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

78K/IVシリーズ

16ビット・シングルチップ・マイクロコンピュータ

ソフトウエア基礎編

78K/IVシリーズ共通

資料番号　U10095JJ2V1AN00（第２版）
発行年月　October 2000 N　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
はじめに	78K/IVシリーズ共通とするため，各サブシリーズ名および各製品名を追加
p.125	**付録A　改版履歴**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。

〔メ　モ〕

はじめに

対象者　このアプリケーション・ノートは，78K/IVシリーズ製品の機能を理解し，78K/IVシリーズ製品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のエンジニアを対象とします。

★ **78K/IVシリーズ**

- μPD784026サブシリーズ　：μPD784020, 784021, 784025, 784026, 78P4026
- μPD784038サブシリーズ　：μPD784031, 784035, 784036, 784037, 784038, 78P4038
- μPD784038Yサブシリーズ　：μPD784031Y, 784035Y, 784036Y, 784037Y, 784038Y, 78P4038Y
- μPD784046サブシリーズ　：μPD784044, 784046, 784054, 78F4046
- μPD784216Aサブシリーズ　：μPD784214A, 784215A, 784216A, 78F4216A
- μPD784216AYサブシリーズ：μPD784214AY, 784215AY, 784216AY, 78F4216AY
- μPD784218Aサブシリーズ　：μPD784217A, 784218A, 78F4218A
- μPD784218AYサブシリーズ：μPD784217AY, 784218AY, 78F4218AY
- μPD784225サブシリーズ　：μPD784224, 784225, 78F4225
- μPD784225Yサブシリーズ　：μPD784224Y, 784225Y, 78F4225Y
- μPD784908サブシリーズ　：μPD784907, 784908, 78P4908
- μPD784915サブシリーズ　：μPD784915B, 784916B, 784P4916
- μPD784928サブシリーズ　：μPD784927, 784928, 78F4928[注]
- μPD784928Yサブシリーズ　：μPD784927Y, 784928Y, 78F4928Y[注]
- μPD784938Aサブシリーズ　：μPD784935A, 784936A, 784937A, 784938A, 78F4938A[注]
- μPD784956Aサブシリーズ　：μPD784953A, 784956A, 78F4956A[注]
- μPD784976Aサブシリーズ　：μPD784975A[注], 78F4976A[注]

注 開発中

目　的　このアプリケーション・ノートは，78K/IVシリーズ製品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ます。
なお，掲載のプログラムおよびハードウエア構成は例示的に示したものであり，量産設計を対象とするものではありません。

構　成　このアプリケーション・ノートでは，基本的な数値演算プログラムなどについて説明しています。

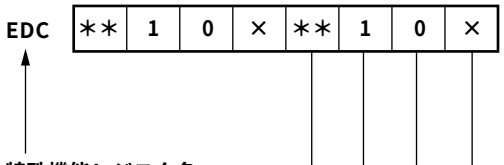
応用分野　78K/IVシリーズは１Mバイトのプログラム・メモリ空間や高速命令実行，低電圧動作などが可能なため，次に示すような広い範囲の応用分野に適応できます。

- 携帯電話
- VTR
- CD-ROM
- プリンタ
- HDD
- オーディオ　など
- カメラ

凡　　例　本アプリケーションの本文中に使用している記号，表記は次の内容を示します。

データ表記の重み	：左側が上位桁，右側が下位桁
アクティブ・ロウの表記	：$\overline{\times\times\times}$（端子，信号名称に上線）
注	：本文に付けた注の説明
注意	：特に気をつけていただきたい内容
備考	：本文の補足説明
数の表記	：２進数・・・××××××××B
	：10進数・・・×××××
	：16進数・・・××××H
まぎらわしい文字	：０（ゼロ），O（オー）
	：１（イチ），l（エル），I（アイ）

特殊機能レジスタ（SFR）表記



ライト動作時	リード動作時
0または1を書き込みます。 いずれの値でも動作には影響を与えません。	0または1を読み出します。
0を書く必要があります。	
1を書く必要があります。	
使用したい機能に応じた値を書き込みます。 表記＊＊の設定値は0または1となります。	動作状態に従った値を読み出します。

本文中のレジスタ表記に『設定禁止』と書いてあるコードの組み合わせは，絶対に書き込まないでください。

特殊機能レジスタ（SFR）表記例

	7	6	5	4	3	2	1	0
STBC	0	0	CK1	CK0	×	0	STP	HLT
設定例	0	0	0	1	0	0	0	0

STP	HLT	動作モード
0	0	通常動作モード
0	1	HALTモード
1	0	STOPモード
1	1	IDLEモード

CK1	CK0	内部システム・クロックの選択
0	0	f_{XX}/2 （12.5 MHz）
0	1	f_{XX}/4 （6.25 MHz）
1	0	f_{XX}/8 （3.125 MHz）
1	1	f_{XX}/16（1.5625 MHz）

（f_{XX} = 25 MHz）

備考 このアプリケーション・ノートでは，設定する特殊機能レジスタのビットに網掛け をしています。利用されるときは，設定例を参考にしてください。

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。

★ ○78K/IVシリーズ共通資料

資　料　名	資料番号	
	和　文	英　文
ユーザーズ・マニュアル　命令編	U10905J	U10905E
アプリケーション・ノート　ソフトウエア基礎編	このマニュアル	−
インストラクション活用表	U10594J	−
インストラクション・セット	U10595J	−
開発ツール　セレクション・ガイド	U11069J	U11069E

★ ○個別資料

***μ*PD784026サブシリーズ**

資　料　名	資料番号	
	和　文	英　文
μPD784020, 784021　データ・シート	U11514J	U11514E
μPD784025, 784026　データ・シート	U11605J	IP-3230
μPD78P4026　データ・シート	U11609J	IP-3231
μPD784026サブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U10898J	U10898E
μPD784026サブシリーズ　特殊機能レジスタ活用表	U10593J	−
μPD784026サブシリーズ　アプリケーション・ノート　ハードウエア基礎編	U10573J	−

***μ*PD784038, 784038Yサブシリーズ**

資　料　名	資料番号	
	和　文	英　文
μPD784031　データ・シート	U11507J	U11507E
μPD784035, 784036, 784037, 784038　データ・シート	U10847J	U10847E
μPD78P4038　データ・シート	U10848J	U10848E
μPD784038サブシリーズ　特殊機能レジスタ活用表	U11090J	−
μPD784031Y　データ・シート	U11504J	U11504E
μPD784035Y, 784036Y, 784037Y, 784038Y　データ・シート	U10741J	U10741E
μPD78P4038Y　データ・シート	U10742J	U10742E
μPD784038Yサブシリーズ　特殊機能レジスタ活用表	U11091J	−
μPD784038, 784038Yサブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U11316J	U11316E

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。

μPD784046サブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784044, 784046　データ・シート	U10951J	U10951E
μPD784044(A), 784046(A)　データ・シート	U13121J	U13121E
μPD784054　データ・シート	U11154J	U11154E
μPD784054(A)　データ・シート	U13122J	U13122E
μPD78F4046　データ・シート	U11447J	U11447E
μPD784046サブシリーズ　特殊機能レジスタ活用表	U10986J	－
μPD784054　特殊機能レジスタ活用表	U11113J	－
μPD784046サブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U11515J	U11515E
μPD784054　ユーザーズ・マニュアル　ハードウエア編	U11719J	U11719E

μPD784216A, 784216AY, 784218A, 784218AYサブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784214A, 784215A, 784216A, 784217A, 784218A, 784214AY, 784215AY, 784216AY, 784217AY, 784218AY　データ・シート	U14121J	U14121E
μPD78F4216A, 78F4218A, 78F4216AY, 78F4218AY　データ・シート	U14125J	作成中
μPD784216A, 784216AYサブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U13570J	U13570E

μPD784225, 784225Yサブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784224, 784225, 784224Y, 784225Y　データ・シート	U12376J	U12376E
μPD78F4225　ペーパ・マシン	U12499J	U12499E
μPD784225サブシリーズ　特殊機能レジスタ表	U12698J	－
μPD784225, 784225Yサブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U12679J	U12679E

μPD784908サブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784907, 784908　データ・シート	U11680J	U11680E
μPD78P4908　データ・シート	U11681J	U11681E
μPD784908サブシリーズ　特殊機能レジスタ活用表	U11589J	－
μPD784908サブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U11787J	U11787E

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。

μPD784915サブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784915B, 784916B　データ・シート	U13118J	U13118E
μPD78P4916　データ・シート	U11045J	U11045E
μPD784915サブシリーズ　特殊機能レジスタ活用表	U10976J	−
μPD784915サブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U10444J	U10444E
μPD784915, 784928, 784928Yサブシリーズ　アプリケーション・ノート　VTRサーボ基礎編	U11361J	U11361E

μPD784928, 784928Yサブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784927, 784928, 784927Y, 784928Y　データ・シート	U12255J	U12255E
μPD784928, 784928Yサブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U12648J	U12648E
μPD784915, 784928, 784928Yサブシリーズ　アプリケーション・ノート　VTRサーボ基礎編	U11361J	U11361E

μPD784938Aサブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784935A, 784936A, 784937A, 784938A　データ・シート	U13572J	作成中
μPD78F4938A　データ・シート	作成予定	作成予定
μPD784938Aサブシリーズ　ユーザーズ・マニュアル　ハードウエア編	作成中	作成予定

μPD784956Aサブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784953A, 784956A　データ・シート	作成予定	作成予定
μPD78F4956A　データ・シート	作成予定	作成予定
μPD784956A　サブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U14395J	作成中

μPD784976Aサブシリーズ

資　料　名	資料番号	
	和　文	英　文
μPD784975A　データ・シート	作成中	作成予定
μPD78F4976A　データ・シート	作成予定	作成予定
μPD784976A　サブシリーズ　ユーザーズ・マニュアル　ハードウエア編	U15017J	作成中

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

第１章　概　　説

１.１　本書の見方

本書では，２進，10進の基本的な四則計算，データ変換，データ転送などのプログラム例をサブルーチン・プログラムとして示し，ユーザ・プログラムとして参考活用していただけるようにアルゴリズムの解説をしています。

項目は次のとおりです。

（１）処理概要

プログラムの処理の概要を説明します。

（２）使用RAM領域

プログラム中で使用しているRAM領域を説明します。

ただし，使用RAM領域がワーク・エリアの場合は，プログラム実行後の内容は不定となります。

（３）使用レジスタ

プログラム中で使用しているレジスタを示します。

すでに別のプログラムで使用しているレジスタの内容を破壊したくない場合は，このプログラムを実行する前に，レジスタ・バンク切り替えなどであらかじめレジスタの内容を退避する必要があります。

（４）入力方法

プログラムを実行する際に必要となる入力の引数を説明します。

（５）出力方法

プログラムを実行したあとの出力の引数を説明します。

（６）プログラム説明

プログラムのアルゴリズムを説明します。

フロー・チャートおよびプログラム・リストをあわせて参照してください。

（７）フロー・チャート

プログラムのアルゴリズムをフロー・チャートで示します。

（8）プログラム・リスト

プログラム・リストを示します。

プログラム・リストはすべてソース・プログラムで記載してあります。実際に配置されるアドレスは，リンク条件によって異なります。

このアプリケーション・ノートのプログラムは例題であり，そのプログラムの動作を保障するものではありません。

1.2　アプリケーション・プログラムの利用方法

基本的な利用方法は，各アプリケーション・プログラムの解説に従ってください。アプリケーション・プログラムによっては，他のアプリケーション・プログラムを呼び出す構成になっています。その場合は，解説に従いリンカ（LK78K/IV）によりリンクしてください。また，ワーク領域を必要とする場合がありますので，解説に従ってRAM領域の確保を 行い，パブリック宣言を行ってください。

1.3　78K/IVシリーズ製品の特徴

近年，マイクロコンピュータ応用機器の高機能化と低価格化に伴い，マイクロコンピュータにも急激に高機能化への対応と，コスト・パフォーマンスの追求という相反する市場要求が強くなっています。また，携帯型機器の発展に伴い，低電圧で低消費電力のマイクロコンピュータに対する要求も強まってきています。

78K/IVシリーズは，このような市場動向に対応して，次のようなコンセプトのもとで開発した16ビットのシングルチップ・マイクロコンピュータです。

78K/IVシリーズの特徴を次に示します。

（1）従来製品との互換性

従来から78Kシリーズに属する８ビット・マイクロコンピュータ（78K/0, 78K/I, 78K/ II）と16ビット・マイクロコンピュータ（78K/III）とソース・レベルでの上位互換性を有していますので，ソフトウエア資産の有効活用が可能となります。

（2）広いリニアなメモリ空間

プログラム・メモリとして最大１Mバイト，データ・メモリとして最大16 Mバイトをサポートしています。

（3）低電圧，低消費電流対応

動作電圧範囲としては，2.7～5.5 Vに対応しており，低電圧でも動作が可能になっています。さらにSTOP/IDLE/HALTの３種類のスタンバイ・モードと，CPUへ供給するクロックを分周する機能を有しており，動作状態に応じたパワー・マネージメントが可能です。

（４）高速な乗除算命令

制御対象の複雑化や，制御アルゴリズムの高精度化に対応して乗除算の高速化を図っています。

８ビット×８ビット = 16ビット（符号なし）	→0.69 μs
16ビット×16ビット = 32ビット（符号なし）	→0.94 μs
16ビット×16ビット = 32ビット（符号付き）	→0.88 μs
16ビット÷８ビット = 商16ビット余り８ビット（符号なし）	→1.06 μs
32ビット÷16ビット = 商32ビット余り16ビット（符号なし）	→1.94 μs

（内部16 MHz動作時）

（５）強力な割り込み応答方法

制御用途のマイクロコンピュータとして，重要なポイントである割り込みの応答方法として次に示す３種類の割り込み機能を内蔵しています。

- ベクタ割り込み
- コンテキスト・スイッチング
- マクロ・サービス

（６）高級言語（Ｃ言語）への対応強化

Ｃコンパイラ向けの命令セット強化と，あわせて効率の良いＣコンパイラと使いやすいソース・ディバグ環境を提供しています。

（７）使いやすいツール

使いやすいツールの提供として，アセンブラ・パッケージ，Ｃコンパイラ，インサーキット・エミュレータ，統合ディバッガを用意しています。

1.4　プログラムについて

このアプリケーション・ノートでは，プログラムによってRAM領域上に，演算結果や数値データを配置します。したがって，このアプリケーション・ノートのプログラムを参考にされる場合は，次のプログラムを参考に，RAM領域の確保を行ってください。

また，２進除算，10進加算，10進減算および10進除算の各プログラムを利用される場合は，必要に応じて演算エラーが発生した場合の処理を追加してください。

```
        PUBLIC  BMLCND,BMLIER,BUFRAM,BRSLT      ; binary multiply
        PUBLIC  DEND,DVISOR,DRMND               ; binany division
        PUBLIC  DMLCND,DMLIER,DRSLT,CARRY       ; decimal multiply
        PUBLIC  DIVSOR,DIVIND,RMIND             ; decimal division
        PUBLIC  SFLAG
        PUBLIC  ERRFLAG                         ; error flag

        BSEG
SFLAG   DBIT                                    ; sin flag
ERRFLAG DBIT                                    ; error flag

WORKSEG DSEG saddr

;-----------------------------------------------
;               binary multiply area
;-----------------------------------------------
BMLCND: DS       4
BMLIER: DS       4
BUFRAM: DS      15
BRSLT:  DS       8

;-----------------------------------------------
;               binary division area
;-----------------------------------------------
DEND:   DS       4
DVISOR: DS       4
DRMND:  DS       4

;-----------------------------------------------
;               decimal multiply area
;-----------------------------------------------
DMLCND: DS       4
DMLIER: DS       4
DRSLT:  DS       8
CARRY:  DS       1

;-----------------------------------------------
;               decimal division
;-----------------------------------------------
DIVSOR: DS       4
DIVIND: DS       4
RMIND:  DS       4

        CSEG
```

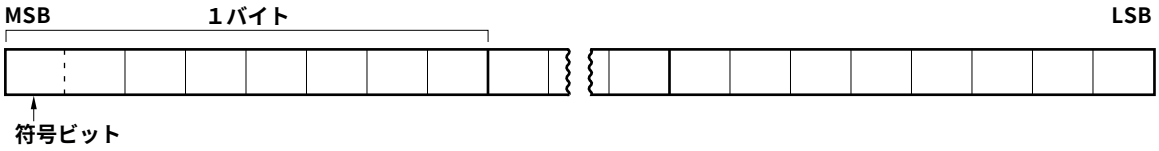
```
          .
          .
        BT  ERRFLAG,$ERROR                ; if(error flag=1) then
          .                               ;           go to ERROR
          .
ERROR:                                    ; error routine
          .
          .                               ; エラー処理，必要に応じて作成
                                          ; してください。
```

〔メ　モ〕

第２章　２進演算

最上位ビットを符号ビットとし，残りのビットで数値を表現します。負の数の表現は，２の補数表現を用います。

図２－１　２進数の表現

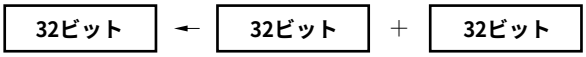


２進演算では，演算に使用するデータ格納領域も，演算後の結果領域も，RAM領域上に配置します。

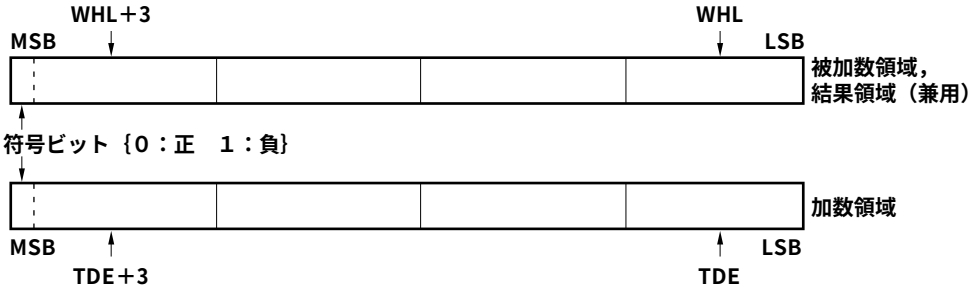
２.１　符号付き32ビット＋32ビットの２進加算

（１）処理概要

符号付き32ビットの被加数と32ビットの加数を加算し，演算結果を32ビットの結果領域（被加数領域と兼用）に格納するプログラム例を紹介します。



（２）使用RAM領域



符号ビット（０）のとき：正（00000000H～7FFFFFFFH）

符号ビット（１）のとき：負（FFFFFFFFH～80000000H）

（３）使用レジスタ

AX，C，VP, TDE, WHLレジスタ

（４）入力方法

WHL, TDEレジスタを次のように設定します。

WHL　：被加数32ビットを格納してあるRAM領域の最下位アドレスを設定します。
TDE　：加数32ビットを格納してあるRAM領域の最下位アドレスを設定します。

（５）出力方法

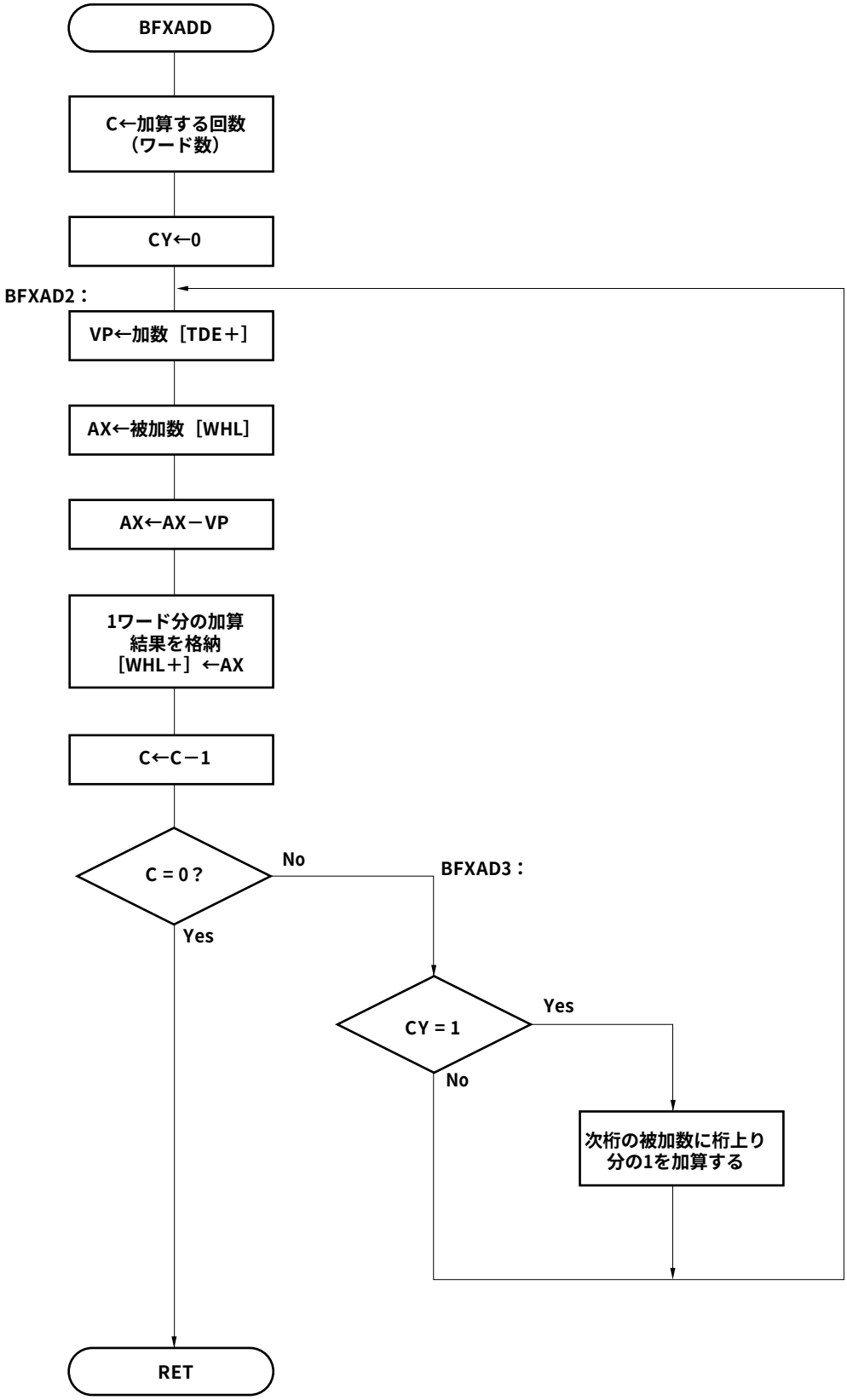
WHLレジスタで示す４バイトのRAM領域に，次の内容が格納されます。

WHL～WHL＋３：加算した演算結果が格納されます。

（６）プログラム説明

（ａ）カウンタ（Ｃレジスタ）に演算するワード数を設定する。
（ｂ）キャリー・フラグ（CY）をあらかじめクリア（０）する。
（ｃ）加数アドレス（TDEレジスタ）で示される加数領域の２バイトをAXレジスタに読み込み，加数アドレス（TDEレジスタ）をインクリメントする。
（ｄ）被加数アドレス（WHLレジスタ）で示される被加数領域の２バイトをAXレジスタに加える。
（ｅ）AXレジスタの値を被加数アドレス（WHLレジスタ）で示されるRAM領域に格納し，被加数アドレス（WHLレジスタ）をインクリメントする。
（ｆ）カウンタ（Ｃレジスタ）をデクリメントし，カウンタ（Ｃレジスタ）が０になれば加算処理を終了する。
（ｇ）加算後，桁上がりが生じていれば，AXレジスタに次の被加数アドレス（WHLレジス タ）で示される結果領域の上位２バイトのデータを読み込み，桁上がり値の１を加算する。
（ｈ）AXレジスタの値を被加数アドレス（WHLレジスタ）で示される結果領域に格納し，（ｃ）の処理に戻る。

（7）フロー・チャート


BFXADD
C←加算する回数
（ワード数）
CY←0
BFXAD2：
VP←加数［TDE＋］
AX←被加数［WHL］
AX←AX－VP
1ワード分の加算
結果を格納
［WHL＋］←AX
C←C－1
C = 0？
No
BFXAD3：
Yes
CY = 1
Yes
No
次桁の被加数に桁上り
分の1を加算する
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

AUGNE：被加数32ビットを格納してあり，結果32ビットを格納するRAM領域の最下位アドレス
（兼用）

ADDEN：加数32ビットを格納してあるRAM領域の最下位アドレス

●メイン・ルーチンのプログラム・リスト記述例

```
     •
     •
MOVG    WHL,#AUGNE  ;
MOVG    TDE,#ADDEN  ;

CALL    !BFXADD     ;data "A+B" subrutin
     •
     •
```

備考　上記記述例のようにWHL, TDEレジスタを設定し，サブルーチンを呼んでください。

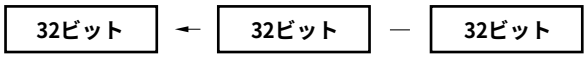
●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BFXADR
;*************************************************
;*      binary addition                          *
;*           32 bit <- 32 bit + 32 bit           *
;*       input condition                          *
;*          WHL-register <- augnend bottom.address *
;*          TDE-register <- addend  bottom.address *
;*       output condition                         *
;*             result <- (HL+3,HL+2,HL+1,HL)      *
;*************************************************
;
        PUBLIC  BFXADD
        CSEG
BFXADD:
        MOV     C,#2        ;
BFXAD1:
        CLR1    CY          ;
BFXAD2:
        MOVW    AX,[TDE+]   ;TDE-register <- addend  address
        MOVW    VP,AX       ;
        MOVW    AX,[WHL]    ;WHL-register <- augnend address
        ADDW    AX,VP       ;AX <- augnend + addend
        MOVW    [WHL+],AX   ;

