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。

RENESAS

アプリケーション・ノート

μPD77016 ファミリ

ディジタル・シグナル・プロセッサ

基本ソフトウエア編

対象デバイス

μPD77016
μPD77018A
μPD77019
μPD77110
μPD77111
μPD77112
μPD77113A
μPD77114
μPD77115
μPD77210
μPD77213

資料番号 U11958JJ2V0AN00（第 2 版）
発行年月 September 2002 NS CP（K）

〔メ　モ〕

目次要約

本製品のうち，外国為替および外国貿易管理法の規定により規制貨物等（または役務）に該当するものについては，日本国外に輸出する際に，同法に基づき日本国政府の輸出許可が必要です。

- **本資料の内容は予告なく変更することがありますので，最新のものであることをご確認の上ご使用ください。**
- **文書による当社の承諾なしに本資料の転載複製を禁じます。**
- **本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的財産権その他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。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。**
- **本資料に記載された回路，ソフトウエア，及びこれらに付随する情報は，半導体製品の動作例，応用例を説明するためのものです。従って，これら回路・ソフトウエア・情報をお客様の機器に使用される場合には，お客様の責任において機器設計をしてください。これらの使用に起因するお客様もしくは第三者の損害に対して，当社は一切その責を負いません。**
- **当社は品質，信頼性の向上に努めていますが，半導体製品はある確率で故障が発生します。当社半導体製品の故障により結果として，人身事故，火災事故，社会的な損害等を生じさせない冗長設計，延焼対策設計，誤動作防止設計等安全設計に十分ご注意願います。**
- **当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定して頂く「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の上ご使用願います。**

  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

  特別水準：輸送機器（自動車，列車，船舶等），交通用信号機器，防災／防犯装置，各種安全装置，生命維持を直接の目的としない医療機器

  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，生命維持のための装置またはシステム等

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。当社製品を上記の「標準水準」の用途以外でご使用をお考えのお客様は，必ず事前に当社販売窓口までご相談頂きますようお願い致します。

M7　98.8

本版で改訂された主な箇所

箇　所	内　　容
全般	対象デバイスにμPD77111 ファミリ（μPD77110, 77111, 77112, 77113A, 77114, 77115），μPD77210 ファミリ（μPD77210, 77213）を追加
p.44	2.1.2　プログラムを修正
p.56	2.4　FFT プログラムの内容を変更，追加。

本文欄外の★印は，本版で改訂された主な箇所を示しています。

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

対象者　このアプリケーション・ノートは，μPD77016 ファミリの機能を理解し，それを用いたアプリケーション・プログラムを設計するユーザを対象としています。

μPD77016 ファミリは，μPD7701x ファミリ（μPD77015, 77016, 77017, 77018A, 77019），μPD77111 ファミリ（μPD77110, 77111, 77112, 77113A, 77114, 77115），およびμPD77210 ファミリ（μPD77210, 77213）の総称です。

目　的　このアプリケーション・ノートは，μPD77016 ファミリの基礎的な機能について，応用プログラムを用いてユーザに理解していただくことを目的としています。なお，掲載したプログラムは例示的に示したものであり，量産設計を対象にしたものではありません。

構　成　このアプリケーション・ノートは，大きく分けて次の内容で構成されています。

第 1 章　基本オペレーション
第 2 章　代表的なアプリケーション

読み方　このマニュアルの読者は，電気，論理回路やマイクロコンピュータ，C 言語に関する一般的知識が必要となります。

μPD7701x ファミリのハードウエア機能を知りたいとき
→**μPD7701x ファミリ　ユーザーズ・マニュアル　アーキテクチャ編**を参照してください。

μPD77111 ファミリのハードウエア機能を知りたいとき
→**μPD77111 ファミリ　ユーザーズ・マニュアル　アーキテクチャ編**を参照してください。

μPD77210 ファミリのハードウエア機能を知りたいとき
→**μPD77210 ファミリ　ユーザーズ・マニュアル　アーキテクチャ編**を参照してください。

μPD77016 ファミリの命令機能を知りたいとき
→**μPD77016 ファミリ　ユーザーズ・マニュアル　命令編**を参照してください。

凡　例

注	：本文中につけた注の説明
注意	：気をつけて読んでいただきたい内容
備考	：本文中の補足説明
数の表記	： 2 進数 … ××××または 0b××××
	10 進数 … ××××
	16 進数 … 0x××××

表現形式と対応レジスタ

表現形式	対応レジスタ
ro，ro’，ro”	R0-R7
rl，rl’	R0L-R7L
rh，rh’	R0H-R7H
re	R0E-R7E
reh	R0EH-R7EH
dp	DP0-DP7
dn	DN0-DN7
dm	DMX，DMY
dpx	DP0-DP3
dpy	DP4-DP7
dpx_mod	DPn，DPn++，DPn--，DPn##，DPn%%，!DPn##(**n**=0-3)
dpy_mod	DPn，DPn++，DPn--，DPn##，DPn%%，!DPn##(**n**=4-7)
dp_imm	DPn##imm(n=0-7)
＊××××	**×××をアドレスとするメモリの内容** **例** DP0 **レジスタの内容が** 1000 **のとき，***DP0 **はメモリの** 1000 **番地の内容を表します。**

関連資料　関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

μPD77016 ファミリに関する資料

資料名 / 品名	パンフレット	データ・シート	ユーザーズ・マニュアル		アプリケーション・ノート	
			アーキテクチャ編	命令編	基本ソフトウエア編	ライブラリ編
μPD77016	−	U10891J	U10503J	U13116J	このマニュアル	U12021J
μPD77018A		U11849J				
μPD77019						
μPD77110	U12395J	U12801J	U14623J			
μPD77111						
μPD77112						
μPD77113A		U14373J				
μPD77114						
μPD77115		U14867J				
μPD77210		U15203J	U15807J			
μPD77213						

開発ツールに関する資料

資 料 名		資料番号
HSM77016　ユーザーズ・マニュアル		U11602J
WB77016　ユーザーズ・マニュアル	言語編	U10078J
	操作編	U11506J
ID77016　ユーザーズ・マニュアル		U10118J
CC77016　ユーザーズ・マニュアル		U15037J
RX77016　ユーザーズ・マニュアル	機能編	U14397J
	コンフィギュレーション・ツール編	U14404J
RX77016　アプリケーション・ノート	HOST API 編	U14371J

注意　上記関連資料は，予告なしに内容を変更することがあります。設計などには，必ず最新の資料をご使用ください。

目　次

図の目次

表の目次

第 1 章　基本オペレーション

この章では，ディジタル・シグナル・プロセッサμPD77016 ファミリの乗算，除算，積和演算を使用した基本的なプログラムと，DSP の応用プログラムを説明します。

なお，このマニュアルに掲載したプログラムは，プログラムの動作を例示するためのものであり，実デバイスにおける動作を保証するものではありません。実デバイスでの動作においてはデバイスのブートアップを行い，**2.1 スタートアップ・プログラム**に示すようなベクタ領域の設定が必要です（**ユーザーズ・マニュアル　アーキテクチャ編**を参照）。また，実デバイスにおいては，データ RAM 領域に対してデータ付き領域確保（DW，DWL）はできません。

備考　使用するレジスタについて汎用レジスタの番号（R0-R7）による制限はありません。また，データ・ポインタについても，DP0-DP3（X データ・メモリ用），DP4-DP7（Y データ・メモリ用）による制限はありません。

1.1　固定小数点倍精度乗算

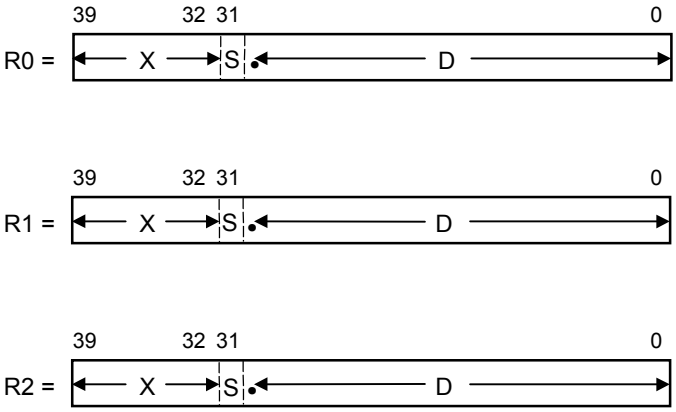
μPD77016 ファミリには乗算命令があります。その中の三項演算命令を用いて 2 つの 32 ビット固定小数点データを乗算し，32 ビットで結果を求めます。負数は 2 の補数で表現します。有効な数値範囲は，－1.0～ ＋0.9999.....です。ここでは，＋0.1422...（0x12345678）× －0.7911...（0x9ABCDEF0）の計算を例に説明します。

1.1.1　使用レジスタの説明

（1）汎用レジスタ

R0，R1：**乗算入力用**

R2：**演算結果保存用**



備考　S：符号，D：数値，X：無効，.：小数点位置

（2）データ・ポインタ

DP0，DP4：**アドレス値**

1.1.2　入力条件

1.1.1　使用レジスタの説明で示すように値を設定します。

R0 レジスタ ← 0x12345678(0.1422222219408)

R1 レジスタ ← 0x9ABCDEF0(－0.7911111190915)

R2 レジスタ ← 0 で初期化(三項演算命令で使用するため)

DP0 ← アドレス値を入力

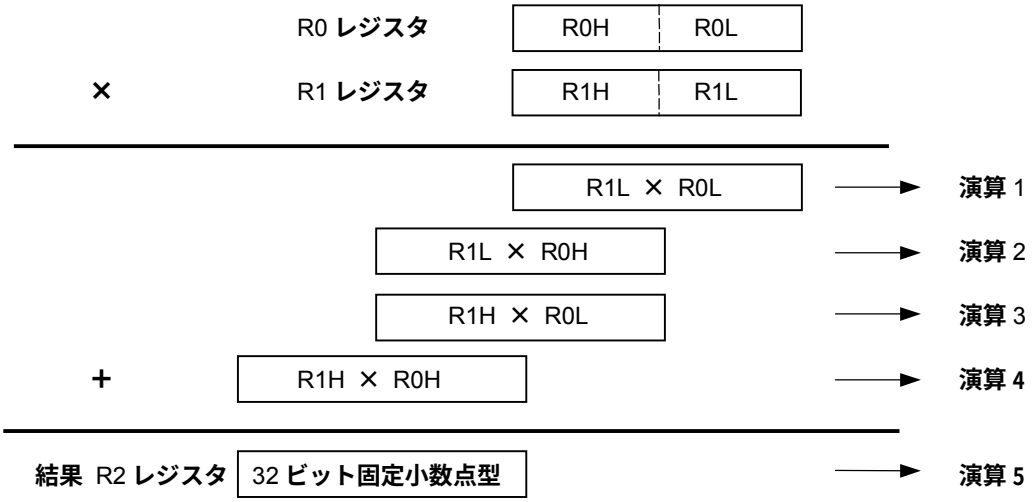
DP4 ← アドレス値を入力

1.1.3　出力条件

演算結果は，R2 レジスタに格納します。

1.1.4　考え方

μPD77016 ファミリの乗算命令には，16 ビット×16 ビットの命令があります。まず，2 つの 32 ビット・レジスタを 16 ビットに分割し，それらの部分積の位取りを考慮しながら加算を行い結果を得ます。



このプログラム例では演算 1 を無視しています。演算 2，演算 3，演算 4 の総和を結果としています。したがって，演算結果 R2 レジスタの LSB には誤差が生じます。

① 演算 2＋演算 3 は，同一の位取りなので単純に計算できる。

② 演算 4 は 16 ビットの位取りの差がある。

③ 演算 2＋演算 3 の結果は 16 ビット右シフトしたあとに，演算 4 と加算しなければならない。

1.1.5　処理手順

① 各レジスタの初期化

② R0 レジスタ ← 0x12345678(0.1422222219408)

③ R1 レジスタ ← 0x9ABCDEF0(－0.79111111909)

④ R2 レジスタ ← R1 レジスタの下位 16 ビット×R0 レジスタの上位 16 ビット

⑤ R2 レジスタ ← ④＋R1 レジスタの上位 16 ビット×R0 レジスタの下位 16 ビット

⑥ R2 レジスタを 16 ビット右シフトする。

⑦ 上記⑥の値と，R1 レジスタの上位 16 ビット×R0 レジスタの上位 16 ビットの演算結果を加算する。

1.1.6　プログラム

プログラム例を示します。例 1 は 1.1.5　処理手順の考え方をもとにそのままコーディングしたものです。

μPD77016 ファミリには，データ用のメモリが 2 面あります。この 2 面のデータ・メモリは同時にアクセスすることができます。たとえば，演算の係数とデータとを別々のメモリに格納しておくことにより，演算対象レジスタへのデータのロードを高速に行うことが可能となります。この特性を利用した例を例 2 に示します。

例 1　最適化していないプログラム

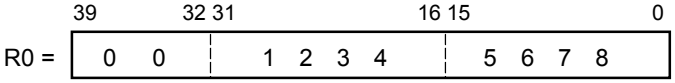
```
;; ============================== ;;
;; ======= Sample Program ======= ;;
;; ============================== ;;

;; ============ Data =============;

X_DATA XRAMSEG at 0x0000

DW  0x1234                        ;
DW  0x5678                        ;
DW  0x9ABC                        ;
DW  0xDEF0                        ;

;; === Program ===

Start imseg at 0x200

      jmp init                    ;
      nop                         ;

org 0x240

init:
      clr( r0 )                   ;  使用するレジスタの初期化
      clr( r1 )                   ;
      clr( r2 )                   ;

      dp0 = 0x0000                ;  データ・ポインタにアドレスを設定
      nop                         ;  データ・ポインタに値を設定した直後，同じデータ・ポインタの値を
                                  ;  アドレスとしたロード/ストアは禁止されています。そこで，この箇
                                  ;  所に nop 命令を記述しています。
main:

      r0  = *dp0++                ;
      r0L = *dp0++                ;

      r1  = *dp0++                ;
      r1L = *dp0++                ;

r2 = r2 + r1H * r0L               ;  No_1
r2 = r2 + r1L * r0H               ;  No_2
r2 = ( r2 >> 16 ) + r1H * r0H     ;  No_3

end
```

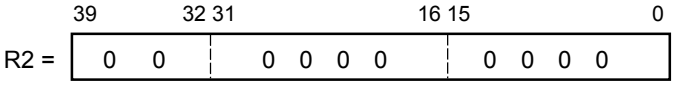
例２　μPD77016 ファミリの特性を考慮し，最適化したプログラム

```
;; ============================== ;;
;; ======= Sample Program ======= ;;
;; ============================== ;;

;; =========== Data ============= ;;

X_DATA XRAMSEG

X1:
DW  0x1234                      ;
DW  0x5678                      ;

;;============================== ;;

Y_DATA YRAMSEG

Y1:
DW  0x9ABC                      ;
DW  0xDEF0                      ;

;; ========== Program =========== ;;

Start imseg at 0x200

   jmp init                     ;
   nop                          ;


   org 0x240

init:

   dp0 = X1                     ; データ・ポインタにアドレスを設定
   dp4 = Y1                     ; dp0-dp3 は X メモリ用，dp4-dp7 は Y メモリ用のポインタ

   clr( r0 )                    ; レジスタの初期化
   clr( r1 )                    ; データ・ポインタの設定とレジスタの初期化の順番を入れ替えること
   clr( r2 )                    ; により例 1 にあった nop が不要となります。

main:

   r0H = *dp0++ r1H = *dp4++    ; データの入力
   r0L = *dp0     r1L = *dp4    ; X メモリ，Y メモリに対する同時アクセスが可能です。

   r2 = r2 + r0L * r1H          ; No_1
   r2 = r2 + r0H * r1L          ; No_2
   r2 = ( r2 >> 16 ) + r0H * r1H ; No_3

end
```

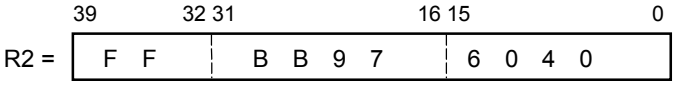
1.1.7　実行結果

（1）初期状態

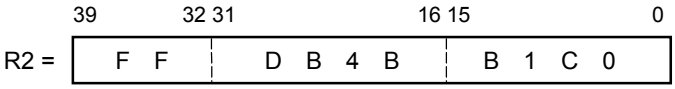




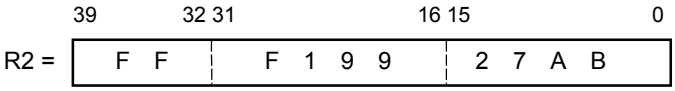
（2）No_1 を実行



（3）No_2 を実行



（4）No_3 を実行(結果)



