CPU 計算部

o 始めに

各 CPU は同一の、独立した計算部を持っている。計算部は、処理の 3 つの型(アドレス、スカラ、ベクトル) に対応した操作レジスタ、機能ユニットから成っている。アドレス処理は、アドレスとインデックスのような内部制御情報の操作を行い、そして、2 レベルの 24bit レジスタと 2 つの整数算術演算機能ユニットを持っている。スカラとベクトル処理はデータについて行われる。

ベクトルとは、順序が付いた要素の集合である。ベクトル命令は、要素の列に、同じ機能を繰り返し、結果の列を生成するという操作を行う。スカラ処理は、1 つの命令を開始し、1 つのオペランドか、オペランドのペアを取り扱う、そして 1 つの結果を生成する。

ベクトルのスカラ処理に対する主な利点は、最初のオペランド以外のすべてのための、命令スタートアップ時間を省けることである。スカラ処理は、2 レベルの 64bit スカラ・レジスタ、単にスカラ処理に専念する 4 つの機能ユニット、ベクトル操作と共用の 3 つの浮動小数点機能ユニットを持つ。ベクトル処理は各 64bit で 64 要素のレジスタの組、ベクトル・アプリケーションに専従する 4 つ(+)の機能ユニット、スカラとベクトルの両操作をサポートする 3 つの浮動小数点機能ユニットを持つ。

(+)第 2 ベクトル論理ユニットがあるシステムでは、5 つのベクトル機能ユニットが利用できる。

アドレス情報は、中央メモリからか、または制御レジスタから、アドレス・レジスタへ流れる。アドレス・レジスタ中の情報は、スカラ、ベクトル、I/O 操作の制御に使用するために、制御ネットワークの様々な部分に配られる。アドレス・レジスタは 2 つの整数機能ユニットにもオペランドを供給できる。そのユニットはアドレスとインデックス情報を生成して、結果をアドレス・レジスタに返す。アドレス情報は、アドレス・レジスタから、中央メモリに送信することもできる。

計算部の中のデータの流れは、中央メモリからレジスタへ、そして、レジスタから、機能ユニットへというものである。結果の流れは、機能ユニットからレジスタへ、そして、レジスタから中央メモリか、または、機能ユニットへ戻る。データの流れは、スカラかベクトルのどちらのパスを伝わるか処理モードに依存している。一つの例外として、スカラ・レジスタは、ベクトル機能ユニットがベクトル操作を実行するために要求された 1 つのオペランドを提供することができる。

整数や浮動小数点の算術演算操作は、計算部で行われる。整数算術演算は、2 の補数モードで動作する。浮動小数点量は、符号付重み表現を持つ。

浮動小数点命令は、加算、減算、乗算、逆数近似を提供する。逆数近似は、複数命令シーケンスを使用した浮動小数点除算操作を提供する。これらの命令は、64bit の結果を生成する(1bit 符号、15bit 指数、48bit 正規化仮数部)。

整数か固定小数点の操作は、整数加算、整数減算、整数乗算である。整数加算と減算操作は、24bit か 64bit の結果を生成する。

整数乗算操作は、乗算部分積を作るために浮動小数点乗算機能ユニットを使用するソフトウェア・アルゴリズムを通して実施される。それから、これらの部分積は 64bit の完全な積を作るために、シフトしてマージされる。整数除算命令は無い; 操作は、浮動小数点ハードウェアを使ったソフトウェア・アルゴリズムを通して成し遂げられる。

命令セットはブーリアン演算、OR,AND,一致(equivalence),排他的論理和(EXOR), マスク制御されたマージ操作、をふくんでいる。シフト操作は 64bit か 128bit のオペランドを操作し、64bit の結果を得る。24it 整数算術演算を例外として、ほとんどの操作はベクトルとスカラの命令として実現されている。整数の積は、インデックス計算のために設計されたスカラ命令である。完全なインデックス機能は、プログラマが、スカラかベクトルどちらのモードでも、メモリを最後までインデックスできる。インデックスは、ベクトル・モードでの、列(row)や斜め(diagonal)の行列操作が、伝統的な欄(column)指向の操作と同じぐらいうまく行くようにする。

ポピュレーション(1 のビットがいくつあるか)とパリティ・カウントはベクトルとスカラ操作の両方で提供されている。加えて

スカラ操作にはリーディング 0 カウント(前に 0 がいくつあるか)がある。

CPU 計算部の特徴は以下のとおり。

・整数と浮動小数点算術演算
・2 の補数の整数算術演算
・符号付き、重み付き浮動小数点算術演算
・アドレス、スカラ、ベクトル処理モード
・13 個の機能ユニット
・8 つの 24bit アドレス(A)レジスタ
・64 個の 24bit 中間アドレス(B)レジスタ
・8 つの 64bit スカラ(S)レジスタ
・64 個の 64bit 中間スカラ(T)レジスタ
・8 個の 64 要素ベクトル(V)レジスタ、各要素は 64bit

o 操作レジスタ (Operating Registers)

操作レジスタ、第一番のプログラム可能な CPU の資源、は、機能ユニットによって作られるデータからの強い要求を満たすことによって、システムの速度を高める。一つの機能ユニットは、要求された機能を実行するために、クロック周期ごとに 1 つから 3 つのオペランドを要求し、そして、CP ごとに 1 つのレートで結果を出すことができる。複数の機能ユニットが並行して使用される。

CPU は 3 つのプライマリ・レジスタの組と 2 つの中間レジスタの組を持っている。プライマリ・レジスタの組は、アドレス、スカラ、ベクトルで、このマニュアルでは、各々、A,S,V と名づけられている。これらのレジスタがプライマリと見なされるのは、機能ユニットが直接にそれらにアクセスできるからである。

A,S レジスタのために、中間レベルのレジスタが存在する。それは、機能ユニットからアクセス可能だが、プライマリ・レジスタのバッファのように働く。ブロック転送はこれらのレジスタと中央メモリの間で可能である。よって、スカラとアドレスをオペランドとして必要とする何種類かのメモリ参照命令を、大幅に減らすことができる。A レジスタをサポートする中間レジスタは、B レジスタとして参照される。S レジスタをサポートする中間レジスタは、T レジスタとして参照される。

o アドレス・レジスタ

図 4-1 は、アドレス処理のために使用されるレジスタと機能ユニットを示している。アドレス・レジスタの 2 つの型が、A レジスタと B レジスタと名付けられていて、以下の段落で説明される。

- A レジスタ

8 つの 24bit の A レジスタは、メモリ参照のためのアドレス・レジスタと、インデックス・レジスタとしての第一の使用以外の様々な適用に役立つ。それらは、シフト・カウンタ、ループ制御、チャンネル I/O 操作のための値を提供し、また、ポピュレーション・カウント、リーディング・ゼロ・カウントの値を受け取る。アドレスの適用において、A レジスタは、スカラ・メモリ参照のためにベース・アドレスをインデックスし、ベクトル・メモリ参照ではベース・アドレスとアドレス増分の両方を提供する。

アドレス機能ユニットは、A レジスタから得たオペランドに 24bit 整数算術演算を実行し、結果を A レジスタの返すことによって、アドレスとインデックスの生成をサポートする。

データは、中央メモリと A レジスタの間で直接に移されるか、B レジスタに置かれる。B レジスタに置かれたデータは、A レジスタと中央メモリ間のデータのバッファとなれる。また、データは、A と S レジスタ間、A と共有アドレス(SB)レジスタ間でも転送が可能である。

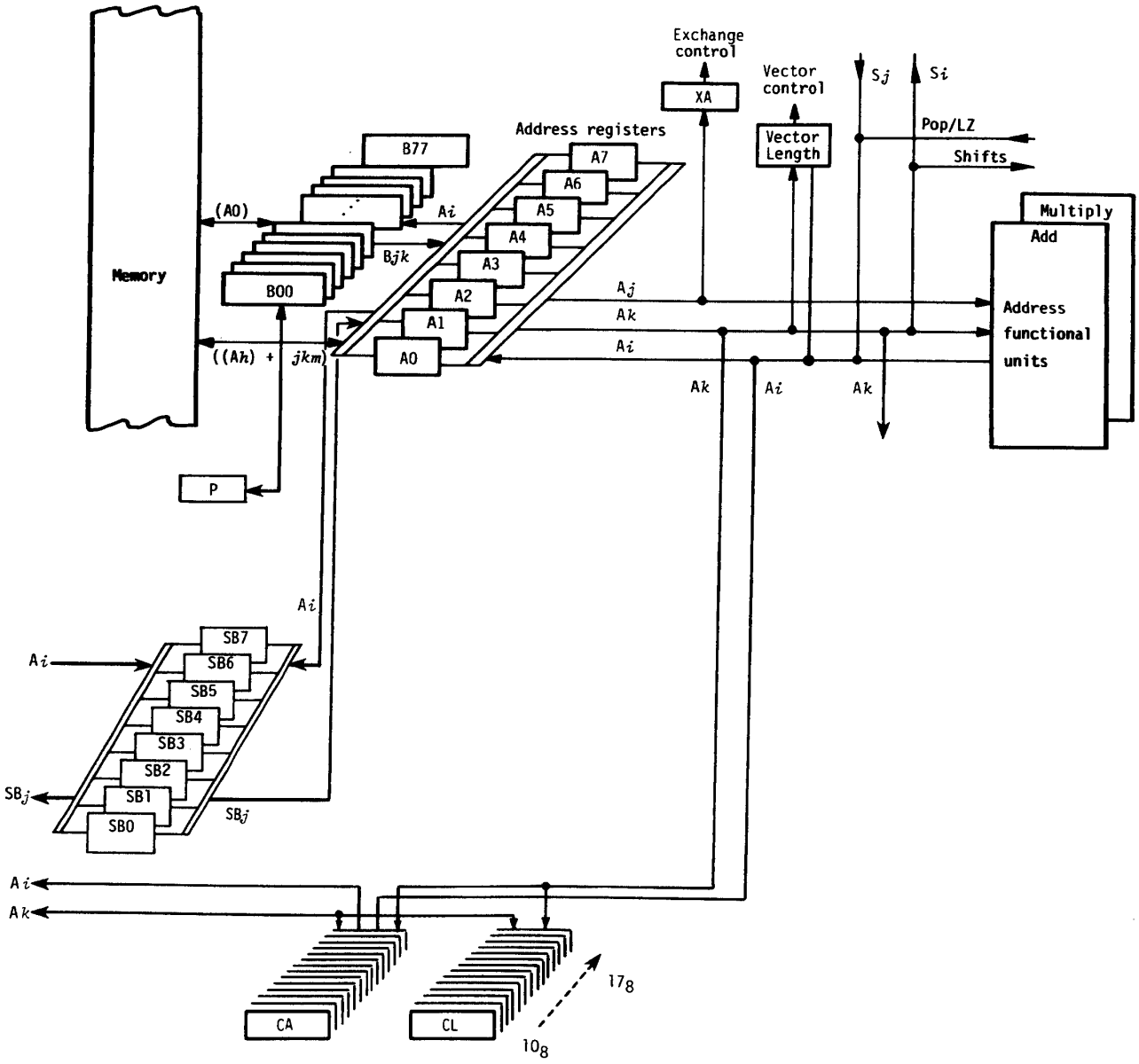


図 4-1. アドレス・レジスタと機能ユニット

ベクトル長(VL)レジスタと交換アドレス(XA)レジスタはAレジスタからそれらの値を転送することによって、セットされる。

新しいデータをAレジスタに届ける命令が発行された時、そのレジスタのための予約がセットされる。その予約は、新しいデータが届くまで、そのレジスタを使用する命令の発行をできなくする。

