは、23 ビット・アドレスをメモリ・インターフェイス・ユニットに伝送し、その後、このユニットがデータ・ライト・バスでドライブされる値を受け取ります。すべてのデータ空間アドレスは、A ユニットで生成されます。

  EAB は、EB のみ、または EB と FB の両方でメモリに伝送されるデータのアドレスを伝送します。

  FAB は、FB でメモリに伝送されるデータのアドレスを伝送します。

1.1.2 メモリ・インターフェイス・ユニット

メモリ・インターフェイスは、CPU とプログラム / データ空間または I/O 空間との間のデータ転送をすべて仲介します。

1.1.3 命令バッファ・ユニット（I ユニット）

各 CPU サイクルの間、I ユニットはプログラム・コードの 4 バイトをその命令バッファ・キュー内に受け取り、そのキューで事前に受け取ったコードの 1 ～ 6 バイトをデコードします。その後、I ユニットは命令を実行するために、P ユニット、A ユニット、および D ユニットにデータを渡します。たとえば、レジスタのロード、シフト・カウントの供給、ビット数の特定などの命令でエンコードされた定数は、I ユニット内で分離され、適切なユニットに渡されます。

CPU が新しいロケーションに分岐すると、必ず命令バッファ・キューは空になります。

命令バッファ・キューは、単一リピート演算およびローカル・リピート演算の場合にロードされます（命令バッファ・キューをフルにする必要はありません）。

1.1.4 プログラム・フロー・ユニット（P ユニット）

P ユニットは、すべてのプログラム空間アドレスを生成し、それらのアドレスを PAB 上で送信します。また、P ユニットは、ハードウェア・ループ、分岐、条件付き実行などの演算を指示することにより、命令のシーケンスを制御します。

1.1.5 アドレス・データ・フロー・ユニット（A ユニット）

A ユニットは、データ空間アドレスの生成とそれらを BAB、CAB、DAB 上で送信するために必要なすべてのロジックとレジスタを含みます。また、A ユニットは、算術、論理、シフト、飽和の各演算を実行可能な 16 ビットの算術論理演算ユニット（ALU）を含みます。

1.1.6 データ計算ユニット（D ユニット）

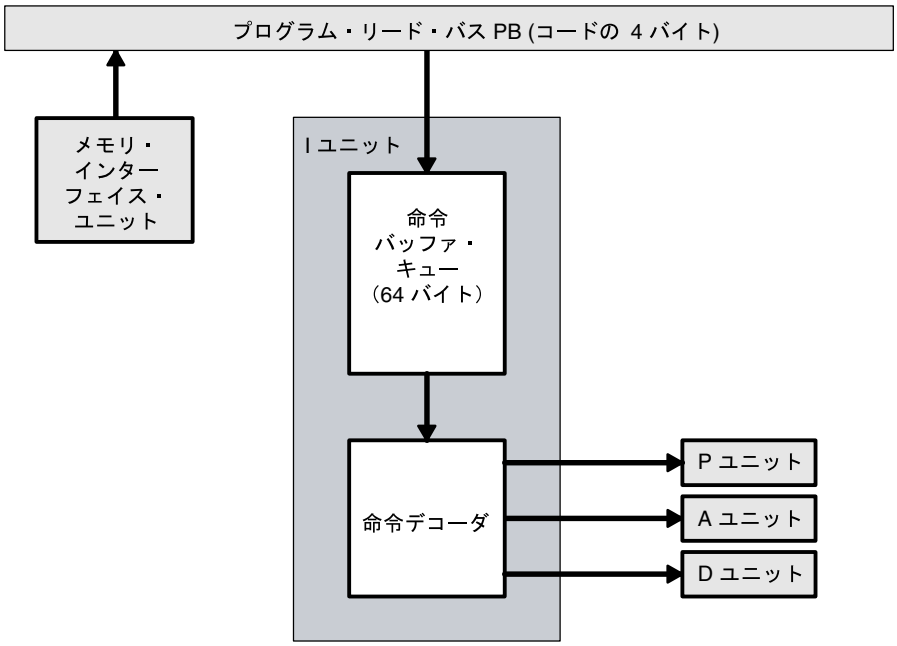
D ユニットは、CPU の主な計算ユニットを含みます。

- ❏ -32 〜 31 のシフト範囲を実現する 40 ビットのバレル・シフタ
- ❏ 算術、論理、丸め、飽和の各演算を実行できる 40 ビットの算術論理演算ユニット（ALU）
- ❏ 17 ビットの乗算と 40 ビットの加算または減算を単一サイクルで実行できる 1 組の積和演算（MAC）ユニット

1.2 命令バッファ・ユニット（I ユニット）

I ユニットは、プログラム・コードをその命令バッファ・キュー内に受け取り、命令をデコードします。その後、I ユニットは命令の実行のために、P ユニット、A ユニット、および D ユニットにデータを渡します。I ユニットの概念ブロック図を図 1-2 に示します。I ユニットの主要な構成部分について 1.2.1 と 1.2.2 の項で説明します。

図 1-2. 命令バッファ・ユニット（I ユニット）の図



1.2.1 命令バッファ・キュー

CPU は、プログラム・メモリから 32 ビットを一度にフェッチします。プログラム・リード・バス（PB）は、これらの 32 ビットをメモリから命令バッファ・キューに伝送します。このキューは、デコードされていない命令を 64 バイトまで保持できます。CPU が命令をデコードするための準備ができている場合、6 バイトがこのキューから命令デコーダに転送されます。

命令のパイプライン化を促進するのに加え、このキューは、次のことを実行します。

- キュー内に格納された 1 ブロックのコードの実行（ローカル・リピート命令）
- 次の命令の 1 つの条件のテスト時の投機的な命令フェッチ
  - 条件付き分岐
  - 条件付きコール
  - 条件付き復帰

1.2.2 命令デコーダ

命令パイプラインのデコード・フェーズでは、命令デコーダは命令バッファ・キューからプログラム・コードの 6 バイトを受け取り、これらのバイトをデコードします。命令デコーダは、次のことを実行します。

- 8 ビット、16 ビット、24 ビット、32 ビット、40 ビット、48 ビットの命令をデコードできるように命令境界を識別します。
- CPU が 2 つの命令を並列に実行するように指示されたかどうかを判別します。
- デコードされた実行コマンドと即値をプログラム・フロー・ユニット（P ユニット）、アドレス・データ・フロー・ユニット（A ユニット）、およびデータ計算ユニット（D ユニット）に送ります。

特定の命令は、専用のデータ・パスを経由して即値を直接メモリまたは I/O 空間に書き込むことを可能にします。

デコーダは、通常、一度に 6 バイトまでしかデコードしませんが、単一命令に対して 7 バイトをデコードする場合があります。表 1-1 に記述されている命令は、4 バイトのオペコードを持ち、Smem に k23 絶対アドレッシング・モードが使用される場合、3 バイト拡張されます。k23 絶対アドレッシング・モードの詳細は、6.2.2 項を参照してください。

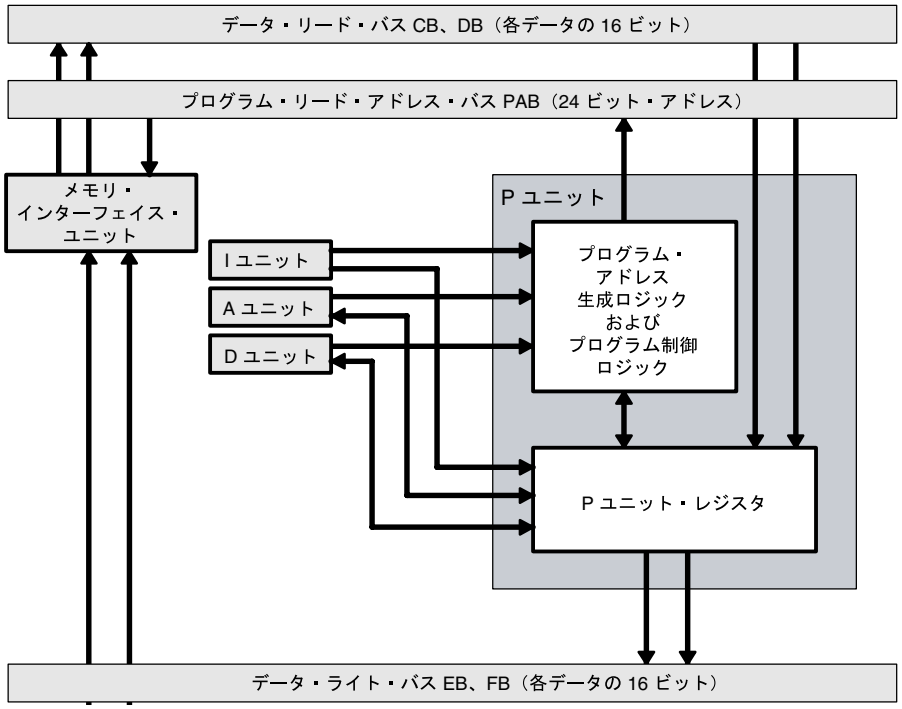
表 1-1. k23 絶対アドレッシング・モード使用時に 7 バイトに拡張される 4 バイト命令

命令構文	命令タイプ
CMP Smem == K16, TCx	メモリと即値との比較
BAND Smem, k16, TCx	ビット単位のメモリと即値との AND 演算
AND k16, Smem	ビット単位の AND 演算
OR k16, Smem	ビット単位の OR 演算
XOR k16, Smem	ビット単位の XOR 演算
ADD k16, Smem	加算
MPYMK[R] [T3 =]Smem, K8, ACx	乗算
MACMK[R] [T3 =]Smem, K8, [ACx,] ACy	積和演算
ADD [uns(]Smem[)] << #SHIFTW, [ACx,] ACy	加算
SUB [uns(]Smem[)] << #SHIFTW, [ACx,] ACy	減算
MOV [uns(]Smem[)] << #SHIFTW, ACx	メモリからアキュムレータへのロード
MOV [rnd(]HI(ACx << #SHIFTW)[)], Smem	アキュムレータ内容のメモリへのストア
MOV [uns(] [rnd(]HI[(saturate] (ACx << #SHIFTW)[)))], Smem	アキュムレータ内容のメモリへのストア
MOV k16, Smem	メモリへの即値のロード

1.3 プログラム・フロー・ユニット（P ユニット）

P ユニットは、すべてのプログラム空間アドレスを生成します。また、P ユニットは命令のシーケンスを制御します。P ユニットの概念ブロック図を図 1-3 に示します。P ユニットの主要な構成部分について 1.3.1 と 1.3.2 の項で説明します。

図 1-3. プログラム・フロー・ユニット（P ユニット）の図



1.3.1 プログラム・アドレス生成およびプログラム制御ロジック

プログラム・アドレス生成ロジックは、プログラム・メモリからのフェッチのために 24 ビットのアドレスを生成する責任があります。通常、このロジックは連続したアドレスを生成します。ただし、連続しないアドレスからのリードが必要な命令の場合は、プログラム・アドレス生成ロジックが I ユニットからの即値データと D ユニットからのレジスタ値を受け取ることができます。アドレスが生成されると、プログラム・リード・アドレス・バス（PAB）によりメモリに伝送されます。

プログラム制御ロジックは、I ユニットからの即値と、A ユニットまたは D ユニットからのテスト結果を受け入れ、次のアクションを実行します。

- ❑ 条件付き命令に対して条件が真かどうかをテストし、その結果をプログラム・アドレス生成ロジックに伝達します。
- ❑ 割り込みが要求され、正常にイネーブルになると、割り込み処理を開始します。
- ❑ 単一リピート命令を前に置いた単一命令のリピート、またはブロック・リピート命令を前に置いた命令ブロックのリピートを制御します。ブロック・リピート演算を別のブロック・リピート演算にネストし、単一リピート演算をリピートされるブロックの一方または両方に組み込むことにより、3 段階のループを実現することができます。リピート演算はすべて割り込み可能です。
- ❑ 並列に実行される命令を管理します。C55x DSP での並列処理は、プログラム制御命令をデータ処理命令と同時に実行することができます。

1.3.2 P ユニットのレジスタ

P ユニットは、以下に示されるレジスタを含み、使用します。プログラム・フロー・レジスタへのアクセスは制限されます。PC からのリードまたは PC へのライトを行うことはできません。RETA と CFCT へのアクセスは、次の構文でのみ行うことができます。

MOV dbl(Lmem), RETA

MOV RETA, dbl(Lmem)

他のレジスタはすべて（I ユニットから）即値がロードされ、データ・メモリ、I/O 空間、A ユニットのレジスタ、および D ユニットのレジスタと双方向に通信できます。

プログラム・フロー・レジスタ

PC	プログラム・カウンタ
RETA	リターン・アドレス・レジスタ
CFCT	制御フロー・コンテキスト・レジスタ

ブロック・リピート・レジスタ

BRC0、BRC1	ブロック・リピート・カウンタ 0 および 1
BRS1	BRC1 格納レジスタ
RSA0、RSA1	ブロック・リピート開始アドレス・レジスタ 0 および 1
REA0、REA1	ブロック・リピート終了アドレス・レジスタ 0 および 1

単一リピート・レジスタ

RPTC	単一リピート・カウンタ
CSR	計算済み単一リピート・レジスタ

割り込みレジスタ

IFR0、IFR1　　割り込みフラグ・レジスタ 0 および 1

IER0、IER1　　割り込みイネーブル・レジスタ 0 および 1

DBIER0、DBIER1　デバッグ割り込みイネーブル・レジスタ 0 および 1

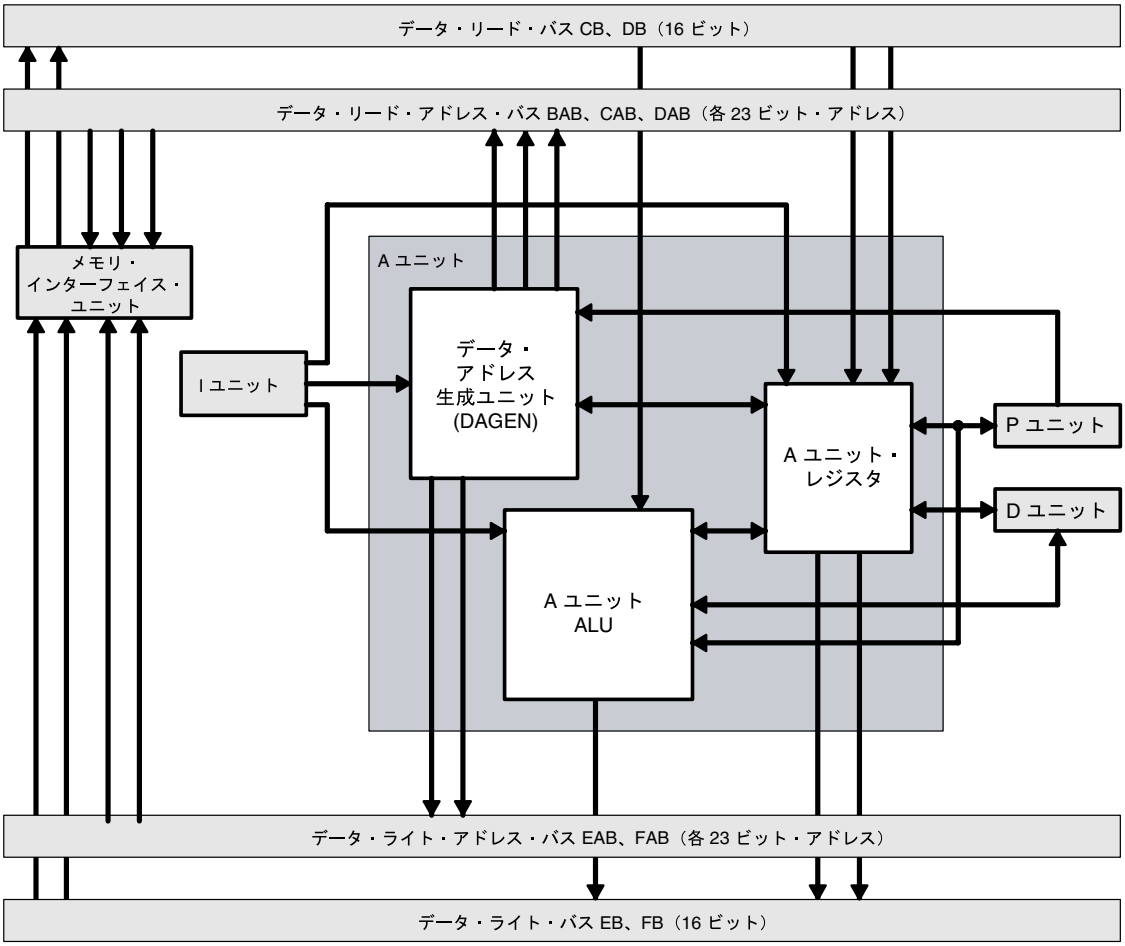
ステータス・レジスタ

ST0_55-ST3_55　　ステータス・レジスタ 0、1、2、3

1.4 アドレス・データ・フロー・ユニット（A ユニット）

A ユニットは、データ空間アドレスと I/O 空間アドレスの生成に必要なすべてのロジックとレジスタを含みます。また、A ユニットは、算術、論理、シフト、飽和の各演算を実行可能な算術論理演算ユニット（ALU）も含みます。A ユニットの概念ブロック図を図 1-4 に示します。1.4.1 ～ 1.4.3 の各項では、A ユニットの主要な構成部分について説明します。

図 1-4. アドレス・データ・フロー・ユニット（A ユニット）の図



1.4.1 データ・アドレス生成ユニット（DAGEN）

DAGEN は、データ空間および I/O 空間とのリードまたはライトのすべてのアドレスを生成します。これにより、I ユニットからの即値と A ユニットからのレジスタ値を受け入れることができます。P ユニットは、間接アドレッシング・モードを使用する命令に対して、リニア・アドレッシングまたはサーキュラー・アドレッシングを使用するかを DAGEN に指示します。

1.4.2 A ユニットの算術論理演算ユニット（A ユニット ALU）

A ユニットは、I ユニットから即値を受け入れる 16 ビットの ALU を含み、メモリ、I/O 空間、A ユニット・レジスタ、D ユニット・レジスタ、P ユニット・レジスタと双方向に通信します。A ユニット ALU は、次のアクションを実行します。

- ❏ 加算、減算、比較、ブーリアン論理演算、符号付きシフト、論理シフト、および絶対値の計算の実行
- ❏ A ユニット・レジスタ・ビットとメモリ・ビットのテスト、セット、クリア、および補数演算
- ❏ レジスタ値の変更および移動
- ❏ レジスタ値のローテーション
- ❏ 特定の結果をシフタから A ユニット・レジスタに移動

1.4.3 A ユニット・レジスタ

A ユニットは、この段落の後に示されるレジスタを含み、使用します。これらのレジスタはすべて、I ユニットから即値データを受け入れ、P ユニット・レジスタ、D ユニット・レジスタ、およびデータ・メモリとの間のデータの受け渡しを行うことできます。A ユニットでは、レジスタは DAGEN と A ユニット ALU との双方向接続を備えています。

データ・ページ・レジスタ

DPH、DP	データ・ページ・レジスタ
PDP	ペリフェラル・データ・ページ・レジスタ

ポインタ

CDPH、CDP	係数データ・ポインタ・レジスタ
SPH、SP、SSP	スタック・ポインタ・レジスタ
XAR0 ～ XAR7	補助レジスタ

サーキュラー・バッファ・レジスタ

BK03、BK47、BKC	サーキュラー・バッファ・サイズ・レジスタ
BSA01、BSA23、BSA45、BSA67、BSAC	サーキュラー・バッファ開始アドレス・レジスタ

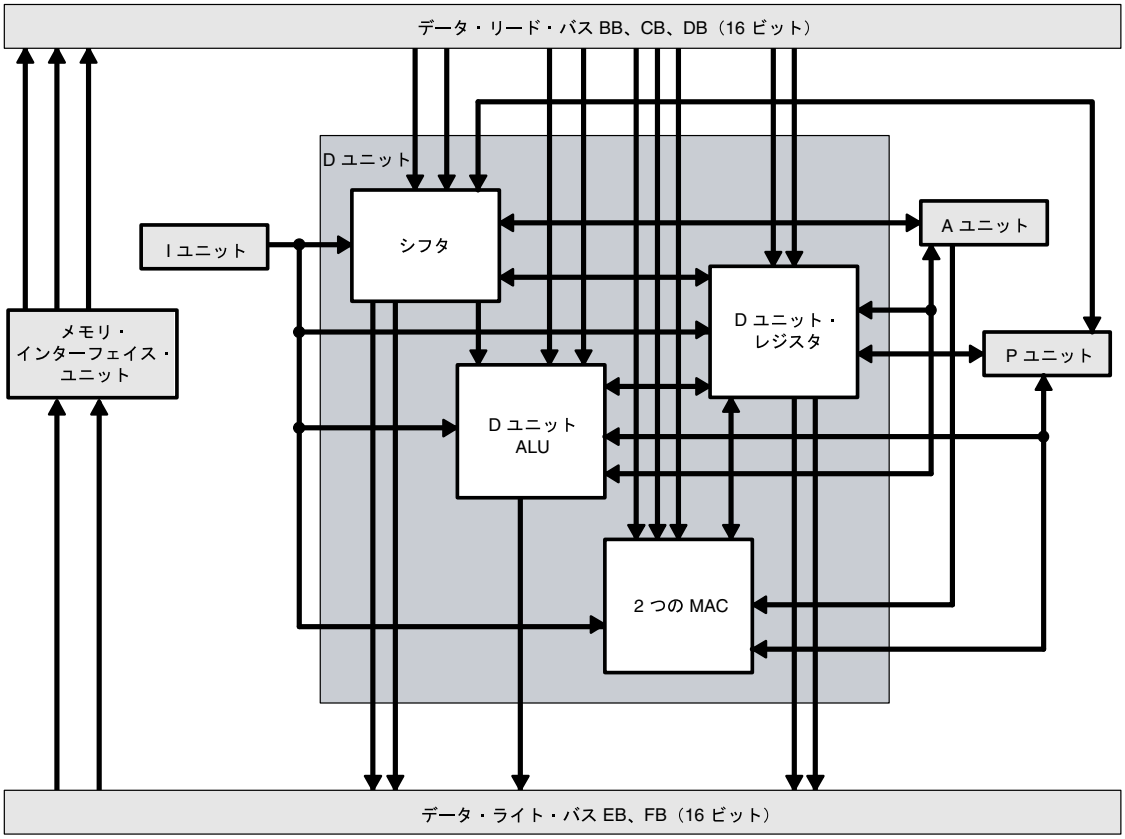
一時レジスタ

T0 ～ T3	一時レジスタ 0、1、2、3

1.5 データ計算ユニット（D ユニット）

D ユニットは、CPU の主な計算ユニットを含みます。D ユニットの概念ブロック図を図 1-5 に示します。D ユニットの主要な構成部分について 1.5.1 ～ 1.5.4 の各項で説明します。

図 1-5. データ計算ユニット（D ユニット）の図



1.5.1 シフタ

D ユニット・シフタは、I ユニットから即値を受け入れ、メモリ、I/O 空間、A ユニット・レジスタ、D ユニット・レジスタ、P ユニット・レジスタと双方向に通信します。さらに、シフトされた値を、D ユニット ALU（以降の計算用の入力として）および A ユニット ALU（A ユニット・レジスタ内に格納される結果として）に供給します。シフタは、次のアクションを実行します。

- 40 ビットのアキュムレータ値を最大 31 ビット左に、または最大 32 ビット右にシフトします。シフト・カウントは、一時レジスタ（T0 ～ T3）の 1 つから読み取られるか、命令内で定数として供給されます。
- 16 ビットのレジスタ、メモリ、または I/O 空間の値を最大 31 ビット左に、または最大 32 ビット右にシフトします。シフト・カウントは、一時レジスタ（T0 ～ T3）の 1 つから読み取られるか、命令内で定数として供給されます。
- 16 ビットの即値を最大 15 ビット左にシフトします。シフト・カウントは、命令内で定数として供給されます。
- アキュムレータ値を正規化します。
- ビット・フィールドを抽出および拡張し、ビットのカウントを実行します。
- レジスタ値をローテーションします。
- アキュムレータ値がデータ・メモリに格納される前にアキュムレータ値の丸めと飽和の一方または両方を実行します。
- シフトを含む一部の命令に対して加算および減算を実行します。

C54x™ 互換モード（C54CM = 1）の場合、オーバーフローの検出は計算の最後の演算に対してのみ実行されます。C55x ネイティブ・モード（C54CM = 0）の場合、オーバーフローの検出は演算（シフト、丸め、加算、減算）ごとに実行されます。

1.5.2 D ユニットの算術論理演算ユニット（D ユニット ALU）

CPU の D ユニットは、I ユニットから即値を受け入れる 40 ビットの ALU を含み、メモリ、I/O 空間、A ユニット・レジスタ、D ユニット・レジスタ、P ユニット・レジスタと双方向に通信します。さらに、シフタからの結果を受け取ります。D ユニット ALU は、次のアクションを実行します。

- ❏ 加算、減算、比較、丸め、飽和、ブーリアン論理演算、および絶対値の計算を実行します。
- ❏ デュアル 16 ビットの算術命令が実行される際に、2 つの算術演算を同時に実行します。
- ❏ D ユニット・レジスタ・ビットのテスト、セット、クリア、および補数演算
- ❏ レジスタ値の移動

1.5.3 2 つの積和演算ユニット（MAC）

2 つの MAC は、乗算と加算 / 減算をサポートします。単一サイクルでは、各 MAC は 17 ビット × 17 ビットの乗算（小数または整数）、および 32 ビットまたは 40 ビットの飽和オプションが付いた 40 ビットの加算または減算を実行することができます。アキュムレータ（D ユニット・レジスタ）は、MAC の結果をすべて受け取ります。

MAC は、I ユニットから即値を受け入れ、メモリ、I/O 空間、A ユニット・レジスタからデータ値を受け入れ、D ユニット・レジスタおよび P ユニット・レジスタと双方向に通信します。ステータス・レジスタ・ビット（P ユニット内）は、MAC 演算から影響を受けます。

オーバーフローの検出は、計算の最後の演算に対してのみ実行されます。

1.5.4 D ユニット・レジスタ

D ユニットは、この段落の後に示されるレジスタを含み、使用します。これらのレジスタはすべて、I ユニットから即値データを受け入れることができ、P ユニット・レジスタ、A ユニット・レジスタ、およびデータ・メモリとの間でデータの受け渡しを行うことできます。D ユニットでは、レジスタはシフタ、D ユニット ALU、および MAC との双方向接続を備えています。

アキュムレータ

AC0 ～ AC3	アキュムレータ 0、1、2、3

トランジッション・レジスタ

TRN0、TRN1	トランジッション・レジスタ 0 および 1

1.6 アドレス・バスおよびデータ・バス

CPU は、1 つの 32 ビット・プログラム・バス（PB）、5 つの 16 ビット・データ・バス（BB、CB、DB、EB、FB）、1 つの 24 ビット・アドレス・バス（PAB）、5 つの 23 ビット・アドレス・バス（BAB、CAB、DAB、EAB、FAB）により構成されています。この並列バス構成は、CPU のクロック・サイクルごとに、最大 1 つの 32 ビット・プログラム・リード、3 つの 16 ビット・データ・リード、および 2 つの 16 ビット・データ・ライトを可能にします。12 のバスの機能を表 1-2 に示します。アクセス・タイプに応じてどのバスが使用されるかを表 1-3 に示します。

表 1-2. アドレス・バスとデータ・バスの機能

バス	幅	機能
PAB	24 ビット	プログラム・リード・アドレス・バス（PAB）は、プログラム空間からのリード用に 24 ビットのバイト・アドレスを伝送します。
PB	32 ビット	プログラム・リード・バス（PB）は、プログラム・コードの 4 バイト（32 ビット）をプログラム・メモリから CPU に伝送します。
CAB、DAB	23 ビットずつ	これらのデータ・リード・アドレス・バスは、それぞれ 23 ビットのワード・アドレスを伝送します。DAB は、データ空間または I/O 空間からリードするアドレスを伝送します。CAB は、デュアル・データのリード時に 2 番目のアドレスを伝送します（表 1-3 を参照）。
CB、DB	16 ビットずつ	これらのデータ・リード・バスは、それぞれ 16 ビットのデータ値を CPU に伝送します。DB は、データ空間または I/O 空間からの値を伝送します。CB は、long 型のデータのリード、およびデュアル・データのリード時に 2 番目の値を伝送します（表 1-3 を参照）。
BAB	23 ビット	このデータ・リード・アドレス・バスは、係数リードに対する 23 ビットのワード・アドレスを伝送します。係数の間接アドレッシング・モードを使用する命令の多くは、BAB を使用して係数データ値を参照し、BB を使用してそのデータ値を伝送します。
BB	16 ビット	このデータ・リード・バスは、16 ビットの係数データ値を内部メモリから CPU に伝送します。BB は、外部メモリには接続されていません。BB が伝送するデータは、BAB を使用してアドレス指定されます。特定の命令は、係数の間接アドレッシング・モードを使用して、1 サイクルで、3 つの 16 ビット・オペランドを CPU に供給するために、BB、CB、および DB を使用します。BB を経由してフェッチされたオペランドは、CB と DB を経由してアクセスされたメモリ・バンク以外のバンク内になければなりません。
EAB、FAB	23 ビットずつ	これらのデータ・ライト・アドレス・バスは、それぞれ 23 ビットのワード・アドレスを伝送します。EAB は、データ空間または I/O 空間へライトするアドレスを伝送します。FAB は、デュアル・データのライト時に 2 番目のアドレスを伝送します（表 1-3 を参照）。
EB、FB	16 ビットずつ	これらのデータ・ライト・バスは、それぞれ CPU から 16 ビットのデータ値を伝送します。EB は、データ空間または I/O 空間に値を伝送します。FB は、long 型のデータのライト、およびデュアル・データのライト時に 2 番目の値を伝送します（表 1-3 を参照）。

注:

デュアル・データを同一アドレスにライトした結果は未定です。

表 1-3. アクセス・タイプに応じたバスの使用

アクセス・タイプ	アドレス・バス	データ・バス	説明
命令フェッチ	PAB	PB	プログラム空間からの 32 ビットのリード
単一データ・リード	DAB	DB	データ・メモリからの 16 ビットのリード
単一 MMR リード	DAB	DB	メモリ・マップド・レジスタ（MMR）からの 16 ビットのリード
単一 I/O リード	DAB	DB	I/O 空間からの 16 ビットのリード
単一データ・ライト	EAB	EB	データ・メモリへの 16 ビットのライト
単一 MMR ライト	EAB	EB	メモリ・マップド・レジスタ（MMR）への 16 ビットのライト
単一 I/O ライト	EAB	EB	I/O 空間への 16 ビットのライト
long 型のデータ・リード	DAB	CB、DB	データ・メモリからの 32 ビットのリード
long 型の MMR リード	DAB	CB、DB	1 つの 32 ビット MMR または 2 つの隣接する 16 ビット MMR からの 32 ビットのリード
long 型のデータ・ライト	EAB	EB、FB	データ・メモリへの 32 ビットのライト
long 型の MMR ライト	EAB	EB、FB	1 つの 32 ビット MMR または 2 つの隣接する 16 ビット MMR への 32 ビットのライト
デュアル・データ・リード	CAB、DAB	CB、DB	データ空間からの 2 つの 16 ビット同時リード。 ❑ 最初のオペランドのリードは、DAB と DB を使用します。2 番目のオペランドのリードは、CAB と CB を使用します。 ❑ 最初のオペランドのリードは、DAB と DB を使用します。このリードは、データ・メモリ、MMR、または I/O 空間から行うことができます。 ❑ 2 番目のオペランドのリードは、CAB と CB を使用します。このリードは、データ・メモリから行う必要があります。

表 1-3. アクセス・タイプに応じたバスの使用（続き）

アクセス・タイプ	アドレス・バス	データ・バス	説明
デュアル・データ・ライト	EAB、FAB	EB、FB	データ空間への 2 つの 16 ビット同時ライト。最初のオペランドのライトは、FAB と FB を使用します。2 番目のオペランドのライトは、EAB と EB を使用します。 ❏ 最初のオペランドのライトは、FAB と FB を使用します。このライトは、データ・メモリに対して行う必要があります。 ❏ 2 番目のオペランドのライトは、EAB と EB を使用します。このライトは、データ・メモリ、MMR、または I/O 空間に対して行うことができます。
単一データ・リード || 単一データ・ライト	DAB、EAB	DB、EB	次の 2 つの演算は、並列に実行されます。 ❏ 単一データ・リード：データ・メモリからの 16 ビット・リード（DAB と DB を使用） ❏ 単一データ・ライト：データ・メモリへの 16 ビット・ライト（EAB と EB を使用）
long 型のデータ・リード || long 型のデータ・ライト	DAB、EAB	CB、DB、EB、FB	次の 2 つの演算は、並列に実行されます。 ❏ long 型のデータ・リード：データ・メモリからの 32 ビット・リード（DAB、CB、DB を使用） ❏ long 型のデータ・ライト：データ・メモリへの 32 ビット・ライト（EAB、EB、FB を使用）
単一データ・リード || 係数データ・リード	DAB、BAB	DB、BB	次の 2 つの演算は、並列に実行されます。 ❏ 単一データ・リード：データ空間からの 16 ビット・リード（DAB と DB を使用） ❏ 係数データ・リード：係数の間接アドレッシング・モードを使用した内部メモリからの 16 ビット・リード（BAB と BB を使用）

表 1-3. アクセス・タイプに応じたバスの使用（続き）

アクセス・タイプ	アドレス・バス	データ・バス	説明
デュアル・データ・リード || 係数データ・リード	CAB、DAB、BAB	CB、DB、BB	次の 2 つの演算は、並列に実行されます。 ❑ デュアル・データ・リード：データ空間からの 2 つの 16 ビット同時リード。最初のオペランドのリードは、DAB と DB を使用します。2 番目のオペランドのリードは、CAB と CB を使用します。 ❑ 係数データ・リード：係数の間接アドレッシング・モードを使用した内部メモリからの 16 ビット・リード（BAB と BB を使用）

1.7 命令パイプライン

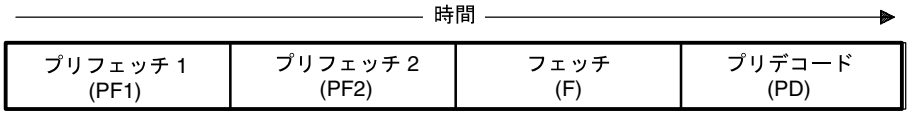
C55x の CPU は、命令パイプラインを使用します。パイプラインの概要を 1.7.1 項で示します。パイプラインで発生する可能性があるコンフリクトを防ぐ方法を 1.7.2 項で説明します。『TMS320C55x DSP Programmer's Guide』（文献番号 SPRU376）では、パイプラインの動作について詳しく説明しています。

1.7.1 パイプライン・フェーズ

C55x の命令パイプラインは、2 つの分離したセグメントをもっている保護パイプラインです。

- ❑ 『フェッチ・パイプライン』と呼ばれる 1 つ目のセグメントは、メモリから 32 ビットの命令パケットをフェッチし、それらを命令バッファ・キュー（IBQ）内に配置します。その後、2 つ目のパイプライン・セグメントに 48 ビットの命令パケットを供給します。このフェッチ・パイプラインを図 1-6 に示します。
- ❑ 『実行パイプライン』と呼ばれる 2 つ目のセグメントは、命令をデコードし、データのアクセスと計算を実行します。この実行パイプラインを図 1-7 に示します。実行パイプラインの主要なフェーズにおけるアクティビティを説明する例を表 1-4 に示します。

図 1-6. 1 つ目のパイプライン・セグメント（フェッチ・パイプライン）



パイプライン・フェーズ	説明
PF1	プログラム・アドレスをメモリに渡します。
PF2	メモリの応答を待機します。
F	命令パケットをメモリからフェッチし、それを IBQ 内に配置します。
PD	IBQ 内の命令を事前にデコードします（命令の開始と終了の位置を識別、並列命令を識別）。

図 1-7. 2 つ目のパイプライン・セグメント（実行パイプライン）

注：メモリ・ライト演算の場合のみ

パイプライン・フェーズ	説明
D	❑ 命令バッファ・キューから 6 バイトリードします。 ❑ 1 組の命令または単一命令をデコードします。 ❑ 命令を適切な CPU 機能ユニットに送ります。 ❑ データ・アドレス生成に関連する STx_55 ビットをリードします。 ST1_55(CPL) ST2_55(ARnLC) ST2_55(ARMS) ST2_55(CDPLC)
AD	❑ データ・アドレス生成に関連するレジスタのリード / 変更。次に例を示します。 - *ARx+(T0) の ARx と T0 - AR2LC = 1 の場合の BK03 - プッシュおよびポップ時の SP - 32 ビット・スタック・モードの場合の SP と同じ SSP ❑ A ユニット ALU を使用する演算を実行します。次に例を示します。 - AADD 命令を使用した算術演算 - SWAP 命令を使用した A ユニット・レジスタのスワップ - A ユニット・レジスタ（BKxx、BSAxx、BRCx、CSR など）への定数のライト ❑ ARx が 0 でないときに分岐する条件付き分岐命令に対する ARx をデクリメントします。 ❑ （例外）XCC 命令の条件を評価します（代数演算命令の構文の（AD ユニット）属性を実行）。

パイプライン・フェーズ	説明
AC1	メモリ・リード演算の場合、アドレスを適切な CPU アドレス・バスに送ります。
AC2	リード要求に応答するためにメモリに 1 サイクル与えます。
R	❑ メモリと MMR アドレス指定のレジスタからのデータをリードします。 ❑ A ユニット・レジスタをリードするのは、特定の D ユニット命令を実行する場合です。この特定の D ユニット命令は、R フェーズで A ユニット・レジスタを「プリフェッチ」します。X フェーズで、A ユニット・レジスタをリードするわけではありません。 ❑ 条件付き命令の条件を評価します。条件の評価の大部分は、R フェーズで実行されます。この表では、例外の場合は「(例外)」と表記しています。
X	❑ MMR アドレス指定以外のレジスタをリードまたは変更します。 ❑ 個々のレジスタ・ビットをリードまたは変更します。 ❑ 条件をセットします。 ❑ (例外) XCCPART 命令の条件を評価します (代数演算命令の構文の (D ユニット) 属性の実行)。ただし、その命令がメモリへのライトを条件としている場合を除きます (条件が R フェーズで評価される場合)。 ❑ (例外) RPTCC 命令の条件を評価します。
W	❑ MMR アドレス指定のレジスタまたは I/O 空間 (ペリフェラルのレジスタ) にデータをライトします。 ❑ メモリにデータをライトします。CPU の観点から、ライト演算はこのパイプライン・フェーズで終了します。
W+	❑ メモリにデータをライトします。メモリの観点から、ライト演算はこのパイプライン・フェーズで終了します。

表 1-4. 実行パイプライン・アクティビティの例

構文例	パイプラインの説明
AMOV #k23, XARx	XARx は、AD フェーズで定数を使用して初期化されます。
MOV #k, ARx	ARx は、MMR アドレス指定ではありません。ARx は、X フェーズで定数を使用して初期化されます。
MOV #k, mmap(ARx)	ARx は、MMR アドレス指定です。ARx は、W フェーズで定数を使用して初期化されます。
AADD #k, ARx	この特殊な命令を使用して、ARx は AD フェーズで定数を使用して初期化されます。
MOV #k, *ARx+	メモリ・ライトが W+ フェーズで発生します。
MOV *ARx+, AC0	ARx は、AD フェーズでリードおよび更新されます。AC0 は、X フェーズでロードされます。
ADD #k, ARx	ARx は、X フェーズの始めにリードされ、X フェーズの終わりに変更されます。
ADD ACy, ACx	ACx と ACy のリード / ライト・アクティビティは、X フェーズで発生します。
MOV mmap(ARx), ACx	ARx は、MMR アドレス指定であり、R フェーズでリードされます。ACx は、X フェーズで変更されます。
MOV ARx, ACx	ARx は、MMR アドレス指定ではなく、X フェーズでリードされます。ACx は、X フェーズで変更されます。
BSET CPL	CPL ビットは、X フェーズでセットされます。
PUSH、POP、RET または AADD #K8, SP	SP は、AD フェーズでリードおよび変更されます。SSP は、32 ビットのスタック・モードが選択されている場合も影響を受けます。
XCCPART overflow (ACx) || MOV *AR1+, AC1	条件は、X フェーズで評価されます。 注：AR1 は、条件が真であるかどうかに関わらずインクリメントされます。
XCCPART overflow (ACx) || MOV AC1, *AR1+	条件は R フェーズで評価されます。これは、メモリへのライトを条件としているためです。 注：AR1 は、条件が真であるかどうかに関わらずインクリメントされます。
XCC overflow (ACx) || MOV *AR1+, AC1	条件は、AD フェーズで評価されます。 注：AR1 は、条件が真である場合のみインクリメントされます。

1.7.2 パイプライン保護

パイプライン内では複数の命令が同時に実行され、さまざまな命令が、別々の実行フェーズでメモリ、I/O 空間、レジスタの値に対する変更を実行します。保護されていないパイプラインでは、この結果、パイプライン・コンフリクト、つまり、同じロケーションでのリードとライトが予期しない順序で行われる可能性があります。ただし、C55x のパイプラインには、このパイプライン・コンフリクトを自動的に防止するメカニズムがあります。このパイプライン保護メカニズムは、コンフリクトを引き起こす命令間にインアクティブ・サイクルを追加します。

大部分のパイプライン保護サイクルは、次の 2 つのルールに基づいて挿入されます。

- ある命令があるロケーションにライトされなければならないのに、前の命令がそのロケーションからまだリードしていない場合に、最初にリードが行われるように余分にサイクルが挿入されます。
- ある命令があるロケーションからリードされなければならないのに、前の命令がそのロケーションにまだライトされていない場合に、最初にライトが行われるように余分にサイクルが挿入されます。

注 :

パイプライン保護メカニズムは、並列に実行されている 2 つの命令間でのパイプライン・コンフリクトを防止できません。

『TMS320C55x DSP Programmer's Guide』（文献番号 SPRU376）では、パイプライン保護に対して挿入されるサイクル数を最小限にする方法を説明しています。

第 2 章

CPU レジスタ

この章では、C55x DSP CPU の主要なレジスタについて説明します。2.1 節「レジスタの要約（アルファベット順）」では、これらのレジスタをアルファベット順に示します。2.2 節「メモリ・マップド・レジスタ」では、メモリ・マップド・レジスタのアドレスを示します。その他の節では、CPU とアイドル状態のレジスタについてさらに詳しく説明します。

項目 **ページ**

2.1 レジスタの要約（アルファベット順）

表 2-1 では、レジスタをアルファベット順に示します。各レジスタに関する詳細は、表の一番右の列に示されているページを参照してください。

表 2-1. レジスタの要約（アルファベット順）

レジスタ名	説明	サイズ	参照先
AC0 ～ AC3	アキュムレータ 0 ～ 3	40 ビットずつ	P. 2-9
AR0 ～ AR7	補助レジスタ 0 ～ 7	16 ビットずつ	P. 2-12
BK03、BK47、BKC	サーキュラ・バッファ・サイズ・レジスタ	16 ビットずつ	P. 2-16
BRC0、BRC1	ブロック・リピート・カウンタ 0 および 1	16 ビットずつ	P. 2-34
BRS1	BRC1 格納レジスタ	16 ビット	P. 2-34
BSA01、BSA23、BSA45、BSA67、BSAC	サーキュラ・バッファ・スタート・アドレス・レジスタ	16 ビットずつ	P. 2-15
CDP	係数データ・ポインタ（XCDP の下位部分）	16 ビット	P. 2-14
CDPH	XCDP の上位部分	7 ビット	P. 2-14
CFCT	制御フロー・コンテキスト・レジスタ	8 ビット	P. 2-21
CSR	計算シングル・リピート・レジスタ	16 ビット	P. 2-34
DBIER0、DBIER1	デバッグ割り込みイネーブル・レジスタ 0 および 1	16 ビットずつ	P. 2-30
DP	データ・ページ・レジスタ（XDP の下位部分）	16 ビット	P. 2-17
DPH	XDP の上位部分	7 ビット	P. 2-17
IER0、IER1	割り込みイネーブル・レジスタ 0 および 1	16 ビットずつ	P. 2-27
IFR0、IFR1	割り込みフラグ・レジスタ 0 および 1	16 ビットずつ	P. 2-24
IVPD、IVPH	割り込みベクタ・ポインタ	16 ビットずつ	P. 2-23
PC	プログラム・カウンタ	24 ビット	P. 2-21
PDP	ペリフェラル・データ・ページ・レジスタ	9 ビット	P. 2-18
REA0、REA1	ブロック・リピート・エンド・アドレス・レジスタ 0 および 1	24 ビットずつ	P. 2-34
RETA	リターン・アドレス・レジスタ	24 ビット	P. 2-21
RPTC	シングル・リピート・カウンタ	16 ビット	P. 2-34
RSA0、RSA1	ブロック・リピート・スタート・アドレス・レジスタ 0 および 1	24 ビットずつ	P. 2-34

表 2-1. レジスタの要約（アルファベット順）（続き）

レジスタ名	説明	サイズ	参照先
SP	データ・スタック・ポインタ（XSP の下位部分）	16 ビット	P. 2-18
SPH	XSP と XSSP の上位部分	7 ビット	P. 2-18
SSP	システム・スタック・ポインタ（XSSP の下位部分）	16 ビット	P. 2-18
ST0_55 ～ ST3_55	ステータス・レジスタ 0 ～ 3	16 ビットずつ	P. 2-37
T0 ～ T3	一時レジスタ	16 ビットずつ	P. 2-11
TRN0、TRN1	トランジッション・レジスタ 0 および 1	16 ビットずつ	P. 2-10
XAR0 ～ XAR7	拡張補助レジスタ 0 ～ 7	23 ビットずつ	P. 2-12
XCDP	拡張係数データ・ポインタ	23 ビット	P. 2-14
XDP	拡張データ・ページ・レジスタ	23 ビット	P. 2-17
XSP	拡張データ・スタック・ポインタ	23 ビット	P. 2-18
XSSP	拡張システム・スタック・ポインタ	23 ビット	P. 2-18

2.2 メモリ・マップド・レジスタ

表 2-2 では、メモリ・マップド・レジスタを示します。これらのレジスタは、DSP のデータ空間内のアドレスにマップされる CPU レジスタです。

注:

1) ST0_55、ST1_55、および ST3_55 は、2 つのアドレスで相互アクセス可能です。1 つのアドレスでは、すべての TMS320C55x ビットが使用可能です。もう一方のアドレス（保護アドレス）では、ある特定のビットは変更できません。保護アドレスが用意されているのは、TMS320C54x™ の ST0、ST1、および PMST（ST3_55 の C54x™ に相当する機能）にライトするコードをサポートするためです。

2) T3、RSA0L、REA0L、および SP は、2 つのアドレスで相互アクセス可能です。DP 直接アドレッシング・モードのメモリ・マップド・レジスタを使用してアクセスする場合、アセンブラは、次のように、2 つのアドレスの上位部分を置き換えます。
T3 = 23h (0Eh ではない)、RSA0L = 3Dh (1Bh ではない)、REA0L = 3Fh (1Ch ではない)、SP = 4Dh (18h ではない)

3) BRC1 をロードする C55x 命令は、同じ値を BRS1 にロードします。

表 2-2. メモリ・マップド・レジスタ

アドレス	レジスタ	説明	ビット範囲	参照先
00 0000h	IER0	割り込みイネーブル・レジスタ 0	15 ～ 0	P. 2-27
00 0001h	IFR0	割り込みフラグ・レジスタ 0	15 ～ 0	P. 2-24
00 0002h（C55x コードの場合）	ST0_55	ステータス・レジスタ 0	15 ～ 0	P. 2-37
注：アドレス 00 0002h は、ST0_55 にアクセスする TMS320C55x のネイティブ・コード用です。ST0 にアクセスするために書き込まれた TMS320C54x コードは、ST0_55 へのアクセスにアドレス 00 0006h を使用します。				
00 0003h（C55x コードの場合）	ST1_55	ステータス・レジスタ 1	15 ～ 0	P. 2-37
注：アドレス 00 0003h は、ST1_55 にアクセスする TMS320C55x のネイティブ・コード用です。ST1 にアクセスするために書き込まれた TMS320C54x コードは、ST1_55 へのアクセスにアドレス 00 0007h を使用します。				
00 0004h（C55x コードの場合）	ST3_55	ステータス・レジスタ 3	15 ～ 0	P. 2-37
注：アドレス 00 0004h は、ST3_55 にアクセスする TMS320C55x のネイティブ・コード用です。プロセッサ・モード・ステータス・レジスタ（PMST）にアクセスするために書き込まれた TMS320C54x コードは、ST3_55 へのアクセスにアドレス 00 001Dh を使用します。				
00 0005h	-	予約済み（このアドレスを使用しないでください）	-	-