結果：0xFFF19927AB（－0.1125135817565）

1.2　除算

μPD77016 ファミリには，1 ビットの商を計算する除算命令があります。マルチビットの演算は，この除算命令をリピート命令と組み合わせることにより実現します。計算で扱える数値フォーマットは固定小数点形式です。負数は 2 の補数で表現します。有効な数値範囲は－1.0～＋0.9999.....です。

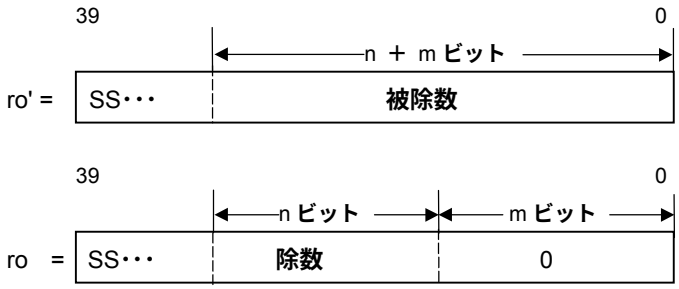
被除数 n＋m ビット，除数 n ビット，商を m ビットとして求める例を説明します。

ro'レジスタに n＋m ビットの被除数を設定し，ro レジスタに n ビットの除数を設定します。ro'の小数点位置は，ビット(n＋m－1)とビット(n＋m－2)の間にあります。ビット 39 -ビット｛39－(n＋m－1)｝が符号ビットです。除算命令　ro' /= ro　を m 回繰り返すと，ro'レジスタに，m ビットの商と n＋1 ビットの剰余が求められます。

注意　被除数レジスタの絶対値は，除数レジスタの絶対値より小さい値にしてください。

1.2.1　使用レジスタの説明

（1）汎用レジスタ



備考　S：符号

1.2.2　入力条件

1.2.1　使用レジスタの説明で示すように，値を設定します。

ro' ← 被除数

ro ← 除数，　m ビット(必要とする商の分)だけ 0 で初期化

1.2.3　出力条件

演算結果は，ro'レジスタに格納します。

1.2.4　処理手順

① ro'レジスタ ← 被除数

② ro レジスタ ← 除数

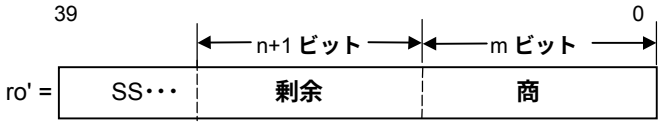
③　除算(リピート命令，除算命令)

注意　被除数レジスタの絶対値は，除数レジスタの絶対値より小さい値（ro'＜ro）にしてください。

1.2.5　プログラム

```
REP   m;
      ro' /= ro;
```

1.2.6　実行結果



注意　商 = $\dfrac{\text{被除数}}{|\text{除数}|}$

（1）剰余を求める

このμPD77016 ファミリの除算命令は，演算結果を 1 ビット左にシフトしてから，ro'レジスタの商のビット 0 に 1，または 0 を演算結果に応じて設定します。剰余が必要な場合には次の処理を実行してください。

```
ro' = ro' SRA 1          ;1 ビット右シフト
IF ( ro' < 0 ) ro'  ＋= ro    ;ro'が負の場合，除数（ro）を加算
ro' = ro' SRA m－1       ;（m－1）ビット右シフト
```

1.2.7　被除数 32 ビット，除数 16 ビットとし，商を 16 ビットの除算

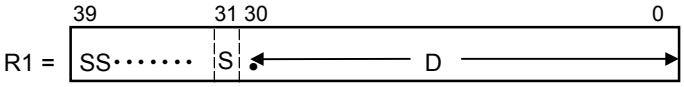
小数点位置は，ビット 31 とビット 30 の間に設定します。ビット 39-ビット 31 は符号ビットです。負数は 2 の補数で表現します。有効な数値範囲は －1.0～ ＋0.9999.....です。

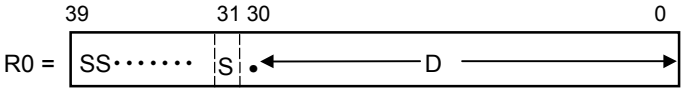
ここでは，－0.00126722920686（0xFFFFD679B6） ÷ ＋0.1422119140625（0x1234）の計算を例に説明します。

（1）使用レジスタの説明

（a）汎用レジスタ

R0，R1：被除数，除数用





備考　S：符号，D：数値，X：無効，．：小数点位置

（b）データ・ポインタ

DP0，DP4：アドレス値

（2）入力条件

（1）使用レジスタの説明で示すように値を設定します。

R1 ← 0xFFFFD679B6（－0.001267229207）

R0 ← 0x0012340000（0.1422119140625）

（3）出力条件

演算結果は，R1 レジスタに格納します。

（4）処理手順

① R1 レジスタ ← 0xFFFFD679B6（－0.001267229207）

② R0H レジスタ ← 0x1234（0.1422119140625）

③ R0L レジスタ ← 0 で初期化

④ REP 命令

⑤ 除算命令

(5) プログラム

プログラム例を示します。(4) 処理手順をもとにそのままコーディングした例を例 1 に，2 面あるデータ・メモリを有効に使用した例を例 2 に示します。

例 1　最適化していないプログラム

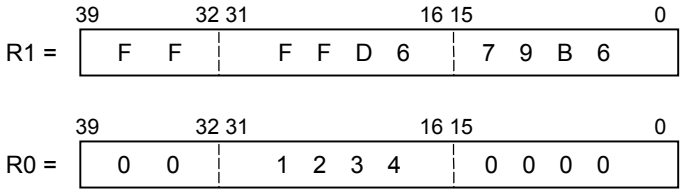
```
;; ====================== ;;
;; === Sample Program === ;;
;; ====================== ;;

;; ======== Data ======== ;;

Y_DATA YRAMSEG at 0x0000

DW 0xffd6                   ; -0.00126722927
DW 0x79b6                   ;
DW 0x1234                   ;  0.1422119141

;; ======= Program ====== ;;

Start imseg at 0x200
jmp init                    ;
nop                         ;

org 0x240

init:
clr( r0 )                   ;  使用するレジスタの初期化
clr( r1 )                   ;

dp4 = 0x0000                ;  データ・ポインタにアドレスを設定
  nop                       ;  データ・ポインタに値を設定した直後，同じデータ・ポインタの値を
                            ;  アドレスとしたロード/ストア命令は禁止されています。そこで，こ
                            ;  の箇所に nop 命 令を記述します。
main:
r1  = *dp4++                ;  被除数の設定
r1l = *dp4++                ;
r0  = *dp4                  ;  除数の設定

rep 16                      ;  除算を実行
          r1 /= r0          ;
end
```

例２　μPD77016 ファミリの特性を考慮し，最適化したプログラム

```
;; ============================ ;;
;; ====== Sample Program ====== ;;
;; ============================ ;;

;; ========== Data ========== ;;

X_DATA XRAMSEG

X0:
DW 0xffd6                       ; -0.00126722927
DW 0x79b6                       ;

Y_DATA YRAMSEG

Y0:
DW 0x1234                       ;  0.1422119141

;; ========== Program ========= ;;

Start imseg at 0x200

   jmp init                     ;
   nop                          ;

org 0x240

init:

   dp0 = X0                     ;  データ・ポインタにアドレスを設定
   dp4 = Y0

   clr( r0 )                    ;  使用するレジスタの初期化
   clr( r1 )                    ;

main:

   r1  = *dp0++  r0 = *dp4      ;  R1←被除数，R0←除数
   r1l = *dp0                   ;

   rep 16                       ;  除算を実行
      r1 /= r0                  ;


end
```

(6) 実行結果

(a) 初期状態



(b) R1 /= R0 を 16 回実行

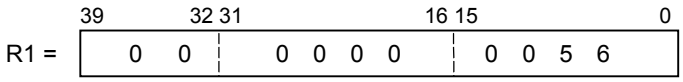
プログラム実行後，R1 レジスタのビット 15-ビット 0 に商を格納します。ビット 15 とビット 14 の間が小数点位置です。ビット 15 は符号ビットです。

商　：0xFEDC = －0x0124 = －0.0089111328125

(c) 剰余を求める

上記の演算結果をもとに，次の命令を行います。

```
R1 = R1 sra 1                ;1 ビット右シフト
IF ( R1 < 0 ) R1 ＋= R0      ;R1 が負の場合，除数(R0)を加算
R1 = R1 sra 15               ;15 ビット右シフト
```



小数点位置は，ビット 31 とビット 30 の間です。

剰余：0x56 = 0.00000004004687070847

1.2.8　被除数 16 ビット，除数 8 ビットとし，商を 8 ビットで求める場合

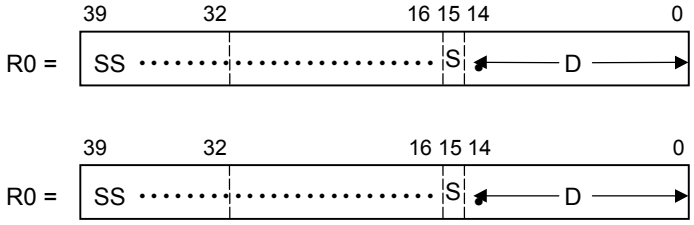
小数点位置は，ビット 15 とビット 14 の間に設定します。ビット 39-ビット 16 は符号ビットです。負数は 2 の補数で表現します。有効な数値範囲は －1.0～ ＋0.9999.....です。

ここでは＋0.5643005371094(0x000000483B) ÷ －0.671875(0xFFFFFFAA00)の計算を例に説明します。

(1) 使用レジスタの説明

(a) 汎用レジスタ

R0，R1：被除数，除数用



備考　S：符号，D：数値，X：無効，．：小数点位置

(b) データ・ポインタ

DP0，DP4：アドレス値

(2) 入力条件

(1) 使用レジスタの説明で示すように値を設定します。

R1 ← 0x000000483B （＋0.5643005371094）

R0 ← 0xFFFFFFAA00（－0.671875）

(3) 出力条件

演算結果は，R1 レジスタに格納します。

(4) 処理手順

① R1 レジスタ ← 0x000000483B(＋0.5643005371094)

② R0 レジスタ ← 0xFFFFFFAA00(－0.671875)

③ REP 命令

④ 除算命令

(5) プログラム

プログラム例を示します。(4) 処理手順をもとにそのままコーディングした例を例 1 に，2 面あるデータ・メモリを有効に使用した例を例 2 に示します。

例 1　最適化していないプログラム

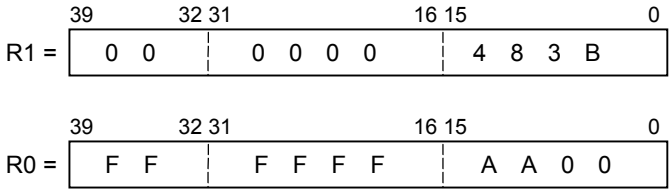
```
;; ======================== ;;
;; ==== Sample Program ==== ;;
;; ======================== ;;

;; ========= Data ========= ;;

Y_DATA YRAMSEG at 0x0000

DW 0x483b                   ;
DW 0xFFFF                   ;
DW 0xAA00                   ;

;; ======= Program ======== ;;

Start imseg at 0x200

    jmp init                ;
    nop                     ;

org 0x240

init:
    clr( r0 )               ; 使用するレジスタの初期化
    clr( r1 )               ;

    dp4 = 0x0000            ; データ・ポインタにアドレスを設定
    nop                     ; データ・ポインタに値を設定した直後，同じデータ・ポインタの値を
                            ; アドレスとしたロード/ストア命令は禁止されています。そこで，こ
                            ; の箇所に nop 命令を記述します。
main:

    r1L = *dp4++            ; 被除数の設定
    r0  = *dp4++            ; 除数の設定
    r0L = *dp4++            ;

    rep 8                   ; 除算を実行
            r1 /= r0        ;

end
```

例2　μPD77016ファミリの特性を考慮し，最適化したプログラム

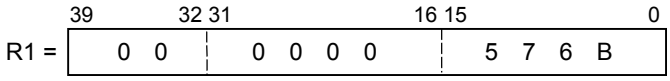
```
;; ================================ ;;
;; ======== Sample Program ======== ;;
;; ================================ ;;

;; ============ Data ============== ;;

X_DATA YRAMSEG

X0:
DW 0xFFFF                          ;
DW 0xAA00                          ;

Y_DATA YRAMSEG

Y0:
DW 0x483b                          ;

;; ========== Program ============= ;;

Start imseg at 0x200
   jmp init                        ;
   nop                             ;

org 0x240

init:

   dp0 = X0                        ; データ・ポインタにアドレスを設定
   dp4 = Y0                        ;

   clr( r0 )                       ; 使用するレジスタの初期化
   clr( r1 )                       ;

main:

   r0  = *dp0++ r1l = *dp4++       ; 被除数，除数の設定
   r0l = *dp0                      ;

   rep 8                           ; 除算を実行
      r1 /= r0                     ;

end
```

(6) 実行結果

(a) 初期状態



(b) R1 /= R0 を 8 回実行(結果)



プログラム実行後，R1 レジスタのビット 7 -ビット 0 に商を格納します。

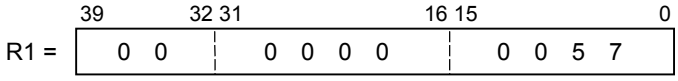
商　：0x6B = 0.8359375

注意　商 = $\dfrac{\text{被除数}}{|\text{除数}|}$

(c) 剰余を求める

上記の演算結果をもとに，次の命令を行います。

```
R1 = R1 sra 1              ;1 ビット右シフト
IF ( R1 < 0 ) R1＋= R0     ;R1 が負の場合，除数(R0)を加算
R1 = R1 sra 7              ;7 ビット右シフト
```



小数点位置は，ビット 15 とビット 14 の間です。

剰余：0x57 = ＋0.002655029296875

1.2.9　整数，あるいは |被除数| ＞ |除数| で除算を行う場合

μPD77016 **ファミリの除算命令は，次のように工夫をすると，整数，および|被除数|＞|除数|の除算ができます。ここでは，**＋100(0x64) ÷ ＋15(0xF)**の計算を例に説明します。**

① 被除数 100(0x64)を設定

小数点位置をビット 31**，ビット** 30 **の間とすると，被除数は次のようになります。**

0011 0010　0.000 0000 0000 0000　0000 0000 0000 0000

上記の値を右に 15 **ビット・シフトします。**

0000 0000　0.000 0000 0110 0100　0000 0000 0000 0000

② 除数 15（0xF）を設定

小数点位置をビット 31，**ビット** 30 **の間とすると，除数は次のようになります。**

0000 0111　1.000 0000 0000 0000　0000 0000 0000 0000

③ 除算する場合，|被除数|＜|除数|でなければならない

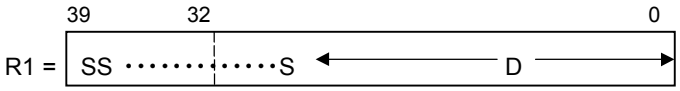
そこで，除数の値を 11 **ビット右シフトします。**

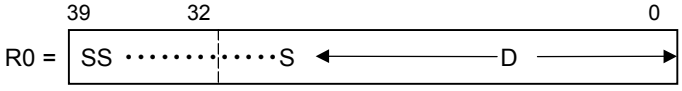
0000 0000　0.000 0000 1111 0000　0000 0000 0000 0000

（1**）使用レジスタの説明**

（a**）汎用レジスタ**

R0，R1**：被除数，除数用**





備考　S**：符号，**D**：数値，**X**：無効，．：小数点位置**

（b**）データ・ポインタ**

DP0，DP4**：アドレス値**

（2**）入力条件**

（1**）使用レジスタの説明で示すように，値を設定します。**

R1 ← 0x0000640000(**被除数**:100)

R0 ← 0x0000F00000(**除数**:15)

（3**）出力条件**

演算結果は，R1 **レジスタに格納します。**

（4**）処理手順**

① R1 **レジスタ** ← 0x0000640000

② R0 **レジスタ** ← 0x0000F00000

③ REP **命令**

④ **除算命令**

(5) プログラム

プログラム例を示します。(4) 処理手順をもとにそのままコーディングした例を例 1 に，2 面あるデータ・メモリを有効に使用した例を例 2 に示します。

例 1　最適化していないプログラム

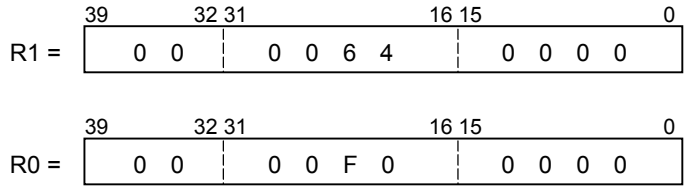
```
;; ====================== ;;
;; === Sample Program === ;;
;; ====================== ;;

;; === Data === ;;

X_DATA XRAMSEG at 0x0000

DW 0x0064              ;  100:0x64
DW 0x00F0              ;   15:0x0F

;; === Program === ;;

Start imseg at 0x200

   jmp init            ;
   nop                 ;

org 0x240

init:
   clr( r0 )           ;  使用するレジスタの初期化
   clr( r1 )           ;

   dp0 = 0x0000        ;  データ・ポインタにアドレスを設定
   nop                 ;  データ・ポインタに値を設定した直後，同じデータ・ポインタの値をアド
                       ;  レスとしたロード/ストア命令は禁止されています。そこで，この箇所に
                       ;  nop 命令を記述します。
main:

   r1 = *dp0++         ;  被除数の設定
   r0 = *dp0++         ;  除数の設定

   rep 16              ;  除算を実行
      r1 /= r0         ;

end
```