このマニュアルでは、Aレジスタは、文字Aとそれに続く0から7の数字で、各々参照される。5章で述べるように、命令は、h, i, j, k 指定子のようなレジスタ番号で、Aレジスタを参照する。

次に示す命令では、暗黙に参照されているレジスタだけが、A0レジスタである。

010ijkm JAZ exp ijkm への分岐,もし(A0)=0

011ijkm JAN exp ijkm への分岐,もし(A0)≠0

012ijkm JAP exp ijkm への分岐,もし(A0)が0以上なら

013ijkm JAM exp ijkm への分岐,もし(A0)が負なら

034ijk Bjk,Ai ,A0 (A0)から、B レジスタ jk へ、(Ai)ワードだけ読み込む

035ijk ,A0 Bjk,Ai (A0)へ、B レジスタ jk から、(Ai)ワードだけ格納

036ijk Tjk,Ai ,A0 (A0)から、T レジスタ jk へ、(Ai)ワードだけ読み込む

037ijk ,A0 Tjk,Ai (A0)へ、T レジスタ jk から、(Ai)ワードだけ格納

176i0k Vi ,A0,Ak (A0)から Vi へ、(VL)ワードだけ読み込み、増分は(Ak)。

1770jk ,A0,Ak Vj (A0)へ Vj から、(VL)ワードだけ格納し、増分は(Ak)。

このマニュアルの 5 章は、命令による A レジスタの使用についての追加情報を含んでいる。

- B レジスタ

計算部は、A レジスタのための中間的な記憶として使用される、64 個の 24bit の B レジスタを含んでいる。典型的には、B レジスタは、十分長い期間、繰り返し参照されるデータを持ち、A レジスタや中央メモリのどちらにも保持する必要を無くす。

使用例としては、ループ・カウンタ、変数配列のベース・アドレス、次元(※訳注:配列のための行や列の計算のための不変値であろう)がある。

A レジスタと B レジスタの間の値の転送は 1CP を要す。B レジスタのブロックは中央メモリから/へ、CP 毎に 1 つの 24bit の値という、最大レートで、転送できる。B レジスタから/へ、ブロック転送している間は、すべての B レジスタに予約が成される。

注意

Cray-X-MP では、B レジスタのブロックが中央メモリから/へ、転送が行われている間も、他の命令は発行できる。

このマニュアルでは、B レジスタは、文字 B とそれに続く 00 から 77 の範囲の 2 桁の八進数で、各々参照される。5 章で述べるように、命令は、jk 指定子のような B レジスタ番号を指定し、B レジスタを参照する。

暗黙に参照された B レジスタだけが、B00 レジスタである。リターン・ジャンプ命令(return jump, 007ijkm)の実行時、レジスタ B00 は次の命令パーセルのアドレス(P) にセットされ、ijkm で指定されたアドレスへの分岐が起こる。制御を受け取ったとき、呼び出されたルーチンは取り決めとして、(B00)をセーブする。呼び出されたルーチンはそれ自身の return jump のための初期化をして、B00 レジスタを利用可能にしなければならないということである。呼び出されたルーチンがその呼び出し元に帰ろうとしたとき、セーブしたアドレスを戻し、命令 0050jk を実行する。

取り決めとして、この(Bjk)へ分岐する命令は、Bjk にセーブされたアドレスを、実行されるべき次の命令パーセルのアドレスとして、P へ入れる。

o スカラ・レジスタ

図 4-2 は、スカラ処理に使用されるレジスタと機能ユニットを描いている。スカラ・レジスタの 2 つの型は、S レジスタと T レジスタであり、以下の段落で詳述される。

- S レジスタ

8 つの 64bit S レジスタは、CPU がスカラの算術演算と論理演算命令を実行するためのオペランドとしてのソースとデスティネーションを供給する、主要なスカラ・レジスタである。スカラ機能ユニットは、整数と浮動小数点の両方の算術演算操作を実行する。

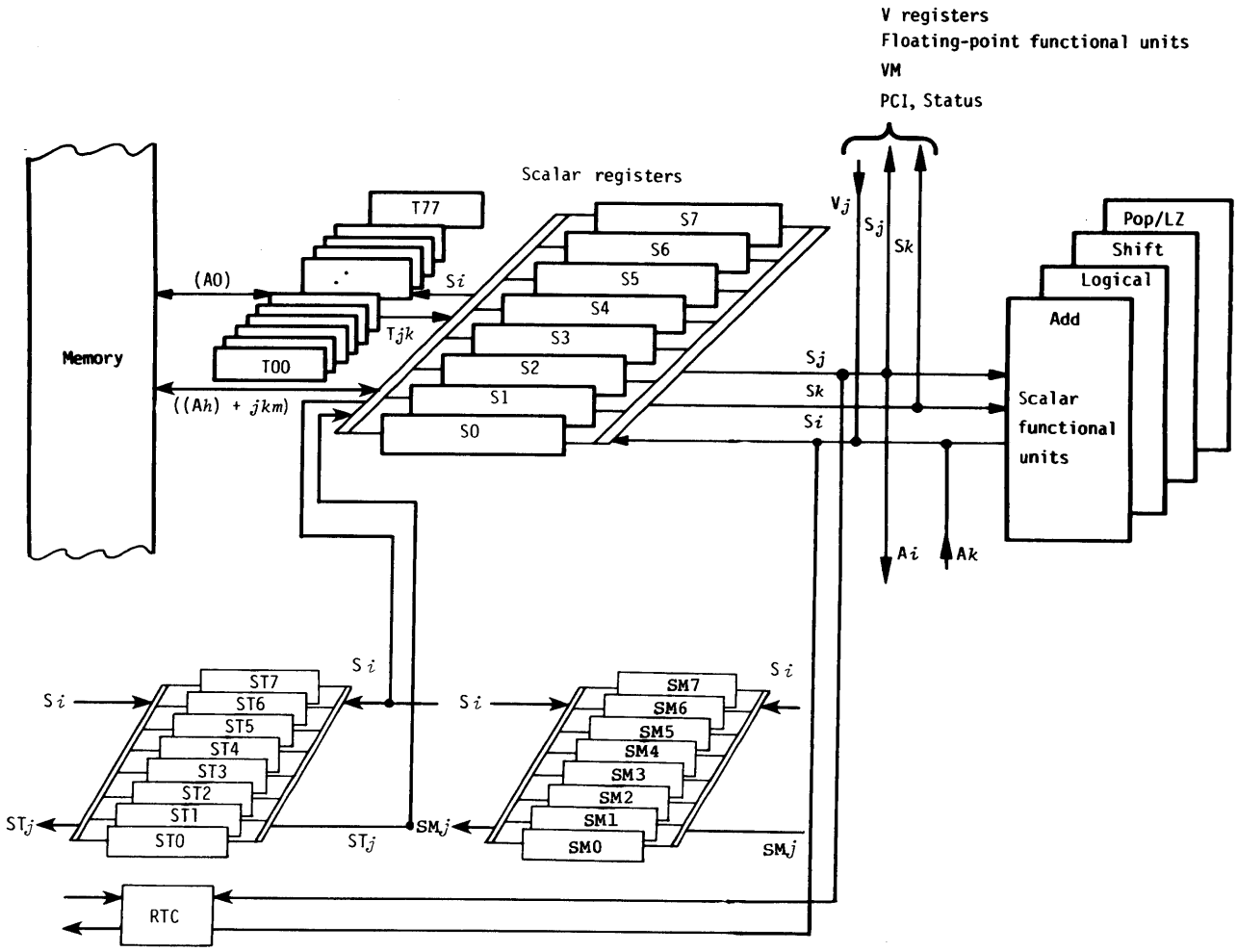


図 4-2. スカラ・レジスタと機能ユニット

Sレジスタはベクトル命令のオペランドを供給できる。SレジスタとVレジスタの要素との間の単一ワードのデータ転送も、可能である。

データは中央メモリとSレジスタの間で直接に移動され、または、Tレジスタに置かれる。この中間的なステップは、Sレジスタと中央メモリの間のスカラ・オペランドをバッファリングすることを可能にする。データは、また、AとBのレジスタ間、Sと共有スカラ(ST)レジスタ間、Sとセマフォ(SM)レジスタ間で転送される。

Sレジスタのその他の使用は、ベクトル・マスク(VM)レジスタか実時間クロック(RTC)レジスタの設定と読み出し、または、割り込みインターバル(II)レジスタの設定がある。

発行中の命令がSレジスタへ新しいデータを供給しようとした時、新しいデータが供給されるまで、そのレジスタを読もうとする命令の発行を止めるための、予約がセットされる。

このマニュアルでは、Sレジスタは、文字Sとそれに続く0から7の範囲の数字で、各々参照される。5章で述べるように、命令は、i, j, k 指定子のようなレジスタ番号で、Sレジスタを参照する。

次に示す命令では、暗黙に参照されているレジスタだけが、S0レジスタである。

014ijkm JSZ exp ijkm へ分岐、もし(S0)=0 なら

015ijkm JSN exp ijkm へ分岐、もし(S0)≠0 なら

016ijkm　JSP exp　ijkm へ分岐、もし(S0)が 0 以上なら

017ijkm　JSM exp　ijkm へ分岐、もし(S0)が負なら

052ijk　S0 Si<exp　(Si)を jk だけ左シフトして、S0 に置く

053ijk　S0 Si>exp　(Si)を jk だけ右シフトして、S0 に置く

8bit のステータス・レジスタは以下のフラグを状態を提供する。

・プロセッサ番号(PN)
・プログラム・ステータス(PS)
・クラスタ番号(CN)
・浮動小数点割り込み許可(IFP)
・浮動小数点エラー(FPE)
・双方向メモリ許可(BDM)
・オペランド範囲割り込み許可(IOR)

命令 073 は、ステータス・レジスタの内容を、S レジスタへ送る。

このマニュアルの 5 章は、命令による S レジスタの使用についての追加情報を含んでいる。

- T レジスタ

計算部は、S レジスタのための中間的な記憶として使用される、64 個の 64bit の T レジスタを含んでいる。データは、T と S レジスタ間と、T レジスタと中央メモリ間で転送される。T レジスタと S レジスタ間の値の転送は、1CP しかかからない。

T レジスタは、ブロック・リードとブロック・ライト命令を通して、中央メモリを参照する。

--
注意

Cray-X-MP では、T レジスタのブロックが中央メモリから/へ、転送が行われている間も、他の命令は発行できる。

--

このマニュアルでは、T レジスタは、文字 T とそれに続く 00 から 77 の範囲の 2 桁の八進数で、各々参照される。5 章で述べるように、命令は、jk 指定子のような T レジスタ番号を指定し、T レジスタを参照する。

o ベクトル・レジスタ

図 4-3 は、ベクトル操作に使用されるレジスタと機能ユニットを説明している。ベクトル・レジスタとベクトル制御レジスタは以下の段落で解説される。

- *V レジスタ*

CPU の主な計算レジスタは各々 64 個の要素を持つ,8 つの V レジスタである。V レジスタの各要素は、64bit である。結びついたデータが、V レジスタの連続した要素としてグループ化される時、レジスタ中の量は、ベクトルとして扱うことができる。ベクトル量の例としては、行列の行や列や、表の要素がある。ベクトルの各要素への同じ処理を行うことによって、計算の効率は実現される。ベクトル命令は、V レジスタの順番の要素への繰り返し処理を提供する。ベクトル操作は、常に、オペランドの V レジスタの最初の要素から、オペランドが得られた時に始まり、そして、結果は V レジスタの最初の要素へ届けられる。順番に並んだ要素は、各 CP 毎に提供され、各操作は実行され、結果の V レジスタの連続した要素へ結果は届けられる。

