お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

お客様各位

資料中の「日立製作所」、「日立XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って三菱電機株式会社及び株式会社日立製作所のマイコン、ロジック、アナログ、ディスクリート半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。従いまして、本資料中には「日立製作所」、「株式会社日立製作所」、「日立半導体」、「日立XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

ルネサステクノロジ ホームページ (http://www.renesas.com)

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

RENESAS

ご注意

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ (http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

Application Note

RENESAS

SH-DSP ソフトウェア編

アプリケーションノート

SuperH RISC engine

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　1999.08

ご注意

1. 本書に記載の製品及び技術のうち「外国為替及び外国貿易法」に基づき安全保障貿易管理関連貨物・技術に該当するものを輸出する場合，または国外に持ち出す場合は日本国政府の許可が必要です。
2. 本書に記載された情報の使用に際して，弊社もしくは第三者の特許権，著作権，商標権，その他の知的所有権等の権利に対する保証または実施権の許諾を行うものではありません。また本書に記載された情報を使用した事により第三者の知的所有権等の権利に関わる問題が生じた場合，弊社はその責を負いませんので予めご了承ください。
3. 製品及び製品仕様は予告無く変更する場合がありますので，最終的な設計，ご購入，ご使用に際しましては，事前に最新の製品規格または仕様書をお求めになりご確認ください。
4. 弊社は品質・信頼性の向上に努めておりますが，宇宙，航空，原子力，燃焼制御，運輸，交通，各種安全装置，ライフサポート関連の医療機器等のように，特別な品質・信頼性が要求され，その故障や誤動作が直接人命を脅かしたり，人体に危害を及ぼす恐れのある用途にご使用をお考えのお客様は，事前に弊社営業担当迄ご相談をお願い致します。
5. 設計に際しては，特に最大定格，動作電源電圧範囲，放熱特性，実装条件及びその他諸条件につきましては，弊社保証範囲内でご使用いただきますようお願い致します。
   保証値を越えてご使用された場合の故障及び事故につきましては，弊社はその責を負いません。
   また保証値内のご使用であっても半導体製品について通常予測される故障発生率，故障モードをご考慮の上，弊社製品の動作が原因でご使用機器が人身事故，火災事故，その他の拡大損害を生じないようにフェールセーフ等のシステム上の対策を講じて頂きますようお願い致します。
6. 本製品は耐放射線設計をしておりません。
7. 本書の一部または全部を弊社の文書による承認なしに転載または複製することを堅くお断り致します。
8. 本書をはじめ弊社半導体についてのお問い合わせ，ご相談は弊社営業担当迄お願い致します。

はじめに

SH-DSPは、SuperH RISC engineファミリの中のCPUコアになります。SH-2 CPUをベースに、信号処理性能を重視したDSPユニットを融合した32ビットRISCマイクロコントローラです。

このアプリケーションノートは、SH-DSPの特長を生かしたソフトウェアの応用例やハードウェアの使用方法をのべております。DSP機能を使用したソフトウェアを設計する際、ご参考としてお役立てていただけるようまとめたものになります。

なお、本アプリケーションノートに掲載されているプログラム等の動作は確認しておりますが、実際にご使用になる場合は、必ず動作確認の上、ご使用くださるようお願いいたします。

ハードウェアについては、各製品別のハードウェアマニュアルをご参照ください。

開発環境システムの詳細については、当社営業までお問い合わせください。

SH-DSPソフトウェア応用例

本アプリケーションノートでは、SH-DSPの特長を生かしたソフトウェア応用例を紹介します。

図1に本アプリケーションノート用に作成したソースプログラムのフォーマットを示します。メインプログラムをXRAMへ転送し、XRAM上でプログラムを実行します。本フォーマットはSH7612用ですが、他のSH-DSPで用いる場合は、以下の項目の変更及び注意が必要です。

・XRAM先頭番地の設定 ……①
・ベクタ及びスタックポインタ(YRAMの最終番地+1バイト)の設定 ……②
・各種SH-DSPマイコンで転用可能……③
・メインプログラムで使用するデータをX/YRAM上に確保しているので、
使用するマイコンによりX/YRAMのアドレス設定を変更 ……④

```
;*******************************************************************************************
;*                                  ｼﾝﾎﾞﾙ定義
;*******************************************************************************************
;
;           [           XRAMｱﾄﾞﾚｽ(SH7612)                ]
XRAM_TOP    .EQU        H'1000E000 ……②
;*******************************************************************************************
;*                                  ﾌﾟﾛｸﾞﾗﾑ転送ﾙｰﾁﾝ
;*******************************************************************************************
;
            .SECTION    VECT,CODE,LOCATE=H'0
;
            .DATA.L                 _PRES               ;_PRES  ……①
            .DATA.L                 H'10020000          ;SP
            .SECTION    ROM,CODE,LOCATE=H'1000

_PRES:      MOV.L       #XRAM_TOP,R1
            MOV.L       #MAIN,R10
            MOV.L       #MAIN_E,R11
PRG_MOVE:
            MOV.W       @R10+,R0
            MOV.W       R0,@R1
            ADD         #2,R1
            CMP/GE      R11,R10
            BF          PRG_MOVE
            MOV.L       #XRAM_TOP,R0
            JMP         @R0                             ;転送先のﾌﾟﾛｸﾞﾗﾑ先頭番地へ分岐
            NOP
```

メｲﾝﾌﾟﾛｸﾞﾗﾑ……③

ﾃﾞｰﾀ……④

```
            .END
```

図1　ソースプログラムフォーマット

目次

1. Cソースプログラムからの関数(DSPライブラリ)呼び出し例

1.1 関数(DSPライブラリ)を使用したCソースプログラムのコーディング

Cソースプログラム上でDSPライブラリ(shdsplib.lib)内の関数"Mean"を呼びだす際の、プログラム例"dsplbr.c"を以下に示します。

```
/*
        <<SH-DSPｱﾌﾟﾘｹｰｼｮﾝﾉｰﾄ>>
                -----DSPﾗｲﾌﾞﾗﾘ使用例------
                "dsplbr.c"
*/
#include "ensigdsp.h"                                   /*  Mean関数を定義  */ ーーーーーーーーーー①
#define N 6                                             /*  入力ﾃﾞｰﾀ数  */

        short dat[6]={45,61,516,3000,-974,10214};       /*  入力ﾃﾞｰﾀ  */

        #pragma section X                               /*  XRAM上のｱﾄﾞﾚｽ  */ ーーーーーーーーーー②
                static short        datx[N];
        #pragma section Y                               /*  YRAM上のｱﾄﾞﾚｽ  */ ーーーーーーーーーー③
                static short        daty[N];
        #pragma section ANS                             /*  平均値を格納するｱﾄﾞﾚｽ  */
                static short        answer;
        #pragma section


main()
{
        short           i,output[1];                    /*  変数iとMean関数演算結果格納用output  */
        int             src_x;                          /*  入力ﾃﾞｰﾀ格納領域指定用引数  */

        for(i=0;i<N;i++)
                {
                datx[i] = dat[i];                       /*  入力ﾃﾞｰﾀをXRAMへｺﾋﾟｰ  */
                daty[i] = dat[i];                       /*  入力ﾃﾞｰﾀをYRAMへｺﾋﾟｰ  */
                }

        /*      select XRAM     */
    ※1
        src_x = 1;                                      /*  Mean関数の演算にXRAM領域を使用  */ ーーー④
                Mean(output,datx,N,src_x);              /*  Mean関数に引数を渡し、平均値の計算  */

                answer = output[0];                     /*  Mean関数演算結果をanswerｱﾄﾞﾚｽに格納  */

        while(1);                                       /*  処理終了  */

}                                                                   ※1  詳細説明は、1.3関数の実行経過参照
```

① shdsplib.libのライブラリ内の関数の型をensigdsp.hヘッダファイルで定義しています。

② DSPユニットでXバスデータ転送を効果的に用いるためにdatX[N]はXRAM上に配置しておく必要があるので、section XはXRAM上のアドレスをリンク時に設定します。(1.2リンク時の指定を参照)

③ DSPユニットでYバスデータ転送を効果的に用いるためにdatY[N]はYRAM上に配置しておく必要があるので、section YはYRAM上のアドレスをリンク時に設定します。(1.2リンク時の指定を参照)

④ src_x =1の場合、Mean関数の演算にXRAM領域を使用します。src_x =0の場合は、YRAM領域を使用します。

1.2 リンク時の指定

DSPライブラリを使用した場合、もっとも注意しなければならないのが、セクションの設定です。1.1で示したdsplbr.cのプログラムでセクションX,Yがありますが、これは、XRAM、YRAM上のアドレスを設定しなければ、関数内で正常に演算を実行することができません。そこで、サブコマンドファイル内で指定します。

1.2.1 リンク時のサブコマンドファイル"prglnk1.sub"

```
INPUT       vect,dsplbr
START       BX(1000ff00),BANS(1000fff0),BY(1001e000) ---- ①
LIBRARY     shdsplib.lib ----------------- ②
PRINT       dsplbr.map
OUTPUT      dsplbr.abs
FORM        A
DEBUG
EXIT
```

① BX(1000ff00)でdsplbr.cの#pragma section XのセクションXをH'1000FF00番地に割り付けます。
BY(1001e000)でdsplbr.cの#pragma section YのセクションYをH'1001E000番地に割り付けます。

② Mean関数を含むshdsplib.libを編集対象のライブラリとして指定します。

1.2.2 アブソリュートファイル作成用バッチファイル"ini.bat"

```
asmsh  vect.src  -cpu=shdsp  -debug  -lis
shc  dsplbr.c -cpu=sh2  -lis  -debug  -include=ensigdsp.h
lnk  -subcommand=prglnk1.sub
```

1.2.3 DSPライブラリ使用プログラム"dsplbr.c"のベクタテーブル"vect.src"

```
;*******************************************************
;*         <<SH-DSPｱﾌﾟﾘｹｰｼｮﾝﾉｰﾄ>>
;*                 -----DSPﾗｲﾌﾞﾗﾘ使用例------
;*
;*                          "vect.src"
;******************************************************

                .import         _main

                .section   vect,data,locate=h'0
                .data.l         _main
                .data.l         h'10020000
                .end
```

1.3　関数の実行経過

1.1に示すdsplbr.cプログラムの抜粋及び、プログラム中で使用している関数のアセンブラ展開を以下に示します。

```
        ⋮
src_x = 1;

Mean(output,datx,N,src_x);

answer = output[0];
        ⋮
```

関数のアセンブラ展開

Address	Label	Assembler	
1001e2fc	_Mean	CMP/PZ	R7
1001e2fe		BF	@1001E322:8
1001e300		MOV	#H'01,R1
1001e302		CMP/GT	R1,R7
⋮			
1001e486		NEG	R2,R2
1001e488		MOV.W	R2,@R4
1001e48a		RTS	

1.1には入力データをH'1000FF00番地から配置してあります。RAM上のデータは0クリアされていると仮定します。関数実行後もデータはこのままです。

表1.1　メモリマップ

XRAM_Memory				
H'1000FF00	002D	003D	0204	0BB8
H'1000FF08	FC32	27E6	0000	0000

表1.2　関数実行経過

dsplbr.cプログラム抜粋	レジスタ内容
Mean(output,datx,N,src_x);	実行前　R4=H'1001FFFC,R5=H'1000FF00,R6=6,R7=1 実行後　R4=H'1001FFFC,R5=H'1000FF0C,R6=6,R7=H'10000

関数の引数は宣言順にR4～R7に割り付けられますので、output=H'1001FFFC,datx=H'1000FF00, N=6,src_x=1が関数に渡されます。演算結果は、@R4に保持されます。

⇩

表1.3　Cソースプログラム実行経過(メモリマップ内の経過)

dsplbr.cプログラム抜粋	YRAM　Memory				
answer = output[0];	実行前　H'1001FF00	0000	0000	0000	0000
	実行後　H'1001FF00	0860	0000	0000	0000

Cソースプログラム上で@R4から関数演算結果をanswer(H'1001FF0)に格納します。

表1.4　Mean関数演算結果

入力値(10進)	入力値(16進)	理論値(10進)	理論値(16進)	出力値(16進)
45	H'2D	2143.666667	H'860 (2144を10進数値として算出)	H'860
61	H'3D			
516	H'204			
3000	H'BB8			
−974	H'FC32			
10214	H'27E6			

2. X/Yバスデータアクセス

2.1 Xメモリリード

概要

XRAM_ADDアドレス(H'1000FF00)、XRAM_ADD+2アドレス(H'1000FF02)のデータをそれぞれX0、X1レジスタへデータ転送する。

説明

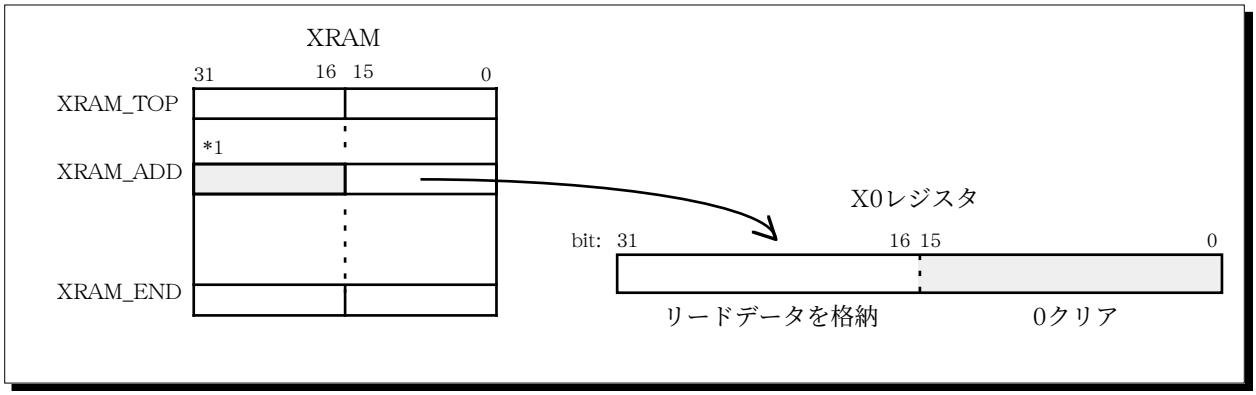
表2.1にXメモリリード命令の種類とオペランドに使えるレジスタを示します。**表2.1**の命令を用いることでXメモリからデータをリードしてくることができます。

Xメモリからリードしてくる場合、転送データ長が16ビットの為、命令で指定したX0、X1レジスタのどちらかの上位ワードへデータが格納されます。この際、X0、X1レジスタの下位ワードは0にクリアされます。フローチャート中の処理①②を以下に示します。

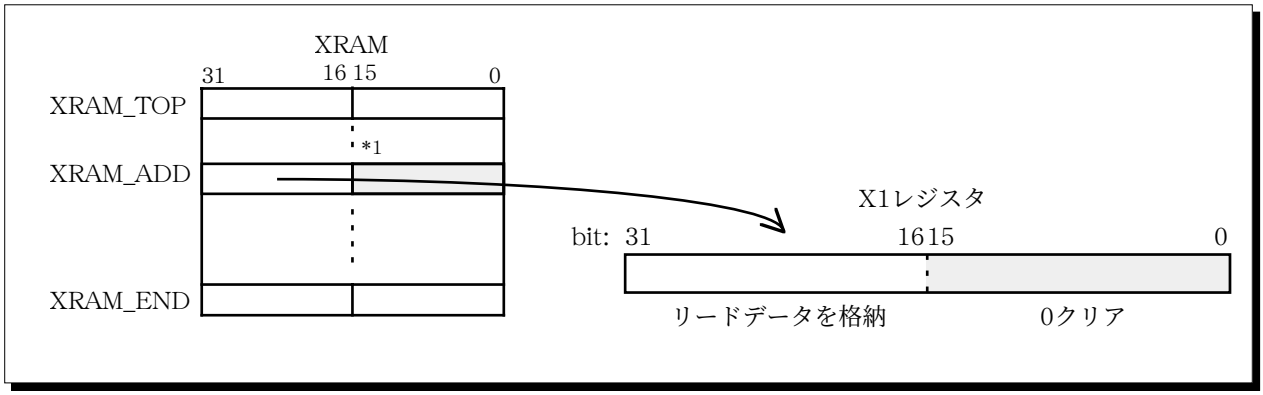
表2.1 Xメモリリード命令の種類

Xﾒﾓﾘﾘｰﾄﾞ命令	ｿｰｽﾚｼﾞｽﾀ(Ax)	ﾃﾞｽﾃｨﾈｰｼｮﾝﾚｼﾞｽﾀ(Dx)	ｲﾝﾃﾞｯｸｽﾚｼﾞｽﾀ(Ix)
MOVX.W @Ax,Dx	R4,R5	X0,X1	R8
MOVX.W @Ax+,Dx			
MOVX.W @Ax+Ix,Dx			

処理①

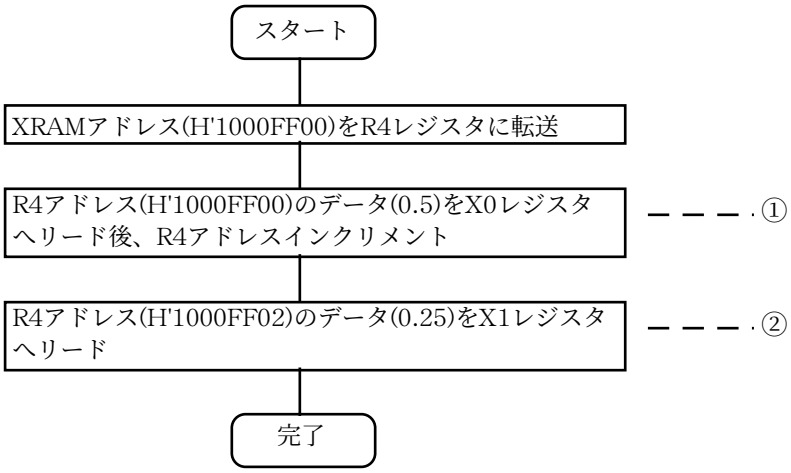


処理②



*1 ⋮⋮⋮：無視

フローチャート



メインプログラム

```
;*****************************************************************************************
;*                                  Xﾒﾓﾘﾘｰﾄﾞ
;*****************************************************************************************
MAIN:     MOV.L       #XRAM_ADD,R4          ;XRAM_ADDｱﾄﾞﾚｽ -> R4ﾚｼﾞｽﾀ
          MOVX.W      @R4+,X0               ;(H'1000FF00) -> X0
          MOVX.W      @R4,X1                ;(H'1000FF02) -> X1
EXIT:     BRA         EXIT
          NOP
MAIN_E:   NOP
```

データ

```
;*****************************************************************
;*                          ﾃﾞｰﾀ
;*****************************************************************
                    .SECTION XRAM,DATA,LOCATE=H'1000FF00
XRAM_ADD:           .XDATA.W        0.5,0.25
```

概要

XRAM_ADD1アドレス(H'1000FF00)、XRAM_ADD1+2アドレス(H'1000FF02)のデータをそれぞれ、XRAM_ADD2アドレス、XRAM_ADD2+2アドレスへ転送する。

説明

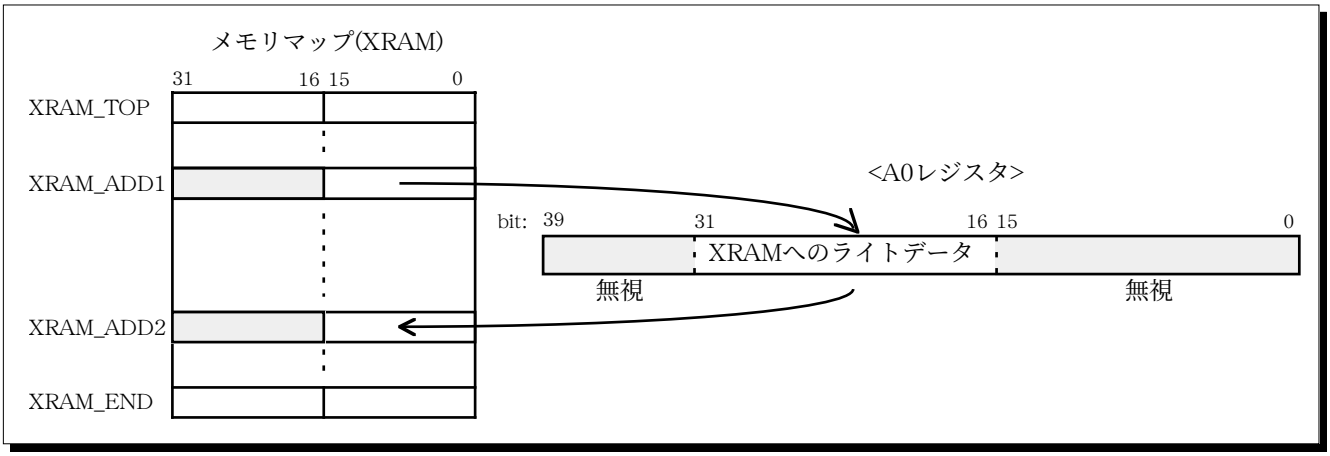
表2.2にXメモリライト命令の種類とオペランドに使えるレジスタを示します。**表2.2**の命令を用いることでXメモリからデータをリードしてくることができます。

Xメモリへライトする場合、転送データ長が16ビットの為、命令で指定したA0、A1レジスタのどちらかの上位ワードデータがXメモリへ格納されます。この際、A0、A1レジスタのガードビット及び、下位ワードは無視されます。Xメモリライト命令では、ソースレジスタにA0、A1レジスタしか使用できない(**表2.2 Xメモリライト命令の種類**参照)ので、A0、A1レジスタへのデータ転送にはA0、A1レジスタをデスティネーションオペランドに持つシングルデータ転送を用いる。フローチャート中の処理①②を以下に示します。

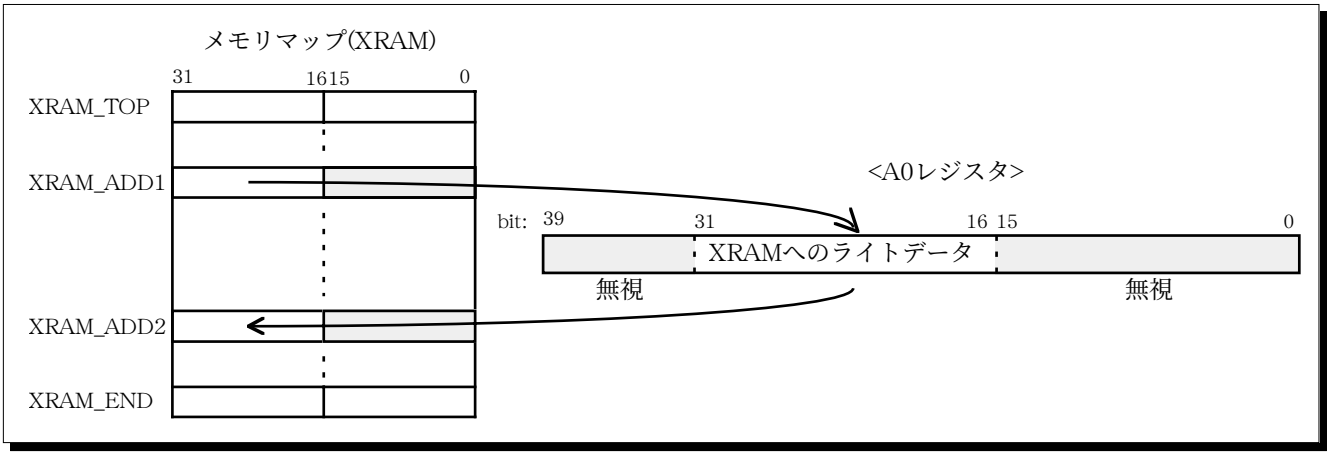
表2.2　Xメモリライト命令の種類

Xﾒﾓﾘﾗｲﾄ命令	ｿｰｽﾚｼﾞｽﾀ(Da)	ﾃﾞｽﾃｨﾈｰｼｮﾝﾚｼﾞｽﾀ(Ax)	ｲﾝﾃﾞｯｸｽﾚｼﾞｽﾀ(Ix)
MOVX.W Da,@Ax	A0,A1	R4,R5	R8
MOVX.W Da,@Ax+			
MOVX.W Da,@Ax+Ix			

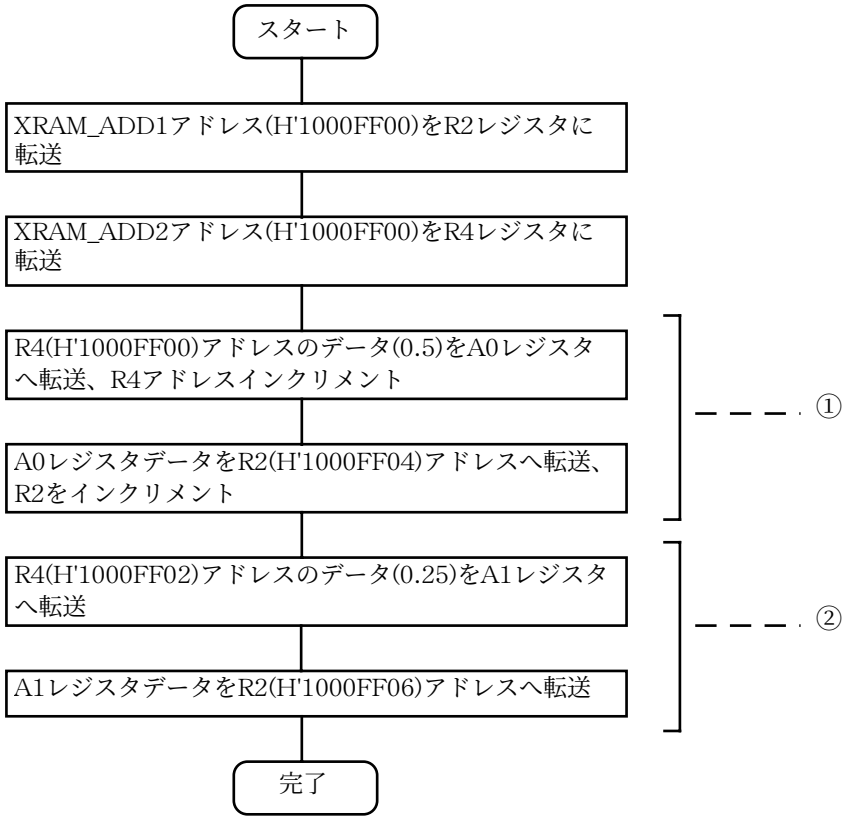
処理①



処理②



フローチャート


スタート
XRAM_ADD1アドレス(H'1000FF00)をR2レジスタに転送
XRAM_ADD2アドレス(H'1000FF00)をR4レジスタに転送
R4(H'1000FF00)アドレスのデータ(0.5)をA0レジスタへ転送、R4アドレスインクリメント
A0レジスタデータをR2(H'1000FF04)アドレスへ転送、R2をインクリメント
①
R4(H'1000FF02)アドレスのデータ(0.25)をA1レジスタへ転送
A1レジスタデータをR2(H'1000FF06)アドレスへ転送
②
完了

メインプログラム

```
;*****************************************************************************
;*                              Xﾒﾓﾘﾗｲﾄ
;*****************************************************************************
MAIN:   MOV.L       #XRAM_ADD1,R2           ;XRAM_ADD1 -> R2ﾚｼﾞｽﾀ
        MOV.L       #XRAM_ADD2,R4           ;XRAM_ADD2 -> R4ﾚｼﾞｽﾀ
        MOVS.W      @R2+,A0                 ;(H'1000FF00) -> A0ﾚｼﾞｽﾀ
        MOVX.W      A0,@R4+                 ;A0ﾚｼﾞｽﾀﾃﾞｰﾀ -> XRAM_ADD2
        MOVS.W      @R2,A1                  ;(H'1000FF00) -> A1ﾚｼﾞｽﾀ
        MOVX.W      A1,@R4                  ;A1ﾚｼﾞｽﾀﾃﾞｰﾀ -> XRAM_ADD2+2
EXIT:   BRA         EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*****************************************************************
;*                          ﾃﾞｰﾀ
;*****************************************************************
                .SECTION XRAM,DATA,LOCATE=H'1000FF00
XRAM_ADD1:      .XDATA.W        0.5,0.25
XRAM_ADD2:      .RES.W          2
```

概要

YRAM_ADDアドレス(H'1001FF00)、YRAM_ADD+2アドレス(H'1001FF02)のデータをそれぞれY0、Y1レジスタへデータ転送する。

説明

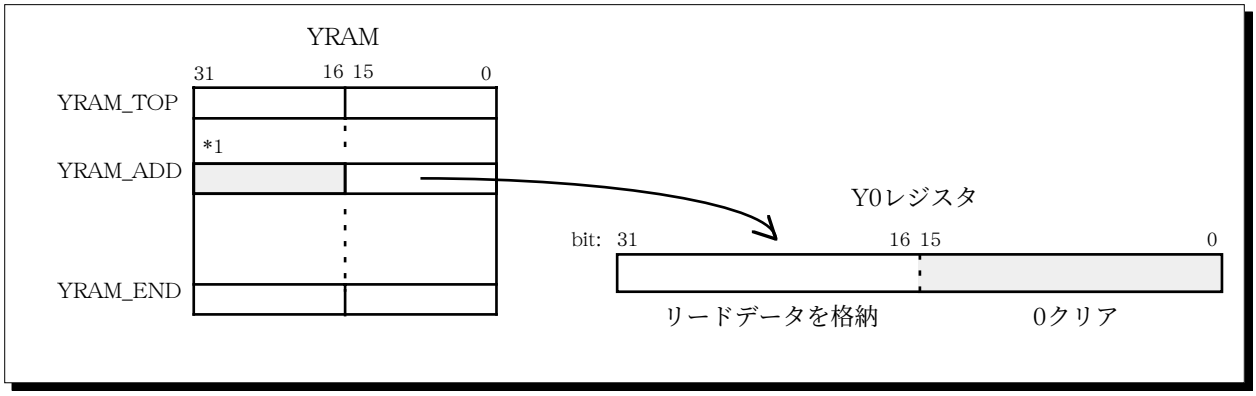
表2.3にYメモリリード命令の種類とオペランドに使えるレジスタを示します。**表2.3**の命令を用いることでYメモリからデータをリードしてくることができます。

Yメモリからリードしてくる場合、転送データ長が16ビットの為、命令で指定したY0、Y1レジスタのどちらかの上位ワードへデータが格納されます。この際、Y0、Y1レジスタの下位ワードは0にクリアされます。フローチャート中の処理①②を以下に示します。

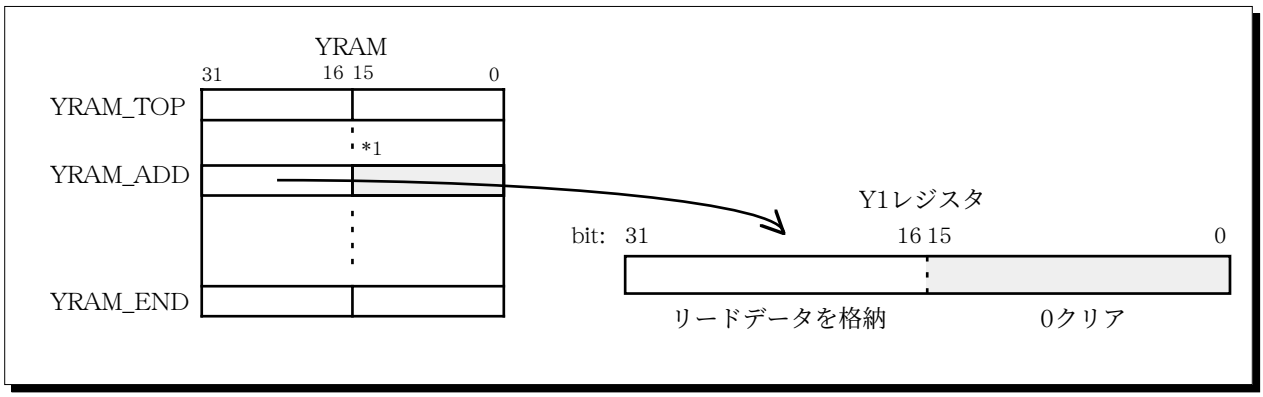
表2.3 Yメモリリード命令の種類

Yﾒﾓﾘﾘｰﾄﾞ命令		ｿｰｽﾚｼﾞｽﾀ(Ay)	ﾃﾞｽﾃｨﾈｰｼｮﾝﾚｼﾞｽﾀ(Dy)	ｲﾝﾃﾞｯｸｽﾚｼﾞｽﾀ(Iy)
MOVY.W	@Ay,Dy	R6,R7	Y0,Y1	R9
MOVY.W	@Ay+,Dy			
MOVY.W	@Ay+Iy,Dy			