表 2-2. メモリ・マップド・レジスタ（続き）

アドレス	レジスタ	説明	ビット範囲	参照先
00 0006h （C54x コードの場合）	ST0 (ST0_55)	ステータス・レジスタ 0	15 ～ 0	P. 2-37
注：アドレス 00 0006h は、ST0_55 の保護アドレスです。このアドレスは、ST0 にアクセスするために書き込まれた TMS320C54x コード用です。TMS320C55x のネイティブ・コードは、ST0_55 へのアクセスにアドレス 00 0002h を使用します。				
00 0007h （C54x コードの場合）	ST1 (ST1_55)	ステータス・レジスタ 1	15 ～ 0	P. 2-37
注：アドレス 00 0007h は、ST1_55 の保護アドレスです。このアドレスは、ST1 にアクセスするために書き込まれた TMS320C54x コード用です。TMS320C55x のネイティブ・コードは、ST1_55 へのアクセスにアドレス 00 0003h を使用します。				
00 0008h 00 0009h 00 000Ah	AC0L AC0H AC0G	アキュムレータ 0	15 ～ 0 31 ～ 16 39 ～ 32	P. 2-9
00 000Bh 00 000Ch 00 000Dh	AC1L AC1H AC1G	アキュムレータ 1	15 ～ 0 31 ～ 16 39 ～ 32	P. 2-9
00 000Eh	T3	一時レジスタ 3	15 ～ 0	P. 2-11
00 000Fh	TRN0	トランジッション・レジスタ 0	15 ～ 0	P. 2-10
00 0010h	AR0	補助レジスタ 0	15 ～ 0	P. 2-12
00 0011h	AR1	補助レジスタ 1	15 ～ 0	P. 2-12
00 0012h	AR2	補助レジスタ 2	15 ～ 0	P. 2-12
00 0013h	AR3	補助レジスタ 3	15 ～ 0	P. 2-12
00 0014h	AR4	補助レジスタ 4	15 ～ 0	P. 2-12
00 0015h	AR5	補助レジスタ 5	15 ～ 0	P. 2-12
00 0016h	AR6	補助レジスタ 6	15 ～ 0	P. 2-12
00 0017h	AR7	補助レジスタ 7	15 ～ 0	P. 2-12
00 0018h	SP	データ・スタック・ポインタ	15 ～ 0	P. 2-18
00 0019h	BK03	AR0 ～ AR3 用サーキュラ・バッファ・サイズ・レジスタ	15 ～ 0	P. 2-16
注：TMS320C54x 互換モード（C54CM = 1）では、BK03 がすべての補助レジスタに対して使用されます。C54CM は、ステータス・レジスタ 1（ST1_55）のビットです。ステータス・レジスタについては、P. 2-37 で説明します。				
00 001Ah	BRC0	ブロック・リピート・カウンタ 0	15 ～ 0	P. 2-34

表 2-2. メモリ・マップド・レジスタ（続き）

アドレス	レジスタ	説明	ビット範囲	参照先
00 001Bh	RSA0L	ブロック・リピート・スタート・アドレス・レジスタ 0 の下位部分	15 ～ 0	P. 2-34
00 001Ch	REA0L	ブロック・リピート・エンド・アドレス・レジスタ 0 の下位部分	15 ～ 0	P. 2-34
00 001Dh（C54x コードの場合）	PMST (ST3_55)	ステータス・レジスタ 3	15 ～ 0	P. 2-37
注：アドレス 00 001Dh は、ST3_55 の保護アドレスです。このアドレスは、プロセッサ・モード・ステータス・レジスタ（PMST）にアクセスするために書き込まれた TMS320C54x コード用です。TMS320C55x のネイティブ・コードは、ST3_55 へのアクセスにアドレス 00 0004h を使用します。				
00 001Eh	XPC	このアドレスは、拡張プログラム・カウンタ（XPC）を使用する TMS320C54x コードとの互換用です。	7 ～ 0	-
00 001Fh	-	予約済み（このアドレスを使用しないでください）	-	-
00 0020h	T0	一時レジスタ 0	15 ～ 0	P. 2-11
00 0021h	T1	一時レジスタ 1	15 ～ 0	P. 2-11
00 0022h	T2	一時レジスタ 2	15 ～ 0	P. 2-11
00 0023h	T3	一時レジスタ 3	15 ～ 0	P. 2-11
00 0024h 00 0025h 00 0026h	AC2L AC2H AC2G	アキュムレータ 2	15 ～ 0 31 ～ 16 39 ～ 32	P. 2-9
00 0027h	CDP	係数データ・ポインタ	15 ～ 0	P. 2-14
00 0028h 00 0029h 00 002Ah	AC3L AC3H AC3G	アキュムレータ 3	15 ～ 0 31 ～ 16 39 ～ 32	P. 2-9
00 002Bh	DPH	拡張データ・ページ・レジスタの上位部分	6 ～ 0	P. 2-17
00 002Ch 00 002Dh	- -	予約済み（これらのアドレスを使用しないでください）	- -	-
00 002Eh	DP	データ・ページ・レジスタ	15 ～ 0	P. 2-17
00 002Fh	PDP	ペリフェラル・データ・ページ・レジスタ	8 ～ 0	P. 2-18
00 0030h	BK47	AR4 ～ AR7 用サーキュラ・バッファ・サイズ・レジスタ	15 ～ 0	P. 2-16

表 2-2. メモリ・マップド・レジスタ（続き）

アドレス	レジスタ	説明	ビット範囲	参照先
00 0031h	BKC	CDP 用サーキュラ・バッファ・サイズ・レジスタ	15 ～ 0	P. 2-16
00 0032h	BSA01	AR0 および AR1 用サーキュラ・バッファ・スタート・アドレス・レジスタ	15 ～ 0	P. 2-15
00 0033h	BSA23	AR2 および AR3 用サーキュラ・バッファ・スタート・アドレス・レジスタ	15 ～ 0	P. 2-15
00 0034h	BSA45	AR4 および AR5 用サーキュラ・バッファ・スタート・アドレス・レジスタ	15 ～ 0	P. 2-15
00 0035h	BSA67	AR6 および AR7 用サーキュラ・バッファ・スタート・アドレス・レジスタ	15 ～ 0	P. 2-15
00 0036h	BSAC	CDP 用サーキュラ・バッファ・スタート・アドレス・レジスタ	15 ～ 0	P. 2-15
00 0037h	-	BIOS 用に予約済み。このロケーションは、BIOS の演算に必要なデータ・テーブル・ポインタ用のスタートアップ格納ロケーションとして使用される 16 ビットのレジスタを含みます。	-	-
00 0038h	TRN1	トランジッション・レジスタ 1	15 ～ 0	P. 2-10
00 0039h	BRC1	ブロック・リピート・カウンタ 1	15 ～ 0	P. 2-34
00 003Ah	BRS1	BRC1 保存レジスタ	15 ～ 0	P. 2-34
00 003Bh	CSR	計算シングル・リピート・レジスタ	15 ～ 0	P. 2-34
00 003Ch 00 003Dh	RSA0H RSA0L	ブロック・リピート・スタート・アドレス・レジスタ 0	23 ～ 16 15 ～ 0	P. 2-34
00 003Eh 00 003Fh	REA0H REA0L	ブロック・リピート・エンド・アドレス・レジスタ 0	23 ～ 16 15 ～ 0	P. 2-34
00 0040h 00 0041h	RSA1H RSA1L	ブロック・リピート・スタート・アドレス・レジスタ 1	23 ～ 16 15 ～ 0	P. 2-34
00 0042h 00 0043h	REA1H REA1L	ブロック・リピート・エンド・アドレス・レジスタ 1	23 ～ 16 15 ～ 0	P. 2-34
00 0044h	RPTC	シングル・リピート・カウンタ	15 ～ 0	P. 2-34
00 0045h	IER1	割り込みイネーブル・レジスタ 1	10 ～ 0	P. 2-27

表 2-2. メモリ・マップド・レジスタ（続き）

アドレス	レジスタ	説明	ビット範囲	参照先
00 0046h	IFR1	割り込みフラグ・レジスタ 1	10 〜 0	P. 2-24
00 0047h	DBIER0	デバッグ割り込みイネーブル・レジスタ 0	15 〜 0	P. 2-30
00 0048h	DBIER1	デバッグ割り込みイネーブル・レジスタ 1	10 〜 0	P. 2-30
00 0049h	IVPD	ベクタ 0 〜 15 および 24 〜 31 用割り込みベクタ・ポインタ	15 〜 0	P. 2-23
00 004Ah	IVPH	ベクタ 16 〜 23 用割り込みベクタ・ポインタ	15 〜 0	P. 2-23
00 004Bh	ST2_55	ステータス・レジスタ 2	15 〜 0	P. 2-37
00 004Ch	SSP	システム・スタック・ポインタ	15 〜 0	P. 2-18
00 004Dh	SP	データ・スタック・ポインタ	15 〜 0	P. 2-18
00 004Eh	SPH	拡張スタック・ポインタの上位部分	6 〜 0	P. 2-18
00 004Fh	CDPH	拡張係数データ・ポインタの上位部分	6 〜 0	P. 2-14
00 0050h 〜 00 005Fh	-	予約済み（これらのアドレスを使用しないでください）	-	-

2.3 アキュムレータ（AC0 ～ AC3）

CPU は、4 つの 40 ビット・アキュムレータ、AC0、AC1、AC2、および AC3 を含みます（図 2-1 を参照）。これらのレジスタの基本機能は、D ユニットの次に示す構成部分、（算術論理演算ユニット（ALU）、積和演算ユニット（MAC）、およびシフタ）でのデータの計算を支援することです。4 つのアキュムレータの機能は基本的に同じですが、一部の命令はある特定のアキュムレータ同士を組み合わせる場合に制限があります。次に例を示します。

`SWAP AC0, AC2;` 有効な命令

`SWAP AC1, AC3;` 有効な命令

ただし、

`SWAP AC0, AC1;` 無効な命令

各アキュムレータは、下位ワード（ACxL）、上位ワード（ACxH）、および 8 つの保護ビット（ACxG）に区分されます。これらの各区分にそれぞれアクセスするには、メモリ・マップド・レジスタにアクセスするアドレッシング・モードを使用できます。

TMS320C54x 互換モード（C54CM = 1）では、アキュムレータ AC0 と AC1 が、TMS320C54x のアキュムレータ A と B にそれぞれ対応します。

図 2-1. アキュムレータ

	39-32	31-16	15-0
AC0	AC0G	AC0H	AC0L
AC1	AC1G	AC1H	AC1L
AC2	AC2G	AC2H	AC2L
AC3	AC3G	AC3H	AC3L

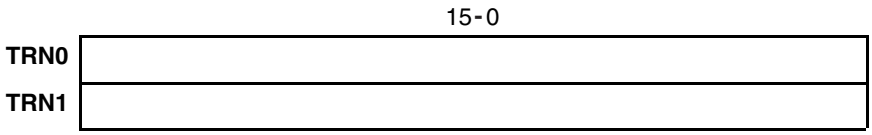
2.4 トランジッション・レジスタ（TRN0、TRN1）

2 つのトランジッション・レジスタ（図 2-2 を参照）が、比較選択極値命令に使用されます。

- ❑ 2 つの 16 ビット極値選択を実行する構文は、2 つのアキュムレータの上位ワードと下位ワードの比較に基づいて、TRN0 と TRN1 を更新します。TRN0 はアキュムレータの上位ワードの比較に基づいて更新され、TRN1 は下位ワードの比較に基づいて更新されます。
- ❑ 1 つの 40 ビット極値選択を実行する構文は、2 つのアキュムレータの 40 ビット全体の比較に基づいて、選択されているトランジッション・レジスタ（TRN0 または TRN1）を更新します。

TRN0 と TRN1 は、ビタビ・アルゴリズム実装における新規メトリックスへのパスの変換決定を保持できます。

図 2-2. トランジッション・レジスタ



2.5 一時レジスタ（T0 ～ T3）

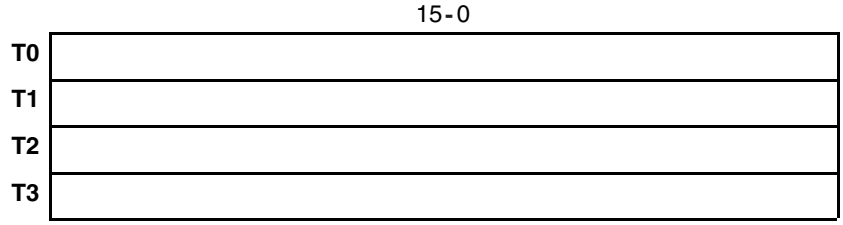
CPU には、4 つの 16 ビット汎用一時レジスタ T0 ～ T3（図 2-3 を参照）があります。一時レジスタの機能の一部を次に示します。

- ❑ 乗算、積和演算、および積差演算の各命令に対するメモリ被乗数の 1 つを保持します。
- ❑ D ユニットで実行される加算、減算、ロードの各命令に使用するシフト・カウントを保持します。
- ❑ 補助レジスタ（AR0 ～ AR7）の内容と一時レジスタをスワップすることにより、より多くのポインタ値を保持します（スワップ命令を使用）。
- ❑ D ユニット ALU で実行されるデュアル 16 ビット演算に対するビタビ・バタフライの変換メトリックを保持します。

注：

C54CM = 1 （TMS320C54x **互換モードがオン）の場合、**T2 **はステータス・レジスタ** ST1_55 **の** ASM **ビットに結び付けられているので、汎用レジスタとして使われません。詳細は、**2.10.2.1 **項「**ASM **ビット・フィールド（**ST1_55**）」を参照してください。**

図 2-3. 一時レジスタ



2.6 データ空間と I/O 空間のアドレス指定に使用するレジスタ

ここでは、次のレジスタについて説明します。

レジスタ	機能	参照先
XAR0 ～ XAR7 および AR0 ～ AR7	間接アドレッシング・モードで行われるアクセスのためにデータ空間の値を指します。	P. 2-12
XCDP および CDP	間接アドレッシング・モードで行われるアクセスのためにデータ空間の値を指します。	P. 2-14
BSA01、BSA23、BSA45、BSA67、BSAC	ポインタに追加されるサーキュラ・バッファ・スタート・アドレスを指定します。	P. 2-15
BK03、BK47、BKC	サーキュラ・バッファ・サイズを指定します。	P. 2-16
XDP および DP	DP 直接アドレッシング・モードで行われるアクセスのためにスタート・アドレスを指定します。	P. 2-17
PDP	I/O 空間へのアクセスのためにペリフェラル・データ・ページを識別します。	P. 2-18
XSP および SP	データ・スタック上の値をポイントします。	P. 2-18
XSSP および SSP	システム・スタック上の値をポイントします。	P. 2-18

2.6.1 補助レジスタ（XAR0 ～ XAR7 / AR0 ～ AR7）

CPU は、8 つの拡張補助レジスタ XAR0 ～ XAR7 を含みます（図 2-4 と表 2-3 を参照）。それぞれの上位部分（たとえば、AR0H）はデータ空間へのアクセスのために 7 ビットのメイン・データ・ページを指定するために使用されます。それぞれの下位部分（たとえば、AR0）は、以下のように使用できます。

- 7 ビットのメイン・データ・ページへの 16 ビット・オフセット（23 ビット・アドレスを形成するため）
- ビット・アドレス（各ビットまたはビット・ペアにアクセスする命令内）
- 汎用レジスタまたはカウンタ
- サーキュラ・バッファのスタート・アドレスに関連するワードを選択するためのインデックス（6.11 節「サーキュラ・アドレッシング」を参照）

図 2-4.　拡張補助レジスタと構成部分

	22-16	15-0
XAR0	AR0H	AR0
XAR1	AR1H	AR1
XAR2	AR2H	AR2
XAR3	AR3H	AR3
XAR4	AR4H	AR4
XAR5	AR5H	AR5
XAR6	AR6H	AR6
XAR7	AR7H	AR7

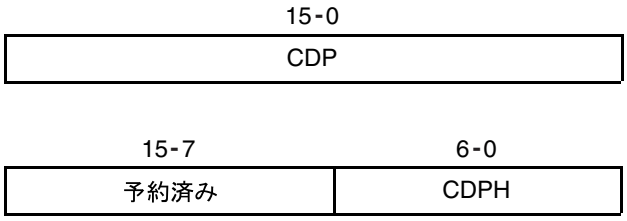
表 2-3.　拡張補助レジスタと構成部分

レジスタ	名称	アクセス方法
XARn	拡張補助レジスタ n	専用の命令を使用してのみアクセス可能。XARn は、メモリにはマップされません。
ARn	補助レジスタ n	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。
ARnH	拡張補助レジスタ n の上位部分	個別でのアクセス不能。ARnH にアクセスするには、XARn にアクセスする必要があります。

XAR0 ～ XAR7 または AR0 ～ AR7 は、AR 間接アドレッシング・モードとデュアル AR 間接アドレッシング・モードで使用されます。算術、論理、シフトの各基本演算は、A ユニットの算術論理演算ユニット（ALU）の AR0 ～ AR7 で実行できます。これらの演算は、データ・アドレス生成ユニット（DAGEN）の補助レジスタで実行されるアドレスの変更と並列に実行できます。

2.6.2 係数データ・ポインタ（XCDP / CDP）

CPU は、そのメモリ・マップ内に係数データ・ポインタ CDP、および関連した拡張レジスタ CDPH を含みます。



CPU は、XCDP と呼ばれる拡張した CDP を形成するために、2 つを連結することができます（図 2-5 と表 2-4 を参照）。上位部分（CDPH）は、データ空間へのアクセスのために 7 ビットのメイン・データ・ページを指定するために使用されます。下位部分（CDP）は以下のように使用できます。

- ❑ 7 ビットのメイン・データ・ページへの 16 ビット・オフセット（23 ビット・アドレスを形成するため）
- ❑ ビット・アドレス（各ビットまたはビット・ペアにアクセスする命令内）
- ❑ 汎用レジスタまたはカウンタ
- ❑ サーキュラ・バッファのスタート・アドレスに関連するワードを選択するためのインデックス（6.11 節「サーキュラ・アドレッシング」を参照）

図 2-5. 拡張係数データ・ポインタと構成部分

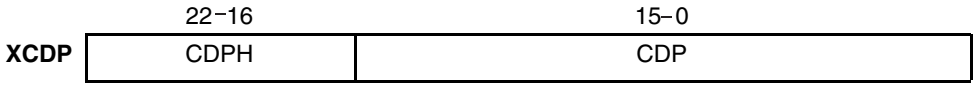


表 2-4. 拡張係数データ・ポインタと構成部分

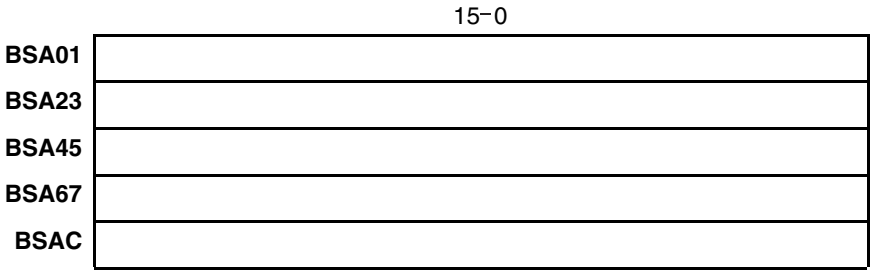
レジスタ	名称	アクセス方法
XCDP	拡張係数データ・ポインタ	専用の命令を使用してのみアクセス可能。XCDP は、メモリにマップされるレジスタではありません。
CDP	係数データ・ポインタ	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。
CDPH	拡張係数データ・ポインタの上位部分	メモリ・マップド・レジスタとしてアクセス可能。また、XCDP にアクセスして、CDPH にアクセスすることも可能です。CDPH には専用の命令はありません。

XCDP と CDP は、CDP 間接アドレッシング・モードおよび係数間接アドレッシング・モードで使用されます。CDP は、1 つのデータ空間値にアクセスするすべての命令で使用できますが、3 番目の独立したオペランドを D ユニットのデュアル MAC 演算子に供給するため、デュアル積和演算（MAC）命令でより有効に使用できます。

2.6.3 サーキュラ・バッファ・スタート・アドレス・レジスタ (BSA01、BSA23、BSA45、BSA67、BSAC)

CPU は、スタート・アドレスにアラインメントの制約を考えずにサーキュラ・バッファを定義することができるように、5 つの 16 ビット・サーキュラ・バッファ・スタート・アドレス・レジスタ（図 2-6 を参照）を含みます。

図 2-6. サーキュラ・バッファ・スタート・アドレス・レジスタ



各バッファ・スタート・アドレス・レジスタは、特定のポインタ（1 つまたは複数）に関連付けられています（表 2-5 を参照）。バッファ・スタート・アドレスは、ポインタがステータス・レジスタ ST2_55 でサーキュラ・アドレッシングに設定されている場合のみポインタ値に追加されます。

表 2-5. サーキュラ・バッファ・スタート・アドレス・レジスタと関連ポインタ

レジスタ	ポインタ	メイン・データ・ページの供給元
BSA01	AR0 または AR1	AR0 には AR0H AR1 には AR1H
BSA23	AR2 または AR3	AR2 には AR2H AR3 には AR3H
BSA45	AR4 または AR5	AR4 には AR4H AR5 には AR5H
BSA67	AR6 または AR7	AR6 には AR6H AR7 には AR7H
BSAC	CDP	CDPH

バッファ・スタート・アドレスを使用する例として、次の命令を参考にしてください。

```
MOV *AR6, T2      ; T2 に XAR6 が参照するワードの
                  ; サーキュラ・バッファからの値をロードします。
```

この例では、AR6 がサーキュラ・アドレッシングに設定されており、生成されるアドレスの形式は次のようになります。メイン・データ・ページ値（AR6H）は、AR6 と関連するバッファ・スタート・アドレス（BSA67）の和と連結されます。

AR6H:(BSA67 + AR6) = XAR6 + BSA67

TMS320C54x コードを互換モード（C54CM = 1）で実行する場合、バッファ・スタート・アドレス・レジスタに 0（ゼロ）が含まれることを確認してください。

2.6.4 サーキュラ・バッファ・サイズ・レジスタ（BK03、BK47、BKC）

3 つの 16 ビット・サーキュラ・バッファ・サイズ・レジスタ（図 2-7 を参照）が、サーキュラ・バッファ内のワード数（最大 65535）を指定します。各バッファ・サイズ・レジスタは、特定のポインタ（1 つまたは複数）に関連付けられています（表 2-6 を参照）。

図 2-7. サーキュラ・バッファ・サイズ・レジスタ

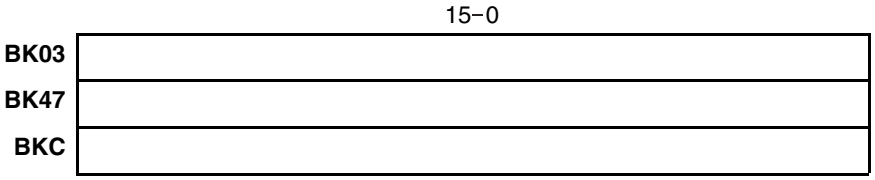


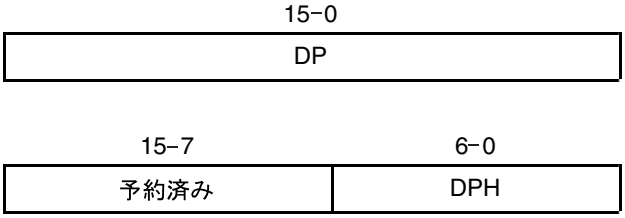
表 2-6. サーキュラ・バッファ・サイズ・レジスタと関連ポインタ

レジスタ	ポインタ
BK03	AR0、AR1、AR2、または AR3
BK47	AR4、AR5、AR6、または AR7
BKC	CDP

TMS320C54x 互換モード（C54CM = 1）では、BK03 がすべての補助レジスタに使用され、BK47 は使用されません。

2.6.5 データ・ページ・レジスタ（XDP / DP）

CPU は、そのメモリ・マップ内にデータ・ページ・レジスタ DP、および関連した拡張レジスタ DPH を含みます。



CPU は、XDP と呼ばれる拡張した DP を形成するため DP と DPH の 2 つを連結します（図 2-8 と表 2-7 を参照）。上位部分（DPH）は、データ空間へのアクセスのための 7 ビットのメイン・データ・ページを指定するために使用されます。下位部分は、23 ビット・アドレスを形成するために、メイン・データ・ページと連結される 16 ビット・オフセットを指定します。

図 2-8. 拡張データ・ページ・レジスタと構成部分

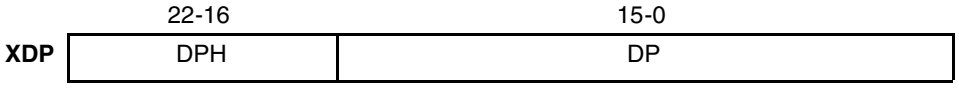


表 2-7. 拡張データ・ページ・レジスタと構成部分

レジスタ	名称	アクセス方法
XDP	拡張データ・ページ・レジスタ	専用の命令を使用してのみアクセス可能。XDP は、メモリにマップされるレジスタではありません。
DP	データ・ページ・レジスタ	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。
DPH	拡張データ・ページ・レジスタの上位部分	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。

DP 直接アドレッシング・モードでは、XDP は 23 ビットのアドレスを指定し、k16 絶対アドレッシング・モードでは、DPH は 23 ビットのアドレスを形成するために 16 ビットの即値と連結されます。

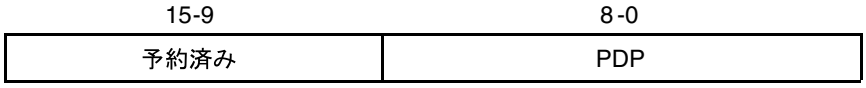
2.6.6 ペリフェラル・データ・ページ・レジスタ（PDP）

PDP 直接アドレッシング・モードの場合、9 ビットのペリフェラル・データ・ページ・レジスタ（PDP）が、64K ワードの I/O 空間内で 128 ワードのページを選択します。

PDP は、16 ビットのレジスタ・ロケーション内の 9 ビット・フィールドです（図 2-9 を参照）。このロケーションのビット 15 ～ 9 は、CPU により無視されます。

このレジスタは、専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能です。

図 2-9. ペリフェラル・データ・ページ・レジスタ



2.6.7 スタック・ポインタ（XSP / SP、XSSP / SSP）

CPU は、そのメモリ・マップ内にデータ・スタック・ポインタ（SP）、システム・スタック・ポインタ（SSP）、および関連した拡張レジスタ（SPH）を含みます。

図 2-10 と表 2-8 を参照してください。データ・スタックにアクセスする場合、CPU は XSP と呼ばれる拡張 SP を形成するために SPH を SP と連結します。XSP は、データ・スタックに前回プッシュされた値のアドレスを含みます。SPH は、メモリの 7 ビットのメイン・データ・ページを保持し、SP はそのページの特定のワードを指します。

同様に、システム・スタックにアクセスする場合、CPU は XSSP を形成するために SPH を SSP と連結します。XSSP は、システム・スタックに前回プッシュされた値のアドレスを含みます。

図 2-10.　拡張スタック・ポインタ

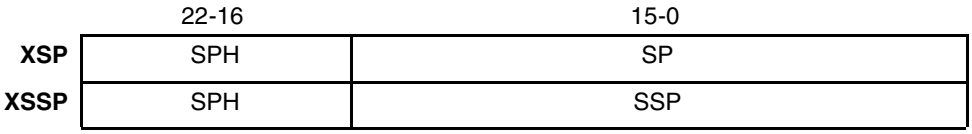


表 2-8.　スタック・ポインタ・レジスタ

レジスタ	名称	アクセス方法
XSP	拡張データ・スタック・ポインタ	専用の命令を使用してのみアクセス可能。XSP は、メモリにマップされるレジスタではありません。
SP	データ・スタック・ポインタ	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。
XSSP	拡張システム・スタック・ポインタ	専用の命令を使用してのみアクセス可能。XSSP は、メモリにマップされるレジスタではありません。
SSP	システム・スタック・ポインタ	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。
SPH	XSP と XSSP の上位部分	メモリ・マップド・レジスタとしてアクセス可能。また、XSP または XSSP にアクセスしても SPH にアクセスできます。SPH には専用の命令はありません。 **注：**SPH は、XSP または XSSP への書き込みによる影響を受けます。

XSP は、SP 直接アドレッシング・モードで使用されます。表 2-9 の命令は、SP と SSP の使用および変更（またはいずれか一方）を行います。

表 2-9.　SP と SSP を使用および変更する命令

命令の種類	説明
ソフトウェア割り込み、ソフトウェア・トラップ、ソフトウェア・リセット、無条件呼び出し、条件付き呼び出し	これらの命令は、データをデータ・スタックとシステム・スタックにそれぞれプッシュします。SP と SSP は、データ値の各ペアがプッシュされる前にデクリメントされます。
プッシュ	この命令は、データをデータ・スタックのみにプッシュします。SP は、データがプッシュされる前にデクリメントされます。
無条件復帰、条件付き復帰、割り込みからの復帰	これらの命令は、データ・スタックとシステム・スタックからデータをポップします。SP と SSP は、データ値の各ペアがポップされてからインクリメントされます。
ポップ	この命令は、データ・スタックからのみデータをポップします。SP は、データがポップされてからインクリメントされます。

スタック・ポインタのインクリメントとデクリメントは、SP と SSP に対して行われます。2 つのメイン・データ・ページをまたぐスタックのアドレス指定は、拡張レジスタ（SPH）の値を変更することなく行うことはできません。

注:

FFFFh を超えてインクリメントしたり、0000h を超えてデクリメントを行うと、ポインタ値が初期状態に戻ってしまいます。ただし、この動作はサポートされている機能ではないので、利用してはいけません。

2.7 プログラム・フロー・レジスタ（PC、RETA、CFCT）

CPU が適切なプログラム・フローを維持するために使用する 3 つのレジスタについて表 2-10 で説明します。

表 2-10. プログラム・フロー・レジスタ

レジスタ	説明
PC	プログラム・カウンタ。この 24 ビットのレジスタは、I ユニットでデコードされるコードの 1 ～ 6 バイトのアドレスを保持します。CPU が割り込みまたは呼び出しを実行すると、現行の PC の値（リターン・アドレス）が格納され、その後 PC に新しいアドレスがロードされます。CPU が割り込みサービス・ルーチンまたはコールされたサブルーチンから復帰すると、リターン・アドレスが PC に復元されます。
RETA	リターン・アドレス・レジスタ。選択されているスタック構成（4.2 節を参照）でファースト・リターン・プロセスを使用している場合、RETA はサブルーチンの実行中にリターン・アドレスの場所を一時的に保持します。RETA は、CFCT を伴って、複数レイヤーのサブルーチンの効率的な実行を可能にします。ユーザーは、専用の 32 ビットのロード命令とストア命令を使用して、RETA と CFCT のペアとのリードまたはライトを行うことができます。
CFCT	制御フロー・コンテキスト・レジスタ。CPU は、アクティブなリピート・ループ（ループ・コンテキスト）を記録します。選択されているスタック構成（4.2 節を参照）でファースト・リターン・プロセスを使用している場合、CFCT はサブルーチンの実行中に 8 ビットのループ・コンテキストの場所を一時的に保持します。CFCT は、RETA を伴って、複数レイヤーのサブルーチンの効率的な実行を可能にします。ユーザーは、専用の 32 ビットのロード命令とストア命令を使用して、RETA と CFCT のペアとのリードまたはライトを行うことができます。

注：

RETA と CFCT は、DSP ハードウェア・リセットによりゼロクリアされます。また、push/pop 命令やソフトウェア・リセット命令の影響も受けません。

2.7.1 CFCT に格納されるコンテキスト・ビット

CPU は、ルーチン内にループ・コンテキスト、つまりリピート・ループのステータス（アクティブまたはインアクティブ）を格納するための内部ビットを持っています。CPU が割り込みまたは呼び出しを実行すると、ループ・コンテキストが CFCT に格納されます。CPU が割り込みまたは呼び出されたサブルーチンから復帰すると、ループ・コンテキストが CFCT から復元されます。8 ビットの CFCT では、表 2-11 に示されているループ・コンテキストのビット形式になります。

表 2-11. CFCT におけるループ・コンテキストのビット形式

ビット	説明
7	このビットは、単一リピート・ループがアクティブかどうかを示します。 0：非アクティブ 1：アクティブ
6	このビットは、条件付きの単一リピート・ループがアクティブかどうかを示します。 0：非アクティブ 1：アクティブ
5 〜 4	予約済み
3 〜 0	この 4 ビットのコードは、ブロック・リピート・ループの 2 つのレベル、外部（レベル 0）ループと内部（レベル 1）ループのステータスを示します。選択するブロック・リピート命令の種類に応じて、アクティブ・ループは、ローカル（すべてのコードが命令バッファ・キュー内から繰り返し実行される）、または外部（コードが命令バッファ・キューを通って CPU に繰り返しフェッチされ、転送される）のいずれかになります。

ブロック・リピートのコード	レベル 0 のループ	レベル 1 のループ
0	非アクティブ	非アクティブ
2	アクティブ、外部	非アクティブ
3	アクティブ、ローカル	非アクティブ
7	アクティブ、外部	アクティブ、外部
8	アクティブ、外部	アクティブ、ローカル
9	アクティブ、ローカル	アクティブ、ローカル
その他：予約済み	-	-

2.8 割り込み管理用レジスタ

ここでは、次のレジスタについて説明します。

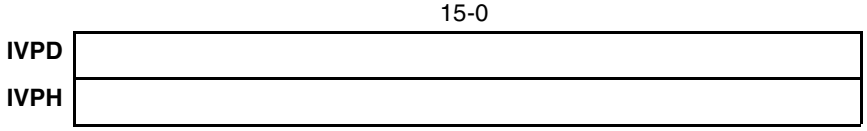
レジスタ	機能	参照先
IVPD	割り込みベクタ 0 ～ 15 および 24 ～ 31 を指します。	2.8.1
IVPH	割り込みベクタ 16 ～ 23 を指します。	2.8.1
IFR0、IFR1	どのマスカブル割り込みが要求されているか示します。	2.8.2
IER0、IER1	マスカブル割り込みをイネーブルまたはディスエーブルにします。	2.8.3
DBIER0、DBIER1	デバッグ時に選択マスカブル割り込みをタイム・クリティカル割り込みに構成します。	2.8.4

2.8.1 割り込みベクタ・ポインタ（IVPD、IVPH）

IVPD と IVPH（図 2-11 を参照）の 2 つの 16 ビット割り込みベクタ・ポインタは、最大 32 のプログラム空間の割り込みベクタを指します。IVPD は、割り込みベクタ（0 ～ 15 および 24 ～ 31）の 256 バイトのプログラム・ページを指します。IVPH は、割り込みベクタ（16 ～ 23）の 256 バイトのプログラム・ページを指します。

IVPD と IVPH に同じ値がある場合、割り込みベクタはすべて同じ 256 バイトのプログラム・ページ内になります。DSP のハードウェア・リセットにより、2 つの IVP に FFFFh がロードされます。2 つの IVP は、ソフトウェア・リセット命令の影響を受けません。

図 2-11. 割り込みベクタ・ポインタ



IVP を変更する前に、次の項目を確認してください。

- ❑ マスカブル割り込みはグローバルにディスエーブルになります（INTM = 1）。これは、2 つの IVP が新規ベクタを指すように変更される前に、マスカブル割り込みが発生するのを防ぎます。
- ❑ ハードウェアの各ノンマスカブル割り込みは、既存の IVPD 値と新規の IVPD 値に対して 1 つのベクタと 1 つの割り込みサービス・ルーチンを持っています。これは、ハードウェアのノンマスカブル割り込みが IVPD の変更中に発生する場合に、不正な命令コードをフェッチするのを防ぎます。

異なる割り込みベクタに対してベクタ・アドレスが形成される方法を表 2-12 に示します。CPU は、16 ビットの割り込みベクタ・ポインタを 5 ビット（たとえば、IV1 の 00001、IV16 の 10000）でコーディングされ、3 ビット左にシフトされたベクタ番号と連結します。

表 2-12. ベクタおよびベクタ・アドレスの形成

ベクタ	割り込み	ベクタ・アドレス		
		ビット 23 ～ 8	ビット 7 ～ 3	ビット 2 ～ 0
IV0	リセット	IVPD	00000	000
IV1	ノンマスカブル・ハードウェア割り込み NMI	IVPD	00001	000
IV2 ～ IV15	マスカブル割り込み	IVPD	00010 ～ 01111	000
IV16 ～ IV23	マスカブル割り込み	IVPH	10000 ～ 10111	000
IV24	バス・エラー割り込み（マスカブル）BERRINT	IVPD	11000	000
IV25	データ・ログ割り込み（マスカブル）DLOGINT	IVPD	11001	000
IV26	リアルタイム・オペレーティング・システム割り込み（マスカブル）RTOSINT	IVPD	11010	000
IV27 ～ IV31	汎用ソフトウェア専用割り込み INT27 ～ INT31	IVPD	11011 ～ 11111	000

2.8.2 割り込みフラグ・レジスタ（IFR0、IFR1）

16 ビットの割り込みフラグ・レジスタ IFR1 と IFR0 は、すべてのマスカブル割り込みに対するフラグ・ビットを含みます。マスカブル割り込み要求が CPU に到達すると、2 つの IFR のうちのいずれかで対応するフラグが 1 にセットされます。これは、割り込みが保留されているか CPU からの応答を待機中かを示します。一般的な C55x の IFR を図 2-12 に示します。これらのビットにマップされる割り込みを確認するには、対応する C55x DSP のデータ・マニュアルを参照してください。

IFR をリードして保留されている割り込みを特定し、IFR にライトして保留されている割り込みをクリアできます。割り込み要求をクリア（その IFR ビットをゼロクリア）するには、対応する IFR ビットに 1 をライトします。次に例を示します。

```
; フラグ IF14 と IF2 をクリア：
MOV #0100000000000100b, mmap(@IFR0)
```

保留されているすべての割り込みは、IFR の現行の内容を IFR に書き戻すことでクリアできます。また、ハードウェア割り込み要求の応答も、対応する IFR ビットをクリアします。デバイス・リセットは、すべての IFR ビットをクリアします。

図 2-12. 割り込みフラグ・レジスタ

IFR1

15				11	10	9	8
予約済み					RTOSINTF	DLOGINTF	BERRINTF
R-0					R/W1C-0	R/W1C-0	R/W1C-0

7	6	5	4	3	2	1	0
IF23	IF22	IF21	IF20	IF19	IF18	IF17	IF16
R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0

IFR0

15	14	13	12	11	10	9	8
IF15	IF14	IF13	IF12	IF11	IF10	IF9	IF8
R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0

7	6	5	4	3	2	1	0
IF7	IF6	IF5	IF4	IF3	IF2	予約済み	
R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R-0	

凡例： R = リード・アクセス。W1C = このビットに 1 をライトすることにより CPU がこのビットをゼロクリアします。*-n* = DSP のハードウェア・リセット後の値。予約済み = このビットにライトしても影響はありません。このフィールド内のビットは、リード演算中に常に 0（ゼロ）と表示されます。

2.8.2.1 IFR1 の RTOSINTF ビット

ビット	名前	説明	アクセス方法	HW リセット
10	RTOSINTF	リアルタイム・オペレーティング・システム割り込み（RTOSINT）用の割り込みフラグ・ビット	リード / ライト	0

RTOSINTF ビットをリードする場合、次のようになります。

RTOSINTF	説明
0	RTOSINT が保留中でない
1	RTOSINT が保留中

このフラグ・ビットをゼロクリアする（かつ対応する割り込み要求をクリアする）には、そのビットに 1 をライトします。

2.8.2.2 IFR1 の DLOGINTF ビット

ビット	名前	説明	アクセス方法	HW リセット
9	DLOGINTF	データ・ログ割り込み（DLOGINT）用の割り込みフラグ・ビット	リード / ライト	0

DLOGINTF ビットをリードする場合、次のようになります。

DLOGINTF	説明
0	DLOGINT が保留中でない
1	DLOGINT が保留中

このフラグ・ビットをゼロクリアする（かつ対応する割り込み要求をクリアする）には、そのビットに 1 をライトします。

2.8.2.3 IFR1 の BERRINTF ビット

ビット	名前	説明	アクセス方法	HW リセット
8	BERRINTF	バス・エラー割り込み（BERRINT）用の割り込みフラグ・ビット	リード / ライト	0

BERRINTF ビットをリードする場合、次のようになります。

BERRINTF	説明
0	BERRINT が保留中でない
1	BERRINT が保留中

このフラグ・ビットをゼロクリアする（かつ対応する割り込み要求をクリアする）には、そのビットに 1 をライトします。

2.8.2.4 IFR1 の IF16 ～ IF23 ビット

ビット	名前	説明	アクセス方法	HW リセット
0	IF16	割り込みフラグ・ビット 16	リード / ライト	0
1	IF17	割り込みフラグ・ビット 17	リード / ライト	0
2	IF18	割り込みフラグ・ビット 18	リード / ライト	0
3	IF19	割り込みフラグ・ビット 19	リード / ライト	0
4	IF20	割り込みフラグ・ビット 20	リード / ライト	0
5	IF21	割り込みフラグ・ビット 21	リード / ライト	0
6	IF22	割り込みフラグ・ビット 22	リード / ライト	0
7	IF23	割り込みフラグ・ビット 23	リード / ライト	0

これらのビットをリードする場合、次のようになります（x は 16 ～ 23 の数字です）。

IFx	説明
0	割り込みベクタ x に関連付けられている割り込みが保留中でない
1	割り込みベクタ x に関連付けられている割り込みが保留中

いずれかのフラグ・ビットをゼロクリアする（かつ対応する割り込み要求をクリアする）には、そのビットに 1 をライトします。

2.8.2.5 IFR0 の IF2 ～ IF15 ビット

ビット	名前	説明	アクセス方法	HW リセット
2	IF2	割り込みフラグ・ビット 2	リード / ライト	0
3	IF3	割り込みフラグ・ビット 3	リード / ライト	0
4	IF4	割り込みフラグ・ビット 4	リード / ライト	0
5	IF5	割り込みフラグ・ビット 5	リード / ライト	0
6	IF6	割り込みフラグ・ビット 6	リード / ライト	0
7	IF7	割り込みフラグ・ビット 7	リード / ライト	0
8	IF8	割り込みフラグ・ビット 8	リード / ライト	0
9	IF9	割り込みフラグ・ビット 9	リード / ライト	0
10	IF10	割り込みフラグ・ビット 10	リード / ライト	0
11	IF11	割り込みフラグ・ビット 11	リード / ライト	0
12	IF12	割り込みフラグ・ビット 12	リード / ライト	0
13	IF13	割り込みフラグ・ビット 13	リード / ライト	0
14	IF14	割り込みフラグ・ビット 14	リード / ライト	0
15	IF15	割り込みフラグ・ビット 15	リード / ライト	0

これらのビットをリードする場合、次のようになります（x は 2 ～ 15 の数字です）。

IFx	説明
0	割り込みベクタ x に関連付けられている割り込みが保留中でない
1	割り込みベクタ x に関連付けられている割り込みが保留中

いずれかのフラグ・ビットをゼロクリアする（かつ対応する割り込み要求をクリアする）には、そのビットに 1 をライトします。

2.8.3 割り込みイネーブル・レジスタ（IER0、IER1）

マスカブル割り込みをイネーブルにするには、IER0 または IER1 の対応するビットを 1 にセットします。マスカブル割り込みをディスエーブルにするには、対応するイネーブル・ビットをゼロクリアします。DSP のハードウェア・リセット時に、すべての IER ビットがゼロクリアされ、すべてのマスカブル割り込みがディスエーブルになります。一般的な C55x の IER を図 2-13 に示しています。これらのビットにマップされる割り込みを確認するには、対応する C55x DSP のデータ・マニュアルを参照してください。

注 :

IER1 と IER0 は、ソフトウェア・リセット命令の影響を受けません。これらのレジスタは、グローバルにマスカブル割り込みをイネーブル（INTM=0）にする前に初期化しなければなりません。

図 2-13. 割り込みイネーブル・レジスタ

IER1

15				11	10	9	8
予約済み					RTOSINTE	DLOGINTE	BERRINTE
R-0					R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
IE23	IE22	IE21	IE20	IE19	IE18	IE17	IE16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

IER0

15	14	13	12	11	10	9	8
IE15	IE14	IE13	IE12	IE11	IE10	IE9	IE8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
IE7	IE6	IE5	IE4	IE3	IE2	予約済み	
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	

凡例： R = リード。W = ライト。*-n* = DSP のハードウェア・リセット後の値。

2.8.3.1 IER1 の RTOSINTE ビット

ビット	名前	説明	アクセス方法	HW リセット
10	RTOSINTE	リアルタイム・オペレーティング・システム割り込み（RTOSINT）用のイネーブル・ビット	リード / ライト	0

RTOSINTE ビットは、RTOSINT をイネーブルまたはディスエーブルにします。

RTOSINTE	説明
0	RTOSINT はディスエーブル
1	RTOSINT はイネーブル

2.8.3.2 IER1 の DLOGINTE ビット

ビット	名前	説明	アクセス方法	HW リセット
9	DLOGINTE	データ・ログ割り込み（DLOGINT）用のイネーブル・ビット	リード / ライト	0

DLOGINTE ビットは、DLOGINT をイネーブルまたはディスエーブルにします。

DLOGINTE	説明
0	DLOGINT はディスエーブル
1	DLOGINT はイネーブル

2.8.3.3 IER1 の BERRINTE ビット

ビット	名前	説明	アクセス方法	HW リセット
8	BERRINTE	バス・エラー割り込み（BERRINT）用のイネーブル・ビット	リード / ライト	0

BERRINTE ビットは、BERRINT をイネーブルまたはディスエーブルにします。

BERRINTE	説明
0	BERRINT はディスエーブル
1	BERRINT はイネーブル

2.8.3.4 IER1 の IE16 ～ IE23 ビット

ビット	名前	説明	アクセス方法	HW リセット
0	IE16	割り込みイネーブル・ビット 16	リード / ライト	0
1	IE17	割り込みイネーブル・ビット 17	リード / ライト	0
2	IE18	割り込みイネーブル・ビット 18	リード / ライト	0
3	IE19	割り込みイネーブル・ビット 19	リード / ライト	0
4	IE20	割り込みイネーブル・ビット 20	リード / ライト	0
5	IE21	割り込みイネーブル・ビット 21	リード / ライト	0
6	IE22	割り込みイネーブル・ビット 22	リード / ライト	0
7	IE23	割り込みイネーブル・ビット 23	リード / ライト	0

これらのビットの機能は、次のように要約されます（x は 16 ～ 23 の数字です）。

IEx	説明
0	割り込みベクタ x に関連付けられている割り込みはディスエーブル
1	割り込みベクタ x に関連付けられている割り込みはイネーブル

2.8.3.5 IER0 の IE2 ～ IE15 ビット

ビット	名前	説明	アクセス方法	HW リセット
2	IE2	割り込みイネーブル・ビット 2	リード / ライト	0
3	IE3	割り込みイネーブル・ビット 3	リード / ライト	0
4	IE4	割り込みイネーブル・ビット 4	リード / ライト	0
5	IE5	割り込みイネーブル・ビット 5	リード / ライト	0
6	IE6	割り込みイネーブル・ビット 6	リード / ライト	0
7	IE7	割り込みイネーブル・ビット 7	リード / ライト	0
8	IE8	割り込みイネーブル・ビット 8	リード / ライト	0
9	IE9	割り込みイネーブル・ビット 9	リード / ライト	0
10	IE10	割り込みイネーブル・ビット 10	リード / ライト	0
11	IE11	割り込みイネーブル・ビット 11	リード / ライト	0
12	IE12	割り込みイネーブル・ビット 12	リード / ライト	0
13	IE13	割り込みイネーブル・ビット 13	リード / ライト	0
14	IE14	割り込みイネーブル・ビット 14	リード / ライト	0
15	IE15	割り込みイネーブル・ビット 15	リード / ライト	0

これらのビットの機能は、次のように要約されます（x は 2 ～ 15 の数字です）。

IEx	説明
0	割り込みベクタ x に関連付けられている割り込みはディスエーブル
1	割り込みベクタ x に関連付けられている割り込みはイネーブル

2.8.4 デバッグ割り込みイネーブル・レジスタ（DBIER0、DBIER1）

16 ビットのデバッグ割り込みイネーブル・レジスタ DBIER1 と DBIER0 は、CPU がデバッガのリアルタイム・エミュレーション・モードで停止している場合のみ使用されます。CPU がリアルタイム・モード状態で動作している場合、標準の割り込み処理プロセスが使用され、DBIER は無視されます。

DBIER 内でイネーブルのマスカブル割り込みは、タイム・クリティカル割り込みと定義されます。CPU がリアルタイム・モード状態で停止する場合、処理される割り込みは、割り込みイネーブル・レジスタ（IER1 または IER0）内でもイネーブルになっているタイム・クリティカル割り込みだけです。

タイム・クリティカル割り込みを識別するには、DBIER をリードします。タイム・クリティカル割り込みをイネーブルまたはディスエーブルにするには、DBIER にライトします。割り込みをイネーブルにするには、対応するビットをセットします。割り込みをディスエーブルにするには、対応するビットをクリアします。一般的な C55x の DBIER を図 2-14 に示します。これらのビットにマップされる割り込みを確認するには、対応する C55x DSP のデータ・マニュアルを参照してください。