命令によって実行される操作の回数が、VL レジスタの内容で指定された回数と同じになるまで、ベクトル操作はは続く。

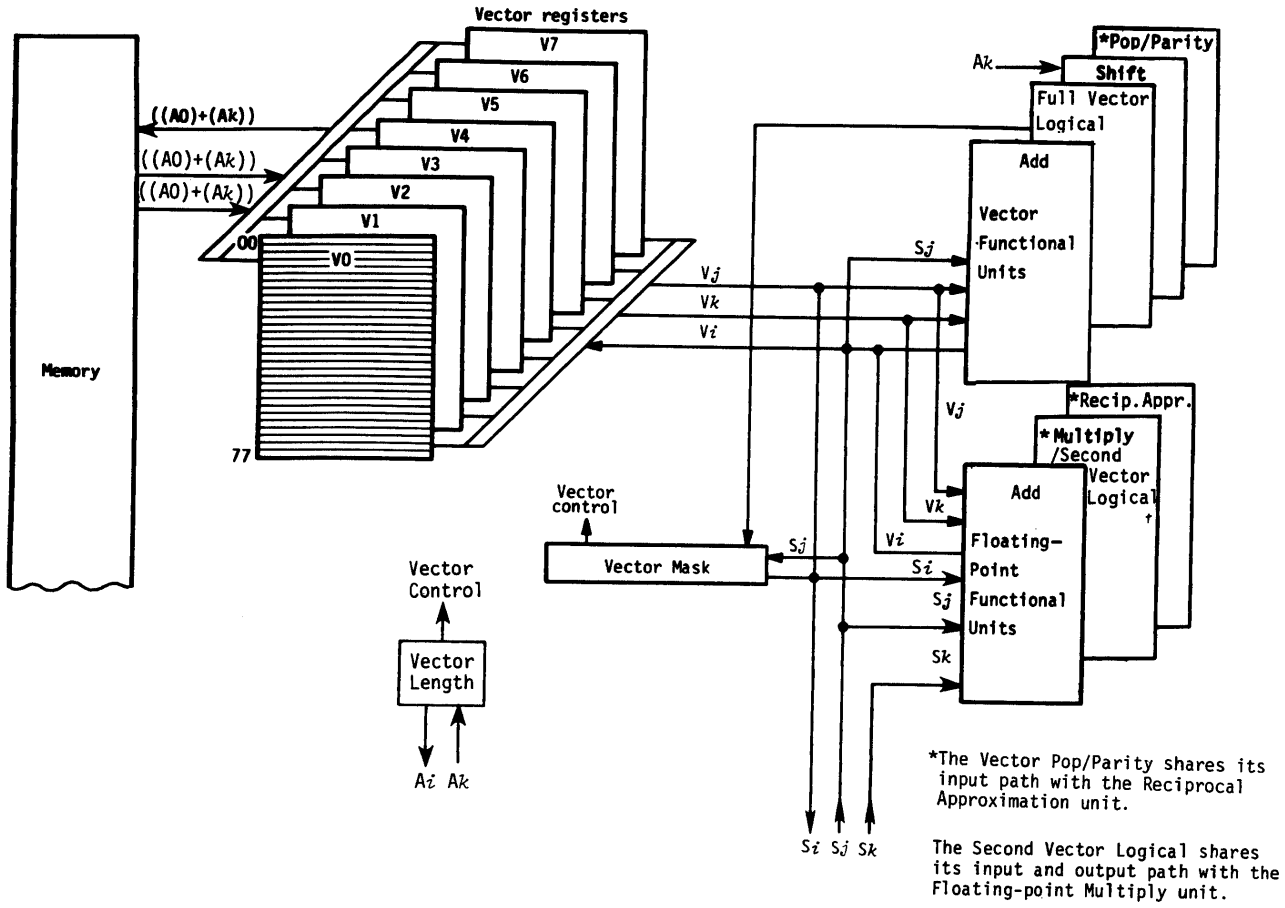


図 4-3. ベクトル・レジスタと機能ユニット

--
+ 第 2 ベクトル論理機能ユニットを装備したシステム上
--

Vレジスタの内容は中央メモリへ/から、中央メモリの最初のワード・アドレス、中央メモリのアドレスの増加か減少、ベクトル長を、指定することによって、ブロック・モードで転送される。そして、転送は、Vレジスタの最初の要素から始まり、バンク衝突に依存するが、最大 1CP あたり 1 ワードの率で進行する。ベクトル・データ・ストリームの切れ目はメモリ衝突の結果として、発生する。

これらの切れ目は、チェイン化された操作を禁止していないけれども、チェイン化された操作のデータ・ストリームに現れる。データ・ストリーム中のどの切れ目もベクトル操作の総実行時間に比例して加算される。

単一ワード・データの転送は、Sレジスタと Vレジスタの要素の間で可能である。

多くのベクトルは 64 要素を超えているから、長いベクトルは 1 つ以上の 64 要素のセグメントと、64 要素より小さい可能な残りとして処理される。一般的に、64 要素のセグメントの残りの数の分を処理をする前に、残りを計算し、この短いセグメントを処理することは、簡便である。しかし、プログラマは、いくつかの方法のうちベクトル・ループ・コードを構成することを選択できる。FORTRAN での長いベクトルの処理はコンパイラによって取り扱われ、また、プログラマに平明である。

Vレジスタの結果の受け取りは、引き続く操作にオペランドを供給することができる。2 つの異なる操作で、結果とオペランドの両方にレジスタを使用することは、2 つ以上のベクトル操作を一緒にチェイン化させ、2 つ以上の結果を CP ごとに生成することを可能にする。チェイン化された操作は、CPU によって自動的に検出され、プログラマによって明示的に指定されることはない。プログラマは、より多くの並行性、つまり、チェイン化された操作の可能性を得られるように、いくつかのコード断片を並び替えることができる。

衝突は、浮動小数点操作と、メモリ・アクセスに関するベクトルとスカラ操作の間で発生する。これらの操作の例外として、機能ユニットはいつもスカラ操作のために用意されている。ベクトル操作は、選択された機能ユニットを、ベクトルが処理されるまで占有する。

並列ベクトル操作は 2 つの方法で処理される:

・違う機能ユニットと、すべて異なるＶレジスタを使用する。

・1 つのＶレジスタからの結果ストリームを、異なる機能ユニットを使用する違う操作のオペランドとして、同時に、使用する。(チェイン・モード)

ベクトルでの並列操作は、CP 毎での、2 つ以上の結果の生成を可能にする。

ほとんどのベクトル操作は、2 つのＶレジスタをオペランドとしてか、1 つのＳと 1 つのＶレジスタをオペランドとして使用する。例外として、ベクトルのシフト、ベクトルの論理演算、ベクトルの逆数、ロード/ストア命令がある。

このマニュアルでは、Ｖレジスタは、文字Ｖとそれに続く 0 から 7 の範囲の数で、各々参照される。5 章で述べるように、ベクトル命令は、i,j,k 指定子のようなＶレジスタ番号を指定し、Ｖレジスタを参照する。

Ｖレジスタの各要素は、このマニュアルでは、十進の 00 から 63 の範囲の数で指定されるこれらは、ベクトル・レジスタ参照の添え字として現れる。例えば、V6 29 は、Ｖレジスタ 6 の、要素 29 を参照する。

注意
Ｖレジスタの並列ロードとストアは可能である;2 つのロード操作と 1 つのストア操作が同時に発生可能である。

- Ｖレジスタ予約とチェイン

予約は、レジスタの使用状態を記述する;すなわち、レジスタが他の操作のための、結果レジスタであるか、オペランド・レジスタであるために、利用不可能である。各レジスタは 2 つの予約状態を持っていて、1 つは、オペランド・レジスタとして予約されていること、もう 1 つは、結果レジスタとして予約されていること。ベクトル命令の実行の間予約は、オペランドＶレジスタと、結果Ｖレジスタの上に置かれる。これらの予約は、Ｖレジスタの個別の要素にではなく、レジスタそのものに置かれる。

もし、Ｖレジスタが、オペランドではなく、結果として予約された時、いつでもオペランドとして使用でき、チェインが発生する。この柔軟なチェインの機構は、結果ベクトル・データストリーム中のいかなる位置でもチェインが開始できるようにする。結果の要素 0 がＶレジスタに到達する以前に、もしチェインを起こす命令が発行されたら、完全なチェインが発生する。部分的なチェインは、要素 0 が到達した後に、命令が発行されたときに、発生する。このように、チェインされた操作の並行性の総合計は、チェインする命令の発行時間と結果ベクトル・データ・ストリームとの間の関係に依存している。

もしＶレジスタがオペランドとして予約されたら、オペランドの予約がクリアされるまで、それは結果やオペランドのレジスタとして使用できない。しかし、同じベクトル操作中に、Ｖレジスタはオペランドと結果の両方に使用することができる。Ｖレジスタは、1 つのベクトル操作だけに 1 つか 2 つのオペランドのソースとして供することができる。Ｖレジスタは結果として、1 つのベクトル操作だけに供することができる。

ベクトル処理中も VL レジスタは予約されることがない。もしベクトル命令が、Ｓレジスタを利用しても、Ｓレジスタには予約はされない。ベクトル関係の次の、ベクトル操作に影響のない、命令はＳレジスタを変更することができる。各ベクトル操作の、長さと(もし使用するなら)スカラ・オペランドは、VL レジスタとＳレジスタとは別に維持されている。異なる長さを利用する複数のベクトル操作は、並行に進めることができる。

ベクトル・メモリ参照の A0 と Ak レジスタは同様に扱われ、使用後、即座に変更のために利用できる。

****
注意

もし、CRAY-1 と CRAY X-MP システムの間の互換性が必要ならば、CRAY リサーチのすべての計算機にはベクトル再帰が無いため、Cray リサーチ社は、ベクトル・レジスタの結果とオペランドの両方としての使用に対して警告する。
*****

o ベクトル制御レジスタ

ベクトル長(VL)レジスタとベクトル・マスク(VM)レジスタは、ベクトル操作の実行で必要とされている制御情報を提供している。そして、以下で述べる。

- ベクトル長レジスタ

7bit のベクトル長(VL)レジスタは、1 から 100(八進)(VL=0 は VL=100(八進)である)を設定され、ベクトル命令によって実行される、すべてのベクトル操作の長さと、V レジスタに保持されるベクトルの長さを指定する。

VL レジスタは、命令 140 から 177 が実行する操作の数を制御し、命令 0020 で A レジスタの値に設定され、また、命令 023i01 で読み出される。

- ベクトル・マスク・レジスタ

ベクトル・マスク(VM)レジスタは 64bit で、それぞれ V レジスタのワード要素に対応している。ビット 2^63 は要素 0 に対応し、ビット 2^0 が要素 63 である。マスクは、ベクトルのマージ、と、各ベクトル要素に実行されるべき操作を許可するテスト命令と一緒に使用される。