処理①

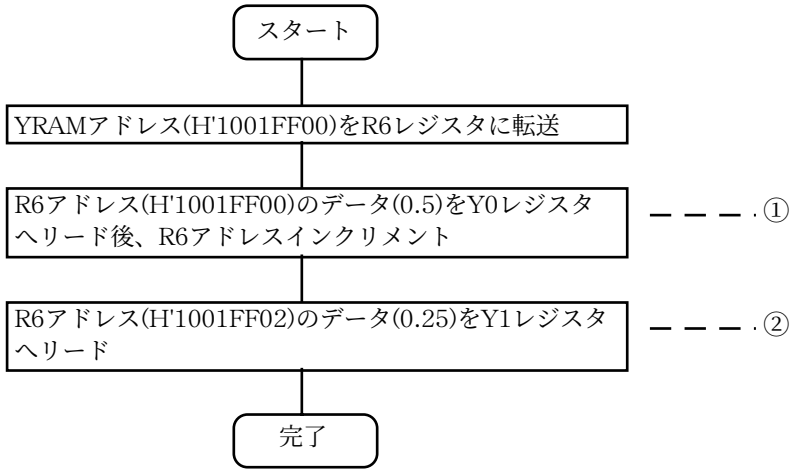


処理②



*1 ::::: : 無視

フローチャート



メインプログラム

```
;*****************************************************************************************
;*                                  Yﾒﾓﾘﾘｰﾄﾞ
;*****************************************************************************************
MAIN:     MOV.L       #YRAM_ADD,R6          ;YRAM_ADDｱﾄﾞﾚｽ -> R6ﾚｼﾞｽﾀ
          MOVX.W      @R6+,Y0               ;(H'1001FF00) -> Y0
          MOVX.W      @R6,Y1                ;(H'1001FF02) -> Y1
EXIT:     BRA         EXIT
          NOP
MAIN_E:   NOP
```

メインプログラム

```
;*****************************************************************
;*                          ﾃﾞｰﾀ
;*****************************************************************
                    .SECTION YRAM,DATA,LOCATE=H'1001FF00
YRAM_ADD:           .XDATA.W            0.5,0.25
```

概要

YRAM_ADD1アドレス(H'1001FF00)、YRAM_ADD1+2アドレス(H'1001FF02)のデータをそれぞれ、XRAM_ADD2アドレス、XRAM_ADD2+2アドレスへ転送する。

説明

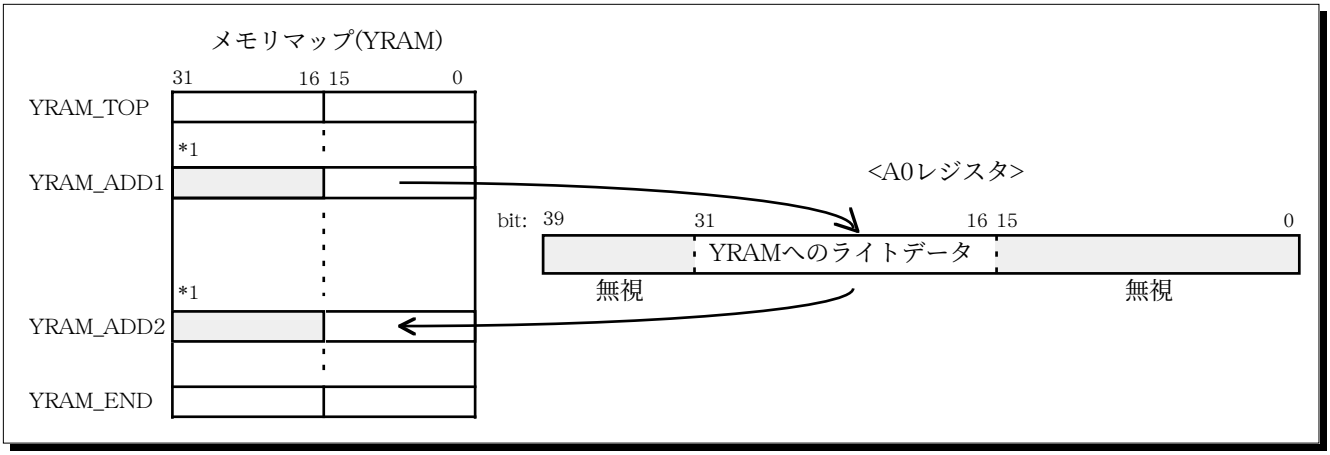
表2.4にYメモリライト命令の種類とオペランドに使えるレジスタを示します。**表2.4**の命令を用いることでYメモリへデータをライトすることができます。

Yメモリへライトする場合、転送データ長が16ビットの為、命令で指定したA0、A1レジスタのどちらかの上位ワードデータがYメモリへ格納されます。この際、A0、A1レジスタのガードビット及び、下位ワードは無視されます。Yメモリライト命令では、ソースレジスタにA0、A1レジスタしか使用できない(**表2.4 Yメモリライト命令の種類**参照)ので、A0、A1レジスタへのデータ転送にはA0、A1レジスタをデスティネーションオペランドに持つシングルデータ転送を用いる。フローチャート中の処理①②を以下に示します。

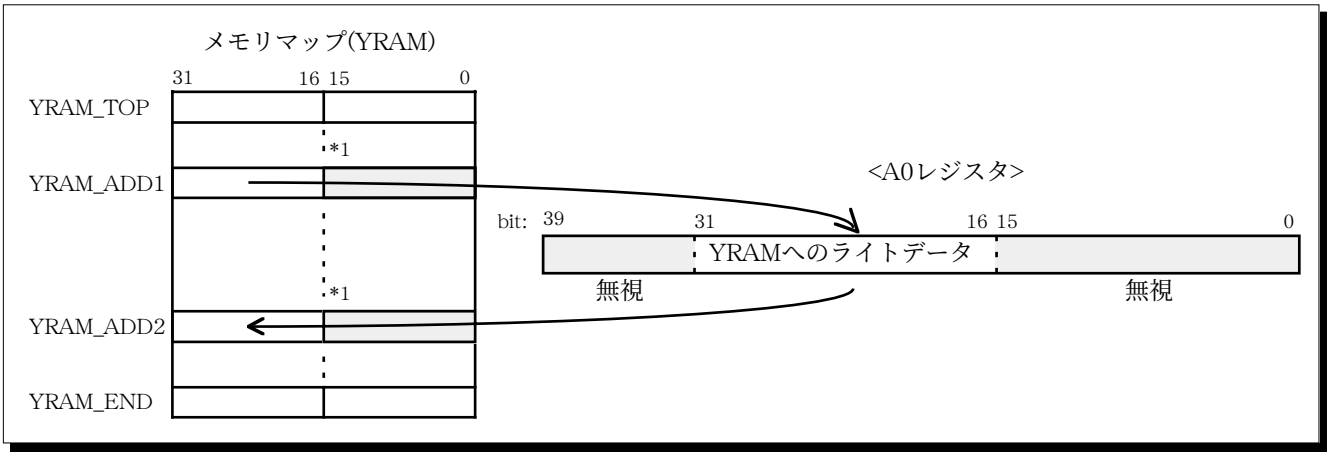
表2.4　Yメモリライト命令の種類

Yﾒﾓﾘﾗｲﾄ命令		ｿｰｽﾚｼﾞｽﾀ(Da)	ﾃﾞｽﾃｨﾈｰｼｮﾝﾚｼﾞｽﾀ(Ax)	ｲﾝﾃﾞｯｸｽﾚｼﾞｽﾀ(Ix)
MOVY.W	Da,@Ax	A0,A1	R6,R7	R9
MOVY.W	Da,@Ax+			
MOVY.W	Da,@Ax+Ix			

処理①

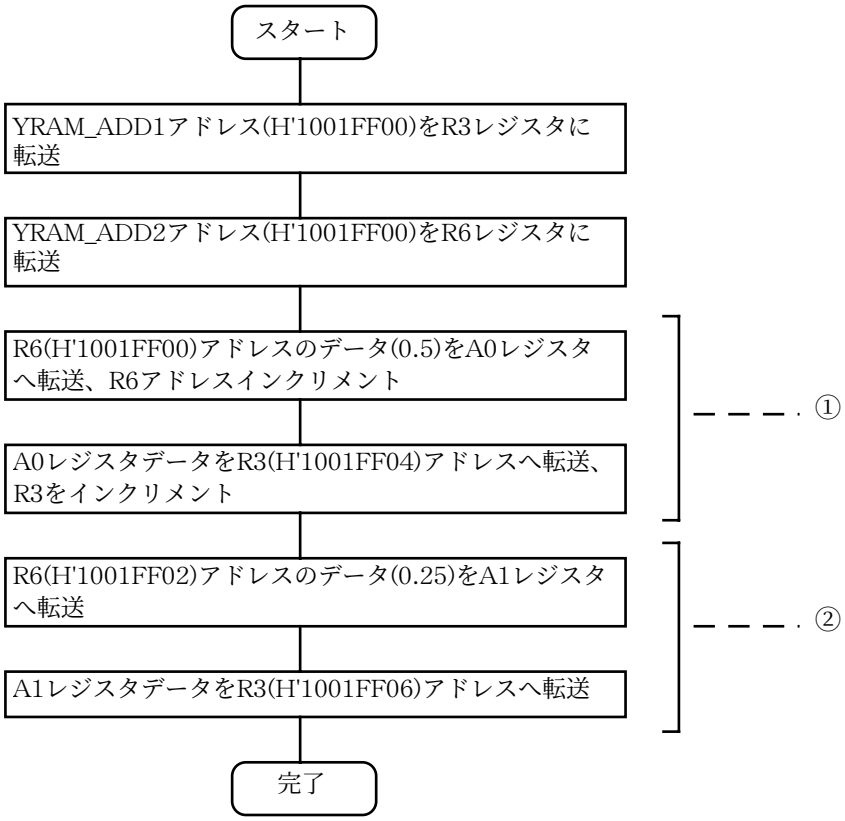


処理②



*1　：無視

フローチャート


スタート
YRAM_ADD1アドレス(H'1001FF00)をR3レジスタに転送
YRAM_ADD2アドレス(H'1001FF00)をR6レジスタに転送
R6(H'1001FF00)アドレスのデータ(0.5)をA0レジスタへ転送、R6アドレスインクリメント
A0レジスタデータをR3(H'1001FF04)アドレスへ転送、R3をインクリメント
①
R6(H'1001FF02)アドレスのデータ(0.25)をA1レジスタへ転送
A1レジスタデータをR3(H'1001FF06)アドレスへ転送
②
完了

メインプログラム

```
;*****************************************************************************
;*                                Yﾒﾓﾘﾗｲﾄ
;*****************************************************************************
MAIN:   MOV.L       #YRAM_ADD1,R3           ;YRAM_ADD1 -> R3ﾚｼﾞｽﾀ
        MOV.L       #YRAM_ADD2,R6           ;YRAM_ADD2 -> R6ﾚｼﾞｽﾀ
        MOVS.W      @R3+,A0                 ;(H'1001FF00) -> A0ﾚｼﾞｽﾀ
        MOVX.W      A0,@R6+                 ;A0ﾚｼﾞｽﾀﾃﾞｰﾀ -> YRAM_ADD2
        MOVS.W      @R3,A1                  ;(H'1001FF00) -> A1ﾚｼﾞｽﾀ
        MOVX.W      A1,@R6                  ;A1ﾚｼﾞｽﾀﾃﾞｰﾀ -> YRAM_ADD2+2
EXIT:   BRA         EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*****************************************************************
;*                          ﾃﾞｰﾀ
;*****************************************************************
                .SECTION YRAM,DATA,LOCATE=H'1001FF00
YRAM_ADD1:      .XDATA.W        0.5,0.25
YRAM_ADD2:      .RES.W          2
```

3. 16ビット固定小数点乗算

概要

XRAM_ADDアドレス(H'1000F000)の16ビットデータとYRAM_ADDアドレス(1001F000)の16ビットデータを乗算します。乗算結果をANSアドレス(H'1001F002)に格納します。

説明

1. データ転送

XRAM_ADDアドレス(H'1000F000)とYRAM_ADDアドレス(1001F000)からのデータ転送を**2.X/Yバスデータアクセス**で紹介したXバスデータ転送、Yバスデータ転送を用います。フローチャート中の処理①では、XRAMとYRAMからデータをリードしてくる作業を同時に実行していまが、XバスとYバスは独立しているのでデータの競合は起こりません。書式は以下の通りです。

［Xバスデータ転送］［Yバスデータ転送]の順序で1ステップ内に記述すれば、命令の組み合わせは、[Xメモリリード］［Yメモリライト]や[Xメモリライト］［Yメモリリード]のように自由です。

書式： MOVX.W @R5,X1 MOVY.W @R7,Y1

2. 固定小数点乗算

フローチャート中の処理②の固定小数点の乗算にはPMULS命令を用います。書式は以下の通りです。固定小数点乗算の流れは、**図3.1**の固定小数点乗算の流れに示したように、ソース1,2には上位ワードのみが有効なデータとなります。例えば、H'12345678のロングワードデータがある場合、実際に乗算に使用されるデータは、H'1234となります。

書式： PMULS Se,Sf,Dg

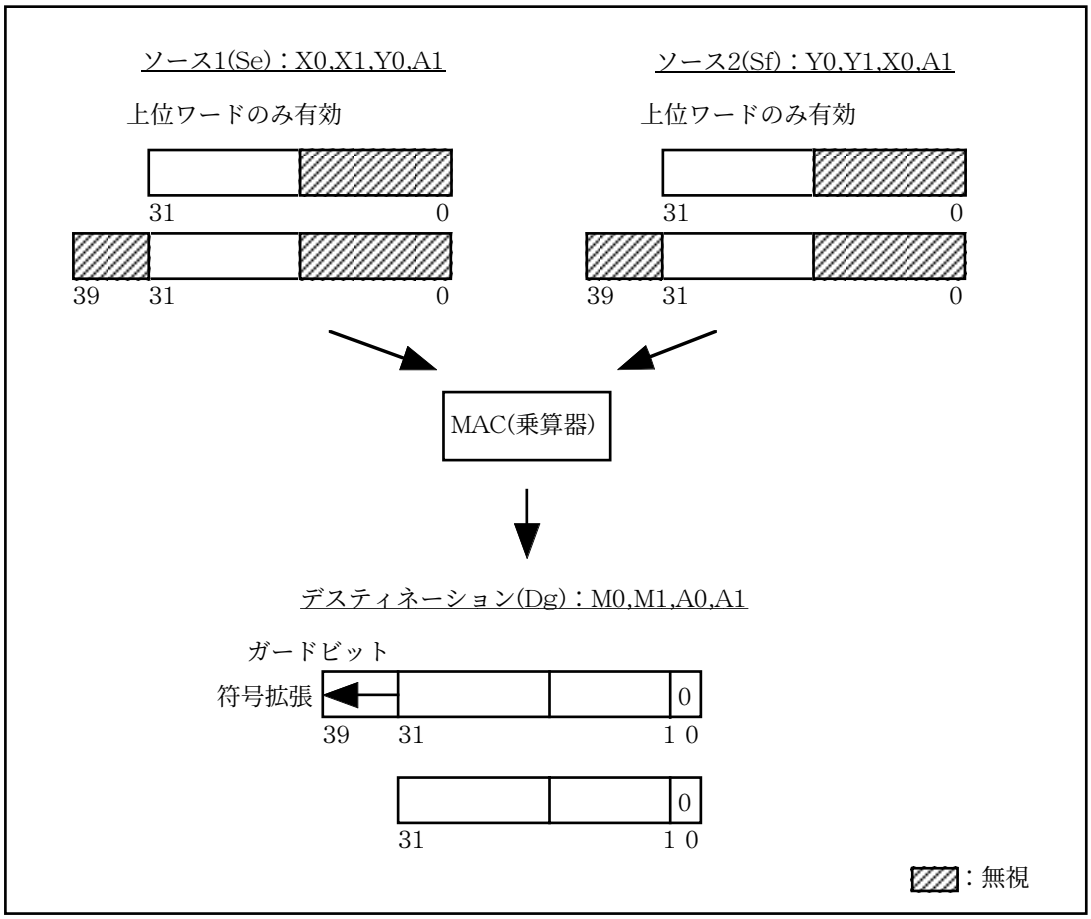


図3.1 固定小数点乗算の流れ

3. オーバフロー

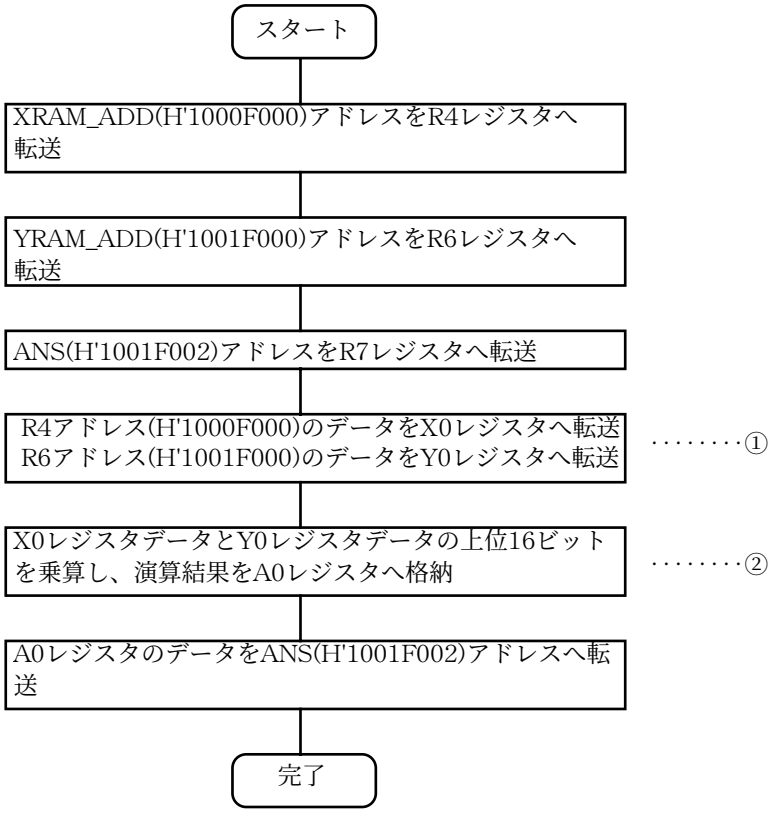
固定小数点乗算では、H'8000(−1.0)×H'8000(−1.0)の場合だけ、オーバーフローが発生し、演算結果はH'8000(−1.0)となります。これはディスティネーションレジスタにガードビット付きのA0、A1レジスタ以外を使用した場合に限ります。ディスティネーションレジスタにA0、A1を指定した場合の演算結果は、H'0080000000(1.0)となり正しい値が得られます。その他、固定小数点乗算実行例を**表3.1**に示します。

本メインプログラムでは、ディスティネーションレジスタにA0レジスタを指定している為、問題ありません。

表3.1　固定小数点乗算実行例

演算例	演算結果の状態	ディスティネーションレジスタ	演算結果
H'4000(0.5)× H'2000(0.25)	正	M0,M1	H'1000 0000(0.125)
		A0,A1	H'00 1000 0000(0.125)
H'0800(0.0625)× H'FC00(−0.03125)	負	M0,M1	H'FFC00 0000(-1.95×10^{-3})
		A0,A1	H'FF FFC00 0000(-1.95×10^{-3})
H'8000(−1.0)× H'8000(−1.0)	ｵｰﾊﾞﾌﾛｰ	M0,M1	H'8000 0000(−0.1)
		A0,A1	H'00 8000 0000(1.0)

フローチャート


スタート
XRAM_ADD(H'1000F000)アドレスをR4レジスタへ転送
YRAM_ADD(H'1001F000)アドレスをR6レジスタへ転送
ANS(H'1001F002)アドレスをR7レジスタへ転送
R4アドレス(H'1000F000)のデータをX0レジスタへ転送
R6アドレス(H'1001F000)のデータをY0レジスタへ転送
・・・・・・・・①
X0レジスタデータとY0レジスタデータの上位16ビットを乗算し、演算結果をA0レジスタへ格納
・・・・・・・・②
A0レジスタのデータをANS(H'1001F002)アドレスへ転送
完了

メインプログラム

```
;*****************************************************************************************************************************************************
;*                                  16ﾋﾞｯﾄ固定小数点乗算ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
MAIN:   MOV.L           #0,R4                                                                                   ;R4ﾚｼﾞｽﾀｸﾘｱ
        MOV.L           #0,R6                                                                                   ;R6ﾚｼﾞｽﾀｸﾘｱ
        MOV.L           #XRAM_ADD,R4                                                                            ;XRAMｱﾄﾞﾚｽ -> R4ﾚｼﾞｽﾀ
        MOV.L           #YRAM_ADD,R6                                                                            ;YRAMｱﾄﾞﾚｽ -> R6ﾚｼﾞｽﾀ
        MOV.L           #ANS,R7                                                                                 ;ansｱﾄﾞﾚｽ -> R7ﾚｼﾞｽﾀ
                                                MOVX.W  @R4,X0  MOVY.W  @R6,Y0                                  ;X/YRAMｱﾄﾞﾚｽﾃﾞｰﾀ -> X0/Y0ﾚｼﾞｽﾀ
                        PMULS   X0,Y0,A0                                                                        ;16ﾋﾞｯﾄ固定小数点演算
                                                                MOVY.W  A0,@R7                                  ;乗算結果格納
EXIT:   BRA             EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*****************************************************************
;*                      ﾃﾞｰﾀ
;*****************************************************************
                .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:       .XDATA.W        0.0625

                .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD:       .XDATA.W        0.03125
ANS:            .RES.W          1
```

4. 並行実行命令

概要

XRAM_ADDアドレス(H'1000F000)とYRAM_ADDアドレス(H'1001F000)から順に確保してある計4つのデータで加算と乗算を行い、加算結果をANS1アドレス(H'1000F004)に、乗算結果をANS2アドレス(H'1001F004)へ格納する。

説明

1. 並行実行命令の構成

並行実行命令は、DSP演算と同時に、DSPレジスタとXメモリ、Yメモリとのデータ転送を行なえます。命令長は32ビットになります。**表4.1**にデータ転送とDSP演算の構成を示します。並行処理命令の構成はDSP演算部とデータ転送部からなります。**表4.2**に並行実行命令の書式例を示します。DSP演算部は、通常PAND、PINC、PSHAのように1命令ですが、**表4.2**に示したようにPADDとPMULS命令、または、PSUBとPMULS命令の場合のみ、2命令の構成となります。データ転送部は、Xメモリとのデータ転送命令とYメモリとのデータ転送命令の2命令の構成となりますが、どちらか一方のデータ転送命令でも構いません。

表4.1　データ転送とDSP演算の構成

	種類	使用バス	転送データ長	DSP演算との並行処理	データ転送の並行処理	命令長
❶	ダブル データ転送	Xバス Yバス	16ビット	なし	なし：どちらか一方のデータ転送	16ビット
					あり：Xメモリ、Yメモリ両方とのデータ転送	
❷				あり	なし：どちらか一方のデータ転送	32ビット
					あり：Xメモリ、Yメモリ両方とのデータ転送	
	シングル データ転送	Cバス*1	16ビット 32ビット	なし		16ビット

表4.2　並行実行命令の書式例

DSP演算部	データ転送部
PADD X0,Y0,A0 PMULS X0,Y0,A1	MOVX.W A0,@R4 MOVY.W A1,@R6
PSUB X1,Y1,A1 PMULS X0,Y1,A0	MOVX.W @R5,X1 MOVY.W @R7,Y1
PADD X0,Y0,A0 PMULS X0,Y0,A1	MOVX.W A0,@R4
PINC X0,Y0,A0	MOVY.W @R6,Y1
PAND X0,Y0,A0	MOVX.W A0,@R5
PSHA X0,Y0,A0	MOVX.W @R4,X1 MOVY.W A1,@R7

*1：ただし、各製品毎に名称がかわります。

2. ダブルデータ転送とDSP演算との並行処理

次項フローチャート中の処理①は**表4.1**での❶のDSP演算命令と並行処理しないダブルデータ転送であり、処理②③は**表4.1**での❷のDSP演算命令と並行処理しているダブルデータ転送です。処理②③は４命令の構成をとっており、1ステップ内で記述できる最大命令数です。この場合、実行ステート数は1ステートです。

3. データ転送部へのDSP演算部結果の影響

表4.3にデータ転送部へのDSP演算部結果の影響を示します。命令2(処理③)ではA0、A1をDSP演算部でデスティネーションレジスタとして使用し、データ転送部でソースレジスタとして使用していますが、DSP演算部の結果がデータ転送部でのストアデータにはなりません。ここでは、下線部のレジスタが影響するので命令1(処理②)の演算部での演算結果が命令2(処理③)のデータ転送部でストアされます。

図4.1に命令2のパイプラインの流れを示します。並列して命令を実行する場合、各命令は**図4.1**のように同時に独立して処理を行ないます。ここで、DSP演算部の結果がデータ転送部でのストアデータにならないのは、PADD,PMULSでのDSP演算を行なうWB/DSPステージがMOVX.W,MOVY.Wでのメモリアクセスを行なうMAステージよりも遅いためです。

なお、命令2(処理③)を実行後のA0、A1レジスタにはX1、Y1の加算、乗算結果が格納されます

表4.3　データ転送部へのDSP演算部結果の影響

メインプログラム抜粋
;命令1 PADD　X0,Y0,A0　PMULS　X0,Y0,A1　MOVX.W　@R4,X1　MOVY.W　@R6,Y1 ;命令2 PADD　X1,Y1,A0　PMULS　X1,Y1,A1　MOVX.W　A0,@R5+　MOVY.W　A1,@R7+
レジスタ内容
命令2実行前：X1=H'1000 0000,Y1=H'0800 0000,A0=H'6000 0000,A1=H'1000 0000 命令2実行後：X1=H'1000 0000,Y1=H'0800 0000,A0=H'1800 0000,A1=H'0100 0000

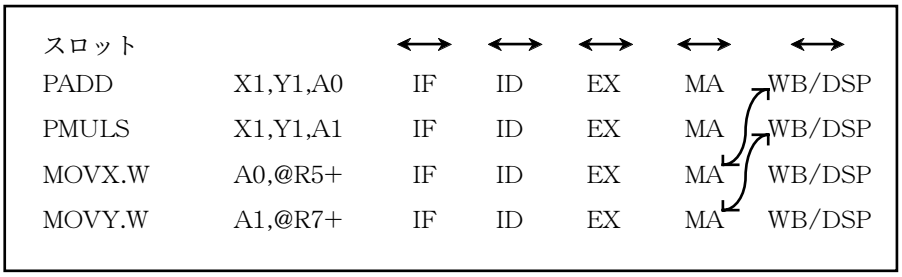
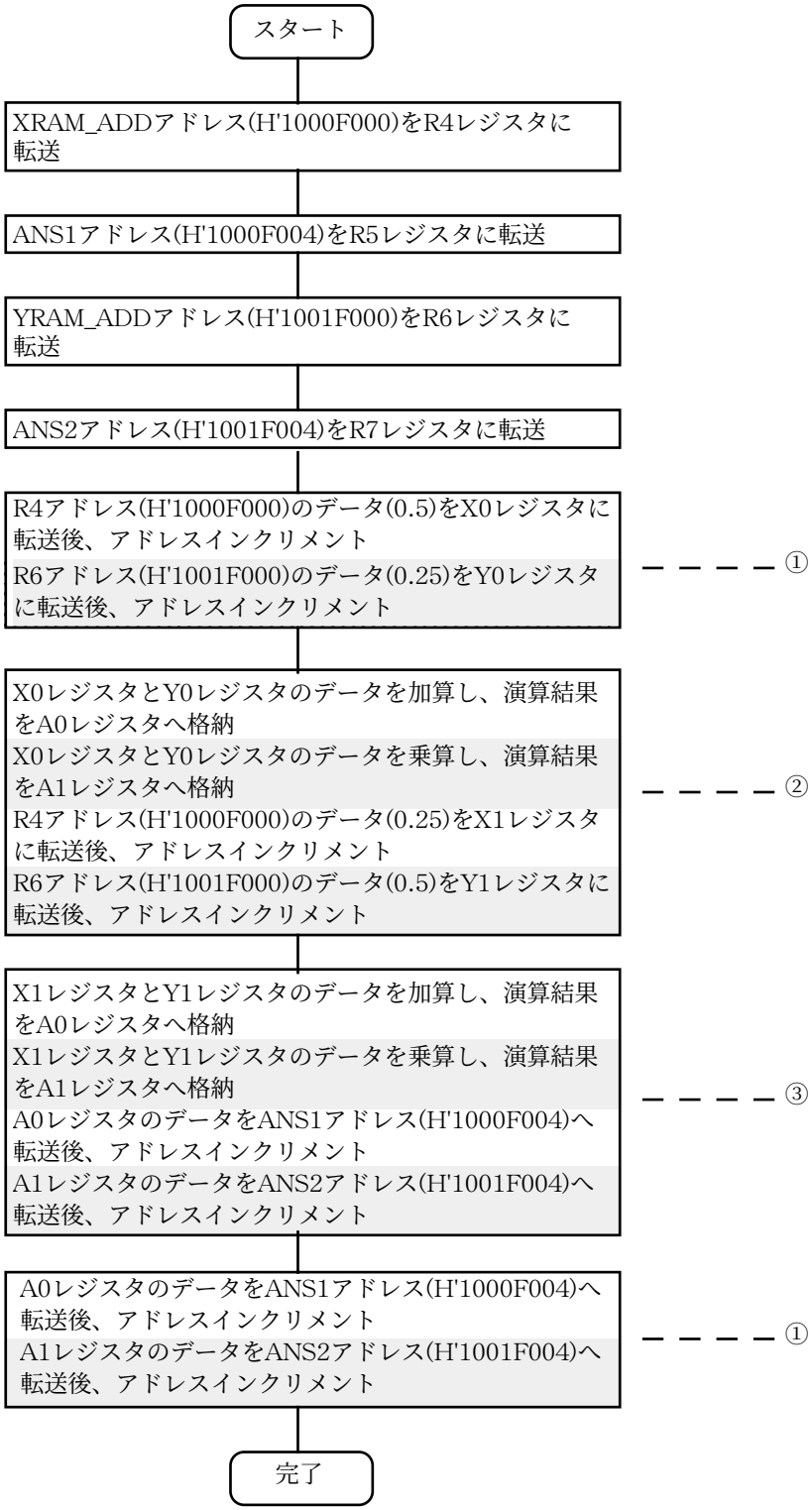


図4.1　命令2のパイプラインの流れ

フローチャート


スタート
XRAM_ADDアドレス(H'1000F000)をR4レジスタに転送
ANS1アドレス(H'1000F004)をR5レジスタに転送
YRAM_ADDアドレス(H'1001F000)をR6レジスタに転送
ANS2アドレス(H'1001F004)をR7レジスタに転送
R4アドレス(H'1000F000)のデータ(0.5)をX0レジスタに転送後、アドレスインクリメント
R6アドレス(H'1001F000)のデータ(0.25)をY0レジスタに転送後、アドレスインクリメント
①
X0レジスタとY0レジスタのデータを加算し、演算結果をA0レジスタへ格納
X0レジスタとY0レジスタのデータを乗算し、演算結果をA1レジスタへ格納
R4アドレス(H'1000F000)のデータ(0.25)をX1レジスタに転送後、アドレスインクリメント
R6アドレス(H'1001F000)のデータ(0.5)をY1レジスタに転送後、アドレスインクリメント
②
X1レジスタとY1レジスタのデータを加算し、演算結果をA0レジスタへ格納
X1レジスタとY1レジスタのデータを乗算し、演算結果をA1レジスタへ格納
A0レジスタのデータをANS1アドレス(H'1000F004)へ転送後、アドレスインクリメント
A1レジスタのデータをANS2アドレス(H'1001F004)へ転送後、アドレスインクリメント
③
A0レジスタのデータをANS1アドレス(H'1000F004)へ転送後、アドレスインクリメント
A1レジスタのデータをANS2アドレス(H'1001F004)へ転送後、アドレスインクリメント
①
完了

メインプログラム

```
;*****************************************************************************************************************************
;*                                    ﾊﾟﾗﾚﾙﾃﾞｰﾀ転送ﾙｰﾁﾝ
;*****************************************************************************************************************************
MAIN:   MOV.L           #XRAM_ADD,R4
        MOV.L           #ANS1,R5
        MOV.L           #YRAM_ADD,R6
        MOV.L           #ANS2,R7
                                                        MOVX.W  @R4+,X0   MOVY.W  @R6+,Y0   ;並行処理無
        PADD    X0,Y0,A0  PMULS   X0,Y0,A1              MOVX.W  @R4,X1    MOVY.W  @R6,Y1    ;並行処理有
        PADD    X1,Y1,A0  PMULS   X1,Y1,A1              MOVX.W  A0,@R5+   MOVY.W  A1,@R7+   ;並行処理有
                                                        MOVX.W  A0,@R5    MOVY.W  A1,@R7    ;並行処理無
EXIT:   BRA             EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*****************************************************************************************
;*                      ﾃﾞｰﾀ(X/YRAM)
;*****************************************************************************************
                .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:       .XDATA.W        0.5,0.125               ;DSP演算用ﾃﾞｰﾀ
ANS1:           .RES.W          2                       ;DSP演算結果格納ｴﾘｱ

                .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD:       .XDATA.W        0.25,0.0625             ;DSP演算用ﾃﾞｰﾀ
ANS2:           .RES.W          2                       ;DSP演算結果格納ｴﾘｱ
```