        DBNZ    C,$BFXAD3   ;add end ?
        BR      ADDEND      ; [yes]
BFXAD3:
        BNC     $BFXAD2     ; [no] CY = 1 ?
        MOVW    AX,[WHL]    ;    [yes]
        ADDW    AX,#1       ;          next data add 1
        MOVW    [WHL],AX    ;
        BR      BFXAD2      ;
ADDEND:
        RET                 ;
```

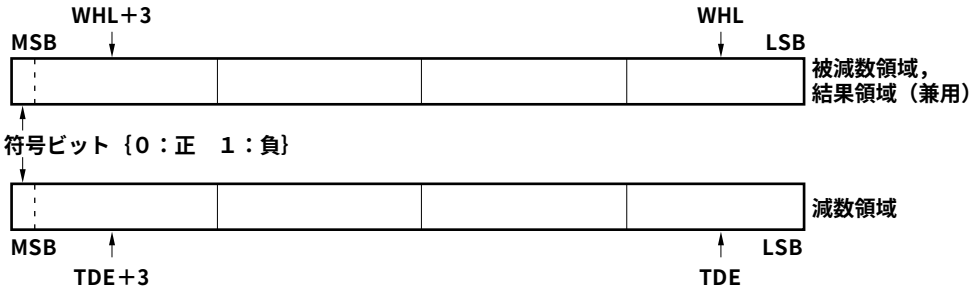
２.２　符号付き32ビット－32ビットの２進減算

（１）処理概要

符号付き32ビットの被減数から32ビットの減数を減算し，演算結果を32ビットの結果領域（被減数領域と兼用）に格納するプログラム例を紹介します。



（２）使用RAM領域



符号ビット（０）のとき：正（00000000H～7FFFFFFFH）

符号ビット（１）のとき：負（FFFFFFFFH～80000000H）

（３）使用レジスタ

AX，C，VP, TDE, WHLレジスタ

（４）入力方法

WHL, TDEレジスタを次のように設定します。

WHL　：被減数32ビットを格納してあるRAM領域の最下位アドレスを設定します。

TDE　：減数32ビットを格納してあるRAM領域の最下位アドレスを設定します。

（５）出力方法

WHLレジスタで示す４バイトのRAM領域に，次の内容が格納されます。

WHL～WHL＋３：減算した演算結果が格納されます。

（６）プログラム説明

（ａ）カウンタ（Ｃレジスタ）に演算するワード数を設定する。

（ｂ）減数領域，被減数領域に，減算するデータを設定する。

（ｃ）キャリー・フラグ（CY）をあらかじめクリア（０）する。

（ｄ）減数アドレス（TDEレジスタ）で示される減数領域の２バイトをAXレジスタに読み込み，減数アドレス（TDEレジスタ）をインクリメントする。

（ｅ）被減数アドレス（WHLレジスタ）で示される被減数領域の２バイトをAXレジスタから引く。

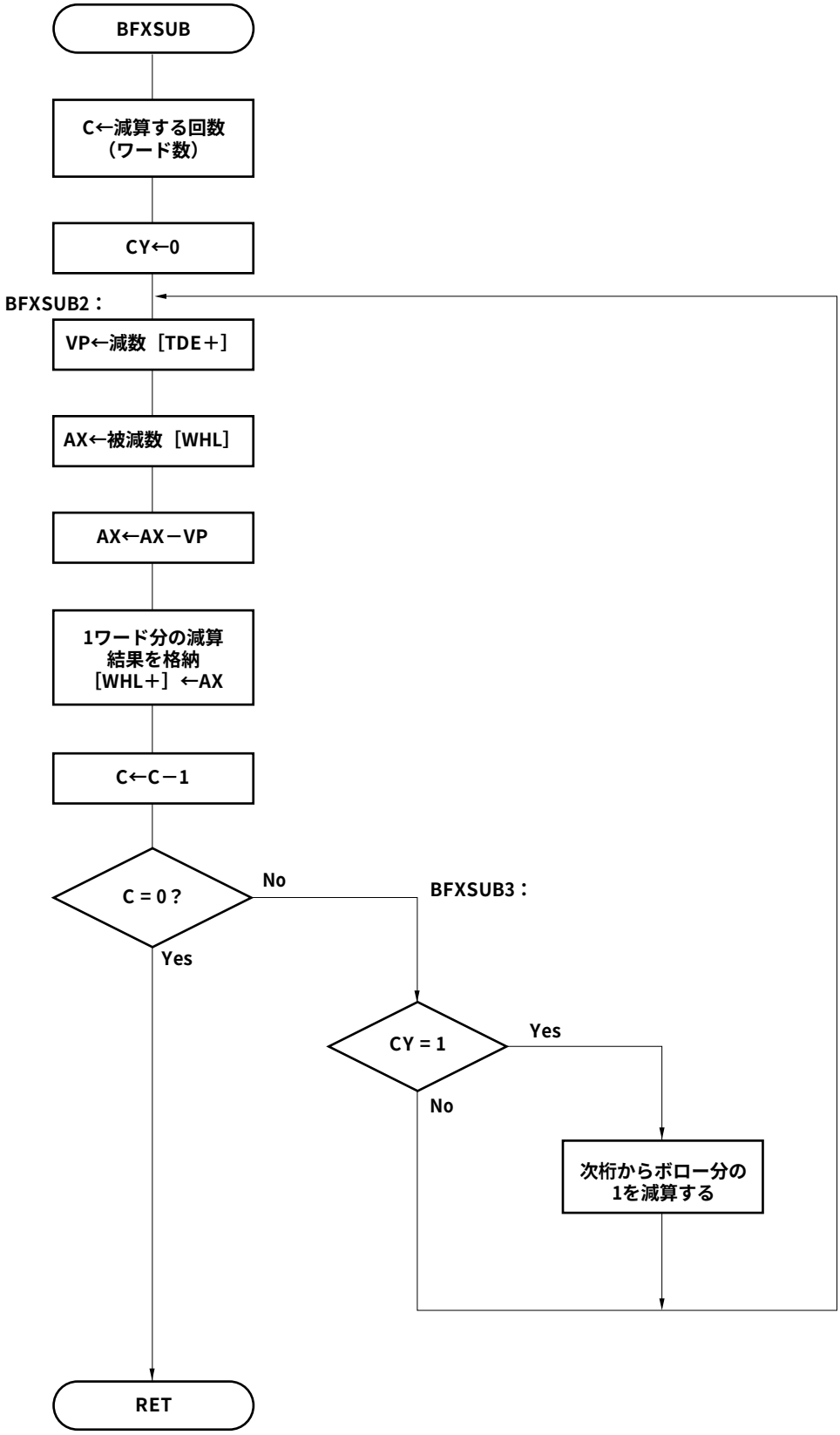
（ｆ）AXレジスタの値を被減数アドレス（WHLレジスタ）で示される結果領域に格納し，被減数アドレス（WHLレジスタ）をインクリメントする。

（ｇ）カウンタ（Ｃレジスタ）をデクリメントし，カウンタ（Ｃレジスタ）が０になれば減算処理を終了する。

（ｈ）減算後，桁借り（ボロー）が発生していれば，AXレジスタに次の被減数アドレス（WHLレジスタ）で示される上位２バイトのデータを読み込み，桁借り分の１を減算する。

（ｉ）AXレジスタの値を被減数アドレス（WHLレジスタ）で示される結果領域に格納し，（ｄ）の処理に戻る。

（7）フロー・チャート


BFXSUB
C←減算する回数
（ワード数）
CY←0
BFXSUB2：
VP←減数［TDE＋］
AX←被減数［WHL］
AX←AX－VP
1ワード分の減算
結果を格納
［WHL＋］←AX
C←C－1
C = 0 ?
No
Yes
BFXSUB3：
CY = 1
Yes
No
次桁からボロー分の
1を減算する
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

MINU ：被減数32ビットを格納してあり，結果32ビットを格納するRAM領域の最下位アドレ ス（兼用）

SUBT ：減数32ビットを格納してあるRAM領域の最下位アドレス

●メイン・ルーチンのプログラム・リスト記述例

```
      •
      •
MOVG    WHL,#MINU   ;
MOVG    TDE,#SUBT   ;

CALL    !BFXSUB     ;data "A-B" subrutin
      •
      •
```

備考　上記記述例のようにWHL, TDEレジスタを設定し，サブルーチンを呼んでください。

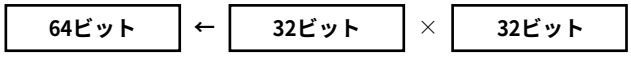
●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BFXSBR

;*******************************************************
;*      binary subtraction                             *
;*          32 bit <- 32 bit - 32 bit                  *
;*       input condition                               *
;*          WHL-register <- minus value bottom.address *
;*          TDE-register <- subtrahend bottom.address  *
;*       output condition                              *
;*              result <- (WHL+3,WHL+2,WHL+1,WHL)      *
;*******************************************************

        PUBLIC  BFXSUB
;
BFXSUB:
        MOV     C,#2        ;
BFXSUB1:
        CLR1    CY          ;
BFXSUB2:
        MOVW    AX,[TDE+]   ;TDE-register <- minus value address
        MOVW    VP,AX       ;
        MOVW    AX,[WHL]    ;WHL-register <- subtrahend  address
        SUBW    AX,VP       ;AX <- minus valse addend - subtrahend
        MOVW    [WHL+],AX   ;

        DBNZ    C,$BFXSUB3  ;sub end?
        BR      SUBEND      ; [yes]
BFXSUB3:
        BNC     $BFXSUB2    ; [no] CY = 1 ?
        MOVW    AX,[WHL]    ;   [yes]
        SUBW    AX,#1       ;       next data sub 1
        MOVW    [WHL],AX    ;
        BR      BFXSUB2     ;
SUBEND:
        RET                 ;
```

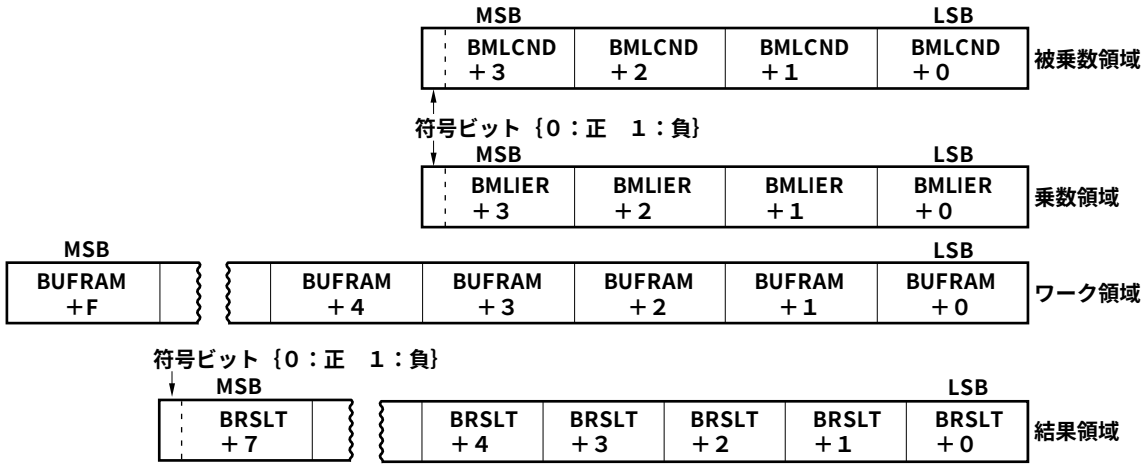
２.３　符号付き32ビット×32ビットの２進乗算

（１）処理概要

符号付き32ビットの被乗数と32ビットの乗数を乗算し，演算結果を64ビットの結果領域に格納するプログラム例を紹介します。



（２）使用RAM領域



符号ビット（０）のとき：正（00000000H～7FFFFFFFH）

符号ビット（１）のとき：負（FFFFFFFFH～80000000H）

（３）使用レジスタ

Ａ，Ｘ，Ｂ，Ｃ，VP, TDE, WHL, RP2, R4レジスタ

（４）入力方法

次に示す各４バイトのRAM領域に，演算に必要なデータを設定します。

BMLCND～BMLCND＋３　　：32ビットの被乗数データを設定します。

BMLIER～BMLIER＋３　　：32ビットの乗数データを設定します。

（５）出力方法

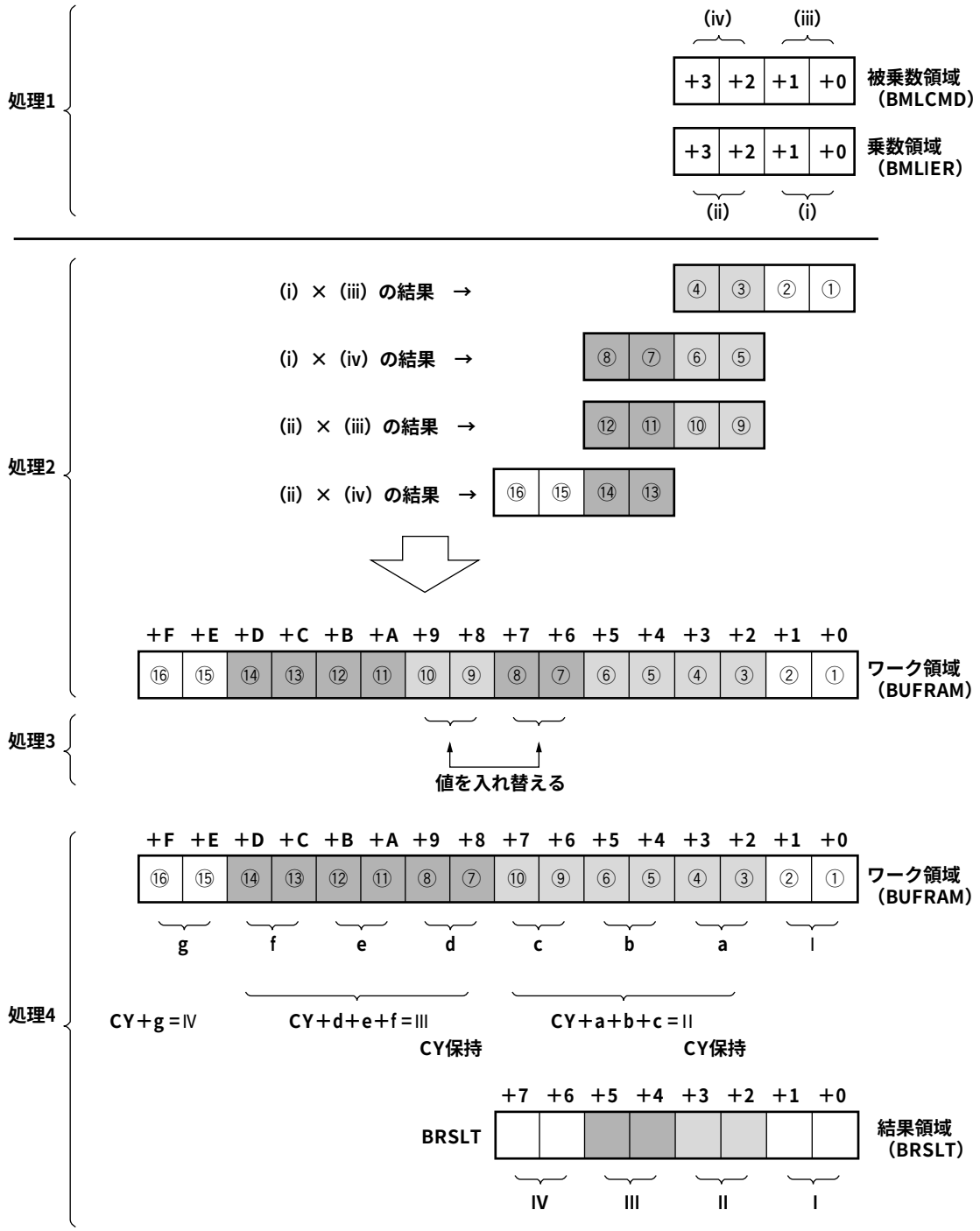
次に示す８バイトのRAM領域に，次の内容が格納されます。

BRSLT～BRSLT＋７：乗算した演算結果が格納されます。

（６）プログラム説明

16ビット乗算命令を用いた２進乗算のアルゴリズムを図２－２に示します。

図２－２　２進乗算のアルゴリズム



備考　CY：キャリー・フラグです。

処理４において，下位桁の演算で発生したキャリー・フラグ（CY）を加算します。

次に本プログラムの処理手順を示します。

（ａ）乗数，被乗数の絶対値をとる。乗数，被乗数が異符号なら符号判別フラグ（SFLAG）をセット（１）し，同符号ならクリア（０）する。

（ｂ）乗算終了判別フラグ（LFLAG）をセット（１）する。

（LFLAG＝１：乗算未終了，LFLAG＝０：乗算終了）

（ｃ）被乗数領域のアドレス・ポインタとしてＢレジスタを，使用し，最下位アドレスを示す０を設定する。

処理１

（ｄ）TDEレジスタにワーク領域の最下位アドレスをWHLレジスタに乗数領域の最下位アドレスを設定する。

（ｅ）VPレジスタにWHLレジスタで示された乗数２バイトを読み込み，その内容を次桁演算のために退避しておく。

処理２

（ｆ）AXレジスタに被乗数領域のアドレス・ポインタ（Ｂレジスタ）の指す被乗数２バイトを読み込み，乗数であるVPレジスタと乗算を行う。

その演算結果を，TDEレジスタが指し示すワーク領域に，４バイトずつ順次乗算した演算結果を格納する。

（ｇ）Ｂレジスタに２を加算して，被乗算領域のアドレス・ポインタを２バイト分，進める。

（ｈ）被乗数領域のアドレス・ポインタ（Ｂレジスタ）の値が被乗数領域の桁数（４）と同じかどうか比較する。

比較結果が同値でなければ，次桁の被乗数との乗算を行うためにVPレジスタに（ｅ）で退避した乗数の値を復帰させて（ｇ）の処理へ戻る。比較結果が同値になるまで，（ｇ）～（ｉ）の処理を繰り返す。

（ｉ）被乗数領域のアドレス・ポインタ（Ｂレジスタ）に最下位アドレスを示す０を設定する。

（ｊ）すべての乗算が終了したか，乗算終了判別フラグ（LFLAG）により判断し，終了していなければ，乗算終了判別フラグ（LFLAG）をクリア（０）して（ｅ）の処理へ戻り，（ｊ）までの処理を繰り返す。

処理３

（ｋ）乗算結果を格納しているワーク領域のBUFRAM＋６，BUFRAM＋７とBUFRAM＋８，BUFRAM＋９の内容を入れ替える。

処理４

（ｌ）BUFRAM＋０，BUFRAM＋１の値を結果領域（BRSLT）の１，２バイト目に格納する。

（ｍ）BUFRAM＋２～BUFRAM＋７の値を２バイトごと，キャリー・フラグ（CY）と一緒に順次加算し，演算結果を結果領域（BRSLT）の３，４バイト目に格納する。

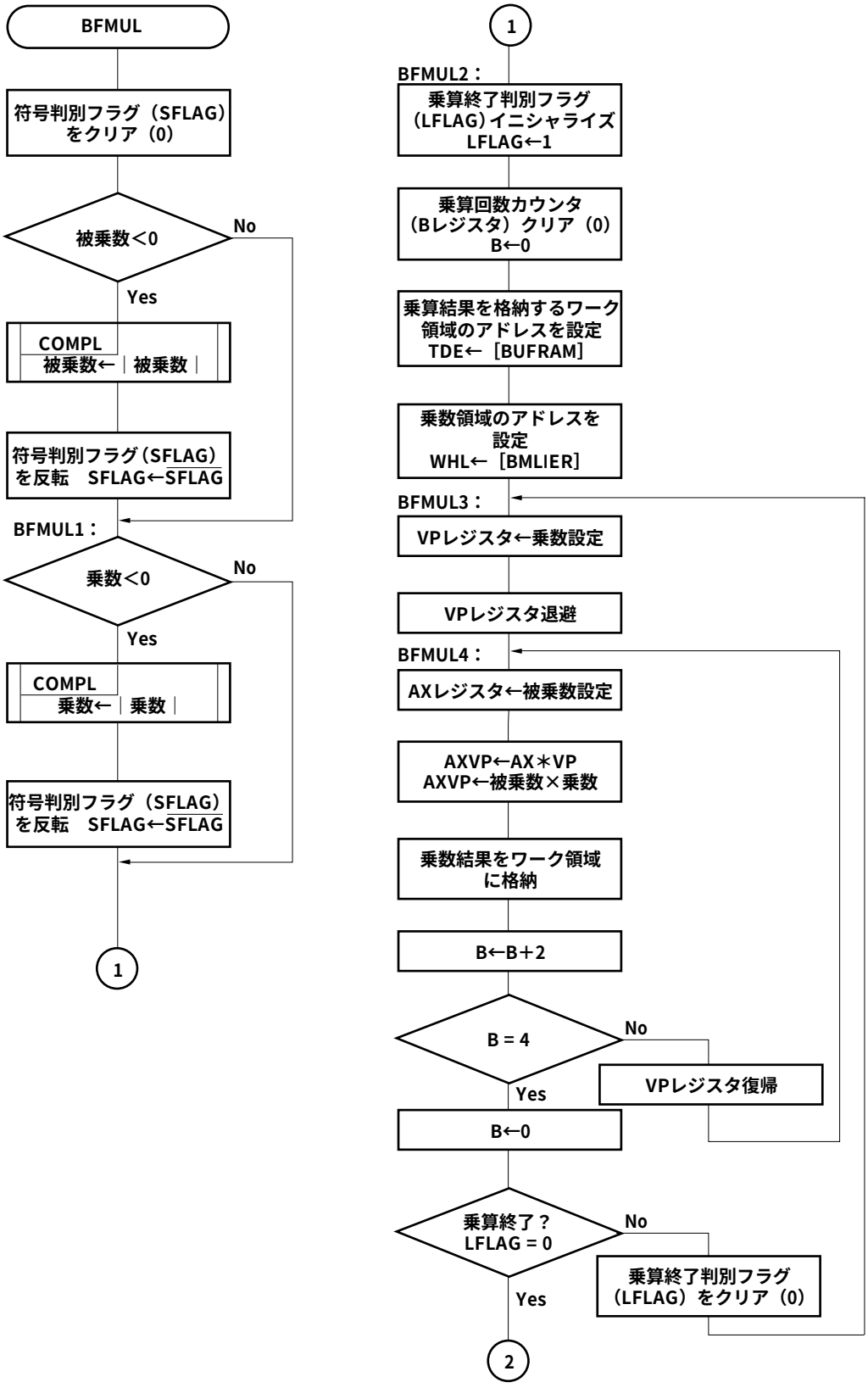
（ｎ）BUFRAM＋８～BUFRAM＋0DHの値を２バイトごと，キャリー・フラグ（CY）と一緒に順次加算し，演算結果を結果領域（BRSLT）の５，６バイト目に格納する。

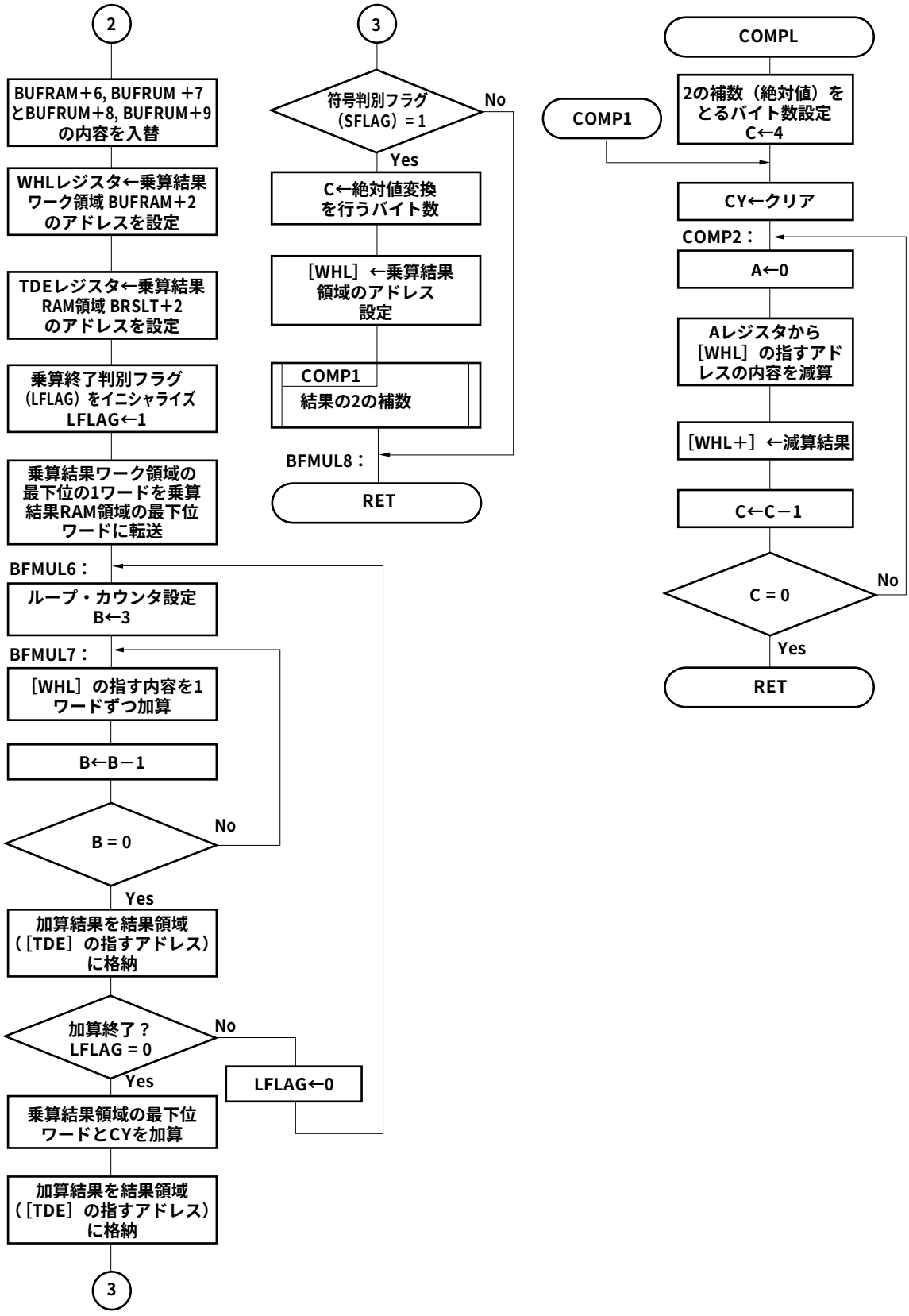
（ｏ）BUFRAM＋0EH, BUFRAM＋0FHの内容とキャリー・フラグ（CY）を加算し，結果領域（BRSLT）の７，８バイト目に格納する。