例2　μPD77016ファミリの特性を考慮し，最適化したプログラム

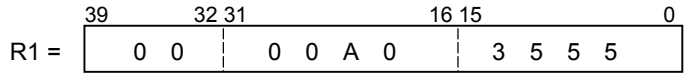
```
;; ======================== ;;
;; === Sample Program ==== ;;
;; ======================== ;;

;; ========= Data ======== ;;

X_DATA XRAMSEG

X0:
DW 0x0064                   ;

Y_DATA YRAMSEG

Y0:
DW 0x00F0                   ;

;; ======= Program ======= ;;

Start imseg at 0x200

   jmp init                 ;
   nop                      ;

org 0x240
init:
   dp0 = X0                 ;  被除数の設定
   dp4 = Y0                 ;  除数の設定

   clr( r0 )                ;  使用するレジスタの初期化
   clr( r1 )                ;

main:
   r1 = *dp0   r0 = *dp4    ;  被除数，除数の設定

   rep 16                   ;  除算を実行
      r1 /= r0              ;

end
```

(6) 実行結果

(a) 初期状態



(b) R1 /= R0 を 16 回実行(結果)



プログラム実行後，R1 レジスタのビット 15-ビット 0 に商が格納されます。

0000 0000　0000 0000 1010 0000　0011 0101 0101 0101

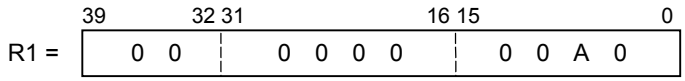
小数点位置はビット 15 とビット 14 の間となり，商は下位 16 ビットの 0.011010101010101 です。しかし，最初に被除数の値を 15 ビット，除数の値を 11 ビット右シフトしていたので，この値はもとの期待される計算結果と比較すると 1/16（4 ビット分）になります。そこで，この値の小数点位置を 4 ビット右にずらした値が商である 110.10101010101 となります。

商：6.66650390625　（110.10101010101）

(c) 剰余を求める

上記の演算結果をもとに，次の命令を行います。

```
R1 = R1 sra 1              ;1 ビット右シフト
IF ( R1 < 0 ) R1 ＋= R0    ;R1 が負の場合，除数(R0)を加算
R1 = R1 sra 15             ;15 ビット右シフト
```



R1 レジスタのビット 15-ビット 0 に剰余が格納されます。

0000 0000　0000 0000 0000 0000　0000 0000 1010 0000

小数点位置はビット 31 とビット 30 の間となり，0.000 0000 0000 0000 0000 0000 1010 0000 となります。被除数と小数点位置が同じになるように，この値を 15 ビット左シフトした値が剰余です。

剰余：0.002441406(0.000 0000 0101)

注意　除算命令における，数値フォーマットは厳密に決まっていません。

最初に仮定した小数点位置はあくまでも除数と被除数との位合わせを行うものです。除算は除数と被除数の相対的な比率を求めるものですので，最初に仮定した小数点位置は厳密な意味を持ちません。商に関する小数点の位置は，ループを n 回行った時点で，被除数レジスタのビット n－1 とビット n－2 の間，商はビット n からビット 0 となります。計算にあたっては，小数点の位置，被除数，除数の絶対値の大きさ，符号（負数表現）につねに注意しながら，数値の持つ意味を考慮して演算を行ってください。

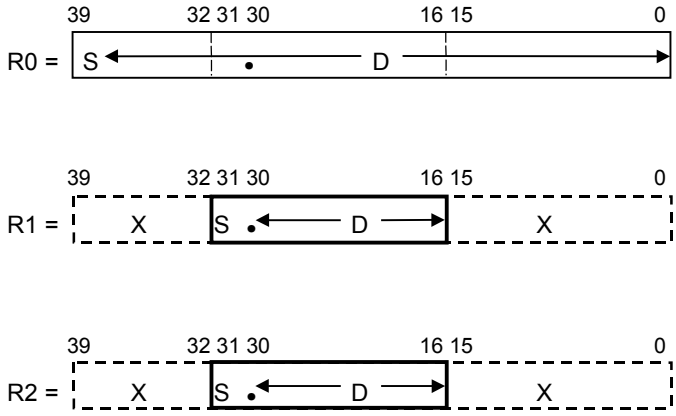
1.3　積和演算

ディジタル信号処理で多用する積和形式の演算に注目し，その記述を解説します。

積和の式とその展開式を次に示します。ここでは，0.1×0.2＋0.3×0.4＋0.5×0.6 **の計算を例に説明します。**

$$S=\sum_{i=0} ai \times bi = a_0 \times b_0 + a_1 \times b_1 + a_2 \times b_2 \cdots + a_n \times b_n$$

1.3.1　使用レジスタの説明



備考　S：符号，D：数値，X：無効，● ：小数点位置

1.3.2　使用レジスタ

(1) 汎用レジスタ

使用レジスタとその詳細は次のとおりです。

R0：**累積用**

R1：**乗算入力用**

R2：**乗算入力用**

(2) データ・ポインタ

2 つの独立したデータ・メモリに対して同時にアクセスが可能なため，乗算入力用の R1，R2 レジスタに対してそれぞれデータ・ポインタを用意します。

DP0：X データ・メモリ・アドレシング用
DP4：Y データ・メモリ・アドレシング用

1.3.3　入力条件

1.3.1 使用レジスタの説明で示すように，初回設定値を設定します。

R0 レジスタ ← 0 で初期化（累積用）
R1 レジスタ ← 0x000CCD0000（乗算用変数　X データ・メモリ:0.1）
R2 レジスタ ← 0x00199A0000（乗算用変数　Y データ・メモリ:0.2）

1.3.4　出力条件

演算結果は，R0 レジスタに格納します。

1.3.5　考え方

μPD77016 ファミリの命令からプログラムの記述に際して次の計画を立てます。

①積和の 1 項は 1 つの乗算とその結果の加算から構成
3 項演算のマルチプライ・アド（MADD）命令を使用します。
累算用レジスタに R0，乗算用レジスタに R1,R2 を使用します。

②並列ロード／ストア命令の同時記述
マルチプライ・アド命令では並列ロード／ストア命令が同時記述できます。
変数は 2 系列に分けそれぞれを X メモリ，Y メモリ上の配列として表現します。
また，並列ロード／ストア命令とともにデータ・ポインタのモディファイ（ポスト・インクリメント）を行います。

これらの結果から，積和の 1 項の処理は，1 インストラクション・サイクルで実行が可能となります。

1.3.6　処理手順

積和項数が多い場合は，次に示す④の前に REP 命令を記述し，④を繰り返す記述をすることにより，記述量およびオブジェクト量は飛躍的に削減できます。

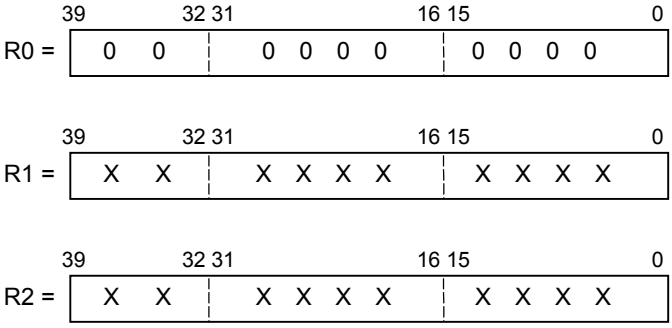
① データ・ポインタの初期化（X,Y メモリともに 0x0000 に設定）
② R0 レジスタのゼロ初期化
③ 変数の読み込みおよびポインタのインクリメント
④ 積和演算，変数の読み込みおよびポインタのインクリメントの同時記述

1.3.7　プログラム

```
;; ========================== ;;
;; ===== Sample Program ===== ;;
;; ========================== ;;

;; === Data ===

X_DATA XRAMSEG at 0x0000

DW  0.1                                                 ; 0x0CCD
DW  0.3                                                 ; 0x2666
DW  0.5                                                 ; 0x4000


Y_DATA YRAMSEG at 0x0000
DW  0.2                                                 ; 0x199A
DW  0.4                                                 ; 0x3333
DW  0.6                                                 ; 0X4CCD


;; === Program ===

Start imseg at 0x200

      jmp init                                          ;
      nop                                               ;

org 0x240

init:
      dp0 = 0x0000                                      ;
      dp4 = 0x0000                                      ;
      clr (r0)                                          ;

main:

      r1 = *dp0++  r2 = *dp4++                          ; No_1
      r0 = r0 + r1h * r2h  r1 = *dp0++ r2 = *dp4++      ; No_2
      r0 = r0 + r1h * r2h  r1 = *dp0++ r2 = *dp4++      ; No_3
      r0 = r0 + r1h * r2h                               ; No_4

end
```

1.3.8　実行結果

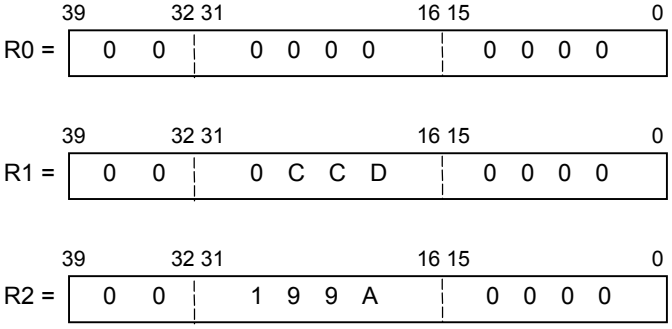
（1）初期状態



DP0 = 0x0000

DP4 = 0x0000

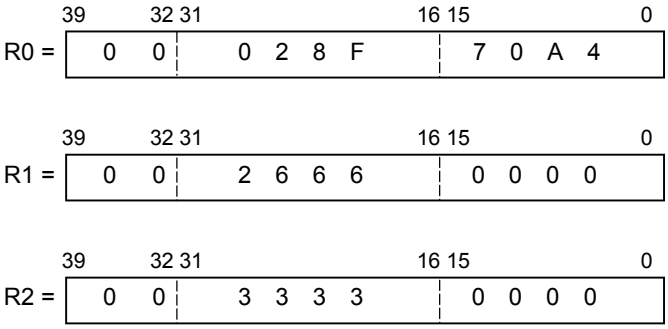
（2）No_1 を実行



DP0 = 0x0001

DP4 = 0x0001

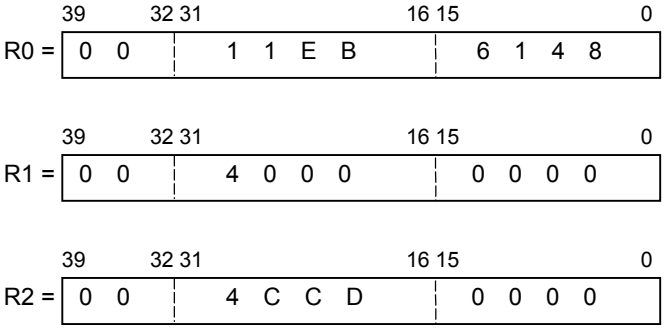
（3）No_2 を実行



DP0 = 0x0002

DP4 = 0x0002

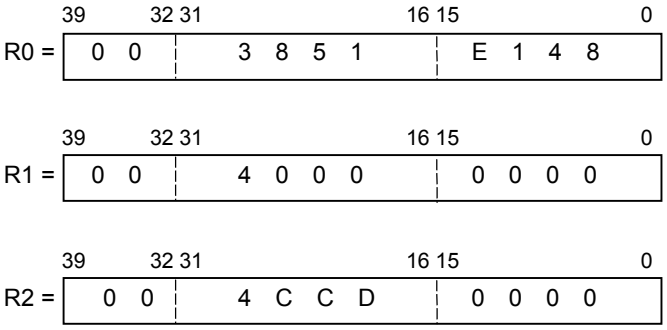
（4）No_3 を実行



DP0 = 0x0003

DP4 = 0x0003

（5）No_4 を実行（結果）



DP0 = 0x0003

DP4 = 0x0003

結果：0x3851E148（＋0.4399987794）

1.4　FIFO

μPD77016 **ファミリは** FIFO **バッファ（巡回バッファ）を実現する機構をハードウエアで装備し，アドレシングのオーバヘッドなしに構築することができます。**

FIFO **バッファの構築では，間接アドレシングのモジュロ・インデクス加算を使用し，リング・カウント動作させることで実現します。**

1.4.1　使用データ・メモリ・エリアの説明

リング・カウント動作するアドレス範囲は，データ・ポインタ（DP0-DP7）とインデクス・レジスタ（DN0-DN7）により決定します。

先頭アドレスは，モジュロ・レジスタの値をもとに次の数式により，データ・ポインタの下位（k－1）ビット（0基数）までを 0 した値となります。

$$2^{k-1} \leqq DMX / Y < 2^{k}$$

ここで，具体例を示してその先頭アドレスおよび最終アドレスを求めてみます。

例　データ・メモリ空間=X **メモリ**, DMX=0xA, DN0=3, DP0=0x15 **の場合**

使用範囲は，上記数式から k=4 となり，データ・ポインタ（0x15=0b00010101）のビット 0-ビット 3 を 0 にした値（0b00010000）つまり先頭アドレス=0x10，それにモジュロ・レジスタの値を加算した値，最終アドレス=0x1A となります。

1.4.2　使用データ・メモリ・エリア

X **データ・メモリ**:0x0000-0x000F
16 **ビット／データ**

1.4.3　使用レジスタ

(1) 汎用レジスタ

データのロード／ストアそれぞれに 1 つずつ，計 2 つを使用します。
使用レジスタとその詳細は次のとおりです。

R0：**ストア用**
R1：**ロード用**

(2) データ・ポインタ

読み込み，書き込みにそれぞれ独立したバッファ制御を行うため，同一メモリ空間上をアクセスするデータ・ポインタを 2 つ使用します。

DP0：**ストア・アドレシング用**
DP1：**ロード・アドレシング用**

(3) インデクス・レジスタ

使用するデータ・ポインタに対応し，2 つ使用します。

DN0：DP0 のモディファイ用
DN1：DP1 のモディファイ用

(4) モジュロ・レジスタ

使用するデータ・メモリ空間（X/Y データ・メモリ）により使用レジスタが決定されます。

DMX：X データ・メモリ

1.4.4　入力条件

データのストア用レジスタ R0=0x0 を初期値とし，1 回の書き込みごとに 1 インクリメント行います。

1.4.5　出力条件

データのロード用レジスタ R1 に読み込みます。

1.4.6　考え方

FIFO バッファの実現では次の特徴があります。

- FIFO では，データの読み出し，書き込みにそれぞれ独立したバッファ制御が必要になるため，データ・ポインタ(DP0-DP7)とインデクス・レジスタ(DN0-DN7)を対にした 2 組を用意します。

注意　同一メモリ空間（X／Y データ・メモリ）上にバッファを複数用意し使用する場合，加算する値と使用アドレス範囲に注意してください。

- リング・カウント動作するアドレス範囲の先頭アドレスは，データ・ポインタ（DP0-DP7）およびモジュロ・レジスタ(DMX, DMY)の値によって決定します。

注意　リング・カウント開始と使用アドレスの先頭アドレスは同一ではありません。

- リング・カウント動作するアドレス範囲の最終アドレスは，先頭アドレスにモジュロ・レジスタ（DMX, DMY）の値を加算した値になります。

- インデクス・レジスタ（DN0-DN7）の設定値は，モジュロ・レジスタ（DMX, DMY）の値より小さな値でなければなりません。

- モジュロ・インデクス加算では，モジュロ・レジスタ（DMX, DMY）に設定できる値は 0x0001-0x7FFF を範囲とした値になります。

1.4.7　処理手順

データのストア（書き込み），ロード（読み出し）は，それぞれマクロで定義します。
メモリ範囲は 0x0000-0x000F（バッファ・サイズ＝16 ワード），それぞれ X,Y データ・メモリに割り当てます。