5. リピート命令

概要

XRAM、YRAMに確保された計10個のデータの平均を取る。この際、XRAM、YRAMからDSPユニットへのデータ転送と10個のデータの加算にリピート機能を用いる。

説明

1. DSPリピート制御

繰り返し制御を行なうためには、Ⅰ繰り返したいプログラムの開始アドレスの設定、Ⅱ繰り返したいプログラムの終了アドレスの設定、Ⅲ繰り返したい回数の設定の3つの設定が必要です。Ⅰ～Ⅲの設定が終了後、Ⅳ繰り返したいプログラムを開始します。なお、Ⅲ、Ⅳの処理の間には少なくとも1命令が必要です。
処理Ⅰ～Ⅳの手順を以下に示します。

Ⅰ　繰り返し開始アドレスをLDRS命令を用いて、RSレジスタに設定します。
Ⅱ　繰り返し終了アドレスをLDRE命令を用いて、REレジスタに設定します。
Ⅲ　繰り返し回数をSETRC命令を用いて、SRレジスタのRCカウンタに設定します。
　　：　　(最低1命令を挿入)
Ⅳ　繰り返しプログラムを開始します。

次項フローチャート中の①の処理は、上記のⅠ～Ⅲに当たります。Ⅳの繰り返しプログラム開始後、②の範囲でリピートします。メインプログラムには2種類のプログラムを載せていますが、機能は同じです。(1)はリピート制御用命令(LDRS,LDRE,SETRC)を用いて行なう方法で、(2)は拡張命令のREPEATを用いて行なう方法です。拡張命令のREPEATはリピート開始アドレスから終了アドレスまでの命令をリピートするためのCPU命令(LDRS,LDRE,SETRC)を自動的に生成します。以下に示す書式内のリピート回数を省略した場合、SETRC命令を生成しません。

書式：

REPEAT　　〔開始アドレス〕,〔終了アドレス〕,〔リピート回数〕

(1)のプログラムでは、リピート開始アドレスとリピート終了アドレスが実際のアドレスと異なりますが、リピートプログラム内の命令数でアドレスの設定が変わるためです。**表5.1**にリピート内命令数によるRS、REの設定値を示します。これは、RS、REにリピート開始アドレスとリピート終了アドレスを設定した場合、実際にプログラムがリピートするアドレスです。このため、ユーザ側で**表5.1**のオフセットを考慮に入れて、リピート開始アドレスとリピート終了アドレスをラベリングする必要があります。次ページに(1)プログラムのRS、REの設定方法を説明します。

RPT_S0+N　：　リピートプログラム開始アドレスの命令の1つ前の命令から、Nバイト目のアドレス
RPT_S　　：　リピートプログラム開始アドレス
RPT_E　　：　リピートプログラム終了アドレス
RPT_E3+4　：　リピートプログラム終了アドレスの命令の3つ前の命令から、4バイト目のアドレス

表5.1　リピート内命令数によるRS、REの設定値

	リピートプログラム内の命令数			
	1	2	3	4
RS	RPT_S0+8	RPT_S0+6	RPT_S0+4	RPT_S
RE	RPT_S0+4	RPT_S0+4	RPT_S0+4	RPT_E3+4

説明

2. CPU命令によるリピート制御

(a)にRS、REアドレスの設定法を示します。3命令の場合、**表5.1**より、RS、REにはそれぞれRPT_S0+4アドレスを設定する必要があります。このプログラムの場合、❶と❷行のダブルデータ転送命令は、16ビット命令長なので、RPT_S0+4アドレスはRPT_E0アドレスにあたります。ここでRS、REにRPT_E0のアドレスを設定することで、(b)のプログラムとなります。

```
        LDRS            RPT_S0+4のｱﾄﾞﾚｽ                                          ;ﾘﾋﾟｰﾄｽﾀｰﾄｱﾄﾞﾚｽ
        LDRE            RPT_S0+4のｱﾄﾞﾚｽ                                          ;ﾘﾋﾟｰﾄｴﾝﾄﾞｱﾄﾞﾚｽ
        SETRC           #5                                                      ;ﾘﾋﾟｰﾄｶｳﾝﾄｾｯﾄ/5回
RPT_S0:                                 ❶  MOVX.W  @R5,X1   MOVY.W  @R7,Y1  ;X1ｸﾘｱ,Y1=1/10
RPT_S:                                  ❷  MOVX.W  @R4+,X0  MOVY.W  @R6+,Y0
RPT_E0: PADD    X0,Y0,M0
RPT_E:  PADD    X1,M0,X1                                                        ;X1/ﾃﾞｰﾀ合計
        PMULS   X1,Y1,A1                                                        ;A1/平均値
```

(a) RS、REアドレスの設定法

```
        LDRS            RPT_E0                                                  ;ﾘﾋﾟｰﾄｽﾀｰﾄｱﾄﾞﾚｽ
        LDRE            RPT_E0                                                  ;ﾘﾋﾟｰﾄｴﾝﾄﾞｱﾄﾞﾚｽ
        SETRC           #5                                                      ;ﾘﾋﾟｰﾄｶｳﾝﾄｾｯﾄ/5回
RPT_S0:                                 MOVX.W  @R5,X1   MOVY.W  @R7,Y1     ;X1ｸﾘｱ,Y1=1/10
RPT_S:                                  MOVX.W  @R4+,X0  MOVY.W  @R6+,Y0
RPT_E0: PADD    X0,Y0,M0
RPT_E:  PADD    X1,M0,X1                                                        ;X1/ﾃﾞｰﾀ合計
        PMULS   X1,Y1,A1                                                        ;A1/平均値
```

(b) RS、REアドレスの設定法

3. 拡張命令によるリピート制御

拡張命令のREPEATでは、CPU命令を用いたリピート制御のように複雑なラベリングをする必要がありません。(a)のリピートプログラム部の展開イメージを例に説明します。REPEATにリピートさせたいプログラム部の開始アドレス(RPT_S)、終了アドレス(RPT_E)のラベルを記述するだけで、アセンブラが自動的にRS、REに設定するアドレス値を計算し(3命令では、RPT_E0)、LDRS、LDRE命令を生成してくれます。実際の拡張命令REPEATを使用した場合、(b)の拡張命令REPEATによるリピートプログラムとなります。

```
          REPEAT   RPT_S,RPT_E,#5
                   ↓
          LDRS              RPT_E0    ;RPT_S0+4
          LDRE              RPT_E0    ;RPT_S0+4
          SETRC             #5
RPT_S0:                                          MOVX.W  @R5,X1   MOVY.W  @R7,Y1
RPT_S:                                           MOVX.W  @R4+,X0  MOVY.W  @R6+,Y0
RPT_E0:   PADD      X0,Y0,M0
RPT_E:    PADD      X1,M0,X1
                    PMULS     X1,Y1,A1
```

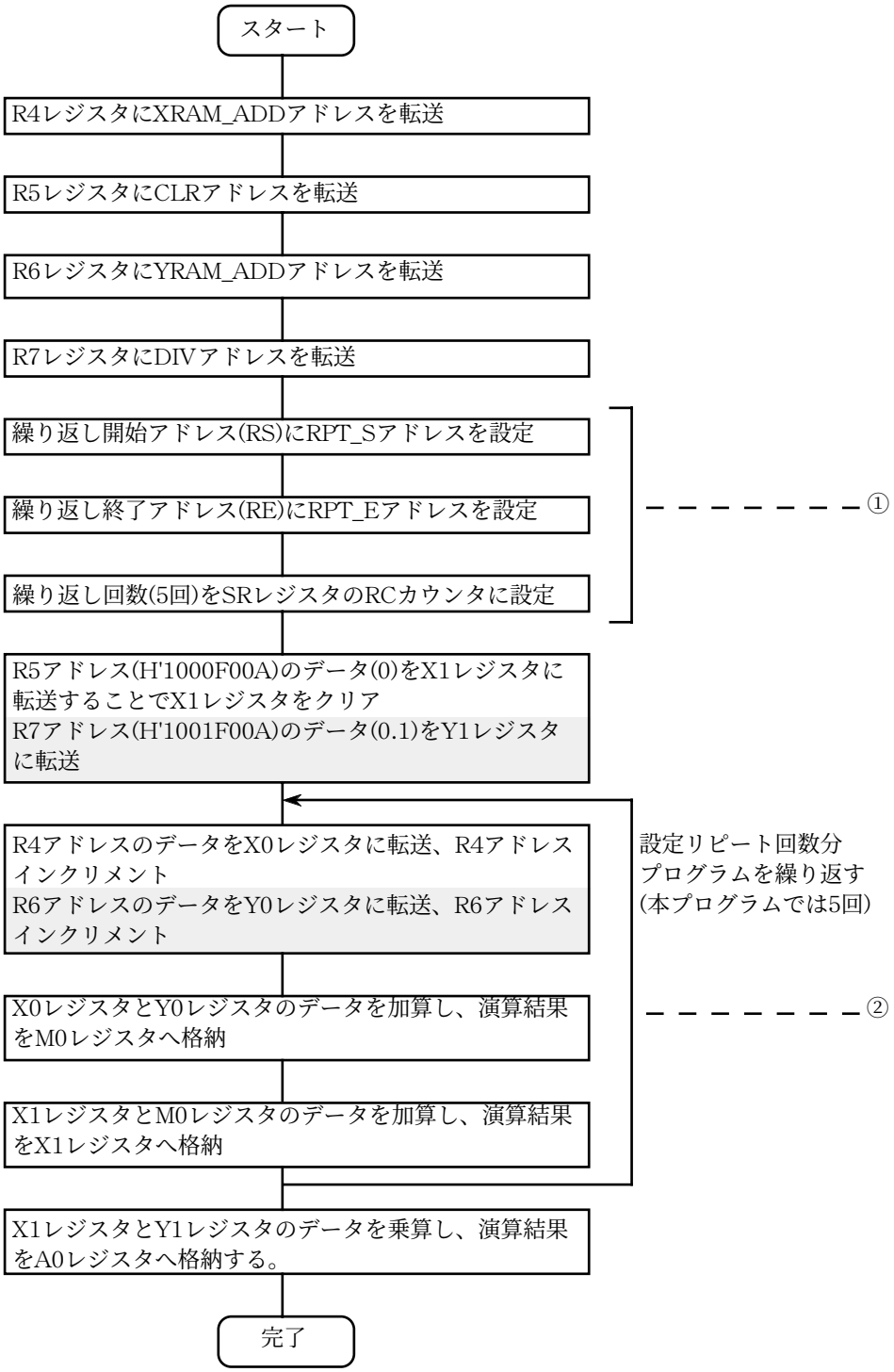
リピート制御用のCPU命令が展開します。

(a)　リピートプログラム部の展開イメージ

```
          REPEAT   RPT_S,RPT_E,#5
RPT_S0:                                          MOVX.W  @R5,X1   MOVY.W  @R7,Y1
RPT_S:                                           MOVX.W  @R4+,X0  MOVY.W  @R6+,Y0
RPT_E0:   PADD      X0,Y0,M0
RPT_E:    PADD      X1,M0,X1
                    PMULS     X1,Y1,A1
```

(b)　拡張命令REPEATによるリピートプログラム

フローチャート


スタート
R4レジスタにXRAM_ADDアドレスを転送
R5レジスタにCLRアドレスを転送
R6レジスタにYRAM_ADDアドレスを転送
R7レジスタにDIVアドレスを転送
繰り返し開始アドレス(RS)にRPT_Sアドレスを設定
繰り返し終了アドレス(RE)にRPT_Eアドレスを設定
繰り返し回数(5回)をSRレジスタのRCカウンタに設定
①
R5アドレス(H'1000F00A)のデータ(0)をX1レジスタに転送することでX1レジスタをクリア
R7アドレス(H'1001F00A)のデータ(0.1)をY1レジスタに転送
R4アドレスのデータをX0レジスタに転送、R4アドレスインクリメント
R6アドレスのデータをY0レジスタに転送、R6アドレスインクリメント
X0レジスタとY0レジスタのデータを加算し、演算結果をM0レジスタへ格納
②
X1レジスタとM0レジスタのデータを加算し、演算結果をX1レジスタへ格納
設定リピート回数分
プログラムを繰り返す
(本プログラムでは5回)
X1レジスタとY1レジスタのデータを乗算し、演算結果をA0レジスタへ格納する。
完了

メインプログラム

(1)CPU命令によるリピート制御

```
;*****************************************************************************************************************************************
;*                                    ﾘﾋﾟｰﾄﾙｰﾁﾝ
;*****************************************************************************************************************************************
MAIN:     MOV.L               #XRAM_ADD,R4
          MOV.L               #CLR,R5
          MOV.L               #YRAM_ADD,R6
          MOV.L               #DIV,R7
          LDRS                RPT_E0                                                                    ;ﾘﾋﾟｰﾄｽﾀｰﾄｱﾄﾞﾚｽ
          LDRE                RPT_E0                                                                    ;ﾘﾋﾟｰﾄｴﾝﾄﾞｱﾄﾞﾚｽ
          SETRC               #5                                                                        ;ﾘﾋﾟｰﾄｶｳﾝﾄｾｯﾄ/5回
                                                          MOVX.W  @R5,X1    MOVY.W  @R7,Y1            ;X1ｸﾘｱ,Y1=1/10
RPT_S:                                                    MOVX.W  @R4+,X0   MOVY.W  @R6+,Y0
RPT_E0:   PADD      X0,Y0,M0
RPT_E:    PADD      X1,M0,X1                                                                          ;X1/ﾃﾞｰﾀ合計
                    PMULS     X1,Y1,A1                                                                ;A1/平均値
EXIT:     BRA       EXIT
          NOP
MAIN_E:   NOP
```

(2)拡張命令REPEATによるリピート制御

```
;*****************************************************************************************************************************************
;*                                    ﾘﾋﾟｰﾄﾙｰﾁﾝ
;*****************************************************************************************************************************************
MAIN:     MOV.L               #XRAM_ADD,R4
          MOV.L               #CLR,R5
          MOV.L               #YRAM_ADD,R6
          MOV.L               #DIV,R7
          MOV.L               #5,R0
          REPEAT    RPT_S,RPT_E,R0                                                                    ;ﾘﾋﾟｰﾄ制御用CPU命令を自動的に生成
                                                          MOVX.W  @R5,X1    MOVY.W  @R7,Y1            ;X1ｸﾘｱ,Y1=1/10
RPT_S:                                                    MOVX.W  @R4+,X0   MOVY.W  @R6+,Y0
          PADD      X0,Y0,M0
RPT_E:    PADD      X1,M0,X1                                                                          ;X1/ﾃﾞｰﾀ合計
                    PMULS     X1,Y1,A1                                                                ;A1/平均値
EXIT:     BRA       EXIT
          NOP
MAIN_E:   NO
```

※メインプログラム(1)(2)共通データ

```
;*******************************************************************************
;*                    ﾃﾞｰﾀ(X/YRAM)
;*******************************************************************************

                .SECTION XRAM,CODE,LOCATE=H'1000F000
XRAM_ADD:       .XDATA.W        0.0625,0.125,0.0625,0.0625,0.03125      ;DSP演算用ﾃﾞｰﾀ
CLR:            .DATA.W         0                                       ;DSP演算結果格納ｴﾘｱ

                .SECTION YRAM,CODE,LOCATE=H'1001F000
YRAM_ADD:       .XDATA.W        0.0625,0.125,0.03125,0.125,0.0625       ;DSP演算用ﾃﾞｰﾀ
DIV:            .XDATA.W        0.1                                     ;DSP演算結果格納ｴﾘｱ
```

6. CPU命令とDSP命令間の引数例

概要

XRAM_ADDアドレス(H'1000F000)とYRAM_ADDアドレス(H'1001F000)に確保してある２つの16ビット固定小数点データをDSP命令とCPU命令で乗算を行ないます。

説明

DSP命令とCPU命令間でデータをやりとりする場合、R4、R5、R6、R7レジスタをポインタとしてX/YRAMを介してデータをやりとりします。DSP側で演算した結果をCPU側で用いる場合は以下のようになります。

本メインプログラムでは、**(2-1)と(3-1)**、**(3-2)**のように②DSP乗算ルーチン、③CPU乗算ルーチン共にX/YRAM上に確保しているデータのリードを行なっています。

引数例：

```
PADD      X0,Y0,A0      ;X0とY0を加算し結果をA0に格納します
MOVX.W    A0,@R4        ;A0のデータをR4アドレスに転送します
MOV.W     @R4,R0        ; R4アドレスのデータをR0に転送します
```

データをやりとりする際に注意する点があります。DSP命令の中には固定小数点でデータを扱うものがあり、固定小数点乗算では演算結果がMSBに合わせられます。CPU命令での乗算では、整数演算であり演算結果がLSBに合わせられるため、DSP命令の演算結果とは異なってきます。

次項フローチャート中の②DSP乗算ルーチン、③CPU乗算ルーチンの**(2-2)と(3-3)**、**(3-4)**での演算経過を**表6.1**に示します。ソースオペランドのデータが同じであっても実行後の演算結果が異なっていることが分かります。DSP命令(PMULS)で整数データを乗算する場合は、演算結果をビットシフトして固定小数点から整数へフォーマットの変換を行なってください。

表6.1　DSP、CPUでの乗算経過

	メインプログラム抜粋		レジスタ内容
②DSP乗算ルーチン	PMULS	X0,Y0,A0	実行前　X0=H'4000,Y0=H'2000 実行後　A0=H'1000 0000
③CPU乗算ルーチン	MULS.W STS	R0,R1 MACL,R14	実行前　R0=H'4000,R1=H'2000 実行後　R14=H'0800 0000

フローチャート


スタート
①
XRAM_ADDアドレス(H'1000F000)をR4レジスタに転送
(1-1)
YRAM_ADDアドレス(H'1001F000)をR6レジスタに転送
(1-2)
②
R4アドレス(H'1000F000)のデータ(H'4000)をX0レジスタに転送
R6アドレス(H'1001F000)のデータ(H'2000)をY0レジスタに転送
(2-1)
X0レジスタとY0レジスタのデータを乗算し、演算結果をA0レジスタへ格納する。
(2-2)
③
R4アドレス(H'1000F000)のデータ(H'4000)をR0レジスタに転送
(3-1)
R6アドレス(H'1001F000)のデータ(H'2000)をR1レジスタに転送
(3-2)
R0レジスタとR1レジスタのデータを乗算する
(3-3)
MACLレジスタのデータ(乗算結果)をR14レジスタへ転送
(3-4)
完了

メインプログラム

```
;*****************************************************************************************************************************
;*                              初期設定ﾙｰﾁﾝ
;*****************************************************************************************************************************
MAIN:   MOV.L           #XRAM_ADD,R4
        MOV.L           #YRAM_ADD,R6

;*****************************************************************************************************************************
;*                              DSP乗算ﾙｰﾁﾝ
;*****************************************************************************************************************************
                                                MOVX.W  @R4,X0    MOVY.W  @R6,Y0        ;0.5,0.25 load
        PMULS   X0,Y0,A0                                                              ;A0=乗算結果

;*****************************************************************************************************************************
;*                              CPU乗算ﾙｰﾁﾝ
;*****************************************************************************************************************************
        MOV.L           @R4,R0                                                        ;H'4000 load
        MOV.L           @R6,R1                                                        ;H'2000 load
        MULS.W          R0,R1
        STS             MACL,R14                                                      ;R14=乗算結果

EXIT:   BRA             EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*******************************************************************************************
;*                      ﾃﾞｰﾀ
;*******************************************************************************************
                .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:       .XDATA.W 0.5                            ;DSP演算用ﾃﾞｰﾀ

                .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD        .XDATA.W 0.25                           ;DSP演算用ﾃﾞｰﾀ
                .END
```

7. 32ビット乗算

概要

XRAM_ADD(H'1000F000)アドレスの32ビットデータとYRAM_ADD(H'1001F000)アドレスの32ビットデータを乗算し、演算結果(64ビット)をANS(H'1001F100)アドレスからANS+7(H'1001F107)アドレスに格納する。

説明

1. 計算法概要

32ビット乗算の乗数と被乗数の格納アドレスと、乗算後の結果格納アドレスを**図7.1**の32ビット乗算に示します。**図7.2**に32ビット乗算の計算法概要を示します。32ビットのデータ(乗数、被乗数)を上位／下位の16ビット(仮にA、B、C、Dと置く)に分けてそれぞれ乗算することによって64ビット演算結果を算出します。乗算器にデータを入力する16ビットデータの先頭ビット(MSB)は、符号ビットとみなされ、$-2^0=-1$の重みを持ちます。そのため、本プログラムでは、最初に先頭ビット(MSB)を強制的に0に置き換えて個々の部分積を計算し、それに先頭ビットによる補正項を加えるという方法で32ビット乗算の結果を求めます。

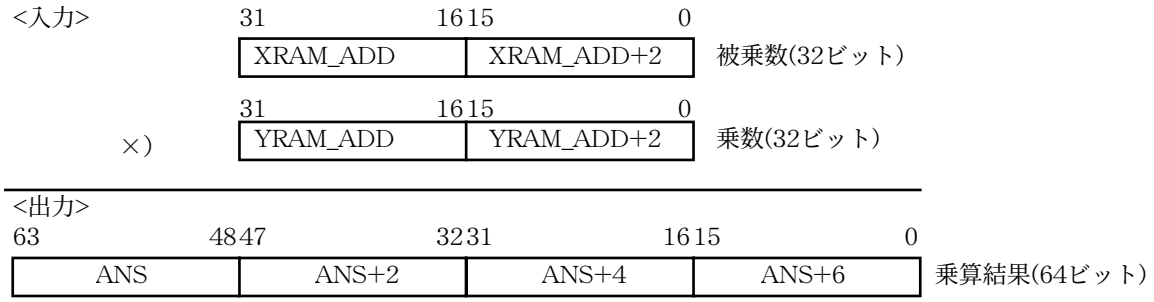


図7.1　32ビット乗算

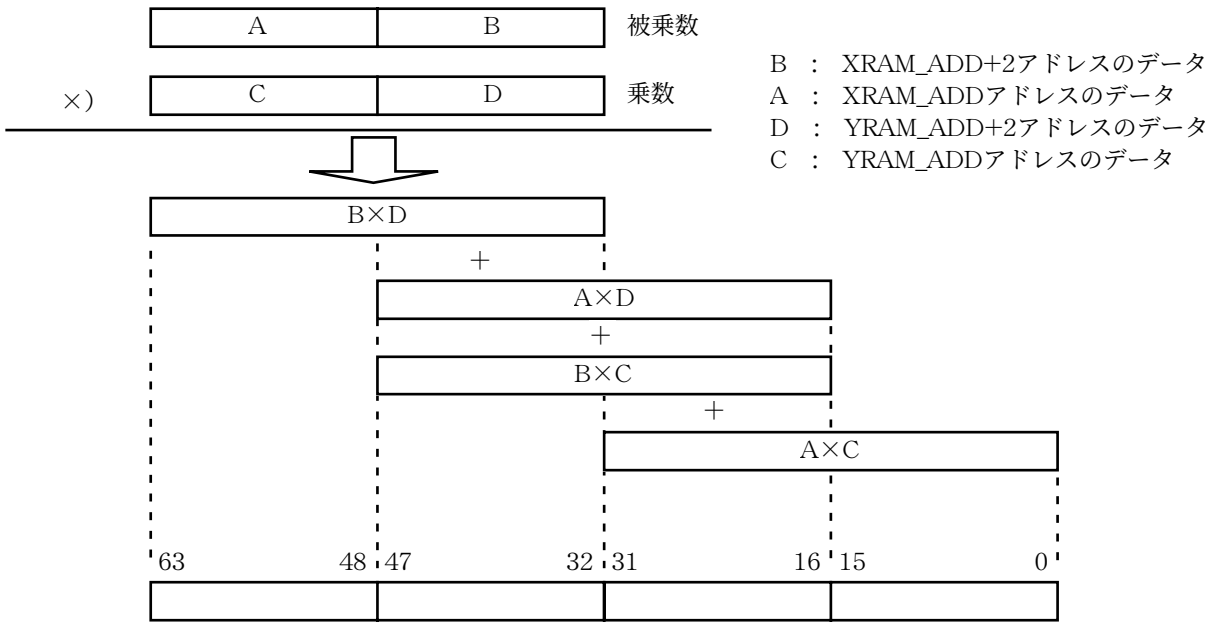


図7.2　32ビット乗算の計算法概要

2. 倍長乗算のアルゴリズム

単精度のビット長をnビットとすると、倍長は2nビットと表現することができます。これより、2nビットの数は、**図7.3**に示すように表すことができます。

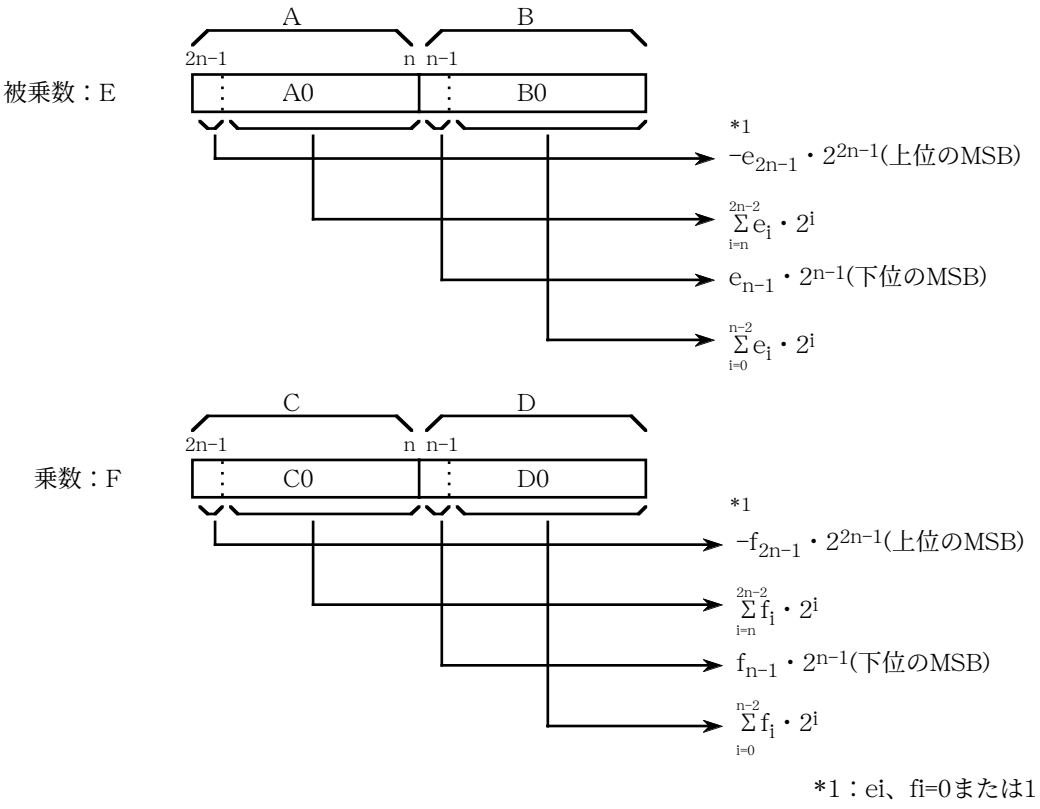


図7.3　2nビットの数の構成

ここで、$\Sigma e_i\cdot 2^i$=A0、$\Sigma e_i\cdot 2^i$=B0、$\Sigma f_i\cdot 2^i$=C0、$\Sigma f_i\cdot 2^i$=D0と置いて、E×Fの倍長乗算を計算すると

$E\times F=(-e_{2n-1}\cdot 2^{2n-1}+A0+e_{2n-1}\cdot 2^{n-1+}+B0)\times(-f_{2n-1}\cdot 2^{2n-1}+C0+f_{2n-1}\cdot 2^{n-1+}+D0)$

$=e_{2n-1}\cdot f_{2n-1}\cdot 2^{4n-2}$❶

$-e_{2n-1}\cdot 2^{2n-1}(C0+f_{n-1}\cdot 2^{n-1+}+D0)$❷

$-f_{2n-1}\cdot 2^{2n-1}(A0+e_{n-1}\cdot 2^{n-1+}+B0)$❸

$+e_{n-1}\cdot 2^{n-1}(C0+f_{n-1}\cdot 2^{n-1+}+D0)$❹

$+f_{n-1}\cdot 2^{n-1}(A0+B0)$❺

$+A0\cdot C0+A0\cdot D0+B0\cdot C0+B0\cdot D0$❻

と表すことができます。上式において❻が部分積、❶～❺が補正項です。
補正項は、符号ビットが"0"または"1"かを判定し"1"ならば、部分積の和に加算もしくは減算します。

図7.4に上記計算方式を用いた32ビット倍長乗算アルゴリズムを示します。全体の処理を大きく分けると①～⑥になります。

①では、A、B、C、Dの符号ビットを0にするために、H'7FFFと論理積をとりA0、B0、C0、D0とします。　②では、A0・C0、A0・D0、B0・C0、B0・D0の4つの部分積を計算します。③～⑥では、同じ桁毎で和を求め、結果をANS、ANS+2、ANS+4、ANS+6アドレスに格納します。