VM レジスタは、S レジスタから命令 003 を通して設定可能であり、また、命令 175 を使用し、その条件で V レジスタをテストすることによって、(値を)生成することができる。マスクは、ベクトル・マージ命令(146 と 147)おける、要素の選択を制御する。命令 073 は VM レジスタの内容を S レジスタに送る。

- 機能ユニット

単純な転送と制御操作以外の命令は、機能ユニットとして知られる、特化したハードウェアで実行される。各ユニットは、アルゴリズムか命令セットの一部を実現している。機能ユニットは、逆数近似とベクトル・ポピュレーション・カウント・ユニット(この章で後ほど詳述)がいくつかの論理を共有していることを除いて、独立した論理を持っている。(第 2 ベクトル論理機能ユニットを持ったシステム上では、浮動小数点乗算と第 2 ベクトル論理ユニットは入力と出力のパスを共有している)全機能ユニットは同時に動作できる。

機能ユニットは、レジスタからオペランドを受け取り、機能の実行が終わった時、結果をレジスタへ届ける。機能ユニットは基本的に、レジスタ指定子に限定されたソースとデスティネーションのアドレスを持った、3 アドレス・モードで動作する。
全機能ユニットは固定時間でアルゴリズムを実行する;オペランドが一度ユニットへ届けられたら、遅延は起こりえない。オペランドの配送から、機能ユニットが計算の完了までに必要な時間が、「機能ユニット時間(functional unit time)」と呼ばれ、9.5 ナノ秒の CP 単位で計測される。

機能ユニットは完全にセグメント分けされている。これは、機能ユニット時間が 1CP より大きくても、無関係なオペランドの新しい組が、CP 毎に機能ユニットに入れることを意味する。このセグメント化は情報が、機能ユニットにやって来た時に機能ユニット内でホールドされ、また、各 CP の終わりに機能ユニット中で転送されることを可能にする。

このマニュアル中で認識される機能ユニットはしいて言えば、4 つのグループに分けられる:アドレス、スカラ、ベクトル、浮動小数点。

最初の 3 つのグループのそれぞれは、メインフレーム中に存在する、アドレス、スカラ、ベクトルのモードの処理をサポートするために、根本的なレジスタ型(A,S,V)の一つとともにある。4 つめのグループ、浮動小数点、はスカラとベクトル操作のどちらかをサポートし、S か V レジスタからのオペランドを受付け、結果をそれらに届ける。加えて、中央メモリは、ベクトル操作のために、機能ユニットのように振舞う。

o アドレス機能ユニット

アドレス機能ユニットは A レジスタから獲得したオペランドに、24bit 整数算術演算を実行し、A レジスタに結果を送る。算術演算は 2 の補数である。

- アドレス加算機能ユニット

アドレス加算機能ユニットは 24bit 整数加算と減算を実行する。ユニットは命令 030 と 031 を処理する。加算と減算は同様のやり方で実行される。命令 031 のための 2 の補数減算は、Ak オペランドの 1 の補数が Aj オペランドに加えられる時に、発生する。その時、結果の最下位ビットの位置に、1 が加えられる。アドレス加算機能ユニットでは、オーバーフローは検出されない。

アドレス加算機能ユニットの機能ユニット時間は 2CP である。

- アドレス乗算機能ユニット

アドレス乗算機能ユニットは命令 032 を実行し、2 つの 24bit オペランドから 24bit の整数積を生成する。いかなる丸めも行われない。結果は、積の下位 24bit が取られる。

この機能ユニットはアドレス・データの容量を超えないアドレス操作を扱うように設計されている。プログラマは、機能ユニットの内部で整数の乗算をしている時、注意しなければならない、なぜなら、ユニットは積のオーバーフローを検出せず、積の上位部分は失われてしまうから。

アドレス乗算機能ユニットのユニット時間は 4CP である。

o スカラ機能ユニット

スカラ機能ユニットはSレジスタから取ってきた64bitオペランドに操作を実行して、多くの場合、64bitの結果をSレジスタに届ける。例外として、ポピュレーション/リーディング0カウント機能ユニットがその7bitの結果をAレジスタの送る。

これから述べるように、4つの機能ユニットは、独占的にスカラ操作に割り当てられている。浮動小数点の節で述べるように、3つの機能ユニットはスカラとベクトルの両方の操作に使用される。

- スカラ加算機能ユニット

スカラ加算機能ユニットは64bit加減算と命令060と061を実行する。加減算は同様のやり方で実行される。命令061のための2の補数減算は、Skオペランドの1の補数がSjオペランドに加えられる時に、発生する。その時、結果の最下位ビットの位置に、1が加えられる。スカラ加算機能ユニットでは、オーバーフローは検出されない。

スカラ加算機能ユニットの機能ユニット時間は3CPである。

- スカラ・シフト機能ユニット

スカラ・シフト機能ユニットはSレジスタの64bit内容や2つのSレジスタをつないだ128bitの倍の内容をシフトする。シフト・カウントはAレジスタか命令のjk部分から取られる。シフトはゼロ・フィル(0を満たす)で終わる。ダブル・シフトではシフト・カウントが64を超えず、iとj指定子が同じで、0でない時、サーキュラ・シフト(循環シフト)が動作する。

スカラ機能ユニットは命令052から057を実行する。単一シフト命令(052から055)は機能ユニット時間が2CPである。ダブル・シフト命令(056と057)は機能ユニット時間が3CPである。

- スカラ論理機能ユニット

スカラ論理機能ユニットは64bit量のbit-by-bitの取り扱いを、Sレジスタから持って来て実行する。それは、命令042から051、マスク、ブーリアン命令を実行する。命令042から051は機能ユニット時間1CPである。

- スカラ・ポピュレーション/パリティ/リーディング・ゼロ・カウント・ユニット

このユニットは、命令026と027を実行する。命令026ij0は、Sレジスタ中のオペランドの中で1であるビットの数を数え、機能ユニット時間は4CPである。命令026ij1は、Sレジスタの内容の1bitのポピュレーション・パリティ・カウント(偶数パリティ)を返す。命令027はオペランドの1であるビットの前にある0のビットの数を数え、機能ユニット時間は3CPである。これらの命令では、64bitオペランドがSレジスタから取られ、7bitの結果がAレジスタに送られる。

o ベクトル機能ユニット

ほとんどのベクトル機能ユニットが 1 つか 2 つの V レジスタからか、V レジスタと S レジスタからオペランドを取って来て操作を実行する。一つしかオペランドを必要としない、逆数、シフト、ポピュレーション/パリティ機能ユニットは例外である。ベクトル機能ユニットからの結果は V レジスタに届けられる。

連続するオペランドの組は、毎 CP ごとに機能ユニットに送られる。対応する結果は、n CP 後に機能ユニットから出現する。ここで n は機能ユニット時間であり、与えられた機能ユニットで一定である。VL レジスタは機能ユニットで処理されるべきオペランドの組の数を決定する。

この節で述べられる機能ユニットはベクトル操作に独占的に割り当てられている。3 つの機能ユニットがベクトルとスカラ操作の両方に割り当てられていて、浮動小数点機能ユニットという表題の節で説明されている。浮動小数点機能ユニットがベクトル操作で使用される時、その節で与えられるベクトル機能ユニットの一般的な説明が、適用される。

- ベクトル機能ユニットの予約

ベクトル操作を引き受けた機能ユニットは、各 CP の間ビジーが続き、他の操作に参加することはできない。この状態では、その機能ユニットは予約される。同じ機能ユニットを要求する他の命令は、前の操作が完了するまで発行できない。各タイプのただ一つの機能ユニットだけが、ベクトル命令ハードウェアに用意される(第 2 ベクトル論理ユニットを装備したシステムは例外)。ベクトル操作が完了した時、予約は落とされ、機能ユニットはその時、他の操作のために利用可能になる。ベクトル機能ユニットは(VL)+4 CP だけ予約される。

- ベクトル加算機能ユニット

ベクトル加算機能ユニットはベクトル操作のための 64bit の加減算を実行し、結果を V レジスタの要素へ届ける。ユニットは、命令 154 から 157 を実行する。加減算は同様のやり方で実行される。減算操作(156,157)のために、Vk オペランドの補数を取ってから加算し、そして結果の最下位ビットの位置に、1 が加えられる。オーバーフローはユニットによって検出されない。

ベクトル加算機能ユニットのユニット時間は 3CP である。

- ベクトル・シフト機能ユニット

ベクトル・シフト機能ユニットは V レジスタの要素の内容の 64bit か、V レジスタの 2 つ並んだ要素の 128bit の値をシフトする。シフト・カウントは A レジスタから来て、ゼロ・フィルされる。

全てのシフト・カウントは正の符号無し整数として扱われる。2^6 より高いビットがセットされていたら、シフトの結果はすべて 0 である。

ベクトル・シフト機能ユニットは命令 150 から 153 を実行する。命令 152 の機能ユニット時間は 4CP、命令 150,151,153 の能ユニット時間は 3CP である。

- フル・ベクトル論理機能ユニット

フル・ベクトル論理機能ユニットは命令 140 から 147 の、64bit 量の bit-by-bit の操作を実行する。また、フル・ベクトル

論理機能ユニットは、命令 175 のベクトル・マスクに対応する論理操作も実行する。命令 175 は命令 140 から 147 と同じ機能ユニットを使用するので、これらの命令をチェインすることはできない。

注意

第 2 ベクトル論理ユニットが付加されたシステムで、そのユニットがイネーブルな時、命令 175 は命令 140 から 145 とチェインできるだろう。これが起こるには、命令 140 から 145 は、フル・ベクトル論理機能ユニットではなく、
第 2 ベクトル論理ユニットを使用しなければならない。

フル・ベクトル論理機能ユニットの機能ユニット時間は 2CP である。

- 第 2 ベクトル論理ユニット (+)

第 2 ベクトル論理ユニットは命令 140 から 145 の、64bit 量の bit-by-bit の操作を実行する。

命令 140 から 145 が CIP にある時、2 つのベクトル論理ユニット(フル・ベクトル論理機能ユニットと第 2 ベクトル論理ユニット)のどちらを使用するかの選択が行われる。もし、第 2 ベクトル論理ユニットがイネーブルなら(交換パッケージを通して)、命令 140 から 145 は、そちら(※訳注:第 2 ベクトル論理ユニット)で先に発行する。もし、ユニットがビジーなら、フル・ベクトル論理機能ユニットで発行する。両方のユニットがビジーなら先にクリアされるユニットが発行のために選ばれる。しかし、もし他の衝突が第 2 ベクトル論理ユニットに現れていたら(例えば、レジスタ予約)、第 2 ベクトル論理ユニットがビジーでないとしても、命令はフル・ベクトル論理機能ユニットで先に発行する

(+) すべてのデュアル・プロセッサ・システムで利用できない

注意

第 2 ベクトル論理ユニットと浮動小数点乗算機能ユニットは、データパスの入出力を共有しているので、それらは同時には使用できない。第 2 ベクトル論理ユニットがイネーブルな時、2 つのユニットが同じ「機能ユニット*ビジー*信号」を共有している。また、第 2 ベクトル論理ユニットは浮動小数点乗算機能ユニットと連携しているため、浮動小数点積に依存しているいくつかのコードは、第 2 ベクトル論理ユニットがイネーブルだと遅く走行することがある。
--

第 2 ベクトル論理ユニットは、ソフトウェアを通して、ユーザ・プログラムの交換パッケージのワード 3 のビット 0 をクリアすることで、ディスエーブルすることができる。第 2 ベクトル論理ユニットがディスエーブルの時(交換パッケージの「イネーブル第 2 ベクトル論理ビット」がクリアされている)、そのユニットの機能ユニットビジー信号が常にセットされているように見え、命令 140 から 145 はすべてフル・ベクトル論理ユニットが使うようになる。