注 :

1) DBIER1 と DBIER0 は、ソフトウェア・リセット命令の影響を受けません。リアルタイム・エミュレーション・モードを使用する前にこれらのレジスタを初期化しなければなりません。
2) DSP ハードウェア・リセットによってすべての DBIER ビットがクリアされ、すべてのタイム・クリティカル割り込みがディスエーブルになります。

図 2-14. デバッグ割り込みイネーブル・レジスタ

DBIER1

15				11	10	9	8
予約済み					RTOSINTD	DLOGINTD	BERRINTD
R- 0					R/W- 0	R/W- 0	R/W- 0

7	6	5	4	3	2	1	0
DBIE23	DBIE22	DBIE21	DBIE20	DBIE19	DBIE18	DBIE17	DBIE16
R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0

DBIER0

15	14	13	12	11	10	9	8
DBIE15	DBIE14	DBIE13	DBIE12	DBIE11	DBIE10	DBIE9	DBIE8
R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0

7	6	5	4	3	2	1	0
DBIE7	DBIE6	DBIE5	DBIE4	DBIE3	DBIE2	予約済み	
R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R 0	

凡例： R = リード。W = ライト。-*n* = DSP のハードウェア・リセット後の値。

2.8.4.1 DBIER1 の RTOSINTD ビット

ビット	名前	説明	アクセス方法	HW リセット
10	RTOSINTD	リアルタイム・オペレーティング・システム割り込み（RTOSINT）用のデバッグ・イネーブル・ビット	リード / ライト	0

RTOSINTD ビットは、タイム・クリティカル割り込みの RTOSINT をイネーブルまたはディスエーブルにします。

RTOSINTD	説明
0	RTOSINT はディスエーブル。
1	RTOSINT はイネーブル。タイム・クリティカル割り込みとして構成されます。

2.8.4.2 DBIER1 の DLOGINTD ビット

ビット	名前	説明	アクセス方法	HW リセット
9	DLOGINTD	データ・ログ割り込み（DLOGINT）用のデバッグ・イネーブル・ビット	リード / ライト	0

DLOGINTD ビットは、タイム・クリティカル割り込みの DLOGINT をイネーブルまたはディスエーブルにします。

DLOGINTD	説明
0	DLOGINT はディスエーブル。
1	DLOGINT はイネーブル。タイム・クリティカル割り込みとして構成されます。

2.8.4.3 DBIER1 の BERRINTD ビット

ビット	名前	説明	アクセス方法	HW リセット
8	BERRINTD	バス・エラー割り込み（BERRINT）用のデバッグ・イネーブル・ビット	リード / ライト	0

BERRINTD ビットは、タイム・クリティカル割り込みの BERRINT をイネーブルまたはディスエーブルにします。

BERRINTD	説明
0	BERRINT はディスエーブル。
1	BERRINT はイネーブル。タイム・クリティカル割り込みとして構成されます。

2.8.4.4 DBIER1 の DBIE16 ～ DBIE23 ビット

ビット	名前	説明	アクセス方法	HW リセット
0	DBIE16	デバッグ割り込みイネーブル・ビット 16	リード / ライト	0
1	DBIE17	デバッグ割り込みイネーブル・ビット 17	リード / ライト	0
2	DBIE18	デバッグ割り込みイネーブル・ビット 18	リード / ライト	0
3	DBIE19	デバッグ割り込みイネーブル・ビット 19	リード / ライト	0
4	DBIE20	デバッグ割り込みイネーブル・ビット 20	リード / ライト	0
5	DBIE21	デバッグ割り込みイネーブル・ビット 21	リード / ライト	0
6	DBIE22	デバッグ割り込みイネーブル・ビット 22	リード / ライト	0
7	DBIE23	デバッグ割り込みイネーブル・ビット 23	リード / ライト	0

これらのビットの機能は、次のように要約されます（x は 16 ～ 23 の数字です）。

DBIEx	説明
0	割り込みベクタ x に関連付けられている割り込みはディスエーブル。
1	割り込みベクタ x に関連付けられている割り込みはイネーブル。タイム・クリティカル割り込みとして構成されます

2.8.4.5 DBIER0 の DBIE2 ～ DBIE15 ビット

ビット	名前	説明	アクセス方法	HW リセット
2	DBIE2	デバッグ割り込みイネーブル・ビット 2	リード / ライト	0
3	DBIE3	デバッグ割り込みイネーブル・ビット 3	リード / ライト	0
4	DBIE4	デバッグ割り込みイネーブル・ビット 4	リード / ライト	0
5	DBIE5	デバッグ割り込みイネーブル・ビット 5	リード / ライト	0
6	DBIE6	デバッグ割り込みイネーブル・ビット 6	リード / ライト	0
7	DBIE7	デバッグ割り込みイネーブル・ビット 7	リード / ライト	0
8	DBIE8	デバッグ割り込みイネーブル・ビット 8	リード / ライト	0
9	DBIE9	デバッグ割り込みイネーブル・ビット 9	リード / ライト	0
10	DBIE10	デバッグ割り込みイネーブル・ビット 10	リード / ライト	0
11	DBIE11	デバッグ割り込みイネーブル・ビット 11	リード / ライト	0
12	DBIE12	デバッグ割り込みイネーブル・ビット 12	リード / ライト	0
13	DBIE13	デバッグ割り込みイネーブル・ビット 13	リード / ライト	0
14	DBIE14	デバッグ割り込みイネーブル・ビット 14	リード / ライト	0
15	DBIE15	デバッグ割り込みイネーブル・ビット 15	リード / ライト	0

これらのビットの機能は、次のように要約されます（x は 2 ～ 15 の数字です）。

DBIEx	説明
0	割り込みベクタ x に関連付けられている割り込みはディスエーブル。
1	割り込みベクタ x に関連付けられている割り込みはイネーブル。タイム・クリティカル割り込みとして構成されます

2.9 リピート・ループ制御のレジスタ

ここでは、リピート・ループの実行を制御するレジスタについて説明します。シングル・リピート・レジスタは、単一命令を繰り返し実行する場合に使用されます。ブロック・リピート・レジスタは、1 つまたは複数のブロック命令を繰り返し実行する場合に使用されます。

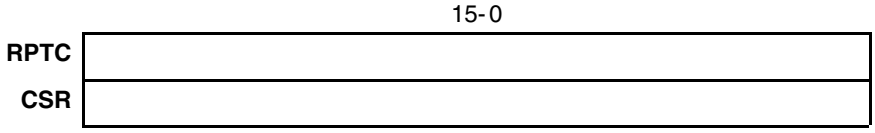
2.9.1 シングル・リピート・レジスタ（RPTC、CSR）

RPTC と CSR の 16 ビット・シングル・リピート命令レジスタは、単一サイクル命令（または並列に実行される 2 つのシングル・サイクル命令）を繰り返し実行します。繰り返し実行回数 N は、最初の実行前にシングル・リピート・カウンタ（RPTC）にロードされます。最初の実行後、命令はさらに N 回実行されるため、合計で N+1 回実行されます。

無条件のシングル・リピート命令の構文の一部では、計算シングル・リピート・レジスタ（CSR) を使用して N 数を指定できます。CSR からの値は、リピートされる命令または命令のペアの最初の実行前に RPTC にコピーされます。

図 2-15 で示されているように、RPTC と CSR には 16 ビットあり、最大 65536 回の連続した命令の実行が可能です（最初の実行と 65535 回の反復)。

図 2-15. シングル・リピート・レジスタ



2.9.2 ブロック・リピート・レジスタ（BRC0、BRC1、BRS1、RSA0、RSA1、REA0、REA1）

ブロック・リピート命令は、命令ブロックを繰り返し実行するループを形成できます。もう一方の中にネストされたループを持つことができ、内側（レベル 1）ループと外側（レベル 0）ループを生成できます。レベル 0 と レベル 1 のループに関連付けられる C55x レジスタについて表 2-13 で説明しています。これらのレジスタの使用は、C54x 互換モード・ビット（C54CM）の影響を受けます（次の説明を参照）。C54x 互換モード・ビットについては、2.10.2.4 項を参照してください。

C54CM = 0：C55x ネイティブ・モードの場合
ループがアクティブのときに割り込みまたは呼び出しが実行されると、CPU はアクティブなリピート・ループを記録します。（2.7 節「プログラム・フロー・レジスタ（PC、RETA、CFCT）」の CFCT の説明を参照）。これにより、サブルーチン内でレベル 0 のリソースの使用が可能になります。CPU がブロック・リピート命令をデコードする場合、ループがすでに実行されているかどうかをまず判別します。CPU がアクティブなレベル 0 のループを検出する場合、レベル 1 のループ・レジスタを使用します。それ以外の場合は、レベル 0 のループ・レジスタを使用します。

C54CM = 1：C54x 互換モードの場合
ブロック・リピート命令は、レベル 0 のループ・レジスタのみをアクティブにします。レベル 1 のループ・レジスタは、使用されません。コンテキスト・セーブ / リストアおよびブロック・リピート・アクティブ・フラグ（BRAF）を使用して、ネストしたブロック・リピート演算を C54x DSP 上に実装できます。ブロック・リピート命令は BRAF をセットし、BRAF は BRC0 に 0（ゼロ）が含まれるブロック・リピート演算の終了時にクリアされます。BRAF の詳細は、2.10.2.2 項「BRAF ビット（ST1_55）」を参照してください。

ブロック・リピート・ループが C54x 互換モード（C54CM = 1）で開始すると、ループが進行中であることを示すために BRAF ビットが自動的にセットされます。使用するプログラムでモードを C54CM = 1 から C54CM = 0 に切り替えなければならない場合は、切り替え前または切り替え中に BRAF ビットをクリアする必要があります。次の 3 つのオプションがあります。

- ループの完了（BRAF が自動的にクリアされる）まで待機し、その後、C54CM をクリアする
- BRAF をクリアし（ループも停止）、その後、C54CM をクリアする
- ステータス・レジスタ ST1_55 を変更する命令を使用して、BRAF と C54CM を同時にクリアする

注：

レベル 0 のループで上述の 3 つの命令を使用する場合、BRC0 にライトしてはいけません。同様に、レベル 0 のループで上述の 3 つの命令を使用する場合も、BRC1 にライトしてはいけません。

表 2-13. ブロック・リピート・レジスタの説明

レベル 0 のループ・レジスタ		レベル 1 のループ・レジスタ（C54CM = 1 の場合は未使用）	
レジスタ	**説明**	**レジスタ**	**説明**
BRC0	ブロック・リピート・カウンタ 0。この 16 ビット・レジスタには、命令ブロックを最初の実行後に繰り返す回数が組み込まれています。	BRC1	ブロック・リピート・カウンタ 1。この 16 ビット・レジスタには、命令ブロックを最初の実行後に繰り返す回数が組み込まれています。
RSA0	ブロック・リピート・スタート・アドレス・レジスタ 0。この 24 ビット・レジスタには、命令ブロック内の最初の命令のアドレスが組み込まれています。	RSA1	ブロック・リピート・スタート・アドレス・レジスタ 1。この 24 ビット・レジスタには、命令ブロック内の最初の命令のアドレスが組み込まれています。
REA0	ブロック・リピート・エンド・アドレス・レジスタ 0。この 24 ビット・レジスタには、命令ブロック内の最後の命令のアドレスが組み込まれています。	REA1	ブロック・リピート・エンド・アドレス・レジスタ 1。この 24 ビット・レジスタには、命令ブロック内の最後の命令のアドレスが組み込まれています。
		BRS1	BRC1 保存レジスタ。BRC1 がロードされるたびに、BRS1 に同じ値がロードされます。レベル 1 のループ実行時、BRS1 の内容は変更されません。レベル 1 のループがトリガされるたびに、BRC1 が BRS1 から再度初期化されます。この機能は、レベル 0 ループの外で BRC1 の初期化を可能にし、それぞれの繰り返しに必要な時間が短縮されます。

注： 24 ビットのレジスタ値が、2 つの連続した 16 ビット・ロケーションに格納されます。ビット 23 ～ 16 は、下位アドレスに格納されます（このロケーションの 8 つの最上位ビットは CPU に無視されます）。ビット 15 ～ 0 は、上位アドレスに格納されます。たとえば、RSA0（23 ～ 16）はアドレス 00 003Ch でアクセス可能で、RSA0（15 ～ 0）はアドレス 00 003Dh でアクセス可能です。

2.10 ステータス・レジスタ（ST0_55 ～ ST3_55）

これらの 4 つの 16 ビット・レジスタ（図 2-16 を参照）は、制御ビットとフラグ・ビットを含みます。制御ビットは C55x DSP の演算に作用し、フラグ・ビットは DSP の現行ステータスを反映するか、または演算結果を示します。

ST0_55、ST1_55、および ST3_55 は、2 つのアドレスでそれぞれアクセス可能です（2.2 節「メモリ・マップド・レジスタ」を参照）。そのうち 1 つのアドレスでは、すべての TMS320C55x ビットが使用可能です。もう 1 つのアドレス（保護アドレス）では、図 2-16 で強調表示されているビットは変更できません。保護アドレスが用意されているのは、ST0、ST1、および PMST（ST3_55 の C54x に相当する機能）にアクセスするために記述された TMS320C54x コードをサポートするためです。予約済みのビットは使用できません。

注：

1) ST3_55 **のビット** 11 ～ 8 **には常に** 1100b (Ch) **をライトしてください。**

2) **一部の** C55x **デバイスには、命令キャッシュ機能がありません。このようなデバイスは、**CAFRZ**、**CAEN**、および** CACLR **の各ビットを使用しません。**

図 2-16. ステータス・レジスタ

ST0_55

15	14	13	12	11	10	9
ACOV2†	**ACOV3**†	**TC1**†	TC2	CARRY	ACOV0	ACOV1
R/W- 0	R/W- 0	R/W- 1	R/W- 1	R/W- 1	R/W- 0	R/W- 0

8	7	6	5	4	3	2	1	0
DP[15:7]								
R/W- 0								

ST1_55

15	14	13	12	11	10	9	8
BRAF	CPL	XF	HM	INTM	**M40**†	SATD	SXMD
R/W- 0	R/W- 0	R/W- 1	R/W- 0	R/W- 1	R/W- 0	R/W- 0	R/W- 1

7	6	5	4	3	2	1	0
C16	FRCT	**C54CM**†	ASM				
R/W- 0	R/W- 0	R/W- 1	R/W- 0				

ST2_55

15	14	13	12	11	10	9	8
ARMS	予約済み		DBGM	EALLOW	RDM	予約済み	CDPLC
R/W- 0	R -11b		R/W- 1	R/W- 0	R/W- 0	R -0	R/W- 0

7	6	5	4	3	2	1	0
AR7LC	AR6LC	AR5LC	AR4LC	AR3LC	AR2LC	AR1LC	AR0LC
R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0	R/W- 0

ST3_55

15	14	13	12	11	10	9	8
CAFRZ†#	**CAEN**†#	**CACLR**†#	**HINT**†‡	予約済み（常に 1100b を書き込む）			
R/W- 0	R/W- 0	R/W- 0	R/W- 1	R/W- 1100b			

7	6	5	4	3	2	1	0
CBERR†	MPNMC§	**SATA**†	予約済み	予約済み	CLKOFF	SMUL	SST
R/W- 0	R/W- ピン	R/W- 0	R/W- 0¶	R- 0	R/W- 0	R/W- 0	R/W- 0

凡例： R = リード。W = ライト。-n = DSP のハードウェア・リセット後の値。

† 強調表示されたビット：ステータス・レジスタの保護アドレスにライトする場合、このビットへのライトによる影響はありません。また、このビットは、リード演算時に常に 0（ゼロ）として表示されます。

‡ HINT ビットは、C55x ホスト・ポート・インターフェイス（HPI）には使用されません。特定の C55x DSP 資料を確認してください。

§ MPNMC のリセット値は、リセット時に事前定義されたピンの状態によって異なることがあります。特定の C55x DSP に関してリセット値を確認するには、対応するデータ・マニュアルを参照してください。

¶ このビットには常に 0（ゼロ）をライトします。

一部の C55x デバイスには、命令キャッシュ機能がありません。このようなデバイスは、CAFRZ、CAEN、および CACLR の各ビットを使用しません。

2.10.1 ST0_55 のビット

ここでは、ST0_55 のビットをアルファベット順に説明します。

2.10.1.1 ACOV0、ACOV1、ACOV2、ACOV3 ビット（ST0_55）

4 つのアキュムレータには、それぞれ ST0_55 内に独自のオーバーフロー・フラグがあります。

ビット	名前	説明	アクセス方法	HW リセット
9	ACOV1	AC1 オーバーフロー・フラグ	リード / ライト	0
10	ACOV0	AC0 オーバーフロー・フラグ	リード / ライト	0
14	ACOV3	AC3 オーバーフロー・フラグ	リード / ライト	0
15	ACOV2	AC2 オーバーフロー・フラグ	リード / ライト	0

これらのフラグに関するポイントは、次のとおりです。

- オーバーフローの検出は、ST1_55 の M40 ビットによって異なります。
  - M40 = 0：オーバーフローは、ビット 31 の位置で検出されます。
  - M40 = 1：オーバーフローは、ビット 39 の位置で検出されます。

  TMS320C54x コードとの互換性が必要な場合は、M40 = 0 であることを確認してください。
- ACOVx は、オーバーフローが ACx で発生する場合にセットされます。ここでの x は、0、1、2、または 3 です。
- オーバーフローが発生すると、次のイベントのうち 1 つが発生するまで ACOVx がセットされたままになります。
  - DSP のハードウェア・リセットまたはソフトウェア・リセットが実行された場合。
  - CPU が、条件付き分岐、コール、復帰、または ACOVx の状態をテストする命令を実行した場合。
  - ACOVx は、ステータス・ビットのクリア命令により明示的にクリアされた場合。たとえば、次の命令を使用して ACOV1 をクリアできます。

    `BCLR ACOV1`

    （`ACOV1` をセットするには、`BSET ACOV1` を使用します）

2.10.1.2 CARRY ビット（ST0_55）

ビット	名前	説明	アクセス方法	HW リセット
11	CARRY	キャリー・ビット	リード / ライト	1

キャリー・ビットに関する主なポイントは、次のとおりです。

- キャリー / ボローの検出は、ST1_55 の M40 ビットによって異なります。
  - M40 = 0：キャリー / ボローは、ビット 31 の位置で検出されます。
  - M40 = 1：キャリー / ボローは、ビット 39 の位置で検出されます。

TMS320C54x コードとの互換性が必要な場合は、M40 = 0 であることを確認してください。

- ❑ 加算が D ユニットの算術論理演算ユニット（D ユニット ALU）で実行される際、加算によりキャリーが生成される場合は CARRY がセットされ、キャリーが生成されない場合は CARRY がクリアされます。この動作には例外が 1 つあります。次の構文が使用されると（16 ビット分 Smem をシフト）、CARRY がキャリーにセットされますが、キャリーが生成されない場合は何も影響はありません。

  ```
  ADD Smem <<#16, [ACx,] ACy
  ```

- ❑ 減算が D ユニット ALU で実行される際、減算によりボローが生成される場合は CARRY がクリアされ、ボローが生成されない場合は CARRY がセットされます。この動作には例外が 1 つあります。次の構文が使用されると（16 ビット分 Smem をシフト）、CARRY がボローに対してクリアされますが、ボローが生成されない場合は何も影響はありません。

  ```
  SUB Smem <<#16, [ACx,] ACy
  ```

- ❑ CARRY は、論理シフト命令により変更されます。
- ❑ 符号付きシフト命令とローテーション命令では、CARRY を変更するかどうかを選択できます。
- ❑ 次の命令構文は、デスティネーション・レジスタがアキュムレータの場合に特定の計算結果を示すために CARRY を変更します。

MIN [src,] dst	最小比較
MAX [src,] dst	最大比較
ABS [src,] dst	絶対値
NEG [src,] dst	否定

- ❑ CARRY のクリアとセットは、次の命令で行うことができます。

```
BCLR CARRY              ; CARRY をクリア
BSET CARRY              ; CARRY をセット
```

2.10.1.3 DP ビット・フィールド（ST0_55）

ビット	名前	説明	アクセス方法	HW リセット
8 ～ 0	DP	データ・ページ・レジスタ（DP）の 9 つの最上位ビットをコピー	リード / ライト	0

この 9 ビットのフィールドは、TMS320C54x DSP から変換されたコードとの互換性を確保するために用意されています。TMS320C55x DSP には、ST0_55 に依存しないデータ・ページ・ポインタ・レジスタがあります。このデータ・ページ・レジスタのビット 15 ～ 7、つまり DP（15 ～ 7）に加えた変更はすべて、DP ステータス・ビットに反映されます。DP ステータス・ビットに加えた変更はすべて、DP（15 ～ 7）に反映されます。DP 直接アドレッシング・モードのアドレスを生成する場合、CPU はデータ・ページ・レジスタ全体、DP（15 ～ 0）を使用します。DP ステータス・ビットを使用する必要はありません。DP は直接変更できます。

注：

ST0_55 をロードしたいが、データ・ページ・レジスタの内容を変更するアクセスをしたくない場合、ST0_55 の 9 つの 最下位ビット（LSB）を変更しないマスク値を使用して OR 演算または AND 演算を使用します。OR 演算の場合、マスク値の 9 つの LSB に 0（ゼロ）を置きます。AND 演算の場合、マスク値の 9 つの LSB に 1 を置きます。

2.10.1.4 TC1、TC2 ビット（ST0_55）

ビット	名前	説明	アクセス方法	HW リセット
12	TC2	テスト / 制御フラグ 2	リード / ライト	1
13	TC1	テスト / 制御フラグ 1	リード / ライト	1

テスト / 制御ビットの主要な機能は、特定の命令が実行されたときのテストの結果を保持することです。テスト / 制御ビットに関する主なポイントは、次のとおりです。

- テスト / 制御フラグに影響を与えるすべての命令では、TC1 または TC2 のどちらが影響を受けるかを選択できます。
- TCx（x = 1 または 2）または TCx のブーリアン表現は条件付き命令でトリガとして使用できます。
- TC1 と TC2 のクリアおよびセットは、次の命令で行うことができます。

```
BCLR TC1; TC1 をクリア
BSET TC1; TC1 をセット
BCLR TC2; TC2 をクリア
BSET TC2; TC2 をセット
```

2.10.2 ST1_55 のビット

ここでは、ST1_55 のビットをアルファベット順に説明します。

2.10.2.1 ASM ビット・フィールド（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
4 ～ 0	ASM	アキュムレータ・シフト・モード・ビット	リード / ライト	00000b

ASM は TMS320C55x ネイティブな命令には使用されませんが、TMS320C55x DSP 上で動作する TMS320C54x コードをサポートする際に使用します。C54x DSP では、ASM フィールドはアキュムレータ値をシフトする特別な命令に対して符号付きシフト・カウントを供給します。C55x ASM フィールドは、C54x 互換モード（C54CM = 1）で使用されます。

以降の説明では、ASM（ステータス・レジスタ ST1_55）を含む C55x レジスタが 2 つのアドレスでアクセス可能であることを理解しておくことが重要です。1 つのアドレス 00 0003h が、C55x ネイティブな命令で使用されます。もう 1 つのアドレス 00 0007h は、0007h で ST1 にアクセスする C54x コードをサポートするために用意されています。

C54CM = 1（C54x 互換モード）の場合

- 00 0007h アドレスへのライトにより ASM がロードされるたびに、5 ビットの ASM 値が 16 ビットに符号拡張され、一時レジスタ 2（T2）にライトされます。ステータス・レジスタのビットをクリア / セットする命令はこのビット・フィールドに影響を与えません。C54x 命令が ASM に応じてアキュムレータをシフトするように CPU に要求する場合、CPU は T2 のシフト・カウントを使用します。
- T2 がロードされるたびに、5 つの最下位ビットが ASM にコピーされます。
- T2 は ASM に結合されるため、T2 は汎用データ・レジスタとして使用することはできません。

C54CM = 0 の場合

- ASM は無視されます。アキュムレータ・シフト演算時に、C55x 命令で指定された一時レジスタ（T0、T1、T2、または T3）、または C55x 命令に埋め込まれた定数から CPU がシフト・カウントをリードします。
- T2 は、汎用データ・レジスタとして使用できます。00 0007h アドレスへのライトによる T2 への影響はありません。また、T2 へのライトは ASM へ影響を与えません。

2.10.2.2 BRAF ビット（ST1_55）

ビット	名前	説明	アクセス方法	リセット値
15	BRAF	ブロック・リピート・アクティブ・フラグ	リード / ライト	0

C54CM = 0 の場合
BRAF は使用されません。リピート演算のステータスは、CPU により自動的に保持されます（2.7 節「プログラム・フロー・レジスタ（PC、RETA、CFCT）」の CFCT の説明を参照）。

C54CM = 1 の場合
BRAF のリードにより、ブロック・リピート演算のステータスが識別されます。

BRAF	ブロック・リピートのアクティビティ
0	ブロック・リピート演算はインアクティブ
1	ブロック・リピート演算はアクティブ

C54x 互換モードでアクティブなブロック・リピート演算を停止するために、次の命令を使用して BRAF をクリアできます。

```
BCLR BRAF  ; BRAF をクリア
```

次の命令を使用して、BRAF をセットできます。

```
BSET BRAF  ; BRAF をセット
```

BRAF のセットとクリアは､ST1_55 を変更する命令を使用して行うこともできます。

BRAF の機能
ブロック・リピート・ループは、RPTB などのブロック・リピート命令で開始します。BRAF は（ループがアクティブであることを示すため）このブロック・リピート命令のアドレス・フェーズでセットされます。

ループの最終命令がパイプラインのデコード・フェーズに入るたびに、CPU は BRAF の値とカウンタ・レジスタ（BRC0）をチェックします。BRAF = 1 かつ BRC0 > 0 の場合、CPU は BRC0 を 1 つデクリメントし、ループの次の繰り返しを開始します。それ以外の場合は、CPU はループを停止します （いずれの場合も、最終命令はパイプラインを通過します）。

BRAF は、次の場合にクリアされます。

- ループの最終命令がデコード・フェーズに入り、BRC0 が 0（ゼロ）にデクリメントされます。BRAF は、1 サイクル後に自動的にクリアされます。
- 命令がブロック・リピート・カウンタ BRC0 に 0（ゼロ）をライトします。BRAF は、1 サイクル後に自動的にクリアされます。
- ファー・ブランチ（FB）命令またはファー・コール（FCALL）命令が実行されます。ただし BRAF は、他のコールまたは分岐命令の実行、あるいは INTR 命令または TRAP 命令の実行ではクリアされません。

- ❑ BRAF は、BCLR BRAF 命令またはステータス・レジスタ ST1_55 を変更する命令により手動でクリアされます。

BRAF は、割り込み命令と割り込みからの復帰命令によるコンテキストの切り替え時に、ST1_55 とともに保存および復元されます。CPU がコール命令に応答する場合、BRAF は保存されません。

ブロック・リピート・ループが進行中で、使用するプログラムでモードを C54CM = 1 から C54CM = 0 に切り替えなければならない場合は、BRAF ビットは切り替え前または切り替え中にクリアする必要があります。次の 3 つのオプションがあります。

- ❑ ループの完了（BRAF が自動的にクリアされる）まで待機し、その後、C54CM をクリアする。
- ❑ BRAF をクリアし（ループも停止）、その後、C54CM をクリアする。
- ❑ ST1_55 を変更する命令を使用して、BRAF と C54CM を同時にクリアする。

パイプラインの考慮点

すでに説明してきたように、BRC0 をクリアする命令の実行の 1 サイクル後に CPU は BRAF をクリアします。この BRAF の変更は、パイプラインによる保護はされません。他の命令が BRAF をリードする前に BRAF を確実に変更するために、BRC0 をクリアする命令と BRAF をリードする命令の間に複数の命令を挿入しなければならない場合があります。次に例を示します。

```
MOV #0, mmap(BRC0)          ; BRC0 をクリア
NOP                         ; BRAF のクリアを待機
NOP
NOP
MOV mmap(ST1_55),AR0        ; ST1_55 をリードする（BRAF を含む）。
```

挿入する命令数は、最初の命令が BRC0 をクリアする段階により異なります。

BRC0 がクリアされるパイプライン・フェーズ†	挿入する命令数
アドレス（AD）フェーズ	0
実行（X）フェーズ	2
ライト（W）フェーズ	3

† 特定の構文のアクティブ・パイプライン・フェーズについては、命令セットの資料を参照してください。

このパイプラインの実行は、ループが終了するときにも影響します。ループの最終命令がデコード・フェーズに到達する前に BRAF を確実に変更するには、BRAF をクリアする命令と最終命令の間に 5 または 6 サイクル挿入する必要があります。

BRAF が変更されるパイプライン・フェーズ†	挿入する命令数
実行（X）フェーズ	5
ライト（W）フェーズ	6

† 特定の構文のアクティブ・パイプライン・フェーズについては、命令セットの資料を参照してください。

リターン命令（RET または RETI）の前の BRAF の更新は、パイプラインで保護されます。リターン後、次の命令が BRAF をリードする場合には更新された値をリードします。

2.10.2.3 C16 ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
7	C16	デュアル 16 ビット算術モード・ビット	リード / ライト	0

TMS320C54x 互換モード（C54CM = 1）では、BRAF はブロック・リピート演算のステータスの識別および制御を行います。このモードでは、一部の命令の実行は C16 の影響を受けます。このような命令が 1 つの 32 ビット演算（倍精度算術）で実行されるのか、または 2 つの 16 ビット演算（デュアル 16 ビット算術）で並列に実行されるのかを C16 が判別します。

C54CM = 1 の場合
D ユニットの ALU で実行される算術演算は、C16 によって異なります。

C16	デュアル 16 ビット・モード
0	オフ。C16 の影響を受ける命令に対して、D ユニットの ALU は 1 つの 32 ビット演算を実行します。
1	オン。C16 の影響を受ける命令に対して、D ユニットの ALU は 2 つの 16 ビット演算を並列に実行します。

C54CM = 0 の場合
CPU は C16 を無視します。命令構文だけでデュアル 16 ビット算術が使用されるのか、32 ビット算術が使用されるのかを判別します。

C16 のクリアとセットは、次の命令で行うことができます。

```
BCLR C16   ; C16 をクリア
BSET C16   ; C16 をセット
```

2.10.2.4 C54CM ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
5	C54CM	TMS320C54x 互換モード・ビット	リード / ライト	1

C54CM ビットは、TMS320C54x DSP 用に開発されたコードを CPU がサポートするかどうかを判別します。

C54CM	C54x 互換モード
0	ディスエーブル。CPU は、TMS320C55x (C55x) DSP 用に書かれたコードをサポートします。
1	イネーブル。このモードは、もとは TMS320C54x (C54x) DSP 用に開発されたコードを使用している場合、セットされなければなりません。すべての C55x CPU リソースは使用できるため、コードを変換すると、C55x に追加された機能を利用してコードを最適化できます。

次の命令とアセンブラのディレクティブを使用して、モードを変更します。

```
BCLR C54CM                ; C54CM をクリア（実行時に発生）
.C54CM_off                ; アセンブラに C54CM = 0 を通知

BSET C54CM                ; C54CM をセット（実行時に発生）
.C54CM_on                 ; アセンブラに C54CM = 1 を通知
```

次の例のように、ブロック・リピート・ループ内の C54CM を変更しないでください。

```
     RPBTLOCALend1        ; ループ 1 の開始
          :
     BSET C54CM
          :
end1 MOV AC0, dbl(*AR3+) ; ループ 1 の終了
```

また、次のようなブロック・リピート命令と並行して C54CM を変更しないでください。

```
     BCLR C54CM || RPTB end2  ; ループ 2 の開始
          :
end2 MOV AC1, dbl(*AR4+)      ; ループ 2 の終了
```

2.10.2.5 CPL ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
14	CPL	コンパイラ・モード・ビット	リード / ライト	0

CPL ビットは、2 つの直接アドレッシング・モードのうちどちらがアクティブかを判別します。

CPL	選択される直接アドレッシング・モード
0	DP 直接アドレッシング・モード。データ空間への直接アクセスは、データ・ページ・レジスタ（DP）に関連して行われます。
1	SP 直接アドレッシング・モード。データ空間への直接アクセスは、データ・スタック・ポインタ（SP）に関連して行われます。DSP はコンパイラ・モードにあると見なされます。

注： I/O 空間への直接アクセスは必ず、ペリフェラル・データ・ページ・レジスタ（PDP）に関連して行われます。

次の命令とアセンブラのディレクティブを使用して、モードを変更します。

```
BCLR CPL   ; CPL をクリア（実行時に発生）
.CPL_off   ; アセンブラに CPL = 0 を通知
BSET CPL   ; CPL をセット（実行時に発生）
.CPL_on    ; アセンブラに CPL = 1 を通知
```

2.10.2.6 FRCT ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
6	FRCT	フラクション・モード・ビット	リード / ライト	0

FRCT ビットは、フラクション・モードをオンまたはオフに切り替えます。

FRCT	フラクション・モード
0	オフ。乗算演算の結果はシフトされません。
1	オン。乗算演算の結果は、小数点の調整のために 1 ビット左にシフトされます。これは、2 つの符号付き Q15 の値を乗算し、Q31 の結果が必要な場合、要求されます。

FRCT のクリアとセットは、次の命令で行うことができます。

```
BCLR FRCT  ; FRCT をクリア
BSET FRCT  ; FRCT をセット
```

2.10.2.7 HM ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
12	HM	ホールド・モード・ビット	リード / ライト	0

DSP の外部メモリ・インターフェイス（EMIF）が HOLD 要求を受信すると、DSP は EMIF 出力ピンをハイ・インピーダンス状態に置きます。HM に応じて、DSP は内部プログラムの実行を停止する場合があります。

HM	保持モード
0	DSP は、内部プログラム・メモリからの命令の実行を続行します。
1	DSP は、内部プログラム・メモリからの命令の実行を停止します。

HM のクリアとセットは、次の命令で行うことができます。

```
BCLR HM    ; HM をクリア
BSET HM    ; HM をセット
```

2.10.2.8 INTM ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
11	INTM	割り込みモード・ビット	リード / ライト	1

INTM ビットは、マスカブル割り込みをグローバルにイネーブルまたはディスエーブルにします（次の一覧を参照）。このビットはノンマスカブル割り込み（ソフトウェアでブロックできない割り込み）には影響を与えません。

INTM	説明
0	すべてのマスクなし割り込みがイネーブルになります。
1	すべてのマスカブル割り込みがディスエーブルになります。

INTM ビットに関する主なポイントは、次のとおりです。

- ステータス・ビットのクリア命令とセット命令を使用して、INTM ビットを変更します。

```
BCLR INTM ; INTM をクリア
BSET INTM ; INTM をセット
```

INTM に影響を与える他の命令は、ソフトウェア割り込み命令とソフトウェア・リセット命令だけです。これらは、割り込みサービス・ルーチンに分岐する前に INTM をセットします。

リビジョン 2.2 より前の CPU コアでは、INTM ビットの更新と割り込みのジャミングとの間にはハードウェアによるパイプライン保護はありません。INTM ビットがパイプラインの実行フェーズで更新されるため、割り込みをグローバルにディスエーブルにする INTM セット命令に続く 5 命令の間に割り込みが取り込まれます。リビジョン 2.2 以降の CPU コアでは、INTM セット命令の後に割り込みを取り込みません。

- CPU が割り込み要求を承認すると、INTM ビットの状態が自動的に保存されます。特に、CPU が ST1_55 をデータ・スタックに保存すると、INTM ビットが保存されます。
- INTR #k5 命令、RESET 命令、またはハードウェア割り込みソースによりトリガされる割り込みサービス・ルーチン（ISR）を実行する前に、CPU はマスカブル割り込みをグローバルにディスエーブルにするために INTM ビットを自動的にセットします。TRAP #k5 命令は INTM ビットには影響を与えません。ISR は、INTM ビットをクリアしてマスカブル割り込みを再度イネーブルにすることができます。
- 割り込みからの復帰命令は、INTM ビットをデータ・スタックから復元します。
- CPU がデバッガのリアルタイム・エミュレーション・モードで停止している場合、INTM は無視され、タイム・クリティカル割り込みだけが処理されます（2.8.4 項「デバッグ割り込みイネーブル・レジスタ（DBIER0、DBIER1）」の説明を参照）。

2.10.2.9 M40 ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
10	M40	D ユニットの計算モード・ビット	リード / ライト	0

M40 ビットは、D ユニットの 2 つの計算モードから 1 つを選択します。

M40 D ユニット計算モード

0 32 ビット・モード。このモードでは、次のようになります。

- ❏ 符号ビットは、ビット 31 の位置から抽出されます。
- ❏ 算術時、キャリーは、ビット 31 の位置で判別されます。
- ❏ オーバーフローは、ビット 31 の位置で検出されます。
- ❏ 飽和時、飽和値は 00 7FFF FFFFh（正のオーバーフロー）または FF 8000 0000h（負のオーバーフロー）になります。
- ❏ 0（ゼロ）に対するアキュムレータの比較は、ビット 31 ～ 0 を使用して行われます。
- ❏ シフト演算またはローテート演算は、32 ビット値で実行されます。
- ❏ アキュムレータの左シフトまたはローテーション時に、シフトアウトされたビットはビット 31 の位置から抽出されます。
- ❏ アキュムレータの右シフトまたはローテーション時に、シフトインされたビットはビット 31 の位置に挿入されます。
- ❏ アキュムレータの符号付きシフト時、SXMD = 0 の場合は 0（ゼロ）がアキュムレータの保護ビットにコピーされ、SXMD = 1 の場合はビット 31 がアキュムレータの保護ビットにコピーされます。アキュムレータのローテーションまたは論理シフト時、デスティネーション・アキュムレータの保護ビットがクリアされます。

注： TMS320C54x 互換モード（C54CM = 1）では、例外がいくつかあります。アキュムレータの符号ビットが、ビット 39 の位置から抽出されます。0（ゼロ）に対するアキュムレータの比較が、ビット 39 ～ 0 を使用して行われます。符号付きシフトは、M40 = 1 のように実行されます。

1 40 ビット・モード。このモードでは、次のようになります。

- ❏ 符号ビットは、ビット 39 の位置から抽出されます。
- ❏ 算術時、キャリーは、ビット 39 の位置で判別されます。
- ❏ オーバーフローは、ビット 39 の位置で検出されます。
- ❏ 飽和時、飽和値は 7F FFFF FFFFh（正のオーバーフロー）または 80 0000 0000h（負のオーバーフロー）になります。
- ❏ 0（ゼロ）に対するアキュムレータの比較は、ビット 39 ～ 0 を使用して行われます。
- ❏ シフト演算またはローテート演算は、40 ビット値で実行されます。
- ❏ アキュムレータの左シフトまたはローテーション時に、シフトアウトされたビットがビット 39 の位置から抽出されます。
- ❏ アキュムレータの右シフトまたはローテーション時に、シフトインされたビットがビット 39 の位置に挿入されます。

M40 のクリアとセットは、次の命令で行うことができます。

```
BCLR M40   ; M40 をクリア
BSET M40   ; M40 をセット
```

2.10.2.10 SATD ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
9	SATD	D ユニットの飽和モード・ビット	リード / ライト	0

SATD ビットは、CPU が D ユニットのオーバーフロー結果を飽和するかどうかを判別します。

SATD	D ユニットの飽和モード
0	オフ。飽和は実行されません。
1	オン。D ユニットが実行する演算がオーバーフローになった場合、結果が飽和されます。飽和は、M40 ビットの値によって異なります。 M40 = 0　CPU は、結果を 00 7FFF FFFFh（正のオーバーフロー）または FF 8000 0000h（負のオーバーフロー）に飽和します。 M40 = 1　CPU は、結果を 7F FFFF FFFFh（正のオーバーフロー）または 80 0000 0000h（負のオーバーフロー）に飽和します。

TMS320C54x コードとの互換性が必要な場合は、M40 = 0 であることを確認してください。

SATD のクリアとセットは、次の命令で行うことができます。

```
BCLR SATD  ; SATD をクリア
BSET SATD  ; SATD をセット
```

2.10.2.11 SXMD ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
8	SXMD	D ユニットの符号拡張モード・ビット	リード / ライト	1

SXMD ビットは、符号拡張モードをオンまたはオフに切り替えます。これは、アキュムレータのロード、および D ユニットで実行される加算、減算、符号付きシフト演算に影響を与えます。SXMD による影響は、次のとおりです。

SXMD　符号拡張モード

0　オフ。符号拡張モードがオフの場合、次のようになります。

- ❏ 40 ビット演算の場合、16 ビット以下のオペランドは 40 ビットにゼロ拡張されます。
- ❏ 条件付き減算命令の場合、16 ビット除数が目的の結果を生成します。
- ❏ D ユニットの算術論理演算ユニット（ALU）がデュアル 16 ビット・モードでローカルに構成されている場合（デュアル 16 ビット算術命令を使用）は、次のようになります。
  - ■ D ユニットの ALU の上位部分で使用される 16 ビット値は、24 ビットにゼロ拡張されます。
  - ■ 2 つに分割された 16 ビット・アキュムレータは両方とも、右にシフトされるとゼロ拡張されます。
- ❏ アキュムレータの符号付きシフトを行う際に、32 ビット演算（M40 = 0）であれば、0（ゼロ）がアキュムレータの保護ビット（39 ～ 32）にコピーされます。
- ❏ アキュムレータの符号付き右シフトを行う際に、シフトされた値がゼロ拡張されます。

1　オン。このモードでは、次のようになります。

- ❏ 40 ビット演算の場合、16 ビット以下のオペランドは 40 ビットに符号拡張されます。
- ❏ 条件付き減算命令の場合、16 ビット除数は正の値でなければなりません（最上位ビット（MSB）は 0（ゼロ）でなければなりません）。
- ❏ D ユニットの ALU がデュアル 16 ビット・モードでローカルに構成されている場合（デュアル 16 ビット算術命令を使用）は、次のようになります。
  - ■ D ユニットの ALU の上位部分で使用される 16 ビット値は、24 ビットに符号拡張されます。
  - ■ 2 つに分割された 16 ビット・アキュムレータは両方とも、右にシフトされると符号拡張されます。
- ❏ アキュムレータの符号付きシフトを行う際に、32 ビット演算（M40 = 0）であれば、ビット 31 がアキュムレータの保護ビット（39 ～ 32）にコピーされます。
- ❏ アキュムレータの符号付き右シフトを行う際に、シフトされた値が符号拡張されます。ただし、アキュムレータ値を符号なしに指定するために uns() オペランド修飾子を使用している場合を除きます。

SXMD は、一部の演算実行時に無視されます。

- ❏ 符号なし演算（ブーリアン論理演算、ローテート演算、論理シフト演算）の場合、入力オペランドは SXMD の値に関係なく常に 40 ビットにゼロ拡張されます。
- ❏ 積和演算ユニット（MAC）内で実行される演算の場合、16 ビットの入力オペランドは SXMD の値に関係なく 17 ビットに符号拡張されます。
- ❏ 命令内のオペランドがオペランド修飾子 uns() で囲まれている場合、オペランドは SXMD の値に関係なく符号なしとして扱われます。

SXMD のクリアとセットは、次の命令で行うことができます。

```
BCLR SXMD ; SXMD をクリア
BSET SXMD ; SXMD をセット
```

2.10.2.12 XF ビット（ST1_55）

ビット	名前	説明	アクセス方法	HW リセット
13	XF	外部フラグ	リード / ライト	1

XF ビットは、汎用出力ビットです。このビットは、XF ピンがある C55x デバイス上の XF ピンに直接接続されます。XF ビットをセットすると、XF ピンがハイにドライブされます。XF ビットをクリアすると、XF ピンがローにドライブされます。XF のクリアとセットは、次の命令で行うことができます。

```
BCLR XF    ; XF をクリア
BSET XF    ; XF をセット
```

2.10.3 ST2_55 のビット

ここでは、ST2_55 のビットをアルファベット順に説明します。

2.10.3.1 AR0LC ～ AR7LC ビット（ST2_55）

CPU には、AR0 ～ AR7 の 8 つの補助レジスタがあります。各補助レジスタ ARn（n = 0、1、2、3、4、5、6、または 7）には、それぞれ ST2_55 にリニア / サーキュラ構成ビットがあります。

ビット	名前	説明	アクセス方法	HW リセット
n	ARnLC	ARn リニア / サーキュラ構成ビット	リード / ライト	0

各 ARnLC ビットは、ARn をリニア・アドレッシングまたはサーキュラ・アドレッシングに使用するかどうかを判別します。

ARnLC	ARn の使用
0	リニア・アドレッシング
1	サーキュラ・アドレッシング

たとえば、AR3LC = 0 の場合は AR3 がリニア・アドレッシングに使用され、AR3LC = 1 の場合は AR3 がサーキュラ・アドレッシングに使用されます。

ステータス・ビットのセット / クリア命令を使用して、ARnLC ビットをクリアおよびセットできます。たとえば、次の命令が、それぞれ AR3LC をクリアおよびセットします。他の ARnLC ビットを変更するためには、3 と表記されている部分を目的の数字に置き換えてください。

```
BCLR AR3LC ; AR3LC をクリア
BSET AR3LC ; AR3LC をセット
```

2.10.3.2 ARMS ビット（ST2_55）

ビット	名前	説明	アクセス方法	HW リセット
15	ARMS	AR モードの切り替え	リード / ライト	0

ARMS ビットは、AR 間接アドレッシング・モードに使用する CPU モードを判別します。

ARMS	使用可能な AR 間接オペランド
0	DSP モード・オペランド。DSP 集約型アプリケーションを効率的に実行します。これらのオペランドには、ポインタとの加算または減算の際に、リバース・キャリー伝搬を使用するオペランドがあります。short 型のオフセットのオペランドは使用できません。
1	制御モード・オペランド。制御システム・アプリケーションに最適なコード・サイズを可能にします。short 型のオペランド *ARn(short(#k3)) が使用可能です。ただし、他のオフセットでは命令に 2 バイトの拡張が必要です。これらの拡張をもつ命令は、他の命令と並行して実行することはできません。

モードを変更するには、次の命令とアセンブラのディレクティブを使用します。

```
BCLR ARMS  ; ARMS をクリアする（実行時に発生）
.ARMS_off  ; アセンブラに ARMS = 0 を通知
BSET ARMS  ; ARMS をセットする（実行時に発生）
.ARMS_on   ; アセンブラに ARMS = 1 を通知
```

2.10.3.3 CDPLC ビット（ST2_55）

ビット	名前	説明	アクセス方法	HW リセット
8	CDPLC	CDP リニア / サーキュラ構成ビット	リード / ライト	0

CDPLC ビットは、係数データ・ポインタ（CDP）をリニア・アドレッシングまたはサーキュラ・アドレッシングに使用するかどうかを判別します。

CDPLC	CDP の使用
0	リニア・アドレッシング
1	サーキュラ・アドレッシング

CDPLC のクリアとセットは、次の命令で行うことができます。

```
BCLR CDPLC ; CDPLC をクリア
BSET CDPLC ; CDPLC をセット
```

2.10.3.4 DBGM ビット（ST2_55）

ビット	名前	説明	アクセス方法	HW リセット
12	DBGM	デバッグ・モード・ビット	リード / ライト	1

DBGM ビットは、プログラムのタイム・クリティカルな部分でデバッグ・イベントをブロックする機能を供給します。

DBGM	デバッグ・イベント
0	イネーブル。
1	ディスエーブル。エミュレータは、メモリまたはレジスタにアクセスできません。ソフトウェア・ブレークポイントは CPU を停止させますが、ハードウェア・ブレークポイントまたは停止要求は無視されます。

DBGM ビットに関する主なポイントは、次のとおりです。

- パイプライン保護のため、DBGM ビットはステータス・ビットのクリア命令とセット命令でのみ変更できます。

```
BCLR DBGM ; DBGM をクリアする
BSET DBGM ; DBGM をセットする
```

ST2_55 へのライトは DBGM ビットには影響を与えません。

- CPU が割り込み要求を承認するか、INTR #k5、TRAP #k5、または RESET の各命令をフェッチすると、DBGM ビットの状態が自動的に保存されます。特に、CPU が ST2_55 をデータ・スタックに保存する場合に、DBGM ビットは保存されます。
- INTR #k5、TRAP #k5、RESET の各命令、またはハードウェア割り込みソースによりトリガされる割り込みサービス・ルーチン（ISR）を実行する前に、CPU はデバッグ・イベントをディスエーブルにするために DBGM ビットを自動的にセットします。ISR は、DBGM ビットをクリアしてデバッグ・イベントを再度イネーブルにすることができます。
- 割り込みからの復帰命令は、DBGM ビットをデータ・スタックから復元します。