説明

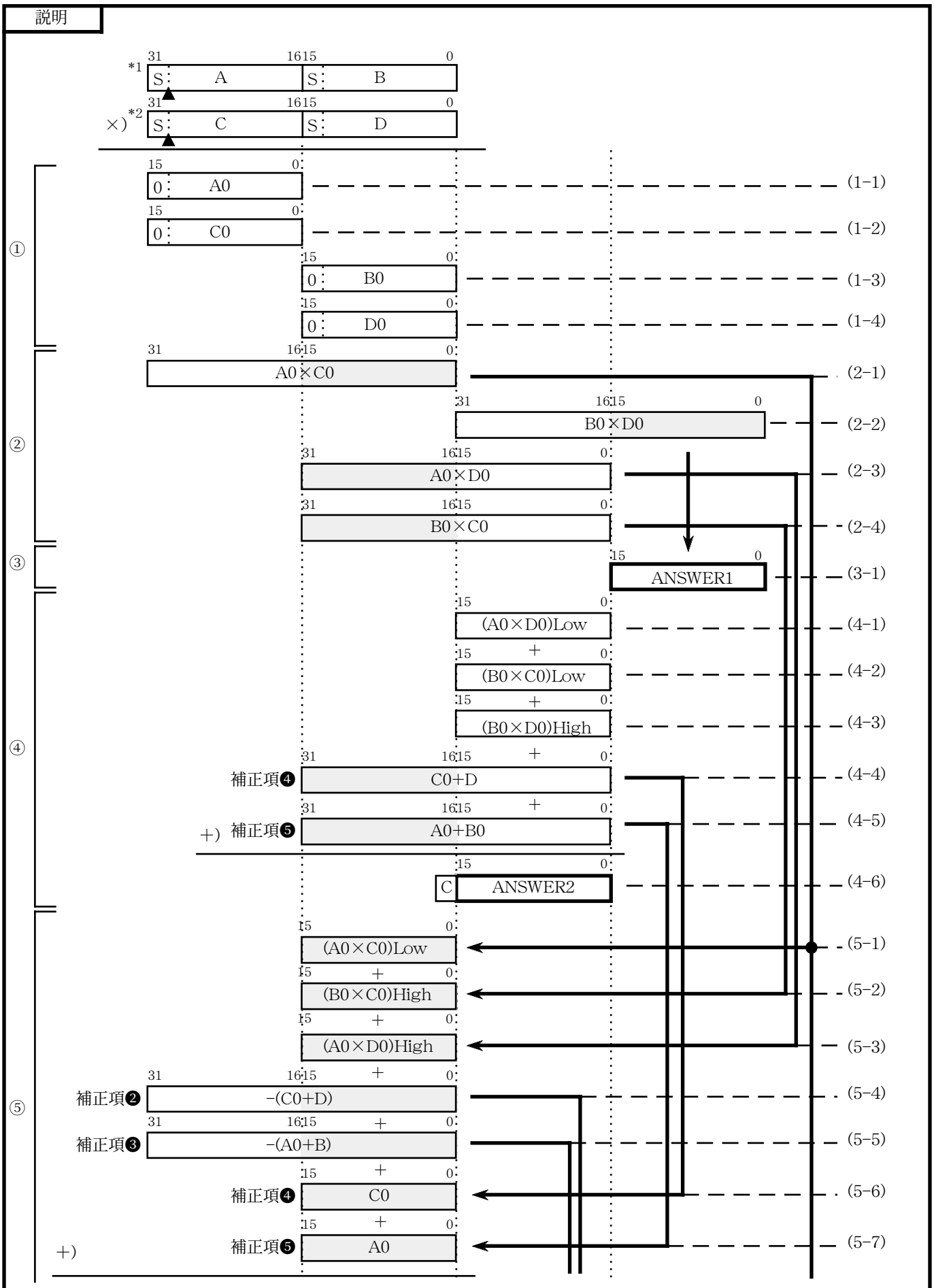


図7.4　32ビット倍長乗算アルゴリズム

*1　S :符号ビット
*2　▲:小数点位置

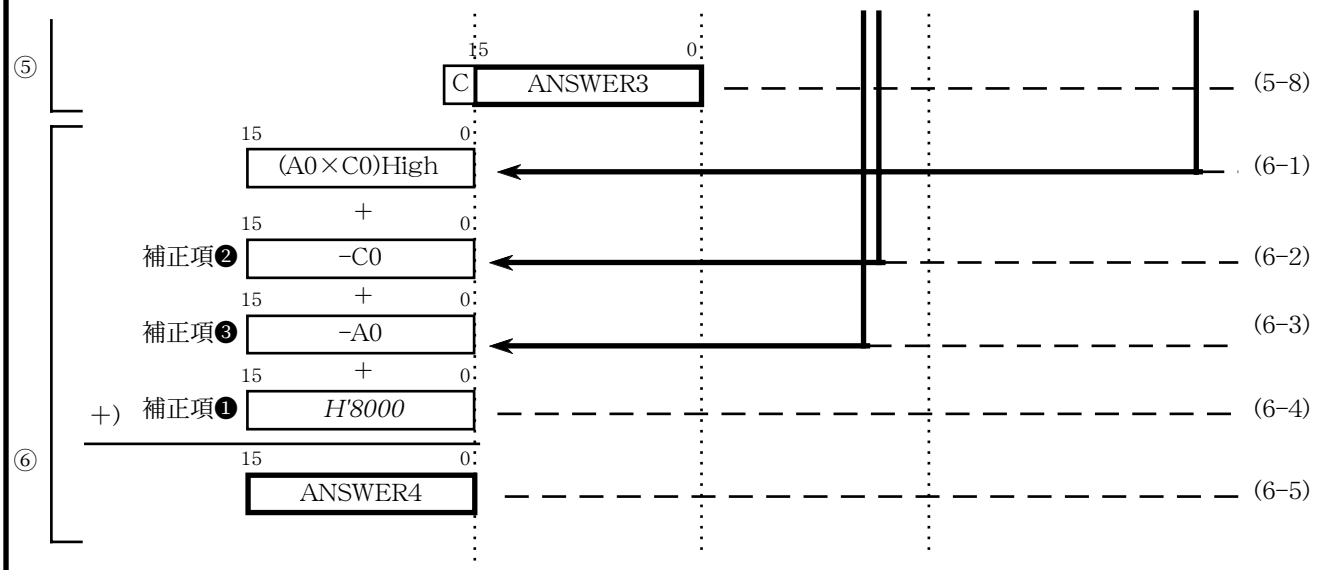
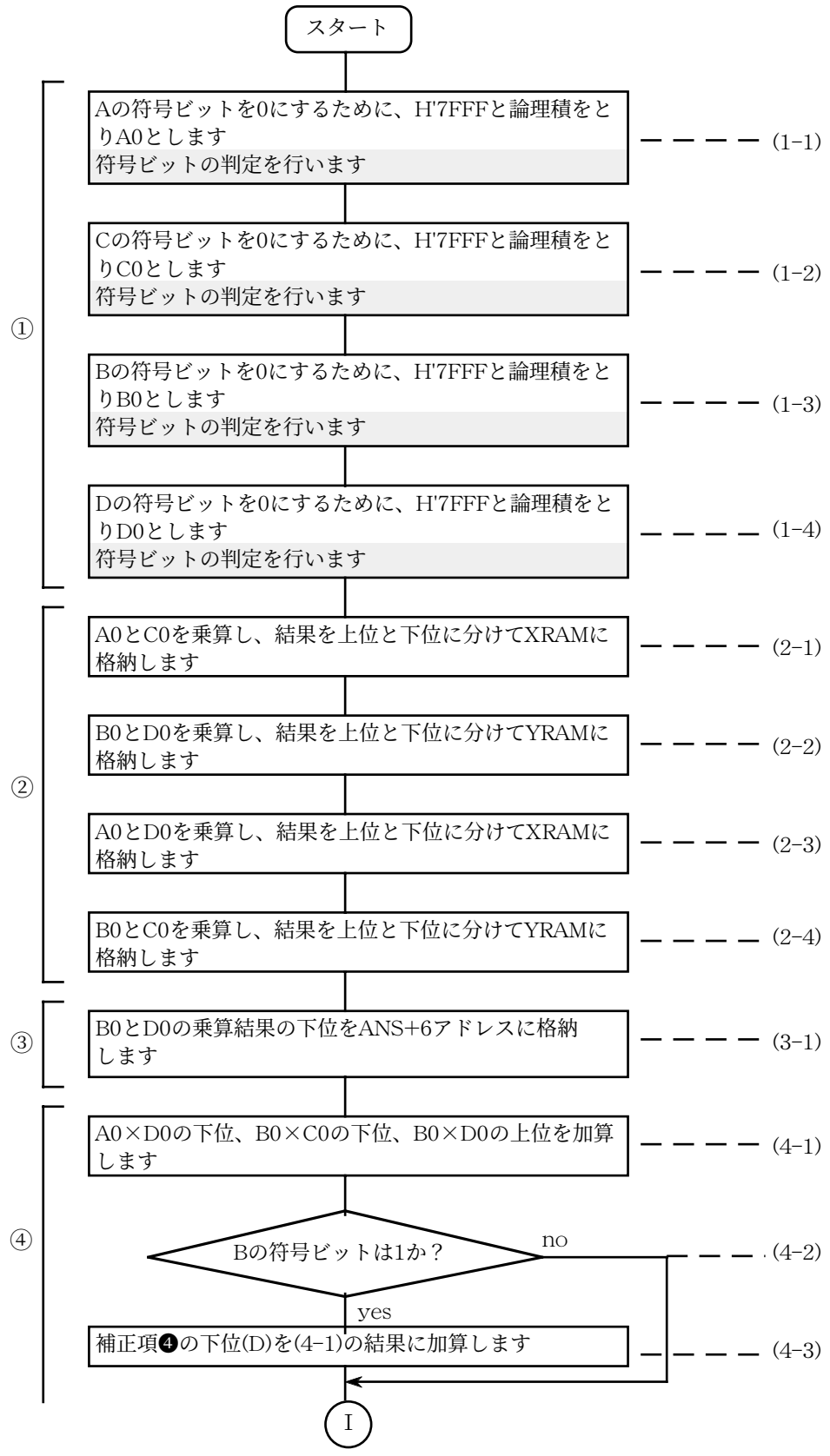
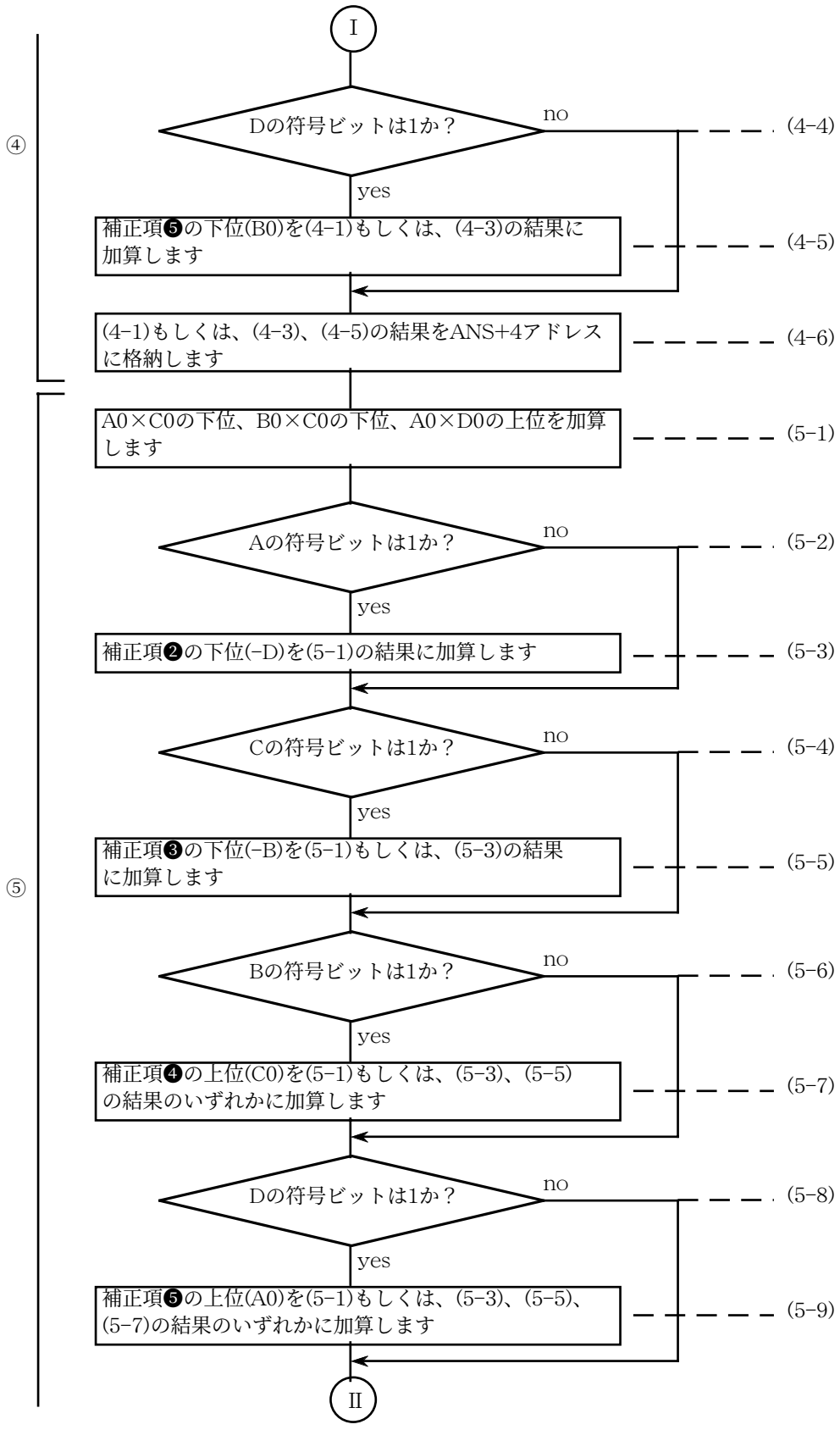


図7.4　32ビット倍長乗算アルゴリズム(続き)

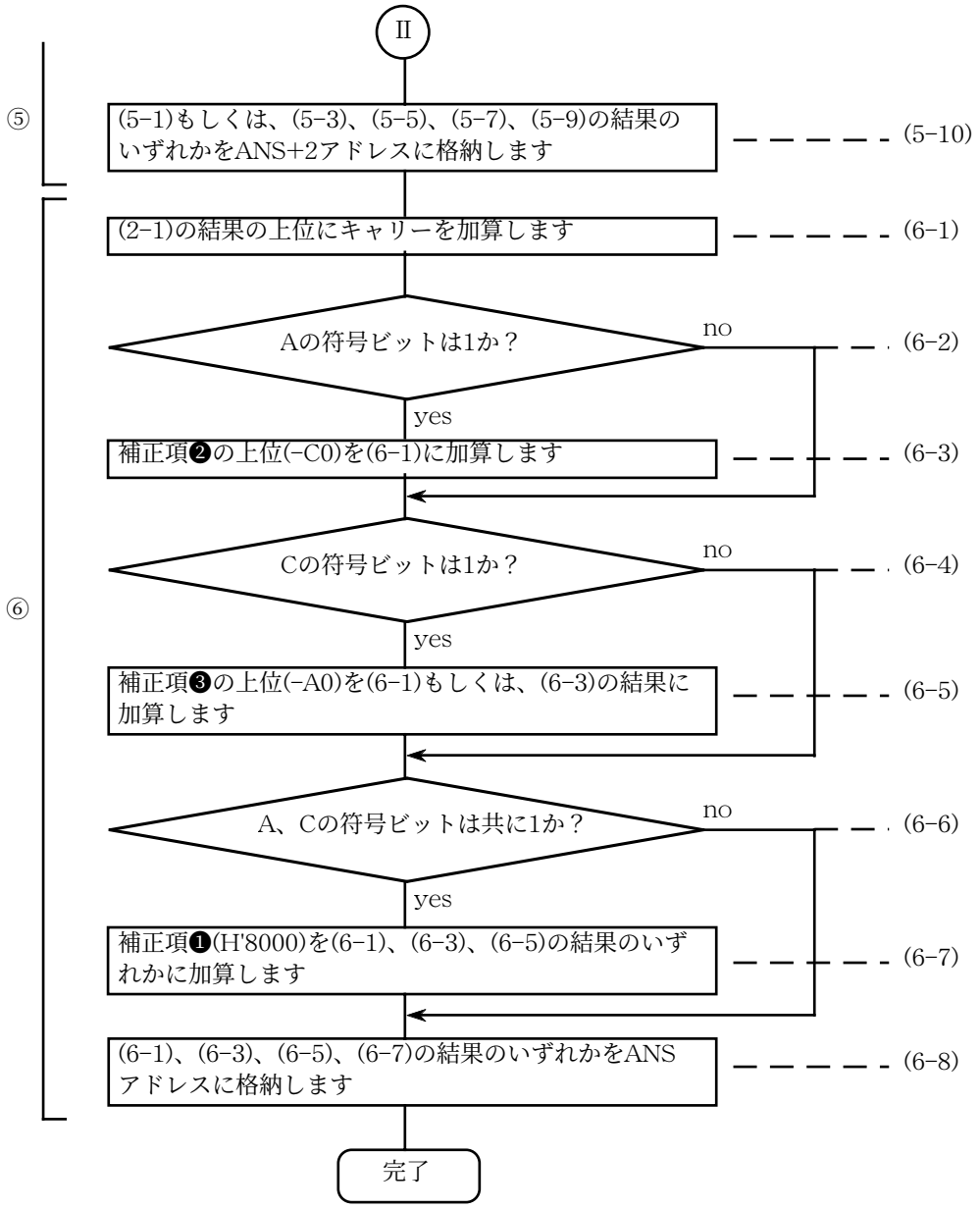
フローチャート


スタート
Aの符号ビットを0にするために、H'7FFFと論理積をとりA0とします
符号ビットの判定を行います
(1-1)
Cの符号ビットを0にするために、H'7FFFと論理積をとりC0とします
符号ビットの判定を行います
(1-2)
Bの符号ビットを0にするために、H'7FFFと論理積をとりB0とします
符号ビットの判定を行います
(1-3)
Dの符号ビットを0にするために、H'7FFFと論理積をとりD0とします
符号ビットの判定を行います
(1-4)
①
A0とC0を乗算し、結果を上位と下位に分けてXRAMに格納します
(2-1)
B0とD0を乗算し、結果を上位と下位に分けてYRAMに格納します
(2-2)
A0とD0を乗算し、結果を上位と下位に分けてXRAMに格納します
(2-3)
B0とC0を乗算し、結果を上位と下位に分けてYRAMに格納します
(2-4)
②
B0とD0の乗算結果の下位をANS+6アドレスに格納します
(3-1)
③
A0×D0の下位、B0×C0の下位、B0×D0の上位を加算します
(4-1)
Bの符号ビットは1か？
no
(4-2)
yes
補正項❹の下位(D)を(4-1)の結果に加算します
(4-3)
④
I

フローチャート


I
Dの符号ビットは1か？
no
(4-4)
yes
補正項❺の下位(B0)を(4-1)もしくは、(4-3)の結果に加算します
(4-5)
(4-1)もしくは、(4-3)、(4-5)の結果をANS+4アドレスに格納します
(4-6)
④
A0×C0の下位、B0×C0の下位、A0×D0の上位を加算します
(5-1)
Aの符号ビットは1か？
no
(5-2)
yes
補正項❷の下位(-D)を(5-1)の結果に加算します
(5-3)
Cの符号ビットは1か？
no
(5-4)
yes
補正項❸の下位(-B)を(5-1)もしくは、(5-3)の結果に加算します
(5-5)
⑤
Bの符号ビットは1か？
no
(5-6)
yes
補正項❹の上位(C0)を(5-1)もしくは、(5-3)、(5-5)の結果のいずれかに加算します
(5-7)
Dの符号ビットは1か？
no
(5-8)
yes
補正項❺の上位(A0)を(5-1)もしくは、(5-3)、(5-5)、(5-7)の結果のいずれかに加算します
(5-9)
II

フローチャート


II
⑤
(5-1)もしくは、(5-3)、(5-5)、(5-7)、(5-9)の結果のいずれかをANS+2アドレスに格納します
(5-10)
(2-1)の結果の上位にキャリーを加算します
(6-1)
Aの符号ビットは1か？
no
(6-2)
yes
補正項❷の上位(-C0)を(6-1)に加算します
(6-3)
Cの符号ビットは1か？
no
(6-4)
⑥
yes
補正項❸の上位(-A0)を(6-1)もしくは、(6-3)の結果に加算します
(6-5)
A、Cの符号ビットは共に1か？
no
(6-6)
yes
補正項❶(H'8000)を(6-1)、(6-3)、(6-5)の結果のいずれかに加算します
(6-7)
(6-1)、(6-3)、(6-5)、(6-7)の結果のいずれかをANSアドレスに格納します
(6-8)
完了

メインプログラム

```
;*****************************************************************************************************************************************
;*                                32ﾋﾞｯﾄ固定小数点乗算ﾙｰﾁﾝ
;*
;*                                [A][B] ×[C][D]
;*
;*****************************************************************************************************************************************
MAIN:   MOV.L   #XRAM_ADD,R4
        MOV.L   #WORKX,R5                                                                       ;ﾜｰｸ用XRAM
        MOV.L   #YRAM_ADD,R6
        MOV.L   #WORKY,R7                                                                       ;ﾜｰｸ用YRAM
;符号ｸﾘｱ
        MOV.W           #H'7FFF,R0
        MOV.W           R0,@R7
        PCLR    A1                      MOVX.W  @R4+,X0         MOVY.W  @R7,Y0          ;A,H'7FFF load
        PAND    X0,Y0,A0                                        MOVY.W  @R6+,Y1         ;A0,C load
        MOV.W           R0,@R5                                                          ;H'7FFF -> #WORKX
        PSHA    #1,X0                   MOVX.W  @R5,X1                                  ;A sign chech,H'7FFF load
DCT     PINC    A1,A1                   MOVX.W  A0,@R5+                                 ;A0 store
        PAND    X1,Y1,A0                MOVX.W  @R4,X0                                  ;C0,B load
        MOV.L           R4,@-R15
        MOV.L           #SIGNA,R4
        PCLR    A1                      MOVX.W  A1,@R4+
        PSHA    #1,Y1                                           MOVY.W  A0,@R7+         ;C sign check,C0 store
DCT     PINC    A1,A1                                           MOVY.W  @R6,Y1          ;B sign check,D load
        PAND    X0,Y0,A0                MOVX.W  A1,@R4+                                 ;B0
        PCLR    A1
        PSHA    #1,X0                   MOVX.W  A0,@R5
DCT     PINC    A1,A1
        PAND    X1,Y1,A0                MOVX.W  A1,@R4+                                 ;D0,B0 store
        PCLR    A1
        PSHA    #1,Y1
DCT     PINC    A1,A1                                           MOVY.W  A0,@R7          ;D0 store
                                        MOVX.W  A1,@R4
        MOV.L           @R15+,R4

;*********************************************************
;*部分積計算ﾙｰﾁﾝ/       B0×D0,A0×C0,B0×C0,A0×D0
;*********************************************************
        MOV.L           #WORKX,R5
        MOV.L           #WORKY,R7
                                        MOVX.W  @R5+,X0         MOVY.W  @R7+,Y0         ;A0,C0
        PMULS   X0,Y0,A1                MOVX.W  @R5+,X1         MOVY.W  @R7+,Y1         ;A0×C0,B0,D0
        PMULS   X1,Y1,A0                MOVX.W  A1,@R5+                                 ;B0×D0,  (A0×C0)H store
        PSHA    #16,A1                                          MOVY.W  A0,@R7+         ;(A0×C0)L,(B0×D0)H store
        PSHA    #16,A0                  MOVX.W  A1,@R5+                                 ;(B0×D0)L,(A0×C0)L store
        PMULS   X0,Y1,A1                                        MOVY.W  A0,@R7+         ;A0×D0,  (B0×D0)L store
        PSHA    #16,A1                  MOVX.W  A1,@R5+                                 ;(A0×D0)L,(A0×D0)H store
        PMULS   X1,Y0,A1                MOVX.W  A1,@R5                                  ;B0×C0,  (A0×D0)L store
        PSHA    #16,A1                                          MOVY.W  A1,@R7+         ;(B0×C0)L,(B0×C0)H store
                                                                MOVY.W  A1,@R7          ;(B0×C0)L store

;******************
;*ANSWER1 STORE
;******************
        MOV.L           R7,@-R15                                                        ;push R7
        MOV.L           #ANS,R7
        ADD             #6,R7
                                                                MOVY.W  A0,@R7+         ;ANS1格納
        ADD             #-2,R7
        MOV.L           R7,R14                                                          ;R14=#ANS+2
        MOV.L           @R15+,R7                                                        ;pop R7
```

メインプログラム

```
;*****************************************************************************************************************************************
;*2ﾜｰﾄﾞ計算ﾙｰﾁﾝ/                R4=#XRAM_ADD+2,R5=#WORKX+10,R6=#YRAM_ADD+2,R7=#WORKY+10
;*****************************************************************************************************************************************
;
            PCOPY     X1,M1
            MOV.L     #-6,R9
            PCLR      A1                MOVX.W   @R5,X1          MOVY.W   @R7+R9,Y1      ;(A0×D0)L lode,(B0×C0)L load
            PADD      X1,Y1,A0                                   MOVY.W   @R7+,Y1        ;(A0×D0)L+(B0×C0)L,(B0×D0)H load
      DCT   PINC      A1,A1                                                              ;carry check
            PADD      A0,Y1,A0                                                           ;(A0×D0)L+(B0×C0)L+(B0×D0)H
      DCT   PINC      A1,A1                                                              ;carry check
            MOV.W               #H'0,R10
            MOV.L               #SIGND,R0
            MOV.W               @R0+,R1
            CMP/EQ              R10,R1                                                 ;Bはﾏｲﾅｽ?
            BT                  HOSEI4_L
                                                                 MOVY.W   @R6,Y1        ;D load
            PADD      A0,Y1,A0                                                           ;D加算
       DCT  PINC      A1,A1
HOSEI4_L:
            MOV.W               @R0,R1
            CMP/EQ              R10,R1                                                 ;Dはﾏｲﾅｽ?
            BT                  HOSEI5_L
            PADD      A0,M1,A0                                                           ;B0加算
      DCT   PINC      A1,A1
HOSEI5_L:
            MOV.L               R4,@-R15                                               ;push R4
            MOV.L               #CARRY,R4
                                MOVX.W   A1,@R4                                        ;carry store
            MOV.L               @R15+,R4                                               ;pop R4
;******************
;*ANSWER2 STORE
;******************
;
            MOV.L               R7,@-R15                                               ;push R7
            MOV.L               R14,R7
                                                                 MOVY.W   A0,@R7+       ;ANS2 store
            ADD                 #-2,R7
            MOV.L               R7,R14                                                 ;R14=#ANS+4
            MOV.L               @R15+,R7                                               ;pop R7
;*****************************************************************************************************************************************
;*3ﾜｰﾄﾞ計算ﾙｰﾁﾝ/                R4=#XRAM_ADD+2,R5=#WORKX+10,R6=#YRAM_ADD+2,R7=#WORKY+6
;*****************************************************************************************************************************************
;
            MOV.L     #-4,R8
            PCOPY     X0,A1             MOVX.W   @R5+R8,X0       MOVY.W   @R7+,Y1        ;dummy load
                                        MOVX.W   @R5+,X0         MOVY.W   @R7+,Y1        ;(A0×C0)L lode,(B0×C0)H load
            PADD      X0,Y1,M1          MOVX.W   @R5,X1                                  ;(A0×C0)L+(B0×C0)H,(A0×D0)H load
            DCT PINC  M0,M0                                                            ;carry check
            PADD      X1,M1,A0                                                         ;(A0×C0)L+(B0×C0)H+(A0×D0)H
            DCT PINC  M0,M0                                                            ;carry check
;補正
            MOV.W               #H'0,R10
            MOV.L               #SIGNA,R0
            MOV.W               @R0+,R1
            CMP/EQ              R10,R1                                                 ;Aはﾏｲﾅｽ?
            BT                  HOSEI2_L
            PSUB      A0,Y1,A0                                                         ;D減算(補正2)
      DCT   PDEC      M0,M0
HOSEI2_L:
            MOV.W               @R0+,R1
            CMP/EQ              R10,R1                                                 ;Cはﾏｲﾅｽ?
            BT                  HOSEI3_L
                                MOVX.W   @R4,X1
            PCOPY     X1,M1
            PSUB      A0,M1,A0                                                         ;B減算(補正3)
      DCT   PDEC      M0,M0
```

メインプログラム

```
HOSEI3_L:
            MOV.W               @R0+,R1
            CMP/EQ              R10,R1                                                          ;Bはﾏｲﾅｽ?
            BT                  HOSEI4_H
            PADD        A0,Y0,A0                                                                ;C0減算(補正4)
    DCT     PINC        M0,M0
HOSEI4_H:
            MOV.W               @R0+,R1
            CMP/EQ              R10,R1                                                          ;Dはﾏｲﾅｽ ?
            BT                  HOSEI5_H
            PCOPY       A1,M1
            PADD        A0,M1,A0                                                                ;A0加算(補正5)
    DCT     PINC        M0,M0
HOSEI5_H:
            PCOPY       A0,M1
            MOV.L               #CARRY,R4
                                                MOVX.W  @R4,X1                                  ;carry load
            PADD        X1,M1,A0                                                                ;ｷｬﾘｰを加算
    DCT     PINC        M0,M0                                                                   ;carry check

;**************
;*ANSWER3 STORE
;**************
            MOV.L               R14,R7
                                                                    MOVY.W  A0,@R7+             ;ANS3 store
            ADD                 #-2,R7

;*****************************************************************************************************************************************
;*4ﾜｰﾄﾞ計算ﾙｰﾁﾝ/               R4=#XRAM_ADD+2,R5=#WORKX+8,R6=#YRAM_ADD+2,R7=#WORKY+10
;*****************************************************************************************************************************************
            PCLR        Y1                      MOVX.W  @R5+R8,X1                               ;dummy load
            PCLR        M1                      MOVX.W  @R5,X1                                  ;(A0×C0)H load
            PADD        X1,M0,A0
    DCT     PINC        M1,M1
;補正
            MOV.L               #SIGNA,R0
            MOV.W               @R0+,R1
            CMP/EQ              R10,R1                                                          ;Aはﾏｲﾅｽ?
            BT                  HOSEI3_H
            PCOPY       A1,M0
            PSUB        A0,M0,A0                                                                ;C0減算(補正2)
    DCT     PDEC        M1,M1
            MOV.L               #H'0,R12
            ADD                 #1,R12
HOSEI2_H:
            MOV.W               @R0+,R1
            CMP/EQ              R10,R1                                                          ;Cはﾏｲﾅｽ?
            BT                  HOSEI4_H
            PSUB        A0,Y0,A0                                                                ;A0減算(補正3)
    DCT     PDEC        M1,M1
            ADD                 #1,R12
HOSEI3_H:
            MOV.L               #2,R1
            CMP/EQ              R1,R12                                                          ;A、C共にﾏｲﾅｽ?
            BF                  FIN
            MOV.W               #H'8000,R10
            MOV.W               R10,@R5
                                                MOVX.W  @R5,X0
            PCOPY       X0,M1                                                                   ;H'8000を加算(補正1)
            PADD        A0,M1,A0

;**************
;*ANSWER4 STORE
;**************
FIN:                                                                MOVY.W  A0,@R7              ;ANS4 store
EXIT:       BRA                 EXIT
            NOP
MAIN_E:     NOP
```

```
;*****************************************************************************************************************
;*                        32ﾋﾞｯﾄ乗算用ﾃﾞｰﾀ(X/YRAM)
;*****************************************************************************************************************
                    .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:           .XDATA.L        0.25002500              ;被乗数
WORKX:              .RES.W          6                       ;ﾜｰｸｴﾘｱ
CARRY:              .RES.W          1                       ;ｷｬﾘｰ用ｴﾘｱ
SIGNA:              .RES.W          1                       ;被乗数上位ﾜｰﾄﾞAの符号判定用
SIGNC:              .RES.W          1                       ;乗数上位ﾜｰﾄﾞCの符号判定用
SIGNB:              .RES.W          1                       ;被乗数下位ﾜｰﾄﾞBの符号判定用
SIGND:              .RES.W          1                       ;乗数下位ﾜｰﾄﾞDの符号判定用

                    .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD:           .XDATA.L        0.50005000              ;乗数
WORKY:              .RES.W          6                       ;ﾜｰｸｴﾘｱ
ANS:                .RES.W          4                       ;乗算結果格納ｴﾘｱ
```

8. 三角関数

概要

三角関数SIN(X)、COS(X)を算出します。

説明

1. 三角関数の求め方

図8.1にSIN(X)、COS(X)のカーブを示します。角度の範囲を$-\pi \leqq X \leqq \pi$とすると、(1)式の関係が成り立ちます。

$$\left.\begin{aligned} SIN(-X) &= -SIN(X) \\ COS(-X) &= COS(X) \end{aligned}\right\} \quad \cdots\cdots\cdots (1)$$

(1)式の関係により、$-\pi \leqq X < 0$のSIN(X)、COS(X)は、$0 \leqq X < \pi$のSIN(X)、COS(X)を求めて符号処理を行なうことにより算出できます。

次に**図8.2(a)、(b)**で示す。$X=\pi/2$を中心にしたSIN(X)、COS(X)の関係を(2)式に示します。

$$\left.\begin{aligned} SIN(X+\pi/2) &= -SIN(\pi/2-X) \\ COS(X+\pi/2) &= COS(\pi/2-X) \end{aligned}\right\} \quad \cdots\cdots\cdots (2)$$

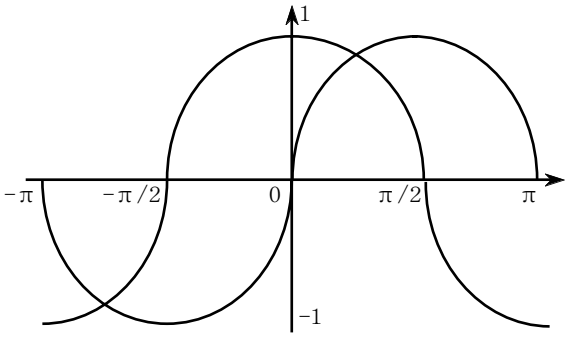
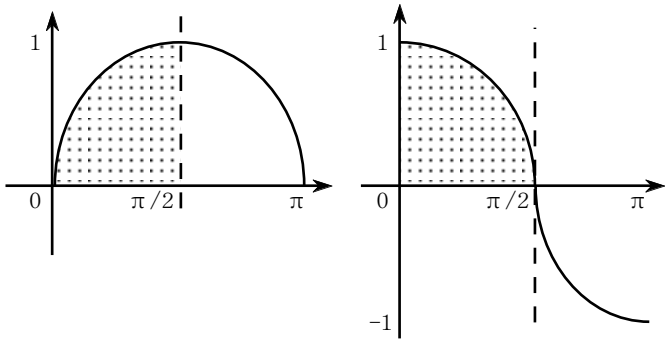


図8.1　SIN(X)、COS(X)のカーブ



(a)　SIN(X)　　(b)　COS(X)

図8.2　π/2を中心としたSIN(X)、COS(X)のカーブ

(1)式、(2)式の関係により、$-\pi \leqq X \leqq \pi$のSIN(X)、COS(X)は、$0 \leqq X \leqq \pi/2$のSIN(X)、COS(X)を求めて、最後に符号処理を行なうことにより算出できることになります。本プログラムでは、$0 \leqq X \leqq \pi/2$を128分割しました。そこで$X=n \cdot \pi/256+\Delta X$(n=1,2,・・・,128)とおけば、三角関数の加法定理により(3)式が成り立ちます。

$$\left.\begin{aligned} SIN(X) &= SIN(n \cdot \pi/256+\Delta X) = SIN(n \cdot \pi/256) \cdot COS(\Delta X) - COS(n \cdot \pi/256) \cdot SIN(\Delta X) \\ COS(X) &= COS(n \cdot \pi/256+\Delta X) = COS(n \cdot \pi/256) \cdot COS(\Delta X) - SIN(n \cdot \pi/256) \cdot SIN(\Delta X) \end{aligned}\right\} \quad \cdots (3)$$

(3)式において、ΔXは非常に小さいと考えて、$SIN(\Delta X)=\Delta X$、$COS(\Delta X)=1-(\Delta X)^2/2$で近似すると(4)式のようになります。

$$\left.\begin{aligned} SIN(X) &= SIN(n\cdot\pi/256)\cdot\{1-(\Delta X)^2/2\}+\Delta X\cdot COS(n\cdot\pi/256) \\ COS(X) &= COS(n\cdot\pi/256)\cdot\{1-(\Delta X)^2/2\}-\Delta X\cdot SIN(n\cdot\pi/256) \end{aligned}\right\} \quad (4)$$

つまりSIN(X)、COS(X)はΔXとテーブルデータ(n・π/256) を用いて(4)式を計算することにより0≦X≦π/2のSIN(X)、COS(X)が求められます。最後に符号処理を行なうことにより最終結果を得られます。