（ｐ）符号判別フラグ（SFLAG）が１ならば，乗算結果の２の補数を乗算結果とする。

備考　処理１～処理４は，図２－２の２進乗算のアルゴリズムで示す処理番号に対応しています。

（7）フロー・チャート


BFMUL
符号判別フラグ（SFLAG）をクリア（0）
被乗数＜0
No
Yes
COMPL
被乗数←｜被乗数｜
符号判別フラグ（SFLAG）を反転　SFLAG←SFLAG
BFMUL1：
乗数＜0
No
Yes
COMPL
乗数←｜乗数｜
符号判別フラグ（SFLAG）を反転　SFLAG←SFLAG
1
1
BFMUL2：
乗算終了判別フラグ（LFLAG）イニシャライズ
LFLAG←1
乗算回数カウンタ（Bレジスタ）クリア（0）
B←0
乗算結果を格納するワーク領域のアドレスを設定
TDE←［BUFRAM］
乗数領域のアドレスを設定
WHL←［BMLIER］
BFMUL3：
VPレジスタ←乗数設定
VPレジスタ退避
BFMUL4：
AXレジスタ←被乗数設定
AXVP←AX＊VP
AXVP←被乗数×乗数
乗数結果をワーク領域に格納
B←B＋2
B = 4
No
Yes
VPレジスタ復帰
B←0
乗算終了？
LFLAG = 0
No
Yes
乗算終了判別フラグ（LFLAG）をクリア（0）
2


2
BUFRAM＋6, BUFRUM ＋7
とBUFRUM＋8, BUFRUM＋9
の内容を入替
WHLレジスタ←乗算結果
ワーク領域 BUFRAM＋2
のアドレスを設定
TDEレジスタ←乗算結果
RAM領域 BRSLT＋2
のアドレスを設定
乗算終了判別フラグ
(LFLAG) をイニシャライズ
LFLAG←1
乗算結果ワーク領域の
最下位の1ワードを乗算
結果RAM領域の最下位
ワードに転送
BFMUL6：
ループ・カウンタ設定
B←3
BFMUL7：
［WHL］の指す内容を1
ワードずつ加算
B←B－1
B = 0
No
Yes
加算結果を結果領域
(［TDE］の指すアドレス)
に格納
加算終了？
LFLAG = 0
No
Yes
LFLAG←0
乗算結果領域の最下位
ワードとCYを加算
加算結果を結果領域
(［TDE］の指すアドレス)
に格納
3
3
符号判別フラグ
(SFLAG) = 1
No
Yes
C←絶対値変換
を行うバイト数
［WHL］←乗算結果
領域のアドレス
設定
COMP1
結果の2の補数
BFMUL8：
RET
COMPL
2の補数（絶対値）を
とるバイト数設定
C←4
COMP1
CY←クリア
COMP2：
A←0
Aレジスタから
［WHL］の指すアド
レスの内容を減算
［WHL＋］←減算結果
C←C－1
C = 0
No
Yes
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

BMLCND：被乗数32ビットを格納しているRAM領域の最下位アドレス

BMLIER　：乗数32ビットを格納しているRAM領域の最下位アドレス

BUFRAM：乗算結果15バイトをいったん格納するワーク領域の最下位アドレス

BRSLT　：最終的な乗算結果を格納するRAM領域の最下位アドレス

SFLAG　：符号判別フラグ

SFLAG = 0…同符号

SFLAG = 1…異符号

LFLAG　：乗算終了判別フラグ

LFLAG = 0…乗算終了

LFLAG = 1…乗算未終了

●メイン・ルーチンのプログラム・リスト記述例

```
        •
        •
    MOVW    BMLCND  ,#02H
    MOVW    BMLCND+2,#00H

    MOVW    BMLIER  ,#08H
    MOVW    BMLIER+2,#15H
;
    CALL    !BFMUL
        •
        •
```

備考　上記記述例のように被乗数と乗数を設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BFMULR

;*******************************************************
;*      binary  multiplication                         *
;*          input condition                            *
;*              multiplicand <- (BMLCND+3,...,BMLCND)  *
;*              multiplier   <- (BMLIER+3,...,BMLIER)  *
;*          output condition                           *
;*              result <- (BRSLT+7,BRSLT+6...,BRSLT)   *
;*******************************************************

        PUBLIC  BFMUL                   ;
        EXTRN   COMPL,COMP1             ;
        EXTRN   BMLCND,BMLIER,BRSLT
        EXTRN   BUFRAM
        EXTBIT  SFLAG,LFLAG
;
        BYTNUM  EQU     4               ; value length
;
        CSEG
BFMUL:
;
;       *** compliment convert ***
;
        CLR1    SFLAG                   ; sign-flag <- 0
        BF      BMLCND+3.7,$BFMUL1      ; if data<0 go to BFMUL1
        MOVG    WHL,#BMLCND             ; WHL-reg. <- BMLCND
        CALL    !COMPL                  ; complement subroutine
        NOT1    SFLAG                   ; not sign-flag

BFMUL1:
        BF      BMLIER+3.7,$BFMUL2      ; if data<0 go to BFMUL2
        MOVG    WHL,#BMLIER             ; WHL-reg. <- BMLIER
        CALL    !COMPL                  ; complement subroutine
        NOT1    SFLAG                   ; not sign-flag
;
```

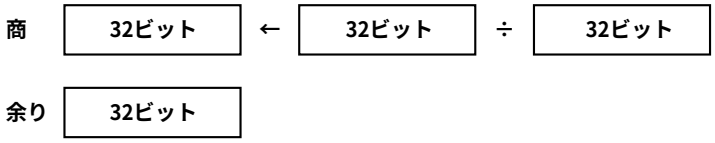
```
;       *** word multiplication process ***
;
BFMUL2:
        SET1    LFLAG                   ;
        MOV     B,#0                    ; MULT loop number clear
        MOVG    TDE,#BUFRAM             ; TDE-reg. <- BUFRAM
        MOVG    WHL,#BMLIER             ; WHL-reg. <- BMLIER
BFMUL3:
        MOVW    AX,[WHL+]               ;
        MOVW    VP,AX                   ; VP <- BMLIER
        PUSH    VP                      ;
BFMUL4:
        MOVW    AX,BMLCND[B]            ; AX <- BMLCND[B]
        MULUW   VP                      ; AXVP <- AX * VP
;
        XCHW    AX,VP                   ;
        MOVW    [TDE+],AX               ;
        XCHW    AX,VP                   ;
        MOVW    [TDE+],AX               ; WORK area <- AXVP
        ADD     B,#2                    ;
        CMP     B,#4                    ;
        BZ      $BFMUL5                 ;
;
        POP     VP                      ;
        BR      BFMUL4                  ;
;
BFMUL5:
        MOV     B,#0                            ;
        BTCLR   LFLAG,$BFMUL3
;
;       ***  multiplicated data add process ***
;
        XCHW    BUFRAM+6,BUFRAM+8       ; BUFRAM+6,BUFRAM+7
                                        ;       <-> BUFRAM+8,BUFRAM+9
        MOVG    WHL,#BUFRAM+2           ;
        MOVG    TDE,#BMSLT+2            ;
        SET1    LFRAG                   ;
        MOVW    RP2,#00H                ;

        MOVW    BRSLT,BUFRAM            ; answer of lower set
BFMUL6:
        MOVW    VP,RP2                  ;
        MOVW    RP2,#0                  ;
        MOV     B,#3                    ; add number set
```

```
BFMUL7:
        MOVW    AX,[WHL+]       ;
        ADDW    VP,AX           ; add
        ADDC    R4,#0           ;
        DBNZ    B,$BFMUL7       ;

        XCHW    AX,VP           ; BMSLT+ <-  data set
        MOVW    [TDE+],AX       ;

        BTCLR   LFLAG,$BFMUL6   ; all add end?
        MOVW    AX,[WHL+]       ; [no] add again
        ADDW    AX,RP2          ;
        MOVW    [TDE],AX        ;
        BF      SFLAG,$BFMUL8   ; if sflag=1 complement convert
        MOV     C,#8
        MOVG    WHL,#BRSLT
        CALL    !COMP1
BFMUL8:
        RET


                NAME    CMPLE

;*************************************************************
;*      complement convert subroutine                        *
;*              input  condition                             *
;*                WHL-register <- complement top.address     *
;*              output condition                             *
;*                (WHL+3,WHL+2,...,WHL) <- convert data      *
;*                                                           *
;*************************************************************

        PUBLIC  COMPL,COMP1
;
BYTNUM  EQU     4               ; value length
        CSEG

COMPL:
        MOV     C,#BYTNUM
COMP1:
        CLR1    CY                  ;
COMP2:
        MOV     A,#0H
        SUBC    A,[WHL]
        MOV     [WHL+],A
        DBNZ    C,$COMP2
        RET
```

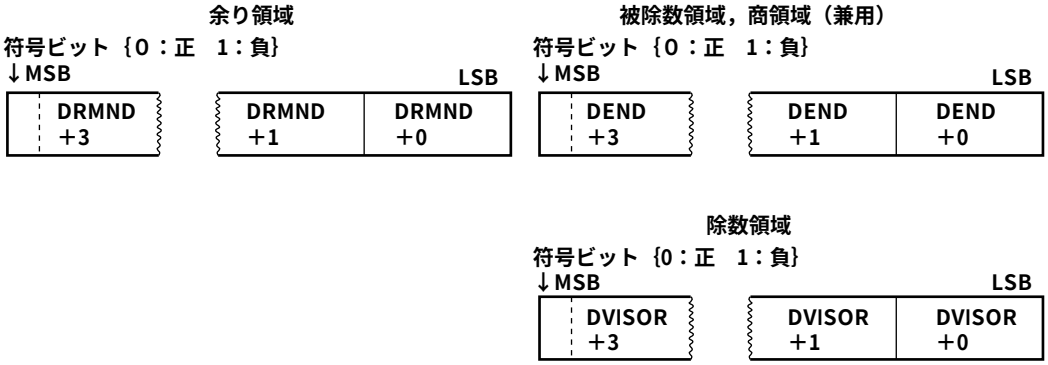
２.４　符号付き32ビット÷32ビットの２進除算

（１）処理概要

符号付き32ビットの被除数から32ビットの除数で除算し，演算結果を32ビットの結果領域に格納するプログラム例を紹介します。



（２）使用RAM領域



符号ビット（０）のとき：正（00000000H～7FFFFFFFH）

符号ビット（１）のとき：負（FFFFFFFFH～80000000H）

注意　被除数，商領域（DEND＋３，…DEND）と，余り領域（DRMND＋３，…DRMND）は８バイト連続したRAM領域に設定してください。

（３）使用レジスタ

Ａ，Ｘ，Ｂ，Ｃ，TDE, WHLレジスタ

（４）入力方法

次に示す各４バイトのRAM領域に，演算に必要なデータを設定します。

DEND～DEND＋３　　：32ビットの被除数データを設定します。

DVISOR～DVISOR＋３：32ビットの除数データを設定します。

（５）出力方法

次に示すフラグに，除算処理状況が設定されます。

ERRFLAG：エラー・フラグ

ERRFLAG＝０…エラー未発生状態（除算正常終了）

ERRFLAG＝１…エラー発生状態（除数が０のため除算できない）

次に示す各４バイトのRAM領域に，次の内容が格納されます。

DEND～DEND＋３　　：除算した演算結果の商が格納されます。[注]

DRMND～DRMND＋３：除算した演算結果の余りが格納されます。[注]

注　エラー・フラグ（ERRFLAG）＝１のとき，これらの値は演算前の値のままです。

備考　メイン・ルーチンでエラー・フラグ（ERRFLAG）の判別を行っています。必要に応じて演算エラー処理を追加作成してください。

（６）プログラム説明

本プログラムでは，２進除算のアルゴリズムとして，引き戻し法を使用しています。
引き戻し法のアルゴリズムを次に示します。

●引き戻し法を用いた除算のアルゴリズム

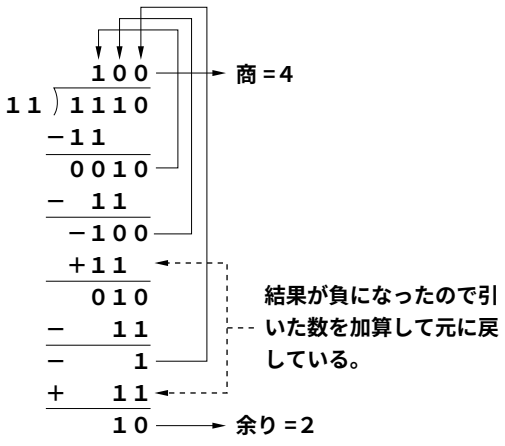
① Q←０
② Y←Y×２$^{m-n}$
③ ｛ X←X－Y
④ if X≧０ then Q←Q＋１
else X←X＋Y
⑤ Q←Q×２
⑥ Y←Y/２ ｝
⑦ ③～⑥の ｛ ｝ 内をｎ回繰り返す
⑧ 結果，商はQ，余りはXに入る。

備考 上記使用の記号の表す意味は，次のとおりです。

Q：商領域
X：被除数領域
Y：除数領域
m：被除数の桁数
n：除数の桁数

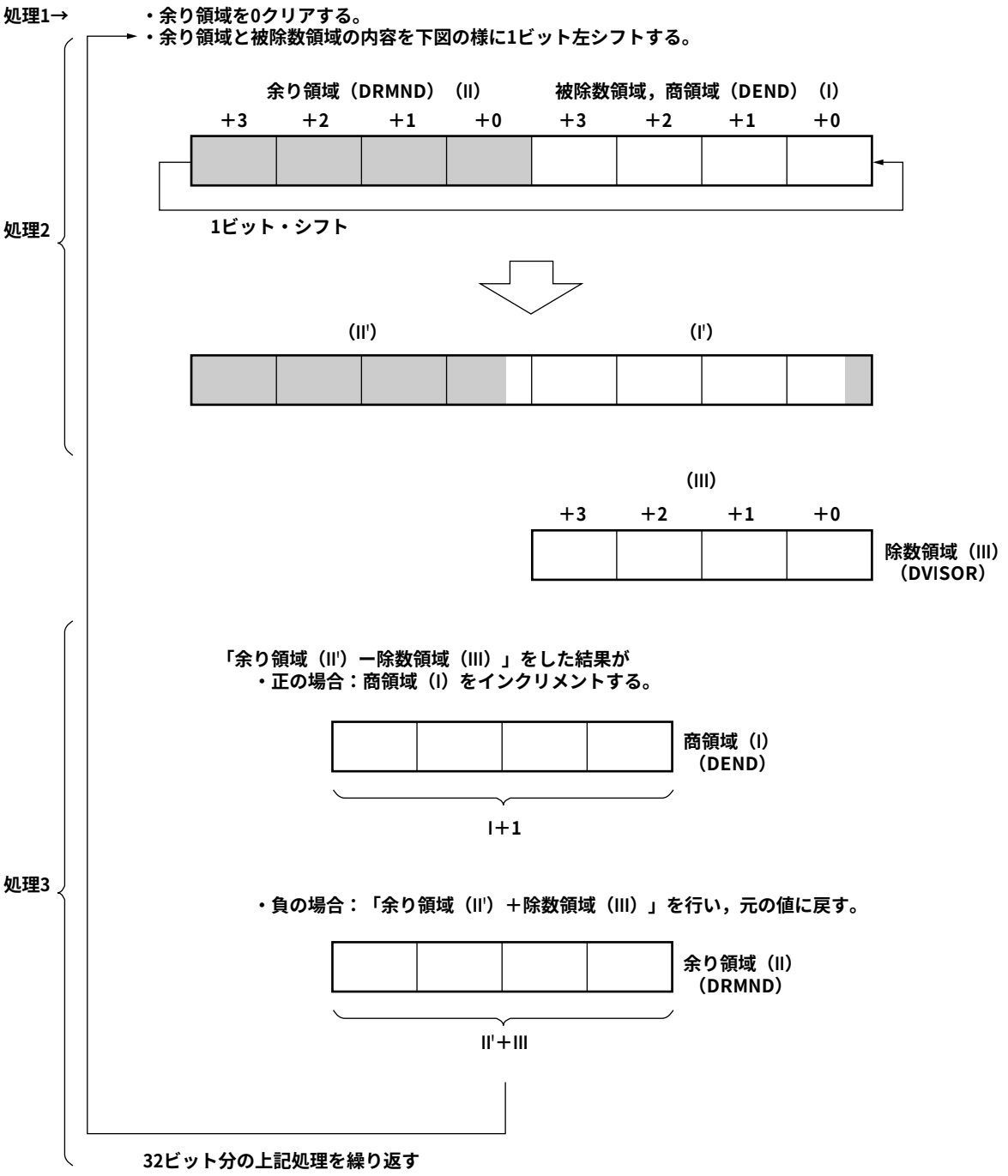
●引き戻し法を用いた２進除算（４ビット÷４ビット）の例

例 14÷３＝商４，余り２（下記計算式は２進数表記）



●本プログラムにおける２進除算（32ビット÷32ビット）のアルゴリズム

図２－３　２進除算のアルゴリズム

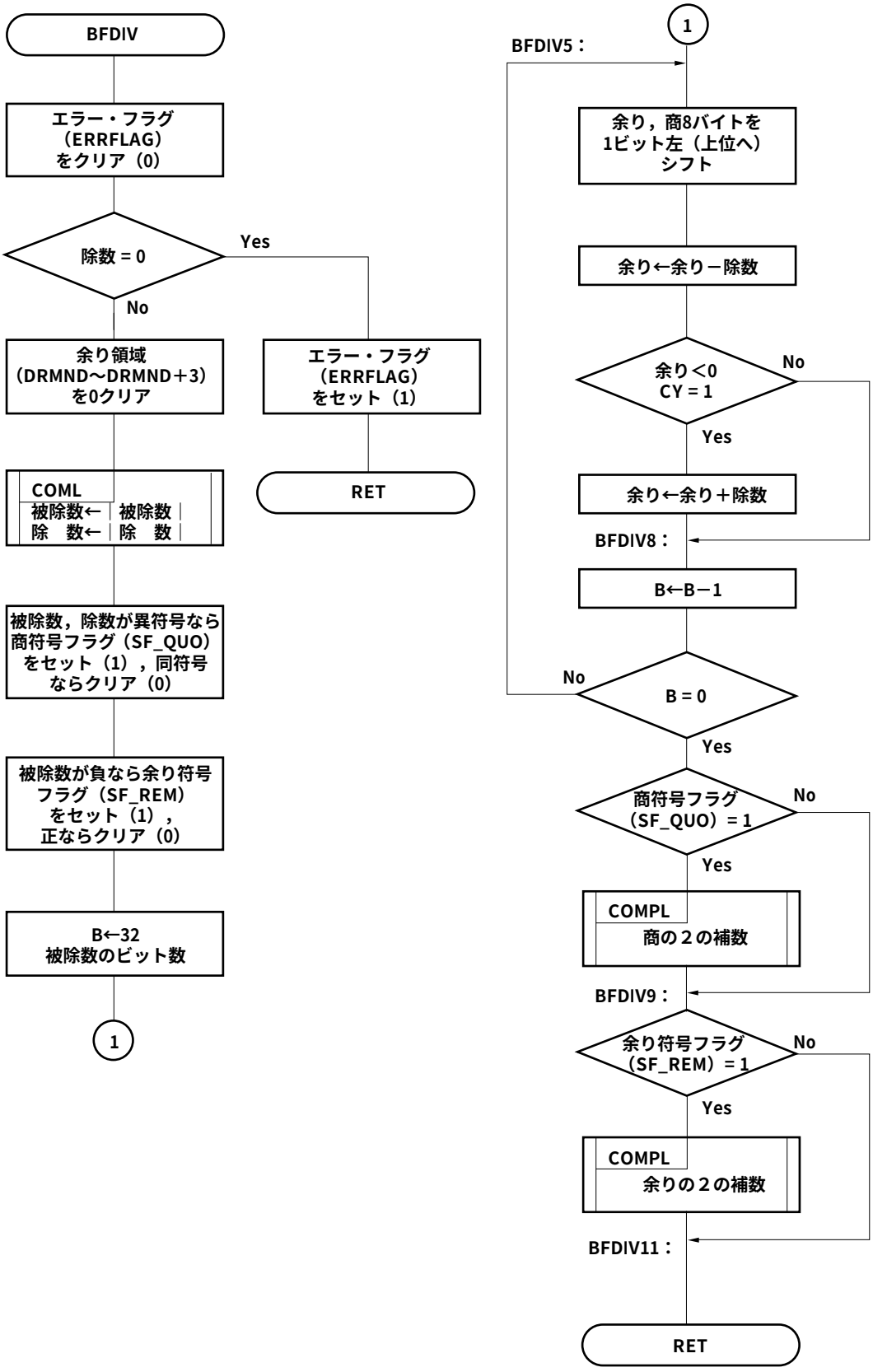


次に，本プログラムの処理手順を示します。

処理１→（ａ）除数領域の値が０か否かを判別する。０ならば，エラー・フラグ（ERRFLAG）をセット（１）し，エラーが発生した情報を残して演算を終了する。
（ERRFLAG＝０…エラー未発生，ERRFLAG＝１…エラー発生）
（ｂ）余り領域をクリア（０）する。
（ｃ）被除数領域と除数領域の値の絶対値をとる。被除数領域，除数領域の値のどちらか一方だけが負ならば，商符号フラグ（QUOFLAG）をセット（１）する。
（QUOFLAG＝０…商の符号が正，QUOFLAG＝１…商の符号が負）
被除数領域の値が負ならば，余り符号フラグ（REMFLAG）をセット（１）する。
（REMFLAG＝０…余りの符号が正，REMFLAG＝１…余りの符号が負）
（ｄ）被除数領域のビット数をカウントするビット・カウンタとしてＢレジスタを使用し，被除数領域のビット数（32）を設定する。
処理２→（ｅ）余り領域と被除数領域（８バイト連続領域）を１ビット左へシフトする。
処理３→（ｆ）“余り領域←余り領域－除数領域”を行う。
演算結果が負ならば（ｈ）へジャンプする。
処理４
（ｇ）商領域（DEND）をインクリメントする。（ｉ）へジャンプする。
（ｈ）引きすぎたので余り領域の値を復元するために，“余り領域←余り領域＋除数領域”を行う。
（ｉ）被除数領域のビット・カウンタ（Ｂレジスタ）をデクリメントし，０になるまで（ｅ）～（ｈ）の処理を繰り返す。
（ｊ）商符号フラグ（QUOFLAG）を調べ，セット（１）されていれば商の２の補数を取る。
余り符号フラグ（REMFLAG）を調べ，セット（１）されていれば余りの２の補数を取る。

備考１． COMPLサブルーチンの詳細については，**２.３　符号付き32ビット×32ビットの２進乗算**を参照してください。
２． 処理１～処理４は，図２－３の２進除算のアルゴリズムで示す処理番号に対応しています。

（7）フロー・チャート


BFDIV
エラー・フラグ（ERRFLAG）をクリア（0）
除数 = 0
Yes
No
エラー・フラグ（ERRFLAG）をセット（1）
RET
余り領域（DRMND～DRMND＋3）を0クリア
COML
被除数← 被除数
除　数← 除　数
被除数，除数が異符号なら商符号フラグ（SF_QUO）をセット（1），同符号ならクリア（0）
被除数が負なら余り符号フラグ（SF_REM）をセット（1），正ならクリア（0）
B←32
被除数のビット数
1
1
BFDIV5：
余り，商8バイトを1ビット左（上位へ）シフト
余り←余り－除数
余り＜0
CY = 1
No
Yes
余り←余り＋除数
BFDIV8：
B←B－1
No
B = 0
Yes
商符号フラグ（SF_QUO）= 1
No
Yes
COMPL
商の２の補数
BFDIV9：
余り符号フラグ（SF_REM）= 1
No
Yes
COMPL
余りの２の補数
BFDIV11：
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

DEND　　：被除数32ビットを格納してあり，商32ビットを格納するRAM領域の最下位アドレス（兼用）

DRMND　：除算結果の余り32ビットを格納するRAM領域の最下位アドレス

DVISOR　：除数32ビットを格納してあるRAM領域の最下位アドレス

BYTNUM　：余り領域のバイト数（余り領域の０クリアに使用）

QUOFLAG ：商符号フラグ

　　QUOFLAG＝０…商の符号が正

　　QUOFLAG＝１…商の符号が負

REMFLAG ：余り符号フラグ

　　REMFLAG＝０…余りの符号が正

　　REMFLAG＝１…余りの符号が負

ERRFLAG ：エラー・フラグ

　　ERRFLAG＝０…エラー未発生状態

　　ERRFLAG＝１…エラー発生状態

●メイン・ルーチンのプログラム・リスト記述例

```
        •
        •
        MOVW    DEND   ,#00          ;data "A"
        MOVW    DEND+2,#32          ;

        MOVW    DVISOR,  #00        ;data "B"
        MOVW    DVISOR+2,#08        ;