2.10.3.5 EALLOW ビット（ST2_55）

ビット	名前	説明	アクセス方法	HW リセット
11	EALLOW	エミュレーション・アクセス・イネーブル・ビット	リード / ライト	0

EALLOW ビットは、非 CPU エミュレーション・レジスタへのライト・アクセスをイネーブルまたはディスエーブルにします。

EALLOW	非エミュレーション・レジスタへのライト・アクセス
0	ディスエーブル
1	イネーブル

EALLOW ビットに関する主なポイントは、次のとおりです。

- ❏ CPU が割り込み要求を承認するか、INTR #k5、TRAP #k5、または RESET の各命令をフェッチすると、EALLOW ビットの状態が自動的に保存されます。特に、CPU が ST2_55 をデータ・スタックに保存する場合に、EALLOW ビットは保存されます。
- ❏ INTR #k5、TRAP #k5、RESET の各命令、またはハードウェア割り込みソースによりトリガされる割り込みサービス・ルーチン（ISR）を実行する前に、CPU は EALLOW ビットを自動的にクリアし、エミュレーション・レジスタへのアクセスを防ぎます。ISR は、EALLOW ビットをセットしてアクセスを再度イネーブルにできます。

  ```
  BSET EALLOW
  ```

  （EALLOW をクリアするために、BCLR EALLOW を使用できます。）
- ❏ 割り込みからの復帰命令は、EALLOW ビットをデータ・スタックから復元します。

2.10.3.6 RDM ビット（ST2_55）

ビット	名前	説明	アクセス方法	HW リセット
10	RDM	丸めモード・ビット	リード / ライト	0

D ユニットで実行される特定の命令は、オペランドを丸めるかどうかを指示できます。実行される丸めのタイプは、RDM ビットの値によって異なります。

RDM	選択される丸めモード
0	無限に丸め。CPU は、8000h（2 の 15 乗）を 40 ビット・オペランドに加算します。その後、CPU は、ビット 15 ～ 0 をクリアし、丸められた結果を 24 ビットまたは 16 ビット表示で生成します。24 ビット表示では、結果のビット 39 ～ 16 だけが有効です。16 ビット表示では、結果のビット 31 ～ 16 だけが有効です。

1	最も近い値に丸め。丸めは、次の IF 文で示されるように 40 ビット・オペランドのビット 15 ～ 0 によって異なります。丸められた結果は、24 ビット表示（ビット 39 ～ 16）または 16 ビット表示（ビット 31 ～ 16）になります。
	If (0 =< ビット 15 ～ 0 < 8000h)
	CPU はビット 15 ～ 0 をクリア
	If (8000h < ビット 15 ～ 0 < 10000h)
	CPU は 8000h をオペランドに加算し、ビット 15 ～ 0 をクリア
	If (ビット 15 ～ 0 == 8000h)
	ビット 31 ～ 16 に奇数値がある場合
	CPU は 8000h をオペランドに加算し、ビット 15 ～ 0 をクリア

TMS320C54x コードとの互換性が必要な場合、RDM = 0 かつ C54CM = 1 であることを確認してください。C54CM = 1（C54x 互換モードがイネーブル）の場合、次の命令は丸め後の結果のビット 15 ～ 0 をクリアしません。

```
SATR [ACx,] ACy                 ; 丸めつき飽和
RND [ACx,] ACy                  ; 丸め
LMS Xmem,Ymem,ACx,ACy           ; 最小 2 乗
```

RDM のクリアとセットは、次の命令で行うことができます。

```
BCLR RDM    ; RDM をクリア
BSET RDM    ; RDM をセット
```

2.10.4 ST3_55 ビット

ここでは、ST3_55 のビットをアルファベット順に説明します。

2.10.4.1 CACLR ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
13	CACLR	キャッシュ・クリア・ビット	リード / ライト	0

命令キャッシュをクリア（またはフラッシュ）する（データ・ラインのすべてのラインを無効にする）には、CACLR ビットをセットします。CACLR をセットするには、次の命令を使用します。

```
BSET CACLR ; CACLR をセット
```

一度セットすると、CACLR はフラッシュ処理が完了するまで 1 のままで、完了時に、CACLR は自動的に 0（ゼロ）にリセットされます。したがって、CACLR をポーリングしてステータスを確認することができます。

CACLR	キャッシュ・クリア・プロセス
0	完了。
1	未完了。すべてのキャッシュ・ブロックが無効です。キャッシュをフラッシュするために必要なサイクル数は、メモリ・アーキテクチャによって異なります。キャッシュがフラッシュされると、命令バッファ・ユニット内のプリフェッチ・キューの内容が自動的にフラッシュされます。

2.10.4.2 CAEN ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
14	CAEN	キャッシュ・イネーブル・ビット	リード / ライト	0

CAEN ビットは、プログラム・キャッシュをイネーブルまたはディスエーブルにします。

CAEN	キャッシュ
0	ディスエーブル。キャッシュ・コントローラは、プログラム要求を受け付けません。すべてのプログラム要求は、デコードされるアドレスに応じて、内部メモリまたは外部メモリのいずれかで処理されます。
1	イネーブル。プログラム・コードは、デコードされるアドレスに応じて、キャッシュ、内部メモリ、または外部メモリからフェッチされます。

重要な点を次に示します。

- CAEN ビットをクリアしてキャッシュがディスエーブルになると、I ユニット内の命令バッファ・キューの内容が自動的にフラッシュされます。
- CAEN のクリアとセットは、次の命令で行うことができます。

```
BCLR CAEN    ; CAEN をクリア
BSET CAEN    ; CAEN をセット
```

2.10.4.3 CAFRZ ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
15	CAFRZ	キャッシュ・フリーズ・ビット	リード / ライト	0

CAFRZ は、プログラム命令キャッシュをロックして、その内容をキャッシュ・ミスで更新せずにキャッシュ・ヒットで使用できるようにします。キャッシュの内容は、CAFRZ がクリアされるまで変更されません。CAFRZ の役割は、次のとおりです。

CAFRZ	説明
0	キャッシュはデフォルトの動作モード
1	キャッシュはフリーズ（キャッシュ内容はロック）

CAFRZ のクリアとセットは、次の命令で行うことができます。

```
BCLR CAFRZ ; CAFRZ をクリア
BSET CAFRZ ; CAFRZ をセット
```

2.10.4.4 CBERR ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
7	CBERR	CPU バス・エラー・フラグ	リード / ライト（0（ゼロ）のみライト可能）	0

CBERR ビットがセットされるのは、内部バス・エラーが検出される場合です。このエラーは、CPU に割り込みフラグ・レジスタ 1 のバス・エラー割り込みフラグ（BERRINTF）をセットさせます。重要な点を次に示します。

- ❏ CBERR ビットに 1 をライトしても影響はありません。このビットが 1 になるのは、内部バス・エラーが発生した場合だけです。
- ❏ バス・エラー割り込み（BERRINT）の割り込みサービス・ルーチンは、割り込みのあったプログラム･コードに制御を戻す前に CBERR ビットをクリアしておく必要があります。

```
BCLR CBERR   ; CBERR をクリア
```

CBERR ビットは、次のように要約できます。

CBERR	説明
0	フラグは、使用するプログラムまたはリセットによりクリアされている。
1	内部バス・エラーが検出されている。

注：

バス・エラーが起きると、エラーを引き起こした命令の機能と並列に実行される命令の機能は保証されません。

2.10.4.5 CLKOFF ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
2	CLKOFF	CLKOUT ディスエーブル・ビット	リード / ライト	0

CLKOFF = 0 の場合、CLKOUT ピンの出力はイネーブルになり、関連するクロック信号がピン上に表示されます。CLKOFF = 1 の場合、CLKOUT ピンはディスエーブルになります。

CLKOFF のクリアとセットは、次の命令で行うことができます。

```
BCLR CLKOFF; CLKOFF をクリア
BSET CLKOFF; CLKOFF をセット
```

2.10.4.6 HINT ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
12	HINT	ホスト割り込みビット	リード / ライト	1

割り込み要求をホスト・ポート・インターフェイスを介してホスト・プロセッサに送るために、HINT ビットを使用します。HINT ビットをクリアし、セットすることにより、アクティブ・ロー割り込みパルスを生成します。

```
BCLR HINT  ; HINT をクリア
BSET HINT  ; HINT をセット
```

注：

HINT ビットは、C55x ホスト・ポート・インターフェイス（HPI）のすべてで使用できるわけではありません。特定の C55x DSP については、その資料を参照してください。

2.10.4.7 MPNMC ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
6	MPNMC	マイクロプロセッサ / マイクロコンピュータ・モード・ビット	リード / ライト	リセット時に事前定義されたピンの状態によって異なることがあります。特定の C55x DSP に関して確認するには、対応するデータ・マニュアルを参照してください。

MPNMC ビットは、オンチップ ROM をイネーブルまたはディスエーブルにします。

MPNMC	モード
0	マイクロコンピュータ・モード。オンチップ ROM はイネーブルです。プログラム空間でアドレス指定可能です。
1	マイクロプロセッサ・モード。オンチップ ROM はディスエーブルです。プログラム空間マップではアドレス指定不可能です。

重要な点を次に示します。

- MPNMC ビットのリセット値は、リセット時に事前定義されたピンの状態によって異なることがあります。特定の C55x DSP に関して確認するには、対応するデータ・マニュアルを参照してください。
- ソフトウェア・リセット命令による MPNMC ビットへの影響はありません。
- MPNMC のクリアとセットは、次の命令を使用して行うことができます。

  ```
  BCLR MPNMC; MPNMC をクリア
  BSET MPNMC; MPNMC をセット
  ```

- MPNMC ビットを変更する命令のすぐ後に分岐命令を配置することはできません。これを行うと、CPU は既存の MPNMC 値を使用して、次の命令を誤ったメモリ・ロケーションからフェッチしてしまうことがあります。MPNMC 更新命令と分岐命令を分離するために必要な最小命令サイクル数は、使用する分岐命令の種類によって異なります。表 2-14 では、分岐命令を 3 つのカテゴリーに分類しています。カテゴリーごとに必要な最小命令サイクル数を表 2-15 に示します。

表 2-14. 分岐命令のカテゴリー

カテゴリー I	カテゴリー II	カテゴリー III
B L7	RET（スロー・リターンが選択される場合）	B ACx
B L16	RETCC（スロー・リターンが選択される場合）	CALL ACx
B P24	RETI（スロー・リターンが選択される場合）	
BCC I4, cond		
BCC L8, cond		
BCC L16, cond		
BCC P24, cond		
CALL L16		
CALL P24		
CALLCC L16, cond		
CALLCC P24, cond		
RET （ファースト・リターンが選択される場合）		
RETCC （ファースト・リターンが選択される場合）		
RETI （ファースト・リターンが選択される場合）		

表 2-15. MPNMC 更新命令と分岐命令の間に必要な最小命令サイクル数

	後続の命令の前に必要なサイクル数		
MPNMC 更新命令	カテゴリー I	カテゴリー II	カテゴリー III
次の命令のうち 1 つの命令 BSET MPNMC BSET k4, ST3_55 BCLR MPNMC BCLR k4, ST3_55	4	0	0
ST3_55 のメモリ・マップド・アドレスへのライト時に MPNMC を変更する命令	5	1	0

MPNMC を変更する BSET 命令の次の例を考慮してください。表 2-14 では、CALL をカテゴリー I の分岐命令として指定しています。BSET MPNMC 命令とカテゴリー I の分岐命令の間には 4 サイクル必要であることを表 2-15 に示します。この例では、4 つの NOP（ノー・オペレーション）命令を挿入して、4 サイクルが指定されています。他の命令をここに挿入することもできます。

```
BSET MPNMC
NOP
NOP
NOP
NOP
CALL #SubroutineA
```

2.10.4.8 SATA ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
5	SATA	A ユニットの飽和モード・ビット	リード / ライト	0

SATA ビットは、CPU が A ユニットの算術論理演算ユニット（A ユニットの ALU）のオーバーフロー結果を飽和するかどうかを判別します。

SATA	A ユニットの飽和モード
0	オフ。飽和は実行されません。
1	オン。A ユニットの ALU の計算がオーバーフローになった場合、その結果は 7FFFh（正の方向のオーバーフロー用）または 8000h（負の方向のオーバーフロー用）に飽和されます。

SATA のクリアとセットは、次の命令で行うことができます。

```
BCLR SATA  ; SATA をクリア
BSET SATA  ; SATA をセット
```

2.10.4.9 SMUL ビット（ST3_55）

ビット	名前	説明	アクセス方法	HW リセット
1	SMUL	飽和乗算モード・ビット	リード / ライト	0

SMUL ビットは、飽和乗算モードをオンまたはオフに切り替えます。

SMUL	飽和乗算モード
0	オフ
1	オン。SMUL = 1、かつ FRCT = 1、かつ SATD = 1 の場合、18000h × 18000h の結果が 7FFF FFFFh に飽和されます（M40 ビットの値に関係なく）。これにより、2 つの負数の積が正数になります。 積和演算 / 積差演算命令の場合、飽和が実行されるのは、乗算の後で、かつ加算 / 減算の前です。

積和演算 / 積差演算命令の場合、飽和が実行されるのは、乗算の後で、かつ加算 / 減算の前です。

SMUL のクリアとセットは、次の命令で行うことができます。

```
BCLR SMUL  ; SMUL をクリア
BSET SMUL  ; SMUL をセット
```

2.10.4.10 SST ビット（ST3_55）

ビット	名前	説明	アクセス方法	リセット値
0	SST	飽和格納モード・ビット	リード / ライト	0

TMS320C54x 互換モード（C54CM = 1）では、一部のアキュムレータ・ストア命令の実行が SST の影響を受けます。SST が 1 の場合、40 ビットのアキュムレータ値がストア演算の前に 32 ビット値に飽和されます。アキュムレータ値がシフトされる場合、CPU がそのシフト後に飽和を実行します。

C54CM = 1 の場合
SST は、飽和ストア・モードのオンまたはオフを切り替えます。

SST	飽和格納モード
0	オフ
1	オン。SST の影響を受ける命令の場合、CPU はシフトされているアキュムレータ値またはシフトされていないアキュムレータ値を飽和してから格納します。飽和は、符号拡張モード・ビット（SXMD）の値によって異なります。 SXMD = 0　40 ビット値が符号なしとして扱われます。40 ビット値が 00 7FFF FFFFh より大きい場合、CPU は 32 ビットの 7FFF FFFFh を生成します。 SXMD = 1　40 ビット値が符号付きとして扱われます。40 ビット値が 00 8000 0000h より小さい場合、CPU は 32 ビットの 8000 0000h を生成します。40 ビット値が 00 7FFF FFFFh より大きい場合、CPU は 7FFF FFFFh を生成します。

C54CM = 0 の場合
CPU は SST を無視します。命令構文だけで飽和が発生するかどうかを判別します。

SST のクリアとセットは、次の命令で行うことができます。

```
BCLR SST   ; SST をクリア
BSET SST   ; SST をセット
```

第 3 章

メモリおよび I/O 空間

C55x DSP は、統合されたデータ / プログラム空間と I/O 空間へのアクセスを提供します。データ空間アドレスは、汎用メモリとメモリ・マップド CPU レジスタにアクセスするために使用されます。プログラム空間アドレスは、メモリから命令をリードするために CPU で使用されます。I/O 空間は、複数のペリフェラルとの双方向通信を行う際に使用できます。オンチップ・ブート・ローダは、内部メモリにコードとデータをロードする際に使用する複数の方法を提供します。

項目 **ページ**

3.1 メモリ・マップ

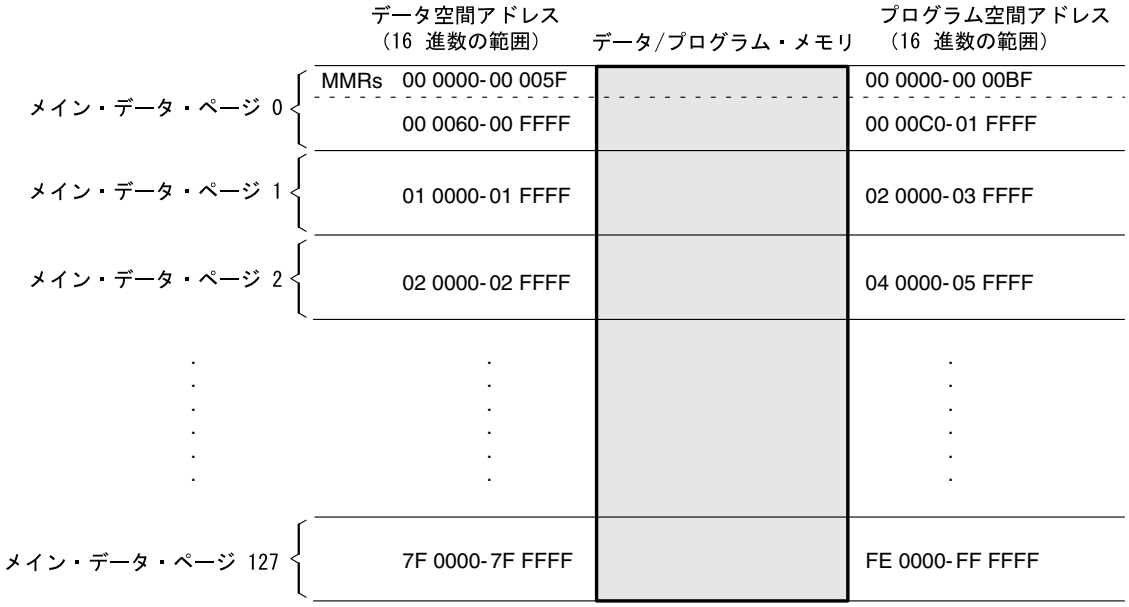
16M バイトのメモリはすべて、プログラム空間またはデータ空間と同様にアドレス指定可能です（図 3-1 を参照）。CPU がプログラム空間を使用してプログラム・コードをメモリからリードする場合、バイトを参照するために 24 ビットのアドレスを使用します。また使用するプログラムがデータ空間にアクセスする場合、16 ビットのワードを参照するために 23 ビットのアドレスを使用します。いずれの場合も、アドレス・バスが 24 ビット値を伝送しますが、データ空間のアクセス時に、アドレス・バス上の最下位ビットが強制的に 0（ゼロ）になります。

データ空間は、128 のメイン・データ・ページに分割されます（0 ～ 127）。各メイン・データ・ページは、64K のアドレスを持っています。メイン・データ・ページを参照する命令は、7 ビットのメイン・データ・ページ値を 16 ビットのオフセットに連結します。

データ・ページ 0 では、最初の 96 アドレス（00 0000h ～ 00 005Fh）がメモリ・マップド・レジスタ（MMR）用に予約済みです。192 のアドレス（00 0000h ～ 00 00BFh）の対応するブロックがプログラム空間にあります。これらのアドレスにプログラム・コードを格納しないことをお勧めします。

アドレスを内部メモリと外部メモリに分割する方法を決定するには、また、内部メモリに関する詳細は、使用する C55x DSP のデータ・マニュアルを参照してください。

図 3-1. メモリ・マップ



3.2 プログラム空間

CPU はプログラム・メモリから命令をリードするときにのみプログラム空間にアクセスします。CPU は異なるサイズ（3.2.2 項を参照）の命令をフェッチするためにバイト・アドレス（3.2.1 項を参照）を使用します。命令フェッチは、偶数アドレスの 32 ビット境界にアラインされます（3.2.3 項を参照）。

3.2.1 バイト・アドレス（24 ビット）

CPU がプログラム・メモリから命令をフェッチする場合、**バイト・アドレス**を使用します。バイト・アドレスは、個々のバイトに割り当てられるアドレスです。これらのアドレスは 24 ビット幅です。32 ビット幅のメモリ行を図 3-2 に示します。各バイトは 1 つのアドレスを割り当てられます。たとえば、バイト 0 はアドレス 00 0100h で、バイト 2 はアドレス 00 0102h です。

図 3-2. 32 ビット幅プログラム・メモリのバイト・アドレス例

バイト・アドレス	バイト 0	バイト 1	バイト 2	バイト 3
00 0100h ～ 00 0103h				

3.2.2 プログラム空間における命令の構成

DSP は、8 ビット、16 ビット、24 ビット、32 ビット、40 ビット、48 ビットの命令をサポートします。命令がプログラム空間でどのように構成されるかを図 3-3 に示します。異なるサイズの 5 つの命令が、32 ビット幅のメモリに格納されています。各命令のアドレスは、その最上位バイトのアドレスです（オペコード）。網掛けされているバイトにはコードは格納されません。

図 3-3. プログラム空間における命令の構成例

命令	サイズ	アドレス
A	24 ビット	00 0101h
B	16 ビット	00 0104h
C	32 ビット	00 0106h
D	8 ビット	00 010Ah
E	24 ビット	00 010Bh

バイト・アドレス	バイト 0	バイト 1	バイト 2	バイト 3
00 0100h 〜 00 0103h		A(23 〜 16)	A(15 〜 8)	A(7 〜 0)
00 0104h 〜 00 0107h	B(15 〜 8)	B(7 〜 0)	C(31 〜 24)	C(23 〜 16)
00 0108h 〜 00 010Bh	C(15 〜 8)	C(7 〜 0)	D(7 〜 0)	E(23 〜 16)
00 010Ch 〜 00 010Fh	E(15 〜 8)	E(7 〜 0)		

3.2.3 プログラム空間からのフェッチのアラインメント

複数の命令をプログラム・メモリに格納する際に、それらの命令をアラインする必要はありませんが、命令フェッチは、偶数アドレスの 32 ビット境界にアラインされます。命令フェッチ時、CPU は 2 つの最下位ビット（LSB）が 0（ゼロ）であるアドレスから 32 ビットをリードします。すなわち、16 進数のフェッチ・アドレスの最下位桁は、常に 0h、4h、8h、または Ch です。

CPU がプログラムの実行を中断すると、プログラム・カウンタ（PC）にライトされるアドレスがフェッチ・アドレスと異なることがあります。PC アドレスとフェッチ・アドレスは、PC アドレスの 2 つの LSB が 0（ゼロ）である場合のみ同じです。サブルーチンを呼び出す、次のアセンブリ・コード・セグメントを考慮してください。

```
CALL #subroutineB
```

サブルーチンの最初の命令がバイト・アドレス 00 0106h の命令 C であると仮定します（図 3-3 を参照）。PC には 00 0106h が含まれますが、プログラム・リード・アドレス・バス（PAB）は、前の 32 ビット境界 00 0104h のバイト・アドレスを伝送します。CPU は、アドレス 00 0104h で始まるコードの 4 バイト・パケットをリードします。命令 C は、最初に実行される命令です。

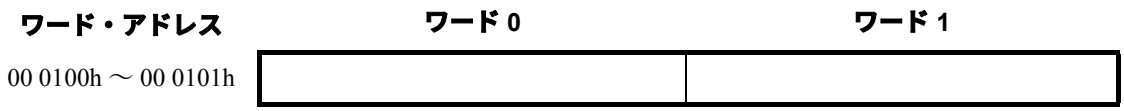
3.3 データ空間

プログラムがメモリまたはレジスタとのリードあるいはライトを行うと、データ空間へのアクセスが行われます。CPU は 8 ビット、16 ビット、または 32 ビットの値（3.3.2 項を参照）をリードまたはライトするためにワード・アドレス（3.3.1 項を参照）を使用します。特定の値に対して生成される必要のあるアドレスは、データ空間のワード境界にどのように格納されるかによって異なります（3.3.3 項を参照）。

3.3.1 ワード・アドレス（23 ビット）

CPU がデータ空間にアクセスする場合、ワード・アドレスを使用します。ワード・アドレスは、個々の 16 ビット・ワードに割り当てられるアドレスです。これらのアドレスは 23 ビット幅です。32 ビット幅のメモリ行を図 3-4 に示します。各ワードは 1 つのアドレスを割り当てられます。ワード 0 はアドレス 00 0100h で、ワード 1 はアドレス 00 0101h です。

図 3-4. 32 ビット幅データ・メモリのワード・アドレス例



アドレス・バスは、24 ビット・バスです。CPU がデータ空間とのリードまたはライトを行うと、23 ビット・アドレスの末尾に 0（ゼロ）を付けて連結されます。たとえば、命令が 23 ビット・アドレス 00 0102h のワードをリードすると仮定します。適切なデータ・リード・アドレス・バスが、24 ビット値 00 0204h を伝送します。

ワード・アドレス : 000 0000 0000 0001 0000 0010

データ・リード・アドレス・バス :0000 0000 0000 0010 0000 0100

3.3.2 データ・タイプ

命令セットは、次のデータ・タイプを処理します。

バイト 8 ビット

ワード 16 ビット

ロング・ワード 32 ビット

専用の命令構文（表 3-1 を参照）は、特定のワードの上位バイトまたは下位バイトを選択できます。バイト・ロード命令は、バイトをリードし、それらをレジスタにロードします。リードされるバイトは、ゼロ拡張（uns() オペランド修飾子が使用される場合）、または符号拡張されてから格納されます。バイト・ストア命令は、レジスタ の 8 つの 最下位ビットをメモリ内の指定のバイトに格納します。

注：

データ空間では、CPU はワードにアクセスするために 23 ビットのアドレスを使用します。バイトにアクセスするためには、CPU はそのバイトを含むワードを操作する必要があります。

表 3-1. バイト・ロード命令およびバイト・ストア命令

演算	命令構文	アクセスされるバイト
バイト・ロード（アキュムレータ、補助レジスタ、または一時レジスタのロード命令）	MOV [uns(]high_byte(Smem)[)], dst	Smem(15 ～ 8)
	MOV [uns(]low_byte(Smem)[)], dst	Smem(7 ～ 0)
	MOV high_byte(Smem) << #SHIFTW, ACx	Smem(15 ～ 8)
	MOV low_byte(Smem) << #SHIFTW, ACx	Smem(7 ～ 0)
バイト・ストア（アキュムレータ、補助レジスタ、または一時レジスタのストア命令）	MOV src, high_byte(Smem)	Smem(15 ～ 8)
	MOV src, low_byte(Smem)	Smem(7 ～ 0)

CPU がロング・ワードにアクセスする場合、アクセスに使用されるアドレスは 32 ビット値の最上位ワード（MSW）のアドレスです。最下位ワード（LSW）のアドレスは、MSW のアドレスによって異なります。

- MSW のアドレスが偶数の場合、LSW は次のアドレスでアクセスされます。次に例を示します。

ワード・アドレス

00 0100h ～ 00 0101h	MSW	LSW

- MSW のアドレスが奇数の場合、LSW は前のアドレスでアクセスされます。次に例を示します。

ワード・アドレス

00 0100h ～ 00 0101h	LSW	MSW

MSW（LSW）のアドレスが与えられると、LSW（MSW）のアドレスを特定するために、その最下位ビットを補数演算します。

3.3.3 データ空間におけるデータの構成

データがデータ空間でどのように構成されるかを図 3-5 に示します。異なるサイズの 7 つのデータ値が、32 ビット幅のメモリに格納されています。アドレス 00 0100h の網掛けされているバイトにはデータ値は格納されていません。この例の重要な点は、次のとおりです。

- ロング・ワードにアクセスするには、その最上位ワード（MSW）を参照する必要があります。C は、アドレス 00 0102h でアクセスされます。D は、アドレス 00 0105h でアクセスされます。
- ワード・アドレスは、データ空間のバイトへのアクセスにも使用されます。たとえば、アドレス 00 0107h は、F（上位バイト）と G（下位バイト）の両方に使用されます。特別のバイト命令が、上位バイトまたは下位バイトがアクセスされるかどうかを識別します。

図 3-5. データ空間におけるデータの構成例

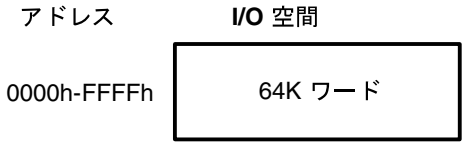
データ値	データ・タイプ	アドレス
A	バイト	00 0100h（下位バイト）
B	ワード	00 0101h
C	ロング・ワード	00 0102h
D	ロング・ワード	00 0105h
E	ワード	00 0106h
F	バイト	00 0107h（上位バイト）
G	バイト	00 0107h（下位バイト）

ワード・アドレス	ワード 0		ワード 1	
00 0100h ～ 00 0101h		A	B	
00 0102h ～ 00 0103h	C の MSW（ビット 31 ～ 16）		C の LSW（ビット 15 ～ 0）	
00 0104h ～ 00 0105h	D の LSW（ビット 15 ～ 0）		D の MSW（ビット 31 ～ 16）	
00 0106h ～ 00 0107h	E		F	G

3.4 I/O 空間

I/O 空間はデータ / プログラム空間と分離されており、DSP のペリフェラルのレジスタにアクセスする場合のみ使用可能です。I/O 空間のワード・アドレスは 16 ビット幅で、64K のロケーションへのアクセスが可能になります（図 3-6 を参照）。

図 3-6. I/O 空間のアドレス・マップ



CPU はリード時にデータ・リード・アドレス・バス DAB を使用し、ライト時にデータ・ライト・アドレス・バス EAB を使用します。CPU が I/O 空間とのリードまたはライトを行う場合、16 ビット・アドレスの先頭に 0（ゼロ）を付けて連結されます。たとえば、命令が 16 ビット・アドレス 0102h のワードをリードすると仮定します。DAB は、24 ビット値 00 0102h を伝送します。

注 :

FFFFh を超えたインクリメントまたは 0000h を超えたデクリメントを行うと I/O アドレスが初期状態に戻ってしまいます。この動作はサポートされていないため使用しないでください。

3.5 ブート・ローダ

オンチップ・ブート・ローダは、起動 / リセット時にコードとデータを外部ソースから C55x DSP 内部の RAM に転送する複数のオプションを提供します。特定の C55x DSP のブート・オプション一覧とふさわしいオプションを選択する方法については、使用する DSP のデータ・マニュアルを参照してください。C55x の hex 変換ユーティリティを使用してブート・ロードを行う方法については、『TMS320C55x Assembly Language Tools User's Guide』（文献番号 SPRU280）を参照してください。

第 4 章

スタック

この章では、各 C55x DSP に搭載されている 2 つのスタックについて説明します。また、この 2 つのスタックがお互いにどのように関係しているかについて、また、自動コンテキスト切り替え（サブルーチンの実行前に重要なレジスタ値を格納し、サブルーチンが完了するとこれらの値を復元する）時に CPU がどのように使用するかについても説明します。

項目 **ページ**

4.1 データ・スタックおよびシステム・スタック

CPU は、データ・スタックとシステム・スタックと呼ばれる 2 つの 16 ビット・ソフトウェア・スタックをサポートします。スタック・ポインタに使用されるレジスタについて図 4-1 と表 4-1 で説明します。データ・スタックにアクセスする場合、CPU は XSP を形成するために SPH と SP を連結します。XSP は、データ・スタックに前回プッシュされた 23 ビット・アドレスの値を含みます。SPH は、7 ビット・メイン・データ・ページのメモリを保持し、SP はそのページの特定のワードを指します。CPU は、値をスタックにプッシュする前に SP をデクリメントし、値をスタックからポップした後に SP をインクリメントします。SPH は、スタック動作時に変更されません。

同様に、システム・スタックにアクセスする場合、CPU は XSSP を形成するために SPH と SSP を連結します。XSSP は、システム・スタックに前回プッシュされたアドレスの値を含みます。CPU は、値をシステム・スタックにプッシュする前に SSP をデクリメントし、値をシステム・スタックからポップした後に SSP をインクリメントします。この場合も、SPH は、スタック動作時に変更されません。

4.2 節「スタック構成」で説明されているように、SSP は、SP にリンクすることも、SP から独立させることもできます。32 ビットのスタック構成を選択する場合、SP を変更する演算が SSP を同様に変更します。デュアル 16 ビットのスタック構成を 1 つ選択する場合、SSP は SP から独立し、SSP は自動コンテキスト切り替え時のみ変更されます（4.4 節を参照）。

図 4-1. 拡張スタック・ポインタ

	22-16	15-0
XSP	SPH	SP
XSSP	SPH	SSP

表 4-1. スタック・ポインタ・レジスタ

レジスタ	名称	アクセス方法
XSP	拡張データ・スタック・ポインタ	専用の命令を使用してのみアクセス可能。XSP は、メモリにマップされるレジスタではありません。
SP	データ・スタック・ポインタ	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。
XSSP	拡張システム・スタック・ポインタ	専用の命令を使用してのみアクセス可能。XSSP は、メモリにマップされるレジスタではありません。
SSP	システム・スタック・ポインタ	専用の命令を使用して、またメモリ・マップド・レジスタとしてアクセス可能。
SPH	XSP と XSSP の上位部分	メモリ・マップド・レジスタとしてアクセス可能。また、XSP または XSSP にアクセスして、SPH にアクセスすることも可能です。SPH にアクセスする専用の命令はありません。 注：SPH は、XSP または XSSP にライトすることによる影響を受けません。

4.2 スタック構成

TMS320C55x DSP では、3 通りのスタック構成が可能です（表 4-2 を参照）。そのうち 1 つはファースト・リターン・プロセスを使用し、他の 2 つはスロー・リターン・プロセスを使用します。これらの 2 つのプロセスの違いについて 4.3 節で説明します。

32 ビットのリセット・ベクタ・ロケーションのビット 29 と 28 に適切な値を設定して、3 通りのスタック構成から 1 つ選択します（表 4-2 を参照）。これは、C55x アセンブリ・コードで .ivec アセンブラ・ディレクティブの一部として行うことができます。このディレクティブについては、『TMS320C55x Assembly Language User's Guide』（文献番号 SPRU280）を参照してください。リセット・ベクタ・ロケーションの 24 の最下位ビットは、リセット割り込みサービス・ルーチン（ISR）のスタート・アドレスでなければなりません。

表 4-2. スタック構成

スタック構成	説明	リセット・ベクタ値† （バイナリ）
デュアル 16 ビット・スタック・ファースト・リターン	データ・スタックとシステム・スタックは独立しています。データ・スタックにアクセスすると、データ・スタック・ポインタ（SP）は変更されますが、システム・スタック・ポインタ（SSP）は変更されません。ファースト・リターンを実現するために、RETA と CFCT のレジスタを使用します（図 4-3 を参照）。	XX00 XXXX:(24 ビット ISR アドレス)
デュアル 16 ビット・スタック・スロー・リターン	データ・スタックとシステム・スタックは独立しています。データ・スタックにアクセスすると、SP は変更されますが、SSP は変更されません。RETA と CFCT のレジスタは使用されません（図 4-2 を参照）。	XX01 XXXX:(24 ビット ISR アドレス)
32 ビット・スタック・スロー・リターン	データ・スタックとシステム・スタックは、単一の 32 ビット・スタックとして機能します。データ・スタックにアクセスすると、SP と SSP が同じインクリメントで変更されます。RETA と CFCT のレジスタは使用されません（図 4-2 を参照）。 注：SP をそのメモリ・マップド・ロケーションを介して変更する場合、SSP は自動的に更新されません。この場合、SSP も変更して 2 つのポインタのアラインを保持する必要があります。	XX10 XXXX:(24 ビット ISR アドレス)
-	予約済み。ビット 29 と 28 の両方を 1 にセットしないでください。	XX11 XXXX:(24 ビット ISR アドレス) (これは不正な値です。)

† X のビットは、0（ゼロ）または 1 です。

4.3 ファースト・リターンとスロー・リターン

ファースト・リターン・プロセスとスロー・リターン・プロセスの違いは、CPU がプログラム・カウンタ（PC）とループ・コンテキスト・レジスタの 2 つの内部レジスタの値を保存し復元する方法です。

PC は、I ユニットでデコードされるコードの 1 ～ 6 バイトの 24 ビット・アドレスを保持します。CPU が割り込みまたは呼び出しを実行すると、現行の PC 値（リターン・アドレス）が格納され、その後 PC に割り込みサービス・ルーチンまたは呼び出されたルーチンのスタート・アドレスがロードされます。CPU がルーチンから復帰するとき、リターン・アドレスが PC に戻され、中断されたプログラム・シーケンスは以前のように続行します。

8 ビットのループ・コンテキスト・レジスタは、アクティブなリピート・ループ（ループ・コンテキスト）を記録します。CPU が割り込みまたはコールを実行すると、現行のループ・コンテキストが格納され、その後サブルーチンの新規コンテキスを作成するために、8 ビット・レジスタがクリアされます。CPU がサブルーチンから復帰するとき、ループ・コンテキストが 8 ビット・レジスタに戻されます。

スロー・リターン・プロセスでは、リターン・アドレスとループ・コンテキストがスタック（メモリ内）に格納されます。CPU がサブルーチンから復帰する場合、これらの値が復元される速度はメモリ・アクセスの速度によって異なります。

ファースト・リターン・プロセスでは、リターン・アドレスとループ・コンテキストがレジスタに保存され、これらの値をただちに復元することができます。これらの特別なレジスタは、リターン・アドレス・レジスタ（RETA）と制御フロー・コンテキスト・レジスタ（CFCT）です。専用の 32 ビットのロード / またはストアする命令を使用して、RETA と CFCT をペアとしてリードまたはライトを行うことできます。

リターン・アドレスとループ・コンテキストがどのように複数レイヤのルーチンで処理されるかを図 4-2（スロー・リターン）と図 4-3（ファースト・リターン）に示します。これらの図では、ルーチン 0 が最上位レベルのルーチンで、ルーチン 1 がルーチン 0 にネストされ、ルーチン 2 がルーチン 1 にネストされています。

図 4-2. スロー・リターン・プロセス時のリターン・アドレスとループ・コンテキストの受け渡し

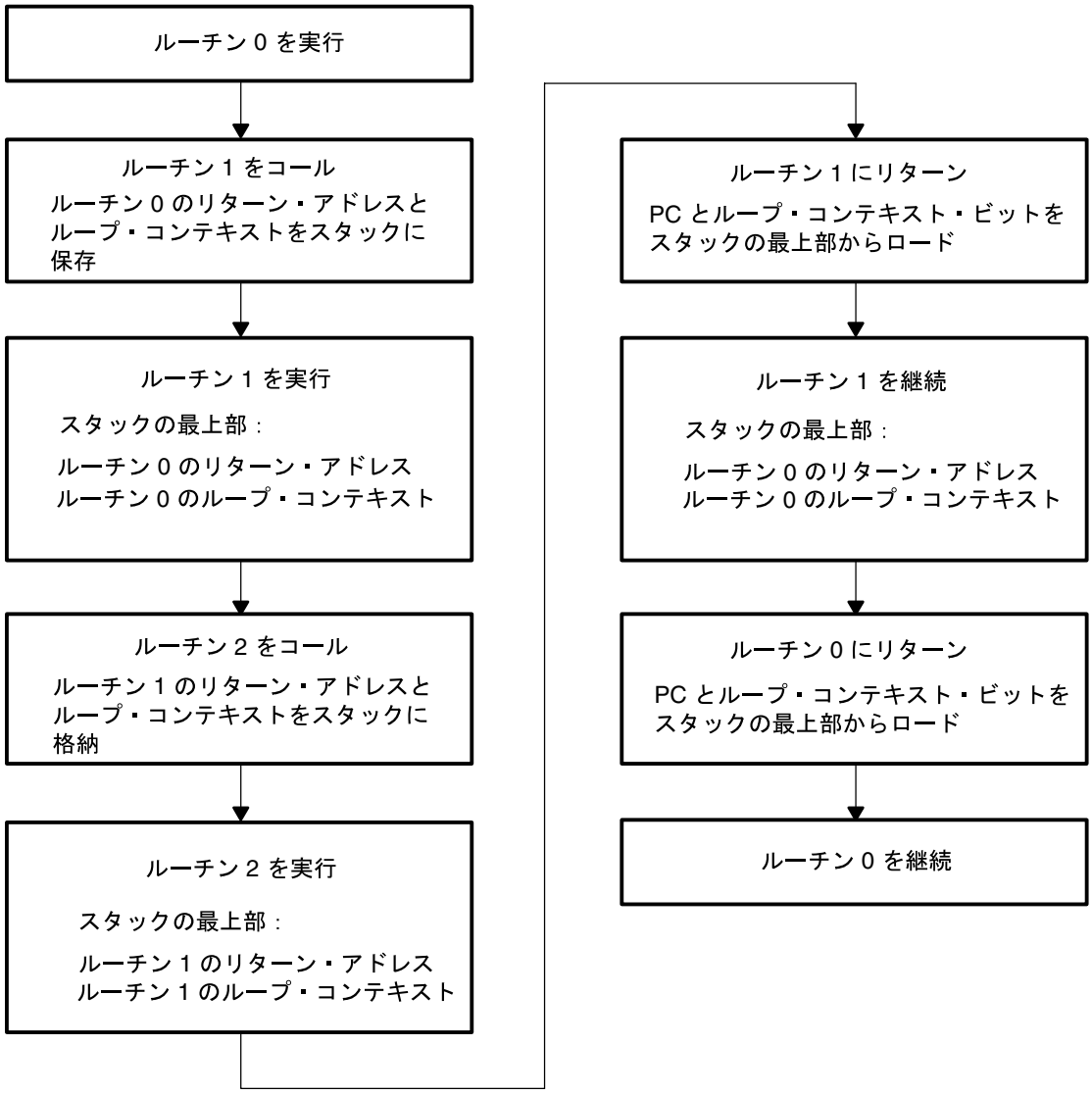
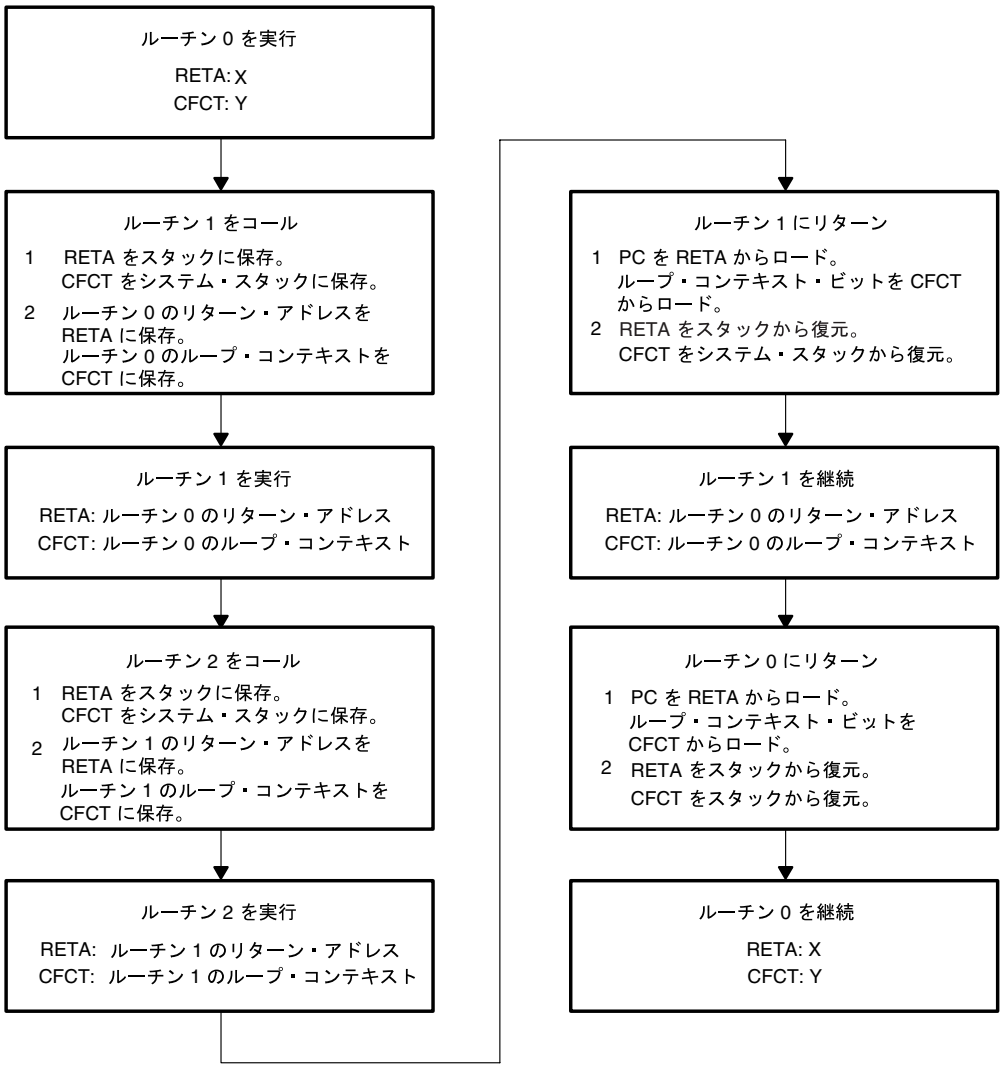


図 4-3. ファースト・リターン・プロセスにおける RETA と CFCT の使用



4.4 自動コンテキスト切り替え

割り込みサービス・ルーチン（ISR）または呼び出されたルーチンを開始する前に、CPU は特定の値を自動的に保存します。CPU は、サブルーチンが完了したときに、中断されたプログラム・シーケンスのコンテキストを再度確立するために、これらの値を使用します。

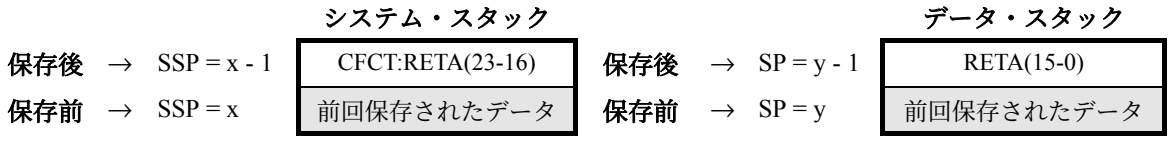
割り込みへの応答でもコールへの応答でも、CPU はリターン・アドレスとループ・コンテキスト・ビットを保存します。プログラム・カウンタ（PC）から渡されたリターン・アドレスは、CPU がサブルーチンから復帰するときに実行される命令のアドレスです。ループ・コンテキスト・ビットは、割り込みまたはコールが発生したときにアクティブであったリピート・ループのタイプとステータスが記録されています。割り込みへの応答時、CPU はさらにステータス・レジスタ 0、1、2 とデバッグ・ステータス・レジスタを保存します。DBSTAT は、エミュレーション時に使用されるデバッグ・コンテキスト情報を保持する DSP レジスタです。

選択されているスタック構成（4.2 節を参照）でファースト・リターン・プロセスを使用している場合、RETA がリターン・アドレスの一時的な格納場所として使用され、CFCT がループ・コンテキスト・ビットの一時的な格納場所として使用されます。選択されているスタック構成でスロー・リターン・プロセスを使用している場合、リターン・アドレスとループ・コンテキスト・ビットはスタックに保存され、スタックから復元されます。

4.4.1 コールのファースト・リターン・コンテキスト切り替え

コールされたルーチンを開始する前に、CPU は次の動作を自動的に実行します。

1) CFCT と RETA をシステム・スタックとデータ・スタックに同時に保存します。スタックごとに、CPU はスタックにライトする前にスタック・ポインタ（SSP または SP）を 1 つデクリメントします。



2) リターン・アドレスを RETA に保存し、ループ・コンテキスト・フラグを CFCT に保存します （次の図を参照）。

サブルーチンの終わりのリターン命令は、CPU に値を逆順で復元させます。最初に、CPU はリターン・アドレスを RETA から PC に送り、そのループ・コンテキスト・フラグを CFCT から復元します。次に、CPU は CFCT と RETA の値をスタックから同時にリードします。スタックごとに、CPU はスタックからリード後にスタック・ポインタ（SSP または SP）を 1 つインクリメントします。

4.4.2 割り込みのファースト・リターン・コンテキスト切り替え

割り込みサービス・ルーチン（ISR）を開始する前に、CPU は次の動作を自動的に実行します。

1) レジスタをシステム・スタックとデータ・スタックに同時に保存します。スタックごとに、CPU はスタックにライトする前に毎回スタック・ポインタ（SSP または SP）を 1 つデクリメントします。

		システム・スタック			データ・スタック
保存後 →	SSP = x - 3	CFCT:RETA(23-16)	保存後 →	SP = y - 3	RETA(15-0)
	SSP = x - 2	DBSTAT		SP = y - 2	ST1_55
	SSP = x - 1	ST0_55		SP = y - 1	ST2_55
保存前 →	SSP = x	前回保存されたデータ	保存前 →	SP = y	前回保存されたデータ

注：

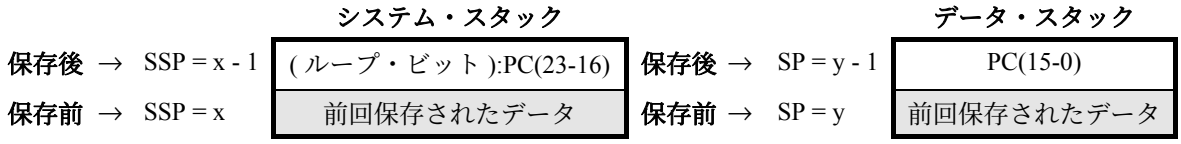
DBSTAT（デバッグ・ステータス・レジスタ）は、エミュレーション時に使用されるデバッグ・コンテキスト情報を保持します。ISR は DBSTAT に戻される値を変更しないことを確認してください。