2. 入力値の変換

本プログラムでは、−π≦X＜πの範囲の入力値Xを、(5)式の変換式を用いて、−1≦X＜1の範囲のaを角度のパラメータとしてDSPユニットに入力します。

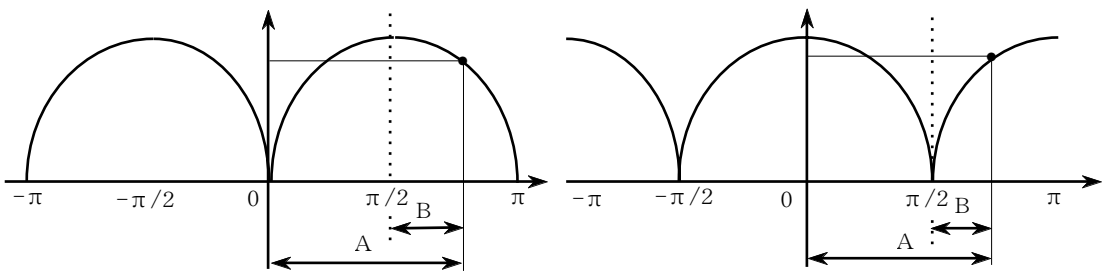
$$\left.\begin{aligned} X &= \pi\cdot a \\ a &= X/\pi \end{aligned}\right\} \quad (5)$$

Xの単位：rad
aの単位 ：rad/π

表8.1　入力値aと極性の関係

入力値＼結果	SIN(X)	COS(X)	｜a｜
−1＜≦a＜−0.5 (−π≦X＜−π/2)	負	負	｜a｜＞0.5
−0.5≦a＜0 (−π/2≦X＜0)	負	正	｜a｜≦0.5
0≦a≦0.5 (0≦X≦π/2)	正	正	｜a｜≦0.5
0.5＜a＜1 (π/2＜X＜π)	正	負	｜a｜＞0.5

ここで、0≦X≦π/2の範囲は、0≦a≦0.5の範囲に対応します。そこで、入力値aを−1＜a≦1の範囲から0≦a'≦0.5の範囲に変換します。**図8.3**に｜SIN(X)｜、｜COS(X)｜のカーブを示します。



(a)　｜SIN(X)｜　　(b)　｜COS(X)｜

図8.3　｜SIN(X)｜、｜COS(X)｜のカーブ

図8.3より、A点のSIN(X)、COS(X)を求める場合、A点をA=π/2+Bとすると、a=0.5+bとなります。つまり、(6)式よりX=π/2に対する偏差｜b｜を求めることができます。

｜b｜ = ｜｜a｜ − 0.5｜ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ (6)

次に偏差｜b｜より、−1＜a≦1の範囲の入力値aを0≦a'≦0.5の変換値a'に変換するには。(7)式を計算します。

a' = ｜｜｜a｜ − 0.5｜ −0.5｜ ‒ ‒ ‒ ‒ ‒ ‒ (7)

3. a'のテーブルデータ

本プログラムでは、128ヶのテーブルを用いています。つまり、0≦a'≦0.5の範囲を128ヶに等分しています。そこで1区間の角度によるa'の差は、(8)式に示すようになります。

0.5/128 = 0.00390625 ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ ‒ (8)

表8.2にテーブルアドレスnとa'の関係を10進表現、16ビット固定小数点表現で示します。

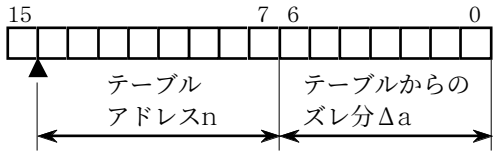
表8.2 テーブルアドレスnとa'の関係

テーブルアドレスn	a'																
	n/256;10進表現 rad]/π	16ビット固定小数点表現															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0.00000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0.00390625	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	0.00781250	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	0.01171875	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0.01562500	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮															
127	0.49609375	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
128	0.50000000	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▲：小数点位置

4. ΔXの算出法

表8.2に示すように固定小数点で表現されたa'のテーブルの上位9ビットがnに対応し、下位7ビットがテーブルデータからのズレ分Δa'となります。**図8.4**にa'のビット構成を示します。a'を求めることにより、(2)式のテーブルデータのアドレス算出(n・π/256の値)およびΔX算出を同時に行なうことができます。最後に**表8.1**を用いて符号処理を行い、−π＜X≦πのSIN(X)、COS(X)を求めます。



▲：小数点位置

図8.4 a'のビット構成

図8.5にテーブル値間からのズレ分ΔXとの関係を示します。また、テーブルのズレ分ΔXはa'のΔaを用いて(9)式より求めることができます。

$$\Delta X = \Delta a \cdot \pi \quad \cdots\cdots\cdots (9)$$

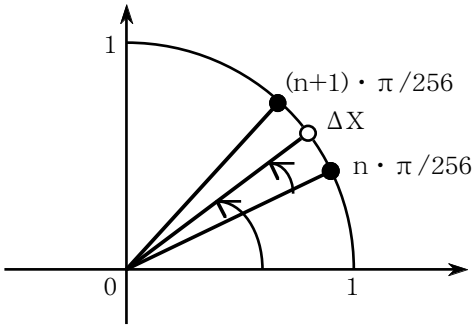


図8.5 テーブル値間からのズレ分ΔXとの関係

5. オーバーフロー時の処理

演算結果が(10)式で示す場合には、オーバーフローが発生したことを示します。

$$\left.\begin{array}{l} |\,SIN(X)\,| \geqq 1 \\ |\,COS(X)\,| < 0 \end{array}\right\} \cdots\cdots\cdots (10)$$

この場合には(11)式で示す値に補正します。

$$\left.\begin{array}{l} |\,SIN(X)\,| = 1-2^{-15} \\ |\,COS(X)\,| = 0 \end{array}\right\} \cdots\cdots (11)$$

説明

6. 三角関数算出アルゴリズム

以下に三角関数算出アルゴリズムを示します。

① 初期設定を行ないます。

② 入力値aをロードし、｜｜｜a｜−0.5｜−0.5｜の計算を行い、a'を算出します。

③ #H'FF80と論理積をとりa'の上位9ビット(n/256)を算出します。さらに、nを算出し、
Yバス用のインデックスレジスタ(R9)へセットします

④ #H'007Fと論理積をとりa'の下位7ビット(Δa')を算出します。

⑤ πΔa'を計算し、ΔXを算出します。

⑥ 1−(ΔX)^2/2を算出します。sin(n×π/256)、cos(n×π/256)をYRAM上のテーブルデータよりロードします。

⑦ sin(X)を算出します。

⑧ sin(X)を符号処理し、sin(X)をストアします。

⑨ cos(X)を算出します。

⑩ cos(X)を符号処理し、cos(X)をストアします。

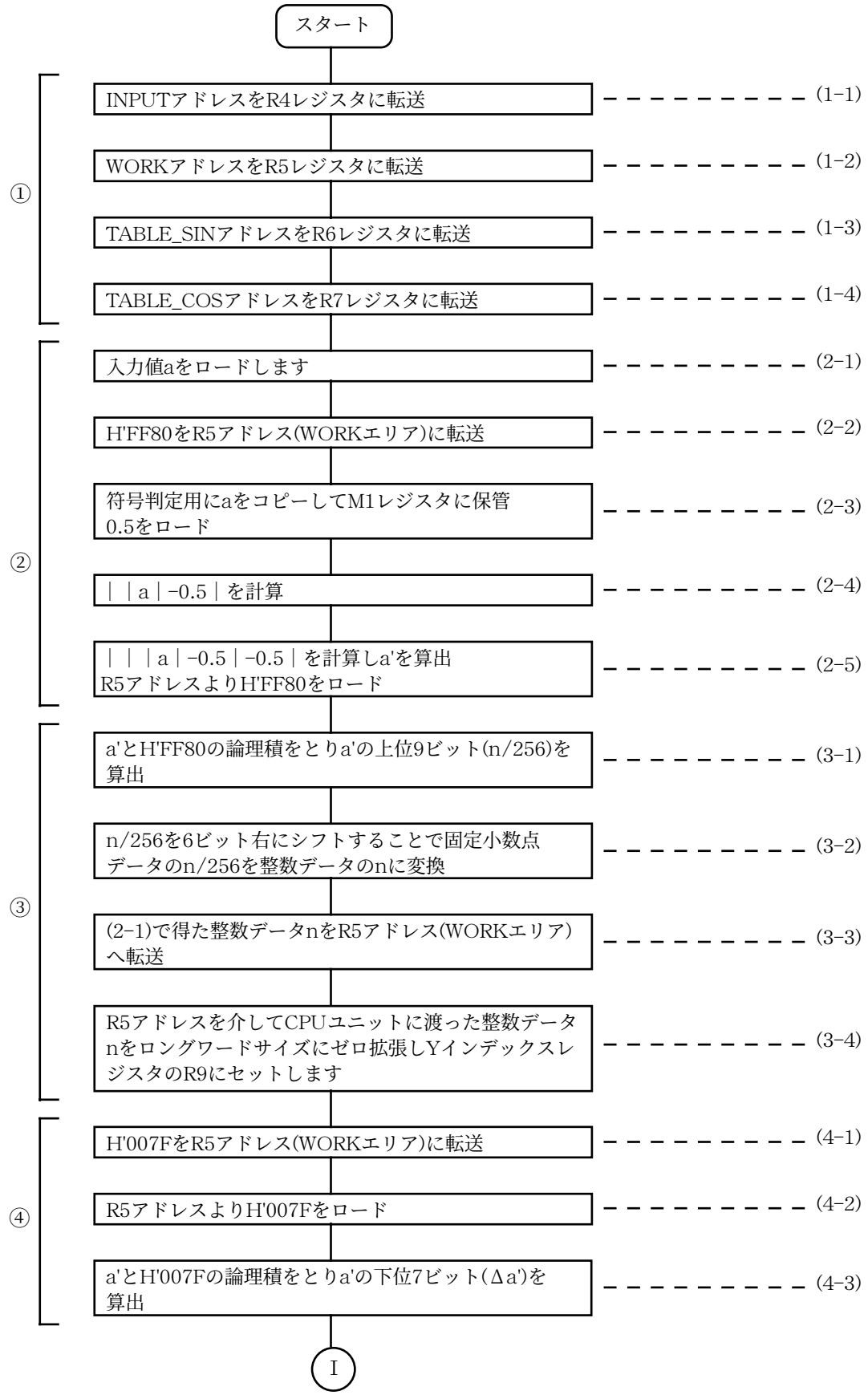
実行例

入力値a(INPUT)によるsin(X)、cos(X)(OUTPUT)の算出結果を**表8.3**に示します。

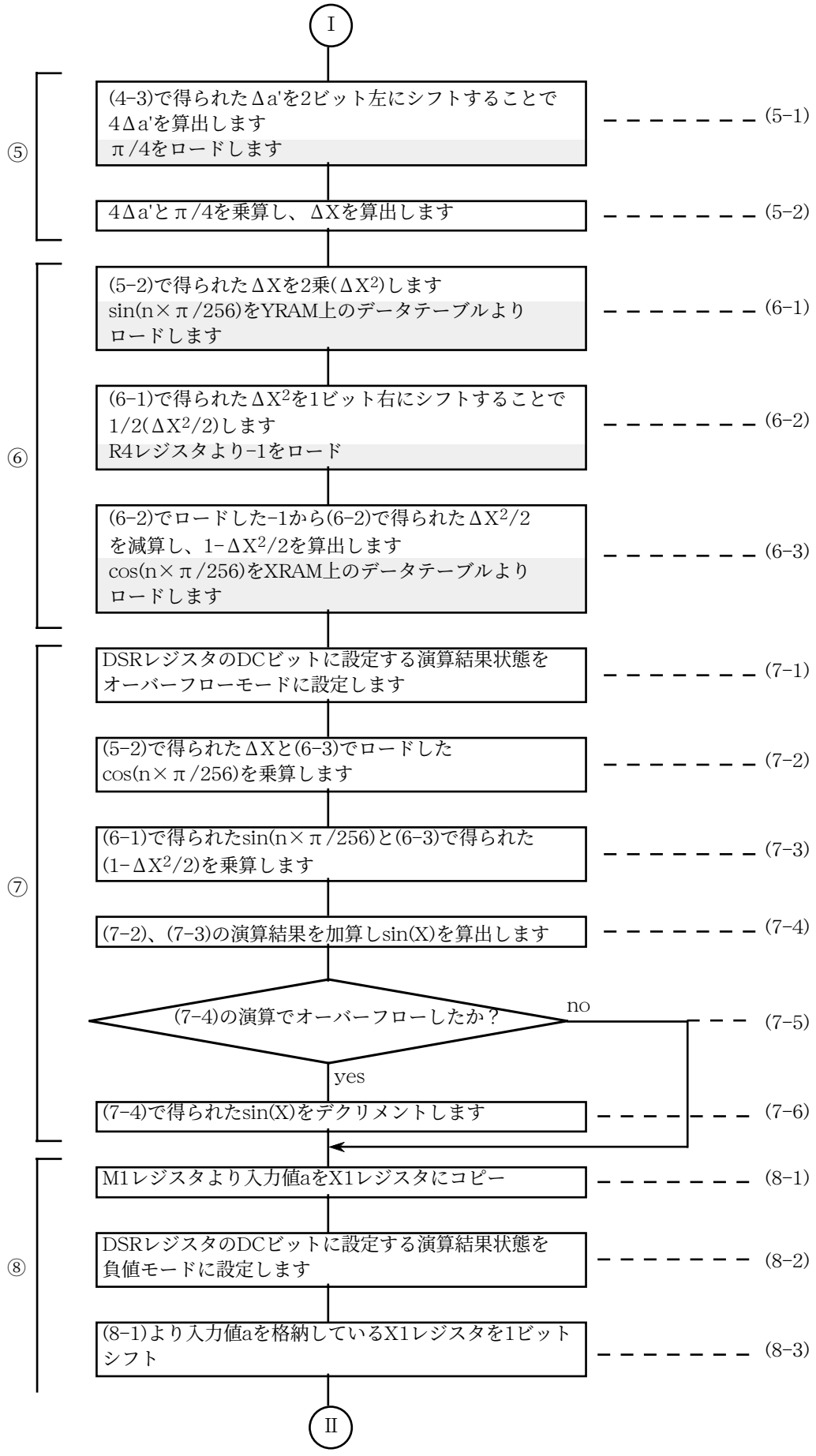
表8.3　sin(X)、cos(X)の算出結果

角度X°	入力値 (a=X/π)	理論値(10進)		理論値(16進)		出力値(16進)	
		sin(X)	cos(X)	sin(X)	cos(X)	sin(X)	cos(X)
0	0	0	1	H'0000	H'7FFF	H'0000	H'7FFF
30	0.16667	0.5	0.86603	H'4000	H'6EDA	H'3FFE	H'6ED9
45	0.25	0.70711	0.70711	H'5A82	H'5A82	H'5A82	H'5A82
89.5	0.49722	0.99996	0.00873	H'7FFE	H'011E	H'7FFD	H'011D
152	0.84444	0.46947	−0.88295	H'3C17	H'8EFC	H'3C19	H'8EFD
179.5	0.99722	0.00873	−0.99996	H'011E	H'8002	H'011C	H'8002
−40	−0.22222	−0.64279	0.76604	H'ADB9	H'620D	H'ADBB	H'620F
−75	−0.41667	−0.96593	0.25882	H'845D	H'2121	H'845D	H'2121
−137	−0.76111	−0.681	−0.73135	H'A8B4	H'A263	H'A8B5	H'A263
−180	−1	0	−1	H'0000	H'8000	H'0002	H'8001

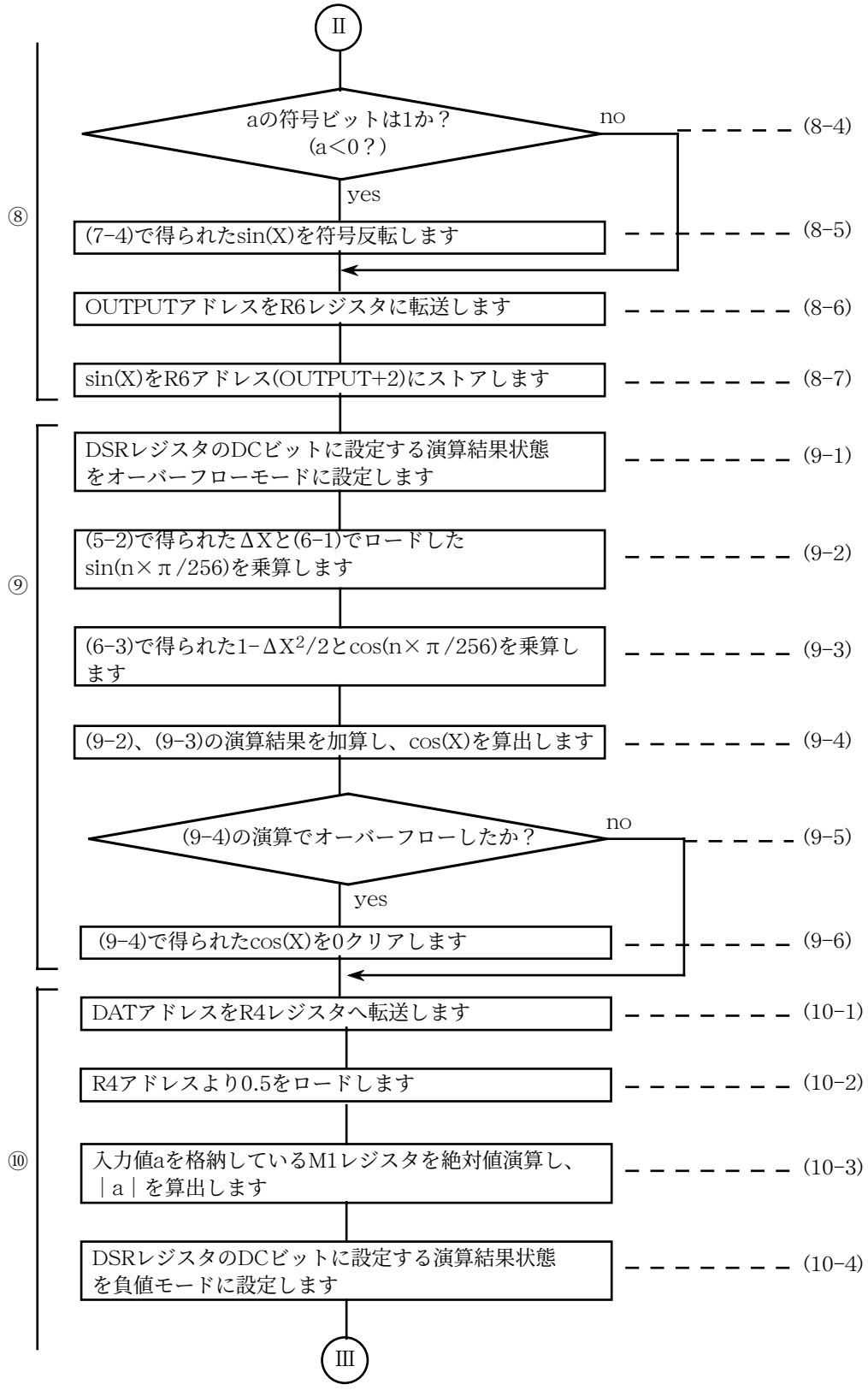
フローチャート


スタート
INPUTアドレスをR4レジスタに転送
(1-1)
WORKアドレスをR5レジスタに転送
(1-2)
TABLE_SINアドレスをR6レジスタに転送
(1-3)
TABLE_COSアドレスをR7レジスタに転送
(1-4)
①
入力値aをロードします
(2-1)
H'FF80をR5アドレス(WORKエリア)に転送
(2-2)
符号判定用にaをコピーしてM1レジスタに保管
0.5をロード
(2-3)
||a|-0.5|を計算
(2-4)
|||a|-0.5|-0.5|を計算しa'を算出
R5アドレスよりH'FF80をロード
(2-5)
②
a'とH'FF80の論理積をとりa'の上位9ビット(n/256)を算出
(3-1)
n/256を6ビット右にシフトすることで固定小数点データのn/256を整数データのnに変換
(3-2)
(2-1)で得た整数データnをR5アドレス(WORKエリア)へ転送
(3-3)
R5アドレスを介してCPUユニットに渡った整数データnをロングワードサイズにゼロ拡張しYインデックスレジスタのR9にセットします
(3-4)
③
H'007FをR5アドレス(WORKエリア)に転送
(4-1)
R5アドレスよりH'007Fをロード
(4-2)
a'とH'007Fの論理積をとりa'の下位7ビット(Δa')を算出
(4-3)
④
I

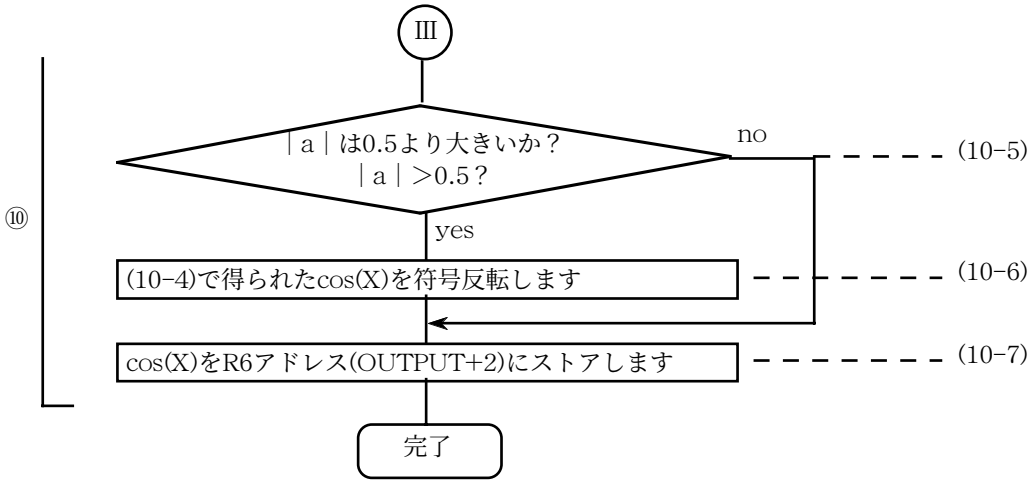
フローチャート


I
⑤
(4-3)で得られたΔa'を2ビット左にシフトすることで4Δa'を算出します
π/4をロードします
(5-1)
4Δa'とπ/4を乗算し、ΔXを算出します
(5-2)
⑥
(5-2)で得られたΔXを2乗(ΔX2)します
sin(n×π/256)をYRAM上のデータテーブルよりロードします
(6-1)
(6-1)で得られたΔX2を1ビット右にシフトすることで1/2(ΔX2/2)します
R4レジスタより-1をロード
(6-2)
(6-2)でロードした-1から(6-2)で得られたΔX2/2を減算し、1-ΔX2/2を算出します
cos(n×π/256)をXRAM上のデータテーブルよりロードします
(6-3)
⑦
DSRレジスタのDCビットに設定する演算結果状態をオーバーフローモードに設定します
(7-1)
(5-2)で得られたΔXと(6-3)でロードしたcos(n×π/256)を乗算します
(7-2)
(6-1)で得られたsin(n×π/256)と(6-3)で得られた(1-ΔX2/2)を乗算します
(7-3)
(7-2)、(7-3)の演算結果を加算しsin(X)を算出します
(7-4)
(7-4)の演算でオーバーフローしたか？
no
(7-5)
yes
(7-4)で得られたsin(X)をデクリメントします
(7-6)
⑧
M1レジスタより入力値aをX1レジスタにコピー
(8-1)
DSRレジスタのDCビットに設定する演算結果状態を負値モードに設定します
(8-2)
(8-1)より入力値aを格納しているX1レジスタを1ビットシフト
(8-3)
II

フローチャート


Ⅱ
aの符号ビットは1か？
(a＜0？)
no
(8-4)
yes
⑧
(7-4)で得られたsin(X)を符号反転します
(8-5)
OUTPUTアドレスをR6レジスタに転送します
(8-6)
sin(X)をR6アドレス(OUTPUT+2)にストアします
(8-7)
DSRレジスタのDCビットに設定する演算結果状態
をオーバーフローモードに設定します
(9-1)
(5-2)で得られたΔXと(6-1)でロードした
sin(n×π/256)を乗算します
(9-2)
⑨
(6-3)で得られた1-ΔX2/2とcos(n×π/256)を乗算し
ます
(9-3)
(9-2)、(9-3)の演算結果を加算し、cos(X)を算出します
(9-4)
(9-4)の演算でオーバーフローしたか？
no
(9-5)
yes
(9-4)で得られたcos(X)を0クリアします
(9-6)
DATアドレスをR4レジスタへ転送します
(10-1)
R4アドレスより0.5をロードします
(10-2)
⑩
入力値aを格納しているM1レジスタを絶対値演算し、
｜a｜を算出します
(10-3)
DSRレジスタのDCビットに設定する演算結果状態
を負値モードに設定します
(10-4)
Ⅲ

フローチャート


Ⅲ
｜a｜は0.5より大きいか？
｜a｜＞0.5？
no
(10-5)
⑩
yes
(10-4)で得られたcos(X)を符号反転します
(10-6)
cos(X)をR6アドレス(OUTPUT+2)にストアします
(10-7)
完了

メインプログラム

```
;*********************************************************************************************************************************************
;*                              三角関数ﾙｰﾁﾝ
;*
;*                              sinX,cosX
;*
;*********************************************************************************************************************************************
;*********************************************************************************************************************************************
;*                              初期設定ﾙｰﾁﾝ
;*********************************************************************************************************************************************
MAIN:
        MOV.L           #INPUT,R4
        MOV.L           #WORK,R5
        MOV.L           #TABLE_SIN,R6
        MOV.L           #TABLE_COS,R7

;*********************************************************************************************************************************************
;*                              a'算出ﾙｰﾁﾝ
;*********************************************************************************************************************************************
                                        MOVX.W  @R4,X0                  ;a load
        MOV.L           #H'FF80,R0                                      ;a'上位9ﾋﾞｯﾄ(N×π/64)抽出用
        MOV.W           R0,@R5
        MOV.L           #DAT,R4
        PCOPY   X0,M1                   MOVX.W  @R4+,X1                 ;M1符号判定用,0.5 load
        PCOPY   X1,Y1
        PSUB    X0,Y1,M0
        PABS    M0,A0                                                   ;｜｜a｜- 0.5｜
        PSUB    A0,Y1,M0                                                ;｜｜｜a｜- 0.5｜- 0.5｜
        PABS    M0,M0                   MOVX.W  @R5,X0                  ;M0 = a',#H'FF80 load
;*********************************************************************************************************************************************
;*                              n算出,R9ｾｯﾄﾙｰﾁﾝ
;*********************************************************************************************************************************************
        PAND    X0,M0,A0                                                ;A1 = n/256
        PSHA    #-6,A0                                                  ;固定小数点nから整数nへ変換
                                        MOVX.W  A0,@R5                  ;整数nをCPUﾕﾆｯﾄに渡す
        MOV.W           @R5,R1
        EXTU.W          R1,R1                                           ;ｾﾞﾛ拡張しﾛﾝｸﾞﾜｰﾄﾞ長のnに変換
        MOV.L           R1,R9                                           ;Yﾊﾞｽ用のｲﾝﾃﾞｯｸｽﾚｼﾞｽﾀにｾｯﾄ

;*********************************************************************************************************************************************
;*                              Δa'算出ﾙｰﾁﾝ
;*********************************************************************************************************************************************
        MOV.L           #H'007F,R0                                      ;a'下位7ﾋﾞｯﾄ(Δa')抽出用
        MOV.W           R0,@R5
                                        MOVX.W  @R5,X1                  ;#H'007F load
        PAND    X1,M0,Y1                                                ;Δa'
;*********************************************************************************************************************************************
;*                              ΔX算出ﾙｰﾁﾝ
;*********************************************************************************************************************************************
        PSHA    #2,Y1                   MOVX.W  @R4+,X1                 ;4Δa',π/4 load
        PMULS   X1,Y1,A1                                                ;Δa'×π
;*********************************************************************************************************************************************
;*                              1-(ΔX²)/2算出、sin(n×π/256)、cos(n×π/256)ﾛｰﾄﾞﾙｰﾁﾝ
;*********************************************************************************************************************************************
        PCOPY   A1,X0                                   MOVY.W  @R6+R9,Y0       ;copy,dummy load
        PMULS   A1,X0,M0                                MOVY.W  @R6,Y0          ;ΔX²,sin(n×π/256) load
        PSHA    #-1,M0                  MOVX.W  @R4,X1  MOVY.W  @R7+R9,Y1       ;ΔX²/2, -1 lode,dummy load
        PSUB    X1,M0,A1                                MOVY.W  @R7,Y1          ;1-ΔX²/2,cos(n×π/256) load

;*********************************************************************************************************************************************
;*                              sin(X)算出ﾙｰﾁﾝ
;*********************************************************************************************************************************************
        MOV.L           #H'6,R0
        LDS             R0,DSR                                          ;ｵｰﾊﾞｰﾌﾛｰﾓｰﾄﾞに設定
        PMULS   X0,Y1,M0                                                ;ΔX・cos(n×π/256)
        PMULS   A1,Y0,A0                                                ;(1-(ΔX²)/2)・sin(n×π/256)
        PABS    A0,A0
        PADD    A0,M0,A0                                                ;A0 = sin(X)
    DCT PDEC    A0,A0                                                   ;ｵｰﾊﾞｰﾌﾛｰ発生時、sin(X)-1
;*********************************************************************************************************************************************
;*                              sin(X)符号処理、ｽﾄｱﾙｰﾁﾝ
;*********************************************************************************************************************************************
        PCOPY   M1,X1
        MOV.L           #H'0,R0,
        LDS             R0,DSR                                          ;ｷｬﾘｰ/ﾎﾞﾛｰﾓｰﾄﾞ
        PSHA    #1,X1
    DCT PNEG    A0,A0                                                   ;a＜0なら符号反転
        MOV.L           #OUTPUT,R6
                                                        MOVY.W  A0,@R6+ ;sin(X)ｽﾄｱ
```

メインプログラム

```
;*****************************************************************************************************************************************************
;*                              cos(X)算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
;
            MOV.L               #H'6,R0
            LDS                 R0,DSR                                                          ;ｵｰﾊﾞｰﾌﾛｰﾓｰﾄﾞに設定
            PMULS     X0,Y0,M0                                                              ;ΔX・SIN(N×π/64)
            PMULS     A1,Y1,A0                                                              ;(1-(ΔX・ΔX)/2)・COS(N×π/64)
            PABS      A0,A0
            PSUB      A0,M0,A0
      DCT   PCLR      A0                                                                    ;ｵｰﾊﾞｰﾌﾛｰ発生時、cos(X)を0ｸﾘｱ
;;****************************************************************************************************************************************************
;*                              cos(X)符号処理、ｽﾄｱﾙｰﾁﾝ
;*****************************************************************************************************************************************************
;
            MOV.L               #DAT,R4
                                          MOVX.W    @R4.X0                                  ;0.5 load
            PABS      M1,M1                                                                 ;｜a｜
            MOV.L               #H'2,R0
            LDS                 R0,DSR                                                      ;負値ﾓｰﾄﾞに設定
            PCMP      X0,M1
        DCT PNEG      A0,A0                                                                 ;｜a｜ ＞0.5なら符号反転
                                                    MOVY.W    A0,@R6

EXIT:       BRA       EXIT
            NOP
MAIN_E:     NOP
```

