To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

APPLICATION NOTE

78K/IV SERIES

16-BIT SINGLE-CHIP MICROCOMPUTER

SOFTWARE BASICS

μPD784026 SUBSERIES
μPD784915 SUBSERIES

Document No. U10095EJ1V0AN00 (1st edition)
Date Published November 1995 P
Printed in Japan

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

PREFACE

Target Users

This application note is for engineers who wish to understand 78K/IV Series devices and design application programs using these devices.

78K/IV Series

- μPD784026 Subseries: μPD784020, 784021, 784025, 784026, 78P4026
- μPD784915 Subseries: μPD784915**Note**, 78P4916**Note**

Note Under development

Objective

The purpose of this application note is to use program examples to help users to understand the basic functions of 78K/IV Series devices.
The program and hardware structures published here are illustrative examples and are not designed for mass production.

Organization

This application note describes the basic numeric operation programs.

Application Area

The 78K/IV Series devices have a 1-MB program memory space and are capable of high-speed instruction execution and low-voltage operation, making them suitable for a wide range of applications, including the following:

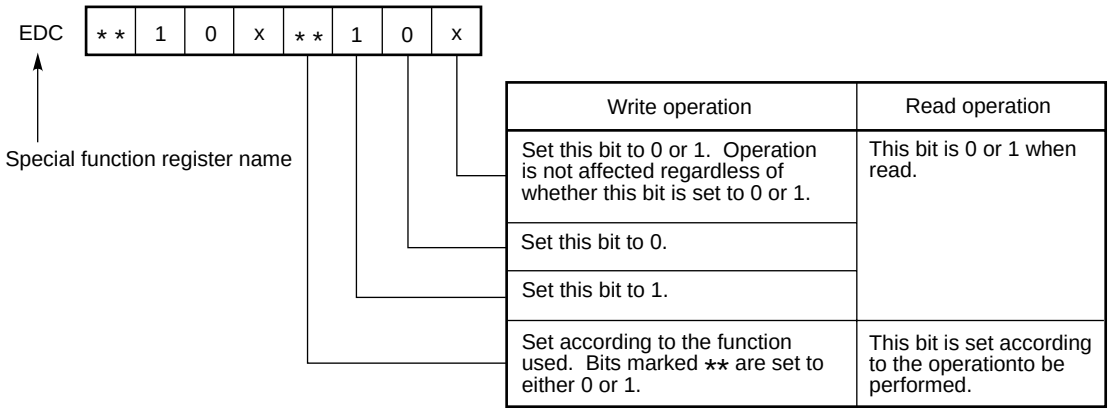
- Portable telephones
- CD-ROMs
- HDDs
- Cameras
- VCRs
- Printers
- Audio systems, etc.

Legend

The symbols and notations used in this manual have the following meanings:

Significance of the data description:		The left side is high-order data and the right side is low-order data.
Active-low description	:	$\overline{xxx}$ (line above pin and signal names)
Note	:	Explanation of the note attached to the text
Caution	:	Contents that should be read carefully
Remark	:	Supplemental explanation of the text
Number descriptions	:	Binary numbers ... xxxxxxxxB
	:	Decimal numbers ... xxxx
	:	Hexadecimal numbers ... xxxxH
Easily confused characters	:	0 (zero), O (oh)
	:	1 (one), l (lowercase letter), I (uppercase letter)

Special Function Register (SFR) Description



Write operation	Read operation
Set this bit to 0 or 1. Operation is not affected regardless of whether this bit is set to 0 or 1.	This bit is 0 or 1 when read.
Set this bit to 0.	
Set this bit to 1.	
Set according to the function used. Bits marked ∗∗ are set to either 0 or 1.	This bit is set according to the operationto be performed.

Do not attempt to enter a combination of codes indicated in "Setting Prohibited" in the register charts provided throughout this document.

Example of Special Function Register (SFR) Description

	7	6	5	4	3	2	1	0
STBC	0	0	CK1	CK0	x	0	STP	HLT
Example	0	0	0	1	0	0	0	0

STP	HLT	Operation mode
0	0	Normal operation mode
0	1	HALT mode
1	0	STOP mode
1	1	IDLE mode

CK1	CK0	Selection of internal system clock
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)

Remark Throughout this application note, those register bits that must be set are indicated by shading. When using a register, refer to the provided example as necessary.

Related Documents The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

• Documents common to 78K/IV series devices

Document name	Document number	
	Japanese	English
User's Manual, Instruction	IEU-844	IEU-1386
Application Note, Software Basics	This manual	—
Instruction Table	IEM-5580	—
Instruction Set	IEM-5572	—
78K Series Selection Guide	IF-258	IF-1141
Development Tool Selection Guide	EF-219	EF-1111

• Individual documents

μPD784026 subseries

Document name	Document number	
	Japanese	English
μPD784020, 784021 Data Sheet	IC-8776	IP-3234
μPD784025, 784026 Data Sheet	IC-8722	IP-3230
μPD78P4026 Data Sheet	IC-8734	IP-3231
μPD784026 Subseries Special Function Register Table	IEM-5579	—
μPD784026 Subseries User's Manual, Hardware	IEU-850	IEU-1379
μPD784026 Subseries Application Note, Hardware Basics	In preparation	—

μPD784915 subseries

Document name	Document number	
	Japanese	English
μPD784915 Product Information	IP-9130	—
μPD78P4916 Product Information	IP-9178	—
μPD784915 Subseries Special Function Register Table	IEM-5602	—
μPD784915 Subseries User's Manual, Hardware	IEU-850	—

Caution The related documents described above may be changed without notice. Always use the newest document when designing.