第 2 ベクトル論理ユニットの機能ユニット時間は、4CP である。

- ベクトル・ポピュレーション/パリティ機能ユニット

ベクトル・ポピュレーション/パリティ機能ユニットは、ソースＶレジスタの各要素中の 1 であるビットをカウントする。1 であるビットの総数がポピュレーション・カウントである。ポピュレーション・カウントは奇数か偶数であり、それはその最下位ビットに示される。

命令 174ij1(ベクトル・ポピュレーション・カウント)と 174ij2(ベクトル・ポピュレーション・カウント・パリティ)は、ベクトル・逆数近似命令と同じオペレーション・コードを使う。逆数近似機能ユニットのいくつかの制限は、ベクトル・ポピュレーション命令にも当てはまる(逆数近似の節を参照)ベクトル・ポピュレーション・カウント命令はポピュレーション・カウントの総数をデスティネーション Ｖレジスタの要素に送る。

ベクトル・ポピュレーション・カウント・パリティ命令は、カウントの最下位ビットをデスティネーション Ｖレジスタに送る。

ベクトル・ポピュレーション/パリティ機能ユニットのユニット時間は 5CP である。

o 浮動小数点機能ユニット

3 つの浮動小数点機能ユニットがスカラとベクトルの算術演算を実行する。スカラ命令を実行する時、オペランドは Ｓレジスタから取られ、結果は Ｓレジスタへ届けられる。

ほとんどのベクトル命令を実行する時、オペランドは、Ｖレジスタのペアか Ｓレジスタと、Ｖレジスタから取られ、結果は Ｖレジスタへ届けられる。例外として、逆数近似ユニットは 1 つの入力オペランドしか必要としない。

浮動小数点範囲外条件の情報は、浮動小数点算術演算の節に含まれている。

- 浮動小数点加算機能ユニット

浮動小数点加算機能ユニットは浮動小数点形式の 64bit オペランドの加減算と命令 062,063 と 170 から 173 を実行する。結果は、オペランドが非正規化でも、正規化される。(正規化浮動小数点数は浮動小数点算術の節で述べられる)指数部の範囲からの逸脱は、検出され、浮動小数点算術の節で述べられる。

浮動小数点加算機能ユニットの機能ユニット時間は 6CP である。

- 浮動小数点乗算機能ユニット

浮動小数点乗算機能ユニットは、命令 064 から 067 と 160 から 167 を実行する。これらの命令は、浮動小数点形式の 64bit オペランドの全精度と半精度の乗算と、逆数の繰り返しのために、2 から浮動小数点積を減ずる計算を、実行する。

半精度の積は丸められる;全精度の積は丸めたり、丸めなかったが可能である。

入力オペランドは正規化されていると仮定されている。浮動小数点乗算機能ユニットは、両方の入力が正規化されている時のみ、正規化された結果を届ける。

注意

第 2 ベクトル論理機能ユニットを装備したシステムでは、浮動小数点乗算と第 2 ベクトル論理機能ユニットは、それらの入力と出力データ・パスを共有しているため、同時には使用できない。どちらかの予約は、他方の予約でもある。

指数の範囲からの逸脱は、検出され、浮動小数点算術の節で述べられる。しかし、2 つのオペランドが指数 0 を持っているとき、結果は整数積とみなし、正規化されず、範囲外とはみなさない。この場合、48bit 整数積の計算の高速な手法が提供される、しかし、この場合、オペランドは乗算操作の前に、シフトされなければならないが。

浮動小数点乗算機能ユニットの機能ユニット時間は 7CP である。

- 逆数近似機能ユニット

逆数近似機能ユニットは、浮動小数点形式の 64bit オペランドの逆数近似を求める。命令 070 と 174ij0 を実行する。このユニットとベクトル・ポピュレーション/パリティ機能ユニットは、いくつかの論理を共有しているので、逆数近似命令が認識されるには、k 指定子は 0 でなければならない。

結果が正しいためには、入力オペランドは正規化済みであることが想定されている。仮数の最上位ビットは 1 であることが想定され、テストされない。指数部の範囲からの逸脱は、検出され、浮動小数点算術の節で述べられる。

逆数近似機能ユニットの機能ユニット時間は 14CP である。

- 算術演算

CPU 内の機能ユニットは、2 の補数整数算術演算か、浮動小数点算術演算を行う。

o 整数算術演算

全部の整数算術演算、24bit か 64bit、は、2 の補数で、レジスタ上では図 4-4 に示すような表現である。アドレス加算とアドレス乗算機能ユニットは 24bit 算術演算を実行する。スカラ加算とベクトル加算機能ユニットは 64bit 算術演算を実行する。

2 つのスカラ(64bit)整数オペランドの乗算は、浮動小数点乗算命令を使用して遂行され、2 つの方法のうちの一つは次のようである。その方法は、オペランドの大きさと積の入るビット数に依存して使用される。

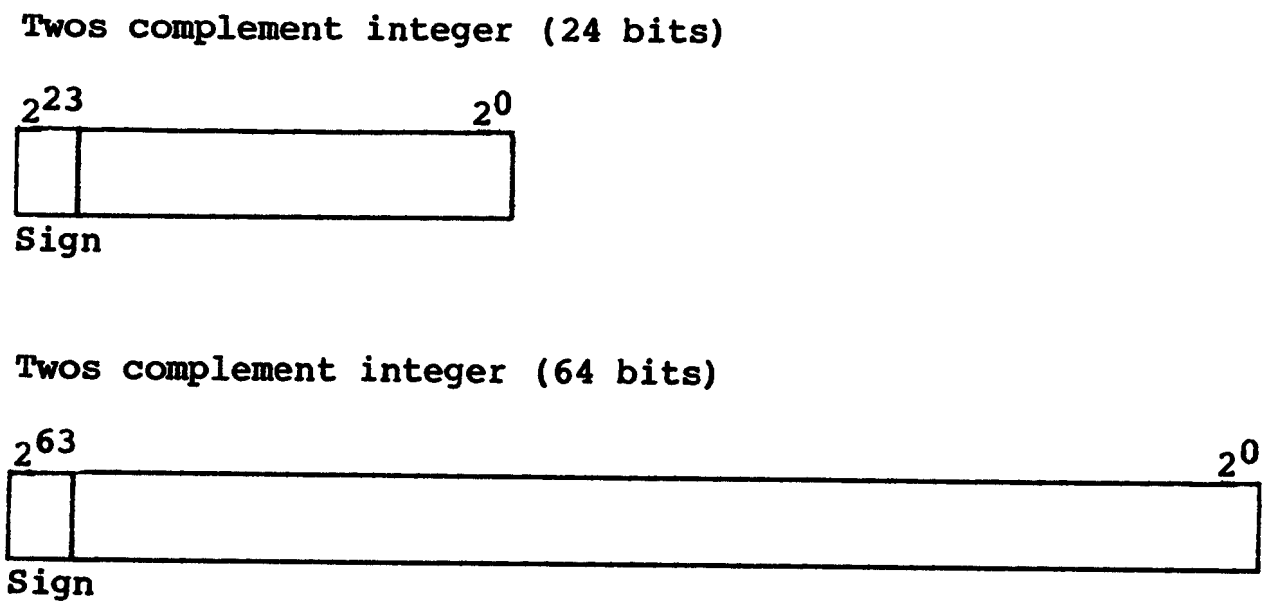


図.4-4. 整数データ形式

もし、両オペランドの下 24bit のビットだけが 0 でなく、乗算操作前に、2 つの整数オペランドが各々、24bit 左シフトによって乗算できる。(浮動小数点乗算機能ユニットは、両オペランドが指数 0 であるという特別な場合を認識する。)浮動小数点乗算機能ユニットは、結果は指数が 0 で仮数に出て、その積の仮数の上位 48bit を返す。図 4-7 を参照。下 24bit が 0 でなくシフト以外でオペランドの仮数を生成される場合、積の上位 48bit は非 0 であろう、そして上位 48bit(返される部分)は期待されるものより 1 大きい、乗算中に常に切り捨て補正値が加えられるから。

もし、オペランドが 24bit より大きい時、乗算は複数の部分積の計算と部分積のシフトと加算で実現される。

除算はアルゴリズムで実現される;

商のビット数に依存した特別なアルゴリズムが使用される。最高速でもっともよく使用される方法は、数値を浮動小数点形式に変換し、浮動小数点機能ユニットを使用することである。

o 浮動小数点算術演算

浮動小数点数はCPU全体で標準の形式で表現されている。この形式は、2進の仮数部と指数部(2のべき乗)を詰め合わせた表現である。仮数部は符号付の48bit小数である。図4-5に示すように、仮数部の符号は仮数部の残りから切り離されている。仮数は符号付の重み数(※訳注:絶対値と符号で表す)であるから、負数でも補数化されない。

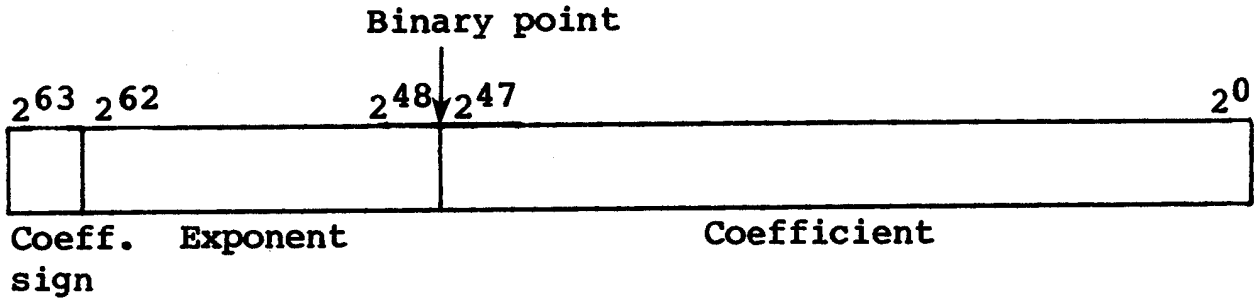


図.4-5. 浮動小数点データ形式

浮動小数点形式の指数部はビット2^62から2^48にバイアスのかかった整数として表現されている。(※訳注:いわゆるオフセット・バイナリである。0が中心値となるように、バイアスを加える)バイアスは指数に40000(八進)(※訳注:4000(16進), 16384(十進))を加える。指数の正数の範囲は40000(八進)から57777(八進)である。指数の負数の範囲は37777(八進)から20000(八進)である。そして、バイアスのかかっていない指数の範囲は次のようである(注意,負数の範囲は1つ大きい):

2^-20000(八進) から 2^+17777(八進)

十進数の言い方では、システムの浮動小数点形式は、数の正確な表現は、およそ15十進桁で、十進でおよそ10^-2466から10^+2466の範囲を許している。

値0やアンダーフローした結果はバイアスされず、全部0のワードとして表現される。

負数の0はどの浮動小数点機能ユニットでも生成されないが、例外として、浮動小数点乗算機能ユニットの1つのオペランドとして負数の0が入った時がある。

正規化浮動小数点数、浮動小数点範囲エラー、倍精度数、そして、加算、乗算、除算アルゴリズムはこの節の残りで説明される。

- 正規化浮動小数点数

もし仮数の最上位ビットが非0であれば、非0の浮動小数点数は正規化されている。この状態は、仮数は可能な限り左シフトされ、指数はそれに応じて調整されていることを暗示している。つまり、浮動小数点数は仮数にリーディング・ゼロを持たない。例外として、正規化された浮動小数点数の0は、すべて0である。