データ

```
;*****************************************************************************************************************************************************
;*                              三角関数用ﾃｰﾌﾞﾙﾃﾞｰﾀ(X/YRAM)
;*****************************************************************************************************************************************************
                .SECTION XRAM,DATA,LOCATE=H'1000FF00
INPUT:          .RES.W      1                                   ;外部入力ﾃﾞｰﾀ格納ｴﾘｱ **  X/180°  **
WORK:           .RES.W      1
DAT:            .XDATA.W    0.5,0.78540,-1                      ;a'算出用,π/4,(1-ΔX2/2)算出用

                .SECTION YRAM,DATA,LOCATE=H'1001F800
TABLE_SIN:      .XDATA.W    0,0.01227,0.02454,0.03681,0.04907,0.06132       ;N/0 - 5
                .XDATA.W    0.07356,0.08580,0.09802,0.11022,0.12241         ;N/6 - 10
                .XDATA.W    0.13458,0.14673,0.15886,0.17096,0.18304         ;N/11 - 15
                .XDATA.W    0.19509,0.20711,0.21910,0.23106,0.24298         ;N/16 - 20
                .XDATA.W    0.25487,0.26671,0.27852,0.29028,0.30201         ;N/21 - 25
                .XDATA.W    0.31368,0.32531,0.33689,0.34842,0.35990         ;N/26 - 30
                .XDATA.W    0.37132,0.38268,0.39400,0.40524,0.41643         ;N/31 - 35
                .XDATA.W    0.42756,0.43862,0.44961,0.46054,0.47140         ;N/36 - 40
                .XDATA.W    0.48218,0.49290,0.50354,0.51410,0.52459         ;N/41 - 45
                .XDATA.W    0.53500,0.54532,0.55557,0.56573,0.57581         ;N/46 - 50
                .XDATA.W    0.58580,0.59570,0.60551,0.61523,0.62486         ;N/51 - 55
                .XDATA.W    0.63439,0.64383,0.65317,0.66242,0.67156         ;N/56 - 60
                .XDATA.W    0.68060,0.68954,0.69838,0.70711,0.71573         ;N/61 - 65
                .XDATA.W    0.72425,0.73265,0.74095,0.74914,0.75721         ;N/66 - 70
                .XDATA.W    0.76517,0.77301,0.78074,0.78835,0.76584         ;N/71 - 75
                .XDATA.W    0.80321,0.81046,0.81758,0.82459,0.83147         ;N/76 - 80
                .XDATA.W    0.83822,0.84485,0.85136,0.85773,0.86397         ;N/81 - 85
                .XDATA.W    0.87009,0.87607,0.88192,0.88764,0.89322         ;N/86 - 90
                .XDATA.W    0.89867,0.90399,0.90917,0.91421,0.91911         ;N/91 - 95
                .XDATA.W    0.92388,0.92851,0.93299,0.93734,0.94154         ;N/96 - 100
                .XDATA.W    0.94561,0.94953,0.95331,0.95694,0.96043         ;N/101 - 105
                .XDATA.W    0.96378,0.96700,0.97003,0.97294,0.97570         ;N/106 - 110
                .XDATA.W    0.97832,0.98079,0.98311,0.98528,0.98730         ;N/111 - 115
                .XDATA.W    0.98918,0.99090,0.99248,0.99391,0.99518         ;N/116 - 120
                .XDATA.W    0.99631,0.99729,0.99812,0.99880,0.99932         ;N/121 - 125
                .XDATA.W    0.99970,0.99992,1                               ;N/126 - 128

TABLE_COS:      .XDATA.W    1,0.99992,0.99970,0.99932,0.99880,0.99812       ;N/0 - 5
                .XDATA.W    0.99729,0.99631,0.99518,0.99391,0.99248         ;N/6 - 10
                .XDATA.W    0.99090,0.98918,0.98730,0.98528,0.98311         ;N/11 - 15
                .XDATA.W    0.98079,0.97832,0.97570,0.97294,0.97003         ;N/16 - 20
                .XDATA.W    0.96700,0.96378,0.96043,0.95694,0.95331         ;N/21 - 25
                .XDATA.W    0.94953,0.94561,0.94154,0.93734,0.93299         ;N/26 - 30
                .XDATA.W    0.92851,0.92388,0.91911,0.91421,0.90917         ;N/31 - 35
                .XDATA.W    0.90399,0.89867,0.89322,0.88764,0.88192         ;N/36 - 40
                .XDATA.W    0.87607,0.87009,0.86397,0.85773,0.85136         ;N/41 - 45
                .XDATA.W    0.84485,0.83822,0.83147,0.82459,0.81758         ;N/46 - 50
                .XDATA.W    0.81046,0.80321,0.76584,0.78835,0.78074         ;N/51 - 55
                .XDATA.W    0.77301,0.76517,0.75721,0.74914,0.74095         ;N/56 - 60
                .XDATA.W    0.73265,0.72425,0.71573,0.70711,0.69838         ;N/61 - 65
                .XDATA.W    0.68954,0.68060,0.67156,0.66242,0.65317         ;N/66 - 70
                .XDATA.W    0.64383,0.63439,0.62486,0.61523,0.60551         ;N/71 - 75
                .XDATA.W    0.59570,0.58580,0.57581,0.56573,0.55557         ;N/76 - 80
                .XDATA.W    0.54532,0.53500,0.52459,0.51410,0.50354         ;N/81 - 85
                .XDATA.W    0.49290,0.48218,0.47140,0.46054,0.44961         ;N/86 - 90
                .XDATA.W    0.43862,0.42756,0.41643,0.40524,0.39400         ;N/91 - 95
                .XDATA.W    0.38268,0.37132,0.35990,0.34842,0.33689         ;N/96 - 100
                .XDATA.W    0.32531,0.31368,0.30201,0.29028,0.27852         ;N/101 - 105
                .XDATA.W    0.26671,0.25487,0.24298,0.23106,0.21910         ;N/106 - 110
                .XDATA.W    0.20711,0.19509,0.18304,0.17096,0.15886         ;N/111 - 115
                .XDATA.W    0.14673,0.13458,0.12241,0.11022,0.09802         ;N/116 - 120
                .XDATA.W    0.08580,0.07356,0.06132,0.04907,0.03681         ;N/121 - 125
                .XDATA.W    0.02454,0.01227,0                               ;N/126 - 128

OUTPUT:         .RES.W      2                                   ;外部出力ﾃﾞｰﾀ格納ｴﾘｱ
```

9. 行列演算

概要

行列A(3,3)と行列B(3,3)を乗算して、32ビット精度の行列積C(3,3)を算出します。行列A,Bは、あらかじめXRAM,YRAM上に設定しておきます。行列積CはYRAMのH'1001FF00番地より格納します。

説明

1. 行列の表し方

図9.1に行列A(n,m)を示します。配列された$a_{i,j}$は行列Aの成分です。成分の横並びを行と呼び、上から順に、1行、2行、3行、……、i行、……と呼びます。また、成分の縦の並びを列と呼び、左から順に1列、2列、3列、……、j列、……と呼びます。i行とj列の交点の位置にある成分を、(i,j)成分と呼びます。行列A(n,m)の(i,j)成分を$a_{i,j}$と表します。

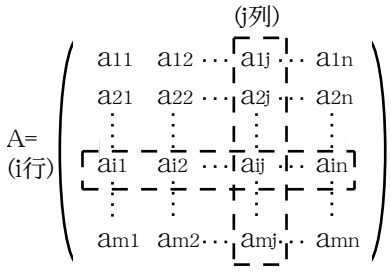


図9.1　行列A

2. 行列積の求め方

図9.2に行列A×行列B=行列積Cの成分表現を示します。

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} {}^{*1}c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

行列A　　行列B　　行列積C

*1　$c_{i,j}$：は32ビット成分です。

図9.2　行列A×行列B=行列積Cの成分表現

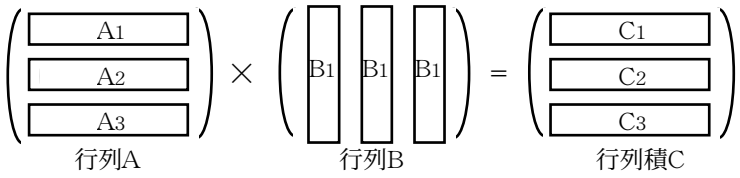
ここで行列積Cの成分$c_{i,j}$は、以下の式で求めることができます。

$$c_{n,m}=\sum_{i=1}^{3}(a_{n,i}\times b_{i,m})$$

上記の式より、行列積Cの成分$c_{i,j}$は行列Aの行成分$a_{n,i}$と行列Bの列成分$b_{i,m}$との積和演算を行なうことにより算出できます。

3.　行列A、行列B、行列積Cの成分格納方法

行列積Cの成分$c_{n,m}$は、行列Aの行成分$a_{n,j}$と行列Bの列成分$b_{i,m}$との積和演算を行なうことにより算出できます。そこで、本サブルーチンでは処理速度の向上のために、あらかじめ**図9.3**にしめすように、X/YRAMへ成分を格納します。



［アドレス］　XRAM

#MATRIXA	$a_{1,1}$	A1
#MATRIXA+2	$a_{1,2}$	
#MATRIXA+4	$a_{1,3}$	
#MATRIXA+6	$a_{2,1}$	A2
#MATRIXA+8	$a_{2,2}$	
#MATRIXA+A	$a_{2,3}$	
#MATRIXA+C	$a_{3,1}$	A3
#MATRIXA+E	$a_{3,2}$	
#MATRIXA+10	$a_{3,3}$	

［アドレス］　YRAM

#MATRIXB	$b_{1,1}$	B1
#MATRIXB+2	$b_{2,1}$	
#MATRIXB+4	$b_{3,1}$	
#MATRIXB+6	$b_{1,2}$	B2
#MATRIXB+8	$b_{2,2}$	
#MATRIXB+A	$b_{3,2}$	
#MATRIXB+C	$b_{1,3}$	B3
#MATRIXB+E	$b_{2,3}$	
#MATRIXB+10	$b_{3,3}$	

［アドレス］　*2　YRAM

#MATRIXC	$CH_{1,1}$	C1
#MATRIXC+2	$CL_{1,1}$	
#MATRIXC+4	$CH_{1,2}$	
#MATRIXC+6	$CL_{1,2}$	
#MATRIXC+8	$CH_{1,3}$	
#MATRIXC+A	$CL_{1,3}$	
#MATRIXC+C	$CH_{2,1}$	C2
#MATRIXC+E	$CL_{2,1}$	
#MATRIXC+10	$CH_{2,2}$	
#MATRIXC+12	$CL_{2,2}$	
#MATRIXC+14	$CH_{2,3}$	
#MATRIXC+16	$CL_{2,3}$	
#MATRIXC+18	$CH_{3,1}$	C3
#MATRIXC+1A	$CL_{3,1}$	
#MATRIXC+1C	$CH_{3,2}$	
#MATRIXC+1E	$CL_{3,2}$	
#MATRIXC+20	$CH_{3,3}$	
#MATRIXC+22	$CL_{3,3}$	

*2　$CH_{i,j}$：$c_{i,j}$の上位16ビット
　　$CL_{i,j}$：$c_{i,j}$の下位16ビット

図9.3　行列A、行列B、行列積Cの成分を格納したメモリマップ

説明

4. 行列積の算出アルゴリズム

図9.4に行列積Cの算出アルゴリズムを示します。以下に詳細アルゴリズムを説明します。

① カウンタ用レジスタのクリアと、Xアドレスレジスタ(R4)、Yアドレスレジスタ(R6,R7)に行列A、行列Bの成分が格納されているアドレスと、行列積Cの成分を格納するアドレスを設定する。

② 行列Aの行成分$a_{n,i}$と行列Bの列成分$b_{i,m}$の積和演算を行ないます。

③ 行列積$c_{n,m}$の上位16ビット$c_{Hn,m}$をMATRIXC+2nアドレスへ格納し下位16ビット$c_{Ln,m}$をMATRIXC+2n+2アドレスに格納します。

④ 行列Aの列成分を1列目に戻します。

⑤ 行列積$c_{n,m}$を1行分算出したかを判定します。nが3でない場合には、②の処理へ戻ります。nが3の場合には、⑥の処理へ移行します。

⑥ 行列Aの行成分を1行下にシフトします。

⑦ 行列積Cを3行まで全て算出したかを判定します。nが3でない場合には、②の処理へ戻ります。nが3の場合には、全ての行列積$c_{n,m}$を算出しているので処理を終了します。

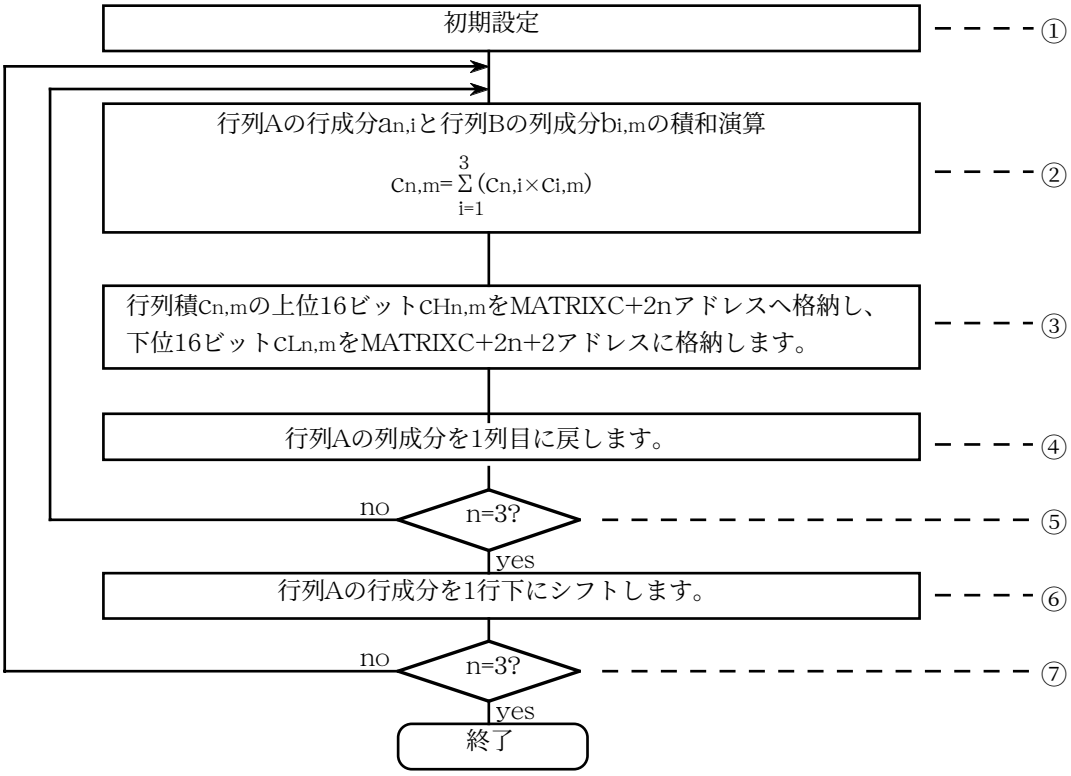
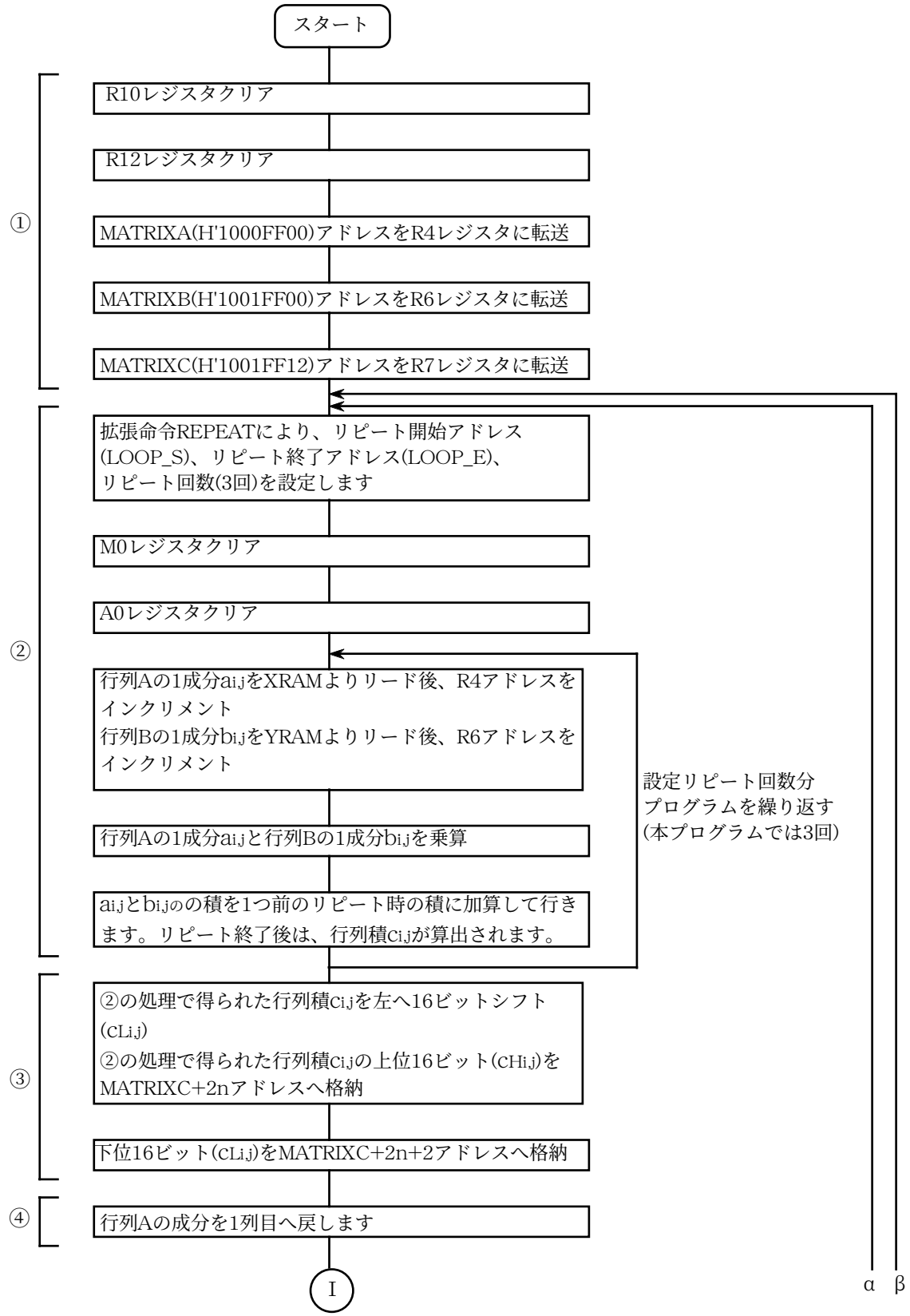
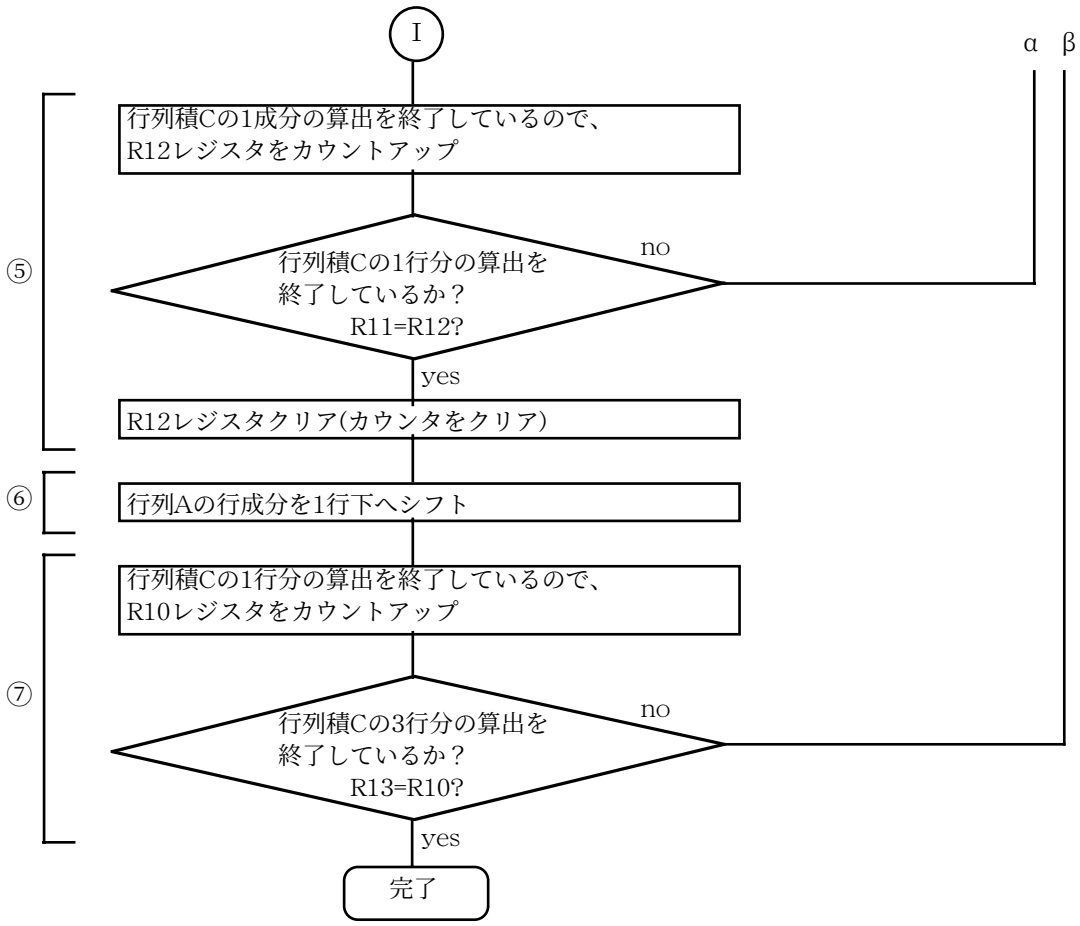


図9.4 行列積算出アルゴリズム

フローチャート


スタート
R10レジスタクリア
R12レジスタクリア
①
MATRIXA(H'1000FF00)アドレスをR4レジスタに転送
MATRIXB(H'1001FF00)アドレスをR6レジスタに転送
MATRIXC(H'1001FF12)アドレスをR7レジスタに転送
拡張命令REPEATにより、リピート開始アドレス(LOOP_S)、リピート終了アドレス(LOOP_E)、リピート回数(3回)を設定します
M0レジスタクリア
A0レジスタクリア
②
行列Aの1成分aᵢ,ⱼをXRAMよりリード後、R4アドレスをインクリメント
行列Bの1成分bᵢ,ⱼをYRAMよりリード後、R6アドレスをインクリメント
設定リピート回数分プログラムを繰り返す(本プログラムでは3回)
行列Aの1成分aᵢ,ⱼと行列Bの1成分bᵢ,ⱼを乗算
aᵢ,ⱼとbᵢ,ⱼのの積を1つ前のリピート時の積に加算して行きます。リピート終了後は、行列積cᵢ,ⱼが算出されます。
②の処理で得られた行列積cᵢ,ⱼを左へ16ビットシフト(cLᵢ,ⱼ)
②の処理で得られた行列積cᵢ,ⱼの上位16ビット(cHᵢ,ⱼ)をMATRIXC+2nアドレスへ格納
③
下位16ビット(cLᵢ,ⱼ)をMATRIXC+2n+2アドレスへ格納
④
行列Aの成分を1列目へ戻します
I
α
β

フローチャート


I
α
β
行列積Cの1成分の算出を終了しているので、
R12レジスタをカウントアップ
⑤
行列積Cの1行分の算出を
終了しているか？
R11=R12?
no
yes
R12レジスタクリア(カウンタをクリア)
⑥
行列Aの行成分を1行下へシフト
行列積Cの1行分の算出を終了しているので、
R10レジスタをカウントアップ
⑦
行列積Cの3行分の算出を
終了しているか？
R13=R10?
no
yes
完了

メインプログラム

matrix.src

```
;*****************************************************************************************************************************************
;*                              行列演算ﾙｰﾁﾝ
;*
;*                              [A][B]=[C]
;*
;*****************************************************************************************************************************************
MAIN:   MOV.L           #0,R10
        MOV.L           #0,R12
        MOV.L           #MATRIXA,R4
        MOV.L           #MATRIXB,R6
        MOV.L           #MATRIXC,R7
;***************************************
;全成分成分算出/R10,R13
;***************************************
        MOV.L           #3,R13                                          ;ﾘﾋﾟｰﾄ値設定(行数)
MATORIX:
;*********************************
;n行目の列成分算出/R11,R12
;*********************************
        MOV.L           #3,R11                                          ;ﾘﾋﾟｰﾄ値設定(列数)
RETSU:
;*****************************
;１成分算出/DSP REPEAT
;*****************************
        BSR             SEIBUN
        NOP
        BSR             STORE
        NOP
;*****************************
        ADD             #-6,R4                                          ;行列Aのi行1列目ｱﾄﾞﾚｽを戻す
        ADD             #1,R12                                          ;行列積Cの1行中の1成分を算出する毎にｶｳﾝﾄｱｯﾌﾟ
        CMP/EQ          R11,R12                                         ;行列積Cの1行分の積和演算が終了したか？
        BF              RETSU
        MOV.L           #0,R12                                          ;ｶｳﾝﾀｸﾘｱ
;*********************************
        ADD             #6,R4
        MOV.L           #MATRIXB,R6
        ADD             #1,R10                                          ;行列積Cの1行分の積和演算が終了したらｶｳﾝﾄｱｯﾌﾟ
        CMP/EQ          R13,R10                                         ;行列積Cの最終行まで積和演算は終了したか？
        BF              MATORIX
;*****************************************

EXIT:   BRA             EXIT
        NOP

;*****************************************************************************************************************************************
;行列Ｃの１成分算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************
SEIBUN:
        REPEAT  LOOP_S,LOOP_E,#3                                        ;行列[A]の行数がﾘﾋﾟｰﾄ回数
        PCLR    M0                                                      ;ﾘﾋﾟｰﾄ用にｸﾘｱ
        PCLR    A0
LOOP_S:
                                        MOVX.W  @R4+,X0  MOVY.W  @R6+,Y0  ;aij,bij load
        PMULS   X0,Y0,M0
LOOP_E: PADD    A0,M0,A0
        RTS
        NOP
;*****************************************************************************************************************************************
;行列Ｃの１成分格納ﾙｰﾁﾝ
;*****************************************************************************************************************************************
STORE:  PSHA    #16,A0                                  MOVY.W  A0,@R7+   ;cij上位格納
                                                        MOVY.W  A0,@R7+   ;cij下位格納
        RTS
        NOP
;**********************
MAIN_E: NOP
```

```
*******************************************************************************
;*                      行列演算用ﾃﾞｰﾀ(X/YRAM)
;*******************************************************************************
                        .SECTION XRAM,DATA,LOCATE=H'1000FF00
MATRIXA:                . XDATA.W               0.5,0.125,0.5,0.125,0.5,0.125,0.5,0.125,0.5

                        .SECTION YRAM,DATA,LOCATE=H'1001FF00
MATRIXB:                .RES.W                  0.25,0.0625,0.25,0.0625,0.25,0.0625,0.25,0.0625,0.25
MATRIXC:                .RES.W                  18
```

概要

2つの零でないn次元空間ベクトルa(16ビット成分)、b(16ビット成分)の内積(32ビット精度)を算出します。n次元空間ベクトルa、bはあらかじめXRAM、YRAM上に設定しておきます。a、bの内積はYRAMのH'1001FF00番地に格納します。

説明

1. 空間ベクトルの表し方

図10.1にn次元の空間ベクトルaの成分表現を示します。n次元の空間ベクトルはn個の実数の組をベクトルと考えます。ベクトルの成分表現には行ベクトルと列ベクトルの2通りの表現方法があります。

$$\overset{*1}{[a_1, a_2, \cdots, a_n]}$$

(a) 行ベクトル

$$\overset{*1}{\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}}$$

(b) 列ベクトル

*1 ai：16ビット

図10.1 n次元の空間ベクトルaの成分表現

2. 内積の求め方

図10.2にn次元空間ベクトルa、bの内積の成分表現を示します。ここで、ベクトルa、bの内積を(a,b)表します。

$$\overset{*1}{[a_1, a_2, \cdots, a_i, \cdots, a_n]} \quad \wedge \quad \overset{*1}{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}} = \overset{*2}{a_1b_1 + a_2b_2 + \cdots + a_ib_i + \cdots + a_nb_n}$$

n次元空間
行ベクトルa

n次元空間
列ベクトルb

*1 ai：16ビット
bi：16ビット
*2 32ビット

図10.2 n次元空間ベクトルa、bの内積の成分表現

ここで内積(a,b)は、以下の式で求めることができます。

$$(a,b) = \sum_{i=1}^{n} a_i b_i$$

上記の式より、内積(a,b)は空間ベクトルaの成分a_iと空間ベクトルbの成分b_iとの積和演算を行なうことにより算出することができます。

3. n次元空間ベクトル*a*、*b*、内積(*a*,*b*)の成分格納法

図10.3にあらかじめ、X/YRAM上へ設定してある3次元空間ベクトル*a*、*b*と内積(*a*,*b*)の成分格納法を示します。

［アドレス］	XRAM
VECTORA	a_1
VECTORA+2	a_2
VECTORA+4	a_3
⋮	⋮
VECTORA+2n-2	a_{n-1}
VECTORA+2n	a_n

［アドレス］	YRAM
VECTORB	b_1
VECTORB+2	b_2
VECTORB+4	b_3
⋮	⋮
VECTORB+2n-2	b_{n-1}
VECTORB+2n	b_n

［アドレス］	*3 YRAM
#IN_PRO	(*a*,*b*)H
#IN_PRO+2	(*a*,*b*)L

*3 (*a*,*b*)H：(*a*,*b*)の上位16ビット
(*a*,*b*)L：(*a*,*b*)の下位16ビット

図10.3 n**次元空間ベクトル***a*、*b***と内積(***a*,*b***)の成分格納法**

4. 内積算出アルゴリズム

図10.4に内積(a,b)の算出アルゴリズムを示します。以下に詳細アルゴリズムを説明します。

① Xアドレスレジスタ(R4)、Yアドレスレジスタ(R6,R7)にn次元空間ベクトルa、bの成分が格納されているアドレスと、a、bの内積を格納するアドレスを設定する。

② n次元空間ベクトルa、bの成分a_iとb_iの積和演算を行ないます。

③ 内積(a,b)の上位16ビット内積(a,b)HをIN_PROアドレスへ格納し、下位16ビット内積(a,b)LをIN_PRO+2アドレスに格納します。ここで処理は終了です。

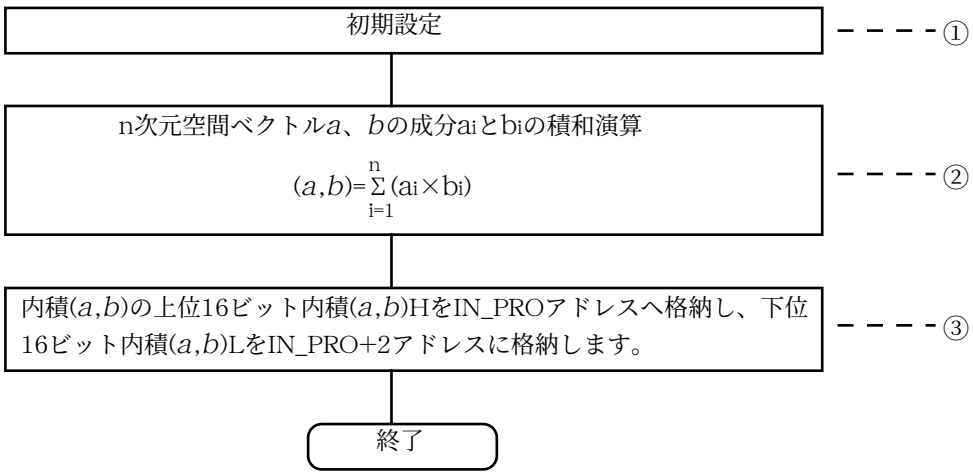
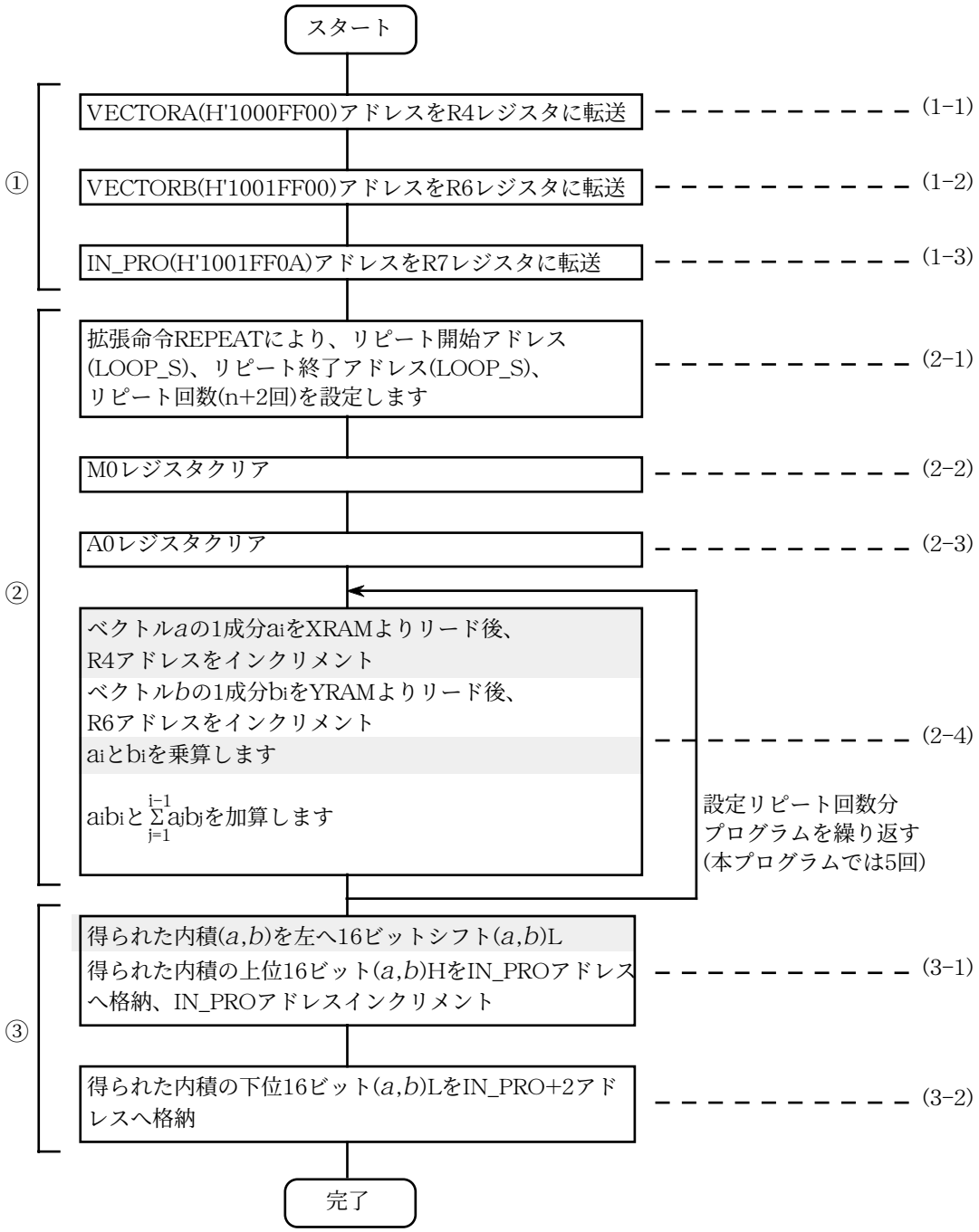