        CALL    !BFDIV              ;data "A/B" subroutine

        BT      ERRFLAG,$ERROR      ;
        BR      $$                  ;
ERROR:
        CLR1    ERRFLAG             ;clear error flag
        •
        •
```

備考　上記記述例のように被除数と除数を設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BFDIVR
;************************************************
;*      binary division                         *
;*           32 bit <- 32 bit / 32 bit          *
;*       input condition                        *
;*             dividend <- (DEND+3,...,DEND)    *
;*             divisor  <- (DVISOR+3,...,DVISOR)*
;*      output  condition                       *
;*             quotient  <- (DEND+3,...,DEND)   *
;*             remainder <- (DRMND+3,...,DRMND) *
;*             z flag <- 0:output ok 1: NG      *
;************************************************
;
        PUBLIC  BFDIV
        EXTRN   COMPL
        EXTRN   DEND,DVISOR,DRMND
        EXTBIT  ERRFLAG
        EXTBIT  REMFLAG,QUOFLAG
;
BYTNUM  EQU     4
;
        CSEG
BFDIV:
        CLR1    ERRFLAG         ; clear error flag
;
;       ****  check / divisor = 0 ?  ****
;
        MOVG    WHL,#DVISOR     ; WHL <- DVISOR
        MOVW    AX,[WHL+]       ;
        CMPW    AX,#0           ;
        BNZ     $BFDIV2         ; [WHL] = 0 ?
        MOVW    AX,[WHL+]       ;
        CMPW    AX,#0           ;
        BNZ     $BFDIV2         ; [WHL] = 0 ?
;
;       ****  divisor = 0  ****
;
        SET1    ERRFLAG         ; OVERFLOW
        RET
;
;       ****  quotient 0-clear  ****
```

```
;
BFDIV2:
        MOVG    TDE,#DRMND      ; TDE-register <- DRMND
        MOV     C,#BYTNUM       ;
        MOV     A,#0            ;
        MOVM    [TDE+],A        ;
;
;       ****  complement convert  ****
;
        CLR1    REMFLAG         ; clear remainder sign-flag
        CLR1    QUOFLAG         ; clear quotient sign-flag
        BF      DEND+3.7,$BFDIV3
        MOVG    WHL,#DEND       ; WHL-register <- DEND
        CALL    !COMPL          ; complement subroutine
        SET1    REMFLAG         ; set remainder sign-flag
        NOT1    QUOFLAG         ; not quotient sign-flag
BFDIV3:
        BF      DVISOR+3.7,$BFDIV4
        MOVG    WHL,#DVISOR     ; WHL-register <- DVISOR
        CALL    !COMPL          ; complement subroutine
        NOT1    QUOFLAG         ; not quotient sign-flag
;
;       ****  byte counter set  ****
;
BFDIV4:
        MOV     B,#32           ; B-register <- 32
;
;       ****  dividend,remainder 1-byte left shift  ****
;
BFDIV5:
        CLR1    CY              ;
        MOVG    WHL,#DEND       ; WHL <- DEND
        MOV     C,#8            ; loop counter
BCDLS1:
        MOV     A,[WHL]         ;
        ROLC    A,1             ;
        MOV     [WHL+],A        ;
        DBNZ    C,$BCDLS1       ;
;
;       ****  subtract divisor from dividend  ****
;
BFDIV6:
        MOVG    TDE,#DVISOR     ; TDE <- DVISOR

        MOVW    AX,[TDE+]       ;
        SUBW    DRMND,AX        ;
        MOV     A,[TDE+]        ;
        SUBC    DRMND+2,A       ;
        MOV     A,[TDE+]        ;
        SUBC    DRMND+3,A       ;
        BC      $BFDIV7         ;
        SET1    DEND.0          ;
        BR      BFDIV8          ;
;
;       ****  if borrow divisor + dividend  ****
```

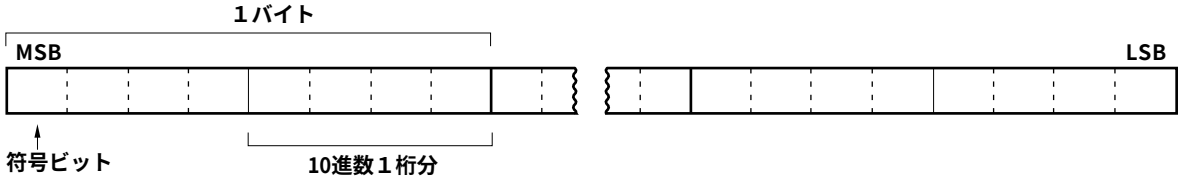
```
;
BFDIV7:
        MOVG    TDE,#DVISOR     ; TDE <- DVISOR

        MOVW    AX,[TDE+]       ;
        ADDW    DRMND,AX        ;
        MOV     A,[TDE+]        ;
        ADDC    DRMND+2,A       ;
        MOV     A,[TDE+]        ;
        ADDC    DRMND+3,A       ;
BFDIV8:
        DBNZ    B,$BFDIV5       ;
;
;       ****  check / division end ?  ****
;
        BF      REMFLAG,$BFDIV9
        MOVG    WHL,#DRMND
        CALL    !COMPL
BFDIV9:
        BF      QUOFLAG,$BFDIV10
        MOVG    WHL,#DEND
        CALL    !COMPL
BFDIV10:
        CLR1    PSWL.6          ;clear z flag
BFDIV11:
        RET
;
```

〔メ　モ〕

第３章　10進演算

10進数の表現は図３－１のように，最上位ビットを符号ビットとし，残りのビットで数値を表現します。10進数はBCDコードで表現されます。

図３－１　10進数の表現

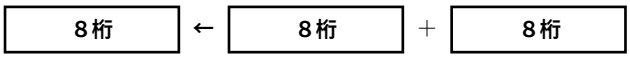


10進演算では，演算に使用するデータ格納領域も，演算後の結果領域も，RAM領域上に配置します。

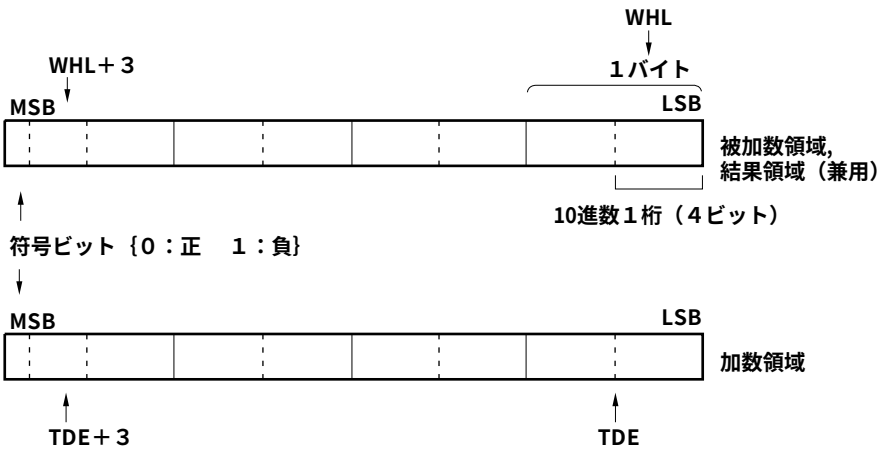
３.１　符号付き８桁＋８桁の10進加算

（１）処理概要

符号付き８桁の被加数と８桁の加数を加算し，演算結果を８桁の結果領域（被加数領域と兼用）に格納するプログラム例を紹介します。



（２）使用RAM領域



符号ビット（０）のとき：正（０～79999999）

符号ビット（１）のとき：負（－１～－79999999）

（３）使用レジスタ

Ａ，Ｃ，Ｂ，TDE, WHLレジスタ

（４）入力方法

WHL, TDEレジスタを次のように設定します。

WHL　：被加数８桁（４バイト）を格納してあるRAM領域の最下位アドレスを設定します。

TDE　：加数８桁（４バイト）を格納してあるRAM領域の最下位アドレスを設定します。

（５）出力方法

次に示すフラグに，除算処理状況が設定されます。

ERRFLAG　：エラー・フラグ

ERRFLAG＝0…エラー未発生状態（加算正常終了）

ERRFLAG＝1…エラー発生状態（オーバフローまたはアンダフロー発生のため加算できない）

WHLレジスタで示す４バイトのRAM領域に，次の内容が格納されます。

WHL～WHL＋３：加算した演算結果が格納されます。[注]

注　エラー・フラグ（ERRFLAG）=１のとき，WHLレジスタで示す４バイトの値は不定となります。

備考１．演算範囲は－79999999～79999999です。
２．メイン・ルーチンでエラー・フラグ（ERRFLAG）の判別を行っています。必要に応じて演算エラー処理を追加作成してください。

（6）プログラム説明

本加算プログラムでは，加数，被加数が同符号ならば加算を行い，異符号ならば減算を行います。

（a）Cカウンタ（Cレジスタ）に10進加算を行うバイト数を設定する。
（b）加数，被加数が異符号ならば（o）の処理へ。
（c）キャリー・フラグ（CY），符号判別フラグ（SFLAG）をクリア（0）する。
（d）被加数アドレス（WHLレジスタ）で示す被加数領域の１バイトをAレジスタに読み込む。
（e）加数アドレス（TDEレジスタ）で示す加数領域の１バイトをキャリー・フラグ（CY）とともにAレジスタに加算し，加数アドレス（TDEレジスタ）をインクリメントする。
演算結果を10進補正し，被加数アドレス（WHLレジスタ）で示される結果領域に格納し，被加数アドレス（WHLレジスタ）をインクリメントする。
（f）カウンタ（Bレジスタ）をデクリメントし，カウンタ（Bレジスタ）が0になるまで（d）から（e）の処理を繰り返す。
（g）被加数アドレス（WHLレジスタ）で示される被加数領域の１バイトをAレジスタに読み込む。
（h）加数アドレス（TDEレジスタ）で示される加数領域の１バイトをキャリー・フラグ（CY）とともにAレジスタに加算する。
（i）キャリー・フラグ（CY）が“０”ならば（k）の処理へ。
（j）符号判別フラグ（SFLAG）をセット（1）し，キャリー・フラグ（CY）をクリア（0）する。
（k）Aレジスタを10進補正する。
（l）キャリー・フラグ（CY）が“１”またはAレジスタの第７ビットが“１”ならばオーバフローなので，エラー・フラグ（ERRFLAG）をセット（1）して演算終了とする。
(m) 符号判別フラグ（SFLAG）が“１”ならばAレジスタの第７ビットをセット（1）する。
（n）Aレジスタの内容を被加数アドレス（WHLレジスタ）で示される結果領域に格納し，演算終了とする。
（o）減数を正にし，符号判別フラグ（SFLAG）をクリア（0）する。
（p）被減数が負ならば被減数を正にし，符号判別フラグをセット（1）する。
（q）キャリー・フラグ（CY）をクリア（0）する。

（ｒ）被減数アドレス（WHLレジスタ）で示される被減数領域の１バイトをＡレジスタに読み込む。

（ｓ）Ａレジスタから減数アドレス（TDEレジスタ）で示される減数領域の１バイトをキャリー・フラグ（CY）とともに引き，減数アドレス（TDEレジスタ）をインクリメントする。演算結果を10進補正し，被減数アドレス（WHLレジスタ）で示される結果領域に格納し，その後，被減数アドレス（WHLレジスタ）をインクリメントする。

（ｔ）カウンタ（Ｃレジスタ）をデクリメントし，カウンタ（Ｃレジスタ）が0になるまで，（ｒ）から（ｓ）の処理を繰り返す。

（ｕ）キャリー・フラグ（CY）が“０”ならば（ｗ）の処理へ。

（ｖ）結果の10の補数をとり，符号判別フラグ（SFLAG）を反転する。

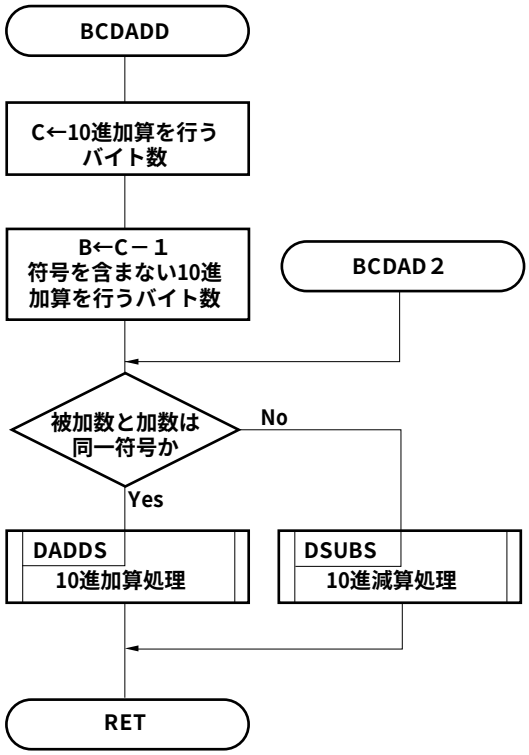
（ｗ）演算結果が０ならば演算終了とする。

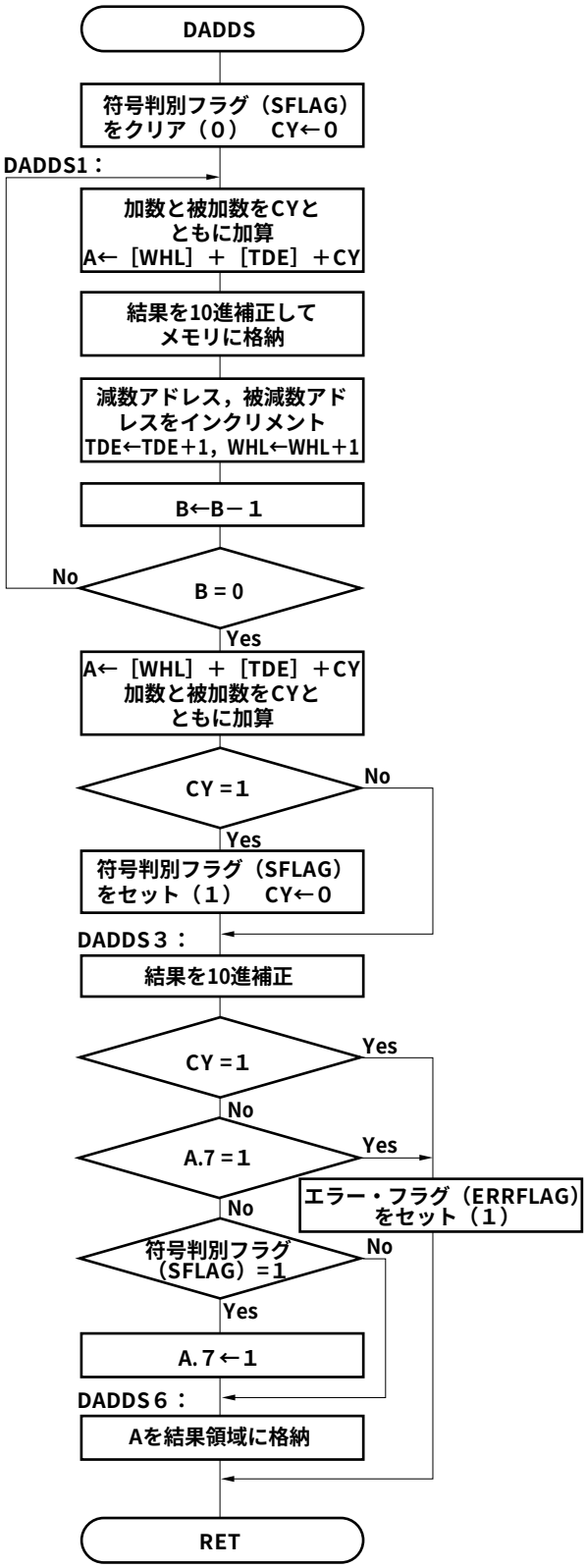
（ｘ）符号判別フラグ（SFLAG）が“１”ならば（ｙ）の処理へ，“０”ならば演算終了とする。

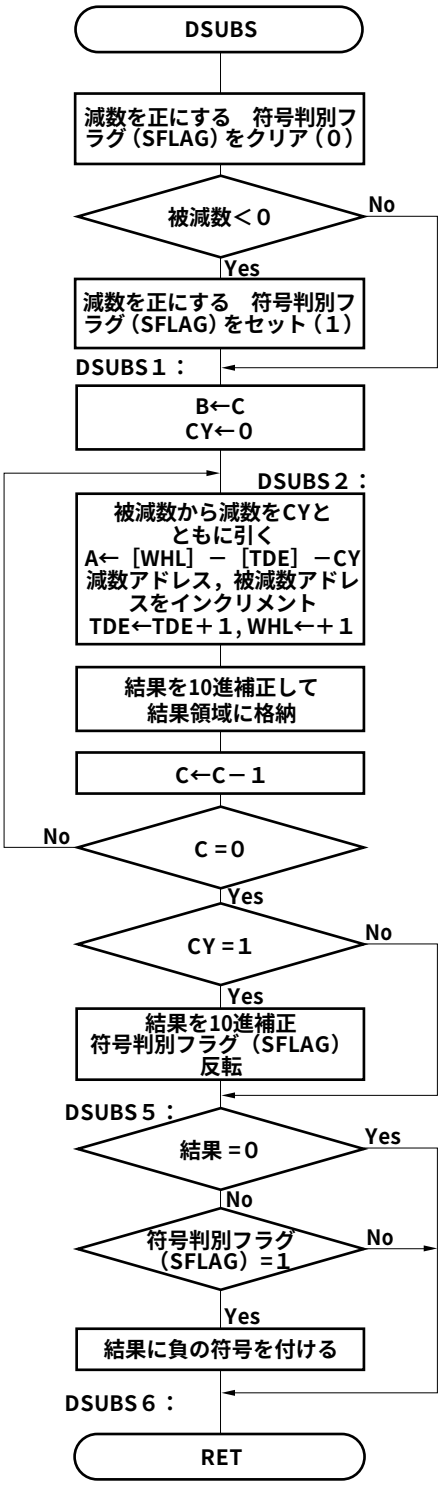
（ｙ）結果の符号ビットをセット（１）し，演算終了とする。

備考　10進減算ルーチンの処理（（ｏ）～（ｙ）の処理）では，被加算領域は，被減算領域に，加数領域は，減算領域と記述しています。

（7）フロー・チャート




DADDS
符号判別フラグ（SFLAG）
をクリア（0）　CY←0
DADDS1：
加数と被加数をCYと
ともに加算
A←［WHL］＋［TDE］＋CY
結果を10進補正して
メモリに格納
減数アドレス，被減数アド
レスをインクリメント
TDE←TDE＋1，WHL←WHL＋1
B←B－1
No
B＝0
Yes
A←［WHL］＋［TDE］＋CY
加数と被加数をCYと
ともに加算
CY＝1
No
Yes
符号判別フラグ（SFLAG）
をセット（1）　CY←0
DADDS3：
結果を10進補正
CY＝1
Yes
No
A.7＝1
Yes
エラー・フラグ（ERRFLAG）
をセット（1）
No
符号判別フラグ
（SFLAG）＝1
No
Yes
A.7←1
DADDS6：
Aを結果領域に格納
RET


DSUBS
減数を正にする　符号判別フ
ラグ（SFLAG）をクリア（0）
被減数＜0
No
Yes
減数を正にする　符号判別フ
ラグ（SFLAG）をセット（1）
DSUBS1：
B←C
CY←0
DSUBS2：
被減数から減数をCYと
ともに引く
A←［WHL］－［TDE］－CY
減数アドレス，被減数アドレ
スをインクリメント
TDE←TDE＋1，WHL←＋1
結果を10進補正して
結果領域に格納
C←C－1
No
C＝0
Yes
CY＝1
No
Yes
結果を10進補正
符号判別フラグ（SFLAG）
反転
DSUBS5：
結果＝0
Yes
No
符号判別フラグ
（SFLAG）＝1
No
Yes
結果に負の符号を付ける
DSUBS6：
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

BCDAUG　：被加数８桁（４バイト）を格納してあり，結果８桁を格納するRAM領域の最下位アドレス（兼用）

BCDADE　：加数８桁（４バイト）を格納してあるRAM領域の最下位アドレス

SFLAG　　：符号判別フラグ

　　SFLAG＝0…同符号

　　SFLAG＝1…異符号

ERRFLAG：エラー・フラグ

　　ERRFLAG＝0…エラー未発生状態

　　ERRFLAG＝1…エラー発生状態

●メイン・ルーチンのプログラム・リスト記述例

```
        •
        •
        MOVG    WHL,#BCDAUG         ;
        MOVG    TDE,#BCDADE         ;

        CALL    !BCDADD
;
        BT      ERRFLAG,$ERROR      ;
        BR      $$                  ;
ERROR:
        CLR1    ERRFLAG             ; clear error flag
        •
        •
```

備考　上記記述例のようにWHL, TDEレジスタを設定し，サブルーチンを呼んでください。また，エラー処理は必要に応じて追加作成してください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BCDADR
;*******************************************************
;*      decimal addition                               *
;*           8 digit <- 8 digit + 8 digit              *
;*       input condition                               *
;*            WHL-register <- augend area top.address  *
;*            TDE-register <- addend area top.address  *
;*       output condition                              *
;*             result <- (WHL,WHL+1,WHL+2,WHL+3)       *
;*******************************************************

        PUBLIC  BCDADD,BCDAD1,BCDAD2
        PUBLIC  DADDS
        PUBLIC  DSUBS
        EXTBIT  SFLAG           ; work flag for sign flag
        EXTBIT  ERRFLAG         ; error sign flag

BYTNUM  EQU     4
        CSEG
BCDADD:
        MOV     C,#BYTNUM       ; C-register <- 4
BCDAD1:
        MOV     B,C             ; B-register <- C-register - 1
        DEC     B
BCDAD2:
        MOV     A,[WHL+BYTNUM-1]
        XOR     A,[TDE+BYTNUM-1]

        CLR1    ERRFLAG         ; clear error flag
        BT      A.7,$BCDAD3
        CALL    !DADDS
        BR      EBCDAD

BCDAD3:
        CALL!DSUBS
EBCDAD:
        RET
```

```
;================================================
;       ***** decimal addition subroutine *****
;================================================

DADDS:
        CLR1    CY
        CLR1    SFLAG           ; clear sign-flag
DADDS1:
        MOV     A,[WHL]
        ADDC    A,[TDE+]
        ADJBA                   ; decimal adjust
        MOV     [WHL+],A
        DBNZ    B,$DADDS1

        MOV     A,[WHL]
        ADDC    A,[TDE]
DADDS2:
        BNC     $DADDS3
        SET1    SFLAG           ; set sign-flag
        CLR1    CY
DADDS3:
        ADJBA                   ; decimal adjust
        BNC     $DADDS4
        BR      DADDS7
DADDS4:
        BF      A.7,$DADDS5
        BR      DADDS7
DADDS5:
        BF      SFLAG,$DADDS6
        SET1    A.7
DADDS6:
        MOV     [WHL],A
        BR      EDADDS

DADDS7:
        SET1    ERRFLAG         ; set error flag
EDADDS:
        RET
```

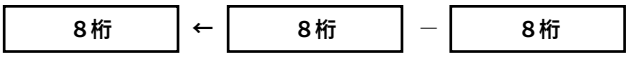
```
;=====================================================
;       ***** decimal subtraction subroutine *****
;=====================================================

DSUBS:
        PUSH    WHL             ; save WHL-register
        CLR1    SFLAG           ; clear sign-flag
        MOV     A,[TDE+BYTNUM-1]
        CLR1    A.7
        MOV     [TDE+BYTNUM-1],A
        MOV     A,[WHL+BYTNUM-1]
        BF      A.7,$DSUBS1

        CLR1    A.7
        MOV     [WHL+BYTNUM-1],A
        SET1    SFLAG           ; set sign-flag
DSUBS1:
        MOV     B,C             ; save C-register
        CLR1    CY
DSUBS2:
        MOV     A,[WHL]
        SUBC    A,[TDE+]
        ADJBS                   ; decimal adjust
        MOV     [WHL+],A
        DBNZ    C,$DSUBS2

        BNC     $DSUBS5
        POP     WHL             ; load WHL-register
        PUSH    WHL             ; save WHL-register
        MOV     C,B             ; load C-register
DSUBS3:
        MOV     A,#99H          ; (WHL) <- 9 - (WHL)
        SUB     A,[WHL]         ;     increment WHL-register
        ADJBS                   ; decimal adjust
        MOV     [WHL+],A
        DBNZ    C,$DSUBS3

        POP     WHL             ; load WHL-register
        PUSH    WHL             ; save WHL-register
        SET1    CY

        MOV     C,B             ; load C-register
DSUBS4:
        MOV     A,#0            ; Acc <- 0
        ADDC    A,[WHL]
        ADJBA                   ; decimal adjust
        MOV     [WHL+],A
        DBNZ    C,$DSUBS4
        NOT1    SFLAG
```

```
;
;       ****  check / result = 0  ****
;
DSUBS5:
        MOV     C,B             ; load C-register
        POP     WHL             ; load WHL-register
        MOVG    TDE,WHL
        MOV     A,#0
        CMPME   [TDE+],A
        BZ      $EDSUBS