浮動小数点数が、40060(八進)の指数を48bitの整数ワードに挿入することによって生成される時、その結果は浮動小数点操作で使用される前に正規化されなければならない。正規化は、正規化されていない浮動小数点オペランドに0を加算することによって遂行される。S0が64bitの0を提供するので、命令のSjフィールドが使用される時、Sk中のオペランドは命令062i0kを使って正規化される。Si,それはSkでもあり,は正規化された結果を含む。

170i0k命令はVkを正規化してViに置く。

- 浮動小数点範囲エラー

浮動小数点の範囲のオーバーフローは、指数の値が 60000(八進)か、またはパック形式より大きいかで示される。オーバーフロー状態の検出は、割り込みを開始する、ただし、モード・レジスタ中の浮動小数点モード・フラグがセットされていて、かつモニタ・モードが無効である時に。浮動小数点モード・フラグはユーザ・モード・プログラムからセットやクリアできる。

Cray オペレーティング・システム(COS)は、モード・ビットの状態を示すためのテーブルの中にビットを保持している。システム・ソフトウェアはモード・ビットを操作し、ユーザのためにどんなモードを残すべきかを示すテーブルのビットを使用する。つまり、通常、ユーザはユーザがモードを変えたい時に、割り当てられたビットをテーブル中に置く必要がある。

浮動小数点範囲エラー状態は、次の段落で述べるように、浮動小数点機能ユニットで検出される。

- 浮動小数点加算機能ユニット - 浮動小数点加算範囲エラー状態は、スカラ・オペランドに、60000(八進)以上の指数がやってきたときに発生する。この状態は、浮動小数点エラー・フラグをセットし、指数 60000(八進)を、計算した仮数と一緒に結果レジスタに送る、次の例のように:

```
 60000.4xxxxxxxxxxxxxx 範囲エラー
+57777.4xxxxxxxxxxxxxx
----------
 60000.6xxxxxxxxxxxxxx 結果レジスタ
```

注意

もし、加減算の結果が、機械の最小値より小さい時、エラーは省略される(たとえ、両オペランドが 60000(八進)以上の指数を持っていたとしても)、なぜなら、機械の最小値はエラー検出に先立って取られるためである。

- 浮動小数点乗算機能ユニット - 範囲外状態が起きているか否かに関わらず、そしてそれらがどう取り扱われるかは、図 4-6 に示されるような指数の行列を使用して決定できる。結果の指数は、いかなる指数の組でも、一意に、7 つのうちの一つの領域に落ちる。各領域は以下。

注意

もしどちらかのオペランドが機械の最小値より小さい時、エラーは省略される(たとえ、もう一つのオペランドが範囲外であったとしても)、なぜなら、機械の最小値はエラー検出に先立って取られるためである。

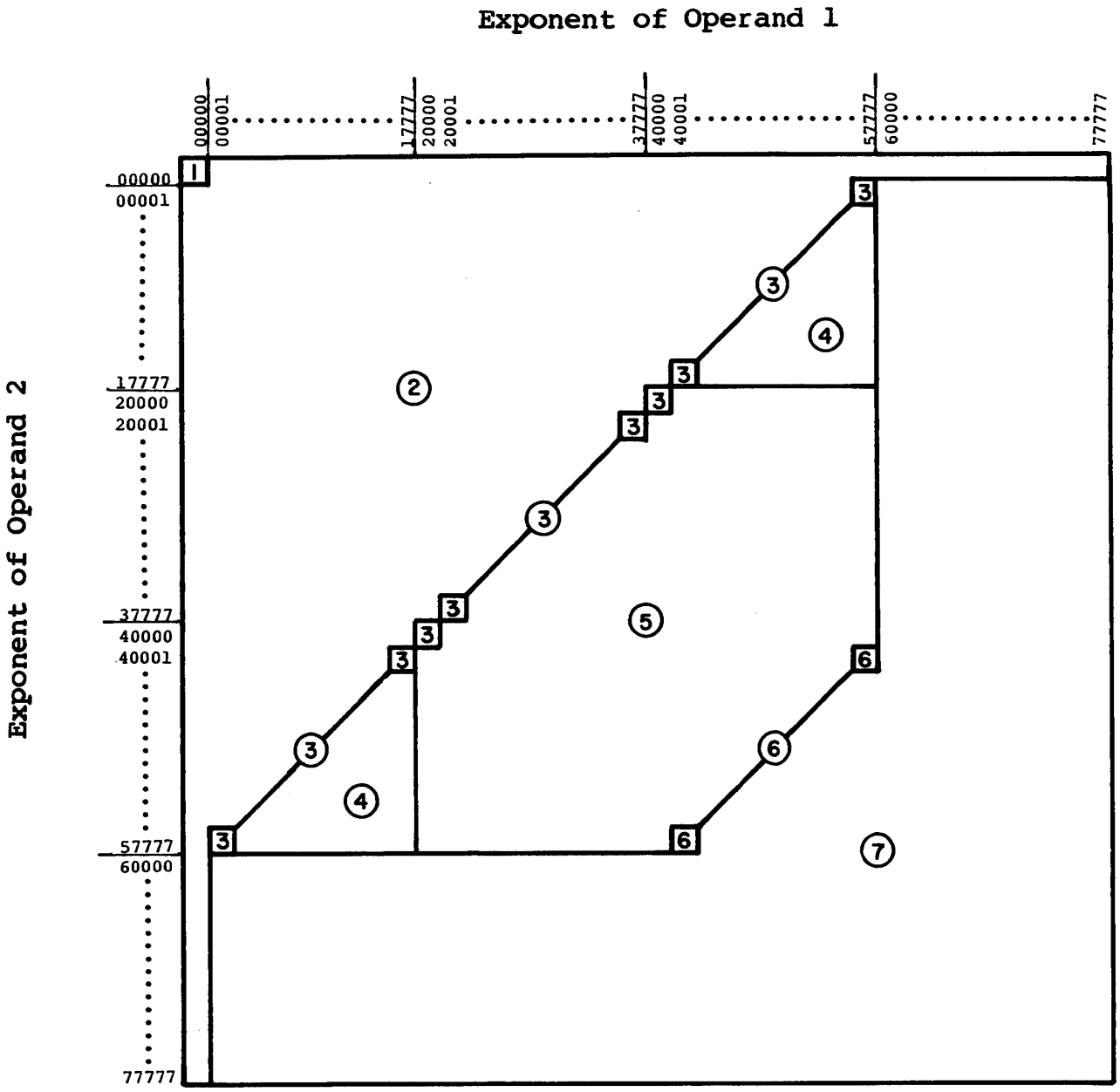


図 4-6. 浮動小数点乗算ユニットのための指数行列

領域　　説明

1 単純な整数乗算を示す; フォールとは起こり得ない。

2 これらの指数は結果がアンダフロー状態であろう。それは、結果が+0 で知らせられる。(0 による乗算はこのグループ)

3 この境界では、アンダフローは起きるだろう。最終の指数は 17777(八進)か 20000(八進)で、正規化のシフトが必要か否かに依存している。
もし指数が 17777(八進)で正規化のシフトが必要なら、アンダーフローは検出され得なく、指数と仮数が 0 になることはない。アンダーフロー検出はシフトされない生成結果の仮数部を使用するための指数部について行われる。

4 もし、最終結果が 20000(八進)から 57777(八進)の範囲なら、アンダフロー指数が使用される。

5 通常のオペランドの範囲で、通常の結果が生成される。

6 この境界では、オーバフローが通知される。もし正規化シフトが必要なら、値は、指数 57777(八進)に結び付けられる。しかし、非正規化シフト状態の指数(それは 60000(八進))を使用して、オーバフローが、検出されるので、60000(八進)が最終の指数として積に挿入される。

7 この領域では、オーバフロー・フォールトは通知され、指数は 60000(八進)としてセットされる。

範囲外の状態は浮動小数点乗算機能ユニット内では正規化前にテストされる。

上に示したように、やってきた両方の指数が 0 であるとき、演算は整数乗算として取り扱われる。その結果は通常、結果の正規化シフト無しとして扱われる。結果は 2^47 から始まる 48bit の量である。この機能を使う時、オペランドは、2^47 から 2^24 にある、24bit 整数と想定される。図 4-7 中のように、もしオペランド 1 が 4 でオペランド 2 が 6 の時、30(八進)という 48bit の積が結果となる。ビット 2^63 は符号の乗算の通常の規則に従い、そして、結果は符号無し整数と符号である。注意として、整数の形式(図 4-4)は整数加算と減算には許容され、2 の補数(符号と符号無し数ではない)のソフトウェアには受け入れられない。したがって、負の積は変換されなければならない。

もし、図 4-7 のオペランド 1 と 2 のビット 2^0 から 2^23 が、1 であるビットを持っていれば、切捨て補正の定数が乗算処理中に加えられるので、積は 1(2^0)であり、大きすぎる。(後の段落で、切捨て補正の定数とその使用について議論する)図 4-7 のオペランド 1 と 2 の影付きの部分の大きさは、両オペランドで同じである必要はない。正しい積を得るには、影付き領域のビット数の合計が 48bit 以上であることだけが要求されている。もし合計が 48bit より大きければ、積の二進小数点(binary point)は合計が 48 を超過した分だけ位置の数だけ左にある(つまり、オペランドの二進小数点が、影付き領域の境界の左にあると考える)

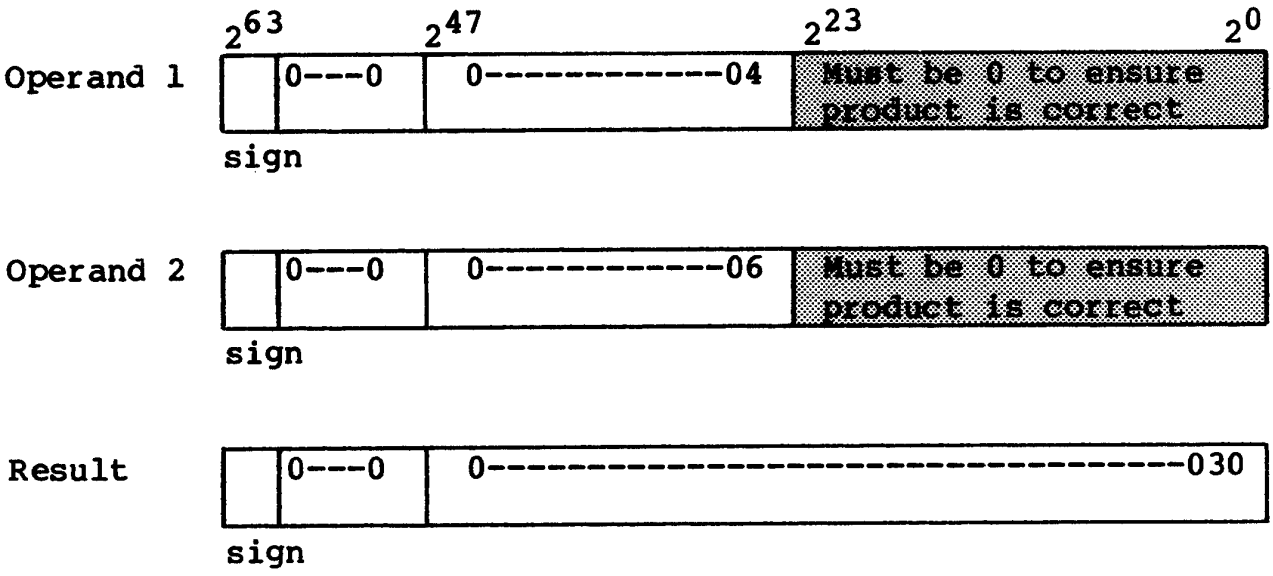


図 4-7. 浮動小数点乗算機能ユニットでの整数乗算

- 浮動小数点逆数近似機能ユニット - 浮動小数点逆数近似機能ユニットのために、20001(八進)以下か 60000(八進)以上の指数を持ってやってきたオペランドは浮動小数点範囲エラーを起こす。エラー・フラグがセットされ指数 60000(八進)と計算された仮数が結果レジスタに送られる。