図10.4　内積算出アルゴリズム

フローチャート


スタート
①
VECTORA(H'1000FF00)アドレスをR4レジスタに転送
(1-1)
VECTORB(H'1001FF00)アドレスをR6レジスタに転送
(1-2)
IN_PRO(H'1001FF0A)アドレスをR7レジスタに転送
(1-3)
②
拡張命令REPEATにより、リピート開始アドレス(LOOP_S)、リピート終了アドレス(LOOP_S)、リピート回数(n+2回)を設定します
(2-1)
M0レジスタクリア
(2-2)
A0レジスタクリア
(2-3)
ベクトルaの1成分aiをXRAMよりリード後、R4アドレスをインクリメント
ベクトルbの1成分biをYRAMよりリード後、R6アドレスをインクリメント
aiとbiを乗算します
aibiと$\sum_{j=1}^{i-1} a_j b_j$を加算します
(2-4)
設定リピート回数分プログラムを繰り返す(本プログラムでは5回)
③
得られた内積(a,b)を左へ16ビットシフト(a,b)L
得られた内積の上位16ビット(a,b)HをIN_PROアドレスへ格納、IN_PROアドレスインクリメント
(3-1)
得られた内積の下位16ビット(a,b)LをIN_PRO+2アドレスへ格納
(3-2)
完了

メインプログラム

本プログラムでは、3次元空間ベクトル{ai、bi　(i = 1,2,3)}での内積の算出を行なえます。

in_pro.src

```
;*************************************************************
;*                              内積演算ﾙｰﾁﾝ
;*
;*                              (a,b)=a1b1+a2b2+a3b3
;*
;*************************************************************
;*************************************************************
;*                              初期設定ﾙｰﾁﾝ
;*************************************************************
MAIN:   MOV.L           #VECTORA,R4
        MOV.L           #VECTORB,R6
        MOV.L           #IN_PRO,R7

;*************************************************************************************************************
;*                              積和演算ﾙｰﾁﾝ
;*************************************************************************************************************
        REPEAT  LOOP_S,LOOP_S,#5                                                        ;ﾍﾞｸﾄﾙaの成分数+2ｶﾞﾘﾋﾟｰﾄ回数
        PCLR    A0
        PCLR    M0
        PCLR    X0
        PCLR    Y0
LOOP_S:
        PADD    A0,M0,A0  PMULS   X0,Y0,M0          MOVX.W  @R4+,X0  MOVY.W  @R6+,Y0  ;ai,bi load

;*************************************************************************************************************
;*                              内積格納ﾙｰﾁﾝ
;*************************************************************************************************************
STORE:  PSHA    #16,A0                                                  MOVY.W  A0,@R7+   ;内積上位格納
                                                                        MOVY.W  A0,@R7    ;内積下位格納

EXIT:   BRA     EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*********************************************************************
;*                      内積演算用ﾃﾞｰﾀ(X/YRAM)
;*********************************************************************
                .SECTION XRAM,DATA,LOCATE=H'1000FF00
VECTORA: .      .XDATA.W        0.5,0.125,0.5,0,0

                .SECTION YRAM,DATA,LOCATE=H'1001FF00
VECTORB: .      .XDATA.W        0.25,0.0625,0.25,0,0
IN_PRO:         .RES.W          2
```

11. 平方根

概要

16ビットの固定小数点の平方根演算を行い、15ビット精度の平方根を算出します。

説明

1. 入出力値のデータフォーマット

図11.1に入出力値のデータフォーマットを示します。被平方根Xは、最上位ビットを0とした16ビットの形式で入力します。但し、被平方根Xは正規化処理を行なって置く必要があります。
被平方根√Xは、最上位ビットを0とした16ビット(1ワード)の形式で出力されます。

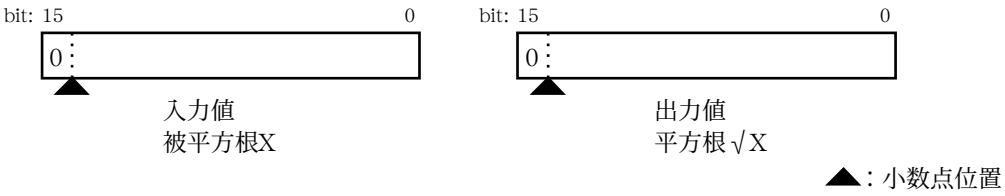


図11.1　入出力値データフォーマット

2. 平方根算出方法

図11.2に平方根関数を示します。本プログラムは、被平方根Xから**図11.2平方根関数**に示す折れ線状グラフより近似値を算出し、次に漸化式より正確な値に収束させ平方根√Xを算出する手法を採用しています。
被平方根を正規化処理すると被平方根Xのとりうる範囲は次のように表されます。

$$0 \leqq X < 1.0$$
$$(H'0000 \leqq X \leqq H'7FFF)$$

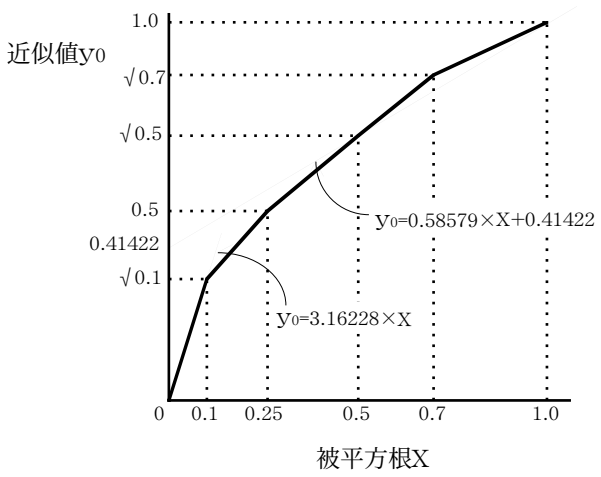


図11.2　平方根関数

図11.2の平方根関数で折れ線グラフの傾きが、比較的緩やかな、0.1より大きい区間と0.1以下の急な区間で近似式(1)(2)を作り、この2式を用いて近似平方根(y0)を算出します。

入力値X＞0.1

$$y_0=0.58579\times X+0.41422 \quad \text{－－－－－－－－－－－－－－－－－－－－} (1)$$

入力値X≦0.1

$$y_0=3.16228\times X \quad \text{－－－－－－－－－－－－－－－－－－－－} (2)$$

(実際のプログラムでは、$y_0=0.79057\times X\times 2^2$として用いる。)

なお、(2)式はこのままでは、固定小数点演算に用いることができないので、正規化処理し、$y_0=0.79057\times X\times 2^2$として用います。

説明

次に近似式(1)(2)より求めたy_0を(3)式に示す漸化式に代入して精度の良い平方根√Xを算出します。

$$y_0=\sqrt{X}=1/2(y_0+X/y_0) \quad \text{－－－－－－－－－－－－－－－－－－} (3)$$

ここで、(3)式の第2項の計算において、被平方根Xは正規化されており、(1)(2)式の計算から$y_0>X$となるためにX/y_0の値は、必ず正規化値となっています。本プログラムでは(3)式に示す漸化式を3回行なうことにより、15ビット精度の平方根を算出します。

3. 固定小数点の平方根算出アルゴリズム

以下に固定小数点の平方根算出アルゴリズムを示します。

① 初期設定を行ないます。

② 被平方根Xが0でないかを判定します。0の場合には、平方根√Xを0として処理を終了します。

③ 被平方根Xが負でないかを判定します。負の場合には、平方根√XをH'FFFFとして処理を終了します。

④ 被平方根XとH'7FFBの大小比較を行ないます。被平方根X＞H'7FFBの場合には、被平方根Xを平方根√X(=X)として処理を終了します。

⑤ 被平方根Xと0.1の大小比較を行ないます。被平方根X＞0.1の場合には、⑥の処理へ移り、被平方根X≦0.1の場合には、⑥'の処理へ移ります。

⑥ (1)式により近似平方根y_0を算出します。⑦の処理へ移ります。

⑥' (2)式により近似平方根y_0を算出します。⑦の処理へ移ります。

⑦ 近似平方根y_0と被平方根Xの大小比較を行ないます。近似平方根y_0=被平方根Xの場合には、近似平方根y_0を2で割り、0.5(H'4000)を加算して平方根√Xとして処理を終了します。

⑧ ⑦の大小比較した結果、被平方根X＞近似平方根y_0の場合には、漸化式のX/y_0が実行できません。そこで平方根√XをH'FFFFとして処理を終了します。

⑨ (3)式の漸化式により平方根yを算出し、√Xとして処理を終了します。

説明

図11.3に平方根算出アルゴリズムを示します。

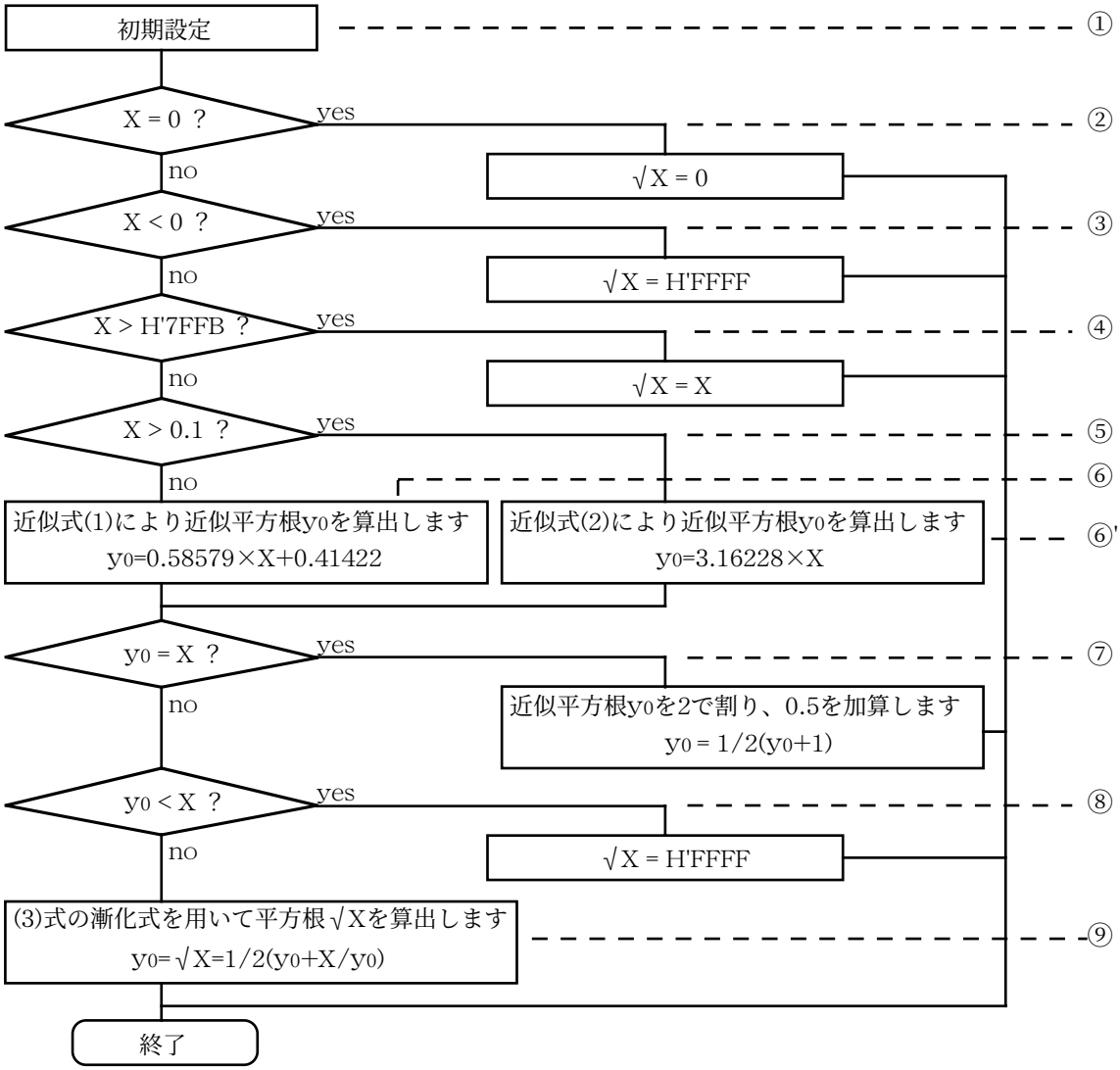
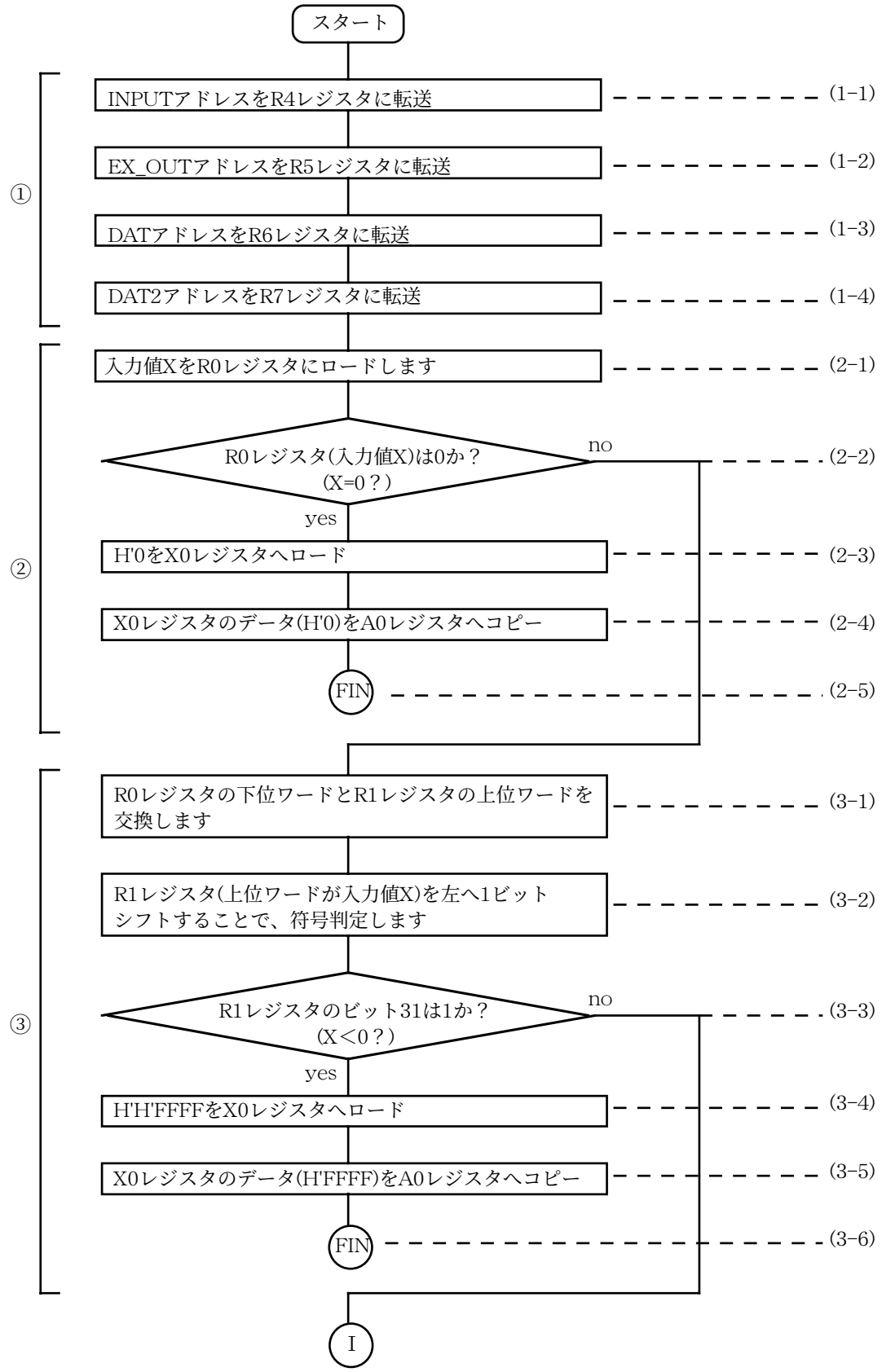
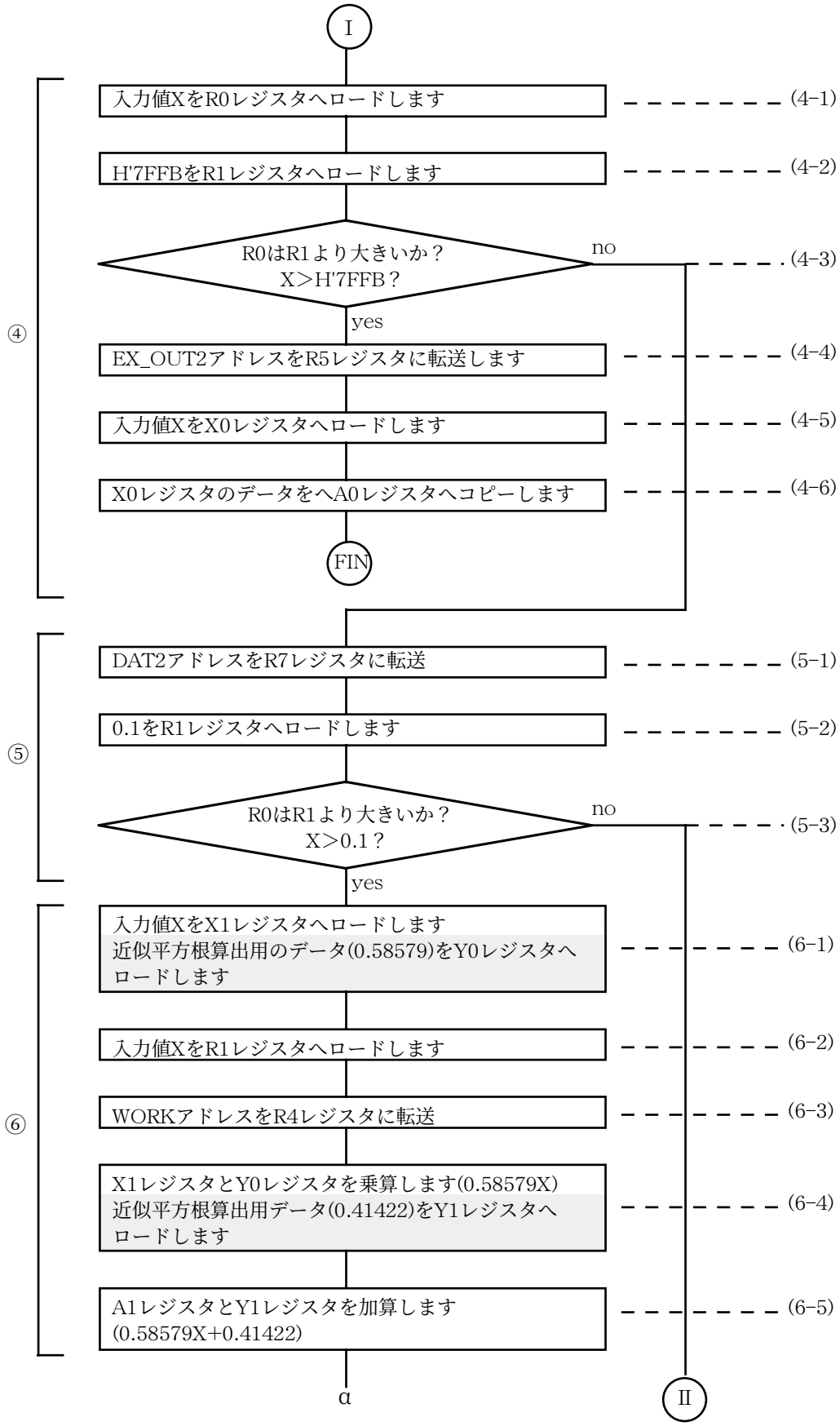


図11.3　平方根算出アルゴリズム

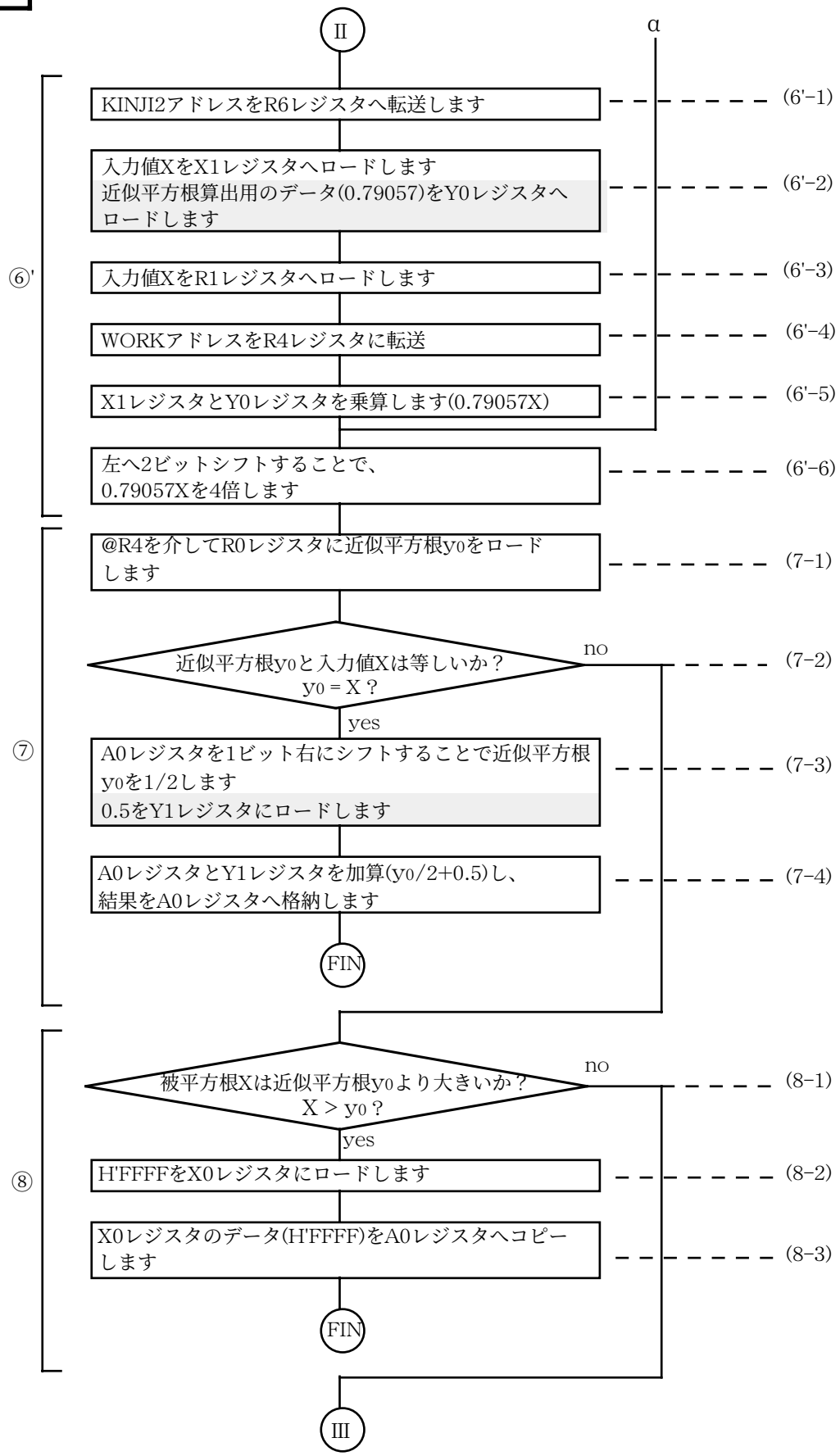
フローチャート


スタート
INPUTアドレスをR4レジスタに転送
EX_OUTアドレスをR5レジスタに転送
DATアドレスをR6レジスタに転送
DAT2アドレスをR7レジスタに転送
①
入力値XをR0レジスタにロードします
R0レジスタ(入力値X)は0か？
(X=0？)
no
yes
H'0をX0レジスタへロード
X0レジスタのデータ(H'0)をA0レジスタへコピー
FIN
②
R0レジスタの下位ワードとR1レジスタの上位ワードを
交換します
R1レジスタ(上位ワードが入力値X)を左へ1ビット
シフトすることで、符号判定します
R1レジスタのビット31は1か？
(X＜0？)
no
yes
H'H'FFFFをX0レジスタへロード
X0レジスタのデータ(H'FFFF)をA0レジスタへコピー
FIN
③
I
(1-1)
(1-2)
(1-3)
(1-4)
(2-1)
(2-2)
(2-3)
(2-4)
(2-5)
(3-1)
(3-2)
(3-3)
(3-4)
(3-5)
(3-6)

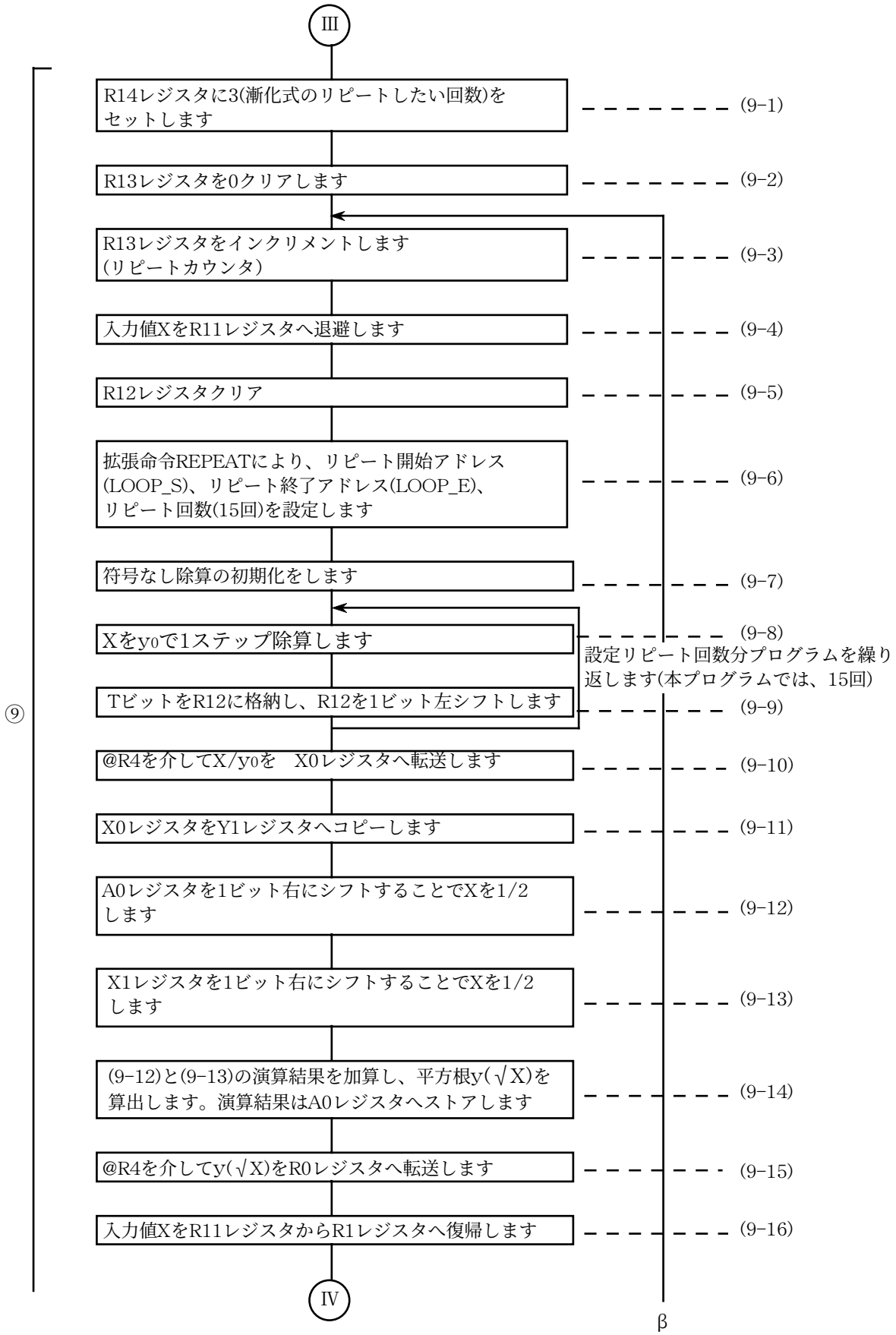
フローチャート


I
入力値XをR0レジスタへロードします
(4-1)
H'7FFBをR1レジスタへロードします
(4-2)
R0はR1より大きいか？
X＞H'7FFB？
no
(4-3)
yes
④
EX_OUT2アドレスをR5レジスタに転送します
(4-4)
入力値XをX0レジスタへロードします
(4-5)
X0レジスタのデータをへA0レジスタへコピーします
(4-6)
FIN
DAT2アドレスをR7レジスタに転送
(5-1)
0.1をR1レジスタへロードします
(5-2)
⑤
R0はR1より大きいか？
X＞0.1？
no
(5-3)
yes
入力値XをX1レジスタへロードします
近似平方根算出用のデータ(0.58579)をY0レジスタへロードします
(6-1)
入力値XをR1レジスタへロードします
(6-2)
WORKアドレスをR4レジスタに転送
(6-3)
⑥
X1レジスタとY0レジスタを乗算します(0.58579X)
近似平方根算出用データ(0.41422)をY1レジスタへロードします
(6-4)
A1レジスタとY1レジスタを加算します
(0.58579X+0.41422)
(6-5)
α
II

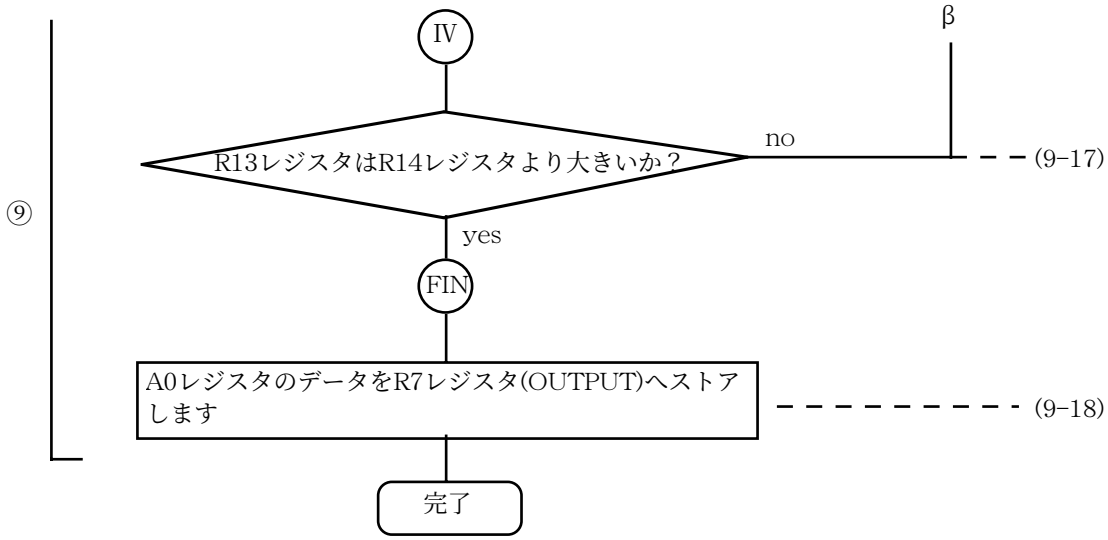
フローチャート


II
α
KINJI2アドレスをR6レジスタへ転送します
(6'-1)
入力値XをX1レジスタへロードします
近似平方根算出用のデータ(0.79057)をY0レジスタへロードします
(6'-2)
⑥'
入力値XをR1レジスタへロードします
(6'-3)
WORKアドレスをR4レジスタに転送
(6'-4)
X1レジスタとY0レジスタを乗算します(0.79057X)
(6'-5)
左へ2ビットシフトすることで、
0.79057Xを4倍します
(6'-6)
@R4を介してR0レジスタに近似平方根y0をロードします
(7-1)
近似平方根y0と入力値Xは等しいか？
y0 = X？
no
(7-2)
yes
⑦
A0レジスタを1ビット右にシフトすることで近似平方根y0を1/2します
0.5をY1レジスタにロードします
(7-3)
A0レジスタとY1レジスタを加算(y0/2+0.5)し、
結果をA0レジスタへ格納します
(7-4)
FIN
被平方根Xは近似平方根y0より大きいか？
X > y0？
no
(8-1)
yes
⑧
H'FFFFをX0レジスタにロードします
(8-2)
X0レジスタのデータ(H'FFFF)をA0レジスタへコピーします
(8-3)
FIN
III