        BF      SFLAG,$EDSUBS
        MOV     A,[WHL+BYTNUM-1]
        SET1    A.7             ; set sign
        MOV     [WHL+BYTNUM-1],A
EDSUBS:
        RET
```

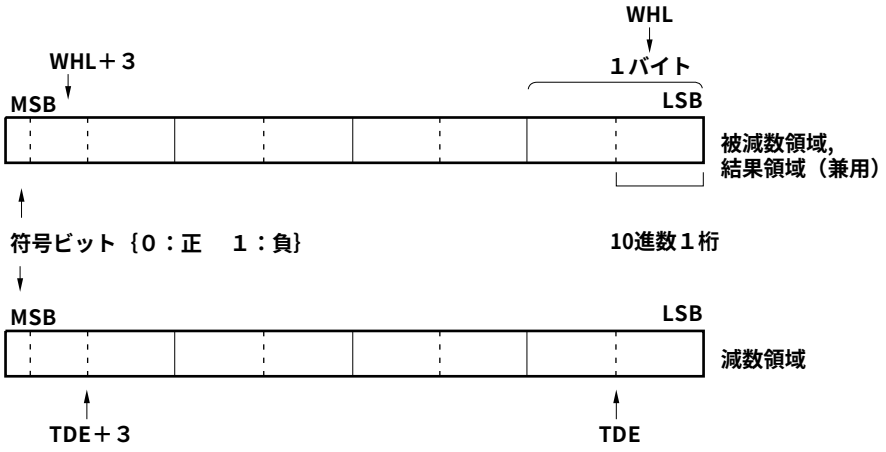
３.２　符号付き８桁－８桁の10進減算

（１）処理概要

符号付き８桁の被減数から８桁の減数を減算し，演算結果を８桁の結果領域（被減数領域と兼用）に格納するプログラム例を紹介します。



（２）使用RAM領域



符号ビット（０）のとき：正（０～79999999）

符号ビット（１）のとき：負（－１～－79999999）

（３）使用レジスタ

Ａ，Ｂ，Ｃ，TDE, WHLレジスタ

（４）入力方法

WHL, TDEレジスタを次のように設定します。

WHL　：被減数８桁（４バイト）を格納してあるRAM領域の最下位アドレスを設定します。

TDE　：減数８桁（４バイト）を格納してあるRAM領域の最下位アドレスを設定します。

（５）出力方法

次に示すフラグに，除算処理状況が設定されます。

ERRFLAG：エラー・フラグ

ERRFLAG=０…エラー未発生状態（減算正常終了）

ERRFLAG=１…エラー発生状態（オーバフローまたはアンダフロー発生のため減算できない）

WHLレジスタで示す４バイトのRAM領域に，次の内容が格納されます。

WHL～WHL＋３：減算した演算結果が格納されます。[注]

注　エラー・フラグ（ERRFLAG）=１のとき，WHLレジスタで示す４バイトの値は不定となります。

備考１．演算範囲は－79999999～79999999です。

２．メイン・ルーチンでエラー・フラグ（ERRFLAG）の判別を行っています。必要に応じて演算エラー処理を追加作成してください。

（６）プログラム説明

本減算プログラムでは，“被減数－減数”を“被減数＋（－減数）”という加算処理変換して行っています。

（ａ）Ｃカウンタ（Ｃレジスタ）に10進減算を行うバイト数を設定する。

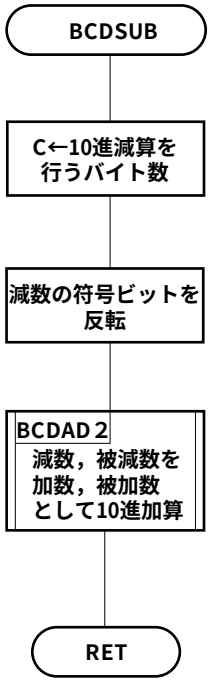
（ｂ）減数領域の符号ビットを反転する。

（ｃ）被減数領域，減数領域をそれぞれ被加数領域，加数領域として10進加算の処理を行う。

備考　BCDAD2サブルーチンの詳細（エラー・フラグ（ERRFLAG）をセット（１）する処理を含む）は，**３.１　符号付き８桁＋８桁の10進加算**を参照してください。

また，演算エラー処理はメイン・ルーチンに含まれています。必要に応じてエラー処理を追加作成してください。

（7）フロー・チャート



（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

BCDMIN ：被減数８桁（４バイト）を格納してあり，結果８桁を格納するRAM領域の最下位 アドレス（兼用）

BCDSUT ：減数８桁（４バイト）を格納してあるRAM領域の最下位アドレス

●メイン・ルーチンのプログラム・リスト記述例

```
          •
          •
        MOVG    WHL,#BCDMIN
        MOVG    TDE,#BCDSUT

        CALL    !BCDSUB
        BT      ERRFLAG,$ERROR      ;
        BR      $$                  ;
ERROR:
        CLR1    ERRFLAG             ; clear error flag
          •
          •
```

備考　上記記述例のようにWHL, TDEレジスタを設定し，サブルーチンを呼んでください。また，エラー処理は必要に応じて追加作成してください。

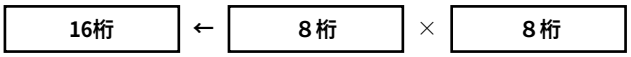
●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BCDSUR
;*************************************************************
;*      decimal subtraction                                  *
;*           8 digit <- 8 digit - 8 digit                    *
;*       input condition                                     *
;*            WHL-register <- minus value area top.address   *
;*            TDE-register <- subtrahend area top.address    *
;*       output condition                                    *
;*             result <- (WHL,WHL+1,WHL+2,WHL+3)             *
;*************************************************************
        PUBLIC  BCDSUB
        EXTRN   BCDAD2
BYTNUM  EQU     4
;
        CSEG
BCDSUB:
        MOV     C,#BYTNUM       ; C-register <- 4
BCDSU1:
        MOV     B,C             ; B-register <- C-register - 1
        DEC     B
        MOV     A,[TDE+BYTNUM-1]
        NOT1    A.7
        MOV     [TDE+BYTNUM-1],A
        CALL    !BCDAD2
        RET
```

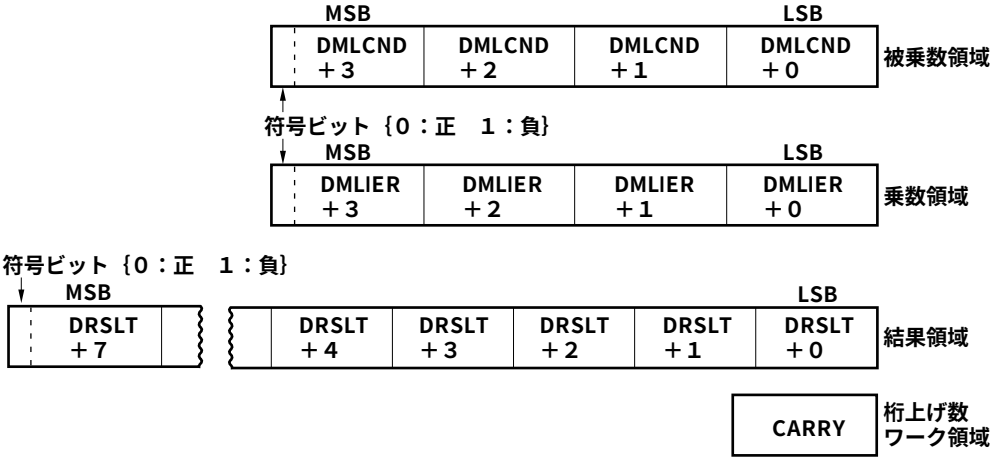
３.３　符号付き８桁×８桁の10進乗算

（１）処理概要

符号付き８桁の被乗数と８桁の乗数を乗算し，演算結果を16桁の結果領域に格納するプログラム例を紹介します。



（２）使用RAM領域



符号ビット（０）のとき：正（０～79999999）

符号ビット（１）のとき：負（－１～－79999999）

（３）使用レジスタ

Ａ，Ｘ，Ｂ，Ｃ，TDE, WHLレジスタ

（４）入力方法

次に示す各４バイトのRAM領域に，演算に必要なデータを設定します。

DMLCND～DMLCND＋３　：８桁の被乗数データを設定します。

DMLIER～DMLIER＋３　：８桁の乗数データを設定します。

（５）出力方法

次に示す８バイトのRAM領域に，次の内容が格納されます。

DRSLT～DRSLT＋７：乗算した演算結果が格納されます。

備考　乗数と被乗数の有効範囲は－79999999～79999999です。
演算結果範囲は－6399999840000001～6399999840000001です。

（６）処理手順

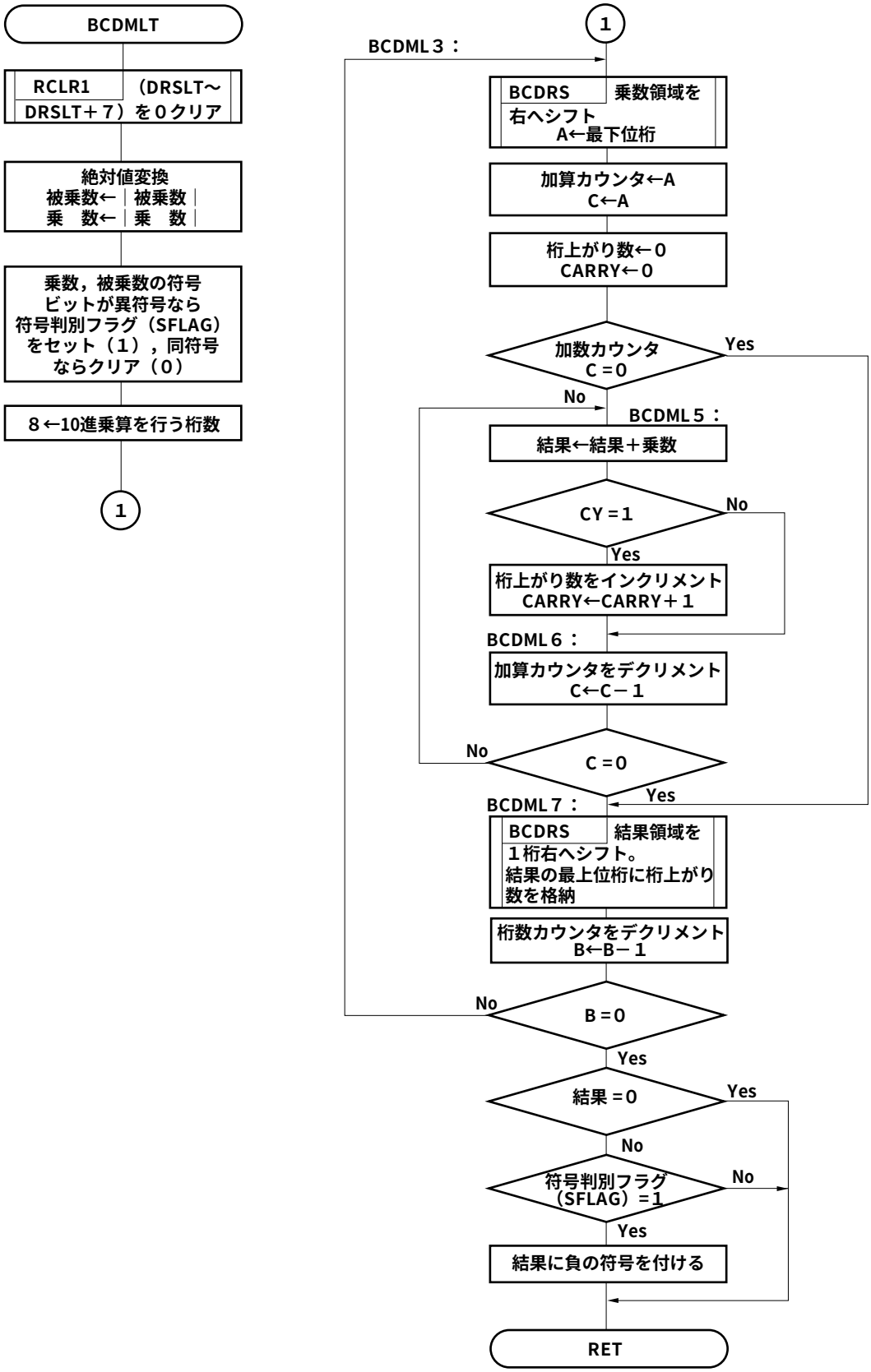
本乗算プログラムでは，乗数を１桁（４ビット）右シフトすることにより，最下位桁から１桁ずつ加算カウンタにロードします。これをカウンタとして，“結果←結果＋被乗数”を繰り返しています。

また，加算カウンタによる加算が終了後，１桁上位の乗数によって加算を行うため，結果領域を１桁（４ビット）右シフトしておきます。

次に処置手順を示します。

（ａ）結果領域をクリア（０）する。
（ｂ）乗数領域と被乗数領域の値の絶対値をとる。乗数領域と被乗数領域の値が異符号なら符号判別フラグ（SFLAG）をセット（１）し，同符号なら符号判別フラグ（SFLAG）をクリア（０）する。
（ｃ）桁カウンタとしてＢレジスタを使用し，乗算の桁数８を設定する。
（ｄ）乗数領域を１桁（４ビット）右シフトすることにより，乗数領域の最下位桁（４ビット）の値を加数カウンタ（Ｃレジスタ）に設定する。
（ｅ）あらかじめ領域を確保している，桁上げ数を退避させておくための桁上げ数ワーク領域（CARRY）をクリア（０）する。
（ｆ）加数カウンタ（Ｃレジスタ）が，０なら（ｉ）の処理へ進む。
（ｇ）10進加算処理“結果領域（上位８桁）←結果領域（上位８桁）＋被乗数領域”を行い，オーバフローが生じたら桁上げ数ワーク領域（CARRY）をインクリメントする。
（ｈ）加数カウンタ（Ｃレジスタ）をデクリメントし，０になるまで（ｇ）の処理を繰り返す。
（ｉ）桁上げ数ワーク領域（CARRY）を含めて，結果領域を１桁（４ビット）右シフトする。
桁上げ数ワーク領域（CARRY）が，結果領域の最上位桁に格納される。
（ｊ）桁カウンタ（Ｂレジスタ）をデクリメントし，０になるまで（ｄ）から（ｉ）の処理を繰り返す。
（ｋ）桁カウンタ（Ｂレジスタ）の内容が０なら演算終了とする。
（ｌ）符号判別フラグ（SFLAG）が“１”ならば結果領域の符号ビットをセット（１）する。

（7）フロー・チャート


BCDMLT
RCLR1　（DRSLT～DRSLT＋７）を０クリア
絶対値変換
被乗数←｜被乗数｜
乗　数←｜乗　数｜
乗数，被乗数の符号ビットが異符号なら符号判別フラグ（SFLAG）をセット（１），同符号ならクリア（０）
８←10進乗算を行う桁数
1
BCDML３：
1
BCDRS　乗数領域を右へシフト
A←最下位桁
加算カウンタ←A
C←A
桁上がり数←０
CARRY←０
加数カウンタ C＝0
Yes
No
BCDML５：
結果←結果＋乗数
CY＝1
No
Yes
桁上がり数をインクリメント
CARRY←CARRY＋１
BCDML６：
加算カウンタをデクリメント
C←C－１
No
C＝0
Yes
BCDML７：
BCDRS　結果領域を１桁右へシフト。
結果の最上位桁に桁上がり数を格納
桁数カウンタをデクリメント
B←B－１
No
B＝0
Yes
結果＝0
Yes
No
符号判別フラグ（SFLAG）＝1
No
Yes
結果に負の符号を付ける
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

DMLCND：被乗数８桁（４バイト）を格納してあるRAM領域の最下位アドレス

DMLIER　：乗数８桁（４バイト）を格納してあるRAM領域の最下位アドレス

CARRY　：桁上げ数ワーク領域

DRSLT　：結果16桁（８バイト）を格納するRAM領域の最下位アドレス

SFLAG　：符号判別フラグ

SFLAG＝0…同符号

SFLAG＝1…異符号

●メイン・ルーチンのプログラム・リスト記述例

```
     •
     •
MOVW    DMLCND   ,#12H
MOVW    DMLCND+2,#00

MOVW    DMLIER   ,#54H
MOVW    DMLIER+2,#12H

CALL    !BCDMLT
     •
     •
```

備考　上記記述例のように被乗数と乗数を設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BCDMLR
;*******************************************************
;*      decimal multiplication                         *
;*           16 digit <- 8 digit * 8 digit             *
;*       input condition                               *
;*             multiplicand <- (DMLCND+3,...,DMLCND)   *
;*             multiplier   <- (DMLIER+3,...,DMLIER)   *
;*       output condition                              *
;*             result <- (DRSLT+7,...,DRSLT)           *
;*******************************************************

        PUBLIC  BCDMLT
        EXTRN   DMLCND,DMLIER,DRSLT
        EXTRN   CARRY                   ;carry data set ram
        EXTBIT  SFLAG                   ;

        CSEG
BCDMLT:
;
;       ****  result area 0-clear  ****
;
        MOV     C,#8                    ; C-register <- 8
        MOVG    TDE,#DRSLT              ; TDE <- DRSLT
        MOV     A,#0                    ; Acc <- 0
        MOVM    [TDE+],A                ;
;
;       ****  check / sign  ****
;
        CLR1    SFLAG                   ; clear sign-flag
        BF      DMLCND+3.7,$BCDML1
        CLR1    DMLCND+3.7
        NOT1    SFLAG                   ; not sign-flag
BCDML1:
        MOV     A,[WHL]
        BF      DMLIER+3.7,$BCDML2
        CLR1    DMLIER+3.7
        NOT1    SFLAG                   ; not sign-flag
;
;       ****  digit counter set  ****
;
BCDML2:
        MOV     B,#8                    ; B-register <- 8
;
;       ****  multiplier right shift  ****
;
```

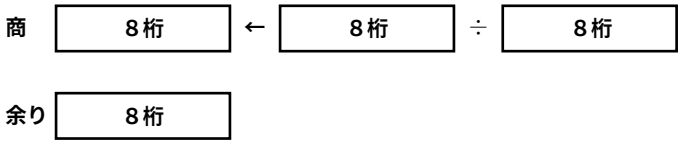
```
BCDML3:
        MOVG    WHL,#DMLIER+3
        MOV     C,#4                    ; C-register <- 4
;
        MOV     A,#0                    ; Acc <- 0
BCDRS:
        ROR4    [WHL]
        DECG    WHL                     ; decrement (WHL)
        DBNZ    C,$BCDRS

        MOV     C,A                     ; C-register <- Acc
        MOV     CARRY,#0                ; carry <- 0
;
;       ****  check / multiplier = 0 ? ****
;
        ADD     A,#0
        BZ      $BCDML7                 ; if Acc = 0 then go to BCDML6
;
;       ****  result <- DMLCND + result  ****
;
BCDML4:
        MOVG    TDE,#DMLCND             ; TDE <- DMLCND
        MOVG    WHL,#DRSLT+4            ; WHL <- DRSLT+4
        CLR1    CY                      ; clear carry
        PUSH    AX                      ; save AX-register
        PUSH    BC                      ; save BC-register
        MOV     C,#4                    ; C-register <- 4
```

```
BCDML5:
        MOV     A,[WHL]
        ADDC    A,[TDE+]
        ADJBA                           ; decimal adjust
        MOV     [WHL+],A
        DBNZ    C,$BCDML5
        POP     BC                      ; load BC-register
        POP     AX                      ; load AX-register
        BNC     $BCDML6
        INC     CARRY
BCDML6:
        DBNZ    C,$BCDML4
;
;       ****  result right shift with carry  ****
;
BCDML7:
        MOV     A,CARRY
        MOVG    WHL,#DRSLT+7            ; WHL <- DRSLT+7
        MOV     C,#8
BCDRS1:
        ROR4    [WHL]
        DECG    WHL                     ; decrement (WHL)
        DBNZ    C,$BCDRS1
;
;       ****  check / multiply end ?  ****
;
        DBNZ    B,$BCDML3
;
;       ****  check / multiply = 0  ****
;
        MOVG    TDE,#DRSLT
        MOV     C,#8
        MOV     A,#0
BCDML8:
        CMPME   [TDE+],A
        BZ      $BCDML9
;
;       ****  check / sign-flag  ****
;
        BF      SFLAG,$BCDML9
        MOVG    WHL,#DRSLT+7
        MOV     A,[WHL]
        SET1    A.7
        MOV     [WHL],A
BCDML9:
        RET
```

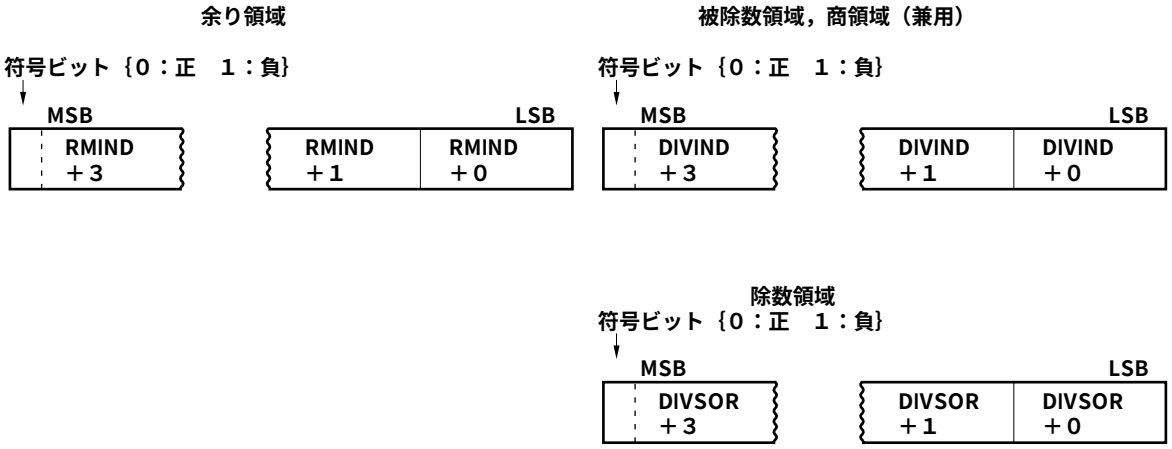
３.４　符号付き８桁÷８桁の10進除算

（１）処理概要

符号付き８桁の被除数から８桁の除数を除算し，演算結果を８桁の結果領域に，余りを８桁の余り領域に格納するプログラム例を紹介します。



（２）使用RAM領域



注意　被除数，商領域（DIVIND，…DIVIND＋３）と，余り領域（RMIND，…RMIND＋３）は，８バイト連続したRAM領域に設定してください。

符号ビット（０）のとき：正（０～79999999）
符号ビット（１）のとき：負（－１～－79999999）

（３）使用レジスタ

Ａ，Ｘ，Ｂ，Ｃ，TDE, WHLレジスタ

備考　Ｘレジスタについて，被除数，除数の絶対値をとったときは次のようになります。

・QUOFLAG：Ｘレジスタの第０ビットを商符号フラグとする。
・REMFLAG：Ｘレジスタの第１ビットを余り符号フラグとする。

（４）入力方法

次に示す各４バイトのRAM領域に，演算に必要なデータを設定します。

DIVIND～DIVIND＋３　：８桁の被除数データを設定します。
DIVSOR～DIVSOR＋３：８桁の除数データを設定します。

（５）出力方法

次に示すフラグに，除算処理状況が設定されます。

ERRFLAG：エラー・フラグ
　　ERRFLAG＝０…エラー未発生状態（除算正常終了）
　　ERRFLAG＝１…エラー発生状態（除数が０のため除算できない）

次に示す各４バイトのRAM領域に，次の内容が格納されます。

DIVIND～DIVIND＋３：除算した演算結果の商が格納されます。[注]
RMIND～RMIND＋３：除算した演算結果の余りが格納されます。[注]

注　エラー・フラグ（ERRFLAG）＝１のとき，これらの値は演算前の値のままです。

備考　メイン・ルーチンでエラー・フラグ（ERRFLAG）の判別を行っています。必要に応じて演算エラー処理を追加作成してください。

（６）処理手順

本除算プログラムでは，被除数，商領域（DIVIND，…DIVIND＋３）と，余り領域（RMIND，…RMIND＋３）を８バイトの連続した領域とします。

被除数，余りの１桁左シフトを行うことにより被除数の最上位桁が余りの最下位領域に転送され，商が最上位から１桁ずつ被除数の最下位領域に入ります。

また，商の各桁は，“余り－除数”の結果が負になるまで繰り返したときの回数とします。

（ａ）除数領域の値が０か否かを判別する。０ならば，エラー・フラグ（ERRFLAG）をセット（１）し，エラーが発生した情報を残して演算を終了する。

（ERRFLAG＝０…エラー未発生，ERRFLAG＝１…エラー発生）

（ｂ）余り領域をクリア（０）する。

（ｃ）被除数領域と除数領域の値の絶対値をとる。

被除数領域，除数領域の値のどちらか一方だけが負ならば，商符号フラグ（QUOFLAG）をセット（１）する。

（QUOFLAG＝０…商の符号が正，QUOFLAG＝１…商の符号が負）

被除数が負ならば，余り符号フラグ（REMFLAG）をセット（１）する。

（REMFLAG＝０…余りの符号が正，REMFLAG＝１…余りの符号が負）

（ｄ）被除数領域のビット数をカウントするビット・カウンタとしてＣレジスタを使用し，被除数の領域のビット数（８）を設定する。

（ｅ）余り領域，商領域（８バイト連続領域）を４ビット左へシフトする。

（ｆ）“余り領域←余り領域－除数領域”を行う。

演算結果が負ならば（ｈ）へジャンプする。

（ｇ）商領域（DIVIND）をインクリメントする。（ｉ）へジャンプする。

（ｈ）引きすぎたので余り領域の値を復元するために，“余り領域←余り領域＋除数領域”を行う。

（ｉ）被除数領域のビット・カウント（Ｃレジスタ）をデクリメントし，０になるまで（ｅ）～（ｈ）の処理を繰り返す。

（ｊ）商が０ならば（ｌ）の処理へ。

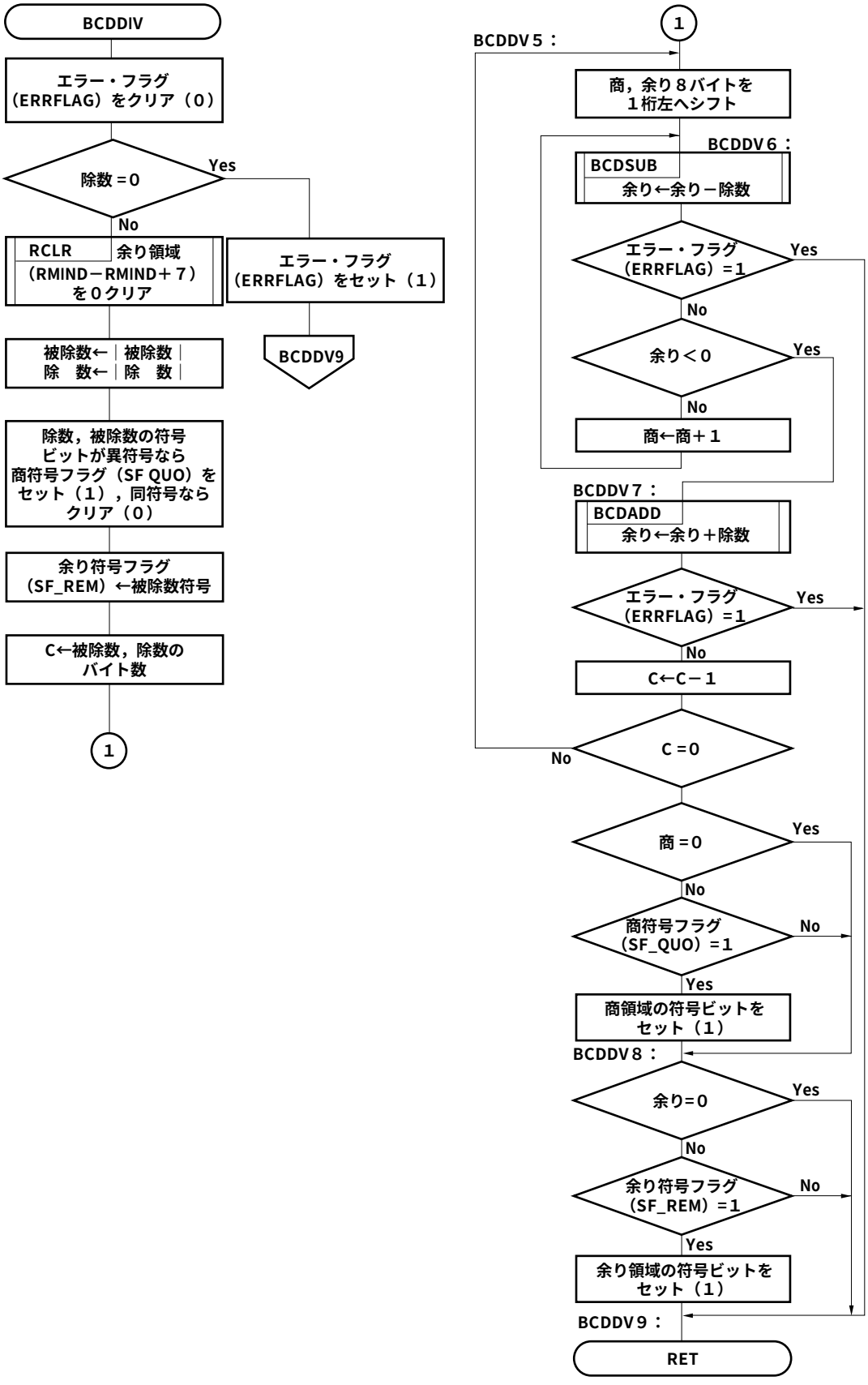
（ｋ）商符号フラグ（QUOFLAG）が“１”ならば商領域の符号ビットをセット（１）する。

（ｌ）余りが０ならば演算を終了する。

（ｍ）余り符号フラグ（REMFLAG）が“１”ならば余り領域の符号ビットをセット（１）する。

備考　BCDAD2サブルーチンの詳細については，**３.１　符号付き８桁＋８桁の10進加算**を
BCDSUBサブルーチンの詳細については，**３.２　符号付き８桁－８桁の10進減算**を参照してください。
エラー・フラグ（ERRFLAG）をセット（１）する処理，演算エラー処理も**３.１　符号付き８桁＋８桁の10進加算**で説明してありますので参照してください。

（7）フロー・チャート


BCDDIV
エラー・フラグ
（ERRFLAG）をクリア（0）
除数＝0
Yes
No
RCLR　余り領域
（RMIND－RMIND＋7）
を0クリア
エラー・フラグ
（ERRFLAG）をセット（1）
BCDDV9
被除数←｜被除数｜
除　数←｜除　数｜
除数，被除数の符号
ビットが異符号なら
商符号フラグ（SF QUO）を
セット（1），同符号なら
クリア（0）
余り符号フラグ
（SF_REM）←被除数符号
C←被除数，除数の
バイト数
1
1
BCDDV５：
商，余り８バイトを
１桁左へシフト
BCDDV６：
BCDSUB
余り←余り－除数
エラー・フラグ
（ERRFLAG）＝1
Yes
No
余り＜0
Yes
No
商←商＋１
BCDDV７：
BCDADD
余り←余り＋除数
エラー・フラグ
（ERRFLAG）＝1
Yes
No
C←C－１
C＝0
No
商＝0
Yes
No
商符号フラグ
（SF_QUO）＝1
No
Yes
商領域の符号ビットを
セット（1）
BCDDV８：
余り=0
Yes
No
余り符号フラグ
（SF_REM）＝1
No
Yes
余り領域の符号ビットを
セット（1）
BCDDV９：
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

DIVSOR　：被除数８桁（４バイト）を格納してあり，商８桁を格納するRAM領域の最下位アドレス（兼用）

DIVIND　：除算結果の余り８桁（４バイト）を格納してある領域の最下位アドレス

RMIND　：除数８桁（４バイト）を格納してある領域の最下位アドレス

QUOFLAG ：商符号フラグ

QUOFLAG＝0…商の符号が正

QUOFLAG＝1…商の符号が負

REMFLAG ：余り符号フラグ

REMFLAG＝0…余りの符号が正

REMFLAG＝1…余りの符号が負

SFLAG　：符号判別フラグ

SFLAG＝0…同符号

SFLAG＝1…異符号

ERRFLAG ：エラー・フラグ

ERRFLAG＝0…エラー未発生状態

ERRFLAG＝1…エラー発生状態

●メイン・ルーチンのプログラム・リスト記述例