① レジスタの初期化
② データのストア（16 回）
③ データのロード（8 回）
④ データのストア（8 回）

1.4.8　プログラム

```
;; ========================== ;;
;; ===== Sample Program ===== ;;
;; ========================== ;;

;; === Macro === ;;
%DEFINE (putfifo(p1)) (      ; P1 <- Data Pointer Register
       *dp0%% = p1;
)

%DEFINE (getfifo(p1)) (      ; P1 <- Data Pointer Register
       p1 = *dp1%%;
)

;; === Data === ;;
X_DATA XRAMSEG at 0x0000
DS 16                        ; 16Words
Y_DATA YRAMSEG at 0x0000
DS 16                        ; 16Words

;; === Program === ;;
Start imseg at 0x200

    jmp init                 ;
    nop                      ;

org 0x240

init:
   dmx = 0x0f                ;
   dp0 = 0x00                ; Put Pointer
   dn0 = 0x01                ; Put Strand
   dp1 = 0x00                ; Get Pointer
   dn1 = 0x01                ; Get Strand

   clr (r0)                  ; for Write
   clr (r1)                  ; for Read
                             ; No_1

main:                        ; Test of Put (#01-#16)
                             ; Whenever written data, increased
                             ; address of pointer
   %putfifo(r0l)             ; #01
   r0 = r0 + 1               ;
   %putfifo(r0l)             ; #02
   r0 = r0 + 1               ;
   %putfifo(r0l)             ; #03
   r0 = r0 + 1               ;
   %putfifo(r0l)             ; #04
   r0 = r0 + 1               ;
   %putfifo(r0l)             ; #05
   r0 = r0 + 1               ;
   %putfifo(r0l)             ; #06
   r0 = r0 + 1               ;
   %putfifo(r0l)             ; #07
   r0 = r0 + 1               ;
   %putfifo(r0l)             ; #08
   r0 = r0 + 1               ;
```

```
%putfifo(r0l)             ; #09
r0 = r0 + 1               ;
%putfifo(r0l)             ; #10
r0 = r0 + 1               ;
%putfifo(r0l)             ; #11
r0 = r0 + 1               ;
%putfifo(r0l)             ; #12
r0 = r0 + 1               ;
%putfifo(r0l)             ; #13
r0 = r0 + 1               ;
%putfifo(r0l)             ; #14
r0 = r0 + 1               ;
%putfifo(r0l)             ; #15
r0 = r0 + 1               ;
%putfifo(r0l)             ; #16
                          ; No_2

                          ; Test of Get (#01-#8)
%getfifo (r1l)            ; #01
%getfifo (r1l)            ; #02
%getfifo (r1l)            ; #03
%getfifo (r1l)            ; #04
%getfifo (r1l)            ; #05
%getfifo (r1l)            ; #06
%getfifo (r1l)            ; #07
%getfifo (r1l)            ; #08
                          ; No_3

                          ; Test of Put (#17-#24)
                          ; Whenever written data, increased
                          ; address of pointer
r0 = r0 + 1               ;
%putfifo(r0l)             ; #17
r0 = r0 + 1               ;
%putfifo(r0l)             ; #18
r0 = r0 + 1               ;
%putfifo(r0l)             ; #19
r0 = r0 + 1               ;
%putfifo(r0l)             ; #20
r0 = r0 + 1               ;
%putfifo(r0l)             ; #21
r0 = r0 + 1               ;
%putfifo(r0l)             ; #22
r0 = r0 + 1               ;
%putfifo(r0l)             ; #23
r0 = r0 + 1               ;
%putfifo(r0l)             ; #24
                          ; No_4

                          ; Test of Get (#09-#24)
%getfifo (r1l)            ; #09
%getfifo (r1l)            ; #10
%getfifo (r1l)            ; #11
%getfifo (r1l)            ; #12
%getfifo (r1l)            ; #13
%getfifo (r1l)            ; #14
%getfifo (r1l)            ; #15
%getfifo (r1l)            ; #16
%getfifo (r1l)            ; #17
%getfifo (r1l)            ; #18
%getfifo (r1l)            ; #19
```

```
    %getfifo (r11)              ; #20
    %getfifo (r11)              ; #21
    %getfifo (r11)              ; #22
    %getfifo (r11)              ; #23
    %getfifo (r11)              ; #24
                                ; NO_5

end
```

1.4.9　実行結果

（1）初期状態

R0 = 0x00

R1 = 0x00

DP0 = 0x0000

DP1 = 0x0000

（2）No_2 を実行

R0 = 0x0F

R1 = 0x00

DP0 = 0x0000

DP1 = 0x0000

（3）No_3 を実行

R0 = 0x0F

R1 = 0x07

DP0 = 0x0000

DP1 = 0x0008

（4）No_4 を実行（結果）

R0 = 0x17

R1 = 0x07

DP0 = 0x0008

DP1 = 0x0008

（5）No_5 を実行

R0 = 0x17

R1 = 0x17

DP0 = 0x0008

DP1 = 0x0008

結果　X **データ・メモリ**[0x0000-0x000F] =0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,0x08, 0x09,0x0A,0x0B, 0x0C, 0x0D, 0x0E, 0x0F

第 2 章　代表的なアプリケーション

代表的なアプリケーション例を次に示します。

2.1　スタートアップ・プログラム

μPD77016 ファミリのスタートアップ・プログラムで行う内容を次に示します。

- 初期化の必要なベクタを設定する。
- 初期化を必要とするレジスタ(SR：ステータス・レジスタなど)を正しく初期化する。
- 必要なメモリを初期化する。

次に，スタートアップ・プログラムの例を説明します。

2.1.1　条件

プログラム例の条件は，次のとおりです。

①内部割り込みは，SO1，SO2 を使用する。
②外部割り込みは，INT1 を RS-232C の送信エンプティによる割り込み，INT2 を RS-232C の受信レディによる割り込み，INT3 をタイマによる割り込みとして，それぞれ使用する。
③汎用入出力ポートはすべて出力に設定する。
④外部命令メモリはすべてノー・ウエイトでアクセスする。
⑤外部データ領域は，0x4000-0x7FFF:X　：システム I/O 領域で 7 ウエイト，0x4000-0x7FFF:Y　：ブート・データ領域で 7 ウエイト，それ以外の外部データ領域はノー・ウエイトでアクセスする。

注意　プログラム例では，割り込みルーチンの処理は記述していません。

2.1.2　プログラム

2.1.1　**条件によるスタートアップ・プログラム例を次に示します。**

【Sample.asm】

```
;;==================================
;;========= Sample Program =========
;;==================================

#define    PCD    *0x3805:Y
#define    DWTR   *0x3808:Y
#define    IWTR   *0x3809:Y

;==================================
;===== 汎用入出力ポートの設定======
;==================================

BIT_ENABLE  equ  0y1000000000000000
PORT_SET    equ  0y0100000000000000
PORT_RESET  equ  0y0000000000000000
MODE_ENABLE equ  0y0010000000000000
SET_INPUT   equ  0y0000000000000000
SET_OUTPUT  equ  0y0001000000000000
PORT_BIT3   equ  0y0000001100000000
PORT_BIT2   equ  0y0000001000000000
PORT_BIT1   equ  0y0000000100000000
PORT_BIT0   equ  0y0000000000000000
M3          equ  0y0000000000001000
M2          equ  0y0000000000000100
M1          equ  0y0000000000000010
M0          equ  0y0000000000000001

;==================================

public     ivReset
extrn      LeftCodecOut, RightCodecOut, BasicTimer, RS232Rx, RS232Tx

;==================================

VectSeg imseg   at  0x200

;==================================
                              ; Reset Vecter
ivReset:                      ; #0
       jmp StartUp            ;
       nop                    ;
       nop                    ;
       nop                    ;

;==================================
                              ; Reserved
ivPriv:                       ; #1 not used
       reti                   ;
       nop                    ;
       nop                    ;
       nop                    ;

;==================================
                              ; Reserved
```

```
ivAddr:                         ; #2 not used
       reti                     ;
       nop                      ;
       nop                      ;
       nop                      ;
;===================================
                                ; Reserved
ivSpare0:                       ; #3 not used
       reti                     ;
       nop                      ;
       nop                      ;
       nop                      ;

;===================================
                                ; INT1：RS232 Tx-EMPTY 割り込み
ivINT1:                         ; #4
       call RS232Tx             ;
       reti                     ;
       nop                      ;
       nop                      ;

;===================================
                                ; INT2：RS232 Rx-READY 割り込み
ivINT2:                         ; #5
       call RS232Rx             ;
       reti                     ;
       nop                      ;
       nop                      ;

;===================================
                                ; INT3：タイマ割り込み
ivINT3:                         ; #6
       call BasicTimer          ;
       reti                     ;
       nop                      ;
       nop                      ;

;===================================
                                ; INT4：
ivINT4:                         ; #7 not used
       reti                     ;
       nop                      ;
       nop                      ;
       nop                      ;

;===================================
                                ; オンチップ・シリアル入力
ivSI1:                          ; #8 not used
       reti                     ;
       nop                      ;
       nop                      ;
       nop                      ;

;===================================
                                ; オンチップ・シリアル出力
ivSO1:                          ; #9
       call LeftCodecOut        ;
       reti                     ;
       nop                      ;
       nop                      ;
```

```
;====================================
                              ; オンチップ・シリアル入力
ivSI2:                        ; #10 not used
       reti                   ;
       nop                    ;
       nop                    ;
       nop                    ;

;====================================
                              ; オンチップ・シリアル出力
ivSO2:                        ; #11
       call RightCodecOut     ;
       reti                   ;
       nop                    ;
       nop                    ;

;====================================
                              ; ホスト入力割り込み
ivHI:                         ; #12 not used
       reti                   ;
       nop                    ;
       nop                    ;
       nop                    ;

;====================================
                              ; ホスト出力割り込み
ivHO:                         ; #13
       reti                   ;
       nop                    ;
       nop                    ;
       nop                    ;

;====================================
                              ; Reserved
ivSarel1:                     ; #14 not used
       reti                   ;
       nop                    ;
       nop                    ;
       nop                    ;

;====================================
                              ; Reserved
ivSpare2:                     ; #15 not used
       reti                   ;
       nop                    ;
       nop                    ;
       nop                    ;

;====================================
;========== Register Clear =========
;====================================

StartUp:
       clr( r0 )              ;
       clr( r1 )              ;
       clr( r2 )              ;
       clr( r3 )              ;
       clr( r4 )              ;
       clr( r5 )              ;
       clr( r6 )              ;
       clr( r7 )              ;
```

```
;===================================
;======= User Stack Pointer =======
;===================================

       dp7 = 0x7ff             ;

;===================================
;========= Status Register =========
;===================================

       r0l = 0y0111111101011000;
       sr = r0L                ;
;===================================
;======== Port Setting =============
;===================================

       r0L = MODE_ENABLE | SET_OUTPUT | M0 | M1 | M2 | M3;
       PCD = r0L               ;
;===================================
;========== Wait Register ==========
;===================================

;============= IWTR ================
                               ; All area : No Wait
       r0L = 0x0000            ;
       IWTR = r0L              ;

;============= DWTR ================; area A(X:0000-3FFF)                    : No Wait
                               ; area B(X:4000-7FFF:MemoryMapped I/O )     : 7 Wait
       r0L = 0x0c0c            ; area C(X:8000-BFFF)                       : No Wait
       DWTR = r0L              ; area D(X:C000-FFFF)                       : No Wait
                               ; area E(Y:0000-3FFF)                       : No Wait
                               ; area F(Y:4000-7FFF:Boot ROM )             : 7 Wait
                               ; area G(Y:8000-BFFF)                       : No Wait
                               ; area H(Y:C000-FFFF)                       : No Wait
;===================================
;======== Internal RAM =============
;===================================

       r0L = 0                 ;
       dp0 = 0                 ;
       dp4 = 0                 ;

       rep 0x800               ;
         *dp0++ = r0L *dp4++ = r0L     ;

end
```

【Int.asm】

```
;;===================================
;;=========== Int Rutine ===========
;;===================================
public LeftCodecOut, RightCodecOut, BasicTimer, RS232Rx, RS232Tx

Start imseg

LeftCodecOut:
   nop;
   reti;

RightCodecOut:
```

```
    nop;
    reti;

BasicTimer:
    nop;
    reti;

RS232Rx:
    nop;
    reti;

RS232Tx:
    nop;
    reti;

end
```

2.1.3 解　　説

2.1.2 **プログラムについて説明します。**

(1) 割り込みルーチン

2.1.1 **条件に示すように，このシステムは内部／外部あわせて** 5 **つの割り込みを使用しています。それらは独立のルーチンとしてプログラミングできます。ですから，このプログラムでは記述せず，処理ルーチンの名前を単に** EXTRN**(外部)宣言にするにとどめています。したがって，それらのルーチンは，別途アセンブルしたうえでリンカにより結合します。**

(2) 未使用のベクタ

未使用のベクタは，設計上ではユーザの自由に使用できるプログラミング領域といえます。しかし，不測の事態にそなえてヌル・リターンやエラー・フォールトにするのが一般的な処理方法です。

(3) ユーザ・スタック

μPD77016 **ファミリでは，ユーザ・データをプッシュ／ポップするスタック機能を，ハードウエアでサポートしていません。したがって，その機能を必要とする場合，**DP**（データ・ポインタ）をスタック・ポインタとしたユーザ・スタックを構成する必要があります。プログラムでは，**DP7 **をスタック・ポインタとして使用し，内蔵** YRAM **の上位領域をスタックとして利用しています。**

(4) IWTR(Instruction Wait Cycle Register)，DWTR(Data Wait Cycle Register)

リセット時にブート動作がセルフ・ブートであれば，IWTR**，**DWTR **はブート・パラメータとして設定することができます。設定された内容が有効であれば，この段階での再設定は必要ありません。**

2.2　FIR フィルタ・プログラム

ディジタル・フィルタは，一般的に，インパルス応答特性から次の 2 つのタイプに分類されます。

- FIR (Finite Impulse Response)型

インパルスをあたえたとき，原理上，有限時間内にその応答が収束するタイプです。一般に，フィードバック・ループを持ちません[注1]。完全な直線位相特性を実現できるので，波形の忠実な伝送が求められる用途に応用されます。

- IIR (Infinite Impulse Response)型

インパルスをあたえたとき，原理上，その応答が永久[注2]に続くタイプです。フィードバック・ループを持ちます。伝達関数の極と零点を自由に配置可能です。そのため，比較的低次数のフィルタで，急峻な振幅遮断特性を実現できます。

注 1．特殊な例では，フィードバック・ループを持つ FIR フィルタも存在します。

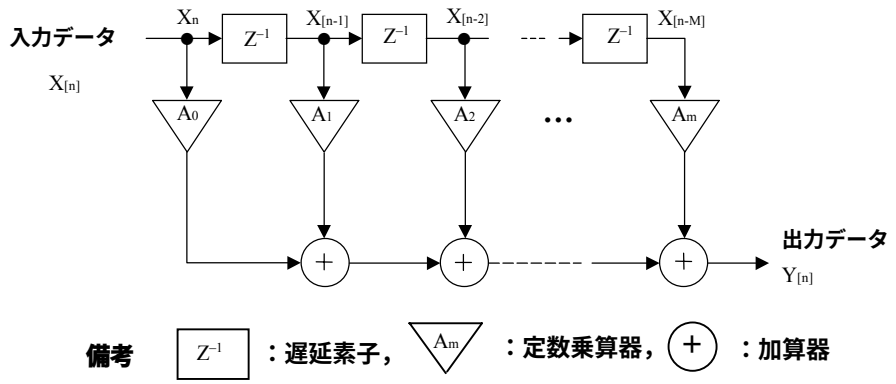
2．“ 永久 ”とは数学的な意味を示し，有限精度の現実の DSP では通常十分な時間が経過したあとに収束します。

ここでは，代表的な FIR フィルタの例を説明します。

2.2.1　フィルタの構成

このプログラム例による FIR フィルタ（32 タップ）の構成を次に示します。

図 2－1　FIR フィルタの構成



FIR フィルタの入力 X[n]と出力 Y[n]の差分方程式は次のようになります。

Am はフィルタの係数，M は FIR フィルタの次数を表します。この場合，M は 32 です。

$$Y[n] = \sum_{m=0}^{M} A_m X[n-m]$$

2.2.2　フィルタの規格

プログラム例のフィルタ規格の条件は，次のとおりです。

表 2－1　FIR フィルタの規格

フィルタ型			32 タップ FIR 型
サイズ	コード		14 ワード
	データ	X メモリ	34 ワード（メモリ：32 ワード，I/O：2 ワード）
		Y メモリ	33 ワード