2) リターン・アドレスを（PC から）RETA に保存し、ループ・コンテキスト・フラグを CFCT に保存します （次の図を参照）。

ISR の終わりの割り込みからのリターン命令は、CPU に値を逆順で復元させます。最初に、CPU はリターン・アドレスを RETA から PC に送り、そのループ・コンテキスト・フラグを CFCT から復元します。次に、CPU はそれらの値をスタックから同時にリードします。スタックごとに、CPU はスタックからリード後に毎回スタック・ポインタ（SSP または SP）を 1 つインクリメントします。

4.4.3 コールのスロー・リターン・コンテキスト切り替え

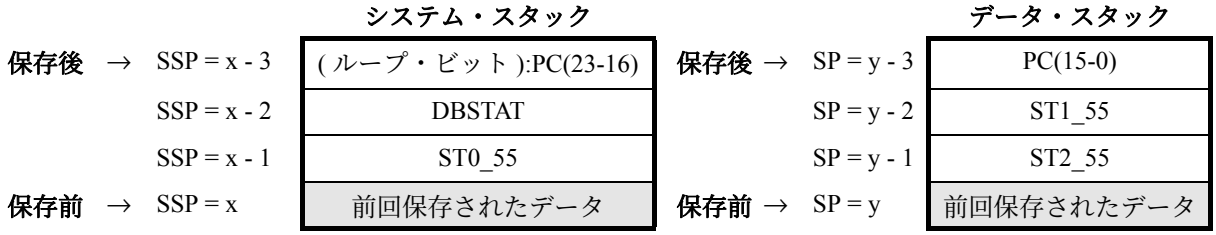
コールされたルーチンを開始する前に、CPU はリターン・アドレス（PC から）とループ・コンテキスト・ビットをシステム・スタックとデータ・スタックに同時に保存します。スタックごとに、CPU はスタックにライトする前にスタック・ポインタ（SSP または SP）を 1 つデクリメントします。



サブルーチンの終わりのリターン命令は、CPU にリターン・アドレスとループ・コンテキストをスタックから復元させます。スタックごとに、CPU はスタックからリード後にスタック・ポインタ（SSP または SP）を 1 つインクリメントします。

4.4.4 割り込みのスロー・リターン・コンテキスト切り替え

割り込みサービス・ルーチン（ISR）を開始する前に、CPU はレジスタをシステム・スタックとデータ・スタックに同時に自動的に保存します。スタックごとに、CPU はスタックにライトする前に毎回スタック・ポインタ（SSP または SP）を 1 つデクリメントします。



注 :

DBSTAT**（デバッグ・ステータス・レジスタ）は、エミュレーション時に使用されるデバッグ・コンテキスト情報を保持します。**ISR **は** DBSTAT **に戻される値を変更しないことを確認してください。**

ISR の終わりの割り込みからのリターン命令は、CPU に値を逆順で復元させます。最初に、CPU はリターン・アドレスとループ・コンテキスト・ビットをスタックから復元します。次に、CPU は他の値をスタックから同時にリードします。スタックごとに、CPU はスタックからリード後に毎回スタック・ポインタ（SSP または SP）を 1 つインクリメントします。

第 5 章

割り込みおよびリセット

この章では、C55x DSP の使用可能な割り込みについて説明します。また、これらの割り込みの一部をソフトウェアでブロックする方法、およびこれらの割り込みのすべてを CPU がどのように処理するかについても説明します。さらに、この章では、2 つのタイプのリセット演算による自動的な効果についても説明します。リセット演算の 1 つはハードウェアで開始され、もう 1 つはソフトウェアで開始されます。

項目 **ページ**

5.1 割り込みの概要

割り込みはハードウェア・ドリブンまたはソフトウェア・ドリブンの信号であり、割り込みにより DSP は現行のプログラム・シーケンスを一時停止し、割り込みサービス・ルーチン（ISR）と呼ばれる別のタスクを実行します。TMS320C55x (C55x) DSP では、32 の ISR をサポートします。一部の ISR はソフトウェアまたはハードウェアでトリガされ、他の ISR はソフトウェアでのみトリガされます。CPU が複数のハードウェア割り込み要求を同時に受け取る場合、CPU は事前に定義された優先順位に応じてそれらの要求を処理します（5.2 節「割り込みベクタと優先順位」を参照）。

C55x 割り込みはすべて、ハードウェア割り込みでもソフトウェア割り込みでも、次の 2 つのカテゴリのいずれかに分類できます。マスカブル割り込みは、ソフトウェアでブロック（マスク）できます。ノンマスカブル割り込みは、ブロックできません。ソフトウェア割り込みはすべてノンマスカブルです。

DSP は、次の 4 つの主なフェーズで割り込みを処理します。

1) 割り込み要求の受信。ソフトウェアまたはハードウェアが、現行のプログラム・シーケンスの一時停止を要求します。

2) 割り込み要求に応答。CPU は、要求に応答する必要があります。割り込みがマスカブルである場合、応答するための所定の条件を満たしている必要があります。ノンマスカブル割り込みの場合、応答は迅速です。

3) 割り込みサービス・ルーチンの準備。CPU で実行される主なタスクは、次のとおりです。

- ❑ 現行の命令の実行を完了、デコード・フェーズに達していない命令をパイプラインからフラッシュ。
- ❑ 特定のレジスタ値をデータ・スタックとシステム・スタックに自動的に格納（4.2 節に記述されている自動コンテキスト切り替えに関する説明を参照）。
- ❑ 事前設定ベクタ・アドレスに格納した割り込みベクタをフェッチ。割り込みベクタは、割り込みサービス・ルーチンを指します。

4) 割り込みサービス・ルーチンを実行。CPU は、ユーザーがライトした ISR を実行します。ISR は割り込みからの復帰命令で終わり、自動的に保存されたレジスタ値が自動的に復元されます（4.4 節に記述されている自動コンテキスト切り替えの説明を参照）。

注:

1) **外部からの割り込みは、**CPU **がリセットを終了してから少なくとも** 3 **サイクルで発生する必要があります。そうしないと、その割り込みは認識されません。**

2) INTM **ビットと** IER0 **レジスタと** IER1 **レジスタの設定にかかわらず、ハードウェア・リセットに追随して、すべての割り込み（マスカブルとノンマスカブル）はディスエーブルになります。スタックポインタ（**SP **レジスタと** SSP **レジスタ）がソフトウェアで初期化されるまで、割り込みはディスエーブルのままです。スタックが初期化されると、**INTM **ビットと** IER0 **レジスタと** IER1 **レジスタは、割り込みのイネーブルを判別します。**

5.2 割り込みベクタと優先順位

TMS320C55x DSP は、32 の割り込みサービス・ルーチン（ISR）をサポートします。割り込み要求を受け取り、応答した後、CPU は割り込みベクタ・アドレスを生成します。ベクタ・アドレスでは、CPU は対応する ISR を指すベクタをフェッチします。複数のハードウェア割り込みが同時に発生する場合、CPU は事前定義されたハードウェア割り込みの優先順位に応じて 1 つずつ処理します。ISR 番号順にベクタを表 5-1 に示します。優先度順にベクタを表 5-2 に示します。いずれの表も、一般的な C55x のベクタを示しています。各ベクタに対応する割り込みを確認するには、使用する C55x DSP のデータ・マニュアルを参照してください。

目的の割り込みベクタ（ISR スタート・アドレス）をベクタ・アドレスにライトする必要があります。各割り込みベクタには、8 バイト必要です。リセット・ベクタのバイト 0 には、スタック・モードの設定があります。その他のベクタのバイト 0 は、無視されます。バイト 1 ～ 3 は、割り込みサービス・ルーチン（ISR）の 24 ビット・バイト・アドレスをエンコードします。バイト 4 ～ 7 は、NOP 命令が埋め込まれなければなりません。

IVPD と IVPH の 2 つのベクタ・ポインタは、プログラム空間の最大 32 の割り込みベクタを指します。IVPD は、割り込みベクタ 0 ～ 15 および 24 ～ 31 に対する 256 バイトのプログラム・ページを指します。IVPH は、割り込みベクタ 16 ～ 23 に対する 256 バイトのプログラム・ページを指します。これらのポインタの詳細は、2.8.1 項を参照してください。

表 5-1. ISR 番号順の割り込みベクタ

ISR 番号	ハードウェア割り込み優先順位	ベクタ名	ベクタ・アドレス（バイト・アドレス）	ISR の対象
0	1（最上位）	RESETIV(IV0)	IVPD:0h	リセット（ハードウェアとソフトウェア）
1	2	NMIV (IV1)	IVPD:8h	ハードウェア・ノンマスカブル割り込み（NMI）またはソフトウェア割り込み 1
2	4	IV2	IVPD:10h	ハードウェアまたはソフトウェア割り込み
3	6	IV3	IVPD:18h	ハードウェアまたはソフトウェア割り込み
4	7	IV4	IVPD:20h	ハードウェアまたはソフトウェア割り込み
5	8	IV5	IVPD:28h	ハードウェアまたはソフトウェア割り込み
6	10	IV6	IVPD:30h	ハードウェアまたはソフトウェア割り込み
7	11	IV7	IVPD:38h	ハードウェアまたはソフトウェア割り込み
8	12	IV8	IVPD:40h	ハードウェアまたはソフトウェア割り込み
9	14	IV9	IVPD:48h	ハードウェアまたはソフトウェア割り込み
10	15	IV10	IVPD:50h	ハードウェアまたはソフトウェア割り込み
11	16	IV11	IVPD:58h	ハードウェアまたはソフトウェア割り込み
12	18	IV12	IVPD:60h	ハードウェアまたはソフトウェア割り込み

表 5-1. ISR 番号順の割り込みベクタ（続き）

ISR 番号	ハードウェア割り込み優先順位	ベクタ名	ベクタ・アドレス（バイト・アドレス）	ISR の対象
13	19	IV13	IVPD:68h	ハードウェアまたはソフトウェア割り込み
14	22	IV14	IVPD:70h	ハードウェアまたはソフトウェア割り込み
15	23	IV15	IVPD:78h	ハードウェアまたはソフトウェア割り込み
16	5	IV16	IVPH:80h	ハードウェアまたはソフトウェア割り込み
17	9	IV17	IVPH:88h	ハードウェアまたはソフトウェア割り込み
18	13	IV18	IVPH:90h	ハードウェアまたはソフトウェア割り込み
19	17	IV19	IVPH:98h	ハードウェアまたはソフトウェア割り込み
20	20	IV20	IVPH:A0h	ハードウェアまたはソフトウェア割り込み
21	21	IV21	IVPH:A8h	ハードウェアまたはソフトウェア割り込み
22	24	IV22	IVPH:B0h	ハードウェアまたはソフトウェア割り込み
23	25	IV23	IVPH:B8h	ハードウェアまたはソフトウェア割り込み
24	3	BERRIV (IV24)	IVPD:C0h	バス・エラー割り込みまたはソフトウェア割り込み
25	26	DLOGIV (IV25)	IVPD:C8h	データ・ログ割り込みまたはソフトウェア割り込み
26	27（最下位）	RTOSIV (IV26)	IVPD:D0h	リアルタイム・オペレーティング・システム割り込みまたはソフトウェア割り込み
27	-	SIV27	IVPD:D8h	予約済み
28	-	SIV28	IVPD:E0h	予約済み
29	-	SIV29	IVPD:E8h	予約済み
30	-	SIV30	IVPD:F0h	ソフトウェア（専用）割り込み
31	-	SIV31	IVPD:F8h	ソフトウェア（専用）割り込み 31

表 5-2. 優先度順の割り込みベクタ

ISR 番号	ハードウェア割り込み優先順位	ベクタ名	ベクタ・アドレス（バイト・アドレス）	ISR の対象
0	1（最上位）	RESETIV(IV0)	IVPD:0h	リセット（ハードウェアとソフトウェア）
1	2	NMIV (IV1)	IVPD:8h	ハードウェア・ノンマスカブル割り込み（NMI）またはソフトウェア割り込み 1
24	3	BERRIV (IV24)	IVPD:C0h	バス・エラー割り込みまたはソフトウェア割り込み
2	4	IV2	IVPD:10h	ハードウェアまたはソフトウェア割り込み
16	5	IV16	IVPH:80h	ハードウェアまたはソフトウェア割り込み
3	6	IV3	IVPD:18h	ハードウェアまたはソフトウェア割り込み
4	7	IV4	IVPD:20h	ハードウェアまたはソフトウェア割り込み
5	8	IV5	IVPD:28h	ハードウェアまたはソフトウェア割り込み
17	9	IV17	IVPH:88h	ハードウェアまたはソフトウェア割り込み
6	10	IV6	IVPD:30h	ハードウェアまたはソフトウェア割り込み
7	11	IV7	IVPD:38h	ハードウェアまたはソフトウェア割り込み
8	12	IV8	IVPD:40h	ハードウェアまたはソフトウェア割り込み
18	13	IV18	IVPH:90h	ハードウェアまたはソフトウェア割り込み
9	14	IV9	IVPD:48h	ハードウェアまたはソフトウェア割り込み
10	15	IV10	IVPD:50h	ハードウェアまたはソフトウェア割り込み
11	16	IV11	IVPD:58h	ハードウェアまたはソフトウェア割り込み
19	17	IV19	IVPH:98h	ハードウェアまたはソフトウェア割り込み
12	18	IV12	IVPD:60h	ハードウェアまたはソフトウェア割り込み
13	19	IV13	IVPD:68h	ハードウェアまたはソフトウェア割り込み
20	20	IV20	IVPH:A0h	ハードウェアまたはソフトウェア割り込み
21	21	IV21	IVPH:A8h	ハードウェアまたはソフトウェア割り込み
14	22	IV14	IVPD:70h	ハードウェアまたはソフトウェア割り込み
15	23	IV15	IVPD:78h	ハードウェアまたはソフトウェア割り込み
22	24	IV22	IVPH:B0h	ハードウェアまたはソフトウェア割り込み
23	25	IV23	IVPH:B8h	ハードウェアまたはソフトウェア割り込み
25	26	DLOGIV (IV25)	IVPD:C8h	データ・ログ割り込みまたはソフトウェア割り込み
26	27（最下位）	RTOSIV (IV26)	IVPD:D0h	リアルタイム・オペレーティング・システム割り込みまたはソフトウェア割り込み
27	-	SIV27	IVPD:D8h	予約済み

表 5-2. 優先度順の割り込みベクタ（続き）

ISR 番号	ハードウェア割り込み優先順位	ベクタ名	ベクタ・アドレス（バイト・アドレス）	ISR の対象
28	-	SIV28	IVPD:E0h	予約済み
29	-	SIV29	IVPD:E8h	予約済み
30	-	SIV30	IVPD:F0h	ソフトウェア（専用）割り込み
31	-	SIV31	IVPD:F8h	ソフトウェア（専用）割り込み 31

5.3 マスカブル割り込み

マスカブル割り込みは、ソフトウェアでブロック（マスク）またはイネーブル（マスク解除）できます。TMS320C55x のマスカブル割り込みはすべてハードウェア割り込みです。

割り込み	説明
割り込みベクタ 2 ～ 23 に関連付けられた割り込み	これらの 22 の割り込みはそれぞれピンでトリガされるか、DSP のペリフェラルによりトリガされます。
BERRINT	バス・エラー割り込み。この割り込みは、システム・バス・エラーが CPU に送られる場合、または CPU でバス・エラーが発生する場合にトリガされます。
DLOGINT	データ・ログ割り込み。DLOGINT は、データ・ログ転送の終わりに DSP によりトリガされます。次のデータ・ログ転送を開始するために DLOGINT 割り込みサービス・ルーチン（ISR）を使用できます。
RTOSINT	リアルタイム・オペレーティング・システム割り込み。RTOSINT は、ハードウェア・ブレークポイントまたはウォッチポイントによりトリガされます。エミュレーション条件に応じてデータ・ログ転送を開始するために、RTOSINT ISR を使用することができます。

マスカブル割り込みがハードウェアから要求されると、対応する割り込みフラグが割り込みフラグ・レジスタの 1 つにセットされます（2.8.2 項に記述されている IFR0 と IFR1 の説明を参照）。フラグがセットされると、適切にイネーブルにならないかぎり割り込みは処理されません。（5.3.1 項「マスカブル割り込みのイネーブル化に使用するビットとレジスタ」を参照）。

マスカブル割り込みの ISR は、ソフトウェアでも実行できます（5.4 節に記述されているノンマスカブル割り込みに関する説明を参照）。

注：

バス・エラーが発生した場合、エラーの原因である命令の機能、および並列に実行される命令の機能は保証されません。

5.3.1 マスカブル割り込みのイネーブル化に使用するビットとレジスタ

次のビットとレジスタを使用して、マスカブル割り込みをイネーブルにします。

ビット / レジスタ	説明
INTM	割り込みモード・ビット。このビットは、マスカブル割り込みをグローバルにイネーブルまたはディスエーブルにします（2.10.2.8 項に記述されている INTM の説明を参照）。
IER0 および IER1	割り込みイネーブル・レジスタ。各マスカブル割り込みには、2 つのレジスタのいずれかにイネーブル・ビットがあります （2.8.3 項に記述されている IER0 と IER1 の説明を参照）。
DBIER0 および DBIER1	デバッグ割り込みイネーブル・レジスタ。各マスカブル割り込みは、2 つのレジスタのいずれかのビットにより、タイム・クリティカルとして定義されます （2.8.4 項に記述されている DBIER0 と DBIER1 の説明を参照）。

INTM ビット、IER ビット、DBIER ビットの役割は、DSP の動作状態によって異なります（次の 2 つの項を参照）。

5.3.2 マスカブル割り込みの標準的なプロセス・フロー

図 5-1 のフロー・チャートでは、マスカブル割り込み処理の標準的なプロセスの概念モデルを示します。フロー・チャートの各ステップについて表 5-3 で説明します。CPU がリアルタイム・エミュレーション・モードで停止する場合、タイム・クリティカル割り込みだけが処理され、プロセスは異なります（5.3.3 項を参照）。

図 5-1.　　マスカブル割り込みの標準的なプロセス・フロー

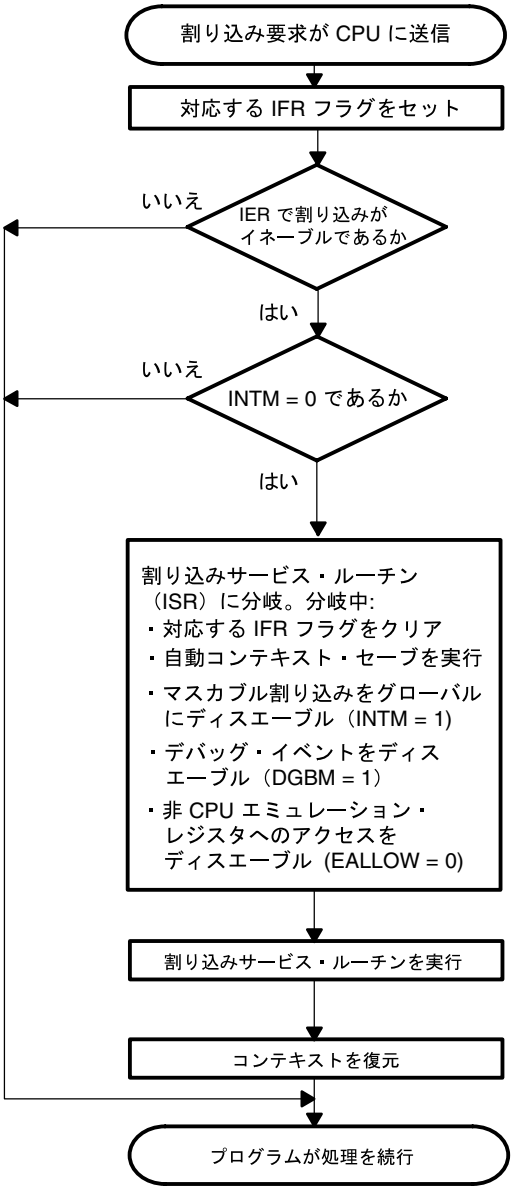


表 5-3. マスカブル割り込みの標準的なプロセス・フローのステップ

ステップ	説明
割り込み要求が CPU に送信	CPU がマスカブル割り込み要求を受け取ります。
対応する IFR フラグをセット	CPU が有効なマスカブル割り込み要求を検出すると、割り込みフラグ・レジスタ（IFR0 または IFR1）のいずれかで対応するフラグをセットし、ラッチします。このフラグは、割り込みに応答があるまで、またはフラグがソフトウェアまたは DSP のハードウェア・リセットによりクリアされるまでラッチされたままになります（2.8.2 項に記述されている IFR0 と IFR1 の説明を参照）。
IER で割り込みがイネーブルであるか	CPU は、割り込みイネーブル・レジスタ（IER0 または IER1）のいずれかで対応するイネーブル・ビットが 1 ではないかぎり割り込みに応答できません（2.8.3 項に記述されている IER0 と IER1 の説明を参照）。
INTM = 0 であるか	CPU は、割り込みモード・ビット（INTM）が 0（ゼロ）ではないかぎり割り込みに応答できません。すなわち、割り込みはグローバルにイネーブルになっていなければなりません（2.10.2.8 項に記述されている INTM の説明を参照）。
割り込みサービス・ルーチンに分岐	CPU は、割り込みベクタに追随して、割り込みサービス・ルーチンに到達します。分岐時、CPU は次のアクションを実行します。 ❑ パイプラインのデコード・フェーズに達した命令を実行します。他の命令は、パイプラインからフラッシュされます。 ❑ IFR0 または IFR1 の対応するフラグをクリアし、割り込みに応答があったことを示します。 ❑ 特定のレジスタ値を自動的に保存し、中断されたプログラム・シーケンスに関する重要なモードとステータス情報を記録します（4.4 節に記述されている自動コンテキスト切り替えに関する説明を参照）。 ❑ INTM = 1（割り込みをグローバルにディスエーブル）、DBGM = 1（デバッグ・イベントをディスエーブル）、および EALLOW = 0（非 CPU エミュレーション・レジスタへのアクセスをディスエーブル）に強制的に設定し、ISR に新しいコンテキストを作成します。
割り込みサービス・ルーチンを実行	CPU は、応答のあった割り込みに対して割り込みサービス・ルーチン（ISR）を実行します。一部のレジスタ値は、ISR への分岐中に自動的に保存されています。ISR の終わりの割り込みからのリターン命令が自動コンテキストの復元動作を実行し（4.4 節に記述されている自動コンテキスト切り替えの説明を参照）、これらのレジスタ値を復元します。ISR が割り込まれたプログラム・シーケンスと他のレジスタを共有する場合、ISR は他のレジスタ値を ISR の始めで保存し、割り込まれたプログラム・シーケンスに復帰する前にこれらの値を復元する必要があります。
プログラムが処理を続行	割り込み要求が適切にイネーブルになっていない場合、CPU はその要求を無視し、プログラムは割り込まれずに処理を続行します。割り込みが適切にイネーブルになっている場合は、その割り込みサービス・ルーチンが実行され、プログラムは割り込まれたところから処理を続行します。

5.3.3 タイム・クリティカル割り込みのプロセス・フロー

図 5-2 のフロー・チャートと表 5-4 の説明では、タイム・クリティカル割り込みが処理されるプロセスの概念モデルを示しています。CPU がリアルタイム・エミュレーション・モードで停止する場合、処理されるマスカブル割り込みはタイム・クリティカル割り込みだけです。それ以外の場合は、CPU は 5.3.2 項で説明している標準的なプロセス・フローを使用します。

図 5-2. タイム・クリティカル割り込みのプロセス・フロー

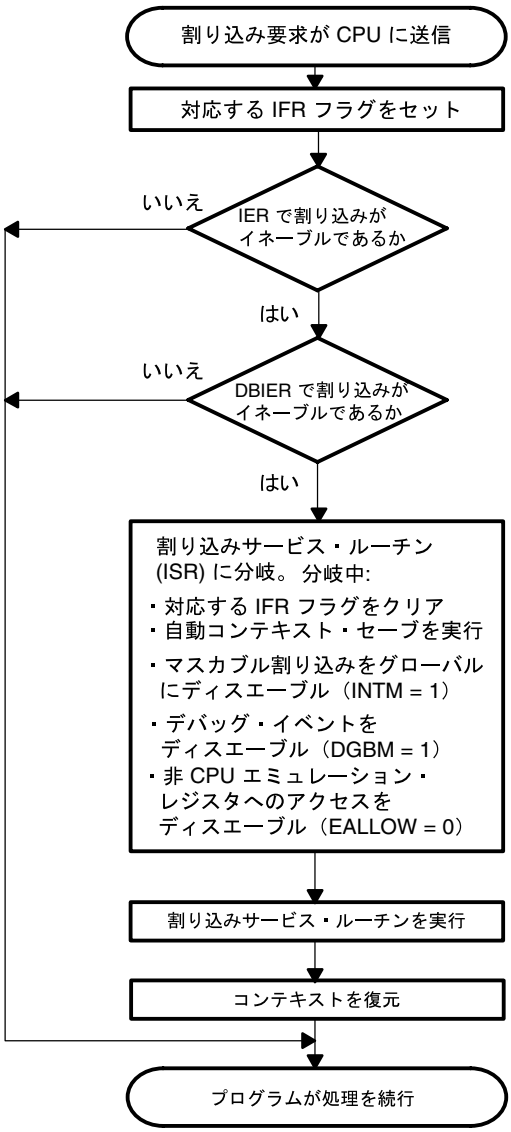


表 5-4. タイム・クリティカル割り込みのプロセス・フローのステップ

ステップ	説明
割り込み要求が CPU に送信	CPU がマスカブル割り込み要求を受け取ります。
対応する IFR フラグをセット	CPU が有効なマスカブル割り込み要求を検出すると、割り込みフラグ・レジスタ（IFR0 または IFR1）のいずれかで対応するフラグをセットし、ラッチします。このフラグは、割り込みに応答があるまで、またはフラグがソフトウェアまたは DSP のハードウェア・リセットによりクリアされるまでラッチされたままになります （2.8.2 項に記述されている IFR0 と IFR1 の説明を参照）。
IER で割り込みがイネーブルであるか	CPU は、割り込みイネーブル・レジスタ（IER0 または IER1）のいずれかで対応するイネーブル・ビットが 1 ではないかぎり割り込みに応答できません （2.8.3 項に記述されている IER0 と IER1 の説明を参照）。
DBIER で割り込みがイネーブルであるか	CPU は、デバッグ割り込みイネーブル・レジスタ（DBIER0 または DBIER1）のいずれかで対応するイネーブル・ビットが 1 ではないかぎり割り込みに応答できません （2.8.4 項に記述されている DBIER0 と DBIER1 の説明を参照）。
割り込みサービス・ルーチンに分岐	CPU は、割り込みベクタに追随して、割り込みサービス・ルーチンに到達します。分岐時、CPU は次のアクションを実行します。 ❑ パイプラインのデコード・フェーズに達した命令を実行します。他の命令は、パイプラインからフラッシュされます。 ❑ IFR0 または IFR1 の対応するフラグをクリアし、割り込みに応答があったことを示します。 ❑ 特定のレジスタ値を自動的に保存し、中断されたプログラム・シーケンスに関する重要なモードとステータス情報を記録します（4.4 節に記述されている自動コンテキスト切り替えに関する説明を参照）。 ❑ INTM = 1（割り込みをグローバルにディスエーブル）、DBGM = 1（デバッグ・イベントをディスエーブル）、および EALLOW = 0（非 CPU エミュレーション・レジスタへのアクセスをディスエーブル）に強制的に設定し、ISR に新しいコンテキストを作成します。
割り込みサービス・ルーチンを実行	CPU は、応答のあった割り込みに対して割り込みサービス・ルーチン（ISR）を実行します。一部のレジスタ値は、ISR への分岐中に自動的に保存されています。ISR の終わりの割り込みからのリターン命令が自動コンテキストの復元動作を実行し（4.4 節に記述されている自動コンテキスト切り替えの説明を参照）、これらのレジスタ値を復元します。ISR が割り込まれたプログラム・シーケンスと他のレジスタを共有する場合、ISR は他のレジスタ値を ISR の始めに格納し、割り込まれたプログラム・シーケンスに復帰する前にこれらの値を復元する必要があります。

表 5-4. タイム・クリティカル割り込みのプロセス・フローのステップ（続き）

ステップ	説明
プログラムが処理を続行	割り込み要求が適切にイネーブルになっていない場合、CPU はその要求を無視し、プログラムは割り込まれずに処理を続行します。割り込みが適切にイネーブルになっている場合は、その割り込みサービス・ルーチンが実行され、プログラムは割り込まれたところから処理を続行します。

5.4 ノンマスカブル割り込み

CPU はノンマスカブル割り込み要求を受け取ると、無条件にその要求に応答し、ただちに対応する割り込みサービス・ルーチン（ISR）に分岐します。ノンマスカブル割り込みは、次のとおりです。

- ハードウェア割り込み $\overline{\text{RESET}}$。$\overline{\text{RESET}}$ ピンをローにドライブすると、DSP ハードウェア・リセットに加えてリセット ISR を実行する割り込みを開始します（DSP ハードウェア・リセットによる特有の影響については、5.5.1 項を参照してください）。
- ハードウェア割り込み $\overline{\text{NMI}}$。$\overline{\text{NMI}}$ ピンをローにドライブすると、CPU は対応する ISR を実行します。$\overline{\text{NMI}}$ は、DSP に無条件に割り込む汎用ハードウェア方式を提供します。
- すべてのソフトウェア割り込みは、次の命令のいずれかで開始されます。

命令	説明
INTR #k5	32 の ISR はいずれも、この命令を使用して開始できます。変数 k5 は、0 ～ 31 の 5 ビットの数字です。ISR を実行する前に、CPU は自動コンテキスト・セーブを実行し（重要なレジスタ値を保存）、INTM ビットをセットします（マスカブル割り込みをグローバルにディスエーブルにする）。
TRAP #k5	この命令は、INTR #k5 と同じ機能を実行します。ただし、INTM ビットには作用**しません**。
RESET	この命令は、ハードウェア・リセット演算のサブセットであるソフトウェア・リセット演算を実行し、その後 CPU にリセット ISR の実行を命令します （ソフトウェア・リセットによる特有の影響については、5.5.2 項を参照してください）。

5.4.1 ノンマスカブル割り込みの標準的なプロセス・フロー

次のフロー・チャートでは、ノンマスカブル割り込み処理の標準的なプロセスの概念モデルを示します。

注 :

割り込みが TRAP **命令で開始された場合、割り込みサービス・ルーチンへの分岐時、**INTM **ビットは影響を受けません。**

図 5-3. ノンマスカブル割り込みの標準的なプロセス・フロー

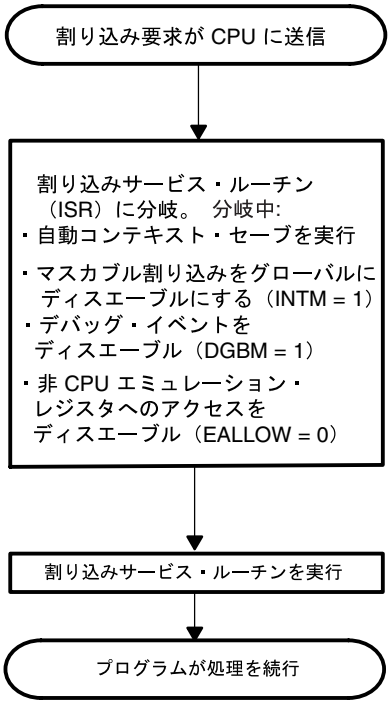


表 5-5. ノンマスカブル割り込みの標準的なプロセス・フローのステップ

ステップ	説明
割り込み要求が CPU に送信	CPU がノンマスカブル割り込み要求を受け取ります。
割り込みサービス・ルーチンに分岐	CPU は、割り込みベクタに追随して、割り込みサービス・ルーチンに到達します。分岐時、CPU は次のアクションを実行します。 ❑ パイプラインのデコード・フェーズに達した命令を実行します。他の命令は、パイプラインからフラッシュされます。 ❑ 特定のレジスタ値を自動的に保存し、割り込まれたプログラム・シーケンスに関する重要なモードとステータス情報を記録します（4.4 節に記述されている自動コンテキスト切り替えに関する説明を参照）。 ❑ INTM = 1（割り込みをグローバルにディスエーブル）、DBGM = 1（デバッグ・イベントをディスエーブル）、および EALLOW = 0（非 CPU エミュレーション・レジスタへのアクセスをディスエーブル）に強制的に設定し、ISR に新しいコンテキストを作成します。
割り込みサービス・ルーチンを実行	CPU は、応答のあった割り込みに対して割り込みサービス・ルーチン（ISR）を実行します。一部のレジスタ値は、ISR への分岐中に自動的に保存されています。ISR の終わりの割り込みからのリターン命令が自動コンテキストの復元動作を実行し（4.4 節に記述されている自動コンテキスト切り替えの説明を参照）、これらのレジスタ値を復元します。ISR が割り込まれたプログラム・シーケンスと他のレジスタを共有する場合、ISR は他のレジスタ値を ISR の始めに保存し、割り込まれたプログラム・シーケンスに復帰する前にこれらの値を復元する必要があります。
プログラムが処理を続行	割り込みサービス・ルーチンが完了した後、プログラムは割り込まれたところから処理を続行します。

5.5 DSP のリセット

ここでは、DSP のハードウェア・リセットとソフトウェア・リセットについて説明します。DSP レジスタでのハードウェア・リセットとソフトウェア・リセットによる影響を表 5-6 にまとめます。

DSP のハードウェア・リセットについて 5.5.1 項に、DSP のソフトウェア・リセットについて 5.5.2 項にそれぞれ説明します。CPU レジスタでの両方のリセットによる影響を表 5-6 にまとめます。

注:

ハードウェア・リセットにより IVPD **と呼ばれる割り込みベクタ・ポインタに** FFFFh **がロードされ、**CPU **がリセット・ベクタをプログラム・アドレス** FF FF00h **からフェッチします。ソフトウェア・リセット時、**IVPD **は変更されません。**CPU **は現行の** IVPD **値を使用して、リセット・ベクタをフェッチします。**

表 5-6. CPU レジスタでのリセットによる影響

		リセット値		
レジスタ	**ビット**	**H/W**	**S/W**	**備考**
AC0 ～ AC3	すべて	0	0	
BK03、BK47、BKC	すべて	0	0	
BRC0、BRC1	すべて	0	0	
BRS1	すべて	0	0	
BSA01	すべて	0	†	
BSA23	すべて	0	†	
BSA45	すべて	0	†	
BSA67	すべて	0	†	
BSAC	すべて	0	†	
CFCT	すべて	0	†	アクティブ・ループの内容がクリアされます。
CSR	すべて	0	0	
DBIER0、DBIER1	すべて	0	†	すべてのタイム・クリティカル割り込みがディスエーブルになります。

† ソフトウェア・リセットによる影響を受けません。

表 5-6　　CPU レジスタでのリセットによる影響（続き）

		リセット値		
レジスタ	**ビット**	**H/W**	**S/W**	**備考**
IER0 IER1	すべて	0	0	すべてのマスカブル割り込みがディスエーブルになります。
IFR0 IFR1	すべて	0	0	すべての保留されている割り込みがクリアされます。
IVPD	すべて	FFFFh	†	IVPD が参照するベクタは、アドレス FFF FF00h で開始する 256 バイトのプログラム・ページにあります。
IVPH	すべて	FFFFh	†	IVPH が参照するベクタは、IVPD が参照するベクタと同じ 256 バイトのプログラム・ページにあります。
PC	すべて	0	0	
PDP	すべて	0	0	
REA0、 REA1	すべて	0	0	
RETA	すべて	0	†	RETA に格納されているリターン・アドレスがクリアされます。
RPTC	すべて	0	0	
RSA0、 RSA1	すべて	0	0	
ST0_55	0 ～ 8:DP	0	0	データ・ページ 0 が選択されます。ST0_55 内のフラグがクリアされます。
	9: ACOV1	0	0	
	10: ACOV0	0	0	
	11: C	1	1	
	12: TC2	1	1	
	13: TC1	1	1	
	14: ACOV3	0	0	
	15: ACOV2	0	0	
ST1_55	0 ～ 4:ASM	0	0	ASM から影響を受ける命令は、0（ゼロ）のシフト・カウント（シフトなし）を使用します。ASM がクリアされると、レジスタ T2 もクリアされます。これは、C54CM = 1 の場合の ASM と T2 との関係によります（2.10.2.1 項に記述されている ASM の説明を参照）。

† ソフトウェア・リセットによる影響を受けません。

表 5-6　　CPU レジスタでのリセットによる影響（続き）

レジスタ	ビット	リセット値 H/W	リセット値 S/W	備考
ST1_55（続き）	5: C54CM	1	1	TMS320C54x 互換モードがオンになります。
	6: FRCT	0	0	乗算演算の結果はシフトされません。
	7: C16	0	0	デュアル 16 ビット・モードがオフになります。C16 から影響を受ける命令に対して、D ユニットの ALU が 2 つの並列 16 ビット演算ではなく、1 つの 32 ビット演算を実行します。
	8: SXMD	1	1	符号拡張モードがオンになります。
	9: SATD	0	0	CPU は、D ユニットのオーバーフロー結果を飽和しません。
	10: M40	0	0	32 ビット（40 ビットではなく）計算モードが D ユニットに対して選択されます。
	11: INTM	1	1	マスカブル割り込みがグローバルにディスエーブルになります。
	12: HM	0	0	アクティブ $\overline{\text{HOLD}}$ 信号により DSP がその外部インターフェイスをハイ・インピーダンス状態に置く場合、DSP が内部メモリからコードを実行し続けます。
	13: XF	1	1	ピン XF が High にドライブされます。
	14: CPL	0	0	DP（SP ではなく）直接アドレッシング・モードが選択されます。データ空間への直接アクセスは、データ・ページ・レジスタ（DP）を基準として行われます。
	15: BRAF	0	0	このフラグがクリアされます （BRAF がブロック・リピート演算のステータスを識別または制御します)。
ST2_55	0: AR0LC	0	0	AR0 がリニア・アドレッシング（サーキュラ・アドレッシングではなく）に使用されます。
	1: AR1LC	0	0	AR1 がリニア・アドレッシングに使用されます。
	2: AR2LC	0	0	AR2 がリニア・アドレッシングに使用されます。
	3: AR3LC	0	0	AR3 がリニア・アドレッシングに使用されます。
	4: AR4LC	0	0	AR4 がリニア・アドレッシングに使用されます。
	5: AR5LC	0	0	AR5 がリニア・アドレッシングに使用されます。
	6: AR6LC	0	0	AR6 がリニア・アドレッシングに使用されます。

† ソフトウェア・リセットによる影響を受けません。

表 5-6　　CPU レジスタでのリセットによる影響（続き）

レジスタ	ビット	リセット値 H/W	リセット値 S/W	備考
ST2_55（続き）	7: AR7LC	0	0	AR7 がリニア・アドレッシングに使用されます。
	8: CDPLC	0	0	CDP がリニア・アドレッシングに使用されます。
	9: 予約済み	0	0	
	10: RDM	0	0	命令がオペランドの丸めを指定する場合、CPU は無限大の丸め（最も近い値への丸めではなく）を使用します。
	11: EALLOW	0	0	プログラムは非 CPU エミュレーション・レジスタにはライトできません。
	12: DBGM	1	1	デバッグ・イベントがディスエーブルになります。
	13 ～ 14: 予約済み	11b	11b	
	15: ARMS	0	0	AR 間接アドレッシング・モードを使用する場合、DSP モード（制御モードではなく）のオペランドが使用可能になります。
ST3_55	0: SST	0	†	TMS320C54x 互換モード（C54CM = 1）では、一部のアキュムレータ・ストア命令の実行が SST から影響を受けます。SST が 0 の場合は、40 ビットのアキュムレータ値がストア演算の前に 32 ビット値に飽和されません。
	1: SMUL	0	†	乗算結果は飽和されません。
	2: CLKOFF	0	†	CLKOUT ピンの出力がイネーブルになり、CLKOUT クロック信号を反映します。
	3-4: 予約済み	0	†	
	5: SATA	0	0	CPU は、A ユニットのオーバーフロー結果を飽和しません。
	6: MPNMC	ピン	†	MPNMC のリセット値は、リセット時に事前定義されたピンの状態によって異なることがあります。特定の C55x DSP に関してリセット値を確認するには、対応するデータ・マニュアルを参照してください。
	7: CBERR	0	†	このフラグがクリアされます（CBERR は、内部バス・エラーが検出される時点を示します）。
	11-8: 予約済み	1100b	†	

† ソフトウェア・リセットによる影響を受けません。

表 5-6　　CPU レジスタでのリセットによる影響（続き）

レジスタ	ビット	リセット値 H/W	リセット値 S/W	備考
ST3_55（続き）	12: HINT	1	†	ホスト・プロセッサへの割り込みに使用する信号は、ハイ・レベルになります。
	13: CACLR	0	†	このビットはクリアされます（CACLR を使用して、命令キャッシュのフラッシュのステータスを開始し、チェックします）。
	14: CAEN	0	†	プログラム・キャッシュがディスエーブルになります。
	15: CAFRZ	0	†	キャッシュはフリーズしません。
T0	すべて	0	0	
T1	すべて	0	0	
T2	すべて	0	0	ST1_55 の ASM フィールドがクリアされるため T2 がクリアされます。これは、C54CM = 1 の場合の ASM と T2 との関係によります（2.10.2.1 項に記述されている ASM の説明を参照）。
T3	すべて	0	0	
TRN0、TRN1	すべて	0	0	
XAR0	すべて（AR0H:AR0）	0	†	
XAR1	すべて（AR1H:AR1）	0	†	
XAR2	すべて（AR2H:AR2）	0	†	
XAR3	すべて（AR3H:AR3）	0	†	
XAR4	すべて（AR4H:AR4）	0	†	
XAR5	すべて（AR5H:AR5）	0	†	
XAR6	すべて（AR6H:AR6）	0	†	
XAR7	すべて（AR7H:AR7）	0	†	
XCDP	すべて（DPH:DP）	0	0	
XDP	すべて（DPH:DP）	0	†	
XSP	すべて（SPH:SP）	0	†	
XSSP	すべて（SPH:SSP）	0	0	

† ソフトウェア・リセットによる影響を受けません。

5.5.1 DSP のハードウェア・リセット

DSP リセット信号は、アサートされると、DSP を既知の状態にします。ハードウェア・リセットの一環として、現行の演算がすべて中止され、命令パイプラインが空になり、CPU レジスタがリセットされます。その後、CPU はリセット割り込みサービス・ルーチンを実行します（5.4.1 項に記述されているノンマスカブル割り込みの標準的なプロセス･フローを参照)。リセット割り込みベクタをリードする場合､CPU は使用するスタック構成を決定するために 32 ビットのリセット・ベクタ・ロケーションのビット 29 と 28 を使用します（4.2 節を参照)。

DSP レジスタでのハードウェア･リセットによる影響を表 5-6 にまとめます。ソフトウェア・リセット（5.5.2 項を参照）は、これらのレジスタ変更のサブセットを実行します。

$\overline{\text{RESET}}$ ピンは、特定のクロック数に対してアサートされる必要があります（対応するデータ・マニュアルを参照）。DSP がエミュレーション用に停止しているときに $\overline{\text{RESET}}$ ピンがアサートおよびディアサートされる場合、リセットは無視されます。

注:

1) **外部からの割り込みは、**CPU **がリセットを終了してから少なくとも** 3 **サイクルで発生する必要があります。そうしないと、その割り込みは認識されません。**

2) **ハードウェア・リセットに追随して、すべての割り込み（マスカブルとノンマスカブル）はディスエーブルになります。スタックポインタ（**SP **レジスタと** SSP **レジスタ）がソフトウェアで初期化されるまで、割り込みはディスエーブルのままです。スタックが初期化されると、**INTM **ビットと** IER0 **レジスタと** IER1 **レジスタは、割り込みのイネーブルを決定します。**

5.5.2 ソフトウェア・リセット

ソフトウェア・リセットは、ソフトウェア・リセット命令で開始されるリセットです。ソフトウェア・リセットは、IFR0、IFR1、ST0_55、ST1_55、および ST2_55 にだけ影響を与えます。その他のレジスタには影響を与えません。ソフトウェア・リセット値（表 5-6 を参照）は、DSP のハードウェア・リセットにより実行される値と同一です。

リセット割り込みベクタをリードする場合、CPU は使用するスタック構成を決定するために 32 ビットのリセット・ベクタ・ロケーションのビット 29 と 28 を使用します（4.2 節を参照)。

第 6 章

アドレッシング・モード

この章では、C55x DSP のデータ空間（CPU レジスタを含む）および I/O 空間のアドレス指定に使用可能なモードについて説明します。

項目 **ページ**

6.1 アドレッシング・モードの概要

TMS320C55x DSP は、データ・メモリ、メモリ・マップド・レジスタ、レジスタ・ビット、および I/O 空間にフレキシブルにアクセスする、3 つのタイプのアドレッシング・モードをサポートします。

- ❑ 絶対アドレッシング・モード（6.2 節を参照）を使用すると、アドレスのすべてまたは一部を定数として命令内に指定することでロケーションを参照できます。
- ❑ 直接アドレッシング・モード（6.3 節を参照）を使用すると、アドレスのオフセットを使用してロケーションを参照できます。
- ❑ 間接アドレッシング・モード（6.4 節を参照）を使用すると、ポインタを使用してロケーションを参照できます。

注:

ある特定の命令は DARAM **または** SARAM **の** 2 **つの別々のブロックが使用されない限り、同一サイクルで** 2 **つの演算を実行できないため注意してください。**

それぞれのアドレッシング・モードは、1 つまたは複数のタイプのオペランドを供給します。アドレッシング・モードのオペランドをサポートする命令は、表 6-1 で説明される構文要素の一つを持っています。

表 6-1. アドレッシング・モードをサポートする構文要素

構文要素	説明
Smem	命令構文に Smem が含まれている場合、その命令はデータ・メモリ、I/O 空間、またはメモリ・マップド・レジスタから 1 ワード（16 ビット）のデータにアクセスできます。この命令を記述する際、Smem を互換性のあるアドレッシング・モードのオペランドに置き換えます。
Lmem	命令構文に Lmem が含まれている場合、その命令はデータ・メモリ、またはメモリ・マップド・レジスタからロング・ワード（32 ビット）のデータにアクセスできます。この命令を記述する際、Lmem を互換性のあるアドレッシング・モードのオペランドに置き換えます。
Xmem および Ymem	命令に Xmem と Ymem が含まれている場合、その命令はデータ・メモリへの 2 つの 16 ビット・アクセスを同時に実行できます。この命令を記述する際、Xmem と Ymem を互換性のあるオペランドに置き換えます。
Cmem	命令構文に Cmem が含まれている場合、その命令はデータ・メモリから 1 ワード（16 ビット）のデータにアクセスできます。この命令を記述する際、Cmem を互換性のあるオペランドに置き換えます。
Baddr	命令に Baddr が含まれている場合、その命令はアキュムレータ（AC0 ～ AC3）、補助レジスタ（AR0 ～ AR7）、または一時レジスタ（T0 ～ T3）内の 1 つまたは 2 つのビットにアクセスできます。Baddr をサポートする命令は、レジスタ・ビットのテスト / セット / クリア / 補数演算の各命令だけです。これらの命令を記述する際、Baddr を互換性のあるオペランドに置き換えます。

6.2 絶対アドレッシング・モード

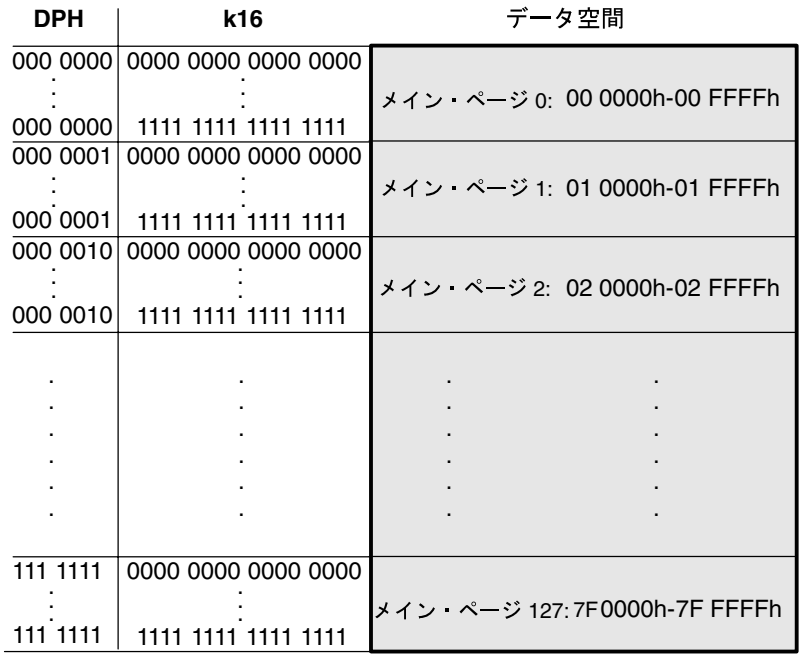
次の 3 つの絶対アドレッシング・モードが使用可能です。

アドレッシング・モード	説明	参照先
k16 絶対	このモードでは、DPH（拡張データ・ページ・レジスタの上位部分）と呼ばれる 7 ビットのレジスタ、および 16 ビットの符号なし定数を使用して、23 ビットのデータ空間アドレスを形成します。このモードを使用して、メモリ・ロケーションまたはメモリ・マップド・レジスタへのアクセスします。	P. 6-4
k23 絶対	このモードでは、すべてのアドレスを 23 ビットの符号なし定数として指定できます。このモードは、メモリ・ロケーションまたはメモリ・マップド・レジスタへのアクセスに使用できます。	P. 6-5
I/O 絶対	このモードでは、I/O アドレスを 16 ビットの符号なし定数として指定できます。このモードは、I/O 空間のロケーションにアクセスするためのモードです。	P. 6-6