```
        •
        •
        MOVW    DIVSOR  ,#42H
        MOVW    DIVSOR+2,#65H

        MOVW    DIVIND  ,#12H
        MOVW    DIVIND+2,#34H

        CALL    !BCDDIV

        BT      ERRFLAG,$ERROR      ;
        BR      $$                  ;
ERROR:
        CLR1    ERRFLAG             ; clear error flag
        •
        •
```

備考　上記記述例のように被除数と除数を設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
NAME    BCDIVR
;*************************************************
;*      decimal division                         *
;*           8 digit <- 8 digit / 8 digit        *
;*       input condition                         *
;*             dividend <- (DIVIND+3,...,DIVIND) *
;*             divisor  <- (DIVSOR+3,...,DIVSOR) *
;*       output condition                        *
;*             quotient  <- (DIVIND+3,...,DIVIND)*
;*             remainder <- (RMIND+3,...,RMIND)  *
;*************************************************
;
        PUBLIC  BCDDIV
        EXTRN   BCDSUB
        EXTRN   BCDADD,BCDAD2
        EXTRN   DIVSOR,DIVIND,RMIND
        EXTBIT  ERRFLAG
        EXTBIT  SFLAG
        EXTBIT  REMFLAG,QUOFLAG

BYTNUM  EQU     4
;
        CSEG
BCDDIV:
        CLR1    ERRFLAG         ; clear error flag
;
;       ****  check / divisor = 0 ?  ****
;
        MOV     C,#4            ; C-register <- 4
        MOV     A,#0            ; Acc <- 0
        MOVG    TDE,#DIVSOR     ; TDE <- DIVSOR
BCDDV1:
        CMPME   [TDE+],A        ; (TDE) = 0 ?
        BNZ     $BCDDV2         ;

        SET1    ERRFLAG         ; overflow
        RET                     ;
;
;       ****  result,remind 0-clear ****
;
BCDDV2:
        MOVG    TDE,#RMIND      ; TDE <- RMIND
        MOV     C,#BYTNUM       ; C-register <- 4
        MOV     A,#0            ; Acc <- 0
        MOVM    [TDE+],A
;
;       ****  check / sign  ****
;
        CLR1    QUOFLAG         ; clear quotient sign-flag
        CLR1    REMFLAG         ; clear remainder sign-flag
```

```
        BF      DIVIND+3.7,$BCDDV3
        CLR1    DIVIND+3.7
        SET1    REMFLAG         ; set remainder sign-flag
        NOT1    QUOFLAG         ; not quotient sign-flag
BCDDV3:
        BF      DIVSOR+3.7,$BCDDV4
        CLR1    DIVSOR+3.7
        NOT1    QUOFLAG
;
;       ****  digit counter set  ****
;
BCDDV4:
        MOV     C,#8
;
;       ****  quotient,remind left shift  ****
;
BCDDV5:
        PUSH    BC
        MOVG    WHL,#DIVIND     ; WHL <- DIVIND
        MOV     C,#16/2         ; C-register <- 8
;
        MOV     A,#0
BCDLS1:
        ROL4    [WHL]
        INCG    WHL             ; increment (WHL)
        DBNZ    C,$BCDLS1
;
;       ****  subtract divisor from dividend  ****
;
BCDDV6:
        MOVG    TDE,#DIVSOR     ; TDE <- DIVSOR
        MOVG    WHL,#RMIND      ; WHL <- RMIND
;
        CALL    !BCDSUB         ; decimal subtraction

        BT      ERRFLAG,$BCDDV9 ; if error then go to BCDDV9

        MOVG    WHL,#RMIND+3
        MOV     A,[WHL]
        BT      A.7,$BCDDV7     ; if borrow then go to BCDDV7

        MOV     A,#1
        MOVG    WHL,#DIVIND
        ADD     A,[WHL]         ; increment (DIVIND)
        MOV     [WHL],A

        BR      BCDDV6
;
;       ****  if borrow then divisor + dividend  ****
```

```
;
BCDDV7:
        MOVG    TDE,#DIVSOR     ; TDE <- DIVSOR
        MOVG    WHL,#RMIND      ; WHL <- RMIND
        CALL    !BCDADD         ; decimal addition

        BT      ERRFLAG,$BCDDV9 ; if error then go to BCDDV9
;
;       ****  check / division end ?  ****
;
        POP     BC
        DBNZ    C,$BCDDV5
;
;       ****  check / quotient = 0  ****
;
        MOVG    TDE,#DIVIND
        MOV     A,#0
        MOV     C,#4
        CMPME   [TDE+],A
        BZ      $BCDDV8
;
;       ****  check / quotient sign-flag  ****
;
        BF      QUOFLAG,$BCDDV8
        MOVG    WHL,#DIVIND+3
        SET1    [WHL].7
;
;       ****  check / remainder = 0  ****
;
BCDDV8:
        MOVG    TDE,#RMIND
        MOV     C,#4
        CMPME   [TDE+],A
        BZ      $BCDDV9
;
;       ****  check / remainder sign-flag  ****
;
        BF      REMFLAG,$BCDDV9
        MOVG    WHL,#RMIND+3
        SET1    [WHL].7
BCDDV9:
        RET
```

〔メ　モ〕

第４章　シフト処理

78K/IVシリーズには，汎用レジスタ（X，A，C，B，E，D，L，H，R4～R11）および汎用レジスタ・ペア（AX, BC, DE, HL, VP, UP, RP2, RP3）の１ビット単位のシフト命令ならびに，［ROR4, ROL4］の４ビット単位のシフト命令が用意されています。

本プログラム例では，次の２種類のシフト例を示します。

（ⅰ）１バイト単位のシフト［XCHM［TDE＋］，A, XCHM［TDE－］，A命令を使用］

（ⅱ）４ビット単位のシフト［ROR4 mem, ROL4 mem命令を使用］

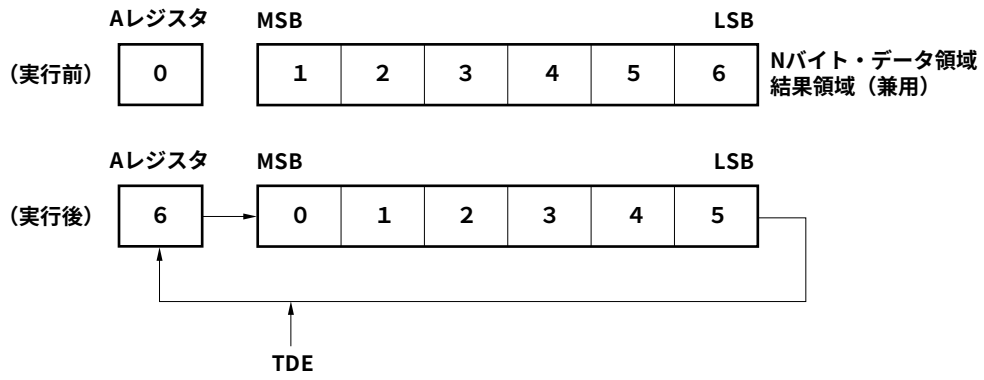
４.１　Ｎバイト・データの１バイト右シフト

（１）処理概要

XCHM命令を利用して，Ｎバイト・データを１バイト右にシフトするプログラム例を紹介します。

下図のように，Ｎバイト・データの１バイト右シフトを行う際，**（４）入力方法**で示す任意の値を各レジスタに設定し，本サブルーチンを実行してください。

（２）使用RAM領域



（３）使用レジスタ

A，C，TDEレジスタ

（４）入力方法

Ａ，Ｃ，TDEレジスタを次のように設定します。

Ａ　　：Ｎバイト・データの最上位アドレスに転送する値を設定します。
Ｃ　　：シフトの対象となるバイト数（Ｎ）を設定します。
TDE　：シフトの対象となるＮバイト・データの最上位アドレスを設定します。

（５）出力方法

TDEレジスタで示すＮバイトのRAM領域に，次の内容が格納されます。

TDE ：Ｎバイト・データを１バイト右シフトした内容が格納されます。
（Ｎバイト・データの最上位には，Ａレジスタの値が入ります）。

（６）プログラム説明

プログラムを実行すると，入力条件で設定したＡレジスタの内容とTDEレジスタで示されたRAM領域の内容を交換すると共に，TDEレジスタの内容をデクリメントします。その後，Ｃレジスタの内容をデクリメントし，Ｃレジスタが０になるまで以上の動作を繰り返します。その結果，Ａレジスタに設定しておいた値が最上位の１バイトに格納され，最下位の１バイトの内容がＡレジスタに出力され，６バイト・データの１バイト右シフトが完了します。

（７）フロー・チャート

なし

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

AREGDT　　：Nバイト・データの最上位アドレスに転送する値

BYTNUM　　：シフトの対象となるバイト数（N）

R6SIFT　　：Nバイト・データの最下位アドレス

●メイン・ルーチンのプログラム・リスト記述例

ここでは，６バイト・データ時の設定例を示します。

```
            •
            •
BYTNUM  EQU     6
            •
            •
        MOV     A,#AREGDT           ; shift in data
        MOV     C,#BYTNUM           ; shift byte number
        MOVG    TDE,#R6SIFT+5       ;

        CALL    !BYTRST             ;right shift
            •
            •
```

備考　上記記述例のようにA，C，TDEレジスタを設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME     BYTRSR

;*************************************************
;*      1_byte data right shift of 6-byte data     *
;*           input condition                       *
;*              TDE-register <- MSB of N-byte data *
;*               C -register <- byte counter       *
;*           output condition                      *
;*               Acc <- LSB of 6-byte data         *
;*************************************************
;
        PUBLIC   BYTRST
;
        CSEG
BYTRST:
        XCHM    [TDE-],A
        RET
```

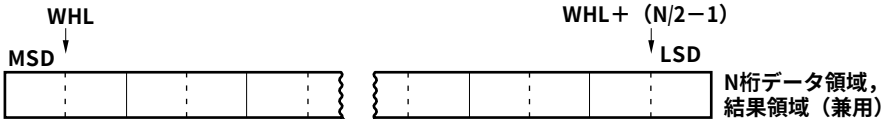
４.２　Ｎ桁データの１桁右シフト（10進 1/10処理）

（１）処理概要

４ビット単位のシフト命令（ROR4 mem, ROL4 mem）を利用して，N桁データを１桁（４ビット）右にシフトするプログラム例を紹介します。

N桁データの１桁（４ビット）右シフトを行う際，**（４）入力方法**で示す任意の値を各レジスタに設定し，本サブルーチンを実行してください。

（２）使用RAM領域



（３）使用レジスタ

Ａ，Ｃ，WHLレジスタ

（４）入力方法

WHL，Ｃレジスタを次のように設定します。

WHL ：N桁データを格納するRAM領域の最上位アドレスを設定します。
Ｃ　　：シフトするバイト数（N/2）を設定します。

（５）出力方法

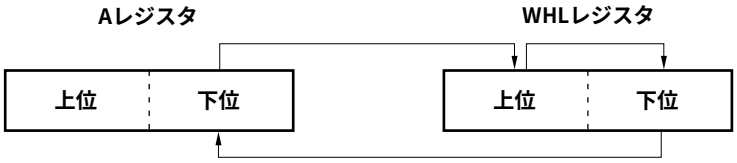
TDEレジスタで示すN/2バイトのRAM領域に，次の内容が格納されます。

TDE ：N桁データを１桁（４ビット）右シフトした内容が格納されます。
（N桁データの最上位桁には，０が入ります）。

（６）プログラム説明

（ａ）Ａレジスタに０を設定する。

（ｂ）Ａレジスタの下位４ビットとWHLレジスタで示されるRAM領域の内容を右方向へローテートする。



（ｃ）シフト回数をカウントするシフト回数カウンタとしてＣレジスタを使用し，カウンタ値をデクリメントする。

　シフト回数カウンタ（Ｃレジスタ）が０になれば処理を終了し，０でなければ（ｂ）の処理へ戻る。

（７）フロー・チャート

なし

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

BYTNUM　　：シフトの対象となるバイト数（N/2）

R4SIFT　　：N桁データを格納してあるRAM領域の最下位アドレス

●メイン・ルーチンのプログラム・リスト記述例

ここでは，８桁データ時の設定例を示します。

```
            •
            •
BYTNUM  EQU     4
            •
            •
MOV     C,#BYTNUM           ;8 digit / 2
MOVG    WHL,#R4SIFT+3       ;
CALL    !BCDRS              ;right shift
            •
            •
```

備考　上記記述例のようにC，WHLレジスタを設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    BCDRSR
;*******************************************************
;*      N-digit data right shift                       *
;*          input condition                            *
;*             WHL-register <- MSD of N-digit data     *
;*              C -register <- digit counter           *
;*          output condition                           *
;*              Acc <- LSD of N-digit data             *
;*******************************************************
        PUBLIC  BCDRS,BCDRS1
;
        CSEG
BCDRS:
        MOV     A,#0            ; Acc <- 0
BCDRS1:
        ROR4    [WHL]
        DECG    WHL             ; decrement (WHL)
        DBNZ    C,$BCDRS1
                RET             ;
```

第５章　ブロック転送処理

ここでは78K/IVシリーズ特有の命令である，ストリング命令の代表的な命令を用いて，ブロック転送プログラムについて説明します。

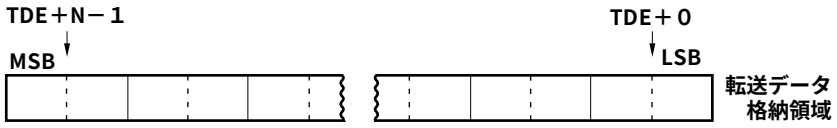
５.１　固定のバイト・データのブロック転送処理

（１）処理概要

MOVM命令を利用して，任意の１バイト・データの値を，指定したＮバイトのRAM領域に格納するプログラム例を紹介します。

この処理は，特定のRAM領域のイニシャライズなどに有効です。

（２）使用RAM領域



（３）使用レジスタ

Ａ，Ｃ，TDEレジスタ

（４）入力方法

Ａ，Ｃ，TDEレジスタを次のように設定します。

Ａ　　：転送データを設定します。
Ｃ　　：転送バイト数を設定します。
TDE　：Ｎバイト転送データを格納するRAM領域の最下位アドレスを設定します。

（５）出力方法

TDEレジスタで示すアドレスのRAM領域に，次の内容が格納されます。

TDE：Ａレジスタの内容がＮバイトのRAM領域に格納されます。

（６）プログラム説明

（ａ）TDEレジスタにＮバイト転送データを格納するRAM領域の最下位アドレスを設定します。

（ｂ）転送バイト数をカウントする転送カウンタとしてＣレジスタを使用し，転送バイト数（BYTNUM）を設定します。

（ｃ）Ａレジスタに転送データを設定します。

（ｄ）Ａレジスタの内容をTDEレジスタで指定されるRAM領域に転送し，転送カウンタ（Ｃレジスタ）をデクリメントします。

転送カウンタ（Ｃレジスタ）が０になると処理を終了します。

（７）フロー・チャート

なし

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

BYTNUM　：ブロック転送の対象となるバイト数（Ｎ）

DATASET ：Ｎバイト・データを転送するRAM領域の最下位アドレス

●本アプリケーション・ルーチンのプログラム・リスト

ここでは，８バイトのRAM領域すべてに，０の値を転送する例を示します。

```
            •
            •
BYTNUM  EQU     8
            •
            •
        MOVG    TDE,#DATASET        ;
        MOV     C,#BYTNUM           ; C   <- byte number(8byte)
        MOV     A,#0                ; Acc <- 0
        MOVM    [TDE+],A            ;
            •
            •
```

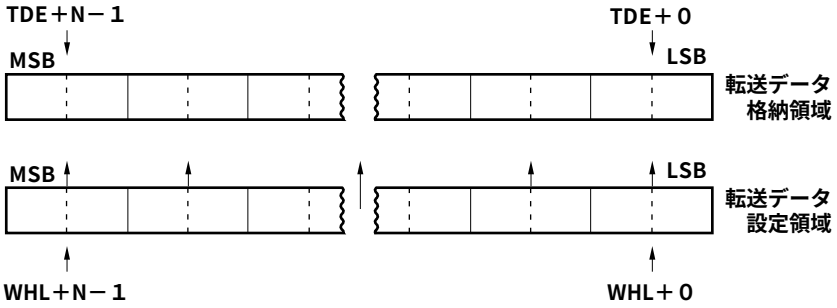
備考　上記のようにプログラムを組むと，TDEレジスタでアドレス指定されたRAM領域に，Ｃレジスタに設定した回数分，Ａレジスタの値が格納されます。

５.２　バイト・データのブロック転送処理

（１）処理概要

MOVBK命令（MOVBK［TDE＋］，［WHL＋］）を利用して，WHLレジスタで示すＮバイトのRAM領域の内容を，TDEレジスタで示すＮバイトのRAM領域にブロック転送するプログラム例を紹介します。

（２）使用RAM領域



（３）使用レジスタ

Ａ，Ｃ，TDE, WHLレジスタ

（４）入力方法

TDE, WHL，Ｃレジスタを次のように設定します。

TDE ：Ｎバイトの転送データを格納するRAM領域の最下位アドレスを設定します。
WHL ：転送するＮバイト・データを設定しているRAM領域の最下位アドレスを設定します。
Ｃ　 ：転送バイト数を設定します。

（５）出力方法

TDEレジスタで示すアドレスのRAM領域に，次の内容が格納されます。

TDE：WHLレジスタで示すNバイトのRAM領域の内容が格納されます。

（６）プログラム説明

（ａ）WHLレジスタに，転送するＮバイト・データを設定しているRAM領域の最下位アドレスを設定します。

（ｂ）TDEレジスタに，転送するＮバイト・データを格納するRAM領域の最下位アドレスを設定します。

（ｃ）転送バイト数をカウントする転送カウンタとしてＣレジスタを使用し，転送バイト数（BYTNUM）を設定します。

（ｄ）WHLレジスタで指定されるRAM領域のデータをTDEレジスタで指定されるRAM領域に転送し，転送カウンタ（Ｃレジスタ）をデクリメントします。

転送カウンタ（Ｃレジスタ）が０になると処理を終了します。

（７）フロー・チャート

なし

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

BYTNUM　：ブロック転送の対象となるバイト数（Ｎ）

TENSODT ：転送するＮバイト・データを設定しているRAM領域の最下位アドレス

KAKUNOU ：転送するＮバイト・データを格納するRAM領域の最下位アドレス

●本アプリケーション・ルーチンのプログラム・リスト

ここでは，８バイトのRAM領域の内容を転送する例を示します。

```
          •
          •
BYTNUM  EQU     8
          •
          •
        MOVG    WHL,#TENSODT        ; set address
        MOVG    TDE,#KAKUNOU        ;
        MOV     C,#BYTNUM           ; C   <- byte number(8byte)
        MOVBK   [TDE+],[WHL+]       ;
          •
          •
```

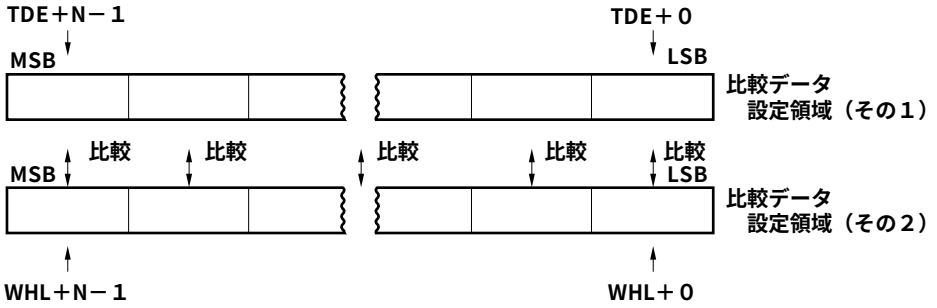
備考　上記のようにプログラムを組むと，TDEレジスタでアドレス指定された領域に，Ｃレジスタに設定した回数分，WHLレジスタでアドレス指定されたRAM領域の値が格納されます。

５.３　バイト・データのブロック比較（一致検出）処理

（１）処理概要

CMPBKE命令（CMPBKE［TDE＋］，［WHL＋］）を利用して，WHLレジスタで示すＮバイトのRAM領域の内容とTDEレジスタで示すＮバイトのRAM領域の内容をブロック比較（一致検出）するプログラム例を紹介します。

（２）使用RAM領域



（３）使用レジスタ

Ａ，Ｃ，TDE, WHLレジスタ

（４）入力方法

TDE, WHL，Ｃレジスタを次のように設定します。

TDE ：比較するＮバイト・データを設定しているRAM領域（その１）の最下位アドレスを設定します。

WHL ：比較するＮバイト・データを設定しているRAM領域（その２）の最下位アドレスを設定します。

Ｃ　 ：比較バイト数を設定します。

（５）出力方法

Ｚ（ゼロ・フラグ）に比較（一致検出）結果が設定されます。

Ｚ（ゼロ・フラグ）：PSW（プログラム・ステータス・ワード）中のＺ（ゼロ・フラグ）

Ｚ＝０…比較結果が不一致

Ｚ＝１…比較結果が一致

（６）プログラム説明

（ａ）WHLレジスタに比較するＮバイト・データを設定しているRAM領域（その１）の最下位アドレスを設定します。

（ｂ）TDEレジスタに比較するＮバイト・データを設定しているRAM領域（その２）の最下位アドレスを設定します。

（ｃ）比較バイト数をカウントする比較カウンタとしてＣレジスタを使用し，比較バイト数（BYTNUM）を設定します。

（ｄ）WHLレジスタで指定されるRAM領域のデータとTDEレジスタで指定されるRAM領域のデータを比較し，比較カウンタ（Ｃレジスタ）をデクリメントします。

比較結果が不一致であるか，比較カウンタ（Ｃレジスタ）が０になると処理を終了します。

（７）フロー・チャート

プログラムのステップ数が少ないため，省略します。

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

COMPDTA ：比較するＮバイト・データを格納しているRAM領域（その１）の最下位アドレス

COMPDTB ：比較するＮバイト・データを格納しているRAM領域（その２）の最下位アドレス

BYTNUM　：ブロック比較の対象となるバイト数（Ｎ）

●本アプリケーション・ルーチンのプログラム・リスト

ここでは，８バイトのRAM領域の内容を比較する例を示します。

```
            •
            •
BYTNUM  EQU     8
            •
            •
        MOVG    WHL,#COMPDTA            ;
        MOVG    TDE,#COMPDTB            ;
        MOV     C,#BYTNUM               ; C   <- byte number(8byte)

        CMPBKE  [TDE+],[WHL+]           ;
            •
            •
```