2.2.3　使用データ・メモリ・エリアの説明

X データ・メモリ：入力信号の格納用

Y データ・メモリ：係数格納用

2.2.4　使用レジスタの説明

（1）汎用レジスタ

R0，R1，R2：演算用

（2）データ・ポインタ

DP0：入力信号の格納先へのポインタ

DP4：係数の格納先へのポインタ

2.2.5　プログラム

2.2.1　フィルタの構成の条件によるプログラム例を次に示します。

```
;;==================================
;;========= Sample Program =========
;;==================================

#define Order 32                    ;  次数の設定

;============ Y Memory =============

DATA1   XRAMSEG at 0x0000

;======================== Signal ===
Sig_top:
   DW  0x7FFF                       ;
   DW  0x4000,  0x2000, 0x1000,  0x6 ;
   DW  0x8,     0xA,    0xC,     0xE ;
   DW  0x10,    0x12,   0x14,    0x16;
   DW  0x18,    0x1A,   0x1C,    0x1E;
   DW  0x20,    0x22,   0x24,    0x26;
   DW  0x28,    0x2A,   0x2C,    0x2E;
   DW  0x30,    0x32,   0x34,    0x36;
   DW  0x38,    0x3A,   0x3C,    0x3E;
```

```
;=================== Output Port ===
Outdata:
   DS  1                            ;
;============ Y Memory =============

DATA2   YRAMSEG  at 0x0000

;=================== Coefficient ===
A_top:
   DW  0x4000, 0x2000, 0x1000, 0x800;
   DW  0x3,    0x4,    0x5,    0x6  ;
   DW  0x7,    0x8,    0x9,    0xA  ;
   DW  0xB,    0xC,    0xD,    0xE  ;
   DW  0xF,    0x10,   0x11,   0x12 ;
   DW  0x13,   0x14,   0x15,   0x16 ;
   DW  0x17,   0x18,   0x19,   0x1A ;
   DW  0x1B,   0x1C,   0x1D,   0x1E ;
   DW  0x1F                     ;

;==================================
;============ Program ==============
;==================================

Startup imseg   at  0x200

      jmp Start                  ;
      nop                        ;

;==================================

Main   imseg   at 0x240

Start:
      dp0 = Sig_top              ;  入力信号の設定
      dp4 = A_top                ;  係数の設定

      clr( r0 )                  ;  レジスタの初期化
      clr( r1 )                  ;
      clr( r2 )                  ;

;==================================

      r1 = *dp0++    r2 = *dp4++  ;

      rep Order+1                ;
      r0 = r0 + r1H * r2H  r1 = *dp0++  r2 = *dp4++;

;==================================

      dp1 = Outdata              ;
      nop                        ;
      *dp1 = r0H                 ;  データの保存

;==================================

      halt                       ;

;==================================

end
```

注意　このプログラムのフィルタ係数は有効なフィルタを構成する値ではありません。
実用的なフィルタを構成する場合，別途フィルタ係数を設定してください。

2.3　BIQUAD フィルタ・プログラム

IIR(Infinite Impulse Response)型フィルタでは，フィード・ループで伝達関数の極を構成し，極と零点を適切に配置することにより，低い次数で急峻なカットオフ極性のフィルタを構成することが可能となります。

一般に，M 次の IIR 型フィルタの入力 x[n]と出力 y[n]の差分方程式は次のようになります。

$$y[n] = \sum_{m=0}^{M} a_m y[n-m] + \sum_{k=0}^{K} b_k x[n-k]$$

伝達関数 H(z)は次の式で表されます。

$$H(Z) = \frac{\sum a_m Z^{-m}}{1 + \sum b_k Z^{-n}}$$

$$H(Z) = h_0 \times \frac{1 + a_1 \times Z^{-1} + a_2 \times Z^{-2} + \cdots + a_m \times Z^{-m}}{1 + b_1 \times Z^{-1} + b_2 \times Z^{-2} + \cdots + b_n \times Z^{-n}}$$

しかし，プログラミングにおいて，高い次数になるにしたがって安定度が悪くなったり，係数感度に敏感になるなど不利な場合があるため次の式に変形します。

$$H(Z) = h_0 \times \Pi \frac{1 + a_1 \times Z^{-1} + a_2 \times Z^{-2}}{1 + b_1 \times Z^{-1} + b_2 \times Z^{-2}}$$

上記の式から，分母分子がともに 2 次の伝達関数をもつ要素となるフィルタを K 個のカスケード接続することにより目的の伝達関数を実現できます。各フィルタを構成する要素フィルタをその伝達関数の形の名前から BIQUAD(BIQUADratic)：双 2 次形式フィルタと呼びます。

ここでは，基本的な BIQUAD フィルタについてプログラム例を使って説明します。

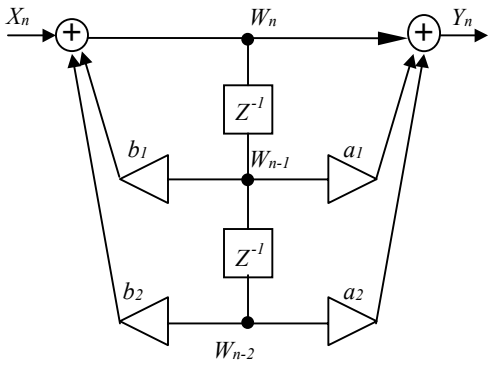
2.3.1　フィルタの構成

このプログラムの差分方程式を次に示します。また，フィルタの構成図を図 2－2 に示します。

$W_n = X_n + b_1 \times W_{n-1} + b_2 \times W_{n-2}$

$Y_n = W_n + a_1 \times W_{n-1} + a_2 \times W_{n-2}$

図 2－2　BIQUAD フィルタ（1 ステージ）の構成



備考　Z^{-1}：遅延素子，▷：定数乗算器，⊕：加算機

2.3.2　フィルタの規格

このプログラムのフィルタの規格を次に示します。

表 2－2　BIQUAD フィルタの規格

フィルタ型			1 ステージ BIQUAD 型
サイズ	コード		4 × 2 ワード（マクロ記述のみ）
	データ	X メモリ	4 ワード（メモリ：2 ワード，I/O：2 ワード）
		Y メモリ	4 ワード（メモリ：4 ワード）

2.3.3　使用データ・メモリ・エリアの説明

アドレスによるプログラムの依存性はありませんので，プログラム中では指定していません。

X データ・メモリ：W_n入出力ポート格納用
Y データ・メモリ：係数格納用

2.3.4　使用レジスタ

(1) 汎用レジスタ

使用するすべてのレジスタは固定小数点型 16 ビット・データ・フォーマットで使用しています。
使用レジスタとその詳細は次のとおりです。

R0：入出力信号
R1：中間ノードの値(W_{n-1})および演算結果
R2：係数
R3：中間ノードの値(W_{n-2})

(2) データ・ポインタ（DP0-DP7）

使用レジスタとその詳細は次のとおりです。

DP0：中間ノードの値(W_n, W_{n-1}, W_{n-2})へのポインタ
DP1：入/出力ポート(X_n, Y_n)へのポインタ
DP4：係数(a_1, a_2, b_1, b_2)へのポインタ

(3) インデクス・レジスタ(DN0-DN7)

W_nおよび係数を格納しているデータ・ポインタに使用します。
使用レジスタとその詳細は次のとおりです。

DN0：W_nおよび係数を格納しているデータへのポインタ

2.3.5　プログラム

1 ステージの演算はマクロで定義しています。

```
;; ========================== ;;
;; ===== Sample Program ===== ;;
;; ========================== ;;

;; === Macro === ;;
%DEFINE    (BIQUAD) (
       R0 = R0 + R1H*R2H   R3 = *DP0      R2 = *DP4++;
       R0 = R0 + R3H*R2H   *DP0-- = R1H    R2 = *DP4++;
       R0 = R0 + R1H*R2H   *DP0## = R0H    R2 = *DP4++;
       R0 = R0 + R3H*R2H   R1 = *DP0++     R2 = *DP4++;
)

;; === Data === ;;
DATA1 XRAMSEG
INDATA:     DW  0x400          ;
Wn_top:     DW  0x4000, 0x2000 ;
Outdata:    DS  1              ;

DATA2 YRAMSEG
Col_top:
b1:         DW  0x4000         ; b1
b2:         DW  0x2000         ; b2
a1:         DW  0x3000         ; a1
a2:         DW  0x1000         ; a2

;; === Program === ;;
Start imseg at 0x200

   jmp init                    ;
   nop                         ;

org 0x240

init:
   clr     (r0)                ;
   clr     (r1)                ;
   clr     (r2)                ;
   clr     (r3)                ;

main:
   dp1 = INDATA                ; Set Input val. address
   dp0 = Wn_top                ; Set Wn address
   dn0 = 2                     ;
   dp4 = Col_top               ;
   r0 = *dp1                   ;
   r1 = *dp0++    R2 = *dp4++  ;
   %BIQUAD                     ; 1 stage
   dp1 = Outdata               ;
   nop                         ;
   *dp1 = r0h                  ;

   halt                        ;

end
```

注意　このプログラムのフィルタ係数は有効なフィルタを構成する値ではありません。実用的なフィルタを構成する場合，別途フィルタ係数を設定してください。

★ 2.4　FFT プログラム

FFT（Fast Fourier Transform：高速フーリエ変換）とは，フーリエ変換をディジタル化した DFT（Discrete Fourier Transform：離散フーリエ変換）を高速に実行するアルゴリズムのことをいいます。現在では，実用的な DFT はほとんどの場合 FFT で実現され，多方面に応用されています。

2.4.1　FFT アルゴリズム

高速フーリエ変換（FFT：Fast Fourier Transform）は，離散フーリエ変換（DFT：Discrete Fourier Transform）を高速実行するアルゴリズムであり，ディジタル信号処理の基礎技術として広く用いられています。ここでは，始めに基本となる DFT について説明したあと，もっとも一般的に用いられている FFT のアルゴリズムについて説明します。

（1）DFT

DFT は，通常のフーリエ変換の無限区間積分を有限の和で置き換えたものであり，長さ N の DFT は次式で表されます。

$$F(k)=\sum_{n=0}^{N-1} f(n)W_N^{kn} \qquad \text{k = 0, 1, ... N-1} \qquad (1)$$

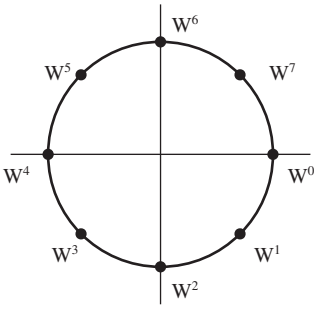
ここで，（1）式における W は回転子と呼ばれ，次式で表されます。

$$W_N = e^{-j2\pi/N} \qquad (2)$$

したがって，DFT は複素数演算であり，その計算回数は N^2 回の複素乗算と N(N-1)回の複素加算となります。

また，（2）式から分かるように，回転子 W の値は単位円上を N 等分した点を取ります。例として N=8 とした場合を図 2－3 に示します。

図 2－3　回転子



回転子の重要な性質として（3）から（6）の式を挙げます。これらの性質は図 2－3 において確認できます。

$$W_N^{k+N/2} = -W_N^k \qquad (3)$$

$$W_N^0 = 1 \qquad (4)$$

$$W_N^{N/4} = e^{-j\pi/2} = -j \qquad (5)$$

$$W_N^2 = e^{-j2\pi/N/2} = W_{N/2} \qquad (6)$$

（3）式は回転子の対称性を示しており，このため FFT に用いる回転子データは N/2 になります。また（4），（5）式より，回転子が 0 および N/4 になる場合には，乗算の必要がありません。これらはプログラミング時において計算量を減らすときに使用します。

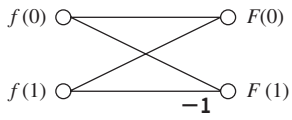
(2) FFT

FFT の基本的な考え方は，N 点の DFT をより小さい DFT に分けることです。たとえば，N 点の DFT を N/2 点の 2 つの DFT に分ければ，乗算回数は N^2 から $2\times(N/2)^2$ に減少します。そして，分けた N/2 の DFT をさらに半分の N/4 に・・・というように細かくし，最終的に 2 点の DFT に分けられます。この結果，乗算回数は（N/2）$\log_2 N$ にまで減少します。たとえば N=1024 の場合，約 100 万回の乗算が約 5000 回に減少します。このような考え方であるため，ポイント数は必然的に 2 のべき乗に限定されます（基数 2 の場合）。なお，2 点 DFT は（1）式および（4）式より，（7）式となります。

$$\begin{aligned} F(0) &= f(0) + f(1) \\ F(1) &= f(0) - f(1) \end{aligned} \qquad (7)$$

シグナル・フローでは図 2－4 のように表されます。この演算は，その形からバタフライ演算と呼ばれ，FFT はこのバタフライ演算の繰り返しで構成されます。

図 2－4　バタフライ演算



このことを式を用いて説明します。（1）式において，入力信号 f(n)に対し n が偶数の場合と奇数の場合で分けると，（8）のようになります。

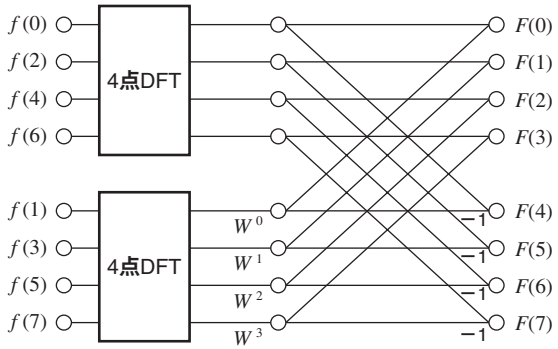
$$\begin{aligned} F(k) &= \sum_{n=0}^{N/2-1} f(2n) W_N^{2nk} + \sum_{n=0}^{N/2-1} f(2n+1) W_N^{(2n+1)k} \\ &= \sum_{n=0}^{N/2-1} f(2n) W_N^{2nk} + W_N^k \sum_{n=0}^{N/2-1} f(2n+1) W_N^{2nk} \end{aligned} \qquad (8)$$

ここで，（6）式を用いると（9）のようになります。

$$F(k)=\sum_{n=0}^{N/2-1} f(2n)W_{N/2}^{nk}+W_N^k\sum_{n=0}^{N/2-1} f(2n+1)W_{N/2}^{nk} \qquad (9)$$

（9）の式から，N 点 DFT が 2 つの N/2 点 DFT に分けられたことがわかります。ここで，DFT の周期性から各 N/2 点 DFT は N/2 を周期とする周期関数です。さらに，第 2 項には回転子が掛けられますが，（3）式より k=0,...,N/2-1 の場合には W_N^k，k=N/2,...,N-1 の場合には $-W_N^{k-N/2}$ となります。よって，k=m と k=m+N/2（ただし，m=0,...,N/2-1）では同じ絶対値を持ち符号の異なる回転子を用いて計算を行います。これは F(k)と F(k+N/2)のバタフライ演算を意味しています。N=8 として（9）式をシグナル・フローで表すと図 2－5 のようになります。

図 2－5　シグナル・フロー（第一段階）



さらに，同様にして，4 点 DFT を 2 つの 2 点 DFT に分けます。そして 2 点 DFT は図 2－4 に示されるようなバタフライ演算となることから，最終的な 8 点 FFT のシグナル・フローは図 2－6 になります。

図 2－6　シグナル・フロー（完成形）

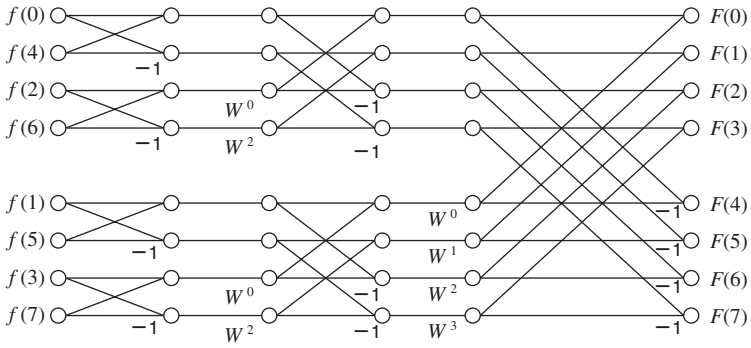


図 2－6 において注意することは，図 2－5 の 4 点 DFT の入力を間引くため入力信号の順番が変化していること（0-4-2-6,1-5-3-7）と，（9）式における各項の回転子の指数が 2nk であるため，図 2－5 における 4 点 DFT の回転子のアクセスが 0→2 となっていることです。この入力データの順番は，図 2－7 に示されるようにビット・リバースを行うことで求められるため，プロセッサによる実行に適しています。この場合，FFT 開始前にビット・リバースを行う必要があります。

図 2－7　ビット・リバース

0	000	000	0
1	001	100	4
2	010	010	2
3	011	→ 110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

このようなアルゴリズムは，入力信号 f(n)を分ける考え方であるため時間間引きと呼ばれます。これに対し，出力信号 F(k)を分けるアルゴリズムを周波数間引きと呼び，（10）の式で表されます。

$$F(2k) = \sum_{n=0}^{N/2-1} (f(n) + f(n+N/2)) W_{N/2}^{kn}$$
$$F(2k+1) = \sum_{n=0}^{N/2-1} ((f(n) - f(n+N/2)) W_N^n) W_{N/2}^{kn} \qquad (10)$$