フローチャート


Ⅲ
R14レジスタに3(漸化式のリピートしたい回数)をセットします
(9-1)
R13レジスタを0クリアします
(9-2)
R13レジスタをインクリメントします(リピートカウンタ)
(9-3)
入力値XをR11レジスタへ退避します
(9-4)
R12レジスタクリア
(9-5)
拡張命令REPEATにより、リピート開始アドレス(LOOP_S)、リピート終了アドレス(LOOP_E)、リピート回数(15回)を設定します
(9-6)
符号なし除算の初期化をします
(9-7)
Xをy0で1ステップ除算します
(9-8)
設定リピート回数分プログラムを繰り返します(本プログラムでは、15回)
TビットをR12に格納し、R12を1ビット左シフトします
(9-9)
⑨
@R4を介してX/y0を　X0レジスタへ転送します
(9-10)
X0レジスタをY1レジスタへコピーします
(9-11)
A0レジスタを1ビット右にシフトすることでXを1/2します
(9-12)
X1レジスタを1ビット右にシフトすることでXを1/2します
(9-13)
(9-12)と(9-13)の演算結果を加算し、平方根y(√X)を算出します。演算結果はA0レジスタへストアします
(9-14)
@R4を介してy(√X)をR0レジスタへ転送します
(9-15)
入力値XをR11レジスタからR1レジスタへ復帰します
(9-16)
Ⅳ
β

フローチャート


⑨
IV
β
R13レジスタはR14レジスタより大きいか？
no
(9-17)
yes
FIN
A0レジスタのデータをR7レジスタ(OUTPUT)へストアします
(9-18)
完了

メインプログラム

rout.src

```
;*****************************************************************************************************************************************************
;*                                        平方根算出ﾙｰﾁﾝ
;*
;*                                        √X
;*
;*****************************************************************************************************************************************************
;*****************************************************************************************************************************************************
;*                                初期設定ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
MAIN:
        MOV.L           #INPUT,R4
        MOV.L           #EX_OUT,R5
        MOV.L           #KINJI1,R6
        MOV.L           #DAT1,R7

;*****************************************************************************************************************************************************
;*                                被平方根のｾﾞﾛﾁｪｯｸﾙｰﾁﾝ
;*****************************************************************************************************************************************************
        MOV.W           @R4,R0
        CMP/EQ          #0,R0
        BF              ZERO_CH                                                 ;ｾﾞﾛなら以下の処理へ
                                        MOVX.W  @R4,X0
        PCOPY   X0,A0
        BRA             FIN                                                     ;処理終了
        NOP

;*****************************************************************************************************************************************************
;*                                被平方根の負値ﾁｪｯｸﾙｰﾁﾝ
;*****************************************************************************************************************************************************
ZERO_CH:
        SWAP            R0,R1
        SHAL            R1
        BF              MINUS_CH                                                ;ﾏｲﾅｽなら以下の処理へ
                                        MOVX.W  @R5,X0
        PCOPY           X0,A0
        BRA             FIN                                                     ;処理終了
        NOP

;*****************************************************************************************************************************************************
;*                                被平方根とH'7FFBの比較ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
MINUS_CH:
        MOV.W           @R4,R0                                                  ;X load
        MOV.W           @R7,R1                                                  ;H'7FFB load
        CMP/GT          R1,R0                                                   ;R0 ＞ R1 ?
        BF              EQU_SEL                                                 ;X＞H'7FFBなら以下の処理へ
        MOV.L           #EX_OUT2,R5
                                        MOVX.W  @R5,X0                          ;X load
        PCOPY   X0,A0
        BRA             FIN
        NOP

;*****************************************************************************************************************************************************
;*                                近似式選択ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
EQU_SEL:
        MOV.L           #DAT2,R7
        MOV.W           @R7,R1
        CMP/GT          R1,R0
        BF              Y0_PRO2                                                 ;X ≦ 0.1ならｼﾞｬﾝﾌﾟ

*****************************************************************************************************************************************************
;*                                近似平方根 y0 算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
Y0_PRO1:
                                        MOVX.W  @R4,X1  MOVY.W  @R6+,Y0         ;入力値 X (被平方根) のﾛｰﾄﾞ,近似平方根算出用
        MOV.W           @R4,R1                                                  ;入力値 X (被平方根)をR1にｷｰﾌﾟ
        MOV.L           #WORK,R4
        PMULS   X1,Y0,A1                        MOVY.W  @R6+,Y1                 ;0.58579X,0.41422 load
        PADD    A1,Y1,A0                                                        ;0.58579X+0.41422 -> y0
        BRA             HIKAKU
        NOP
```

メインプログラム

```
;*****************************************************************************************************************************************
;*                              近似式(2) y0 算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************
Y0_PRO2:
        MOV.L           #KINJI2,R6
                                        MOVX.W  @R4,X1  MOVY.W  @R6+,Y0     ;入力値 X (被平方根)のﾛｰﾄﾞ,近似平方根算出用
        MOV.W           @R4,R1                          ;入力値 X (被平方根)をR1にｷｰﾌﾟ
        MOV.L           #WORK,R4
        PMULS   X1,Y0,A1                MOVY.W  @R6+,Y1 ;0.58579X,0.41422 load
        PSHA    #2,A0                                   ;0.58579X+0.41422 -> y0

*****************************************************************************************************************************************
;*                              近似平方根と被平方根の比較ﾙｰﾁﾝ/Part1
;*****************************************************************************************************************************************
HIKAKU:
                                        MOVX.W  A0,@R4                      ;CPUﾕﾆｯﾄに渡す
        MOV.W           @R4,R0
        CMP/EQ          R0,R1                           ;近似平方根  y0= 入力値 X (被平方根) ?
        BF              NOT_EQ                          ; y0 ≠ X なら以下の処理へ
        PSHA    #-1,A0                  MOVY.W  @R6,Y1  ;y0/2,0.5 load
        PADD    A0,Y1,A0                                ;y0/2-0.5
        BRA             FIN                             ;処理終了
        NOP

;*****************************************************************************************************************************************
;*                              近似平方根と被平方根の比較ﾙｰﾁﾝ/Part2
;*****************************************************************************************************************************************
NOT_EQ:
        CMP/GT          R0,R1
        BF              NOT_GT                          ; y0＜Xなら以下の処理へ
                                        MOVX.W  @R5,X0  ;H'FFFF load
        PCOPY   X0,A0
        BRA             FIN
        NOP

;*****************************************************************************************************************************************
;*                              漸化式によるy算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************
NOT_GT:
        MOV.L           #3,R14                          ;ﾘﾋﾟｰﾄ回数の設定
        MOV.L           #0,R13
LENEAR_LP:
        ADD             #1,R13                          ;ｶｳﾝﾄｱｯﾌﾟ

        MOV             R1,R11                          ;push X
        MOV.L           #0,R12                          ;R12ﾚｼﾞｽﾀｸﾘｱ
        REPEAT          LOOP_S,LOOP_E,#15
        DIV0U                                           ;符号なし初期化
LOOP_S:
        DIV1            R0,R1                           ;R1/R0
LOOP_E:
        ROTCL           R12                             ;Tﾋﾞｯﾄ格納
        MOV.W           R12,@R4
                                        MOVX.W  @R4,X0
        PCOPY   X0,Y1
        PSHA    #-1,A0                                  ;y0/2
        PSHA    #-1,Y1                                  ;(X/y0)/2
        PADD    A0,Y1,A0
                                        MOVX.W  A0,@R4
        MOV.W           @R4,R0
        MOV             R11,R1                          ;pop X

        CMP/GT          R14,R13
        BF              LENEAR_LP                       ;設定回数分ﾘﾋﾟｰﾄしたら抜ける

FIN:    MOV.L           #OUTPUT,R7
                                                MOVY.W  A0,@R7      ;平方根√Xを格納

EXIT:   BRA             EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*****************************************************************************************************************************
;*                                        平方根算出用ﾃﾞｰﾀ(X/YRAM)
;*****************************************************************************************************************************
                .SECTION XRAM,DATA,LOCATE=H'1000FF00
INPUT:          .RES.W          1                                               ;外部入力ﾃﾞｰﾀ格納ｴﾘｱ
WORK:           .RES.W          1                                               ;ﾜｰｸｴﾘｱ
EX_OUT:         .DATA.W         H'FFFF                                          ;入力値X＜0時の出力値
EX_OUT2:        .XDATA.W        1                                               ;入力値X＞H'7FFB時の出力値

                .SECTION YRAM,DATA,LOCATE=H'1001FF00
KINJI1:         .XDATA.W        0.58579,0.41422,0.5                             ;近似式(1)
KINJI2:         .XDATA.W        0.79057                                         ;近似式(2)
DAT1:           .DATA.W         H'7FFB
DAT2:           .XDATA.W        0.1
OUTPUT:         .RES.W          1                                               ;外部出力ﾃﾞｰﾀ格納ｴﾘｱ
```

実行例

入力値X(INPUT)の平方根√X(OUTPUT)の算出結果を**表11.1**に示します。

表11.1　平方根√Xの算出結果(漸化式3回実行時)

入力値X(10進)	入力値X(16進)	理論値(10進) √X	理論値(16進) √X	出力値(16進) √X
0.9999	H'7FFC	0.99995	H'7FFE	H'7FFF
0.99987	H'7FFB	0.99993	H'7FFD	H'7FFD
0.85	H'6CCD	0.92195	H'7602	H'7602
0.523	H'42F1	0.72319	H'5C91	H'5C90
0.34	H'2BB5	0.5831	H'4AA3	H'4AA2
0.136	H'1168	0.36878	H'2F34	H'2F33
0.087	H'0B23	0.29496	H'25C1	H'25C1
0.01	H'0147	0.1	H'0CCD	H'0CC9
0	H'0000	0	H'0000	H'0000
−0.7	H'A667	——	——	H'FFFF

12.　2乗平均誤差

概要

2つの変量a[i](16ビット成分)、b[i](16ビット成分)の2乗平均誤差を算出します。
(i = 1,2,⋯,n)

説明

1.　2乗平均誤差の求め方

2乗平均誤差を求める場合、はじめに2つの変量a[i]、b[i]間の誤差e[i]を考えます。関係式を(1)式に示します。

$$e[i] = a[i] - b[i] \quad (i = 1,2,\cdots,n) \quad \cdots\cdots (1)$$ *1

次に誤差分散Se^2を求めます。誤差分散Se^2は誤差e[i]の2乗の総和を、成分数(n)で除算することで算出できます。誤差e[i]の2乗の総和を以下に成分表現します。

$$1/n \cdot \Sigma e[i]^2 = 1/n \cdot (a[1] - b[1])^2 + (a[2] - b[2])^2 + \cdots + (a[n] - b[n])^2$$

ここで、誤差分散Se^2は以下の(2)式のように求めることができます。

$$Se^2 = 1/n \cdot \sum_{i=1}^{n} (a[i] - b[i])^2 \quad \cdots\cdots (2)$$

ここで2乗平均誤差$E[Se^2]$は誤差分散Se^2の平方根で表されます。(3)式に2乗平均誤差$E[Se^2]$の関係式を示します。

$$E[Se^2] = \sqrt{1/n \cdot \sum_{i=1}^{n} (a[i] - b[i])^2} \quad \cdots\cdots (3)$$

*1　a[i]：16ビット
　　b[i]：16ビット
　　e[i]：16ビット

2. 変量a[i]とb[i]等の成分格納法

2乗平均誤差を求める際に、誤差e[i]の2乗の総和を算出を行ないます。この処理速度を向上させるためにあらかじめ**図12.1**に示すようにX/YRAMへa[i]とb[i]の成分を格納します。なお、VECTORA+2n、VECTORA+2n+2及び、VECTORB+2n、VECTORB+2n+2には0をX/YRAMに格納しておきます。0を格納していない場合、本プログラムは正しく実行されません。

成分数nによる除算用に1/nの数値をXRAMへ格納しておきます。実際のプログラムでは、DSP命令に除算がないので、1/nで乗算する仕様になっています。

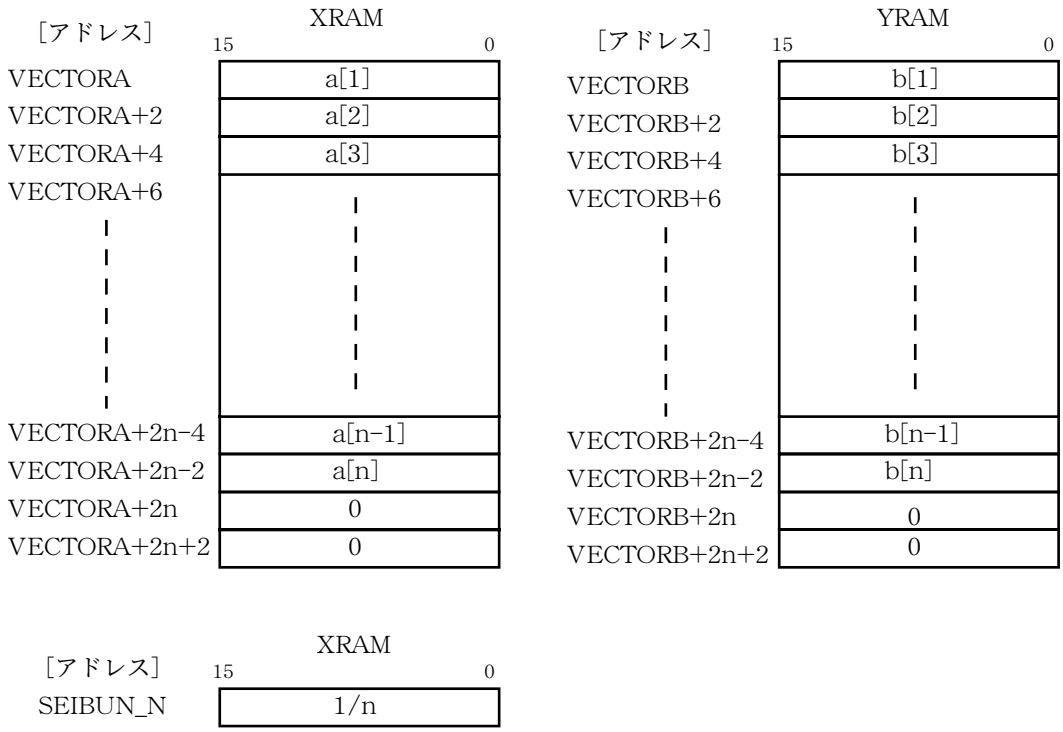


図12.1　変量a[i]とb[i]等を格納したメモリマップ

説明

3.　2乗平均誤差算出アルゴリズム

以下に2乗平均誤差算出アルゴリズム

① 初期設定を行ないます。

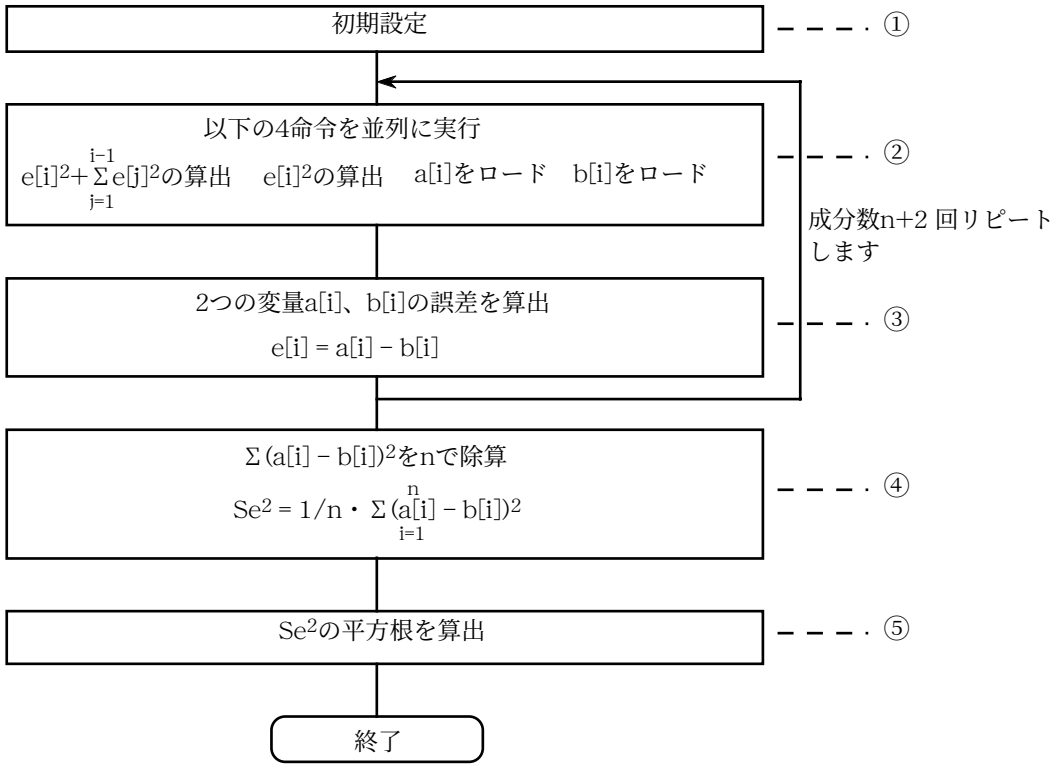
② ②③の項目で成分数n+2回リピートするように設定します。これは、以下の4つの命令を並列に実行しているため、2回余分にリピートしています。

$e[i]^2+\sum_{j=1}^{i-1} e[j]^2$の算出、e[i]の算出、a[i]のロード、b[i]のロード

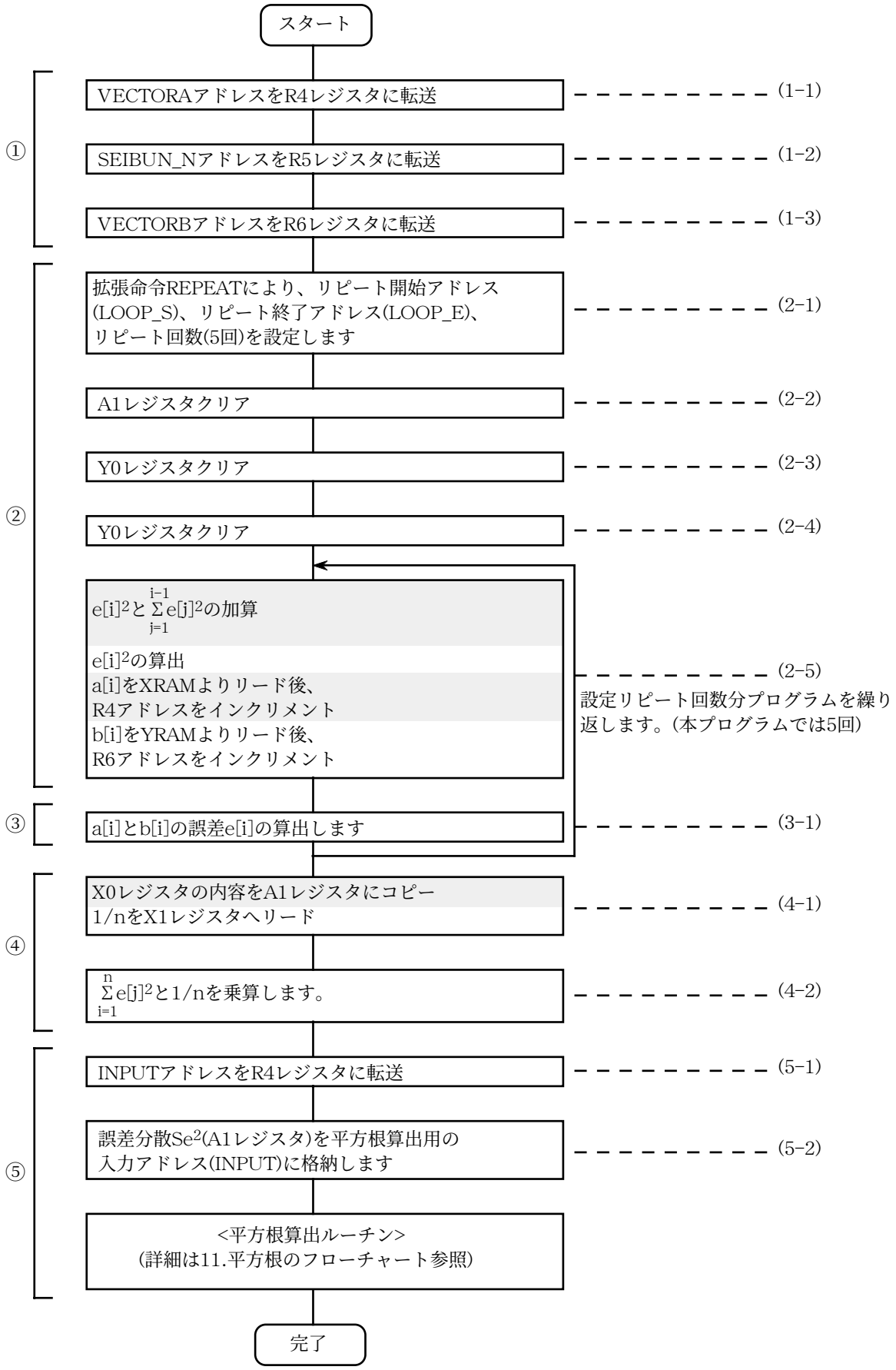
③ a[i]とb[i]の誤差e[i]を算出します。

④ ②③の処理で得られた$\sum_{i=1}^{n}(a[i]-b[i])^2$をnで除算します。

⑤ 入力された誤差分散Se^2の平方根を算出します。これにより2乗平均誤差が求められ、処理を終了します。
(詳細は**11.平方根**の**3.固定小数点の平方根算出アルゴリズム**を参照)



フローチャート


スタート
VECTORAアドレスをR4レジスタに転送
(1-1)
①
SEIBUN_NアドレスをR5レジスタに転送
(1-2)
VECTORBアドレスをR6レジスタに転送
(1-3)
拡張命令REPEATにより、リピート開始アドレス
(LOOP_S)、リピート終了アドレス(LOOP_E)、
リピート回数(5回)を設定します
(2-1)
A1レジスタクリア
(2-2)
Y0レジスタクリア
(2-3)
②
Y0レジスタクリア
(2-4)
e[i]2とΣe[j]2の加算
e[i]2の算出
a[i]をXRAMよりリード後、
R4アドレスをインクリメント
b[i]をYRAMよりリード後、
R6アドレスをインクリメント
(2-5)
設定リピート回数分プログラムを繰り
返します。(本プログラムでは5回)
③
a[i]とb[i]の誤差e[i]の算出します
(3-1)
X0レジスタの内容をA1レジスタにコピー
1/nをX1レジスタへリード
(4-1)
④
Σe[j]2と1/nを乗算します。
(4-2)
INPUTアドレスをR4レジスタに転送
(5-1)
誤差分散Se2(A1レジスタ)を平方根算出用の
入力アドレス(INPUT)に格納します
(5-2)
⑤
<平方根算出ルーチン>
(詳細は11.平方根のフローチャート参照)
完了

メインプログラム

本プログラムでは3成分{a[i]、b[i]　(i=1,2,3)}での2乗平均誤差の算出を行なえます。

squ_ave.src

```
;*****************************************************************************************************************************************************
;*                                        2乗平均ﾙｰﾁﾝ
;*
;*                                        a[i],b[i]
;*
;*****************************************************************************************************************************************************
;*****************************************************************************************************************************************************
;*                                    初期設定ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
MAIN:
        MOV.L           #VECTORA,R4
        MOV.L           #SEIBUN_N,R5
        MOV.L           #VECTORB,R6

;*****************************************************************************************************************************************************
;*                                    誤差分散算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
        REPEAT  LOOP_S,LOOP_E,#5                                                                        ;ﾍﾞｸﾄﾙaの成分数+2がﾘﾋﾟｰﾄ回数
        PCLR    A1
        PCLR    Y0
        PCLR    A0
LOOP_S:
        PADD    A0,Y0,Y0  PMULS   A1,A1,A0        MOVX.W  @R4+,X0  MOVY.W  @R6+,Y1       ;a[i],b[i ]load
LOOP_E:
        PSUB    X0,Y1,A1

        PCOPY   Y0,A1                             MOVX.W  @R5,X1                        ;1/3 load
        PMULS   X1,A1,A1                                                                ;0.33333 × Σ(a[i] - b[i])2

;*****************************************************************************************************************************************************
;*                                    被平方根格納ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
        MOV.L           #INPUT,R4
                                                  MOVX.W  A1,@R4                        ;

;*****************************************************************************************************************************************************
;*                                    平方根算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
;*****************************************************************************************************************************************************
;*                                    初期設定ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
SEMI_MAIN:
        MOV.L           #EX_OUT,R5
        MOV.L           #DAT,R6
        MOV.L           #DAT2,R7

;*****************************************************************************************************************************************************
;*                                    被平方根のｾﾞﾛﾁｪｯｸﾙｰﾁﾝ
;*****************************************************************************************************************************************************
        MOV.W           @R4,R0
        CMP/EQ          #0,R0
        BF              ZERO_CH
                                                  MOVX.W  @R4,X0                        ;H'0 load
        PCOPY   X0,A0                                                                   ;
        BRA             FIN                                                             ;処理終了
        NOP

;*****************************************************************************************************************************************************
;*                                    被平方根の負値ﾁｪｯｸﾙｰﾁﾝ
;*****************************************************************************************************************************************************
ZERO_CH:
        SWAP            R0,R1
        SHAL            R1
        BF              MINUS_CH                                                        ;ﾏｲﾅｽなら以下の処理へ進む
                                                  MOVX.W  @R5,X0                        ;H'FFFF load
        PCOPY           X0,A0
        BRA             FIN                                                             ;処理終了
        NOP
```

メインプログラム

```
;*****************************************************************************************************************************************************
;*                              被平方根とH'7FFBの比較ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
MINUS_CH:
        MOV.W           @R4,R0                                                          ;X load
        MOV.W           @R7,R1                                                          ;H'7FFB load
        CMP/GT          R1,R0                                                           ;R0 ＞ R1 ?
        BF              EQU__SEL                                                        ;R1が大きければｼﾞｬﾝﾌﾟ
        MOV.L           #EX_OUT2,R5
                                        MOVX.W  @R5,X0                                  ;X load
        PCOPY   X0,A0
        BRA             FIN
        NOP

;*****************************************************************************************************************************************************
;*                              近似式選択ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
EQU_SEL:
        MOV.L           #DAT2,R7
        MOV.W           @R7,R1
        CMP/GT          R1,R0
        BF              Y0_PRO2

;*****************************************************************************************************************************************************
;*                              近似式(1) y0 算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
Y0_PRO1:
                                        MOVX.W  @R4,X1  MOVY.W  @R6+,Y0         ;入力値 X (被平方根)のﾛｰﾄﾞ,近似平方根算出用
        MOV.W           @R4,R1                                                  ;入力値 X (被平方根)をR1にｷｰﾌﾟ
        MOV.L           #WORK,R4
        PMULS   X1,Y0,A1                        MOVY.W  @R6+,Y1                 ;0.58579X,0.41422 load
        PADD    A1,Y1,A0                                                        ;0.58579X+0.41422 -> y0
        BRA             HIKAKU
        NOP

;*****************************************************************************************************************************************************
;*                              近似式(2) y0 算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
Y0_PRO2:
        MOV.L           #KINJI2,R6
                                        MOVX.W  @R4,X1  MOVY.W  @R6+,Y0         ;入力値 X (被平方根)のﾛｰﾄﾞ,近似平方根算出用
        MOV.W           @R4,R1                                                  ;入力値 X (被平方根)をR1にｷｰﾌﾟ
        MOV.L           #WORK,R4
        PMULS   X1,Y0,A0                                                        ;0.79057 × X
        PSHA    #2,A0                                                           ;(0.79057 × X) ×4

;*****************************************************************************************************************************************************
;*                              近似平方根と被平方根の比較ﾙｰﾁﾝ/Part1
;*****************************************************************************************************************************************************
HIKAKU:
                                        MOVX.W  A0,@R4                          ;CPUﾕﾆｯﾄに渡す
        MOV.W           @R4,R0
        CMP/EQ          R0,R1                                                   ;近似平方根 = 入力値 X (被平方根) ?
        BF              NOT_EQ
        PSHA    #-1,A0                          MOVY.W  @R6,Y1                  ;y0/2,0.5 load
        PADD    A0,Y1,A0                                                        ;y0/2-0.5
        BRA             FIN
        NOP
;*****************************************************************************************************************************************************
;*                              近似平方根と被平方根の比較ﾙｰﾁﾝ/Part2
;*****************************************************************************************************************************************************
NOT_EQ:
        CMP/GT          R0,R1
        BF              NOT_GT
                                        MOVX.W  @R5,X0                          ;H'FFFF load
        PCOPY   X0,A0
        BRA             FIN
        NOP
;
```

メインプログラム

```
;*****************************************************************************************************************************************************
;*                                  漸化式によるy算出ﾙｰﾁﾝ
;*****************************************************************************************************************************************************
NOT_GT:
        MOV.L       #3,R14                                                                          ;ﾘﾋﾟｰﾄ回数の設定
        MOV.L       #0,R13
LENEAR_LP:
        ADD         #1,R13                                                                          ;ｶｳﾝﾄｱｯﾌﾟ

        MOV         R1,R11
        MOV.L       #0,R12
        REPEAT      DIV_S,DIV_E,#15
        DIV0U                                                                                       ;符号なし初期化
DIV_S:
        DIV1        R0,R1                                                                           ;R1/R0
DIV_E:
        ROTCL       R12                                                                             ;Tﾋﾞｯﾄ格納
        MOV.W       R12,@R4
                                        MOVX.W  @R4,X0
        PCOPY   X0,Y1
        PSHA    #-1,A0                                                                              ;y0/2
        PSHA    #-1,Y1                                                                              ;(X/y0)/2
        PADD    A0,Y1,A0
                                        MOVX.W  A0,@R4
        MOV.W       @R4,R0
        MOV         R11,R1

        CMP/GT      R14,R13
        BF          LENEAR_LP

FIN:    MOV.L       #OUTPUT,R7
                                                        MOVY.W  A0,@R7                              ;平方根√Xを格納


EXIT:   BRA     EXIT
        NOP
MAIN_E: NOP
```

データ

```
;*****************************************************************************************
;*                    2乗平均演算用ﾃﾞｰﾀ(X/YRAM)
;*****************************************************************************************
                    .SECTION XRAM,DATA,LOCATE=H'1000FF00
VECTERA:            .XDATA.W        0.5,0.125,0.5,0,0
SEIBUN_N:           .XDATA.W        0.33333                 ;1 / 成分数(n)

;*平方根算出用*
INPUT:              .RES.W          1
WORK:               .RES.W          1
EX_OUT:             .DATA.W         H'FFFF
EX_OUT2:            .XDATA.W        1

                    .SECTION YRAM,DATA,LOCATE=H'1001FF00
VECTERB:            .XDATA.W        0.25,0.0625,0.25,0,0

;*平方根算出用*
KINJI1:             .XDATA.W        0.58579,0.41422,0.5     ;近似式(1)
KINJI2:             .XDATA.W        0.79057                 ;近似式(2)
DAT1:               .DATA.W         H'7FFB
DAT2:               .XDATA.W        0.1
OUTPUT:             .RES.W          1
```

13. DSP命令による各プログラムの性能について

各機能のプログラムファイルの実行サイクル数は**表13.1、13.2**の通りです。

表13.1の測定条件はエミュレータE8000(SH7612)にて測定、各プログラムのメインプログラムはXRAMに割り付け、データをX/YRAMに割り付けました。

表13.2の測定条件はシミュレータ(SH-DSP)にて測定、各プログラムのメインプログラムはXROMに割り付け、データをX/YRAMに割り付けました。

表13.1　DSP命令を用いた各プログラムの性能について

プログラムファイル名	機能	実行サイクル数(Cycle)	備考
pmuls32.src	32ビット乗算	116	
tri_fun.src	3角関数	62	
matrix.src	行列演算	238	3×3行列演算
in_pro.src	内積	15	3次元空間ベクトル
rout.src	平方根	104	
squ_ave.src	2乗平均誤差	114	n = 3 (3成分)

表13.2　DSP命令を用いた各プログラムの性能について

プログラムファイル名	機能	実行サイクル数(Cycle)	備考
pmuls32.src	32ビット乗算	172	
tri_fun.src	3角関数	80	
matrix.src	行列演算	378	3×3行列演算
in_pro.src	内積	21	3次元空間ベクトル
rout.src	平方根	272	
squ_ave.src	2乗平均誤差	292	n = 3 (3成分)

SH-DSPソフトウェア編　アプリケーションノート
発行年月　平成11年8月　第1版
発　　行　株式会社　日立製作所
　　半導体グループ電子統括営業本部
編　　集　株式会社　超Lメディア
　　技術ドキュメントグループ

SH-DSP ソフトウェア編
アプリケーションノート

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

ADJ-502-076