備考　上記のようにプログラムを組むと，TDEレジスタでアドレス指定された領域の値と，WHLレジスタでアドレス指定された領域の値をＣレジスタに設定した回数分，比較します。

第６章　データ変換処理

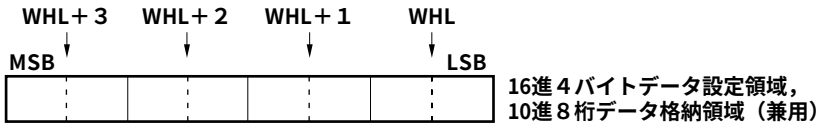
ここでは，数値データの表現形式を，16進と10進，16進とASCIIコードの各形式間で変換するプログラムについて説明します。

６.１　16進（HEX）を10進（BCD）に変換

（１）処理概要

16進４バイトのデータを10進８桁へ変換するプログラム例を紹介します。

（２）使用RAM領域



（３）使用レジスタ

Ａ，Ｘ，Ｄ，Ｅ，UP, WHLレジスタ

（４）入力方法

WHLレジスタを次のように設定します。

WHL：変換する16進４バイトの値を格納してあるRAM領域の最下位アドレスを設定します。

（５）出力方法

次に示すフラグに変換処理状況が設定されます。

ERRFLAG：エラー・フラグ

ERRFLAG＝０…エラー未発生状態（データ変換正常終了）

ERRFLAG＝１…エラー発生状態（変換するデータが範囲外のため変換できない）

WHLレジスタで示す４バイトの領域に次の内容が格納されます。

WHL～WHL＋３：変換された10進８桁の値が格納されます。[注]

注　エラー・フラグ（ERRFLAG）=１のとき，WHLレジスタで示す４バイトの値は不定となります。

備考１．変換範囲は，00000000H（0）～05F5E0FFH（99999999）です。

２．メイン・ルーチンでエラー・フラグ（ERRFLAG）の判別を行っています。必要に応じて演算エラー処理を追加作成してください。

（6）プログラム説明

（ａ）WHLレジスタで示される16進４バイト・データと，05F5EFFHを比較する。

（ｂ）16進４バイト・データが05F5E0FFHより大きい場合は変換範囲外のため，エラー・フラグ（ERRFLAG）をセット（１）して変換を終了する。

（ｃ）AX, DEレジスタに変換する16進４バイト・データを設定する。
（AXレジスタに上位２バイト，DEレジスタに下位２バイトを設定する）。

（ｄ）除数（10）をUPレジスタに設定する。

（ｅ）“16進４バイト・データ／除数”を行う（UPレジスタには余りが入る）。

（ｆ）余りをWHLレジスタで示される10進８桁データ格納領域の下位４ビットに格納する。

（ｇ）AX, DEレジスタに保持されている商を除数と比較して，商の方が大きければ（ｈ）の処理を行い，小さければWHLレジスタで示される10進８桁データ格納領域の上位４ビットに商を格納する。

（ｈ）除数（10）をUPレジスタに設定する。

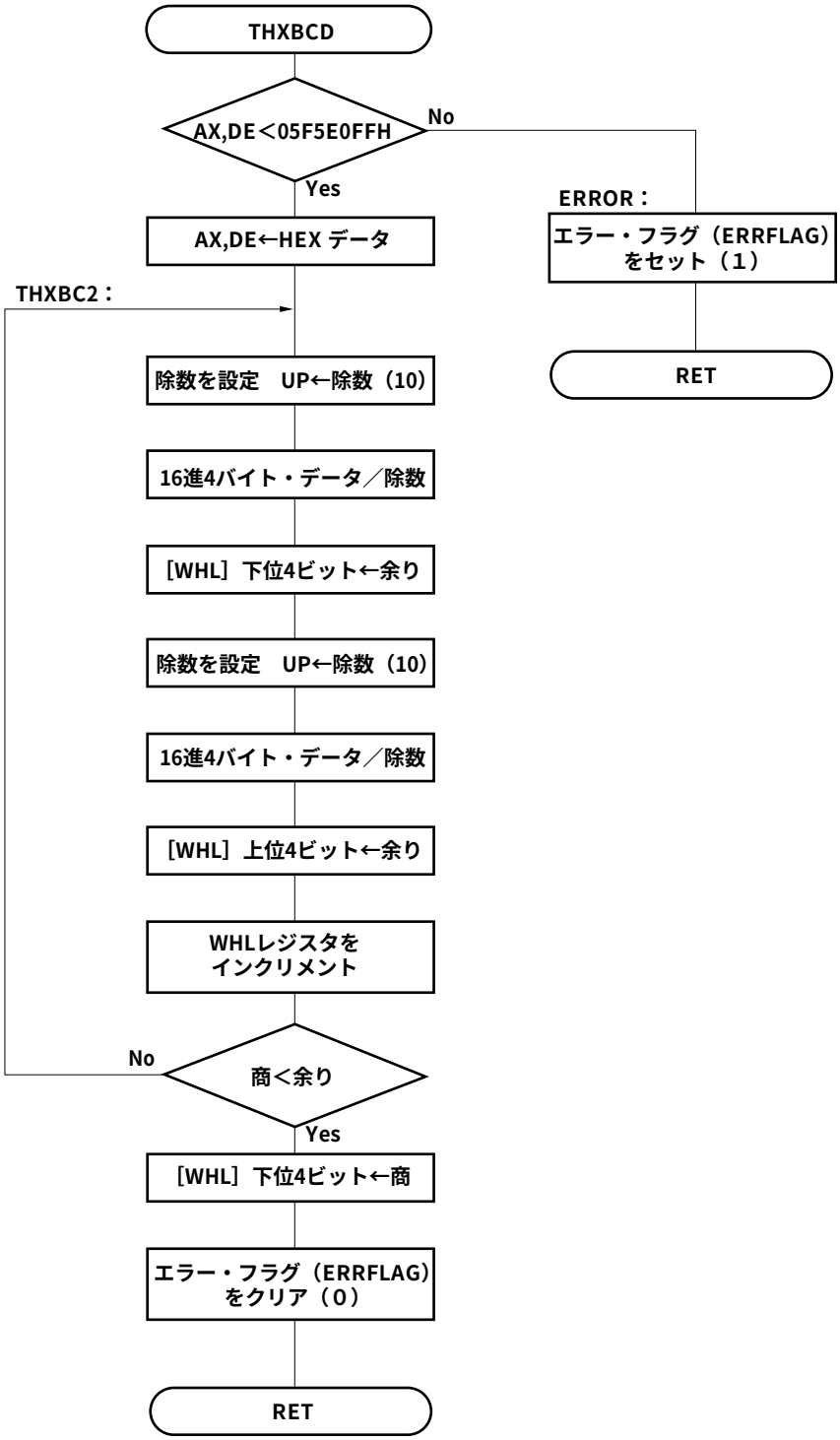
（ｉ）“16進２バイト入力データ／除数”を行う（UPレジスタには余りが入る）。

（ｊ）余りをWHLレジスタで示される10進８桁データ格納領域の上位４ビットに格納し，WHLレジスタをインクリメントする。

（ｋ）AX, DEレジスタに保持されている商を除数と比較して，商の方が大きければ（ｄ）の処理を行い，小さければWHLレジスタで示される10進８桁データ格納領域の下位４ビットに商を格納する。

（ｌ）エラーなく変換を終了したことを示すために，エラー・フラグ（ERRFLAG）をクリア（０）し，変換処理を終了する。

（7）フロー・チャート


THXBCD
AX,DE＜05F5E0FFH
No
Yes
ERROR：
エラー・フラグ（ERRFLAG）
をセット（1）
AX,DE←HEX データ
RET
THXBC2：
除数を設定　UP←除数（10）
16進4バイト・データ／除数
［WHL］下位4ビット←余り
除数を設定　UP←除数（10）
16進4バイト・データ／除数
［WHL］上位4ビット←余り
WHLレジスタを
インクリメント
No
商＜余り
Yes
［WHL］下位4ビット←商
エラー・フラグ（ERRFLAG）
をクリア（0）
RET

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベル／フラグの説明

HEXDAT　：変換する16進４バイトの値を格納してあるRAM領域の最下位アドレス

ERRFLAG　：エラー・フラグ

ERRFLAG＝０…エラー未発生状態

ERRFLAG＝１…エラー発生状態

●メイン・ルーチンのプログラム・リスト記述例

```
        ・
        ・
        MOVG    WHL,#HEXDAT
        CALL    !THXBCD
        BT      ERRFLAG,$ERROR
        BR      $$
ERROR:
        CLR1    ERRFLAG                 ; clear error flag
        ・
        ・
```

備考　上記記述例のようにWHLレジスタを設定し，サブルーチンを呼んでください。また，エラー処理は必要に応じて追加作成してください。

●本アプリケーション・ルーチンのプログラム・リスト

```
                NAME    TRBCDR
;************************************************
;*      transform BCD <- HEX                    *
;*         input condition                      *
;*              WHL-register <- HEX-4byte data  *
;*                                   LSB address*
;*         output condition                     *
;*              normal ... cy = 0               *
;*                  decimal 8-digit -> (WHL-WHL+3)*
;*              overflow ... cy = 1             *
;*                  HEX data > 99999999         *
;************************************************
        PUBLIC  THXBCD
        EXTBIT  ERRFLAG         ;
;
        CSEG
THXBCD:
        MOVW    AX,[WHL+0]      ;
        MOVW    DE,AX           ;
        MOVW    AX,[WHL+2]      ;
        CMPW    AX,#5F5H        ;
        BC      $THXBC2         ;
        BNZ     $ERHXBCD        ;
        MOVW    DE,AX           ;
        MOVW    AX,[WHL+0]      ;
        CMPW    AX,#0E0FFH      ;
        BC      $THXBC1         ;
        BNZ     $ERHXBCD        ;
THXBC1:
        XCHW    AX,DE           ; AXDE <- hex data set
THXBC2:
        MOVW    UP,#10          ; set divisor
        DIVUX   UP              ; AXDE / UP
        PUSH    AX              ; save AX-register
        MOVW    AX,UP           ;
        MOVW    [WHL],AX        ;
        POP     AX              ;load AX-register
        CMPW    AX,#00          ;
        BNZ     $THXBC3         ;
        CMPW    DE,#10          ;
        BC      $HENEND         ;
```

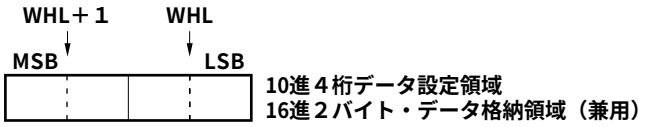
```
;
THXBC3:
        MOVW    UP,#10          ; set divisor
        DIVUX   UP              ; AXDE / UP
        PUSH    AX              ; save AX-register
        MOVW    AX,UP           ;
        ROL     X,4             ;
        XCH     A,X             ;
        OR      [WHL+],A        ;
        POP     AX              ;load AX-register
        CMPW    AX,#00          ;
        BNZ     $THXBC2         ;
        CMPW    DE,#10          ;
        BNC     $THXBC2         ;
        XCH     A,E             ;
        MOV     [WHL],A         ;
        BR      HENEND1         ;
HENEND:
        ROL     E,4             ;
        XCH     A,E             ;
        OR      [WHL],A         ;
HENEND1:
        CLR1    ERRFLAG         ; no error
        RET
;
ERHXBCD:                        ;
        SET1    ERRFLAG         ; set error
        RET
        END
```

６.２　10進（BCD）を16進（HEX）に変換

（１）処理概要

10進４桁のデータを16進２バイトのデータに変換するプログラム例を紹介します。

（２）使用RAM領域



（３）使用レジスタ

A, X, B, C, DE, WHLレジスタ

（４）入力方法

WHLレジスタを次のように設定します。

WHL：変換する10進４桁の値を格納してあるRAM領域の最下位アドレス

（５）出力方法

次に示すフラグに，変換処理状況が設定されます。

ERRFLAG：エラー・フラグ

ERRFLAG＝０…エラー未発生状態（データ変換正常終了）

ERRFLAG＝１…エラー発生状態（変換するデータが10進でないため変換できない）

WHLレジスタで示す２バイトの領域に次の内容が格納されます。

WHL～WHL＋１：変換された16進４桁の値が格納されます。[注]

注　エラー・フラグ（ERRFLAG）＝１のときWHLレジスタで示す２バイトの値は不定となります。

備考１．変換範囲は，０～9999です。

２．メイン・ルーチンでエラー・フラグ（ERRFLAG）の判別を行っています。必要に応じて演算エラー処理を追加作成してください。

（６）プログラム説明

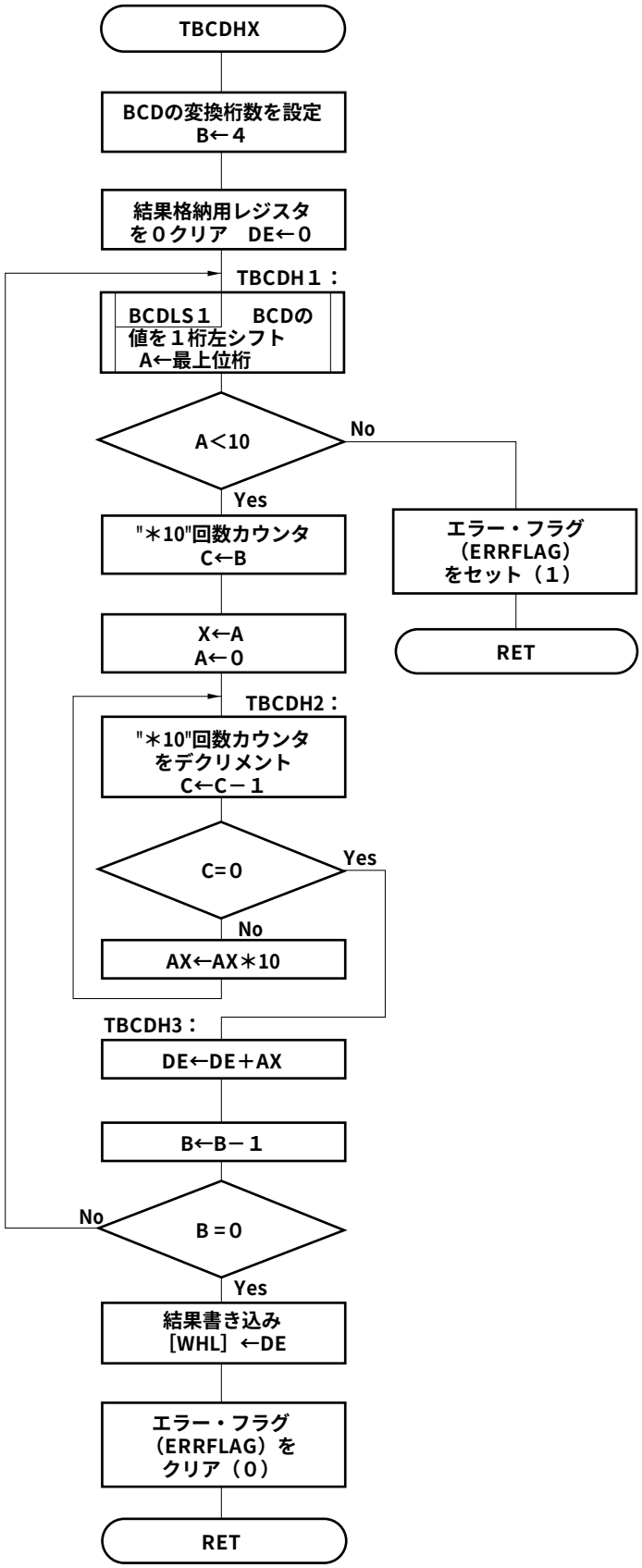
本変換プログラムでは，10進４桁のデータを最上位桁から１桁（４ビット）左シフトにより，１桁ずつＡレジスタに転送し，次の処理を４回繰り返して変換しています。

（格納エリア）←（格納エリア）×10＋Ａレジスタ

次に処理手順を示します。

（ａ）変換する10進４桁データのアドレス・ポインタとしてＢレジスタを使用し，変換桁数（４）を設定する。
（ｂ）変換結果を格納する結果格納用レジスタとしてDEレジスタを使用し，結果確認用レジスタの値をクリア（０）する。
（ｃ）10進４桁データ設定領域の値を１桁（４ビット）左シフトし，10進４桁データ設定領域の最上位桁の値をＡレジスタに読み込む。
（ｄ）Ａレジスタに読み込んだ10進４桁データ設定領域の値が，10進データ（０～９）かをチェックする。10進でなければ，変換不可能として，エラー・フラグ（ERRFLAG）をセット（１）する。
（ｅ）“結果格納用レジスタ（DEレジスタ）←結果格納用レジスタ（DEレジスタ）×10＋Ａレジスタ”を行う。
（ｆ）10進４桁データのアドレス・ポインタ（Ｂレジスタ）をデクリメントし，０でなければ（ｃ）から（ｅ）を繰り返す。
（ｇ）結果格納用レジスタ（DEレジスタ）を，WHLレジスタで示される16進２バイト・データ格納領域に格納する。
（ｈ）エラーなく変換を終了したことを示すために，エラー・フラグ（ERRFLAG）をクリア（０）し，変換処理を終了する。

（7）フロー・チャート


TBCDHX
BCDの変換桁数を設定
B←４
結果格納用レジスタ
を０クリア　DE←０
TBCDH１：
BCDLS１　BCDの
値を１桁左シフト
A←最上位桁
A＜10
No
Yes
"＊10"回数カウンタ
C←B
エラー・フラグ
（ERRFLAG）
をセット（１）
RET
X←A
A←０
TBCDH2：
"＊10"回数カウンタ
をデクリメント
C←C－１
C=０
Yes
No
AX←AX＊10
TBCDH3：
DE←DE＋AX
B←B－１
No
B＝０
Yes
結果書き込み
［WHL］←DE
エラー・フラグ
（ERRFLAG）を
クリア（０）
RET

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベル／フラグの説明

BCDDAT　：変換する10進４桁を格納してあるRAM領域の最下位アドレス

ERRFLAG　：エラー・フラグ

ERRFLAG＝０…エラー未発生状態

ERRFLAG＝１…エラー発生状態

●メイン・ルーチンのプログラム・リスト記述例

```
        ・
        ・
        MOVG    WHL,#BCDDAT
        CALL    !TBCDHX
        BT      ERRFLAG,$ERROR
        BR      $$
ERROR:
        CLR1    ERRFLAG                 ; clear error flag
        ・
        ・
```

備考　上記記述例のようにWHLレジスタを設定し，サブルーチンを呼んでください。また，エラー処理は必要に応じて追加作成してください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    TRHEXR

;*******************************************************
;*      transform HEX <- BCD                           *
;*          input condition                            *
;*              WHL-register <- decimal 4 digit data   *
;*                                  LSD address        *
;*          output condition                           *
;*              normal ... cy = 0                      *
;*                     HEX 2 byte -> (WHL,WHL+1)       *
;*              error  ... cy = 1                      *
;*******************************************************
;
        PUBLIC  TBCDHX
        EXTRN   BCDLS1
        EXTBIT  ERRFLAG         ;
;
        CSEG
TBCDHX:
        MOV     B,#4            ; BCD length
        MOVW    DE,#0           ; result work
TBCDH1:
        PUSH    WHL             ; save pointer
        MOV     C,#2            ; shift counter
        MOV     A,#0
        CALL    !BCDLS1         ; BCD left shift

        POP     WHL             ; restore pointer
        CMP     A,#10           ; error check
        NOT1    CY
        BC      $ERBCDHX        ; error return

        MOV     C,B
        MOV     X,#0
        XCH     A,X
TBCDH2:
        DEC     C
        BZ      $TBCDH3

        PUSH    BC              ; AX <- AX * 10
        MOVW    BC,AX
        SHLW    AX,2
        ADDW    AX,BC
        SHLW    AX,1
        POP     BC
        BR      $TBCDH2
```

```
TBCDH3:
        ADDW    AX,DE               ; result addition
        MOVW    DE,AX
        DBNZ    B,$TBCDH1           ; check length

        MOVW    AX,DE               ; write result to memory
        XCH     A,X
        MOV     [WHL+],A
        MOV     A,X
        MOV     [WHL],A
TBCDH4:
        CLR1    ERRFLAG             ; no error
        RET
ERBCDHX:
        SET1    ERRFLAG             ; set error
        RET
```

６.３　ASCIIコードを16進コードに変換

（１）処理概要

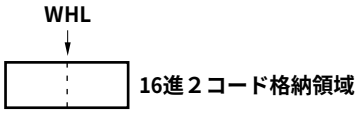
ASCIIコード（30H～39H, 41H～46H）の２コード（２バイト）を16進コード（00H～0FH）の２コード（１バイト）に変換するプログラム例を紹介します。

ASCIIコードと16進コードは，次のように対応しています。

ASCIIコード	30H	31H	32H	33H	34H	35H	36H	37H	38H	39H
16進コード	0H	1H	2H	3H	4H	5H	6H	7H	8H	9H

ASCIIコード	41H	42H	43H	44H	45H	46H
16進コード	AH	BH	CH	DH	EH	FH

（２）使用RAM領域



（３）使用レジスタ

Ａ，Ｂ，Ｃ，WHLレジスタ

（４）入力方法

BC, WHLレジスタを次のように設定します。

BC　：変換するASCIIコードの２コードを設定します。

WHL ：変換した16進コードの２コードを格納するRAM領域のアドレスを設定します。

（５）出力方法

次に示すフラグに，変換処理状況が設定されます。

ERRFLAG：エラー・フラグ

ERRFLAG＝0…エラー未発生状態（データ変換正常終了）

ERRFLAG＝1…エラー発生状態（変換するデータがASCIIコードでないため変換できない）

WHLレジスタで示す1バイトのRAM領域に，次の内容が格納されます。

WHL：変換された16進コードの２コードが格納されます。[注]

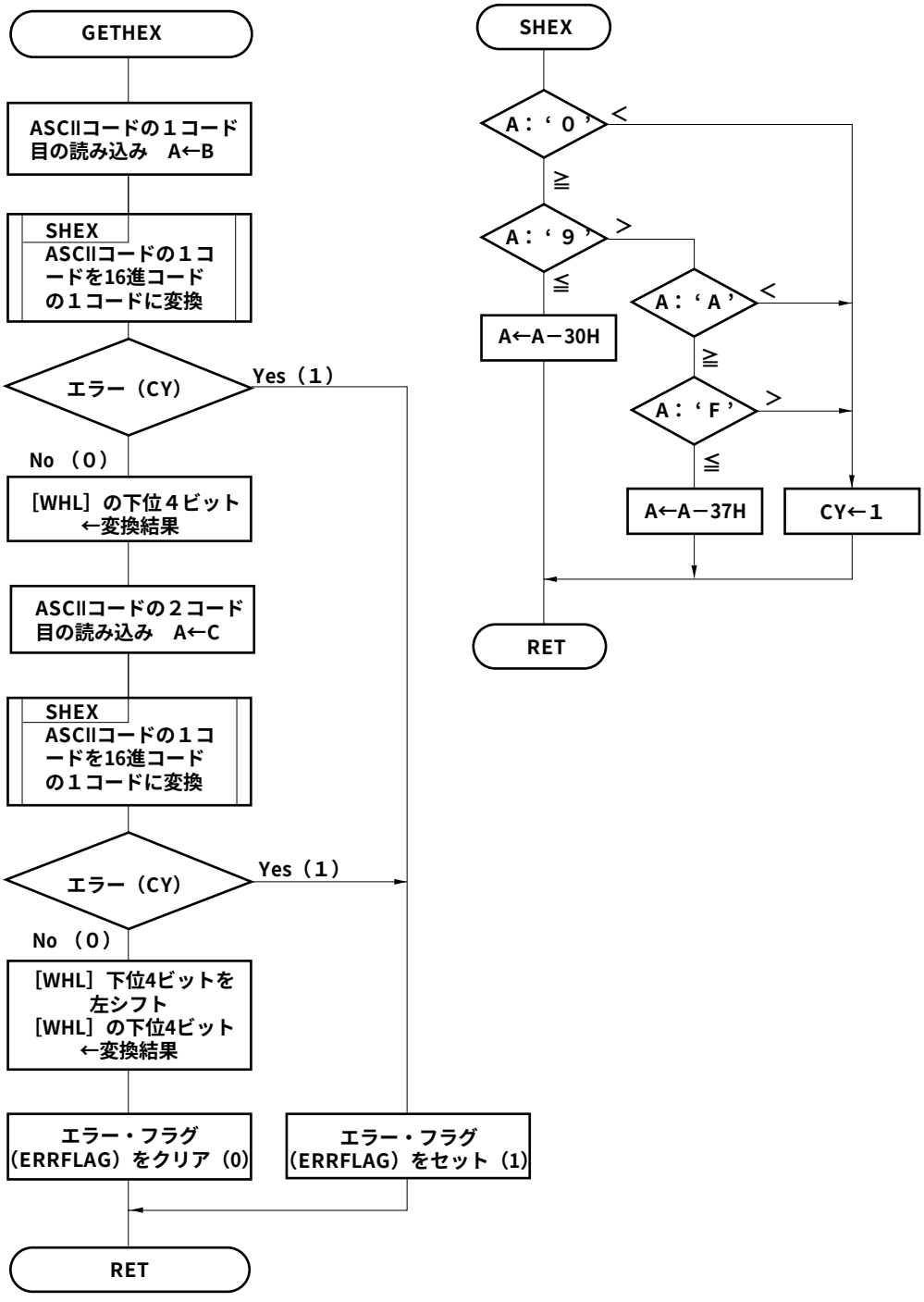
注　エラー・フラグ（ERRFLAG）=１のとき，WHLレジスタで示す１バイトの値は不定となります。

備考　メイン・ルーチンでエラー・フラグ（ERRFLAG）の判別を行っています。必要に応じて演算エラー処理を追加作成してください。

（6）プログラム説明

（ａ）ASCIIコードの１コード目（Ｂレジスタ）をＡレジスタに読み込む。
（ｂ）Ａレジスタの内容が30H～39H, 41H～46Hの範囲にあるかチェックする。なければ変換不可能として，エラー・フラグ（ERRFLAG）をセット（１）し，変換処理を終了する。
（ｃ）Ａレジスタの内容が30H～39Hならば30Hを引く。
　　Ａレジスタの内容が41H～46Hならば37Hを引く。
（ｄ）WHLレジスタで示される16進２コード格納領域の内容を４ビット左シフトし，下位の４ビットにＡレジスタの内容を格納する。
（ｅ）ASCIIコードの２コード目（Ｃレジスタ）をＡレジスタに読み込み，（ｂ）から（ｄ）の処理を行う。

（7）フロー・チャート


GETHEX
ASCIIコードの１コード目の読み込み　A←B
SHEX
ASCIIコードの１コードを16進コードの１コードに変換
エラー（CY）
Yes（1）
No （0）
［WHL］の下位４ビット←変換結果
ASCIIコードの２コード目の読み込み　A←C
SHEX
ASCIIコードの１コードを16進コードの１コードに変換
エラー（CY）
Yes（1）
No （0）
［WHL］下位4ビットを左シフト
［WHL］の下位4ビット←変換結果
エラー・フラグ（ERRFLAG）をクリア（0）
エラー・フラグ（ERRFLAG）をセット（1）
RET
SHEX
A：‘ 0 ’
<
≧
A：‘ 9 ’
>
≦
A：‘ A ’
<
A←A－30H
≧
A：‘ F ’
>
≦
A←A－37H
CY←1
RET

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベル／フラグの説明

ASCDAT　：変換するASCIIコードの２コード（２バイト）を格納してあるRAM領域の最上位アドレス

HEXDAT　：変換された16進コードの２コード（1バイト）を格納するRAM領域のアドレス

ERRFLAG　：エラー・フラグ

ERRFLAG＝0…エラー未発生状態

ERRFLAG＝1…エラー発生状態

●メイン・ルーチンのプログラム・リスト記述例