（10）式より周波数間引きのバタフライ演算は，図 2－8 で表されます。周波数間引きは，時間間引きと比べ，回転子を掛ける順番が異なります。また，時間間引きと同様に（10）式はさらに細かな DFT に分けられます。N=8 を例とし，このシグナル・フローを図 2－9 に示します。図 2－9 において注意することは，
図 2－6 の時間間引きと比べて計算順序が逆であることと，ビット・リバース順の出力が得られることです。
したがって，このアルゴリズムでは，FFT 終了後にビット・リバースが必要です。

なお，時間間引き，周波数間引きともに，図に示される計算順序を逆にすることも可能です。この場合は，回転子をビット・リバース順にアクセスすることが必要です。以上より FFT は，時間間引きと周波数間引き，および入力データ整列型と出力データ整列型のそれぞれを組み合わせた 4 種類に大別できます。

図 2－8　周波数間引きのバタフライ演算

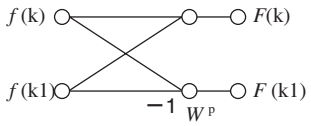
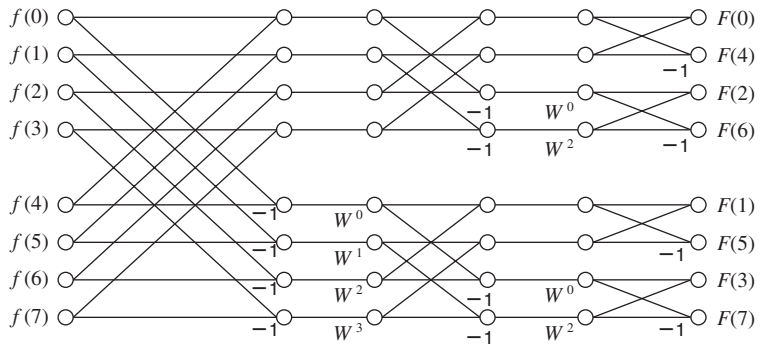


図 2－9　周波数間引きのシグナル・フロー



2.4.2　フィルタの規格

表 2－3　FFT プログラムのフィルタ規格

フィルタ型			基数 2，周波数間引き，入力データ整列型
ポイント数			8 - 1024 点（DEFINE.H にて指定）
サイズ	コード		76×2 ワード（セグメント名 FFTSeg のみ）
	データ	X データ	29 - 3585 ワード
		Y データ	28 - 3584 ワード

2.4.3　使用データ・メモリ・エリア

作業領域を含め，使用するデータの内容とそのアドレスを次に示します。

●X メモリ

内容	開始アドレス	終了アドレス	サイズ
実部データ格納領域	0x0000	0x0007-0x03FF	8-1024 ワード
実部入力値格納領域	0x0800	0x0807-0x0BFF	8-1024 ワード
実部出力値格納領域	0x2000	0x2007-0x23FF	8-1024 ワード
正弦データ格納領域	0x0400	0x0403-0x05FF	4-512 ワード
作業領域	−	−	1 ワード

●Y メモリ

内容	開始アドレス	終了アドレス	サイズ
虚部データ格納領域	0x0000	0x0007-0x03FF	8-1024 ワード
虚部入力値格納領域	0x0800	0x0807-0x0BFF	8-1024 ワード
虚部出力値格納領域	0x2000	0x2007-0x23FF	8-1024 ワード
余弦データ格納領域	0x0400	0x0403-0x05FF	4-512 ワード

2.4.4　プログラム・ファイル

このプログラムでは 3 つのファイルに分割して作成しています。

FFT.ASM　　**：メイン・プログラム・ファイル**

DEFINE.H　　**：インクルード・ファイル**

SINDAT.ASM　　**：正弦，余弦データ**

● FFT.ASM

```
------------------------------------------------------------------------------------------
#include "define.h"

EXTRN   SinData
EXTRN   CosData

/*---Memory ------------------*/
ReDataSeg XRAMSEG at 0x0000
      RealData: DS Point
ImDataSeg YRAMSEG at 0x0000
      ImagData: DS Point

InputSeg XRAMSEG at 0x800
      InputXData: DS Point
InputYSeg YRAMSEG at 0x800
      InputYData: DS Point

OutputSeg XRAMSEG at 0x2000
      OutputXData: DS Point
OutputYSeg YRAMSEG at 0x2000
      OutputYData: DS Point

XDataSeg XRAMSEG
        BLK_BUF: DS 1

/*-----------------------------*/
/*--    Interrupt Vector     --*/
/*-----------------------------*/
IntVect IMSEG at 0x200
ivReset:
      JMP StartUp;
      NOP;
```

```
/*-----------------------------*/
/*--        Main Proc        --*/
/*-----------------------------*/
FFTMain IMSEG
StartUp:
      CLR(R0);
      CLR(R1);
      CLR(R2);
      CLR(R3);
      CLR(R4);
      CLR(R5);
      CLR(R6);
      CLR(R7);
      CALL FFT;

WaitINT:
      NOP;
      NOP;
      JMP WaitINT;

/*-----------------------------*/
/*--    Data Input Proc      --*/
/*-----------------------------*/
InputDataSeg IMSEG
InputData:
      DP1 = RealData;
      DP5 = ImagData;
      DP0 = InputXData;
      DP4 = InputYData;
      LOOP Point {
             R1 = *DP0++       R2 = *DP4++;
             R1 = R1 SRA XX;           /* scaling */
             R2 = R2 SRA XX;
             *DP1++ = R1H      *DP5++ = R2H;
             };
             RET;

/*-----------------------------*/
/*--      Data Output Proc    --*/
/*-----------------------------*/
OutputDataSeg IMSEG
OutputData:
      DN1 = DN_DATA;
```

```
      DN5 = DN_DATA;
      DP1 = 0x0004;    /* Bit Reverse of OutputXData address */
      DP5 = 0x0004;    /* Bit Reverse of OutputYData address */
      DP0 = RealData;
      DP4 = ImagData;
      LOOP Point {
                  R1 = *DP0++          R2 = *DP4++;
                  *!DP1## = R1H        *!DP5## = R2H;
      };
      RET;

/*-----------------------------*/
/*--    FFT Main Proc        --*/
/*-----------------------------*/
FFTSeg IMSEG
FFT:
       CALL InputData;

       /* initialize */
       DP2 = BLK_BUF;
       R0L = 1;                /* ブロック数 */
       R7L = Point/2;          /* バタフライ演算回数 */
       *DP2 = R0L;             /* ブロック数をセーブ */

       /* start */
       LOOP XX-2               /* 最終段の 2 つ前まで */
       {
           DP0 = RealData;     /* Xr(k)のアドレス初期値 */
           R1 = R7 + RealData;
           DP1 = R1L;          /* Xr(k1)のアドレス初期値 */
           DP4 = ImagData;     /* Xi(k)のアドレス初期値 */
           DP5 = R1L;          /* Xi(k1)のアドレス初期値 */

           DN0 = R7L;
           DN1 = R7L;
           DN4 = R7L;
           DN5 = R7L;

           DN7 = R0L;
           DN3 = R0L;

           LOOP R0L
           {
```

```
        DP7 = CosData;
        DP3 = SinData;

        /* --- prologue of software pipeline --- */
        R1 = *DP0      R2 = *DP4;
        R3 = *DP1      R4 = *DP5;
        R5 = R1 + R3   R0 = *DP3##     R6 = *DP7##;
        R5 = R1 - R3   *DP0++ = R5H;
        R6 = R2 + R4   *DP1++ = R5H;
        R4 = R2 - R4   R1 = *DP0       *DP4++ = R6H;
        R0 = R7 - 1    R3 = *DP1       R2 = *DP4;

        LOOP R0L       /* バタフライ演算 */
        {
          R5 = R1 + R3                    *DP5++ = R4H;
          R5 = R1 - R3   *DP0++ = R5H     R4 = *DP5;  /* R5 : Xr(k) - Xr(k1) */
          R6 = R2 + R4    R0 = *DP3##     R3 = *DP7##;       /* R0 : Sin   R3 : Cos */

          R6 = R2 - R4    R1 = *DP0      *DP4++ = R6H;       /* R6 : Xi(k) - Xi(k1) */
          R2 = R3H * R5H;                                    /* Cos * ( Xr(k) - Xr(k1) ) */

          R2 = R2 + R0H * R6H;        /* Cos*(Xr(k) - Xr(k1)) + Sin*(Xi(k) - Xi(k1)) */

          R4 = R3H * R6H  *DP1++ = R2H    R2 = *DP4;   /* Cos * ( Xi(k) - Xi(k1) ) */
          R4 = R4 - R0H * R5H  R3 = *DP1;  /* Cos*(Xi(k) - Xi(k1)) - Sin*(Xr(k) - Xr(k1)) */
         };
          /* --- epilogue of software pipeline --- */
          *DP5++ = R4H;

          R1 = *DP0##  R2 = *DP4##;
          R1 = *DP1##  R2 = *DP5##;
        };
        R0L = *DP2;
        R0 = R0 SLL 0x1;
        *DP2 = R0L;
        R7 = R7 SRL 0x1;
};

/* 最終段の 1 つ前 */
DP0 = RealData;
DP1 = RealData + 2;
DP4 = ImagData;
DP5 = ImagData + 2;
```

```
        DN0 = 0x3;
        DN1 = 0x3;
        DN4 = 0x3;
        DN5 = 0x3;

        R1 = *DP0;
        R3 = *DP1       R4 = *DP5;
        LOOP Point/4
        {
                R5 = R1 + R3    R2 = *DP4;
                R6 = R1 - R3    *DP0++ = R5H;
                R5 = R2 + R4    *DP1++ = R6H;
                R6 = R2 - R4    R1 = *DP0           *DP4++ = R5H;
                R3 = *DP1       *DP5++ = R6H;

                R5 = R1 + R3    R2 = *DP4;
                R5 = R3 - R1    *DP0## = R5H        R4 = *DP5;
                R6 = R2 - R4    R1 = *DP0           *DP5## = R5H;
                R6 = R2 + R4    *DP1## = R6H        R4 = *DP5;
                R3 = *DP1       *DP4## = R6H;
        };

        /* 最終段 */
        DP0 = RealData;
        DP4 = ImagData;

        R1 = *DP0++;
        R3 = *DP0--;
        LOOP Point/2
        {
                R5 = R1 + R3    R2 = *DP4++;
                R6 = R1 - R3    *DP0++ = R5H        R4 = *DP4--;
                R5 = R2 + R4    *DP0++ = R6H;
                R6 = R2 - R4    R1 = *DP0++         *DP4++ = R5H;
                R3 = *DP0--     *DP4++ = R6H;
        };

        CALL OutputData;

        RET;
        END
```

● DEFINE.H

```
------------------------------------------------------------------------------------------------
;;====================================;;
;;===== Sample Program (DEFINE.H) =====;;
;;====================================;;
#define XX 10    /* Point =2^XX ( XX = 3〜10) */

#if(XX ==3)
#define Point 8
#define DN_DATA 0x2000
#endif

#if(XX ==4)
#define Point 16
#define DN_DATA 0x1000
#endif

#if(XX ==5)
#define Point 32
#define DN_DATA 0x0800
#endif

#if(XX ==6)
#define Point 64
#define DN_DATA 0x0400
#endif

#if(XX ==7)
#define Point 128
#define DN_DATA 0x0200
#endif

#if(XX ==8)
#define Point 256
#define DN_DATA 0x0100
#endif

#if(XX ==9)
#define Point 512
#define DN_DATA 0x0080
#endif

#if(XX ==10)
```

```
#define Point 1024
#define DN_DATA 0x0040
#endif
```