6.2.1 k16 絶対アドレッシング・モード

k16 絶対アドレッシング・モードは、オペランド *abs16(#k16) を使用します。この k16 は、16 ビットの符号なし定数です。23 ビットのデータ空間アドレスを形成するために、DPH（拡張データ・ページ・レジスタの上位部分）と k16 を連結する方法を図 6-1 に示します。命令がこのアドレッシング・モードを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このモードを使用する命令は別の命令と並列に実行することはできません。

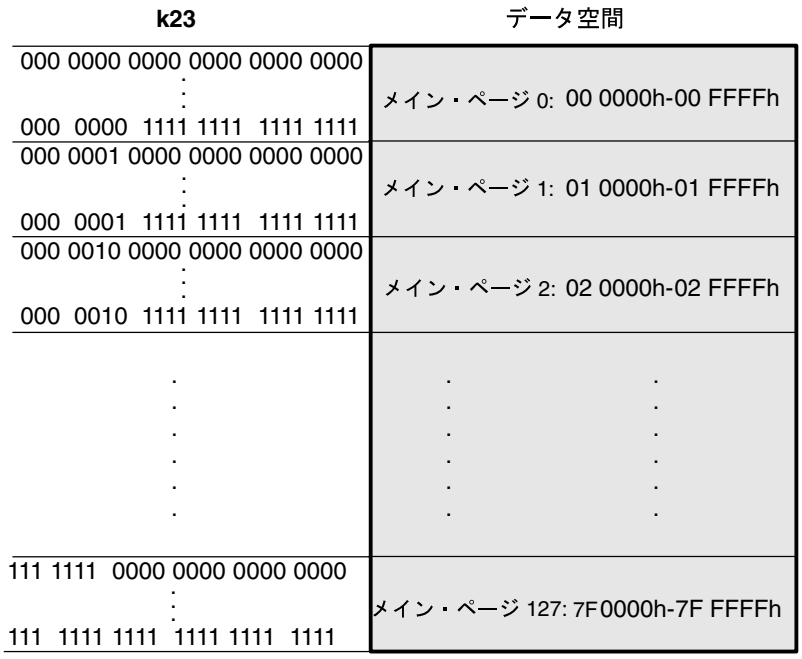
図 6-1.　　k16 絶対アドレッシング・モード



6.2.2　k23 絶対アドレッシング・モード

k23 絶対アドレッシング・モードは、オペランド *(#k23) を使用します。この k23 は、23 ビットの符号なし定数です。k23 を使用してデータ空間をアドレス指定する方法を図 6-2 に示します。このアドレッシング・モードを使用する命令は、定数を命令への 3 バイト拡張としてエンコードします（この 3 バイト拡張の最上位ビットは破棄されます)。この拡張のため、このモードを使用する命令は別の命令と並列に実行することはできません。

図 6-2.　　k23 絶対アドレッシング・モード

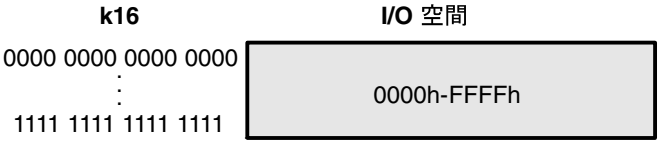


6.2.3　I/O 絶対アドレッシング・モード

代数表記命令セットを使用する場合、I/O 絶対アドレッシング・モードは、オペランド *port(#k16) を使用します。この k16 は、16 ビットの符号なし定数です。ニーモニック命令セットを使用する場合、I/O 絶対アドレッシング機能は port() オペランド修飾子により指定されます。16 ビットの符号なし定数を port() 修飾子の括弧で囲んでください。たとえば、port(#k16)（この場合、先頭にアスタリスク * は必要ありません）。

k16 を使用して I/O 空間をアドレス指定する方法を図 6-3 に示します。命令がこのアドレッシング・モードを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このモードを使用する命令は別の命令と並列に実行することはできません。

図 6-3.　　I/O 絶対アドレッシング・モード



6.3 直接アドレッシング・モード

次の直接アドレッシング・モードが使用可能です。

アドレッシング・モード	説明	参照先
DP 直接	このモードは、DPH（拡張データ・ページ・レジスタの上位部分）で指定されるメイン・データ・ページをデータ・ページ・レジスタ（DP）と合わせて使用します。このモードは、メモリ・ロケーションまたはメモリ・マップド・レジスタへのアクセスに使用されます。	P. 6-8
SP 直接	このモードは、SPH（拡張スタック・ポインタの上位部分）で指定されるメイン・データ・ページをデータ・スタック・ポインタ（SP）と合わせて使用します。このモードを使用して、データ・メモリ内のスタック値にアクセスします。	P. 6-10
レジスタ・ビット直接	このモードはオフセットを使用して、ビット・アドレスの指定します。このモードは、1つのレジスタ・ビットまたは隣接したレジスタ・ビットへのアクセスに使用されます。	P. 6-11
PDP 直接	このモードは、ペリフェラル・データ・ページ・レジスタ（PDP）とオフセットを使用して、I/O アドレスを指定します。このモードは、I/O 空間のロケーションへのアクセスに使用されます。	P. 6-12

DP 直接アドレッシング・モードと SP 直接アドレッシング・モードは、相互に排他的です。選択されるモードは、ステータス・レジスタ ST1_55 の CPL ビットによって異なります。

CPL	選択されるアドレッシング・モード
0	DP 直接アドレッシング・モード
1	SP 直接アドレッシング・モード

レジスタ・ビット直接アドレッシング・モードと PDP 直接アドレッシング・モードは、CPL ビットとは無関係です。

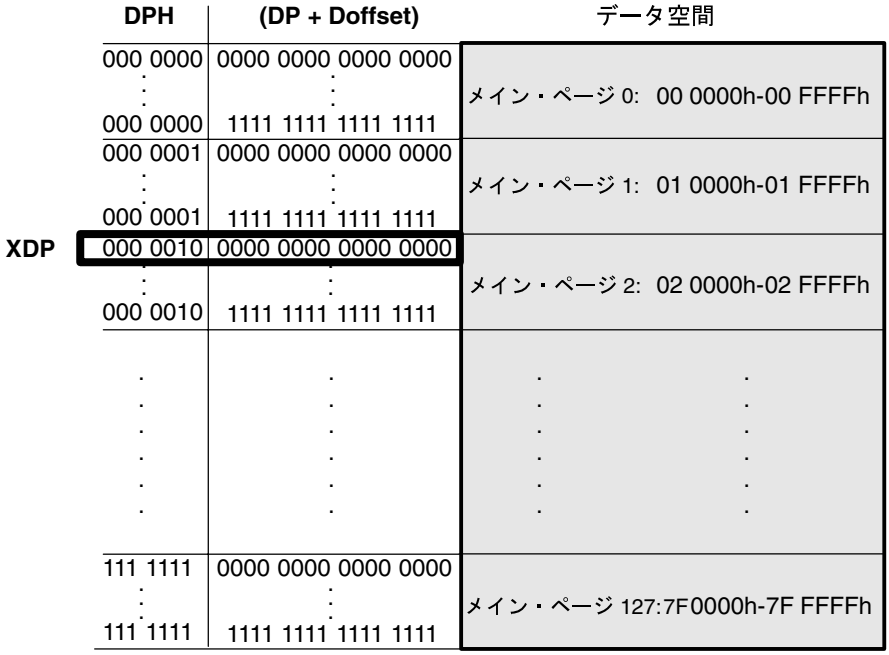
6.3.1 DP 直接アドレッシング・モード

DP 直接アドレッシング・モードの 23 ビット・アドレスの構成要素を図 6-4 に示します。DPH と呼ばれるレジスタから 7 つの最上位ビットが引き渡されます。これらのビットは、128 のメイン・データ・ページ（0 ～ 127）から 1 つ選択します。16 の最下位ビットは、次の 2 つの値の合計です。

- ❑ データ・ページ・レジスタ（DP）の値。DP は、メイン・データ・ページ内で 128 ワードのローカル・データ・ページのスタート・アドレスを識別します。このスタート・アドレスには、選択されたメイン・データ・ページ内の任意のアドレスを指定できます。
- ❑ アセンブラによって計算される 7 ビット・オフセット（Doffset）。この計算は、データ・メモリにアクセスしているか、またはメモリ・マップド・レジスタに（mmap() 修飾子を使用して）アクセスしているかによって異なります。この計算に関する詳細は、6.3.1.1 項を参照してください。

DPH と DP の連結は、拡張データ・ページ・レジスタ（XDP）と呼ばれます。DPH と DP は別々にロードできます。あるいは、XDP をロードする命令を使用できます。

図 6-4. DP 直接アドレッシング・モード



6.3.1.1 アセンブラによる DP 直接アドレッシング・モードの Doffset の計算

2 つのタイプの DP 直接アクセスに対してアセンブラが Doffset 値を計算する方法について表 6-2 で説明します。次の表の後に例を示します。

表 6-2. DP 直接アドレッシング・モードの Doffset の計算

アクセス先	Doffset 計算	説明
データ・メモリ	Doffset = (Daddr - .dp) & 7Fh	Daddr は、リード演算またはライト演算のための 16 ビット・ローカル・アドレスです。.dp は、.dp アセンブラ・ディレクティブで指定する値です（.dp は通常 DP と同じです）。& 記号は、ビットごとの AND 演算を識別します。
メモリ・マップド・レジスタ（mmap() 修飾子を使用）	Doffset = Daddr & 7Fh	Daddr は、リード演算またはライト演算のための 16 ビット・ローカル・アドレスです。& 記号は、ビットごとの AND 演算を識別します。.dp 値は使用されません。mmap() 修飾子を使用すると、データ・ページが 0 であるかのように CPU は動作します。

注 :
ローカル・アドレスは、メイン・データ・ページ内のアドレスです。23 ビットのデータ空間アドレスの 16 の LSB で表します。たとえば、ローカル・アドレス 0005h は、すべてのメイン・データ・ページにあります。

次のコード例では、データ・メモリへのアクセスに DP 直接アドレッシングを使用しています。

```
AMOV #03FFF0h, XDP   ; メイン・データ・ページは 03 です。実行時、
                     ; DP は FFF0h です。
.dp #0FFF0h          ; アセンブリ時、.dp は FFF0h です。
MOV @0FFF4h, T2      ; T2 にローカル・アドレス FFF4h の値をロードします。
```

アセンブラにより Doffset が計算されます。

Doffset = (Daddr - .dp) & 7Fh = (FFF4h – FFF0h) & 7Fh = 04h

Doffset は、命令 ***MOV @0FFF4h, T2*** でエンコードされます。実行時、23 ビットのデータ空間アドレスが生成されます。

23 ビット・アドレス = DPH:(DP + Doffset) = 03:(FFF0h + 0004h) = 03 FFF4h

次のコード例では、メモリ・マップド・レジスタ（MMR）へのアクセスに DP 直接アドレッシングを使用します。

```
MOV mmap(@AR0), T2   ; T2 に AR0 の値をロードします。
                     ; mmap() 修飾子は MMR へのアクセスを示します。
                     ; AR0 はデータ空間のアドレス 000010h にマップされます。
```

アセンブラにより Doffset が計算されます。

Doffset = Daddr & 7Fh = 0010h & 7Fh = 10h

Doffset は、命令 ***MOV mmap(@AR0), T2*** でエンコードされます。実行時、23 ビットのデータ空間アドレスが生成されます（DPH = DP = 0 であるかのように CPU は動作します）。

23 ビット・アドレス = DPH:(DP + Doffset) = 00:(0000h + 0010h) = 00 0010h

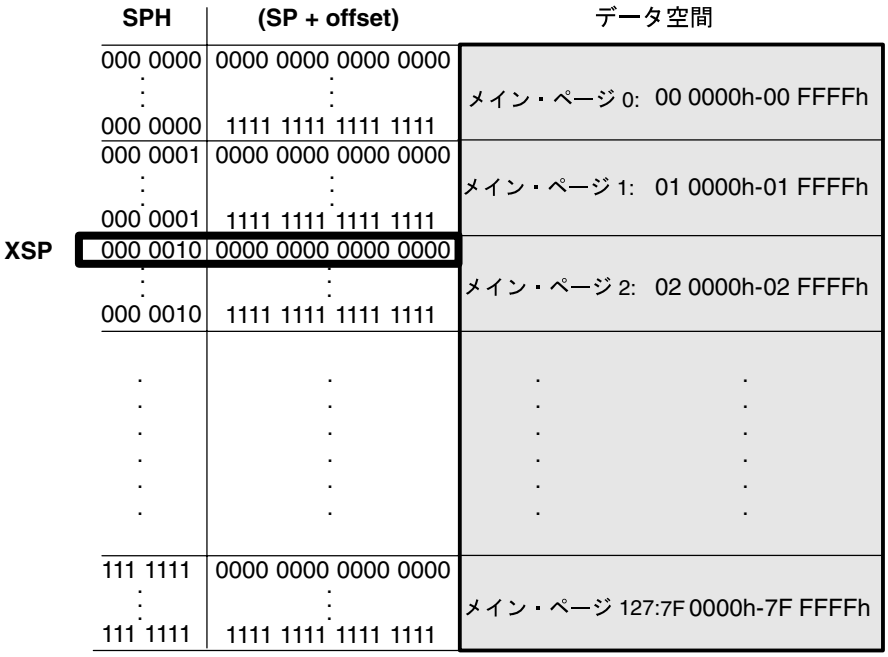
> **注:**
>
> **ニーモニック命令を使用する場合、**mmap() **を使用して修飾されるオペランドを囲みます。代数表記命令を使用する場合、**mmap() **はメモリ・マップド・レジスタのアクセスを実行する命令と並列に指定される命令修飾子です。**

6.3.2 SP 直接アドレッシング・モード

命令が SP 直接アドレッシング・モードを使用する場合、図 6-5 で示されるように 23 ビットのアドレスが形成されます。7 つの最上位ビットが SPH と呼ばれるレジスタで指定されます。16 の最下位ビットは、SP 値と命令で指定する 7 ビット・オフセットの合計です。オフセットは、0 ～ 127 の値です。SPH と SP の連結は、拡張データ・スタック・ポインタ（XSP）と呼ばれます。SPH と SP は別々にロードすることもでき、また XSP をロードする命令を使用することもできます。

最初のメイン・データ・ページでは、アドレス 00 0000h ～ 00 005Fh がメモリ・マップド・レジスタに予約されています。任意のデータ・スタックがメイン・データ・ページ 0 にある場合、そのページのアドレス 00 0060h ～ 00 FFFFh だけを使用します。

図 6-5.　SP 直接アドレッシング・モード

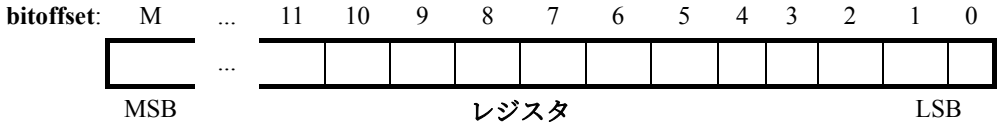


6.3.3　レジスタ・ビット直接アドレッシング・モード

レジスタ・ビット直接アドレッシング・モードでは、オペランドで指定するオフセット @bitoffset は、レジスタの最下位ビット（LSB）からのオフセットです（図 6-6 を参照）。たとえば、bitoffset が 0（ゼロ）の場合、レジスタの最下位ビット（LSB）がアドレス指定されます。bitoffset が 3 の場合は、レジスタのビット 3 がアドレス指定されます。

このモードをサポートする命令は、レジスタ・ビットのテスト / セット / クリア / 補数演算の各命令だけです。これらの命令を使用すると、アキュムレータ（AC0 ～ AC3）、補助レジスタ（AR0 ～ AR7）、および一時レジスタ（T0 ～ T3）のビットだけにアクセスできます。

図 6-6.　レジスタ・ビット直接アドレッシング・モード



注 :　ビット・アドレス M は、レジスタのサイズに応じて、39 または 15 になります。

6.3.4 PDP 直接アドレッシング・モード

命令が PDP 直接アドレッシング・モードを使用する場合、図 6-7 で示されるように 16 ビットの I/O アドレスが形成されます。9 ビットのペリフェラル・データ・ページ・レジスタ（PDP）は、512 のペリフェラル・データ・ページ（0 ～ 511）から 1 つ選択します。各ページは 128 ワードです（0 ～ 127）。命令中で 7 ビット・オフセット（Poffset）を指定して、特定のワードを選択します。たとえば、ページの最初のワードにアクセスするには、0（ゼロ）のオフセットを使用します。

図 6-7. PDP 直接アドレッシング・モード

PDP	Poffset	I/O 空間 (64K)
0000 0000 0 ⋮ 0000 0000 0	000 0000 ⋮ 111 1111	ペリフェラル・ページ 0: 0000h-007Fh
0000 0000 1 ⋮ 0000 0000 1	000 0000 ⋮ 111 1111	ペリフェラル・ページ 1: 0080h-00FFh
0000 0001 0 ⋮ 0000 0001 0	000 0000 ⋮ 111 1111	ペリフェラル・ページ 2: 0100h-017Fh
. . . . . .	. . . . . .	. . . . . .
1111 1111 1 ⋮ 1111 1111 1	000 0000 ⋮ 111 1111	ペリフェラル・ページ 511: FF80h-FFFFh

6.4 間接アドレッシング・モード

CPU は、次の間接アドレッシング・モードをサポートします。これらのモードは、リニア・アドレッシングまたはサーキュラ・アドレッシングに使用できます。

アドレッシング・モード	説明	参照先
AR 間接	このモードは、データを指定するために 8 つの補助レジスタ（AR0 ～ AR7）の 1 つを使用します。CPU が補助レジスタを使用してアドレスを生成する方法は、アクセスしている対象であるデータ空間（メモリまたはメモリ・マップド・レジスタ）、各レジスタ・ビット、または I/O 空間によって異なります。	6.4.1
デュアル AR 間接	このモードは、AR 間接アドレッシング・モードと同じアドレス生成プロセスを使用します。このモードは、2 つ以上のデータ・メモリ・ロケーションに同時にアクセスする命令で使用されます。	6.4.2
CDP 間接	このモードは、データを指定するために係数データ・ポインタ（CDP）を使用します。CPU が CDP を使用してアドレスを生成する方法は、アクセスしている対象であるデータ空間（メモリまたはメモリ・マップド・レジスタ）、各レジスタ・ビット、または I/O 空間によって異なります。	6.4.3
係数間接	このモードは、CDP 間接アドレッシング・モードと同じアドレス生成プロセスを使用します。このモードでは、デュアル AR 間接アドレッシング・モードを使用して 2 つのデータ・メモリ値にアクセスするのと同時にデータ・メモリの係数にアクセスできる命令をサポートできます。	6.4.4

6.4.1 AR 間接アドレッシング・モード

このモードは、データを指定するために補助レジスタ ARn（n = 0、1、2、3、4、5、6、または 7）を使用します。CPU が ARn を使用してアドレスを生成する方法は、アクセスのタイプによって異なります（表 6-3 を参照）。

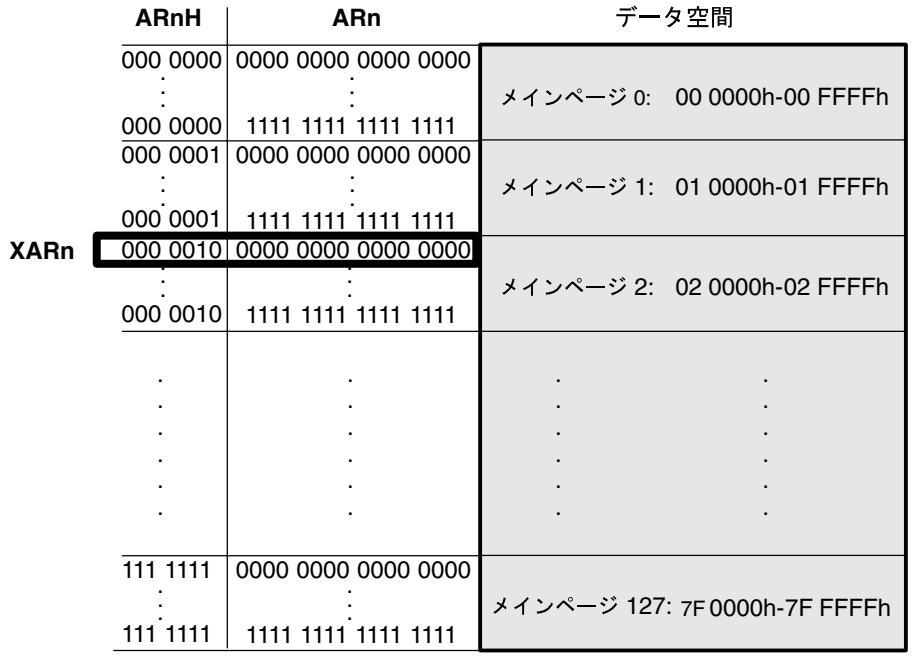
表 6-3. AR 間接アドレッシング・モードの補助レジスタ（ARn）の使用

アクセス先	ARn の内容
データ空間 （メモリまたはレジスタ）	23 ビット・アドレスの 16 の最下位ビット（LSB）。7 つの最上位ビット（MSB）は、拡張補助レジスタ XARn の上位部分である ARnH で指定されます。6.4.1.1 項を参照してください。
単一レジスタ・ビット （またはビット・ペア）	ビット数。6.4.1.2 項を参照してください。
I/O 空間	16 ビットの I/O アドレス。6.4.1.3 項を参照してください。

6.4.1.1 データ空間の AR 間接アクセス

AR 間接アドレッシング・モードのデータ空間アドレスを CPU が生成する方法を図 6-8 に示します （データ・メモリとメモリ・マップド・レジスタは両方ともデータ空間にマップされることに注意してください）。任意のアクセスに対して、補助レジスタ n（n = 0、1、2、3、4、5、6、または 7）には 16 の最下位ビットが用意されていて、関連するレジスタ ARnH には 7 つの最上位ビットが用意されています。ARnH と ARn の連結は、拡張補助レジスタ n (XARn) と呼ばれます。データ空間へのアクセスには、XARn をロードする命令を使用します。ARn は個別にロードできますが、ARnH はロードできません。

図 6-8.　AR 間接アドレッシング・モードを使用したデータ空間へのアクセス

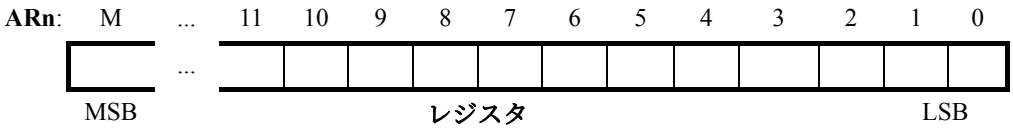


6.4.1.2　レジスタ・ビットの AR 間接アクセス

レジスタ・ビットにアクセスするために AR 間接アドレッシング・モードを使用する場合、選択される 16 ビットの補助レジスタ ARn にビット数が含まれます（図 6-9 を参照）。たとえば、AR2 に 0（ゼロ）が含まれる場合、AR2 はビット 0（レジスタの最下位ビット（LSB））を指します。

レジスタ・ビットのテスト / セット / クリア / 補数演算の命令だけが、レジスタ・ビットへの AR 間接アクセスをサポートします。これらの命令を使用すると、アキュムレータ（AC0 ～ AC3）、補助レジスタ（AR0 ～ AR7）、および一時レジスタ（T0 ～ T3）のビットだけにアクセスできます。

図 6-9.　AR 間接アドレッシング・モードを使用したレジスタ・ビットへのアクセス

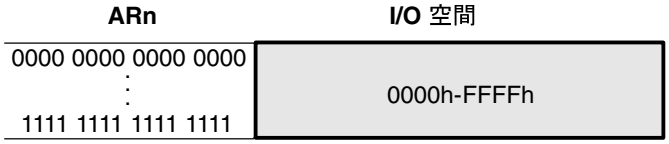


注 : ビット・アドレス M は、レジスタのサイズに応じて、39 または 15 になります。

6.4.1.3 I/O 空間への AR 間接アクセス

I/O 空間のワードは、16 ビット・アドレスでアクセスされます。I/O 空間にアクセスするために AR 間接アドレッシング・モードを使用する場合、選択される 16 ビットの補助レジスタ ARn に完全な I/O アドレスが含まれます。

図 6-10. AR 間接アドレッシング・モードを使用した I/O 空間へのアクセス



6.4.1.4 AR 間接オペランド

このモードで使用可能なアドレッシング・モードのオペランドのタイプは、ステータス・レジスタ ST2_55 の ARMS ビットによって異なります。

ARMS	DSP モードまたは制御モード
0	DSP モード。CPU は、DSP モードのオペランド（表 6-4 を参照）を使用できます。これらのオペランドは、DSP 集約型アプリケーションを効率的に実行します。
1	制御モード。CPU は、制御モードのオペランド（表 6-5 を参照）を使用できます。これらのオペランドは、制御システム・アプリケーションに最適なコード・サイズを可能にします。

AR 間接アドレッシング・モードで使用可能な DSP モードのオペランドについて表 6-4 で説明します。制御モードのオペランドについて表 6-5 で説明します。すべての AR 間接オペランドについて表 6-6 でまとめます。ここでは、オペランドが補助レジスタを変更するかどうか、またこの変更のタイミングが命令のアドレスが生成される前か後かに基づいてまとめています。これらの表を参照する際は、次のことを確認してください。

- ポインタの変更およびアドレスの生成は、ステータス・レジスタ ST2_55 のポインタ構成に応じてリニアまたはサーキュラになります。選択されているポインタに対してサーキュラ・アドレッシングがアクティブである場合にのみ、適切な 16 ビットのバッファ・スタート・アドレス・レジスタ（BSA01、BSA23、BSA45、または BSA67）の内容が追加されます。

- インクリメントとデクリメントは、16 ビットのポインタに対してのみ行われます。データのメイン・データ・ページへのアドレス指定は、拡張レジスタ（ARnH）の値を変更することなく行うことはできません。ARnH を変更するには、23 ビットのレジスタ XARn のすべてにライトする必要があります。

注：

FFFFh を超えたインクリメントまたは 0000h を超えたデクリメントを行うとポインタ値が初期状態に戻ってしまいます。この動作はサポートされている機能ではないので、利用してはいけません。また、サーキュラ・アドレッシング時、BSAxx を加算する場合、FFFFh を超えたアドレスをインクリメントしてはなりません。

表 6-4. AR 間接アドレッシング・モードの DSP モード（ARMS = 0）のオペランド

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*ARn	ARn は変更されません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*ARn+	ARn はアドレスの生成後にインクリメントされます。†‡	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*ARn-	ARn はアドレスの生成後にデクリメントされます。§¶	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

‡ 32 ビット /2 ビット演算の場合：ARn = ARn + 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-4. AR 間接アドレッシング・モードの DSP モード（ARMS = 0）のオペランド（続き）

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*+ARn	ARn はアドレスの生成前にインクリメントされます。†‡	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*-ARn	ARn はアドレスの生成前にデクリメントされます。§¶	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*(ARn + T0/AR0)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn に加算されます。 C54CM = 0 の場合 : ARn = ARn + T0 C54CM = 1 の場合 : ARn = ARn + AR0	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*(ARn - T0/AR0)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn から減算されます。 C54CM = 0 の場合 : ARn = ARn - T0 C54CM = 1 の場合 : ARn = ARn - AR0	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

‡ 32 ビット /2 ビット演算の場合：ARn = ARn + 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-4. AR 間接アドレッシング・モードの DSP モード（ARMS = 0）のオペランド（続き）

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*ARn(T0/AR0)	ARn は変更されません。ARn はベース・ポインタとして使用されます。T0 または AR0 の 16 ビット符号付き定数がそのベース・ポインタからのオフセットとして使用されます。 C54CM = 0 の場合、T0 が使用されます。 C54CM = 1 の場合、AR0 が使用されます。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*(ARn + T0B/AR0B)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn に加算されます。 C54CM = 0 の場合 : ARn = ARn + T0 C54CM = 1 の場合 : ARn = ARn + AR0 （いずれの加算もビット・リバース・アドレッシングを作成するためにリバース・キャリー伝搬を使用して行われます） 注：このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAxx）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*(ARn - T0B/AR0B)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn から減算されます。 C54CM = 0 の場合 : ARn = ARn - T0 C54CM = 1 の場合 : ARn = ARn - AR0 （いずれの減算もリバース・キャリー伝搬を使用して行われます） 注：このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAxx）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

‡ 32 ビット /2 ビット演算の場合：ARn = ARn + 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-4. AR 間接アドレッシング・モードの DSP モード（ARMS = 0）のオペランド（続き）

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*(ARn + T1)	T1 の 16 ビット符号付き定数がアドレスの生成後に ARn に加算されます。 ARn = ARn + T1	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*(ARn - T1)	T1 の 16 ビット符号付き定数がアドレスの生成後に ARn から減算されます。 ARn = ARn - T1	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*ARn(T1)	ARn は変更されません。ARn はベース・ポインタとして使用されます。T1 の 16 ビット符号付き定数がそのベース・ポインタからのオフセットとして使用されます。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

‡ 32 ビット /2 ビット演算の場合：ARn = ARn + 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-4. AR 間接アドレッシング・モードの DSP モード（ARMS = 0）のオペランド（続き）

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*ARn(#K16)	ARn は変更されません。ARn はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。 注：命令がこのオペランドを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr）
*+ARn(#K16)	16 ビット符号付き定数（K16）がアドレスの生成前に ARn に加算されます。 注：命令がこのオペランドを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

‡ 32 ビット /2 ビット演算の場合：ARn = ARn + 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-5. AR 間接アドレッシング・モードの制御モード（ARMS = 1）のオペランド

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*ARn	ARn は変更されません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*ARn+	ARn はアドレスの生成後にインクリメントされます。†‡	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*ARn-	ARn はアドレスの生成後にデクリメントされます。§¶	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*(ARn + T0/AR0)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn に加算されます。 C54CM = 0 の場合 : ARn = ARn + T0 C54CM = 1 の場合 : ARn = ARn + AR0	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、32 ビット /2 ビット演算：ARn = ARn + 2

‡ 1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス 2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-5. AR 間接アドレッシング・モードの制御モード（ARMS = 1）のオペランド（続き）

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*(ARn - T0/AR0)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn から減算されます。 C54CM = 0 の場合 : ARn = ARn - T0 C54CM = 1 の場合 : ARn = ARn - AR0	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*ARn(T0/AR0)	ARn は変更されません。ARn はベース・ポインタとして使用されます。T0 または AR0 の 16 ビット符号付き定数がそのベース・ポインタからのオフセットとして使用されます。 C54CM = 0 の場合、T0 が使用されます。 C54CM = 1 の場合、AR0 が使用されます。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*ARn(#K16)	ARn は変更されません。ARn はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。 注：命令がこのオペランドを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、32 ビット /2 ビット演算：ARn = ARn + 2

‡ 1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス 2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-5. AR 間接アドレッシング・モードの制御モード（ARMS = 1）のオペランド（続き）

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*+ARn(#K16)	16 ビット符号付き定数（K16）がアドレスの生成前に ARn に加算されます。 ARn = ARn + K16 注：命令がこのオペランドを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr）
*ARn(short(#k3))	ARn は変更されません。ARn はベース・ポインタとして使用されます。3 ビット符号なし定数（k3）がそのベース・ポインタからのオフセットとして使用されます。k3 の範囲は、1 ～ 7 です。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）

† 16 ビット /1 ビット演算の場合：ARn = ARn + 1、32 ビット /2 ビット演算：ARn = ARn + 2

‡ 1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス 2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：ARn = ARn -1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：ARn = ARn -2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-6. 間接オペランドの要約

未変更	変更後	変更前
	AR 間接、DSP モード (ARMS = 0)	
*ARn *ARn(T0/AR0) *ARnN(T1) *ARn(#K16)	*ARn+ *ARn- *(ARn + T0/AR0) *(ARn - T0/AR0) *(ARn + T0B/AR0B) *(ARn - T0B/AR0B) *(ARn + T1) *(ARn - T1)	*+ARn *-ARn *+ARn(#K16)
	AR 間接、制御モード (ARMS = 1)	
*ARn *ARn(T0/AR0) *ARn(#K16) *ARn(short(#k3))	*ARn+ *ARn- *(ARn + T0/AR0) *(ARn - T0/AR0) *(ARn + T0B/AR0B) *(ARn - T0B/AR0B) *(ARn + T1) *(ARn - T1)	*+ARn(#K16)

6.4.2 デュアル AR 間接アドレッシング・モード

デュアル AR 間接アドレッシング・モードを使用すると、8 つの補助レジスタ（AR0 ～ AR7）で 2 つのデータ・メモリ・アクセスを行うことができます。データ空間への単一 AR 間接アクセスと同様（6.4.1.1 項を参照）、CPU は拡張補助レジスタを使用してそれぞれの 23 ビット・アドレスを作成します。2 つのアクセスのそれぞれにリニア・アドレッシングまたはサーキュラ・アドレッシングを使用できます。

デュアル AR 間接アドレッシング・モードは、次の操作に使用できます。

- 2 つの 16 ビット・データ・メモリ・アクセスを行う命令の実行。この場合、命令構文に Xmem と Ymem の 2 つのデータ・メモリ・オペランドが指定されます。次に例を示します。

  ADD Xmem, Ymem, ACx

- ❑ 2 つの命令を並列に実行。この場合、2 つの命令はそれぞれ、単一のメモリ値にアクセスする必要があります。単一メモリ値は命令構文で Smem または Lmem として指定されます。次に例を示します。

  MOV Smem, dst
  || AND Smem, src, dst

  最初の命令のオペランドは Xmem オペランドとして処理され、2 番目の命令のオペランドは Ymem オペランドとして処理されます。

使用可能なデュアル AR 間接オペランドは、AR 間接オペランドのサブセットです。ARMS ステータス・ビットによる使用可能なデュアル AR 間接オペランド・セットへの影響はありません。

注:

デュアル・オペランドに対し更新の仕方が異なる同じ補助レジスタを使用しているコードは、アセンブラによって除去されます。2 つのオペランドのいずれかが ARn を変更しない場合のみ、デュアル・オペランドに同じ ARn を使用できます。

6.4.2.1 デュアル AR 間接オペランド

AR 間接アドレッシング・モードで使用可能なオペランドについて表 6-7 で説明します。次のことに注意してください。

- ❑ ポインタの変更およびアドレスの生成は、ステータス・レジスタ ST2_55 のポインタ設定に応じてリニアまたはサーキュラになります。選択されているポインタに対してサーキュラ・アドレッシングがアクティブである場合にのみ、適切な 16 ビットのバッファ・スタート・アドレス・レジスタ（BSA01、BSA23、BSA45、または BSA67）の内容が追加されます。
- ❑ インクリメントとデクリメントは、16 ビットのポインタに対してのみ行われます。データのメイン・データ・ページへのアドレス指定は、拡張レジスタ（ARnH）の値を変更することなく行うことはできません。ARnH を変更するには、23 ビットのレジスタ XARn のすべてにライトする必要があります。

注:

FFFFh を超えたインクリメントまたは 0000h を超えたデクリメントを行うとポインタ値が初期状態に戻ってしまいます。この動作はサポートされている機能ではないので、利用してはいけません。また、サーキュラ・アドレッシング時、BSAxx を加算する場合、FFFFh を超えたアドレスをインクリメントしてはなりません。

表 6-7. デュアル AR 間接オペランド

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*ARn	ARn は変更されません。	データ・メモリ（Smem、Lmem、Xmem、Ymem）
*ARn+	ARn はアドレスの生成後にインクリメントされます。†‡	データ・メモリ（Smem、Lmem、Xmem、Ymem）
*ARn-	ARn はアドレスの生成後にデクリメントされます。§¶	データ・メモリ（Smem、Lmem、Xmem、Ymem）
*(ARn + T0/AR0)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn に加算されます。 C54CM = 0 の場合 : ARn = ARn + T0 C54CM = 1 の場合 : ARn = ARn + AR0	データ・メモリ（Smem、Lmem、Xmem、Ymem）
*(ARn - T0/AR0)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に ARn から減算されます。 C54CM = 0 の場合 : ARn = ARn - T0 C54CM = 1 の場合 : ARn = ARn - AR0	データ・メモリ（Smem、Lmem、Xmem、Ymem）
*ARn(T0/AR0)	ARn は変更されません。ARn はベース・ポインタとして使用されます。T0 または AR0 の 16 ビット符号付き定数がそのベース・ポインタからのオフセットとして使用されます。 C54CM = 0 の場合、T0 が使用されます。 C54CM = 1 の場合、AR0 が使用されます。	データ・メモリ（Smem、Lmem、Xmem、Ymem）
*(ARn + T1)	T1 の 16 ビット符号付き定数がアドレスの生成後に ARn に加算されます。 ARn = ARn + T1	データ・メモリ（Smem、Lmem、Xmem、Ymem）

† 16 ビット演算の場合：ARn = ARn + 1
‡ 32 ビット演算の場合：ARn = ARn + 2
§ 16 ビット演算の場合：ARn = ARn -1
¶ 32 ビット演算の場合：ARn = ARn -2

6.4.3 CDP 間接アドレッシング・モード

このモードは、データを指定するために、係数データ・ポインタ（CDP）を使用します。CPU が CDP を使用してアドレスを生成する方法は、アクセスのタイプによって異なります（表 6-8 を参照）。

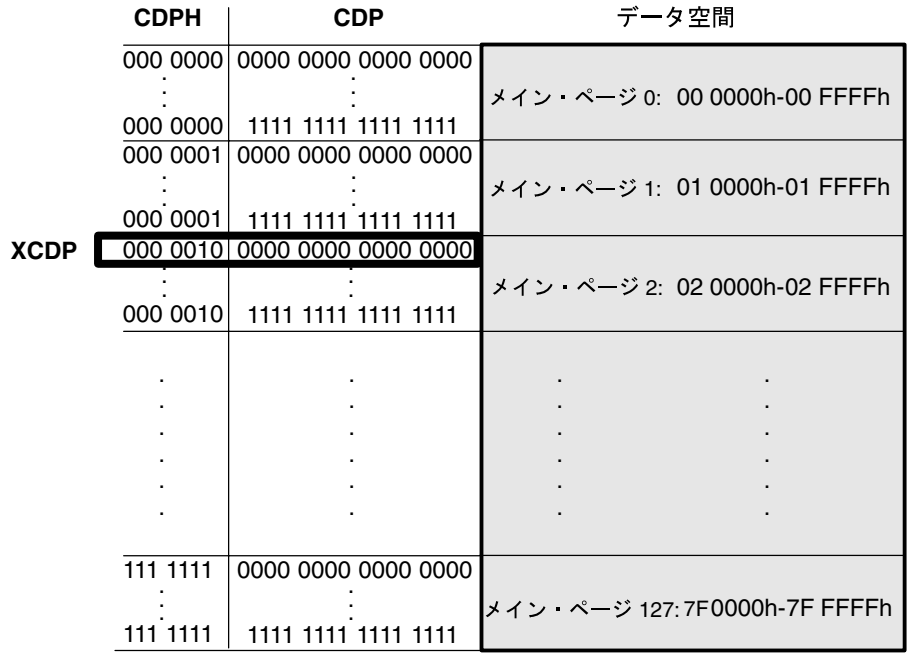
表 6-8. CDP 間接アドレッシング・モードの係数データ・ポインタ（CDP）の使用

アクセス先	CDP の内容
データ空間 （メモリまたはレジスタ）	23 ビット・アドレスの 16 の最下位ビット（LSB）。7 つの最上位ビット（MSB）は、拡張係数データ・ポインタ（XCDP）の上位部分である CDPH で指定されます。6.4.3.1 項を参照してください。
単一レジスタ・ビット （またはビット・ペア）	ビット数。6.4.3.2 項を参照してください。
I/O 空間	16 ビットの I/O アドレス。6.4.3.3 項を参照してください。

6.4.3.1 データ空間の CDP 間接アクセス

CDP 間接アドレッシング・モードのデータ空間アドレスを CPU が生成する方法を図 6-11 に示します（データ・メモリとメモリ・マップド・レジスタは両方ともデータ空間にマップされることに注意してください）。CDPH には 7 つの最上位ビットが用意されていて、係数データ・ポインタ（CDP）には 16 最下位ビットが用意されています。CDPH と CDP の連結は、拡張係数データ・ポインタ（XCDP）と呼ばれます。

図 6-11. CDP 間接アドレッシング・モードを使用したデータ空間へのアクセス

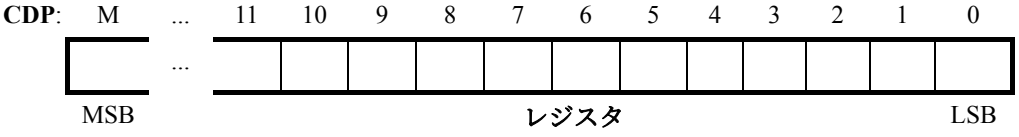


6.4.3.2 レジスタ・ビットの CDP 間接アクセス

レジスタ・ビットにアクセスするために CDP 間接アドレッシング・モードが使用される場合、CDP はビット数を含みます。たとえば、CDP が 0（ゼロ）が含む場合、CDP はビット 0（レジスタの最下位ビット（LSB））を指します。

レジスタ・ビットのテスト / セット / クリア / 補数演算の命令のみが、レジスタ・ビットへの CDP 間接アクセスをサポートします。これらの命令を使用すると、アキュムレータ（AC0 ～ AC3）、補助レジスタ（AR0 ～ AR7）、および一時レジスタ（T0 ～ T3）のビットだけにアクセスできます。

図 6-12. CDP 間接アドレッシング・モードを使用したレジスタ・ビットへのアクセス

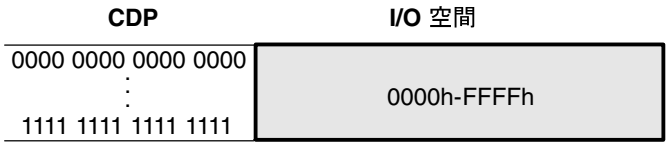


注 : ビット・アドレス M は、レジスタのサイズに応じて、39 または 15 になります。

6.4.3.3 I/O 空間の CDP 間接アクセス

I/O 空間のワードは、16 ビット・アドレスでアクセスされます。CDP 間接アドレッシング・モードを使用して I/O 空間にアクセスする場合、16 ビットの CDP に完全な I/O アドレスが含まれます。

図 6-13. CDP 間接アドレッシング・モードを使用した I/O 空間へのアクセス



6.4.3.4 CDP 間接オペランド

CDP 間接アドレッシング・モードに使用可能なオペランドについて表 6-9 で説明します。次のことに注意してください。

- ポインタの変更およびアドレスの生成は、ステータス・レジスタ ST2_55 のポインタ設定に応じてリニアまたはサーキュラになります。サーキュラ・アドレッシングが CDP に対してアクティブである場合にのみ、16 ビットのバッファ・スタート・アドレス・レジスタ BSAC の内容が追加されます。
- インクリメントとデクリメントは、16 ビットのポインタに対してのみ行われます。データのメイン・データ・ページへのアドレス指定は、拡張レジスタ（CDPH）の値を変更することなく行うことはできません。

注 :

FFFFh を超えたインクリメントまたは 0000h を超えたデクリメントを行うとポインタ値が初期状態に戻ってしまいます。この動作はサポートされている機能ではないので、利用してはいけません。また、サーキュラ・アドレッシング時、BSAC を加算する場合、FFFFh を超えたアドレスをインクリメントしてはなりません。

表 6-9. CDP 間接オペランド

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*CDP	CDP は変更されません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*CDP+	CDP はアドレスの生成後にインクリメントされます。†‡	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*CDP-	CDP はアドレスの生成後にデクリメントされます。§¶	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr） I/O 空間（Smem）
*CDP(#K16)	CDP は変更されません。CDP はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。 注：命令がこのオペランドを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr）

† 16 ビット /1 ビット演算の場合：CDP = CDP + 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

‡ 32 ビット /2 ビット演算の場合：CDP = CDP + 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：CDP = CDP - 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：CDP = CDP - 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

表 6-9. CDP 間接オペランド（続き）

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*+CDP(#K16)	16 ビット符号付き定数（K16）がアドレスの生成前に CDP に加算されます。 CDP = CDP + K16 注：命令がこのオペランドを使用する場合、定数はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。	データ・メモリ（Smem、Lmem） メモリ・マップド・レジスタ（Smem、Lmem） レジスタ・ビット（Baddr）

† 16 ビット /1 ビット演算の場合：CDP = CDP + 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

‡ 32 ビット /2 ビット演算の場合：CDP = CDP + 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

§ 16 ビット /1 ビット演算の場合：CDP = CDP - 1、1 ビット演算：レジスタの 1 つのビットをリードまたは変更するレジスタ・ビット・アクセス

¶ 32 ビット /2 ビット演算の場合：CDP = CDP - 2、2 ビット演算：レジスタのビット・ペアをリードまたは変更するレジスタ・ビット・アクセス

6.4.4 係数間接アドレッシング・モード

このモードは、データ空間アクセスに CDP 間接アドレッシング・モードと同じアドレス生成プロセスを使用します。係数間接アドレッシング・モードは、選択メモリの移動 / 初期化構文、および次の算術命令によりサポートされています。

- ❏ FIR フィルタ
- ❏ 乗算
- ❏ 積和演算
- ❏ 積差演算
- ❏ デュアル乗算、デュアル積和演算およびデュアル積差演算

データにアクセスするために係数間接アドレッシング・モードを使用する命令は主にサイクルごとに 3 つのメモリ・オペランドで演算を実行する命令です。これらのオペランドのうち 2 つ（Xmem と Ymem）は、デュアル AR 間接アドレッシング・モードでアクセスされます。3 番目のオペランド（Cmem）は、係数間接アドレッシング・モードでアクセスされます。Cmem オペランドは、CPU の BB バス上で伝送されます（1.6 節「アドレス・バスおよびデータ・バス」を参照）。

次の命令構文を考えてみましょう。1 つのサイクルで、2 つの乗算が並列に実行されます。1 つのメモリ・オペランド（Cmem）は 2 つの乗算に対して共通ですが、デュアル AR 間接オペランド（Xmem と Ymem）は乗算での他の値に対して使用されます。

```
MPY Xmem, Cmem, ACx
:: MPY Ymem, Cmem, ACy
```

1 つのサイクルで（上記の例のように）3 つのメモリ値にアクセスするためには、Cmem で参照される値は Xmem と Ymem の値を含むメモリ・バンクとは異なるメモリ・バンク内になければなりません。

6.4.4.1 BB バスに関する重要事項

係数間接アドレッシング・モードを使用する場合は、次の BB バスに関する重要事項を確認してください。

- ❑ 次に示す命令は Cmem オペランドにアクセスしますが、BB バスを使用しないで 16 ビットまたは 32 ビットの Cmem オペランドをフェッチします。

命令構文	Cmem アクセスの説明	Cmem へのアクセスに使用するバス
MOV Cmem, Smem	Cmem からの 16 ビット・リード	DB
MOV Smem, Cmem	Cmem への 16 ビット・ライト	EB
MOV Cmem, dbl(Lmem)	Cmem からの 32 ビット・リード	最上位ワード（MSW）の CB 最下位ワード（LSW）の DB
MOV dbl(Lmem), Cmem	Cmem への 32 ビット・ライト	MSW の FB LSW の EB

- ❑ BB バスは、外部メモリには接続されていません。Cmem オペランドが BB を経由してアクセスされる場合は、そのオペランドは内部メモリ内になければなりません。

6.4.4.2 係数間接オペランド

係数間接アドレッシング・モードに使用可能なオペランドについて表 6-10 で説明します。次のことに注意してください。

- ❑ ポインタの変更およびアドレスの生成は、ステータス・レジスタ ST2_55 のポインタ設定に応じてリニアまたはサーキュラになります。サーキュラ・アドレッシングが CDP に対してアクティブである場合にのみ、16 ビットのバッファ・スタート・アドレス・レジスタ BSAC の内容が追加されます。

- ❑ インクリメントとデクリメントは、16 ビットのポインタに対してのみ行われます。データのメイン・データ・ページへのアドレス指定は、拡張レジスタ（CDPH）の値を変更することなく行うことはできません。

注：

1) FFFFh **を超えたインクリメントまたは** 0000h **を超えたデクリメントを行うとポインタ値が初期状態に戻ってしまいます。この動作はサポートされている機能ではないので、利用してはいけません。また、サーキュラ・アドレッシング時、**BSAC **を加算する場合、**FFFFh **を超えたアドレスをインクリメントしてはなりません。**