```
        •
        •
        MOVW    BC,ASCDAT
        MOVG    WHL,#HEXDAT
        CALL    !GETHEX
        BT      ERRFLAG,$ERROR
        BR      $$
ERROR:
        CLR1    ERRFLAG                 ; clear error flag
        •
        •
```

備考　上記記述例のようにWHLレジスタを設定し，サブルーチンを呼んでください。また，エラー処理は必要に応じて追加作成してください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    GHEXR
;************************************************
;*      transform   HEX <-  ASCII                *
;*                  (2code)  (2code)             *
;*                input  condition               *
;*                  BC-register <- ASCII         *
;*                output condition               *
;*                  (WHL)  <-  hex               *
;************************************************
        PUBLIC  GETHEX
        PUBLIC  SHEX
        EXTBIT  ERRFLAG         ;
        CSEG
GETHEX:
        MOV     A,B             ; ASCII upper-code load
        CALL    !SHEX           ; get  hex 1th code
        BC      $ERGTHEX
        ROL4    [WHL]
        MOV     A,C             ; ASCII lower-code load
        CALL    !SHEX           ; get  hex 2th code
        BC      $ERGTHEX
        ROL4    [WHL]
GTHEX1:
        CLR1    ERRFLAG         ; no error
        RET
ERGTHEX:
        SET1    ERRFLAG         ; set error
        RET
;************************************************
;*      subroutine  /  get hex 1-code(Acc)      *
;************************************************

SHEX:
        CMP     A,#'0'          ; check / ASCII < 30H
        BC      $ERSHEX

        CMP     A,#'9'+1        ; check / ASCII > 39H
        BNC     $SHEX1
        SUB     A,#30H
        BR      ENDSHEX
SHEX1:
        CMP     A,#'A'          ; check / ASCII < 41H
        BC      $ERSHEX

        CMP     A,#'F'+1        ; check / ASCII > 46H
        BNC     $ERSHEX
        SUB     A,#37H
        BR      ENDSHEX
ERSHEX:
        SET1    CY              ;set error(CY<-1)
ENDSHEX:
        RET
```

６.４　16進コードをASCIIコードに変換

（1）処理概要

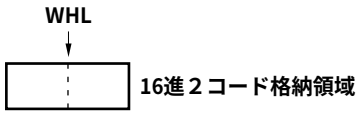
16進コード（00H～0FH）の２コード（１バイト）をASCIIコード（30H～39H, 41H～46H）の２コード（２バイト）に変換するプログラム例を紹介します。

16進コードとASCIIコードは，次のように対応しています。

16進コード	0H	1H	2H	3H	4H	5H	6H	7H	8H	9H
ASCIIコード	30H	31H	32H	33H	34H	35H	36H	37H	38H	39H

16進コード	AH	BH	CH	DH	EH	FH
ASCIIコード	41H	42H	43H	44H	45H	46H

（2）使用RAM領域



（3）使用レジスタ

Ａ，Ｂ，Ｃ，WHLレジスタ

（4）入力方法

WHLレジスタを次のように設定します。

WHL ：変換する16進コードの２コードを格納するRAM領域のアドレスを設定します。

（5）出力方法

BCレジスタに次の内容が格納されます。

BC：変換したASCIIコードの２コードが格納されます。

（６）処理手順

（ａ）WHLレジスタで示される16進２コード格納領域の１コード目（上位４ビット）をＡレジスタに読み込む。

（ｂ）Ａレジスタの内容が10以上かチェックする。10未満ならば（ｄ）の処理へ。

（ｃ）Ａレジスタの内容に７を加える。

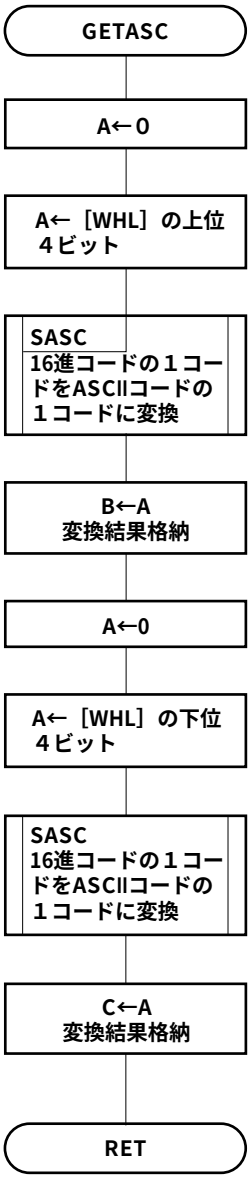
（ｄ）Ａレジスタの内容に30Hを加える。

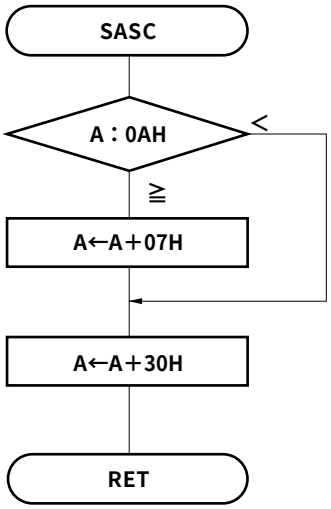
（ｅ）ＢレジスタにＡレジスタの内容を格納する。

（ｆ）WHLレジスタで示される16進２コード格納領域の２コード目（下位４ビット）をＡレジスタに読み込む。

（ｇ）（ｂ）から（ｃ）の処理を行い，ＣレジスタにＡレジスタの内容を格納する。

（7）フロー・チャート


GETASC
A←0
A←［WHL］の上位
４ビット
SASC
16進コードの１コー
ドをASCIIコードの
１コードに変換
B←A
変換結果格納
A←0
A←［WHL］の下位
４ビット
SASC
16進コードの１コー
ドをASCIIコードの
１コードに変換
C←A
変換結果格納
RET


SASC
A：0AH
<
≧
A←A＋07H
A←A＋30H
RET

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベル／フラグの説明

HEXDAT：変換する16進コードの２コード（１バイト）を格納してあるRAM領域のアドレス

ASCDAT：変換されたASCIIコードの２コード（２バイト）を格納するRAM領域の最上位アドレス

●メイン・ルーチンのプログラム・リスト記述例

```
          •
          •
        MOVG    WHL,#HEXDAT
        CALL    !GETASC
        MOVW    ASCDAT,BC
        BT      ERRFLAG,$ERROR
        BR      $$
ERROR:
        CLR1    ERRFLAG             ; clear error flag
          •
          •
```

備考　上記記述例のようにWHLレジスタを設定し，サブルーチンを呼んでください。また，エラー処理は必要に応じて追加作成してください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    ASCII
;*********************************************
;*      transform   ASCII   <-  HEX           *
;*                (2code)     (2code)         *
;*              input  condition              *
;*                (WHL) <- hex 2-code         *
;*              output condition              *
;*                BC-register <- ASCII 2-code *
;*********************************************
;
        PUBLIC  GETASC
        PUBLIC  SASC
;
        CSEG
GETASC:
        MOV     A,#0
        ROL4    [WHL]           ; hex upper code load
        CALL    !SASC
        MOV     B,A             ; store result

        MOV     A,#0
        ROL4    [WHL]           ; hex lower code load
        CALL    !SASC
        MOV     C,A             ; store result
        RET
;*******************************************************
;*      subroutine  /  get ASCII 1-code(BC-register)   *
;*******************************************************
SASC:
        CMP     A,#0AH          ; check / hex > 9
        BC      $SASC1
        ADD     A,#07H          ; bias (+7)
SASC1:
        ADD     A,#30H          ; bias (+30H)
        RET
```

第７章 データ処理

ここでは，データ処理の例としてデータの配列ならびに検索のプログラム例を示します。

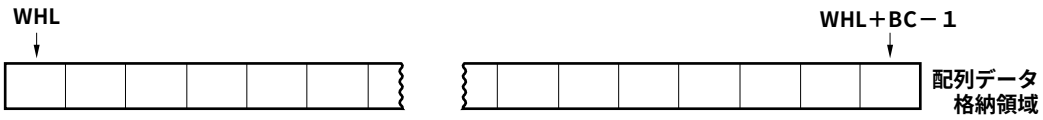
７.１　１バイト・データの配列

（１）処理概要

データ長が１バイトであるデータ・ファイルのデータを，昇順に配列するプログラム例を紹介します。配列の方法としては，バブル・ソートを用いています。

バブル・ソートとは，配列の始めから隣りどうしを比較して，逆順であったら交換します。配列の最後まで比較したら最初にもどり，交換がなくなるまで比較を繰り返し配列を行う方法です。

（２）使用RAM領域



（３）使用レジスタ

Ａ，BC, WHLレジスタ（ただし，WHLレジスタは保存されます）。

（４）入力方法

WHL, BCレジスタを次のように設定します。

WHL　：配列を行うデータを格納するRAM領域のアドレスを設定します。
BC　　：配列を行うデータの個数（バイト数）を設定します。

（５）出力方法

WHLレジスタで示されたRAM領域に，次の内容が格納されます。

WHL～WHL＋BC－1：昇順に配列されたデータが格納されます。

（６）プログラム説明

本配列プログラムでは，配列の方法としてバブル・ソートを用いています。

次に処理手順を示します。

（ａ）交換が行われたことを示す変換フラグ（CHGFLAG）をクリア（０）する。

（ｂ）配列データのバイト数を示すバイト・カウンタとしてBCレジスタを使用し，バイト・カウンタをデクリメントする。０ならば配列処理を終了する。

（ｃ）WHLレジスタ，BCレジスタを退避する。

（ｄ）WHLレジスタで示される配列データ格納領域の値（［WHL］と表記）と，WHLレジスタ＋１で示される配列データ格納領域の値（［WHL＋１］と表記）を比較する。

［WHL］＜［WHL＋１］の場合は（ｅ）の処理へ。

［WHL］≧［WHL＋１］の場合は（ｆ）の処理へ。

（ｅ）WHLレジスタで示される配列データ格納領域の内容と，WHLレジスタ＋１で示される配列データ格納領域の内容を交換し，変換フラグ（CHNGFL）をセット（１）する。

（ｆ）配列データ格納領域のアドレスを示すWHLレジスタをインクリメント，バイト・カウンタ（BCレジスタ）をデクリメントする。

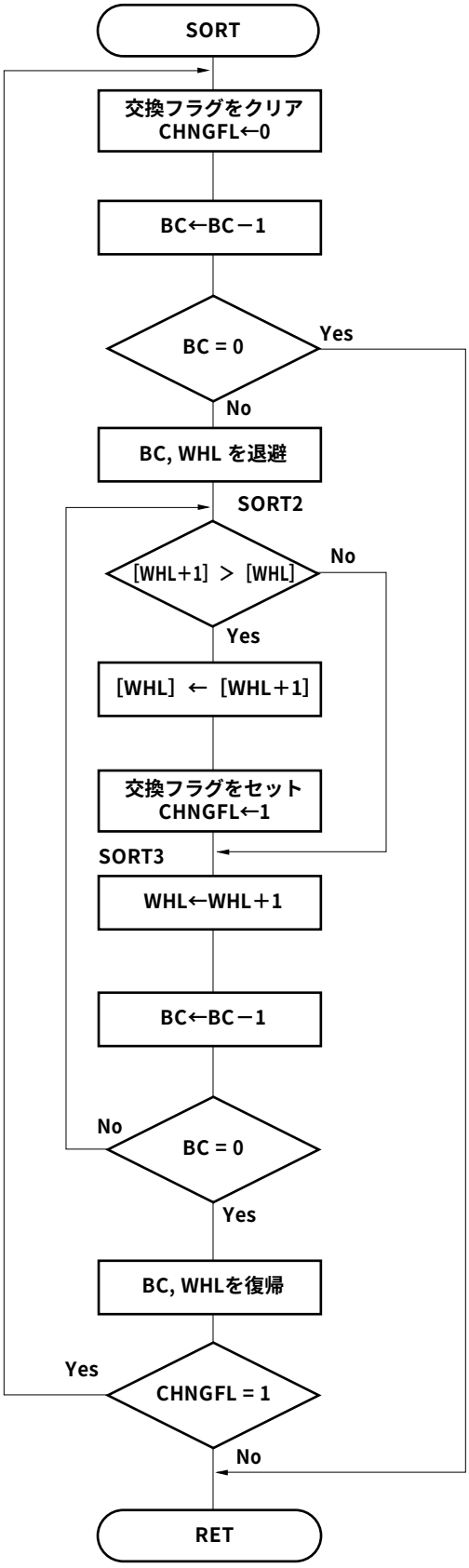
（ｇ）バイト・カウンタ（BCレジスタ）の値が０でなければ，（ｄ）から（ｆ）の処理を繰り返す。

（ｈ）（ｃ）で退避した配列を行うデータを，配列データ格納領域のアドレスを示すWHLレジスタ，配列のバイト数を示すBCレジスタの値を復帰する。

（ｉ）交換フラグ（CHGFLAG）がセット（１）されていれば（ａ）から（ｈ）の処理を繰り返す。

セット（１）されていなければ，配列処理を終了する。

（7）フロー・チャート


SORT
交換フラグをクリア
CHNGFL←0
BC←BC－1
BC = 0
Yes
No
BC, WHL を退避
SORT2
[WHL+1] > [WHL]
No
Yes
[WHL] ← [WHL+1]
交換フラグをセット
CHNGFL←1
SORT3
WHL←WHL＋1
BC←BC－1
No
BC = 0
Yes
BC, WHLを復帰
Yes
CHNGFL = 1
No
RET

（8）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

SORTDAT　：配列を行うデータ列の先頭アドレス

BCレジスタ：配列を行う個数

CHGFLAG　：データ交換が行われたか否かを示すフラグ

CHGFLAG＝１…データ交換済み

CHGFLAG＝０…データ未交換

●メイン・ルーチンのプログラム・リスト記述例

```
      •
      •
MOVW    BC,#10H          ;ﾃﾞｰﾀ長が16ﾊﾞｲﾄ
MOVG    WHL,#SORTDAT
CALL    !SORT
      •
      •
```

備考　上記記述例のようにBC, WHLレジスタを設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    SORTR

;*************************************************
;*      bubble  sort                             *
;*       input condition                         *
;*             BC-register  <- number of data    *
;*             WHL-register <- data top.address  *
;*       output condition                        *
;*             WHL-register <- data top.address  *
;*************************************************
;
        PUBLIC  SORT
;
        BSEG                    ;
CHGFLAG DBIT                    ; change-flag
;
        CSEG
SORT:
        CLR1    CHGFLAG         ; change-flag <- 0
        DECW    BC
        MOV     A,B
        OR      A,C
        BNZ     $SORT1
        BR      ENDSORT
SORT1:
        PUSH    BC              ; save pointer/counter
        PUSH    WHL
SORT2:
        MOV     A,[WHL]         ; change process
        CMP     A,[WHL+1]
        BC      $SORT3          ; A <= [WHL+1] goto $SORT3
        BZ      $SORT3
        XCH     A,[WHL+1]
        MOV     [WHL],A
        SET1    CHGFLAG         ; change-flag <- 1
SORT3:
        INCG    WHL             ; pointer increment
        DECW    BC
        MOV     A,B
        OR      A,C
        BNZ     $SORT2
        POP     WHL             ; restore pointer/counter
        POP     BC
        BT      CHGFLAG,$SORT
ENDSORT:
        RET
```

７.２　データの検索

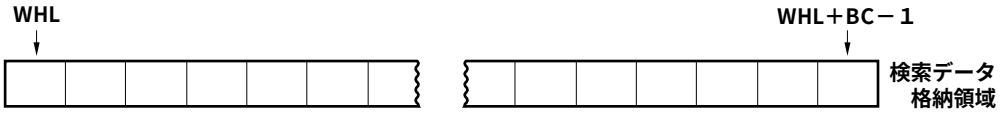
（１）処理概要

特定のデータを検索し，発見したらそのデータの格納アドレスを返すプログラム例を紹介します。検索の方法として２分探索（バイナリ・サーチ）を用いています。

２分探索とは，特定のデータ配置を定義するためにデータの集まりを検索する方法の１つです。

ここでは，昇順に配列されたデータの集まりを使用します。検索したいデータとデータの集まりの中間値を比較し，そのどちらが大きいかでデータの集まりの半分が削除されます。この操作を繰り返し行うことにより検索したいデータにたどりつくという方法で実現しています。

（２）使用RAM領域



（３）使用レジスタ

Ａ，BC, WHL, UUP, VVPレジスタ

（４）入力方法

Ａ，WHL, BCレジスタを次のように設定します。

Ａ　　：検索するデータを設定します。
WHL　：検索データ格納領域（昇順に配列済み）の先頭アドレスを設定します。
BC　　：検索データ数（バイト数）を設定します。

（５）出力方法

次に示すフラグに検索処理状況が設定されます。

CY：キャリー・フラグ
　　CY＝０…検索終了状態
　　CY＝１…検索データ無し状態

WHLレジスタに次の内容が格納されます。

WHLレジスタ：検索された検索データのアドレスが格納されます。[注]

注　検索データが発見されなかった場合は，WHLレジスタの値は不定になります。

（６）プログラム説明

本検索プログラムでは，検索の方法として２分探索（バイナリ・サーチ）を用いています。次に処理手順を示します。

（ａ）UUPレジスタに検索データ格納領域の先頭アドレス，VVPレジスタに検索データ格納領域の最終アドレスを設定する。

（ｂ）検索データ格納領域の先頭アドレス（UUPレジスタ）と検索データ格納領域の最終アドレス（VVPレジスタ）の値を比較する。

UUPレジスタ≦VVPレジスタ　の場合は（ｄ）の処理へ。

UUPレジスタ＞VVPレジスタ　の場合は（ｃ）の処理へ。

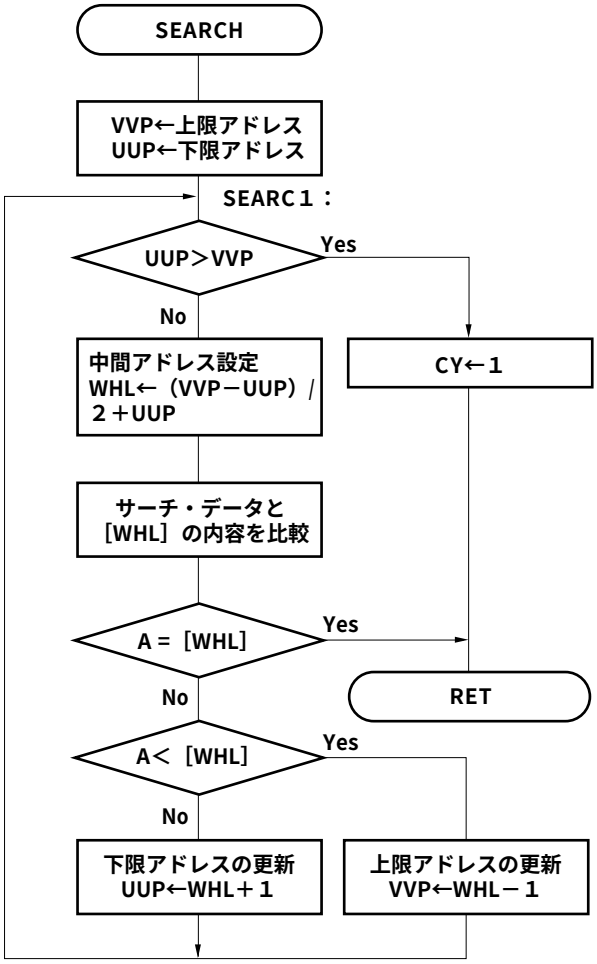
（ｃ）キャリー・フラグ（CY）をセット（１）して処理を終了する。

（ｄ）WHLレジスタに検索データ格納領域の中間アドレス（UUPレジスタとVVPレジスタで示されるアドレスの中間アドレス）を設定する。

（ｅ）検索データとWHLレジスタで示される検索データ格納領域の中間アドレス（UUPレジスタとVVPレジスタで示されるアドレスの中間アドレス）の内容を比較する。一致（発見）した場合は検索処理を終了する。

（ｆ）キャリー・フラグ（CY）=1ならば，“ VVP←WHL－１ ”を行い，最終アドレスをあらたに設定し，キャリー・フラグ（CY）=０ならば，“ UUP←WHL＋１ ”を行い，先頭アドレスをあらたに設定し，（ｃ）の処理へ戻る。

（7）フロー・チャート


SEARCH
VVP←上限アドレス
UUP←下限アドレス
SEARC１：
UUP＞VVP
Yes
No
中間アドレス設定
WHL←（VVP－UUP）/
２＋UUP
CY←１
サーチ・データと
［WHL］の内容を比較
A＝［WHL］
Yes
No
RET
A＜［WHL］
Yes
No
下限アドレスの更新
UUP←WHL＋１
上限アドレスの更新
VVP←WHL－１

（８）プログラム・リスト

●アプリケーション・ルーチン実行時に使用するレーベルの説明

SORTDAT：配列を行うデータ列の先頭アドレス

●メイン・ルーチンのプログラム・リスト記述例

```
      •
      •
MOVW    BC,#10H
MOVG    WHL,#SEACHDAT
MOV     A,#0AAH
CALL    !SEARCH
      •
      •
```

備考　上記記述例のようにBC, WHL, Aレジスタを設定し，サブルーチンを呼んでください。

●本アプリケーション・ルーチンのプログラム・リスト

```
        NAME    SEARCR
;*************************************************
;*      binary search                            *
;*       input condition                         *
;*             A-register   <- search data       *
;*             BC-register  <- number of data    *
;*             WHL-register <- data top.address  *
;*       output condition                        *
;*             WHL-register <- found data address  *
;*************************************************
        PUBLIC  SEARCH
        CSEG
SEARCH:
        MOVG    UUP,WHL         ; UUP-register <- lower.address
        DECW    BC
        MOVG    VVP,#0
        MOVW    VP,BC
        ADDG    VVP,UUP         ; VVP-register <- upper.address
SEARC1:
        MOVG    WHL,VVP
        SUBG    WHL,UUP
        BC      $SEARC4         ; search end check
        SHRW    HL,1
        ADDG    WHL,UUP
        CMP     A,[WHL]
        BNZ     $SEARC2
        BR      SEARC5          ; found data
SEARC2:
        BC      $SEARC3
        INCG    WHL             ; 'CY' = 0
        MOVG    UUP,WHL
        BR      $SEARC1
SEARC3:
        DECG    WHL             ; 'CY' = 1
        MOVG    VVP,WHL
        BR      $SEARC1
SEARC4:
        SET1    CY
SEARC5:
        RET
```

★

付　録　改版履歴

これまでの改版履歴を次に示します。なお，適用箇所は各版での章を示します。

版　数	前版からの改版内容	適用箇所
第２版	78K/IVシリーズ共通とするため，各サブシリーズ名および各製品名を追加	はじめに

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　：s-info@saed.tmg.nec.co.jp

【営業関係お問い合わせ先】

第一販売事業部
東　京　(03)3798−6106, 6107, 6108
名古屋　(052)222−2375
大　阪　(06)6945−3178, 3200, 3208, 3212
仙　台　(022)267−8740
郡　山　(024)923−5591
千　葉　(043)238−8116

第二販売事業部
東　京　(03)3798−6110, 6111, 6112
立　川　(042)526−5981, 6167
松　本　(0263)35−1662
静　岡　(054)254−4794
金　沢　(076)232−7303
松　山　(089)945−4149

第三販売事業部
東　京　(03)3798−6151, 6155, 6586, 1622, 1623, 6156
水　戸　(029)226−1702
広　島　(082)242−5504
高　崎　(027)326−1303
鳥　取　(0857)27−5313
太　田　(0276)46−4014
名古屋　(052)222−2170, 2190
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。

【インターネット電子デバイス・ニュース】

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］78K/IVシリーズ　アプリケーション・ノート　ソフトウエア基礎編
（U10095JJ2V1AN00（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