●SINDAT.ASM

```
;; ===================================== ;;
;; ===== Sample Program (SINDAT.ASM)===== ;;
;; ===================================== ;;
#include "define.h"

PUBLIC  SinData
PUBLIC  CosData

SinDataSeg     XRAMSEG at 0x0400
SinData:
#if(XX == 3)
   DW     0x0000, 0x5A81, 0x7FFF, 0x5A81
#endif

#if(XX == 4)
   DW     0x0000, 0x30FB, 0x5A81, 0x7640, 0x7FFF, 0x7640, 0x5A81, 0x30FB
#endif

#if(XX == 5)
   DW     0x0000, 0x18F8, 0x30FB, 0x471C, 0x5A81, 0x6A6C, 0x7640, 0x7D89
   DW     0x7FFF, 0x7D89, 0x7640, 0x6A6C, 0x5A81, 0x471C, 0x30FB, 0x18F8
#endif

#if(XX == 6)
   DW     0x0000, 0x0C8B, 0x18F8, 0x2527, 0x30FB, 0x3C56, 0x471C, 0x5133
   DW     0x5A81, 0x62F1, 0x6A6C, 0x70E1, 0x7640, 0x7A7C, 0x7D89, 0x7F61
   DW     0x7FFF, 0x7F61, 0x7D89, 0x7A7C, 0x7640, 0x70E1, 0x6A6C, 0x62F1
   DW     0x5A81, 0x5133, 0x471C, 0x3C56, 0x30FB, 0x2527, 0x18F8, 0x0C8B
#endif

#if(XX == 7)
   DW     0x0000, 0x0647, 0x0C8B, 0x12C7, 0x18F8, 0x1F19, 0x2527, 0x2B1E
   DW     0x30FB, 0x36B9, 0x3C56, 0x41CD, 0x471C, 0x4C3F, 0x5133, 0x55F4
   DW     0x5A81, 0x5ED6, 0x62F1, 0x66CE, 0x6A6C, 0x6DC9, 0x70E1, 0x73B5
   DW     0x7640, 0x7883, 0x7A7C, 0x7C29, 0x7D89, 0x7E9C, 0x7F61, 0x7FD7
   DW     0x7FFF, 0x7FD7, 0x7F61, 0x7E9C, 0x7D89, 0x7C29, 0x7A7C, 0x7883
```

```
    DW      0x7640, 0x73B5, 0x70E1, 0x6DC9, 0x6A6C, 0x66CE, 0x62F1, 0x5ED6
    DW      0x5A81, 0x55F4, 0x5133, 0x4C3F, 0x471C, 0x41CD, 0x3C56, 0x36B9
    DW      0x30FB, 0x2B1E, 0x2527, 0x1F19, 0x18F8, 0x12C7, 0x0C8B, 0x0647
#endif

#if(XX == 8)
    DW      0x0000, 0x0324, 0x0647, 0x096A, 0x0C8B, 0x0FAB, 0x12C7, 0x15E1
    DW      0x18F8, 0x1C0B, 0x1F19, 0x2223, 0x2527, 0x2826, 0x2B1E, 0x2E10
    DW      0x30FB, 0x33DE, 0x36B9, 0x398C, 0x3C56, 0x3F16, 0x41CD, 0x447A
    DW      0x471C, 0x49B3, 0x4C3F, 0x4EBF, 0x5133, 0x539A, 0x55F4, 0x5842
    DW      0x5A81, 0x5CB3, 0x5ED6, 0x60EB, 0x62F1, 0x64E7, 0x66CE, 0x68A5
    DW      0x6A6C, 0x6C23, 0x6DC9, 0x6F5E, 0x70E1, 0x7254, 0x73B5, 0x7503
    DW      0x7640, 0x776B, 0x7883, 0x7989, 0x7A7C, 0x7B5C, 0x7C29, 0x7CE2
    DW      0x7D89, 0x7E1C, 0x7E9C, 0x7F08, 0x7F61, 0x7FA6, 0x7FD7, 0x7FF5
    DW      0x7FFF, 0x7FF5, 0x7FD7, 0x7FA6, 0x7F61, 0x7F08, 0x7E9C, 0x7E1C
    DW      0x7D89, 0x7CE2, 0x7C29, 0x7B5C, 0x7A7C, 0x7989, 0x7883, 0x776B
    DW      0x7640, 0x7503, 0x73B5, 0x7254, 0x70E1, 0x6F5E, 0x6DC9, 0x6C23
    DW      0x6A6C, 0x68A5, 0x66CE, 0x64E7, 0x62F1, 0x60EB, 0x5ED6, 0x5CB3
    DW      0x5A81, 0x5842, 0x55F4, 0x539A, 0x5133, 0x4EBF, 0x4C3F, 0x49B3
    DW      0x471C, 0x447A, 0x41CD, 0x3F16, 0x3C56, 0x398C, 0x36B9, 0x33DE
    DW      0x30FB, 0x2E10, 0x2B1E, 0x2826, 0x2527, 0x2223, 0x1F19, 0x1C0B
    DW      0x18F8, 0x15E1, 0x12C7, 0x0FAB, 0x0C8B, 0x096A, 0x0647, 0x0324
#endif

 #if(XX == 9)
    DW      0x0000, 0x0192, 0x0324, 0x04B6, 0x0647, 0x07D9, 0x096A, 0x0AFB
    DW      0x0C8B, 0x0E1B, 0x0FAB, 0x1139, 0x12C7, 0x1455, 0x15E1, 0x176D
    DW      0x18F8, 0x1A82, 0x1C0B, 0x1D93, 0x1F19, 0x209F, 0x2223, 0x23A6
    DW      0x2527, 0x26A7, 0x2826, 0x29A3, 0x2B1E, 0x2C98, 0x2E10, 0x2F86
    DW      0x30FB, 0x326D, 0x33DE, 0x354D, 0x36B9, 0x3824, 0x398C, 0x3AF2
    DW      0x3C56, 0x3DB7, 0x3F16, 0x4073, 0x41CD, 0x4325, 0x447A, 0x45CC
    DW      0x471C, 0x4869, 0x49B3, 0x4AFA, 0x4C3F, 0x4D80, 0x4EBF, 0x4FFA
    DW      0x5133, 0x5268, 0x539A, 0x54C9, 0x55F4, 0x571D, 0x5842, 0x5963
    DW      0x5A81, 0x5B9C, 0x5CB3, 0x5DC6, 0x5ED6, 0x5FE2, 0x60EB, 0x61F0
    DW      0x62F1, 0x63EE, 0x64E7, 0x65DD, 0x66CE, 0x67BC, 0x68A5, 0x698B
    DW      0x6A6C, 0x6B4A, 0x6C23, 0x6CF8, 0x6DC9, 0x6E95, 0x6F5E, 0x7022
    DW      0x70E1, 0x719D, 0x7254, 0x7306, 0x73B5, 0x745E, 0x7503, 0x75A4
    DW      0x7640, 0x76D8, 0x776B, 0x77F9, 0x7883, 0x7908, 0x7989, 0x7A04
    DW      0x7A7C, 0x7AEE, 0x7B5C, 0x7BC4, 0x7C29, 0x7C88, 0x7CE2, 0x7D38
    DW      0x7D89, 0x7DD5, 0x7E1C, 0x7E5E, 0x7E9C, 0x7ED4, 0x7F08, 0x7F37
    DW      0x7F61, 0x7F86, 0x7FA6, 0x7FC1, 0x7FD7, 0x7FE8, 0x7FF5, 0x7FFC
    DW      0x7FFF, 0x7FFC, 0x7FF5, 0x7FE8, 0x7FD7, 0x7FC1, 0x7FA6, 0x7F86
    DW      0x7F61, 0x7F37, 0x7F08, 0x7ED4, 0x7E9C, 0x7E5E, 0x7E1C, 0x7DD5
```

```
    DW      0x7D89, 0x7D38, 0x7CE2, 0x7C88, 0x7C29, 0x7BC4, 0x7B5C, 0x7AEE
    DW      0x7A7C, 0x7A04, 0x7989, 0x7908, 0x7883, 0x77F9, 0x776B, 0x76D8
    DW      0x7640, 0x75A4, 0x7503, 0x745E, 0x73B5, 0x7306, 0x7254, 0x719D
    DW      0x70E1, 0x7022, 0x6F5E, 0x6E95, 0x6DC9, 0x6CF8, 0x6C23, 0x6B4A
    DW      0x6A6C, 0x698B, 0x68A5, 0x67BC, 0x66CE, 0x65DD, 0x64E7, 0x63EE
    DW      0x62F1, 0x61F0, 0x60EB, 0x5FE2, 0x5ED6, 0x5DC6, 0x5CB3, 0x5B9C
    DW      0x5A81, 0x5963, 0x5842, 0x571D, 0x55F4, 0x54C9, 0x539A, 0x5268
    DW      0x5133, 0x4FFA, 0x4EBF, 0x4D80, 0x4C3F, 0x4AFA, 0x49B3, 0x4869
    DW      0x471C, 0x45CC, 0x447A, 0x4325, 0x41CD, 0x4073, 0x3F16, 0x3DB7
    DW      0x3C56, 0x3AF2, 0x398C, 0x3824, 0x36B9, 0x354D, 0x33DE, 0x326D
    DW      0x30FB, 0x2F86, 0x2E10, 0x2C98, 0x2B1E, 0x29A3, 0x2826, 0x26A7
    DW      0x2527, 0x23A6, 0x2223, 0x209F, 0x1F19, 0x1D93, 0x1C0B, 0x1A82
    DW      0x18F8, 0x176D, 0x15E1, 0x1455, 0x12C7, 0x1139, 0x0FAB, 0x0E1B
    DW      0x0C8B, 0x0AFB, 0x096A, 0x07D9, 0x0647, 0x04B6, 0x0324, 0x0192
#endif

#if(XX == 10)
    DW      0x0000, 0x00C9, 0x0192, 0x025B, 0x0324, 0x03ED, 0x04B6, 0x057E
    DW      0x0647, 0x0710, 0x07D9, 0x08A1, 0x096A, 0x0A32, 0x0AFB, 0x0BC3
    DW      0x0C8B, 0x0D53, 0x0E1B, 0x0EE3, 0x0FAB, 0x1072, 0x1139, 0x1200
    DW      0x12C7, 0x138E, 0x1455, 0x151B, 0x15E1, 0x16A7, 0x176D, 0x1833
    DW      0x18F8, 0x19BD, 0x1A82, 0x1B46, 0x1C0B, 0x1CCF, 0x1D93, 0x1E56
    DW      0x1F19, 0x1FDC, 0x209F, 0x2161, 0x2223, 0x22E4, 0x23A6, 0x2467
    DW      0x2527, 0x25E7, 0x26A7, 0x2767, 0x2826, 0x28E5, 0x29A3, 0x2A61
    DW      0x2B1E, 0x2BDB, 0x2C98, 0x2D54, 0x2E10, 0x2ECC, 0x2F86, 0x3041
    DW      0x30FB, 0x31B4, 0x326D, 0x3326, 0x33DE, 0x3496, 0x354D, 0x3603
    DW      0x36B9, 0x376F, 0x3824, 0x38D8, 0x398C, 0x3A3F, 0x3AF2, 0x3BA4
    DW      0x3C56, 0x3D07, 0x3DB7, 0x3E67, 0x3F16, 0x3FC5, 0x4073, 0x4120
    DW      0x41CD, 0x4279, 0x4325, 0x43D0, 0x447A, 0x4523, 0x45CC, 0x4674
    DW      0x471C, 0x47C3, 0x4869, 0x490E, 0x49B3, 0x4A57, 0x4AFA, 0x4B9D
    DW      0x4C3F, 0x4CE0, 0x4D80, 0x4E20, 0x4EBF, 0x4F5D, 0x4FFA, 0x5097
    DW      0x5133, 0x51CE, 0x5268, 0x5301, 0x539A, 0x5432, 0x54C9, 0x555F
    DW      0x55F4, 0x5689, 0x571D, 0x57B0, 0x5842, 0x58D3, 0x5963, 0x59F3
    DW      0x5A81, 0x5B0F, 0x5B9C, 0x5C28, 0x5CB3, 0x5D3D, 0x5DC6, 0x5E4F
    DW      0x5ED6, 0x5F5D, 0x5FE2, 0x6067, 0x60EB, 0x616E, 0x61F0, 0x6271
    DW      0x62F1, 0x6370, 0x63EE, 0x646B, 0x64E7, 0x6562, 0x65DD, 0x6656
    DW      0x66CE, 0x6745, 0x67BC, 0x6831, 0x68A5, 0x6919, 0x698B, 0x69FC
    DW      0x6A6C, 0x6ADB, 0x6B4A, 0x6BB7, 0x6C23, 0x6C8E, 0x6CF8, 0x6D61
    DW      0x6DC9, 0x6E30, 0x6E95, 0x6EFA, 0x6F5E, 0x6FC0, 0x7022, 0x7082
    DW      0x70E1, 0x7140, 0x719D, 0x71F9, 0x7254, 0x72AE, 0x7306, 0x735E
    DW      0x73B5, 0x740A, 0x745E, 0x74B1, 0x7503, 0x7554, 0x75A4, 0x75F3
    DW      0x7640, 0x768D, 0x76D8, 0x7722, 0x776B, 0x77B3, 0x77F9, 0x783F
    DW      0x7883, 0x78C6, 0x7908, 0x7949, 0x7989, 0x79C7, 0x7A04, 0x7A41
```

```
    DW      0x7A7C, 0x7AB5, 0x7AEE, 0x7B25, 0x7B5C, 0x7B91, 0x7BC4, 0x7BF7
    DW      0x7C29, 0x7C59, 0x7C88, 0x7CB6, 0x7CE2, 0x7D0E, 0x7D38, 0x7D61
    DW      0x7D89, 0x7DB0, 0x7DD5, 0x7DF9, 0x7E1C, 0x7E3E, 0x7E5E, 0x7E7E
    DW      0x7E9C, 0x7EB9, 0x7ED4, 0x7EEF, 0x7F08, 0x7F20, 0x7F37, 0x7F4C
    DW      0x7F61, 0x7F74, 0x7F86, 0x7F96, 0x7FA6, 0x7FB4, 0x7FC1, 0x7FCD
    DW      0x7FD7, 0x7FE0, 0x7FE8, 0x7FEF, 0x7FF5, 0x7FF9, 0x7FFC, 0x7FFE
    DW      0x7FFF, 0x7FFE, 0x7FFC, 0x7FF9, 0x7FF5, 0x7FEF, 0x7FE8, 0x7FE0
    DW      0x7FD7, 0x7FCD, 0x7FC1, 0x7FB4, 0x7FA6, 0x7F96, 0x7F86, 0x7F74
    DW      0x7F61, 0x7F4C, 0x7F37, 0x7F20, 0x7F08, 0x7EEF, 0x7ED4, 0x7EB9
    DW      0x7E9C, 0x7E7E, 0x7E5E, 0x7E3E, 0x7E1C, 0x7DF9, 0x7DD5, 0x7DB0
    DW      0x7D89, 0x7D61, 0x7D38, 0x7D0E, 0x7CE2, 0x7CB6, 0x7C88, 0x7C59
    DW      0x7C29, 0x7BF7, 0x7BC4, 0x7B91, 0x7B5C, 0x7B25, 0x7AEE, 0x7AB5
    DW      0x7A7C, 0x7A41, 0x7A04, 0x79C7, 0x7989, 0x7949, 0x7908, 0x78C6
    DW      0x7883, 0x783F, 0x77F9, 0x77B3, 0x776B, 0x7722, 0x76D8, 0x768D
    DW      0x7640, 0x75F3, 0x75A4, 0x7554, 0x7503, 0x74B1, 0x745E, 0x740A
    DW      0x73B5, 0x735E, 0x7306, 0x72AE, 0x7254, 0x71F9, 0x719D, 0x7140
    DW      0x70E1, 0x7082, 0x7022, 0x6FC0, 0x6F5E, 0x6EFA, 0x6E95, 0x6E30
    DW      0x6DC9, 0x6D61, 0x6CF8, 0x6C8E, 0x6C23, 0x6BB7, 0x6B4A, 0x6ADB
    DW      0x6A6C, 0x69FC, 0x698B, 0x6919, 0x68A5, 0x6831, 0x67BC, 0x6745
    DW      0x66CE, 0x6656, 0x65DD, 0x6562, 0x64E7, 0x646B, 0x63EE, 0x6370
    DW      0x62F1, 0x6271, 0x61F0, 0x616E, 0x60EB, 0x6067, 0x5FE2, 0x5F5D
    DW      0x5ED6, 0x5E4F, 0x5DC6, 0x5D3D, 0x5CB3, 0x5C28, 0x5B9C, 0x5B0F
    DW      0x5A81, 0x59F3, 0x5963, 0x58D3, 0x5842, 0x57B0, 0x571D, 0x5689
    DW      0x55F4, 0x555F, 0x54C9, 0x5432, 0x539A, 0x5301, 0x5268, 0x51CE
    DW      0x5133, 0x5097, 0x4FFA, 0x4F5D, 0x4EBF, 0x4E20, 0x4D80, 0x4CE0
    DW      0x4C3F, 0x4B9D, 0x4AFA, 0x4A57, 0x49B3, 0x490E, 0x4869, 0x47C3
    DW      0x471C, 0x4674, 0x45CC, 0x4523, 0x447A, 0x43D0, 0x4325, 0x4279
    DW      0x41CD, 0x4120, 0x4073, 0x3FC5, 0x3F16, 0x3E67, 0x3DB7, 0x3D07
    DW      0x3C56, 0x3BA4, 0x3AF2, 0x3A3F, 0x398C, 0x38D8, 0x3824, 0x376F
    DW      0x36B9, 0x3603, 0x354D, 0x3496, 0x33DE, 0x3326, 0x326D, 0x31B4
    DW      0x30FB, 0x3041, 0x2F86, 0x2ECC, 0x2E10, 0x2D54, 0x2C98, 0x2BDB
    DW      0x2B1E, 0x2A61, 0x29A3, 0x28E5, 0x2826, 0x2767, 0x26A7, 0x25E7
    DW      0x2527, 0x2467, 0x23A6, 0x22E4, 0x2223, 0x2161, 0x209F, 0x1FDC
    DW      0x1F19, 0x1E56, 0x1D93, 0x1CCF, 0x1C0B, 0x1B46, 0x1A82, 0x19BD
    DW      0x18F8, 0x1833, 0x176D, 0x16A7, 0x15E1, 0x151B, 0x1455, 0x138E
    DW      0x12C7, 0x1200, 0x1139, 0x1072, 0x0FAB, 0x0EE3, 0x0E1B, 0x0D53
    DW      0x0C8B, 0x0BC3, 0x0AFB, 0x0A32, 0x096A, 0x08A1, 0x07D9, 0x0710
    DW      0x0647, 0x057E, 0x04B6, 0x03ED, 0x0324, 0x025B, 0x0192, 0x00C9
#endif



CosDataSeg    YRAMSEG at 0x0400

CosData:
```

```
#if(XX == 3)
    DW      0x7FFF, 0x5A81, 0x0000, 0xA57F
#endif

#if(XX == 4)
    DW      0x7FFF, 0x7640, 0x5A81, 0x30FB, 0x0000, 0xCF05, 0xA57F, 0x89C0
#endif

#if(XX == 5)
    DW      0x7FFF, 0x7D89, 0x7640, 0x6A6C, 0x5A81, 0x471C, 0x30FB, 0x18F8
    DW      0x0000, 0xE708, 0xCF05, 0xB8E4, 0xA57F, 0x9594, 0x89C0, 0x8277
#endif

#if(XX == 6)
    DW      0x7FFF, 0x7F61, 0x7D89, 0x7A7C, 0x7640, 0x70E1, 0x6A6C, 0x62F1
    DW      0x5A81, 0x5133, 0x471C, 0x3C56, 0x30FB, 0x2527, 0x18F8, 0x0C8B
    DW      0x0000, 0xF375, 0xE708, 0xDAD9, 0xCF05, 0xC3AA, 0xB8E4, 0xAECD
    DW      0xA57F, 0x9D0F, 0x9594, 0x8F1F, 0x89C0, 0x8584, 0x8277, 0x809F
#endif

#if(XX == 7)
    DW      0x7FFF, 0x7FD7, 0x7F61, 0x7E9C, 0x7D89, 0x7C29, 0x7A7C, 0x7883
    DW      0x7640, 0x73B5, 0x70E1, 0x6DC9, 0x6A6C, 0x66CE, 0x62F1, 0x5ED6
    DW      0x5A81, 0x55F4, 0x5133, 0x4C3F, 0x471C, 0x41CD, 0x3C56, 0x36B9
    DW      0x30FB, 0x2B1E, 0x2527, 0x1F19, 0x18F8, 0x12C7, 0x0C8B, 0x0647
    DW      0x0000, 0xF9B9, 0xF375, 0xED39, 0xE708, 0xE0E7, 0xDAD9, 0xD4E2
    DW      0xCF05, 0xC947, 0xC3AA, 0xBE33, 0xB8E4, 0xB3C1, 0xAECD, 0xAA0C
    DW      0xA57F, 0xA12A, 0x9D0F, 0x9932, 0x9594, 0x9237, 0x8F1F, 0x8C4B
    DW      0x89C0, 0x877D, 0x8584, 0x83D7, 0x8277, 0x8164, 0x809F, 0x8029
#endif

#if(XX == 8)
    DW      0x7FFF, 0x7FF5, 0x7FD7, 0x7FA6, 0x7F61, 0x7F08, 0x7E9C, 0x7E1C
    DW      0x7D89, 0x7CE2, 0x7C29, 0x7B5C, 0x7A7C, 0x7989, 0x7883, 0x776B
    DW      0x7640, 0x7503, 0x73B5, 0x7254, 0x70E1, 0x6F5E, 0x6DC9, 0x6C23
    DW      0x6A6C, 0x68A5, 0x66CE, 0x64E7, 0x62F1, 0x60EB, 0x5ED6, 0x5CB3
    DW      0x5A81, 0x5842, 0x55F4, 0x539A, 0x5133, 0x4EBF, 0x4C3F, 0x49B3
    DW      0x471C, 0x447A, 0x41CD, 0x3F16, 0x3C56, 0x398C, 0x36B9, 0x33DE
    DW      0x30FB, 0x2E10, 0x2B1E, 0x2826, 0x2527, 0x2223, 0x1F19, 0x1C0B
    DW      0x18F8, 0x15E1, 0x12C7, 0x0FAB, 0x0C8B, 0x096A, 0x0647, 0x0324
    DW      0x0000, 0xFCDC, 0xF9B9, 0xF696, 0xF375, 0xF055, 0xED39, 0xEA1F
    DW      0xE708, 0xE3F5, 0xE0E7, 0xDDDD, 0xDAD9, 0xD7DA, 0xD4E2, 0xD1F0
    DW      0xCF05, 0xCC22, 0xC947, 0xC674, 0xC3AA, 0xC0EA, 0xBE33, 0xBB86
```

```
    DW      0xB8E4, 0xB64D, 0xB3C1, 0xB141, 0xAECD, 0xAC66, 0xAA0C, 0xA7BE
    DW      0xA57F, 0xA34D, 0xA12A, 0x9F15, 0x9D0F, 0x9B19, 0x9932, 0x975B
    DW      0x9594, 0x93DD, 0x9237, 0x90A2, 0x8F1F, 0x8DAC, 0x8C4B, 0x8AFD
    DW      0x89C0, 0x8895, 0x877D, 0x8677, 0x8584, 0x84A4, 0x83D7, 0x831E
    DW      0x8277, 0x81E4, 0x8164, 0x80F8, 0x809F, 0x805A, 0x8029, 0x800B
#endif

#if(XX == 9)
    DW      0x7FFF, 0x7FFC, 0x7FF5, 0x7FE8, 0x7FD7, 0x7FC1, 0x7FA6, 0x7F86
    DW      0x7F61, 0x7F37, 0x7F08, 0x7ED4, 0x7E9C, 0x7E5E, 0x7E1C, 0x7DD5
    DW      0x7D89, 0x7D38, 0x7CE2, 0x7C88, 0x7C29, 0x7BC4, 0x7B5C, 0x7AEE
    DW      0x7A7C, 0x7A04, 0x7989, 0x7908, 0x7883, 0x77F9, 0x776B, 0x76D8
    DW      0x7640, 0x75A4, 0x7503, 0x745E, 0x73B5, 0x7306, 0x7254, 0x719D
    DW      0x70E1, 0x7022, 0x6F5E, 0x6E95, 0x6DC9, 0x6CF8, 0x6C23, 0x6B4A
    DW      0x6A6C, 0x698B, 0x68A5, 0x67BC, 0x66CE, 0x65DD, 0x64E7, 0x63EE
    DW      0x62F1, 0x61F0, 0x60EB, 0x5FE2, 0x5ED6, 0x5DC6, 0x5CB3, 0x5B9C
    DW      0x5A81, 0x5963, 0x5842, 0x571D, 0x55F4, 0x54C9, 0x539A, 0x5268
    DW      0x5133, 0x4FFA, 0x4EBF, 0x4D80, 0x4C3F, 0x4AFA, 0x49B3, 0x4869
    DW      0x471C, 0x45CC, 0x447A, 0x4325, 0x41CD, 0x4073, 0x3F16, 0x3DB7
    DW      0x3C56, 0x3AF2, 0x398C, 0x3824, 0x36B9, 0x354D, 0x33DE, 0x326D
    DW      0x30FB, 0x2F86, 0x2E10, 0x2C98, 0x2B1E, 0x29A3, 0x2826, 0x26A7
    DW      0x2527, 0x23A6, 0x2223, 0x209F, 0x1F19, 0x1D93, 0x1C0B, 0x1A82
    DW      0x18F8, 0x176D, 0x15E1, 0x1455, 0x12C7, 0x1139, 0x0FAB, 0x0E1B
    DW      0x0C8B, 0x0AFB, 0x096A, 0x07D9, 0x0647, 0x04B6, 0x0324, 0x0192
    DW      0x0000, 0xFE6E, 0xFCDC, 0xFB4A, 0xF9B9, 0xF827, 0xF696, 0xF505
    DW      0xF375, 0xF1E5, 0xF055, 0xEEC7, 0xED39, 0xEBAB, 0xEA1F, 0xE893
    DW      0xE708, 0xE57E, 0xE3F5, 0xE26D, 0xE0E7, 0xDF61, 0xDDDD, 0xDC5A
    DW      0xDAD9, 0xD959, 0xD7DA, 0xD65D, 0xD4E2, 0xD368, 0xD1F0, 0xD07A
    DW      0xCF05, 0xCD93, 0xCC22, 0xCAB3, 0xC947, 0xC7DC, 0xC674, 0xC50E
    DW      0xC3AA, 0xC249, 0xC0EA, 0xBF8D, 0xBE33, 0xBCDB, 0xBB86, 0xBA34
    DW      0xB8E4, 0xB797, 0xB64D, 0xB506, 0xB3C1, 0xB280, 0xB141, 0xB006
    DW      0xAECD, 0xAD98, 0xAC66, 0xAB37, 0xAA0C, 0xA8E3, 0xA7BE, 0xA69D
    DW      0xA57F, 0xA464, 0xA34D, 0xA23A, 0xA12A, 0xA01E, 0x9F15, 0x9E10
    DW      0x9D0F, 0x9C12, 0x9B19, 0x9A23, 0x9932, 0x9844, 0x975B, 0x9675
    DW      0x9594, 0x94B6, 0x93DD, 0x9308, 0x9237, 0x916B, 0x90A2, 0x8FDE
    DW      0x8F1F, 0x8E63, 0x8DAC, 0x8CFA, 0x8C4B, 0x8BA2, 0x8AFD, 0x8A5C
    DW      0x89C0, 0x8928, 0x8895, 0x8807, 0x877D, 0x86F8, 0x8677, 0x85FC
    DW      0x8584, 0x8512, 0x84A4, 0x843C, 0x83D7, 0x8378, 0x831E, 0x82C8
    DW      0x8277, 0x822B, 0x81E4, 0x81A2, 0x8164, 0x812C, 0x80F8, 0x80C9
    DW      0x809F, 0x807A, 0x805A, 0x803F, 0x8029, 0x8018, 0x800B, 0x8004
#endif
```

```
#if(XX == 10)
   DW     0x7FFF, 0x7FFE, 0x7FFC, 0x7FF9, 0x7FF5, 0x7FEF, 0x7FE8, 0x7FE0
   DW     0x7FD7, 0x7FCD, 0x7FC1, 0x7FB4, 0x7FA6, 0x7F96, 0x7F86, 0x7F74
   DW     0x7F61, 0x7F4C, 0x7F37, 0x7F20, 0x7F08, 0x7EEF, 0x7ED4, 0x7EB9
   DW     0x7E9C, 0x7E7E, 0x7E5E, 0x7E3E, 0x7E1C, 0x7DF9, 0x7DD5, 0x7DB0
   DW     0x7D89, 0x7D61, 0x7D38, 0x7D0E, 0x7CE2, 0x7CB6, 0x7C88, 0x7C59
   DW     0x7C29, 0x7BF7, 0x7BC4, 0x7B91, 0x7B5C, 0x7B25, 0x7AEE, 0x7AB5
   DW     0x7A7C, 0x7A41, 0x7A04, 0x79C7, 0x7989, 0x7949, 0x7908, 0x78C6
   DW     0x7883, 0x783F, 0x77F9, 0x77B3, 0x776B, 0x7722, 0x76D8, 0x768D
   DW     0x7640, 0x75F3, 0x75A4, 0x7554, 0x7503, 0x74B1, 0x745E, 0x740A
   DW     0x73B5, 0x735E, 0x7306, 0x72AE, 0x7254, 0x71F9, 0x719D, 0x7140
   DW     0x70E1, 0x7082, 0x7022, 0x6FC0, 0x6F5E, 0x6EFA, 0x6E95, 0x6E30
   DW     0x6DC9, 0x6D61, 0x6CF8, 0x6C8E, 0x6C23, 0x6BB7, 0x6B4A, 0x6ADB
   DW     0x6A6C, 0x69FC, 0x698B, 0x6919, 0x68A5, 0x6831, 0x67BC, 0x6745
   DW     0x66CE, 0x6656, 0x65DD, 0x6562, 0x64E7, 0x646B, 0x63EE, 0x6370
   DW     0x62F1, 0x6271, 0x61F0, 0x616E, 0x60EB, 0x6067, 0x5FE2, 0x5F5D
   DW     0x5ED6, 0x5E4F, 0x5DC6, 0x5D3D, 0x5CB3, 0x5C28, 0x5B9C, 0x5B0F
   DW     0x5A81, 0x59F3, 0x5963, 0x58D3, 0x5842, 0x57B0, 0x571D, 0x5689
   DW     0x55F4, 0x555F, 0x54C9, 0x5432, 0x539A, 0x5301, 0x5268, 0x51CE
   DW     0x5133, 0x5097, 0x4FFA, 0x4F5D, 0x4EBF, 0x4E20, 0x4D80, 0x4CE0
   DW     0x4C3F, 0x4B9D, 0x4AFA, 0x4A57, 0x49B3, 0x490E, 0x4869, 0x47C3
   DW     0x471C, 0x4674, 0x45CC, 0x4523, 0x447A, 0x43D0, 0x4325, 0x4279
   DW     0x41CD, 0x4120, 0x4073, 0x3FC5, 0x3F16, 0x3E67, 0x3DB7, 0x3D07
   DW     0x3C56, 0x3BA4, 0x3AF2, 0x3A3F, 0x398C, 0x38D8, 0x3824, 0x376F
   DW     0x36B9, 0x3603, 0x354D, 0x3496, 0x33DE, 0x3326, 0x326D, 0x31B4
   DW     0x30FB, 0x3041, 0x2F86, 0x2ECC, 0x2E10, 0x2D54, 0x2C98, 0x2BDB
   DW     0x2B1E, 0x2A61, 0x29A3, 0x28E5, 0x2826, 0x2767, 0x26A7, 0x25E7
   DW     0x2527, 0x2467, 0x23A6, 0x22E4, 0x2223, 0x2161, 0x209F, 0x1FDC
   DW     0x1F19, 0x1E56, 0x1D93, 0x1CCF, 0x1C0B, 0x1B46, 0x1A82, 0x19BD
   DW     0x18F8, 0x1833, 0x176D, 0x16A7, 0x15E1, 0x151B, 0x1455, 0x138E
   DW     0x12C7, 0x1200, 0x1139, 0x1072, 0x0FAB, 0x0EE3, 0x0E1B, 0x0D53
   DW     0x0C8B, 0x0BC3, 0x0AFB, 0x0A32, 0x096A, 0x08A1, 0x07D9, 0x0710
   DW     0x0647, 0x057E, 0x04B6, 0x03ED, 0x0324, 0x025B, 0x0192, 0x00C9
   DW     0x0000, 0xFF37, 0xFE6E, 0xFDA5, 0xFCDC, 0xFC13, 0xFB4A, 0xFA82
   DW     0xF9B9, 0xF8F0, 0xF827, 0xF75F, 0xF696, 0xF5CE, 0xF505, 0xF43D
   DW     0xF375, 0xF2AD, 0xF1E5, 0xF11D, 0xF055, 0xEF8E, 0xEEC7, 0xEE00
   DW     0xED39, 0xEC72, 0xEBAB, 0xEAE5, 0xEA1F, 0xE959, 0xE893, 0xE7CD
   DW     0xE708, 0xE643, 0xE57E, 0xE4BA, 0xE3F5, 0xE331, 0xE26D, 0xE1AA
   DW     0xE0E7, 0xE024, 0xDF61, 0xDE9F, 0xDDDD, 0xDD1C, 0xDC5A, 0xDB99
   DW     0xDAD9, 0xDA19, 0xD959, 0xD899, 0xD7DA, 0xD71B, 0xD65D, 0xD59F
   DW     0xD4E2, 0xD425, 0xD368, 0xD2AC, 0xD1F0, 0xD134, 0xD07A, 0xCFBF
   DW     0xCF05, 0xCE4C, 0xCD93, 0xCCDA, 0xCC22, 0xCB6A, 0xCAB3, 0xC9FD
   DW     0xC947, 0xC891, 0xC7DC, 0xC728, 0xC674, 0xC5C1, 0xC50E, 0xC45C
```

```
    DW      0xC3AA, 0xC2F9, 0xC249, 0xC199, 0xC0EA, 0xC03B, 0xBF8D, 0xBEE0
    DW      0xBE33, 0xBD87, 0xBCDB, 0xBC30, 0xBB86, 0xBADD, 0xBA34, 0xB98C
    DW      0xB8E4, 0xB83D, 0xB797, 0xB6F2, 0xB64D, 0xB5A9, 0xB506, 0xB463
    DW      0xB3C1, 0xB320, 0xB280, 0xB1E0, 0xB141, 0xB0A3, 0xB006, 0xAF69
    DW      0xAECD, 0xAE32, 0xAD98, 0xACFF, 0xAC66, 0xABCE, 0xAB37, 0xAAA1
    DW      0xAA0C, 0xA977, 0xA8E3, 0xA850, 0xA7BE, 0xA72D, 0xA69D, 0xA60D
    DW      0xA57F, 0xA4F1, 0xA464, 0xA3D8, 0xA34D, 0xA2C3, 0xA23A, 0xA1B1
    DW      0xA12A, 0xA0A3, 0xA01E, 0x9F99, 0x9F15, 0x9E92, 0x9E10, 0x9D8F
    DW      0x9D0F, 0x9C90, 0x9C12, 0x9B95, 0x9B19, 0x9A9E, 0x9A23, 0x99AA
    DW      0x9932, 0x98BB, 0x9844, 0x97CF, 0x975B, 0x96E7, 0x9675, 0x9604
    DW      0x9594, 0x9525, 0x94B6, 0x9449, 0x93DD, 0x9372, 0x9308, 0x929F
    DW      0x9237, 0x91D0, 0x916B, 0x9106, 0x90A2, 0x9040, 0x8FDE, 0x8F7E
    DW      0x8F1F, 0x8EC0, 0x8E63, 0x8E07, 0x8DAC, 0x8D52, 0x8CFA, 0x8CA2
    DW      0x8C4B, 0x8BF6, 0x8BA2, 0x8B4F, 0x8AFD, 0x8AAC, 0x8A5C, 0x8A0D
    DW      0x89C0, 0x8973, 0x8928, 0x88DE, 0x8895, 0x884D, 0x8807, 0x87C1
    DW      0x877D, 0x873A, 0x86F8, 0x86B7, 0x8677, 0x8639, 0x85FC, 0x85BF
    DW      0x8584, 0x854B, 0x8512, 0x84DB, 0x84A4, 0x846F, 0x843C, 0x8409
    DW      0x83D7, 0x83A7, 0x8378, 0x834A, 0x831E, 0x82F2, 0x82C8, 0x829F
    DW      0x8277, 0x8250, 0x822B, 0x8207, 0x81E4, 0x81C2, 0x81A2, 0x8182
    DW      0x8164, 0x8147, 0x812C, 0x8111, 0x80F8, 0x80E0, 0x80C9, 0x80B4
    DW      0x809F, 0x808C, 0x807A, 0x806A, 0x805A, 0x804C, 0x803F, 0x8033
    DW      0x8029, 0x8020, 0x8018, 0x8011, 0x800B, 0x8007, 0x8004, 0x8002
#endif
END
```