- 倍精度数

CPU は、倍精度や、多倍精度の操作のための特別なハードウェアを提供していない。95bit 精度を持つ倍精度計算は、Cray Research, Inc.が提供するソフトウェアのルーチンを通して利用できる。

- 加算アルゴリズム

浮動小数点加算と減算は、49bit レジスタで実行される(図 4-8)。

オペランドを並べるための、シフトダウンされるオペランドを選ぶために、指数の減算を試行する。大きいほうの指数のオペランドが符号を持つ。大きい指数の数の仮数と並ぶように、小さい指数の方の数の仮数部が右へシフトされる。シフトでレジスタからこぼれたビットは失われる;切り上げは一切無い。もし和のキャリーが上位ビットに出たとき、下位ビットは捨てられ、指数は適切に補正される。すべての結果は正規化され、もし、結果が機械最小値より小さい時は、エラーは省略される。

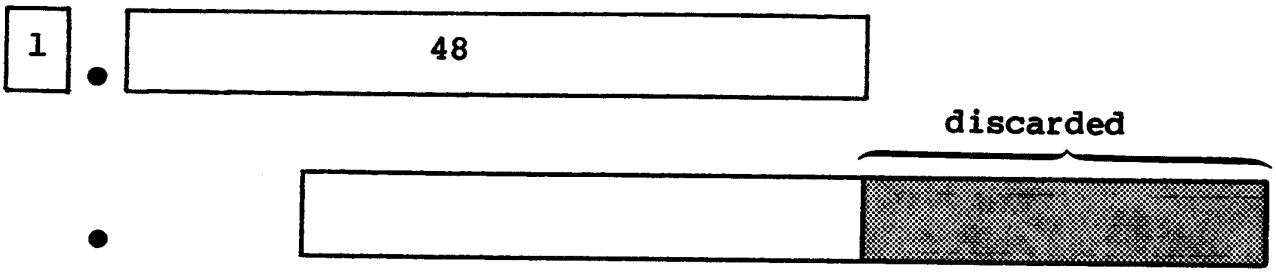


図 4-8. 49bit 浮動小数点加算

浮動小数点加算機能ユニットはメインフレーム(訳注:Cray X-MP の中という意味)の浮動小数点数システムの形式のいかなる浮動小数点数でも正規化する。機能ユニットは結果を正規化するために、結果ごとに、右へ 1 シフトか左へ最大 48 までシフトする。

1 つの 0 オペランドと 1 つの正しいオペランドは浮動小数点加算機能ユニットへ送ることができ、正しいオペランドは正規化ユニットを通して送られる。並行に、機能ユニットはオーバフローとアンダフローをチェックする;アンダフローはエラーのフラグを立てない。

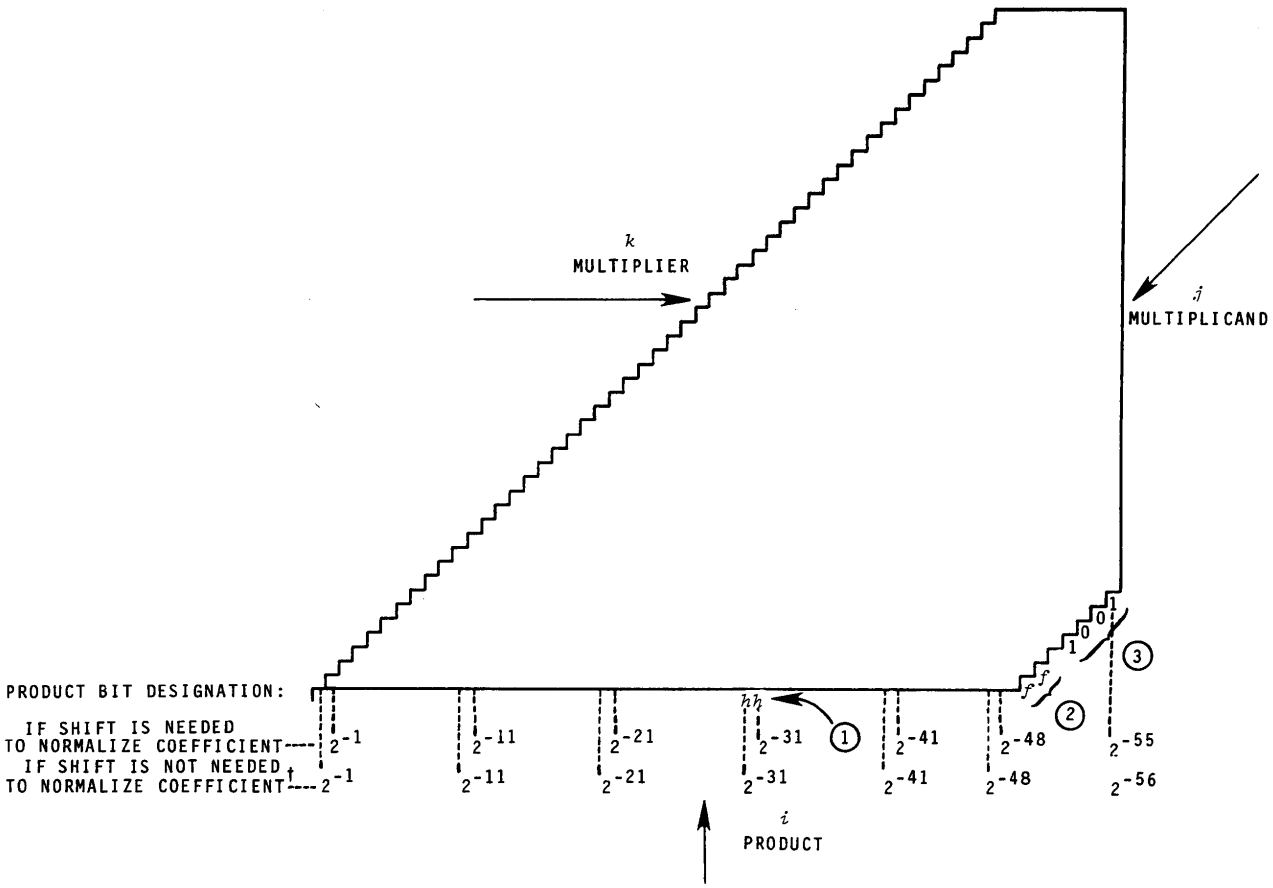
- 乗算アルゴリズム

浮動小数点乗算機能ユニットは乗算ピラミッド(図 4-9)の入力として 2 つの 48bit 仮数を取る。もし、両方の仮数が正規化済みなら、完全な積は 95bit か 96bit であり、そのどちらであるかは仮数の値に依存している。96bit 積は生成とともに正規化済みである。95bit 積は最終の仮数を生成するために、左へ 1 シフトが必要である。もし、シフトが行われれば、そのシフトを反映するために、最終の指数は 1 減らされる。

続く議論は、生成されたものが、最終の形式がシフトの不要であるような積を想定した、2 のべき乗の指定子使用する。

システムにおいて、ピラミッドは 96bit 積の下位ビットの部分を切り捨てる。この切捨てを補正するために、切捨ての下で、無条件に定数を加える。この切捨ての平均的な値は 9.25×2^-56 であり、これは、可能な組み合わせの数だけの積の、切捨てられるであろうキャリーをすべて加えたもの、と、可能な組み合わせの数だけ合計を分割したもの、から

決定されている。9 個のキャリーは、切り捨てられたビットを補償するために、2^-56 の位置に注入を行う。



(1) hh = 11_2 for half-precision round, 00_2 for full-precision rounded or full-precision unrounded multiply

(2) ff = 11_2 for full-precision round, 00_2 for half-precision rounded or full-precision unrounded multiply

(3) Truncation compensation constant, 1001_2 used for all multiplies

図 4-9. 浮動小数点乗算部分積合計ピラミッド

脚注
+ ビット指定子は、浮動小数点乗算機能ユニット操作の説明のために使用される

補正なしの切捨ての影響は、せいぜい、結果の仮数が期待されたものより 1 小さいということである。

大きすぎる 1 からと、小ささすぎる 1 までが結果の範囲で、12˜48 のビット位置で補正する。生成されたものから約 99% が偏差 0 で、完全な 96bit ピラミッドが出力する。乗算は可換である;つまり、A 掛ける B は、B 掛ける A である。

切捨て補正ではない、丸めはオプションである。丸めの方法は、定数を加える、つまり、平均してほぼ 0 の丸め誤差と

なる結果をつくるのに、全場合の 38%の 50%を切り上げ(.25×2^-48;高い)、全場合の 62%の 25%を切捨て(.125×2^-48;低い)する。(※訳注:1/2 ビット×2^-48 が丸められるビットである。それの 50%確率の期待値としては、0.25×2^-48。25%の期待値は 0.125×2^-48)全精度の丸め付き乗算では、ピラミッドの 2^-50 と 2^-51 のビット位置に、2bit の丸めビットが入れられ、ピラミッドに伝播できる。

半精度乗算では、ピラミッドの 2^-32 と 2^-31 のビット位置に、丸めビットが入れられる。このエントリからの結果のキャリーは、上に伝播させることができ、正規化した結果の 29MSB(最上位ビット)が送り返される。

この切捨てと丸めの偏差はつぎの範囲である:
-0.23×2^-48 から +0.57×2^-48 または
-8.17×10^-16 から +20.25×10^-16

フル 96bit ピラミッドでの丸めは、1/2 最下位ビット(LSB)と同一である、偏差は次のように期待される:

-0.5×2^-48 から +0.5×2^-48

- 除算アルゴリズム

システムは、完全に部分分けされた機能ユニットのハードウェアの実装を容易にするために、浮動小数点除算を逆数近似を使って。この部分分けのおかげで、オペランドは各 CP の間に逆数ユニットに入る。ベクトル・モードでは、結果は、1CP の率で生成され、それは、チェインニングの間、他のベクトル操作で使用される。なぜなら、システム中の全機能ユニットは同じ結果のレートを持っているからである。(※訳注:全機能ユニットは 1CP ごとに結果を出す)逆数近似はニュートン法に基づいている。

- ニュートン法

除算アルゴリズムは任意の方程式 F(x)=0 の真の解の近似のための、ニュートン法の応用である。ここで、F(x)は二回微分可能で連続な二次派生関数を持たなければならない。

この方法は、初期におよその値(推測値)を作ることが必要である,x0 真の解と交わるので十分、xt が求める解(図 4-10 参照)。よりよい近似のために y=F(x)のグラフの(x0, F(x0))の点にタンジェントの線を引く。このタンジェント線と出会う X は、よりよい近似 x1 である。これを、次は、x1 を使って、x2 を探すというように、繰り返すことができる。