2) **代数表記命令を使用する場合、構文要素** coef() **を使用して、係数間接オペランドを囲む必要があります（**6.4.4.3 **項を参照）。ニーモニック命令を使用する場合、表** 6-10 **に示すオペランドを使用できます。**

表 6-10. 係数間接オペランド

オペランド	ポインタの変更	サポートされるアクセス・タイプ
*CDP	CDP は変更されません。	データ・メモリ
*CDP+	CDP はアドレスの生成後にインクリメントされます。 16 ビット演算の場合：CDP = CDP + 1 32 ビット演算の場合：CDP = CDP + 2	データ・メモリ
*CDP-	CDP はアドレスの生成後にデクリメントされます。 16 ビット演算の場合：CDP = CDP - 1 32 ビット演算の場合：CDP = CDP - 2	データ・メモリ
*(CDP + T0/AR0)	T0 または AR0 の 16 ビット符号付き定数がアドレスの生成後に CDP に加算されます。 C54CM = 0 の場合 : CDP = CDP + T0 C54CM = 1 の場合 : CDP = CDP + AR0	データ・メモリ

6.4.4.3 代数表記命令の係数間接オペランドに必要な coef()

代数表記命令を使用する場合は、各係数間接（Cmem）オペランドを coef() 構文要素で囲む必要があります。たとえば、次の代数表記命令の構文があると仮定します。

ACx = ACx + (Smem * Cmem)

ACx = AC0 および Smem = *AR0 と仮定し、Cmem に *CDP を使用するものとします。命令は次のようになります。

```
AC0 = AC0 + (*AR0 * coef(*CDP))
```

6.5 データ・メモリのアドレッシング

絶対、直接、間接のアドレッシングは、データ・メモリの値をアドレス指定するために使用されます。

アドレッシング・タイプ	参照先
絶対	このページ
直接	P. 6-37
間接	P. 6-38

6.5.1 絶対アドレッシング・モードを使用したデータ・メモリのアドレッシング

k16 絶対オペランド *abs16(#k16) または k23 絶対オペランド *(#k23) は、次のいずれかの構文要素を持つ任意の命令のデータ・メモリにアクセスするために使用できます。

Smem　1 ワード（16 ビット）のデータを表します。

Lmem　2 つの隣接したワード（32 ビット）のデータを表します。

これらのオペランドの使用例を表 6-11 と表 6-12 に示します。

注:

マルチバイト拡張のため、*abs16(#k16) または *(#k23) を使用する命令は別の命令と並列に実行することはできません

- **命令が *abs16(#k16) を使用する場合、定数 k16 はその命令に 2 バイト拡張でエンコードされます。**
- **命令が *(#k23) を使用する場合、定数 k23 はその命令に 3 バイト拡張でエンコードされます。**

表 6-11.　*abs16(#k16) を使用したデータ・メモリのアクセス

構文例	命令例	生成されるアドレス（DPH = 3 の場合）	説明
MOV Smem, dst	MOV *abs16(#2002h), T2	DPH:k16 = 03 2002h	CPU はアドレス 03 2002h の値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*abs16(#2002h)), pair(T2)	DPH:k16 = 03 2002h (DPH:k16) + 1 = 03 2003h	CPU はアドレス 03 2002h と 03 2003h の値をリードし、これらの値を T2 と T3 にそれぞれコピーします。

表 6-12. *(#k23) を使用したデータ・メモリのアクセス

構文例	命令例	生成されるアドレス	説明
MOV Smem, dst	MOV *(#032002h), T2	k23 = 03 2002h	CPU はアドレス 03 2002h の値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(#032002h)), pair(T2)	k23 = 03 2002h k23 + 1 = 03 2003h	CPU はアドレス 03 2002h と 03 2003h の値をリードし、これらの値を T2 と T3 にそれぞれコピーします。

6.5.2 直接アドレッシング・モードを使用したデータ・メモリのアドレッシング

次のいずれかの構文要素を持つ任意の命令のデータ・メモリにアクセスするために直接アドレッシング・モードを使用することができます。

Smem　1 ワード（16 ビット）のデータを表します。

Lmem　2 つの隣接したワード（32 ビット）のデータを表します。

CPL ビットが 0（ゼロ）の場合、DP 直接オペランド (@Daddr) を使用できます。CPL ビットが 1 の場合、SP 直接オペランド *SP(offset) を使用できます。これらのオペランドを使用してデータ・メモリにアクセスする例を表 6-13 と表 6-14 に示します。

DP 直接アドレッシング・モードの場合、アセンブラはアドレスの Doffset を次のように計算します。

Doffset = (Daddr - .dp) & 7Fh

ここで、.dp は .dp アセンブラ・ディレクティブで割り当てられる値で、& はビットごとの AND 演算を示します。.dp ディレクティブと Doffset 計算の例は、6.3.1.1 項を参照してください。表 6-13 に示されているように、DP = .dp の場合、Doffset は Daddr と等しくなります（この場合は両方とも 0005h）。

表 6-13. @Daddr を使用したデータ・メモリのアクセス

構文例	命令例	生成されるアドレス（DPH = 3、DP = .dp = 0 の場合）	説明
MOV Smem, dst	MOV @0005h, T2	DPH:(DP + Doffset) = 03:(0000h + 0005h)= 03 0005h	CPU はアドレス 03 0005h の値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(@0005h), pair(T2)	DPH:(DP + Doffset) = 03 0005h DPH:(DP + Doffset – 1) = 03 0004h	CPU はアドレス 03 0005h と 03 0004h の値をリードし、これらの値を T2 と T3 にそれぞれコピーします。ロング・ワードのアラインメント・ルールに従って、先頭の偶数アドレスから 2 番目のワードがリードされます。

表 6-14. *SP(offset) を使用したデータ・メモリのアクセス

構文例	命令例	生成されるアドレス（SP = FF00h および SPH = 0 の場合）	説明
MOV Smem, dst	MOV *SP(5), T2	SPH:(SP + offset) = 00 FF05h	CPU はアドレス 00 FF05h の値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*SP(5)), pair(T2)	SPH:(SP + offset) = 00 FF05h SPH:(SP + offset - 1) = 00 FF04h	CPU はアドレス 00 FF05h と 00 FF04h の値をリードし、これらの値を T2 と T3 にそれぞれコピーします。ロング・ワードのアラインメント・ルールに従って、先頭の偶数アドレスから 2 番目のワードがリードされます。

6.5.3 間接アドレッシング・モードを使用したデータ・メモリのアドレッシング

データ・メモリにアクセスするために間接オペランドを使用したい場合、特定の命令に対して使用できるオペランドを確認しておく必要があります。データ･メモリへの間接アクセスをサポートする命令構文は、表 6-15 の構文要素のいずれかを含みます。この表の一番右の列には、データ・メモリへのアクセスに対して構文要素の代わりに使用できる間接オペランドが示されています。使用できる AR 間接オペランドは、ARMS ビットで選択されているモードが DSP モードか制御モードかによって異なります（6.4.1.4 項を参照）。表 6-15 に記述されているオペランドの詳細は、表以下に続く項を参照してください。データ・メモリにおけるロング･ワードのアラインメントの詳細は、3.3.2 項「データ・タイプ」を参照してください。

表 6-15. 間接オペランドの選択（データ・メモリのアクセス）

構文要素	説明	使用可能な間接オペランド
Smem または Lmem	Smem は 1 ワード（16 ビット）のデータを表します。Lmem は 2 つの隣接したワード（32 ビット）のデータを表します。	AR 間接アドレッシング・モード : DSP モード (ARMS = 0): / 制御モード (ARMS = 1): *ARn / *ARn *ARn+ / *ARn+ *ARn- / *ARn- *+ARn *-ARn *(ARn + T0/AR0) / *(ARn + T0/AR0) *(ARn - T0/AR0) / *(ARn - T0/AR0) *ARn(T0/AR0) / *ARn(T0/AR0) *(ARn + T0B/AR0B) *(ARn - T0B/AR0B) *(ARn + T1) *(ARn - T1) *ARn(T1) *ARn(#K16) / *ARn(#K16) *+ARn(#K16) / *+ARn(#K16) *ARn(short(#k3)) CDP 間接アドレッシング・モード *CDP *CDP+ *CDP- *CDP(#k16) *+CDP(#k16)
Xmem または Ymem	1 ワード（16 ビット）のデータ	デュアル AR 間接アドレッシング・モード *ARn *ARn+ *ARn- *(ARn + T0/AR0) *(ARn - T0/AR0) *ARn(T0/AR0) *(ARn + T1) *(ARn - T1)
Cmem	1 ワード（16 ビット）のデータ	係数間接アドレッシング・モード : *CDP *CDP+ *CDP- *(CDP + T0/AR0)

6.5.3.1 *ARn を使用したデータ・メモリのアクセス

オペランド	説明
*ARn	生成されるアドレス : ARnH:([BSAyy +] ARn) ARn は変更されません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *AR4, T2	AR4H:AR4 = XAR4	CPU はアドレス XAR4 の値をリードし、その値を T2 にロードします。AR4 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR4), pair(T2)	**最初のアドレス：** XAR4 **2 番目のアドレス：** XAR4 が偶数の場合 XAR4 + 1 XAR4 が奇数の場合 XAR4 - 1	CPU はアドレス XAR4 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。AR4 は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.2 *ARn+ を使用したデータ・メモリのアクセス

オペランド	説明
*ARn+	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: 16 ビット演算の場合：ARn = ARn + 1 32 ビット演算の場合：ARn = ARn + 2

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *AR4+, T2	AR4H:AR4 = XAR4	CPU はアドレス XAR4 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は 1 つインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR4+), pair(T2)	**最初のアドレス：** XAR4 **2 番目のアドレス：** XAR4 が偶数の場合 XAR4 + 1 XAR4 が奇数の場合 XAR4 - 1	CPU はアドレス XAR4 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR4 は 2 つインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.3 *ARn- を使用したデータ・メモリのアクセス

オペランド	説明
*ARn-	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: 16 ビット演算の場合：ARn = ARn - 1 32 ビット演算の場合：ARn = ARn - 2

構文例	命令例	生成されるアドレス (リニア・ アドレッシング)	説明
MOV Smem, dst	MOV *AR4-, T2	AR4H:AR4 = XAR4	CPU はアドレス XAR4 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は 1 つデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR4-), pair(T2)	**最初のアドレス：** XAR4 **2 番目のアドレス：** XAR4 が偶数の場合 XAR4 + 1 XAR4 が奇数の場合 XAR4 - 1	CPU はアドレス XAR4 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR4 は 2 つデクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.4 *+ARn を使用したデータ・メモリのアクセス

オペランド	説明
*+ARn	1) 変更される ARn: 16 ビット演算の場合：ARn = ARn + 1 32 ビット演算の場合：ARn = ARn + 2 2) 生成されるアドレス : 16 ビット演算の場合：ARnH:([BSAyy +] ARn + 1) 32 ビット演算の場合：ARnH:([BSAyy +] ARn + 2)

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *+AR4, T2	AR4H:(AR4 + 1) = XAR4 + 1	アドレスに使用される前に、AR4 が 1 つインクリメントされます。CPU はアドレス XAR4 + 1 の値をリードし、その値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*+AR4), pair(T2)	**最初のアドレス：** AR4H:(AR4 + 2) = XAR4 + 2 **2 番目のアドレス：** XAR4 + 2 が偶数の場合 (XAR4 + 2) + 1 XAR4 + 2 が奇数の場合 (XAR4 + 2) - 1	アドレスに使用される前に、AR4 が 2 つインクリメントされます。CPU はアドレス XAR4 + 2 の値と次のアドレスまたは前のアドレスの値をリードし、それらの値を T2 と T3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.5 *-ARn を使用したデータ・メモリのアクセス

オペランド	説明
*-ARn	1) 変更される ARn: 16 ビット演算の場合：ARn = ARn - 1 32 ビット演算の場合：ARn = ARn - 2 2) 生成されるアドレス : 16 ビット演算の場合：ARnH:([BSAyy +] ARn - 1) 32 ビット演算の場合：ARnH:([BSAyy +] ARn - 2)

構文例	命令例	生成されるアドレス (リニア・ アドレッシング)	説明
MOV Smem, dst	MOV *-AR4, T2	AR4H:(AR4 - 1) = XAR4 - 1	アドレスに使用される前に、AR4 が 1 つデクリメントされます。CPU はアドレス XAR4 - 1 の値をリードし、その値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*-AR4), pair(T2)	**最初のアドレス：** AR4H:(AR4 - 2) = XAR4 - 2 **2 番目のアドレス：** XAR4 - 2 が偶数の場合 (XAR4 - 2) + 1 XAR4 - 2 が奇数の場合 (XAR4 - 2) - 1	アドレスに使用される前に、AR4 が 2 つデクリメントされます。CPU はアドレス XAR4 - 2 の値と次のアドレスまたは前のアドレスの値をリードし、それらの値を T2 と T3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.6 *(ARn + T0/AR0) を使用したデータ・メモリのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + T0	*(ARn + AR0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR4 + T0), T2	AR4H:AR4 = XAR4	CPU はアドレス XAR4 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は T0 の数だけインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR4 + T0)), pair(T2)	**最初のアドレス：** XAR4 **2 番目のアドレス：** XAR4 が偶数の場合 XAR4 + 1 XAR4 が奇数の場合 XAR4 - 1	CPU はアドレス XAR4 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR4 は T0 の数だけインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.7 *(ARn - T0/AR0) を使用したデータ・メモリのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - T0	*(ARn - AR0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR4 - T0), T2	AR4H:AR4 = XAR4	CPU はアドレス XAR4 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は T0 の数だけデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR4 - T0)), pair(T2)	**最初のアドレス：** XAR4 **2 番目のアドレス：** XAR4 が偶数の場合 XAR4 + 1 XAR4 が奇数の場合 XAR4 - 1	CPU はアドレス XAR4 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR4 は T0 の数だけデクリメントされます。データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.8 *ARn(T0/AR0) を使用したデータ・メモリのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*ARn(T0)	生成されるアドレス : ARnH:([BSAyy +] ARn + T0) ARn は変更されません。ARn はベース・ポインタとして使用されます。T0 がそのベース・ポインタからのオフセットとして使用されます。	*ARn(AR0)	生成されるアドレス : ARnH:([BSAyy +] ARn + AR0) ARn は変更されません。ARn はベース・ポインタとして使用されます。AR0 がそのベース・ポインタからのオフセットとして使用されます。

構文例	**命令例**	**生成されるアドレス (リニア・アドレッシング)**	**説明**
MOV Smem, dst	MOV *AR4(T0), T2	AR4H:(AR4 + T0) = XAR4 + T0	CPU はアドレス XAR4 + T0 の値をリードし、その値を T2 にロードします。AR4 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR4(T0)), pair(T2)	**最初のアドレス：** XAR4 + T0 **2 番目のアドレス：** XAR4 + T0 が偶数の場合 (XAR4 + T0) + 1 XAR4 + T0 が奇数の場合 (XAR4 + T0) - 1	CPU はアドレス XAR4 + T0 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。AR4 は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.9 *(ARn + T0B/AR0B) を使用したデータ・メモリのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn + AR0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *(AR4 + T0B), T2	AR4H:AR4 = XAR4	CPU はアドレス XAR4 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR4 + T0B)), pair(T2)	**最初のアドレス：** XAR4 **2 番目のアドレス：** XAR4 が偶数の場合 XAR4 + 1 XAR4 が奇数の場合 XAR4 - 1	CPU はアドレス XAR4 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR4 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.10 *(ARn - T0B/AR0B) を使用したデータ・メモリのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn - AR0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR4 - T0B), T2	AR4H:AR4 = XAR4	CPU はアドレス XAR4 の値を T2 にロードします。アドレスに使用された後、AR4 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR4 - T0B)), pair(T2)	**最初のアドレス：** XAR4 **2 番目のアドレス：** XAR4 が偶数の場合 XAR4 + 1 XAR4 が奇数の場合 XAR4 - 1	CPU はアドレス XAR4 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR4 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.11 *(ARn + T1) を使用したデータ・メモリのアクセス

オペランド	説明
*(ARn + T1)	1) 生成されるアドレス : ARnH: [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T1

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR7 + T1), AR3	AR7H:AR7 = XAR7	CPU はアドレス XAR7 の値をリードし、その値を AR3 にロードします。アドレスに使用された後、AR7 は T1 の数だけインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR5 + T1)), pair(AR2)	**最初のアドレス：** AR5H:AR5 = XAR5 **2 番目のアドレス：** XAR5 が偶数の場合 XAR5 + 1 XAR5 が奇数の場合 XAR5 - 1	CPU はアドレス XAR5 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。アドレスに使用された後、AR5 は T1 の数だけインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.12 *(ARn - T1) を使用したデータ・メモリのアクセス

オペランド	説明
*(ARn - T1)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - T1

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR7 - T1), AR3	AR7H:AR7 = XAR7	CPU はアドレス XAR7 の値をリードし、その値を AR3 にロードします。アドレスに使用された後、AR7 は T1 の数だけデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR5 - T1)), pair(AR2)	**最初のアドレス：** AR5H:AR5 = XAR5 **2 番目のアドレス：** XAR5 が偶数の場合 XAR5 + 1 XAR5 が奇数の場合 XAR5 - 1	CPU はアドレス XAR5 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。アドレスに使用された後、AR5 は T1 の数だけデクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.13 *ARn(T1) を使用したデータ・メモリのアクセス

オペランド	説明
*ARn(T1)	生成されるアドレス : ARnH:([BSAyy +] ARn + T1) ARn は変更されません。ARn はベース・ポインタとして使用されます。T1 がそのベース・ポインタからのオフセットとして使用されます。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *AR7(T1), AR3	AR7H:(AR7 + T1) = XAR7 + T1	CPU はアドレス XAR7 + T1 の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR5(T1)), pair(AR2)	**最初のアドレス：** AR5H:(AR5 + T1) = XAR5 + T1 **2 番目のアドレス：** XAR5 + T1 が偶数の場合 (XAR5 + T1) + 1 XAR5 + T1 が奇数の場合 (XAR5 + T1) - 1	CPU はアドレス XAR5 + T1 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。AR5 は変更されません。データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.14 *ARn(#K16) を使用したデータ・メモリのアクセス

オペランド	説明
*ARn(#K16)	生成されるアドレス : ARnH:([BSAyy +] ARn + K16) ARn は変更されません。ARn はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *AR7(#8), AR3	AR7H:(AR7 + 8) = XAR7 + 8	CPU はアドレス XAR7 + 8 の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR5(#20)), pair(AR2)	**最初のアドレス：** AR5H:(AR5 + 20) = XAR5 + 20 **2 番目のアドレス：** XAR5 + 20 が偶数の場合 (XAR5 + 20) + 1 XAR5 + 20 が奇数の場合 (XAR5 + 20) - 1	CPU はアドレス XAR5 + 20 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。AR5 は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.15 *+ARn(#K16) を使用したデータ・メモリのアクセス

オペランド	説明
*+ARn(#K16)	1) 変更される ARn: ARn = ARn + K16 2) 生成されるアドレス : ARnH:([BSAyy +] ARn + K16)

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *+AR7(#8), AR3	AR7H:(AR7 + 8) = XAR7 + 8	AR7 がアドレスに使用される前に、定数が AR7 に加算されます。CPU はアドレス XAR7 + 8 の値をリードし、その値を AR3 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*+AR5(#20)), pair(AR2)	**最初のアドレス：** AR5H:(AR5 + 20) = XAR5 + 20 **2 番目のアドレス：** XAR5 + 20 が偶数の場合 (XAR5 + 20) + 1 XAR5 + 20 が奇数の場合 (XAR5 + 20) - 1	AR5 がアドレスに使用される前に、定数が AR5 に加算されます。CPU はアドレス XAR5 + 20 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.16 *ARn(short(#k3)) を使用したデータ・メモリのアクセス

オペランド	説明
*ARn(short(#k3))	生成されるアドレス : ARnH:([BSAyy +] ARn + k3) ARn は変更されません。ARn はベース・ポインタとして使用されます。3 ビット符号なし定数（k3）がそのベース・ポインタからのオフセットとして使用されます。k3 の範囲は、1 ～ 7 です。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *AR7(short(#1)), AR3	AR7H:(AR7 + 1) = XAR7 + 1	CPU はアドレス XAR7 + 1 の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR5(short(#7))), pair(AR2)	**最初のアドレス：** AR5H:(AR5 + 7) = XAR5 + 7 **2 番目のアドレス：** XAR5 + 7 が偶数の場合 (XAR5 + 7) + 1 XAR5 + 7 が奇数の場合 (XAR5 + 7) - 1	CPU はアドレス XAR5 + 7 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。AR5 は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.17 *CDP を使用したデータ・メモリのアクセス

オペランド	説明
*CDP	生成されるアドレス : CDPH:[BSAC +] CDP CDP は変更されません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *CDP, T2	CDPH:CDP = XCDP	CPU はアドレス XCDP の値をリードし、その値を T2 にロードします。CDP は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP), pair(T2)	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XCDP の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。CDP は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。
MPY Xmem, Cmem, ACx :: MPY Ymem, Cmem, ACy	MPY *AR0, *CDP, AC0 :: MPY *AR1, *CDP, AC1	CDPH:CDP = XCDP	CPU はアドレス XAR0 の値にアドレス XCDP の係数を乗算し、その結果を AC0 に格納します。同時に、CPU はアドレス XAR1 の値に同じ係数を乗算し、その結果を AC1 に格納します。CDP は変更されません。
MOV dbl(Lmem), Cmem	MOV dbl(*AR7), *CDP	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XAR7 の値と XAR7 ± 1 の値をリードし、これらの値をアドレス XCDP と XCDP ± 1 にそれぞれライトします。CDP は変更されません。

注： 代数表記命令を使用する場合は、各係数間接（Cmem）オペランドを coef() 構文要素で囲む必要があります。例については、6.4.4.3 項を参照してください。

6.5.3.18 *CDP+ を使用したデータ・メモリのアクセス

オペランド	説明
*CDP+	1) 生成されるアドレス : CDPH:[BSAC +] CDP 2) 変更される CDP: 16 ビット演算の場合：CDP = CDP + 1 32 ビット演算の場合：CDP = CDP + 2

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *CDP+, T2	CDPH:CDP = XCDP	CPU はアドレス XCDP の値をリードし、その値を T2 にロードします。アドレスに使用された後、CDP は 1 つインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP+), pair(T2)	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XCDP の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、CDP は 2 つインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。
MPY Xmem, Cmem, ACx :: MPY Ymem, Cmem, ACy	MPY *AR0, *CDP+, AC0 :: MPY *AR1, *CDP+, AC1	CDPH:CDP = XCDP	CPU はアドレス XAR0 の値にアドレス XCDP の係数を乗算し、その結果を AC0 に格納します。同時に、CPU はアドレス XAR1 の値に同じ係数を乗算し、その結果を AC1 に格納します。アドレスに使用された後、CDP は 1 つインクリメントされます。
MOV dbl(Lmem), Cmem	MOV dbl(*AR7), *CDP+	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XAR7 の値と XAR7 ± 1 の値をリードし、これらの値をアドレス XCDP と XCDP ± 1 にそれぞれライトします。アドレスに使用された後、CDP は 2 つインクリメントされます。

注： 代数表記命令を使用する場合は、各係数間接（Cmem）オペランドを coef() 構文要素で囲む必要があります。例については、6.4.4.3 項を参照してください。

6.5.3.19 *CDP- を使用したデータ・メモリのアクセス

オペランド	説明
*CDP-	1) 生成されるアドレス : CDPH:[BSAC +] CDP 2) 変更される CDP: 16 ビット演算の場合：CDP = CDP - 1 32 ビット演算の場合：CDP = CDP - 2

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *CDP-, T2	CDPH:CDP = XCDP	CPU はアドレス XCDP の値をリードし、その値を T2 にロードします。アドレスに使用された後、CDP は 1 つデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP-), pair(T2)	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XCDP の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、CDP は 2 つデクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。
MPY Xmem, Cmem, ACx :: MPY Ymem, Cmem, ACy	MPY *AR0, *CDP-, AC0 :: MPY *AR1, *CDP-, AC1	CDPH:CDP = XCDP	CPU はアドレス XAR0 の値にアドレス XCDP の係数を乗算し、その結果を AC0 に格納します。同時に、CPU はアドレス XAR1 の値に同じ係数を乗算し、その結果を AC1 に格納します。アドレスに使用された後、CDP は 1 つデクリメントされます。
MOV dbl(Lmem), Cmem	MOV dbl(*AR7), *CDP-	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XAR7 の値と XAR7 ± 1 の値をリードし、これらの値をアドレス XCDP と XCDP ± 1 にそれぞれライトします。アドレスに使用された後、CDP は 2 つデクリメントされます。

注： 代数表記命令を使用する場合は、各係数間接（Cmem）オペランドを coef() 構文要素で囲む必要があります。例については、6.4.4.3 項を参照してください。

6.5.3.20 *CDP(#K16) を使用したデータ・メモリのアクセス

オペランド	説明
*CDP(#K16)	生成されるアドレス : CDPH:([BSAC +] CDP + K16) CDP は変更されません。CDP はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *CDP(#8), AR3	CDPH:(CDP + 8) = XCDP + 8	CPU はアドレス XCDP + 8 の値をリードし、その値を AR3 にロードします。CDP は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP(#20)), pair(AR2)	**最初のアドレス：** CDPH:(CDP + 20) = XCDP + 20 **2 番目のアドレス：** XCDP + 20 が偶数の場合 (XCDP + 20) + 1 XCDP + 20 が奇数の場合 (XCDP + 20) - 1	CPU はアドレス XCDP + 20 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。CDP は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.21 *+CDP(#K16) を使用したデータ・メモリのアクセス

オペランド	説明
*+CDP(#K16)	1) 変更される CDP: CDP = CDP + K16 2) 生成されるアドレス : CDPH:([BSAC +] CDP + K16)

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *+CDP(#8), AR3	CDPH:(CDP + 8) = XCDP + 8	CDP がアドレスに使用される前に、定数が CDP に加算されます。CPU はアドレス XCDP + 8 の値をリードし、その値を AR3 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*+CDP(#20)), pair(AR2)	**最初のアドレス：** CDPH:(CDP + 20) = XCDP + 20 **2 番目のアドレス：** XCDP + 20 が偶数の場合 (XCDP + 20) + 1 XCDP + 20 が奇数の場合 (XCDP + 20) - 1	CDP がアドレスに使用される前に、定数が CDP に加算されます。CPU はアドレス XCDP + 20 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.5.3.22 *(CDP + T0/AR0) を使用したデータ・メモリのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(CDP + T0)	1) 生成されるアドレス : CDPH:[BSAC +] CDP 2) CDP = CDP + T0	*(CDP + AR0)	1) 生成されるアドレス : CDPH:[BSAC +] CDP 2) CDP = CDP + AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MPY Xmem, Cmem, ACx :: MPY Ymem, Cmem, ACy	MPY *AR0, *(CDP + T0), AC0 :: MPY *AR1, *(CDP + T0), AC1	CDPH:CDP = XCDP	CPU はアドレス XAR0 の値にアドレス XCDP の係数を乗算し、その結果を AC0 に格納します。同時に、CPU はアドレス XAR1 の値に同じ係数を乗算し、その結果を AC1 に格納します。アドレスに使用された後、CDP は T0 の数だけインクリメントされます。
MOV dbl(Lmem), Cmem	MOV dbl(*AR7), *(CDP + T0)	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XAR7 の値と XAR7 ± 1 の値をリードし、これらの値をアドレス XCDP と XCDP ± 1 にそれぞれライトします。アドレスに使用された後、CDP は T0 の数だけインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

注： 代数表記命令を使用する場合は、各係数間接（Cmem）オペランドを coef() 構文要素で囲む必要があります。例については、6.4.4.3 項を参照してください。

6.6 メモリ・マップド・レジスタのアドレッシング

絶対、直接、間接のアドレッシングを使用して、メモリ・マップド・レジスタ（MMR）をアドレス指定できます。

アドレッシング・タイプ	参照先
絶対	このページ
直接	P. 6-63
間接	P. 6-65

メモリ・マップド・レジスタへのアクセスにはいくつか制約があります。詳細は、6.7 節を参照してください。

6.6.1 k16 および k23 絶対アドレッシング・モードを使用した MMR のアドレッシング

k16 絶対オペランド *abs16(#k16) および k23 絶対オペランド *(#k23) を使用して、次のいずれかの構文要素で任意の命令のメモリ・マップド・レジスタにアクセスできます。

Smem　1 ワード（16 ビット）のデータを表します。

Lmem　2 つの隣接したワード（32 ビット）のデータを表します。

表 6-16 と表 6-17 の例を参照してください。メモリ・マップド・レジスタはメイン・データ・ページ 0 にあるため、*abs16(#k16) でこれらのレジスタにアクセスするにはまず DPH = 0 であることを確認しておく必要があります。

注：

マルチバイト拡張のため、*abs16(#k16) または *(#k23) を使用する命令は別の命令と並列に実行することはできません

- **命令が *abs16(#k16) を使用する場合、定数 k16 はその命令に 2 バイト拡張でエンコードされます。**
- **命令が *(#k23) を使用する場合、定数 k23 はその命令に 3 バイト拡張でエンコードされます。**

表 6-16. *abs16(#k16) を使用したメモリ・マップド・レジスタのアクセス

構文例	命令例	生成されるアドレス（DPH は必ず 0）	説明
MOV Smem, dst	MOV *abs16(#AR2), T2	DPH:k16 = 00 0012h	AR2 は、アドレス 00 0012h です。CPU は AR2 の内容を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*abs16(#AR2)), pair(T2)	**最初のアドレス：** DPH:k16 = 00 0012h **2 番目のアドレス：** (DPH:k16) + 1 = 00 0013h	AR2 と AR3 は、それぞれアドレス 00 0012h と 00 0013h です。CPU は AR2 と AR3 の内容をリードし、これらの内容を T2 と T3 にそれぞれコピーします。

表 6-17. *(#k23) を使用したメモリ・マップド・レジスタのアクセス

構文例	命令例	生成されるアドレス	説明
MOV Smem, dst	MOV *(#AC0L), T2	k23 = 00 0008h	AC0L は AC0 の 16 の LSB のアドレスを示します。CPU は AC0(15 ～ 0) の内容を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(#AC0L)), pair(T2)	**最初のアドレス：** k23 = 00 0008h **2 番目のアドレス：** k23 + 1 = 00 0009h	AC0L は AC0 の 16 の LSB のアドレスを示します。CPU は AC0(15 ～ 0) の内容を T2 にロードし、AC0(31 ～ 16) の内容を T3 にロードします。

6.6.2 DP 直接アドレッシング・モードを使用した MMR のアドレッシング

CPL ビットが 0（ゼロ）の場合、次のいずれかの構文要素が命令にあると、メモリ・マップド・レジスタ（MMR）にアクセスするために DP 直接オペランドを使用できます。

Smem　1 ワード（16 ビット）のデータを表します。

Lmem　2 つの隣接したワード（32 ビット）のデータを表します。

データ・メモリ・ロケーションへのアクセスではなく、メモリ・マップド・レジスタへのアクセスであることを示すために、mmap() 修飾子を使用してください。代数表記命令を使用する場合、mmap() はメモリ・マップド・レジスタのアクセスを実行

する命令と並列に指定される命令修飾子です。ニーモニック命令を使用する場合、mmap() は修飾されるオペランドを囲みます。

mmap() 修飾子は、アドレス生成ユニット（DAGEN）を次のように動作させます。

DPH = 0 アクセスはメイン・データ・ページ 0 に対して行われます。

CPL = 0 アクセスは DP に関連して行われます。

DP = 0　アクセスは 0000h のローカル・アドレスに関連して行われます。

レジスタにアクセスするために @Daddr と mmap() を使用する例を表 6-18 に示します。アドレスに対して、アセンブラは Doffset を次のように計算します。

Doffset = Daddr & 7Fh

ここで、& はビットごとの AND 演算を表します。Daddr が AC0L（AC0 の下位 16 ビット）によって与えられていることに注意してください。AC0L はデータ空間のアドレス 00 0008h にマップされています。

表 6-18.　@Daddr を使用したメモリ・マップド・レジスタのアクセス

構文例	命令例	生成されるアドレス (DPH = DP = 0)	説明
MOV Smem, dst	MOV mmap(@AC0L), AR2	DPH:(DP + Doffset) = 00:(0000h + 0008h) = 00 0008h	AC0L は AC0 の 16 の LSB のアドレスを示します。CPU は AC0(15 ～ 0) の内容を AR2 にコピーします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(mmap(@AC0L)), pair(AR2)	**最初のアドレス：** DPH:(DP + Doffset) = 00 0008h **2 番目のアドレス：** DPH:(DP + Doffset + 1) = 00 0009h	AC0L は AC0 の 16 の LSB のアドレスを示します。CPU は AC0(15 ～ 0) の内容を AR2 にコピーし、AC0(31 ～ 16) の内容を AR3 にコピーします。

6.6.3 間接アドレッシング・モードを使用した MMR のアドレッシング

次のいずれかの構文要素が命令にある場合、メモリ・マップド・レジスタにアクセスするために、間接オペランドを使用できます。

Smem　1 ワード（16 ビット）のデータを表します。

Xmem　DB バスを通過する 1 ワード（16 ビット）のリード・データを表します。

Ymem　EB バスを通過する 1 ワード（16 ビット）のライト・データを表します。

Lmem　2 つの隣接したワード（32 ビット）のデータを表します。

まず、ポインタが正しいデータ空間アドレスを含むことを確認しておく必要があります。たとえば、XAR6 がポインタで、AC0 の下位ワードにアクセスする場合、次の命令を使用して XAR6 を初期化します。

AMOV #AC0L, XAR6

ここで、AC0L は AC0 の下位ワードのアドレスを示すキーワードです。同様に、ポインタが CDP の場合は、次の命令を使用できます。

AMOV #AC0L, XCDP

メモリ・マップド・レジスタのアクセスをサポートする間接オペランドを図 6-14 に示します。また、この表以下に続く項では、これらのオペランドの詳細と例を示します。

図 6-14.　メモリ・マップド・レジスタのアクセスの間接オペランド

AR 間接アドレッシング・モード :

DSP モード (ARMS = 0):	制御モード (ARMS = 1):
*ARn	*ARn
*ARn+	*ARn+
*ARn-	*ARn-
*+ARn	
*-ARn	
*(ARn + T0/AR0)	*(ARn + T0/AR0)
*(ARn - T0/AR0)	*(ARn - T0/AR0)
*ARn(T0/AR0)	*ARn(T0/AR0)
*(ARn + T0B/AR0B)	
*(ARn - T0B/AR0B)	
*(ARn + T1)	
*(ARn - T1)	
*ARn(T1)	
*ARn(#K16)	*ARn(#K16)
*+ARn(#K16)	*+ARn(#K16)
	*ARn(short(#k3))

CDP 間接アドレッシング・モード :

*CDP
*CDP+
*CDP-
*CDP(#k16)
*+CDP(#k16)

6.6.3.1 *ARn を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*ARn	生成されるアドレス : ARnH:([BSAyy +] ARn) ARn は変更されません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *AR6, T2 （AR6 はレジスタを参照）	AR6H:AR6 = XAR6	CPU はアドレス XAR6 の値をリードし、その値を T2 にロードします。AR6 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR6), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 **2 番目のアドレス：** XAR6 が偶数の場合 XAR6 + 1 XAR6 が奇数の場合 XAR6 - 1	CPU はアドレス XAR6 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。AR6 は変更されません。データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.2 *ARn+ を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*ARn+	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: 16 ビット演算の場合：ARn = ARn + 1 32 ビット演算の場合：ARn = ARn + 2

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *AR6+, T2 （AR6 はレジスタを参照）	AR6H:AR6 = XAR6	CPU はアドレス XAR6 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR6 は 1 つインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR6+), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 **2 番目のアドレス：** XAR6 が偶数の場合 XAR6 + 1 XAR6 が奇数の場合 XAR6 - 1	CPU はアドレス XAR6 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR6 は 2 つインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.3 *ARn- を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*ARn-	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: 16 ビット演算の場合：ARn = ARn - 1 32 ビット演算の場合：ARn = ARn - 2

構文例	命令例	生成されるアドレス (リニア・ アドレッシング)	説明
MOV Smem, dst	MOV *AR6-, T2 （AR6 はレジスタを参照）	AR6H:AR6 = XAR6	CPU はアドレス XAR6 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR6 は 1 つデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR6-), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 **2 番目のアドレス：** XAR6 が偶数の場合 XAR6 + 1 XAR6 が奇数の場合 XAR6 - 1	CPU はアドレス XAR6 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR6 は 2 つデクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.4 *+ARn を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*+ARn	1) 変更される ARn: 16 ビット演算の場合：ARn = ARn + 1 32 ビット演算の場合：ARn = ARn + 2 2) 生成されるアドレス : 16 ビット演算の場合：ARnH:([BSAyy +] ARn + 1) 32 ビット演算の場合：ARnH:([BSAyy +] ARn + 2)

構文例	命令例	生成されるアドレス (リニア・ アドレッシング)	説明
MOV Smem, dst	MOV *+AR6, T2 （AR6 はレジスタを参照）	AR6H:(AR6 + 1) = XAR6 + 1	アドレスに使用される前に、AR6 が 1 つインクリメントされます。CPU はアドレス XAR6 + 1 の値をリードし、その値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*+AR6), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** AR6H:(AR6 + 2) = XAR6 + 2 **2 番目のアドレス：** XAR6 + 2 が偶数の場合 (XAR6 + 2) + 1 XAR6 + 2 が奇数の場合 (XAR6 + 2) - 1	アドレスに使用される前に、AR6 が 2 つインクリメントされます。CPU はアドレス XAR6 + 2 の値と次のアドレスまたは前のアドレスの値をリードし、それらの値を T2 と T3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.5 *-ARn を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*-ARn	1) 変更される ARn: 16 ビット演算の場合：ARn = ARn - 1 32 ビット演算の場合：ARn = ARn - 2 2) 生成されるアドレス : 16 ビット演算の場合：ARnH:([BSAyy +] ARn - 1) 32 ビット演算の場合：ARnH:([BSAyy +] ARn - 2)

構文例	命令例	生成されるアドレス (リニア・ アドレッシング)	説明
MOV Smem, dst	MOV *-AR6, T2 （AR6 はレジスタを参照）	AR6H:(AR6 - 1) = XAR6 - 1	アドレスに使用される前に、AR6 が 1 つデクリメントされます。CPU はアドレス XAR6 - 1 の値をリードし、その値を T2 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*-AR6), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** AR6H:(AR6 - 2) = XAR6 - 2 **2 番目のアドレス：** XAR6 - 2 が偶数の場合 (XAR6 - 2) + 1 XAR6 - 2 が奇数の場合 (XAR6 - 2) – 1	アドレスに使用される前に、AR6 が 2 つデクリメントされます。CPU はアドレス XAR6 - 2 の値と次のアドレスまたは前のアドレスの値をリードし、それらの値を T2 と T3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.6 *(ARn + T0/AR0) を使用したメモリ・マップド・レジスタのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + T0	*(ARn + AR0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR6 + T0), T2 （AR6 はレジスタを参照）	AR6H:AR6 = XAR6	CPU はアドレス XAR6 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR6 は T0 の数だけインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR6 + T0)), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 **2 番目のアドレス：** XAR6 が偶数の場合 XAR6 + 1 XAR6 が奇数の場合 XAR6 - 1	CPU はアドレス XAR6 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR6 は T0 の数だけインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.7 *(ARn - T0/AR0) を使用したメモリ・マップド・レジスタのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - T0	*(ARn - AR0)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR6 - T0), T2 （AR6 はレジスタを参照）	AR6H:AR6 = XAR6	CPU はアドレス XAR6 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR6 は T0 の数だけデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR6 - T0)), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 **2 番目のアドレス：** XAR6 が偶数の場合 XAR6 + 1 XAR6 が奇数の場合 XAR6 - 1	CPU はアドレス XAR6 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR6 は T0 の数だけデクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.8 *ARn(T0/AR0) を使用したメモリ・マップド・レジスタのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*ARn(T0)	生成されるアドレス : ARnH:([BSAyy +] ARn + T0) ARn は変更されません。ARn はベース・ポインタとして使用されます。T0 がそのベース・ポインタからのオフセットとして使用されます。	*ARn(AR0)	生成されるアドレス : ARnH:([BSAyy +] ARn + AR0) ARn は変更されません。ARn はベース・ポインタとして使用されます。AR0 がそのベース・ポインタからのオフセットとして使用されます。

構文例	**命令例**	**生成されるアドレス (リニア・アドレッシング)**	**説明**
MOV Smem, dst	MOV *AR6(T0), T2 （AR6 はレジスタを参照）	AR6H:(AR6 + T0) = XAR6 + T0	CPU はアドレス XAR6 + T0 の値をリードし、その値を T2 にロードします。AR6 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR6(T0)), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 + T0 **2 番目のアドレス：** XAR6 + T0 が偶数の場合 (XAR6 + T0) + 1 XAR6 + T0 が奇数の場合 (XAR6 + T0) - 1	CPU はアドレス XAR6 + T0 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。AR6 は変更されません。データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.9 *(ARn + T0B/AR0B) を使用したメモリ・マップド・レジスタのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn + AR0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn + AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR6 + T0B), T2 （AR6 はレジスタを参照）	AR6H:AR6 = XAR6	CPU はアドレス XAR6 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR6 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR6 + T0B)), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 **2 番目のアドレス：** XAR6 が偶数の場合 XAR6 + 1 XAR6 が奇数の場合 XAR6 - 1	CPU はアドレス XAR6 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR6 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.10 *(ARn - T0B/AR0B) を使用したメモリ・マップド・レジスタのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn - AR0B)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR6 - T0B), T2 （AR6 はレジスタを参照）	AR6H:AR6 = XAR6	CPU はアドレス XAR6 の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR6 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR6 - T0B)), pair(T2) （AR6 はレジスタを参照）	**最初のアドレス：** XAR6 **2 番目のアドレス：** XAR6 が偶数の場合 XAR6 + 1 XAR6 が奇数の場合 XAR6 - 1	CPU はアドレス XAR6 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、AR6 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.11 *(ARn + T1) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*(ARn + T1)	1) 生成されるアドレス : ARnH: [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T1

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *(AR7 + T1), AR3 （AR7 はレジスタを参照）	AR7H:AR7 = XAR7	CPU はアドレス XAR7 の値をリードし、その値を AR3 にロードします。アドレスに使用された後、AR7 は T1 の数だけインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR5 + T1)), pair(AR2) （AR5 はレジスタを参照）	**最初のアドレス：** AR5H:AR5 = XAR5 **2 番目のアドレス：** XAR5 が偶数の場合 XAR5 + 1 XAR5 が奇数の場合 XAR5 - 1	CPU はアドレス XAR5 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。アドレスに使用された後、AR5 は T1 の数だけインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.12 *(ARn - T1) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*(ARn - T1)	1) 生成されるアドレス : ARnH:([BSAyy +] ARn) 2) 変更される ARn: ARn = ARn - T1

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *(AR7 - T1), AR3 （AR7 はレジスタを参照）	AR7H:AR7 = XAR7	CPU はアドレス XAR7 の値をリードし、その値を AR3 にロードします。アドレスに使用された後、AR7 は T1 の数だけデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*(AR5 - T1)), pair(AR2) （AR5 はレジスタを参照）	**最初のアドレス：** AR5H:AR5 = XAR5 **2 番目のアドレス：** XAR5 が偶数の場合 XAR5 + 1 XAR5 が奇数の場合 XAR5 - 1	CPU はアドレス XAR5 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。アドレスに使用された後、AR5 は T1 の数だけデクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.13 *ARn(T1) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*ARn(T1)	生成されるアドレス : ARnH:([BSAyy +] ARn + T1) ARn は変更されません。ARn はベース・ポインタとして使用されます。T1 がそのベース・ポインタからのオフセットとして使用されます。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *AR7(T1), AR3 （AR7 はレジスタを参照）	AR7H:(AR7 + T1) = XAR7 + T1	CPU はアドレス XAR7 + T1 の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR5(T1)), pair(AR2) （AR5 はレジスタを参照）	**最初のアドレス：** AR5H:(AR5 + T1) = XAR5 + T1 **2 番目のアドレス：** XAR5 + T1 が偶数の場合 (XAR5 + T1) + 1 XAR5 + T1 が奇数の場合 (XAR5 + T1) - 1	CPU はアドレス XAR5 + T1 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。AR5 は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.14 *ARn(#K16) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*ARn(#K16)	生成されるアドレス : ARnH:([BSAyy +] ARn + K16) ARn は変更されません。ARn はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *AR7(#9), AR3 （AR7 はレジスタを参照）	AR7H:(AR7 + 9) = XAR7 + 9	CPU はアドレス XAR7 + 9 の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR5(#40)), pair(AR2) （AR5 はレジスタを参照）	**最初のアドレス：** AR5H:(AR5 + 40) = XAR5 + 40 **2 番目のアドレス：** XAR5 + 40 が偶数の場合 (XAR5+ 40) + 1 XAR5 + 40 が奇数の場合 (XAR5+ 40) - 1	CPU はアドレス XAR5 + 40 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。AR5 は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.15 *+ARn(#K16) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*+ARn(#K16)	1) 変更される ARn: ARn = ARn + K16 2) 生成されるアドレス : ARnH:([BSAyy +] ARn + K16)

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *+AR7(#9), AR3 （AR7 はレジスタを参照）	AR7H:(AR7 + 9) = XAR7 + 9	AR7 がアドレスに使用される前に、定数が AR7 に加算されます。CPU はアドレス XAR7 + 9 の値をリードし、その値を AR3 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*+AR5(#40)), pair(AR2) （AR5 はレジスタを参照）	**最初のアドレス：** AR5H:(AR5 + 40) = XAR5 + 40 **2 番目のアドレス：** XAR5 + 40 が偶数の場合 (XAR5 + 40) + 1 XAR5 + 40 が奇数の場合 (XAR5 + 40) - 1	AR5 がアドレスに使用される前に、定数が AR5 に加算されます。CPU はアドレス XAR5 + 40 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.16 *ARn(short(#k3)) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*ARn(short(#k3))	生成されるアドレス : ARnH:([BSAyy +] ARn + k3) ARn は変更されません。ARn はベース・ポインタとして使用されます。3 ビット符号なし定数（k3）がそのベース・ポインタからのオフセットとして使用されます。k3 の範囲は、1 ～ 7 です。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *AR7(short(#2)), AR3 （AR7 はレジスタを参照）	AR7H:(AR7 + 2) = XAR7 + 2	CPU はアドレス XAR7 + 2 の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*AR5(short(#7))), pair(AR2) （AR5 はレジスタを参照）	**最初のアドレス：** AR5H:(AR5 + 7) = XAR5 + 7 **2 番目のアドレス：** XAR5 + 7 が偶数の場合 (XAR5+ 7) + 1 XAR5 + 7 が奇数の場合 (XAR5+ 7) - 1	CPU はアドレス XAR5 + 7 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。AR5 は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.17 *CDP を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*CDP	生成されるアドレス : CDPH:[BSAC +] CDP CDP は変更されません。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *CDP, T2 （CDP はレジスタを参照）	CDPH:CDP = XCDP	CPU はアドレス XCDP の値をリードし、その値を T2 にロードします。CDP は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP), pair(T2) （CDP はレジスタを参照）	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XCDP の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。CDP は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.18 *CDP+ を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*CDP+	1) 生成されるアドレス : CDPH:[BSAC +] CDP 2) 変更される CDP: 16 ビット演算の場合：CDP = CDP + 1 32 ビット演算の場合：CDP = CDP + 2

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *CDP+, T2 （CDP はレジスタを参照）	CDPH:CDP = XCDP	CPU はアドレス XCDP の値をリードし、その値を T2 にロードします。アドレスに使用された後、CDP は 1 つインクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP+), pair(T2) （CDP はレジスタを参照）	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XCDP の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、CDP は 2 つインクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.19 *CDP- を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*CDP-	1) 生成されるアドレス : CDPH:[BSAC +] CDP 2) 変更される CDP: 16 ビット演算の場合：CDP = CDP - 1 32 ビット演算の場合：CDP = CDP - 2

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV *CDP-, T2 （CDP はレジスタを参照）	CDPH:CDP = XCDP	CPU はアドレス XCDP の値をリードし、その値を T2 にロードします。アドレスに使用された後、CDP は 1 つデクリメントされます。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP-), pair(T2) （CDP はレジスタを参照）	**最初のアドレス：** XCDP **2 番目のアドレス：** XCDP が偶数の場合 XCDP + 1 XCDP が奇数の場合 XCDP - 1	CPU はアドレス XCDP の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を T2 と T3 にそれぞれロードします。アドレスに使用された後、CDP は 2 つデクリメントされます。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.20 *CDP(#K16) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*CDP(#K16)	生成されるアドレス : CDPH:([BSAC +] CDP + K16) CDP は変更されません。CDP はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *CDP(#9), AR3 （CDP はレジスタを参照）	CDPH:(CDP + 9) = XCDP + 9	CPU はアドレス XCDP + 9 の値をリードし、その値を AR3 にロードします。CDP は変更されません。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*CDP(#40)), pair(AR2) （CDP はレジスタを参照）	**最初のアドレス：** CDPH:(CDP + 40) = XCDP + 40 **2 番目のアドレス：** XCDP + 40 が偶数の場合 (XCDP+ 40) + 1 XCDP + 40 が奇数の場合 (XCDP+ 40) - 1	CPU はアドレス XCDP + 40 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。CDP は変更されません。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.6.3.21 *+CDP(#K16) を使用したメモリ・マップド・レジスタのアクセス