［メ　　モ］

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044−435−9494
FAX　：044−435−9608
E-mail　：info@lsi.nec.co.jp

【営業関係お問い合わせ先】

システムLSI第一営業事業部
東　京　(03)3798−6106, 6107, 6108, 6155
大　阪　(06)6945−3178, 3200, 3208
名古屋　(052)222−2375
仙　台　(022)267−8740
水　戸　(029)226−1702
広　島　(082)242−5504
鳥　取　(0857)27−5313
松　山　(089)945−4149

システムLSI第二営業事業部
東　京　(03)3798−6110, 6111, 6112, 6151, 6156
名古屋　(052)222−2170, 2190
松　本　(0263)35−1662
前　橋　(027)243−6060
立　川　(042)526−5981
静　岡　(054)254−4794
金　沢　(076)232−7303
福　岡　(092)261−2806

【資料の請求先】

上記営業関係お問い合わせ先またはNEC特約店へお申しつけください。

【NECエレクトロンデバイス　ホームページ】

NECエレクトロンデバイスの情報がインターネットでご覧になれます。　URL（アドレス）　http://www.ic.nec.co.jp/

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］ μPD77016 ファミリ　アプリケーション・ノート　基本ソフトウエア編
（U11958JJ2V0AN00（第 2 版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　）
ご住所　（　　）
お電話番号　（　　）
お仕事の内容　（　　）
お名前　（　　）

1．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは
NEC 販売員，特約店販売員， その他（　　）

ご協力ありがとうございました。
下記あてに FAX で送信いただくか，最寄りの販売員にコピーをお渡しください。

日本電気(株) NEC エレクトロンデバイス
半導体テクニカルホットライン
FAX：（044）435-9608

2000.6

キリトリ