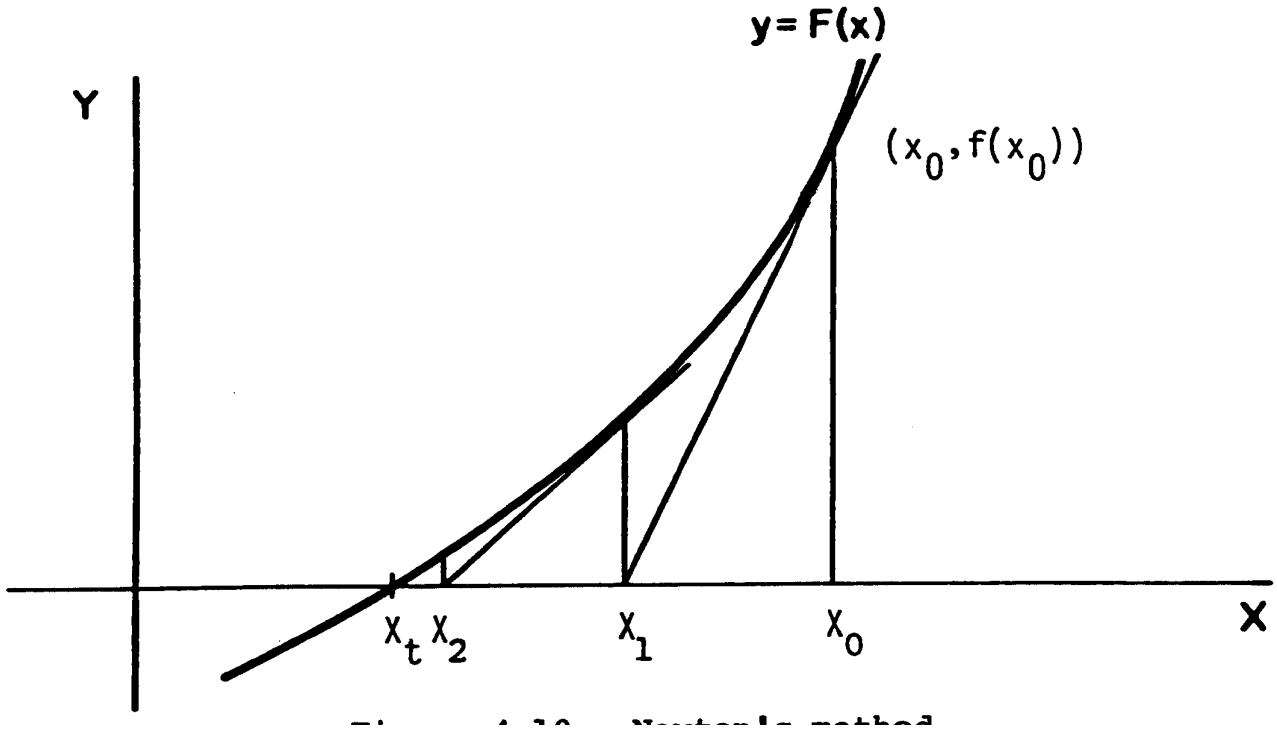


図 4-10. ニュートン法

- 除算アルゴリズムの誘導

関数 F(x)から派生した F'(x)の xt 点は、

$$F'(xt) = \lim_{x \to xt} \frac{F(x) - F(xt)}{x - xt}$$

もし、limit が存在すれば。もし、limit が存在しなければ、F(x)は t 点では微分可能ではない。

xt 付近のいかなる点 xi においても

$$F'(xt) \fallingdotseq \frac{F(xi) - F(xt)}{xi - xt}$$

ここで≒は近似等号。

この近似は、xi が xt へ近づくのを改善する。xi を近似解決に立たせ、xt を求めるべき本当の解に立たせる。F(x)が 0 になる x が厳密な解はである。

x=xt の時、上の式の F(xt)は 0 に置き換えられ、下の近似となる:

$$F'(xt) \fallingdotseq \frac{F(xi)}{xi - xt}$$

注意として、近似解へ適用する Xt−Xi は補正であり、解の近似(1)は(xt − xi)として与える:

$$xt - xi = \text{補正} \fallingdotseq \frac{-F(xi)}{F'(xi)}$$

つまり $\frac{-F(xi)}{F'(xi)}$ が、近似補正である。

もしこの量が近似に置き換えられれば、

xt≒ (xi + 近似補正) = X i+1 。

これは、次の式を与える:

$$Xi+1 = xi - \frac{F(xi)}{F'(xi)} \quad \text{方程式(1)}$$

ここで、Xi+1 が Xi よりも、求めるべき真の値 Xt に近い近似である。一般的に、正確な答えは一度では得られない、なぜならば補正項が一般的に正確でないからである。したがって、実際の使用に十分になるまで操作を繰り返す。

数 B の逆数を見つけるためにニュートン法の使用するに、単純に F(x) = (1/x − B)を使用する。

まず、F'(x)を得る:

ここで

$$F'(x) = \left(\frac{1}{x} - B\right)' = \left(\frac{-1}{x^2}\right)$$

ただし、どこでも、X1 ≠ 0 であるとする。

$$F'(x1) = -\frac{1}{x1^2}$$

x は、1/B の近傍で式(1)を適用して選ぶ。

$$x_2 = x_1 - \frac{\frac{1}{x_1} - B}{-\frac{1}{x_1^2}},$$

$$x_2 = x_1 + x_1^2\left(\frac{1}{x_1} - B\right),$$

$$x_2 = x_1 + x_1 - x_1^2 B,$$

$$x_2 = 2x_1 - x_1^2 B = x_1(2 - x_1 B)$$

システムでは、括弧内の x1 と量の乗算は浮動小数点乗算で行われる。2-x1B は逆数近似命令で実行される。X1 は 1/B の近傍の X であり、半精度逆数近似命令で作られる。

このニュートン法を使用した近似テクニックは、システムに実装されている。ハードウェアのテーブル参照が、手続きを開始するための、大まかな初期値,x0,を提供する。

X0(2 - X0B) 一次近似, I1

X1(2 - X1B) 二次近似, I2 } 逆数ユニットで行われる

X2(2 - X2B) 三次近似, I3

X3(2 - X3B) 四次近似 ソフトウェアで行われる

システムの逆数近似機能ユニットは 3 つの繰り返し: I1, I2, I3 を実行する。I1 は、8bit 精度で大まかな初期値 x0 を選ぶためのテーブル参照の後、見つかる。I2 は、2 回繰り返しで、16bit 精度である。I3 は、逆数近似機能ユニットの最終(3 回目)繰り返しの答えであり、その結果は 30bit 精度である。

4 回繰り返しは、補正項を計算するために、浮動小数点乗算機能ユニットのための特殊命令を使用する。この繰り返しは、逆数ユニットの答えの精度を全精度にまで高めるのに使用される。

S1/S2 の全精度の計算を行う除算アルゴリズムは、次の操作を必要とする:

S3 = 1/S2
　逆数近似機能ユニットで実行される

S4 = (2 − (S3 * S2))
　繰り返しモードの浮動小数点乗算機能ユニットで実行される

S5 = S4 * S3
　全精度を使用した浮動小数点乗算機能ユニットで実行される。ここで、S5 は 48bit 精度の 1/S2 と同一である

S6 = S5 * S1
　全精度、丸め付きを使用した浮動小数点乗算機能ユニットで実行される。
ステップ 1 の逆数近似は、30bit の補正である。操作 2 と 3 での、追加のニュートン繰り返し(4 回繰り返し)は 48bit まで精度を高める。この繰り返しの答えは、商の精度を 48bit にするために、全精度で丸めつきの乗算操作のオペランドとして適用される。結果の誤差が可能な範囲であれば、追加の繰り返しを試みてはいけない。

*****
注意

逆数繰り返しは、半精度逆数の生成ごとにm一度だけ使うように設計されている。

もし 4 回繰り返し(プログラムされた繰り返し)の結果の正確な逆数か、また、もし他の方法で生成された正確な逆数、他の繰り返しは、最終的に誤った逆数を結果としてしまう。
****

29bit 精度で十分なら。逆数近似命令は、半精度乗算と使用して、半精度の商を得るのに、たった 2 つの操作である。

S3 = 1/S2
　半精度で逆数近似機能ユニットで実行

S6 = S1 * S3
半精度で浮動小数点乗算機能ユニットで実行

半精度の結果の下位の 19bit は 0 として返され結果の 29bit の下位ビットには丸めが適用される。

除算計算の他の方法は次:

S3 = 1/S2
　逆数近似機能ユニットで実行

S5 = S1 * S3
　浮動小数点乗算機能ユニットで実行

S4 = (2 − (S3 * S2))
　浮動小数点乗算機能ユニットで実行

S6 = S4 * S5
　浮動小数点乗算機能ユニットで実行

スカラ商は、操作 2,3 が引き続く CP で発行されると、29CP で計算される。

この方法を用いると、全精度の逆数に至る補正は、分子が半精度の逆数を乗じられた後に適用される、乗じる前ではなく。

この手続きを使用するベクトル商は、操作 1,2 が一緒にチェイン化できるので、ベクトル時間は 4 よりも少なくてすむ。

これは、乗算操作の一つをオーバーラップする。(ベクトル中の各要素のベクトル時間は 1CP である)

*****
注意

別な方法で生成された逆数の仮数部は、全精度の逆数を生成するために述べた最初の方法から、最大 2×2^-48 異なる。この差は、他の方法がまったく丸めない間に、1 番目の方法が 2 回丸め(繰上げ)を行うことがあるから発生する。1 つの丸めは補正が生成される間に起こることがあり、最終的な商が生成される時に 2 回目の丸めが起こることがある。

したがって、もし逆数が比較されることがあるなら、逆数が生成される度に、同じ方法を使用しなければならない。Cray FORTRAN(CFT)は、一貫性のある方法を使用し、数の逆数は常に同じであるようにする。

****

例えば、2 つの 64 要素のベクトルの除算は、3*64CP にオーバーヘッドを加えただけである。(このケースでの機能ユニットのオーバーヘッドは 38CP である)

- 論理操作

スカラとベクトルの論理ユニットは 64bit 量の bit-by-bit の操作を実行する。操作は、論理的な積、差、和、併合の計算を提供している。

論理積は AND 機能である:

オペランド 1 1010
オペランド 2 1100
結果 1000

AND と似たような操作は次のような結果を生成する:

オペランド 1 1010
オペランド 2 1100
結果 0100

論理積(AND)操作は、1 で指定したビットが残されるというマスク操作に使用される。AND のこの変種は、0 で指定したビットが残される(オペランド 1 がマスクである)。

論理和は、包含 OR である:

オペランド 1 1010
オペランド 2 1100
結果 1110

論理差は、排他的 OR である:

オペランド 1 1010
オペランド 2 1100
結果 0110

論理一致は、排他的 NOR である:

オペランド 1 1010
オペランド 2 1100
結果 1001

論理併合(merge)は、2 つのオペランドとマスクを使用し、次のような結果を生成する

オペランド 1 10101010
オペランド 2 11001100
マスク 11110000
結果 10101100

マスク・ビットが 1 であるオペランド 1 のビットはパスする。マスク・ビットが 0 であるオペランド 2 のビットはパスする。

pp.4-36