オペランド	説明
*+CDP(#K16)	1) 変更される CDP: CDP = CDP + K16 2) 生成されるアドレス : CDPH:([BSAC +] CDP + K16)

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV *+CDP(#9), AR3 （CDP はレジスタを参照）	CDPH:(CDP + 9) = XCDP + 9	CDP がアドレスに使用される前に、定数が CDP に加算されます。CPU はアドレス XCDP + 9 の値をリードし、その値を AR3 にロードします。
MOV dbl(Lmem), pair(TAx)	MOV dbl(*+CDP(#40)), pair(AR2) （CDP はレジスタを参照）	**最初のアドレス：** CDPH:(CDP + 40) = XCDP + 40 **2 番目のアドレス：** XCDP + 40 が偶数の場合 (XCDP + 40) + 1 XCDP + 40 が奇数の場合 (XCDP + 40) - 1	CDP がアドレスに使用される前に、定数が CDP に加算されます。CPU はアドレス XCDP + 40 の値と次のアドレスまたは前のアドレスの値をリードし、これらの値を AR2 と AR3 にそれぞれロードします。 データ・メモリにおけるロング・ワードのアラインメントの詳細は、3.3.2 節「データ・タイプ」を参照してください。

6.7 メモリ・マップド・レジスタへのアクセスに関する制約

表 6-19 に記述されている命令構文では、Smem はメモリ・マップド・レジスタ（MMR）を参照できません。メモリ・マップド・レジスタ内のバイトにアクセスできる命令はありません。次の構文のいずれかで Smem が MMR の場合、DSP はハードウェア・バス・エラー割り込み（BERRINT）の要求を CPU に送ります。

表 6-19. Smem オペランドが MMR を参照できない命令構文

Smem が MMR を参照できない構文	命令タイプ

MOV [uns(]high_byte(Smem)[)], dst
MOV [uns(]low_byte(Smem)[)], dst
MOV high_byte(Smem) << #SHIFTW, ACx
MOV low_byte(Smem) << #SHIFTW, ACx
	アキュムレータ、補助レジスタ、一時レジスタのロード

MOV src, high_byte(Smem) MOV src, low_byte(Smem)	アキュムレータ、補助レジスタ、一時レジスタのストア

表 6-20 に記述されている状況では、メモリ・マップド・レジスタ（MMR）を参照できません。それ以外の場合、DSP はハードウェア・バス・エラー割り込み（BERRINT）の要求を CPU に送ります。

表 6-20. MMR を参照できない状況

MMR を参照できない状況	注
デュアル・ライトの Xmem	FAB バスは MMR にアクセスできません。
デュアル・リードの Ymem	CAB バスは MMR にアクセスできません。
係数リードの Cmem	BAB バスは MMR にアクセスできません。
スタックのアクセス	呼び出し / 復帰 / 割り込み / トラップ / プッシュ / ポップによるアクセス

バス・エラーが発生した場合、エラーの原因である命令の機能、および並列に実行される命令の機能は保証されません。

6.8 レジスタ・ビットのアドレッシング

直接アドレッシング（6.8.1 項を参照）および間接アドレッシング（6.8.2 項を参照）は、単一のレジスタ・ビットまたはペアのレジスタ・ビットをアドレス指定するために使用できます。絶対アドレッシング・モードは、レジスタ・ビットへのアクセスをサポートしていません。I/O 空間へのアクセスは、これらのアドレッシング・レジスタ・ビットと結合することはできません。

6.8.1 レジスタ・ビット直接アドレッシング・モードを使用したレジスタ・ビットのアドレッシング

次の構文要素が命令にある場合、レジスタ・ビットにアクセスするためにレジスタ・ビット直接オペランド @bitoffset を使用することができます。

Baddr　データのビット・アドレスを表します。レジスタ・ビットのテスト / セット / クリア / 補数演算の各命令だけが Baddr をサポートします。これらの命令は、レジスタ（アキュムレータ（AC0 ～ AC3）、補助レジスタ（AR0 ～ AR7）、一時レジスタ（T0 ～ T3））のみのビットにアクセスすることを可能にします。

@bitoffset を使用してレジスタ・ビットにアクセスする例を表 6-21 に示します。

表 6-21. @bitoffset を使用したレジスタ・ビットのアクセス

構文例	命令例	生成されるビット・アドレス	説明
BSET Baddr, src	BSET @0, AC3	0	CPU は AC3 のビット 0 をセットします。
BTSTP Baddr, src	BTSTP @30, AC3	30 および 31	CPU は AC3 のビット 30 と 31 をテストします。AC0(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC0(31) の内容を ST0_55 の TC2 ビットにコピーします。

6.8.2 間接アドレッシング・モードを使用したレジスタ・ビットのアドレッシング

次の構文要素が命令にある場合、レジスタ・ビットにアクセスするために間接オペランドを使用できます。

Baddr　データのビット・アドレスを表します。レジスタ・ビットのテスト / セット / クリア / 補数演算の各命令だけが Baddr をサポートします。これらの命令は、レジスタ（アキュムレータ（AC0 ～ AC3）、補助レジスタ（AR0 ～ AR7）、一時レジスタ（T0 ～ T3））のみのビットにアクセスすることを可能にします。

まず、ポインタが正しいビット数を含むことを確認しておく必要があります。たとえば、AR6 がポインタで、レジスタのビット 15 にアクセスする場合は、AR6 を初期化するために、次の命令を使用できます。

MOV #15, AR6

レジスタ・ビットのアクセスをサポートする間接オペランドを図 6-15 に示します。また、この表以下に続く項では、これらのオペランドの詳細と例を示します。

図 6-15. レジスタ・ビット・アクセスの間接オペランド

AR 間接アドレッシング・モード :

DSP モード (ARMS = 0):	制御モード (ARMS = 1):
*ARn	*ARn
*ARn+	*ARn+
*ARn-	*ARn-
*+ARn	
*-ARn	
*(ARn + T0/AR0)	*(ARn + T0/AR0)
*(ARn - T0/AR0)	*(ARn - T0/AR0)
*ARn(T0/AR0)	*ARn(T0/AR0)
*(ARn + T0B/AR0B)	
*(ARn - T0B/AR0B)	
*(ARn + T1)	
*(ARn - T1)	
*ARn(T1)	
*ARn(#K16)	*ARn(#K16)
*+ARn(#K16)	*+ARn(#K16)
	*ARn(short(#k3))

CDP 間接アドレッシング・モード :

*CDP
*CDP+
*CDP-
*CDP(#k16)
*+CDP(#k16)

6.8.2.1 *ARn を使用したレジスタ・ビットのアクセス

オペランド	説明
*ARn	生成されるビット・アドレス : [BSAyy +] ARn ARn は変更されません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *AR2, AC3	AR2 AR2 = 0 と仮定。	CPU は AC3 のビット 0 をセットします。AR2 は変更されません。
BTSTP Baddr, src	BTSTP *AR5, AC3	AR5 AR5 + 1 AR5 = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。AR5 は変更されません。

6.8.2.2 *ARn+ を使用したレジスタ・ビットのアクセス

オペランド	説明
*ARn+	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: 1 ビット演算の場合：ARn = ARn + 1 2 ビット演算の場合：ARn = ARn + 2

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *AR2+, AC3	AR2 AR2 = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、AR2 は 1 つインクリメントされます。
BTSTP Baddr, src	BTSTP *AR5+, AC3	AR5 AR5 + 1 AR5 = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR5 は 2 つインクリメントされます。

6.8.2.3 *ARn- を使用したレジスタ・ビットのアクセス

オペランド	説明
*ARn-	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: 1 ビット演算の場合：ARn = ARn - 1 2 ビット演算の場合：ARn = ARn - 2

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *AR2-, AC3	AR2 AR2 = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、AR2 は 1 つデクリメントされます。
BTSTP Baddr, src	BTSTP *AR5-, AC3	AR5 AR5 + 1 AR5 = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR5 は 2 つデクリメントされます。

6.8.2.4 *+ARn を使用したレジスタ・ビットのアクセス

オペランド	説明
*+ARn	1) 変更される ARn: 1 ビット演算の場合：ARn = ARn + 1 2 ビット演算の場合：ARn = ARn + 2 2) 生成されるビット・アドレス : 1 ビット演算の場合：[BSAyy +] ARn + 1 2 ビット演算の場合：[BSAyy +] ARn + 2

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *+AR2, AC3	(AR2 + 1) (AR2 + 1) = 1 と仮定。	アドレスに使用される前に、AR2 は 1 つインクリメントされます。CPU は AC3 のビット 1 をセットします。
BTSTP Baddr, src	BTSTP *+AR5, AC3	(AR5 + 2) (AR5 + 2) + 1 (AR5 + 2) = 30 と仮定。	アドレスに使用される前に、AR5 は 2 つインクリメントされます。CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。

6.8.2.5 *-ARn を使用したレジスタ・ビットのアクセス

オペランド	説明
*-ARn	1) 変更される ARn: 1 ビット演算の場合：ARn = ARn - 1 2 ビット演算の場合：ARn = ARn - 2 2) 生成されるビット・アドレス : 1 ビット演算の場合：[BSAyy +] ARn - 1 2 ビット演算の場合：[BSAyy +] ARn - 2

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *-AR2, AC3	(AR2 - 1) (AR2 - 1) = 1 と仮定。	アドレスに使用される前に、AR2 は 1 つデクリメントされます。CPU は AC3 のビット 1 をセットします。
BTSTP Baddr, src	BTSTP *-AR5, AC3	(AR5 - 2) (AR5 - 2) + 1 (AR5 - 2) = 30 と仮定。	アドレスに使用される前に、AR5 は 2 つデクリメントされます。CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。

6.8.2.6 *(ARn + T0/AR0) を使用したレジスタ・ビットのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T0	*(ARn + AR0)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *(AR2 + T0), AC3	AR2 AR2 = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、AR2 は T0 の数だけインクリメントされます。
BTSTP Baddr, src	BTSTP *(AR5 + T0), AC3	AR5 AR5 + 1 AR5 = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR5 は T0 の数だけインクリメントされます。

6.8.2.7 *(ARn - T0/AR0) を使用したレジスタ・ビットのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - T0	*(ARn - AR0)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *(AR2 - T0), AC3	AR2 AR2 = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、AR2 は T0 の数だけデクリメントされます。
BTSTP Baddr, src	BTSTP *(AR5 - T0), AC3	AR5 AR5 + 1 AR5 = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR5 は T0 の数だけデクリメントされます。

6.8.2.8 *ARn(T0/AR0) を使用したレジスタ・ビットのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*ARn(T0)	生成されるビット・アドレス : [BSAyy +] ARn + T0 ARn は変更されません。ARn はベース・ポインタとして使用されます。T0 がそのベース・ポインタからのオフセットとして使用されます。	*ARn(AR0)	生成されるビット・アドレス : [BSAyy +] ARn + AR0 ARn は変更されません。ARn はベース・ポインタとして使用されます。AR0 がそのベース・ポインタからのオフセットとして使用されます。

構文例	**命令例**	**生成されるアドレス（リニア・アドレッシング）**	**説明**
BSET Baddr, src	BSET *AR2(T0), AC3	(AR2 + T0) AR2 = 0 かつ T0 = 15 と仮定。	CPU は AC3 のビット 15 をセットします。AR2 は変更されません。
BTSTP Baddr, src	BTSTP *AR5(T0), AC3	(AR5 + T0) (AR5 + T0) + 1 AR5 = 25 かつ T0 = 5 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。AR5 は変更されません。

6.8.2.9 *(ARn + T0B/AR0B) を使用したレジスタ・ビットのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0B)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn + AR0B)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *(AR2 + T0B), AC3	AR2 AR2 = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、AR25 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。
BTSTP Baddr, src	BTSTP *(AR5 + T0B), AC3	AR5 AR5 + 1 AR5 = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR5 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。

6.8.2.10 *(ARn - T0B/AR0B) を使用したレジスタ・ビットのアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0B)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn - AR0B)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *(AR2 - T0B), AC3	AR2 AR2 = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、AR2 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。
BTSTP Baddr, src	BTSTP *(AR5 - T0B), AC3	AR5 AR5 + 1 AR5 = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR5 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。

6.8.2.11 *(ARn + T1) を使用したレジスタ・ビットのアクセス

オペランド	説明
*(ARn + T1)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T1

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *(AR4 + T1), AC2	AR4 AR4 = 0 と仮定。	CPU は AC2 のビット 0 をクリアします。アドレスに使用された後、AR4 は T1 の数だけインクリメントされます。
BTSTP Baddr, src	BTSTP *(AR1 + T1), AC2	AR1 AR1 + 1 AR1 = 30 と仮定。	CPU は AC2 のビット 30 と 31 をテストします。AC2(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR1 は T1 の数だけインクリメントされます。

6.8.2.12 *(ARn - T1) を使用したレジスタ・ビットのアクセス

オペランド	説明
*(ARn - T1)	1) 生成されるビット・アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - T1

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *(AR4 - T1), AC2	AR4 AR4 = 0 と仮定。	CPU は AC2 のビット 0 をクリアします。アドレスに使用された後、AR4 は T1 の数だけデクリメントされます。
BTSTP Baddr, src	BTSTP *(AR1 - T1), AC2	AR1 AR1 + 1 AR1 = 30 と仮定。	CPU は AC2 のビット 30 と 31 をテストします。AC2(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、AR1 は T1 の数だけデクリメントされます。

6.8.2.13 *ARn(T1) を使用したレジスタ・ビットのアクセス

オペランド	説明
*ARn(T1)	生成されるビット・アドレス : [BSAyy +] ARn + T1 ARn は変更されません。ARn はベース・ポインタとして使用されます。T1 がそのベース・ポインタからのオフセットとして使用されます。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *AR4(T1), AC2	(AR4 + T1) AR4 = 0 かつ T1 = 15 と仮定。	CPU は AC2 のビット 15 をクリアします。AR4 は変更されません。
BTSTP Baddr, src	BTSTP *AR1(T1), AC2	(AR1 + T1) (AR1+ T1) + 1 AR1 = 25 かつ T1 = 5 と仮定。	CPU は AC2 のビット 30 と 31 をテストします。AC2(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(31) の内容を ST0_55 の TC2 ビットにコピーします。AR1 は変更されません。

6.8.2.14 *ARn(#K16) を使用したレジスタ・ビットのアクセス

オペランド	説明
*ARn(#K16)	生成されるビット・アドレス : [BSAyy +] ARn + K16 ARn は変更されません。ARn はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *AR4(#31), AC2	(AR4 + 31) AR4 = 0 と仮定。	CPU は AC2 のビット 31 をクリアします。AR4 は変更されません。
BTSTP Baddr, src	BTSTP *AR1(#5), AC2	(AR1 + 5) (AR1+ 5) + 1 AR1 = 16 と仮定。	CPU は AC2 のビット 21 と 22 をテストします。AC2(21) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(22) の内容を ST0_55 の TC2 ビットにコピーします。AR1 は変更されません。

6.8.2.15 *+ARn(#K16) を使用したレジスタ・ビットのアクセス

オペランド	説明
*+ARn(#K16)	1) 変更される ARn: ARn = ARn + K16 2) 生成されるビット・アドレス : [BSAyy +] ARn + K16

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *+AR4(#31), AC2	(AR4 + 31) AR4 = 0 と仮定。	AR4 がアドレスに使用される前に、定数が AR4 に加算されます。CPU は AC2 のビット 31 をクリアします。
BTSTP Baddr, src	BTSTP *+AR1(#5), AC2	(AR1 + 5) (AR1 + 5) + 1 AR1 = 16 と仮定。	AR1 がアドレスに使用される前に、定数が AR1 に加算されます。CPU は AC2 のビット 21 と 22 をテストします。AC2(21) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(22) の内容を ST0_55 の TC2 ビットにコピーします。

6.8.2.16 *ARn(short(#k3)) を使用したレジスタ・ビットのアクセス

オペランド	説明
*ARn(short(#k3))	生成されるビット・アドレス : [BSAyy +] ARn + k3 ARn は変更されません。ARn はベース・ポインタとして使用されます。3 ビット符号なし定数（k3）がそのベース・ポインタからのオフセットとして使用されます。k3 の範囲は、1 ～ 7 です。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *AR4(short(#3)), AC2	(AR4 + 3) AR4 = 0 と仮定。	CPU は AC2 のビット 3 をクリアします。AR4 は変更されません。
BTSTP Baddr, src	BTSTP *AR1(short(#5)), AC2	(AR1 + 5) (AR1+ 5) + 1 AR1 = 16 と仮定。	CPU は AC2 のビット 21 と 22 をテストします。AC2(21) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(22) の内容を ST0_55 の TC2 ビットにコピーします。AR1 は変更されません。

6.8.2.17 *CDP を使用したレジスタ・ビットのアクセス

オペランド	説明
*CDP	生成されるビット・アドレス : [BSAC +] CDP CDP は変更されません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *CDP, AC3	CDP CDP = 0 と仮定。	CPU は AC3 のビット 0 をセットします。CDP は変更されません。
BTSTP Baddr, src	BTSTP *CDP, AC3	CDP CDP + 1 CDP = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。CDP は変更されません。

6.8.2.18 *CDP+ を使用したレジスタ・ビットのアクセス

オペランド	説明
*CDP+	1) 生成されるビット・アドレス : [BSAC +] CDP 2) 変更される CDP: 1 ビット演算の場合：CDP = CDP + 1 2 ビット演算の場合：CDP = CDP + 2

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *CDP+, AC3	CDP CDP = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、CDP は 1 つインクリメントされます。
BTSTP Baddr, src	BTSTP *CDP+, AC3	CDP CDP + 1 CDP = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、CDP は 2 つインクリメントされます。

6.8.2.19 *CDP- を使用したレジスタ・ビットのアクセス

オペランド	説明
*CDP-	1) 生成されるビット・アドレス : [BSAC +] CDP 2) 変更される CDP: 1 ビット演算の場合：CDP = CDP - 1 2 ビット演算の場合：CDP = CDP - 2

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BSET Baddr, src	BSET *CDP-, AC3	CDP CDP = 0 と仮定。	CPU は AC3 のビット 0 をセットします。アドレスに使用された後、CDP は 1 つデクリメントされます。
BTSTP Baddr, src	BTSTP *CDP-, AC3	CDP CDP + 1 CDP = 30 と仮定。	CPU は AC3 のビット 30 と 31 をテストします。AC3(30) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC3(31) の内容を ST0_55 の TC2 ビットにコピーします。アドレスに使用された後、CDP は 2 つデクリメントされます。

6.8.2.20 *CDP(#K16) を使用したレジスタ・ビットのアクセス

オペランド	説明
*CDP(#K16)	生成されるビット・アドレス : [BSAC +] CDP + K16 CDP は変更されません。CDP はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *CDP(#31), AC2	(CDP + 31) CDP = 0 と仮定。	CPU は AC2 のビット 31 をクリアします。CDP は変更されません。
BTSTP Baddr, src	BTSTP *CDP(#5), AC2	(CDP + 5) (CDP+ 5) + 1 CDP = 16 と仮定。	CPU は AC2 のビット 21 と 22 をテストします。AC2(21) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(22) の内容を ST0_55 の TC2 ビットにコピーします。CDP は変更されません。

6.8.2.21 *+CDP(#K16) を使用したレジスタ・ビットのアクセス

オペランド	説明
*+CDP(#K16)	1) 変更される CDP: CDP = CDP + K16 2) 生成されるビット・アドレス : [BSAC +] CDP + K16

注： 命令がこのオペランドを使用する場合、定数 K16 はその命令に 2 バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
BCLR Baddr, src	BCLR *+CDP(#31), AC2	(CDP + 31) CDP = 0 と仮定。	CDP がアドレスに使用される前に、定数が CDP に加算されます。CPU は AC2 のビット 31 をクリアします。
BTSTP Baddr, src	BTSTP *+CDP(#5), AC2	(CDP + 5) (CDP + 5) + 1 CDP = 16 と仮定。	CDP がアドレスに使用される前に、定数が CDP に加算されます。CPU は AC2 のビット 21 と 22 をテストします。AC2(21) の内容をステータス・レジスタ ST0_55 の TC1 ビットにコピーし、AC2(22) の内容を ST0_55 の TC2 ビットにコピーします。

6.9 I/O 空間のアドレッシング

絶対、直接、間接のアドレッシングは、I/O 空間のペリフェラル・レジスタをアドレス指定するために使用できます。

アドレッシング・タイプ	参照先
絶対	このページ
直接	P. 6-102
間接	P. 6-102

I/O 空間へのアクセスにはいくつか制約があります。詳細は、6.10 節を参照してください。

6.9.1 I/O 絶対アドレッシング・モードを使用した I/O 空間のアドレッシング

代数表記構文の定義オペランド *port(#k16) またはニーモニック構文の修飾オペランド port(#k16) は、I/O 空間にアクセスするためにのみ使用できます。次の構文要素を使用して、このオペランドを任意の命令に使用できます。

Smem　1 ワード（16 ビット）のデータを表します。

*port(#k16) または port(#k16) を使用して I/O 空間のロケーションにアクセスする例を表 6-22 に示します。

注 :

命令が *port(#k16) **または** port(#k16) **を使用する場合、符号なしの** 16 **ビット定数** k16 **はその命令に** 2 **バイト拡張でエンコードされます。この拡張のため、このオペランドを使用する命令は別の命令と並列に実行することはできません。**

表 6-22.　*port(#k16) または port(#k16) を使用した I/O 空間のアクセス

構文例	命令例	生成されるアドレス	説明
dst = Smem または MOV Smem, dst	AR2 = *port(#2) または MOV port(#2), AR2	k16 = 0002h	CPU は I/O アドレス 0002h の値を AR2 にロードします。
Smem = src または MOV src, Smem	*port(#0F000h) = AR0 または MOV AR0, port(#0F000h)	k16 = F000h	CPU は AR0 の内容を I/O アドレス F000h のロケーションに格納します。

6.9.2 PDP 直接アドレッシング・モードを使用した I/O 空間のアドレッシング

次の構文要素が命令にある場合、I/O 空間にアクセスするために、PDP 直接オペランド @Poffset を使用できます。

Smem　1 ワード（16 ビット）のデータを表します。

データ・メモリ・ロケーションへのアクセスではなく、I/O 空間ロケーションへのアクセスであることを示すために、port() 修飾子を使用する必要があります。代数表記命令を使用する場合、readport() または writeport() の命令修飾子を I/O 空間のアクセスを実行する命令と並列に指定します。ニーモニック命令を使用する場合、port() が修飾されるリードまたはライトのオペランドを囲まなければなりません。

I/O 空間にアクセスするために @Poffset と port() 修飾子を使用する例を表 6-23 に示します。9 ビットのペリフェラル・データ・ページ（PDP）の値が、Poffset の 7 ビットと連結されます。

表 6-23. @Poffset を使用した I/O 空間のアクセス

構文例	命令例	生成されるアドレス（PDP = 511 の場合）	説明
MOV Smem, dst	MOV port(@0), T2	PDP:Poffset = FF80h	0 のオフセットは、現行のペリフェラル・データ・ページの最上部を示します。CPU はペリフェラル・データ・ページ 511 の最上部（アドレス FF80h）の値をコピーし、その値を T2 にロードします。
MOV src, Smem	MOV T2, port(@127)	PDP:Poffset = FFFFh	127 のオフセットは、現行のペリフェラル・データ・ページの最下部を示します。CPU は T2 の内容をコピーし、その内容をペリフェラル・データ・ページ 511 の最下部（アドレス FFFFh）にライトします。

6.9.3 間接アドレッシング・モードを使用した I/O 空間のアドレッシング

次の構文要素が命令にある場合、I/O 空間にアクセスするために、間接オペランドを使用できます。

Smem　1 ワード（16 ビット）のデータを表します。

データ・メモリ・ロケーションへのアクセスではなく、I/O 空間ロケーションへのアクセスであることを示すために、port() 修飾子を使用する必要があります。代数表記命令を使用する場合、readport() または writeport() の命令修飾子を I/O 空間のアクセスを実行する命令と並列に指定します。ニーモニック命令を使用する場合、port() が修飾されるリードまたはライトのオペランドを囲まなければなりません。

I/O 空間へのアクセスをサポートする間接オペランドを図 6-16 に示します。また、この表以下に続く項では、これらのオペランドの詳細と例を示します。これらの例は、ニーモニック命令を使用しています。

図 6-16. I/O 空間のアクセスの間接オペランド

AR 間接アドレッシング・モード :

DSP モード (ARMS = 0):	制御モード (ARMS = 1):
*ARn	*ARn
*ARn+	*ARn+
*ARn-	*ARn-
*+ARn	
*-ARn	
*(ARn + T0/AR0)	*(ARn + T0/AR0)
*(ARn - T0/AR0)	*(ARn - T0/AR0)
*ARn(T0/AR0)	*ARn(T0/AR0)
*(ARn + T0B/AR0B)	
*(ARn - T0B/AR0B)	
*(ARn + T1)	
*(ARn - T1)	
*ARn(T1)	
	*ARn(short(#k3))

CDP 間接アドレッシング・モード :

*CDP
*CDP+
*CDP-

6.9.3.1 *ARn を使用した I/O 空間のアクセス

オペランド	説明
*ARn	生成される I/O アドレス : [BSAyy +] ARn ARn は変更されません。

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV port(*AR4), T2	AR4 AR4 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。AR4 は変更されません。
MOV src, Smem	MOV T2, port(*AR5)	AR5 AR5 = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。AR5 は変更されません。

6.9.3.2 *ARn+ を使用した I/O 空間のアクセス

オペランド	説明
*ARn+	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + 1

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV port(*AR4+), T2	AR4 AR4 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は 1 つインクリメントされます。
MOV src, Smem	MOV T2, port(*AR5+)	AR5 AR5 = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は 1 つインクリメントされます。

6.9.3.3 *ARn- を使用した I/O 空間のアクセス

オペランド	説明
*ARn-	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - 1

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV port(*AR4-), T2	AR4 AR4 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は 1 つデクリメントされます。
MOV src, Smem	MOV T2, port(*AR5-)	AR5 AR5 = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は 1 つデクリメントされます。

6.9.3.4 *+ARn を使用した I/O 空間のアクセス

オペランド	説明
*+ARn	1) 変更される ARn: ARn = ARn + 1 2) 生成される I/O アドレス : [BSAyy +] ARn + 1

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*+AR4), T2	(AR4 + 1) AR4 = FF7Fh と仮定。	アドレスに使用される前に、AR4 が 1 つインクリメントされます。CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。
MOV src, Smem	MOV T2, port(*+AR5)	(AR5 + 1) AR5 = FFFEh と仮定。	アドレスに使用される前に、AR5 が 1 つインクリメントされます。CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。

6.9.3.5 *-ARn を使用した I/O 空間のアクセス

オペランド	説明
*-ARn	1) 変更される ARn: ARn = ARn - 1 2) 生成される I/O アドレス : [BSAyy +] ARn - 1

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*-AR4), T2	(AR4 - 1) AR4 = FF80h と仮定。	アドレスに使用される前に、AR4 が 1 つデクリメントされます。CPU は I/O アドレス FF7Fh の値をリードし、その値を T2 にロードします。
MOV src, Smem	MOV T2, port(*-AR5)	(AR5 - 1) AR5 = FFFFh と仮定。	アドレスに使用される前に、AR5 が 1 つデクリメントされます。CPU は T2 の内容をリードし、その内容を I/O アドレス FFFEh にライトします。

6.9.3.6 *(ARn + T0/AR0) を使用した I/O 空間のアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T0	*(ARn + AR0)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV port(*(AR4 + T0)), T2	AR4 AR4 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は T0 の数だけインクリメントされます。
MOV src, Smem	MOV T2, port(*(AR5 + T0))	AR5 AR5 = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は T0 の数だけインクリメントされます。

6.9.3.7 *(ARn - T0/AR0) を使用した I/O 空間のアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - T0	*(ARn - AR0)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - AR0

構文例	命令例	生成されるアドレス (リニア・アドレッシング)	説明
MOV Smem, dst	MOV port(*(AR4 - T0)), T2	AR4 AR4 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は T0 の数だけデクリメントされます。
MOV src, Smem	MOV T2, port(*(AR5 - T0))	AR5 AR5 = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は T0 の数だけデクリメントされます。

6.9.3.8 *ARn(T0/AR0) を使用した I/O 空間のアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*ARn(T0)	生成される I/O アドレス : [BSAyy +] ARn + T0 ARn は変更されません。ARn はベース・ポインタとして使用されます。T0 がそのベース・ポインタからのオフセットとして使用されます。	*ARn(AR0)	生成される I/O アドレス : [BSAyy +] ARn + AR0 ARn は変更されません。ARn はベース・ポインタとして使用されます。AR0 がそのベース・ポインタからのオフセットとして使用されます。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*AR4(T0)), T2	(AR4 + T0) AR4 = FF7Dh かつ T0 = 3 と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。AR4 は変更されません。
MOV src, Smem	MOV T2, port(*AR5(T0))	(AR5 + T0) AR5 = FFFAh かつ T0 = 5 と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。AR5 は変更されません。

6.9.3.9 *(ARn + T0B/AR0B) を使用した I/O 空間のアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn + T0B)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn + AR0B)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*(AR4 + T0B)), T2	AR4 AR4 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。 アドレスに使用された後、AR4 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。
MOV src, Smem	MOV T2, port(*(AR5 + T0B))	AR5 AR5 = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は T0 の数だけインクリメントされます。リバース・キャリー伝搬が加算時に使用されます。

6.9.3.10 *(ARn - T0B/AR0B) を使用した I/O 空間のアクセス

C54CM = 0		C54CM = 1	
オペランド	**説明**	**オペランド**	**説明**
*(ARn - T0B)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - T0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。	*(ARn - AR0B)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - AR0 （リバース・キャリー伝搬を使用して行われます） サーキュラ・アドレッシングの制約に関する注を参照してください。

注： このビット・リバース・オペランドを使用する場合、ARn をサーキュラ・ポインタとして使用することはできません。ARn が ST2_55 でサーキュラ・アドレッシングに構成されている場合、対応するバッファ・スタート・アドレス・レジスタ値（BSAyy）が ARn に加算されますが、ARn は変更されずにサーキュラ・バッファ内のままです。

構文例	命令例	生成されるアドレス （リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*(AR4 - T0B)), T2	AR4 AR4 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。アドレスに使用された後、AR4 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。
MOV src, Smem	MOV T2, port(*(AR5 - T0B))	AR5 AR5 = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は T0 の数だけデクリメントされます。リバース・キャリー伝搬が減算時に使用されます。

6.9.3.11 *(ARn + T1) を使用した I/O 空間のアクセス

オペランド	説明
*(ARn + T1)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn + T1

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*(AR7 + T1)), AR3	AR7 AR7 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を AR3 にロードします。アドレスに使用された後、AR7 は T1 の数だけインクリメントされます。
MOV src, Smem	MOV AR4, port(*(AR5 + T1))	AR5 AR5 = FFFFh と仮定。	CPU は AR4 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は T1 の数だけインクリメントされます。

6.9.3.12 *(ARn - T1) を使用した I/O 空間のアクセス

オペランド	説明
*(ARn - T1)	1) 生成される I/O アドレス : [BSAyy +] ARn 2) 変更される ARn: ARn = ARn - T1

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*(AR7 - T1)), AR3	AR7 AR7 = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を AR3 にロードします。アドレスに使用された後、AR7 は T1 の数だけデクリメントされます。
MOV src, Smem	MOV AR4, port(*(AR5 - T1))	AR5 AR5 = FFFFh と仮定。	CPU は AR4 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、AR5 は T1 の数だけデクリメントされます。

6.9.3.13 *ARn(T1) を使用した I/O 空間のアクセス

オペランド	説明
*ARn(T1)	生成される I/O アドレス : [BSAyy +] ARn + T1 ARn は変更されません。ARn はベース・ポインタとして使用されます。T1 がそのベース・ポインタからのオフセットとして使用されます。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*AR7(T1)), AR3	(AR7 + T1) AR7 = FF7Dh かつ T1 = 3 と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV src, Smem	MOV AR4, port(*AR5(T1))	(AR5 + T1) AR5 = FFFAh かつ T1 = 5 と仮定。	CPU は AR4 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。AR5 は変更されません。

6.9.3.14 *ARn(short(#k3)) を使用した I/O 空間のアクセス

オペランド	説明
*ARn(short(#k3))	生成されるアドレス : [BSAyy +] ARn + k3 ARn は変更されません。ARn はベース・ポインタとして使用されます。3 ビット符号なし定数（k3）がそのベース・ポインタからのオフセットとして使用されます。k3 の範囲は、1 ～ 7 です。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*AR7(short(#4))), AR3	(AR7 + 4) AR7 = FF70h と仮定。	CPU は I/O アドレス FF74h の値をリードし、その値を AR3 にロードします。AR7 は変更されません。
MOV src, Smem	MOV AR4, port(*AR5(short(#7)))	(AR5 + 7) AR5 = FFF0h と仮定。	CPU は AR4 の内容をリードし、その内容を I/O アドレス FFF7h にライトします。AR5 は変更されません。

6.9.3.15 *CDP を使用した I/O 空間のアクセス

オペランド	説明
*CDP	生成される I/O アドレス : [BSAC +] CDP CDP は変更されません。

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*CDP), T2	CDP CDP = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。CDP は変更されません。
MOV src, Smem	MOV T2, port(*CDP)	CDP CDP = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。CDP は変更されません。

6.9.3.16 *CDP+ を使用した I/O 空間のアクセス

オペランド	説明
*CDP+	1) 生成される I/O アドレス : [BSAC +] CDP 2) 変更される CDP: CDP = CDP + 1

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*CDP+), T2	CDP CDP = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。アドレスに使用された後、CDP は 1 つインクリメントされます。
MOV src, Smem	MOV T2, port(*CDP+)	CDP CDP = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、CDP は 1 つインクリメントされます。

6.9.3.17 *CDP- を使用した I/O 空間のアクセス

オペランド	説明
*CDP-	1) 生成される I/O アドレス : [BSAC +] CDP 2) 変更される CDP: CDP = CDP - 1

構文例	命令例	生成されるアドレス（リニア・アドレッシング）	説明
MOV Smem, dst	MOV port(*CDP-), T2	CDP CDP = FF80h と仮定。	CPU は I/O アドレス FF80h の値をリードし、その値を T2 にロードします。アドレスに使用された後、CDP は 1 つデクリメントされます。
MOV src, Smem	MOV T2, port(*CDP-)	CDP CDP = FFFFh と仮定。	CPU は T2 の内容をリードし、その内容を I/O アドレス FFFFh にライトします。アドレスに使用された後、CDP は 1 つデクリメントされます。

6.10 I/O 空間へのアクセスに関する制約

次の間接オペランドは I/O 空間へのアクセスに使用することはできません。これらのオペランドのいずれかを使用する命令は、定数に対して 2 バイトの拡張を必要とします。この拡張は、I/O 空間のアクセスを識別するために必要な port() 修飾子の使用を妨げます。

表 6-24. I/O 空間へのアクセスをサポートしない間接オペランド

I/O 空間へのアクセスをサポートしないオペランド	ポインタの変更
*ARn(#K16)	ARn は変更されません。ARn はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。
*+ARn(#K16)	16 ビット符号付き定数（K16）がアドレス生成前に ARn に加算されます。
*CDP(#K16)	CDP は変更されません。CDP はベース・ポインタとして使用されます。16 ビット符号付き定数（K16）がそのベース・ポインタからのオフセットとして使用されます。
*+CDP(#K16)	16 ビット符号付き定数（K16）がアドレス生成前に CDP に加算されます。

また、遅延演算を I/O 空間へのアクセスに使用することはできません。このため、表 6-25 の構文は I/O 空間へのアクセスをサポートしていません。

表 6-25. I/O 空間へのアクセスをサポートしない命令構文

I/O 空間へのアクセスをサポートしない構文	命令タイプ
DELAY Smem	メモリの遅延
MACM[R]Z [T3 =] Smem, Cmem, ACx	遅延のある積和演算

I/O 空間に不正にアクセスすると、ハードウェア・バス・エラー割り込み（BERRINT）が生成され、CPU により処理されます。

6.11 サーキュラ・アドレッシング

サーキュラ・アドレッシングは、任意の間接アドレッシング・モードで使用できます。8 つの各補助レジスタ（AR0 ～ AR7）と係数データ・ポインタ（CDP）は、データまたはレジスタ・ビットへのポインタとして動作する場合にリニアまたはサーキュラに変更されるように個別に構成できます。この構成は、ステータス・レジスタ ST2_55 内のビットで行うことができます（表 6-26 を参照）。サーキュラ変更を選択するには、このビットをセットします。

サーキュラ・バッファのサイズは、BK03、BK47、または BKC の 3 つのレジスタのいずれかで指定されます（表 6-26 を参照）。サーキュラ・アドレッシングはワードのバッファとビットのバッファに使用できます。バッファ・サイズ・レジスタは、ワードのバッファ内のワード数を指定するか、レジスタ内のビットのバッファ内のビット数を指定します。

データ空間におけるワードのバッファの場合、バッファはデータ空間の 128 メイン・データ・ページのいずれかに置いておく必要があります。このバッファ内の各アドレスは 23 ビットあり、上位 7 ビットがメイン・データ・ページです。CDPH または ARnH（n は補助レジスタの番号）のメイン・データ・ページをセットします。CDPH は単独でロードできますが、ARnH はその拡張補助レジスタを介してロードする必要があります。たとえば、AR0H をロードするには、XAR0 をロードする必要があります。XARO は、AR0H:AR0 の連結です。メイン・データ・ページ内では、バッファのスタート・アドレスは適切な 16 ビット・バッファ・スタート・アドレス・レジスタにロードする値により指定されます（表 6-26 を参照）。ポインタにロードする値（ARn または CDP）はインデックスとして動作し、スタート・アドレスを基準とするワードを選択します。

ビットのバッファの場合、バッファ・スタート・アドレス・レジスタは参照ビットを指定し、ポインタはその参照ビットの位置を基準とするビットを選択します。ARn または CDP だけをロードする必要があります。XARn と XCDP をロードする必要はありません。

表 6-26. サーキュラ・アドレッシングのポインタおよび関連するビットとレジスタ

ポインタ	リニア / サーキュラ構成ビット	メイン・データ・ページの供給元	バッファ・スタート・アドレスレジスタ	バッファ・サイズ・レジスタ
AR0	ST2_55(0) = AR0LC	AR0H	BSA01	BK03
AR1	ST2_55(1) = AR1LC	AR1H	BSA01	BK03
AR2	ST2_55(2) = AR2LC	AR2H	BSA23	BK03
AR3	ST2_55(3) = AR3LC	AR3H	BSA23	BK03
AR4	ST2_55(4) = AR4LC	AR4H	BSA45	BK47
AR5	ST2_55(5) = AR5LC	AR5H	BSA45	BK47
AR6	ST2_55(6) = AR6LC	AR6H	BSA67	BK47
AR7	ST2_55(7) = AR7LC	AR7H	BSA67	BK47
CDP	ST2_55(8) = CDPLC	CDPH	BSAC	BKC

6.11.1 サーキュラ・アドレッシングの AR0 ～ AR7 および CDP の構成

各補助レジスタ ARn は、ST2_55 に独自のリニア / サーキュラ構成ビットを持っています。

ARnLC	ARn の使用
0	リニア・アドレッシング
1	サーキュラ・アドレッシング

ステータス・レジスタ ST2_55 の CDPLC ビットは、CDP をリニア・アドレッシングまたはサーキュラ・アドレッシングに使用するように DSP を構成します。

CDPLC	CDP の使用
0	リニア・アドレッシング
1	サーキュラ・アドレッシング

命令で使用されるすべてのポインタをサーキュラに変更する場合は、サーキュラ・アドレッシングの命令修飾子を使用します。ニーモニック命令を使用している場合は、ニーモニック命令の末尾に .CR を追加してください（例：ADD.CR）。代数表記命令を使用している場合は、その命令と並列に circular() 修飾子を追加してください（命令 || circular()）。サーキュラ・アドレッシングの命令修飾子は、ST2_55 のリニア / サーキュラ構成を無視します。

6.11.2 サーキュラ・バッファの実行

サーキュラ・バッファを設定する例として、データ・メモリ内のワードのサーキュラ・バッファに対して次の手順を参考にしてください。

1) 適切なバッファ・サイズ・レジスタ（BK03、BK47、または BKC）を初期化します。たとえば、サイズ 8 のバッファの場合、BK レジスタに 8 をロードします。

2) 選択されたポインタに対してサーキュラ変更を選択するために、ST2_55 の適切な構成ビットを初期化します。

3) 上位 7 ビットでメイン・データ・ページを選択するために、適切な拡張レジスタ（XARn または XCDP）を初期化します。たとえば、補助レジスタ 3（AR3）がサーキュラ・ポインタの場合、拡張補助レジスタ 3（XAR3）をロードします。CDP がサーキュラ・ポインタの場合、XCDP をロードします。

4) 適切なバッファ・スタート・アドレス・レジスタ（BSA01、BSA23、BSA45、BSA67、または BSAC）を初期化します。BSA レジスタの内容と連結された、XARn(22 ～ 16) または XCDP(22 ～ 16) のメイン・データ・ページが、バッファの 23 ビット・スタート・アドレスを指定します。

5) 選択されたポインタ ARn または CDP に 0 ～（バッファ・サイズ - 1）の値をロードします。たとえば、AR1 を使用し、バッファ・サイズが 8 の場合、AR1 に 7 以下の値をロードします。

サイズ R のサーキュラ・バッファは、N ビット境界上で開始しなければなりません。ここで、N は $2^N > R$ を満たす最小の整数です。たとえば、バッファ・サイズ R = 8 の場合、N は 4 になります。この場合、サーキュラ・バッファの先頭は、ポインタ（ARn または CDP）の下位 4 ビットが 0（ゼロ）のときに生成されるアドレスです。アドレスをインクリメントしたときにバッファを越える場合、ポインタの 4 つの LSB が強制的に 0（ゼロ）になります。

オフセットつきの間接アドレッシング・オペランドを使用している場合、各オフセットの絶対値が (バッファ・サイズ - 1) 以下であることを確認してください。同様に、サーキュラ・ポインタがプログラムされている数（定数または T0、AR0、T1 で指定）だけインクリメントまたはデクリメントされる場合、その数の絶対値が (バッファ・サイズ - 1) 以下であることを確認してください。

初期化後、次の形式の 23 ビット・アドレスが生成されます。

ARnH:(BSAxx + ARn)
または
CDPH:(BSAC + CDP)

インクリメントとデクリメントは、16 ビットのポインタ（ARn または CDP）に対してのみ行われます。データのメイン・データ・ページへのアドレス指定は、対応する拡張レジスタ（ARnH または CDPH）の値を変更することなく行うことはできません。

注：

FFFFh を超えたインクリメントまたは 0000h を超えたデクリメントを行うとポインタ値が初期状態に戻ってしまいます。この動作はサポートされている機能ではないので、利用してはいけません。また、BSAxx または BSAC を加算する場合、FFFFh を超えたアドレスをインクリメントしてはなりません。

サーキュラ・バッファを初期化し、アクセスするコードを例 6-1 に示します。

例 6-1. C55x サーキュラ・バッファの初期化およびアクセス

```
MOV #3, BK03                ; サーキュラ・バッファのサイズは 3 ワード
BSET AR1LC                  ; AR1 はサーキュラに変更されるように構成
AMOV #010000h, XAR1         ; サーキュラ・バッファはメイン・データ・ページ 01 内
MOV #0A02h, BSA01           ; サーキュラ・バッファ・スタート・アドレスは 010A02h
MOV #0000h, AR1             ; インデックス（AR1 内）は 0000h

MOV *AR1+, AC0              ; AC0 は 010A02h + (AR1) = 010A02h からロード、
                            ; その後 AR1 = 0001h
MOV *AR1+, AC0              ; AC0 は 010A02h + (AR1) = 010A03h からロード、
                            ; その後 AR1 = 0002h
MOV *AR1+, AC0              ; AC0 は 010A02h + (AR1) = 010A04h からロード、
                            ; その後 AR1 = 0000h
MOV *AR1+, AC0              ; AC0 は 010A02h + (AR1) = 010A02h からロード、
                            ; その後 AR1 = 0001h
```

6.11.3 TMS320C54x の互換性

TMS320C54x 互換モード（C54CM ビットが 1）では、サーキュラ・バッファ・サイズ・レジスタ BK03 がすべての補助レジスタに使用され、BK47 は使用されません。TMS320C55x デバイスを使用すると、次のプログラミング・ルールに従って、TMS320C54x のサーキュラ・バッファ管理をエミュレートできます。

- ❏ BK03 を指定されたバッファ・サイズで初期化する。
- ❏ 選択されたポインタに対してサーキュラ動作をセットするために、ST2_55 の適切な構成ビットを初期化する。
- ❏ 上位 7 ビット（ARnH）のメイン・データ・ページで適切な拡張補助レジスタ（XARn）を初期化する。
- ❏ スタート・アドレスをセットするために、ポインタ（ARn または CDP）を初期化する。
- ❏ 適切なバッファ・スタート・
- ❏
- ❏ アドレス・レジスタを 0（ゼロ）に初期化し、無効にする。

間接アドレッシング・オペランドとオフセットを使用している場合、各オフセットの絶対値が (バッファ・サイズ - 1) 以下であることを確認してください。同様に、サーキュラ・ポインタがプログラムされている数（定数または T0、AR0、T1 で指定）だけインクリメントまたはデクリメントされる場合、その数の絶対値が (バッファ・サイズ - 1) 以下であることを確認してください。

C54x のサーキュラ・バッファをエミュレートするコードを例 6-2 に示します。

例 6-2. C54x サーキュラ・バッファのエミュレート

```
MOV #3, BK03              ; サーキュラ・バッファ・サイズは 3 ワード
BSET AR1LC                ; AR1 はサーキュラに変更されるように構成
AMOV #010000h, XAR1       ; サーキュラ・バッファはメイン・データ・ページ 01 内
MOV #0A01h, AR1           ; サーキュラ・バッファ・スタート・アドレスは 010A00h
MOV #0h, BSA01            ; BSA01 は 0 になり無効

MOV *AR1+, AC0            ; AC0 は 010A01h からロード、その後 AR1 = 0A02h
MOV *AR1+, AC0            ; AC0 は 010A02h からロード、その後 AR1 = 0A00h
MOV *AR1+, AC0            ; AC0 は 010A00h からロード、その後 AR1 = 0A01h
```

このサーキュラ・バッファの実行では、サーキュラ・バッファを 8 ワードのアドレス境界にアラインする必要があります。この制約を回避するために、BSA01 をオフセットで初期化できます（例 6-3 を参照）。

例 6-3. 例 6-2 におけるアラインメント制約の回避

```
MOV #3, BK03              ; サーキュラ・バッファ・サイズは 3 ワード
BSET AR1LC                ; AR0 はサーキュラに変更されるように構成
AMOV #010000h, XAR1       ; サーキュラ・バッファはメイン・データ・ページ 01 内
MOV #0A01h, AR1           ; サーキュラ・バッファ・スタート・アドレスは 010A00h
MOV #2h, BSA01            ; 2 のオフセットをバッファ・スタート・アドレスに追加、
                          ; 有効なスタート・アドレスは 010A02h

MOV *AR1+, AC0            ; AC0 は 010A01h + 2h = 010A03h からロード、
                          ; その後 AR1 = 0A02h
MOV *AR1+, AC0            ; AC0 は 010A02h + 2h = 010A04h からロード、
                          ; その後 AR1 = 0A00h
MOV *AR1+, AC0            ; AC0 は 010A00h + 2h = 010A02h からロード、
                          ; その後 AR1 = 0A01h
```

索引

記号

数字

A

B

C

D

E

F

H

I

K

L

M

P

R

S

T

X

Y

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

の

は

ひ

ふ

へ

ほ

ま

め

も

よ

り

る

れ

ろ

わ

日本テキサス・インスツルメンツ株式会社

本　　社　〒160-8366　東京都新宿区西新宿6丁目24番1号　西新宿三井ビルディング3階　03(4331)2000(番号案内)

西日本ビジネスセンター　〒530-6026　大阪市北区天満橋1丁目8番30号　OAPオフィスタワー26階　06(6356)4500(代　表)

■お問い合わせ先

プロダクト・インフォメーション・センター（PIC）――――――――　URL: http: //www.tij.co.jp/pic/

TMS320C55x DSP CPU
リファレンス・ガイド

第 1 版 2004 年 9 月

発行所 **日本テキサス・インスツルメンツ株式会社**
〒160-8366
東京都新宿区西新宿 6-24-1 (西新宿三井ビルディング)

Made in Japan
’04.09

日本テキサス・インスツルメンツ株式会社

SPRU826