Phase-out/Discontinued

[MEMO]

CONTENTS

LIST OF FIGURES

CHAPTER 1 GENERAL

1.1 READING THIS DOCUMENT

This application note introduces examples of programs for basic binary and decimal arithmetic operations, such as addition, subtraction, multiplication, and division, as well as data exchange and data transfer as subroutine programs. The algorithms of these programs are described so that they can be used as user programs.

The program examples given in this document are presented in the following format:

(1) Outline of processing

Outlines the processing performed by the program.

(2) RAM area

Describes the RAM area used by the program.

Note that if a work area is used as the RAM area, its contents will be undefined after the program has been executed.

(3) Registers

Describes the registers used in the program.

To protect the contents of registers that have already been used by another program, save the contents of those registers by switching the register bank before executing the described program.

(4) Input

Describes the arguments that must be input when executing the program.

(5) Output

Describes the arguments that are output after the program has been executed.

(6) Program description

Describes the algorithm on which the program is based.

Also refer to the flowchart and program listing.

(7) Flowchart

Illustrates the algorithm of the program by means of a flowchart.

(8) Program listing

Presents a listing of the program.

All program listing are described as source programs. The address, therefore, differs depending on the link condition.

The programs shown in this application note are provided only as examples and their actual operation is not guaranteed.

1.2 USING APPLICATION PROGRAMS

Basically, use the application programs presented in this document as specified in the description of each application program. Some application programs call other application programs. In this case, link the programs by means of a linker (LK78K/IV) and by referring to the description. A work area may also be necessary. Reserve a RAM area according to the description and make a public declaration.

1.3 FEATURES OF 78K/IV SERIES DEVICES

As the functions of microcomputer-based products have improved and their cost fallen, the demand for microcomputers that can satisfy these mutually contradicting requirements — more sophisticated functions at lower cost — has steadily increased. In addition, with the explosive growth in portable systems such as cellular telephones, the demand for microcomputers that can operate at lower voltages while dissipating less power has also grown.

NEC's 78K/IV Series of 16-bit single-chip microcomputers was developed in response to these market demands.

The features of the 78K/IV series of microcomputers are as follows:

(1) Compatibility with existing models

The 78K/IV Series microcomputers maintain, at source level, upward compatibility with the 8-bit models (78K/0, 78K/I, and 78K/II Series) and 16-bit models (78K/III Series) of the 78K Series, thus protecting your investment in software.

(2) Wide linear memory space

Up to 1 MB of program memory and 16 MB of data memory are supported.

(3) Low voltage, low current dissipation

The operating voltage ranges from 2.7 to 5.5 V, so that 78K/IV Series devices can operate at very low voltages. In addition, three standby modes, STOP, IDLE, and HALT, and a function for dividing the clock to be supplied to the CPU are provided to enable power management according to the operating status.

(4) High-speed multiplication and division instructions

High-speed multiplication and division instructions are provided to support complicated control systems and high-accuracy control algorithms.

8 bits x 8 bits = 16 bits (unsigned)	–> 0.69 μs
16 bits x 16 bits = 32 bits (unsigned)	–> 0.94 μs
16 bits x 16 bits = 32 bits (signed)	–> 0.88 μs
16 bits/8 bits = 16-bit quotient, 8-bit remainder (unsigned)	–> 1.06 μs
32 bits/16 bits = 32-bit quotient, 16-bit remainder (unsigned)	–> 1.94 μs

(with internal 16-MHz clock)

(5) Powerful interrupt response

For control applications, the way in which a microcomputer responds to an interrupt is particularly important. The 78K/IV Series supports the following three interrupt functions:

- Vector interrupt
- Context switching
- Macro service

(6) Support of high-level language (C)

An enhanced instruction set for a C compiler, an efficient C compiler, and easy-to-use source debugging environments are supported.

(7) Easy-to-use tools

Easy-to-use tools, such as an assembler package, C compiler, in-circuit emulator, and integrated debugger are also supported.

1.4 PROGRAM

Some example programs presented in this application note require that the results of operations and numeric data be written into RAM. When referring to the programs in this application note, therefore, reserve the necessary amount of RAM by using the program below.

When using programs that perform binary division, decimal addition, decimal subtraction, and decimal division, add the processing to be performed in the case of an error, as necessary.

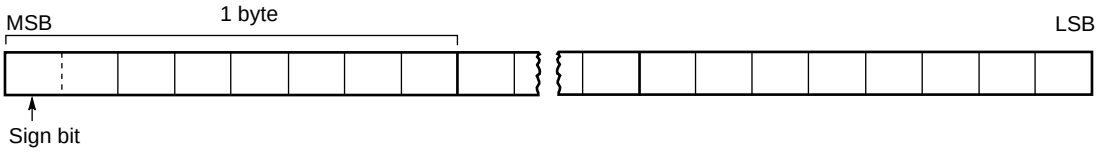
```
            PUBLIC  BMLCND, BMLIER, BUFRAM, BRSLT          ; binary multiply
            PUBLIC  DEND, DVISOR, DRMND                    ; binary division
            PUBLIC  DMLCND, DMLIER, DRSLT, CARRY           ; decimal multiply
            PUBLIC  DIVSOR, DIVIND, RMIND                  ; decimal 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
            .
            .                           ; Error processing.  Prepare as
                                        ; necessary.
```

CHAPTER 2 BINARY OPERATIONS

In binary operations, the most significant bit is used as a sign bit, the remaining bits expressing a numeric value. Negative numbers are expressed as a 2's complement.

Figure 2-1. Expressing Binary Numbers

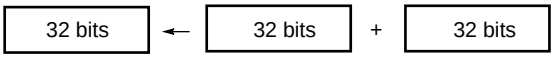


For binary operations, the data storage area used for the operation and the area storing the result of the operation are located in RAM.

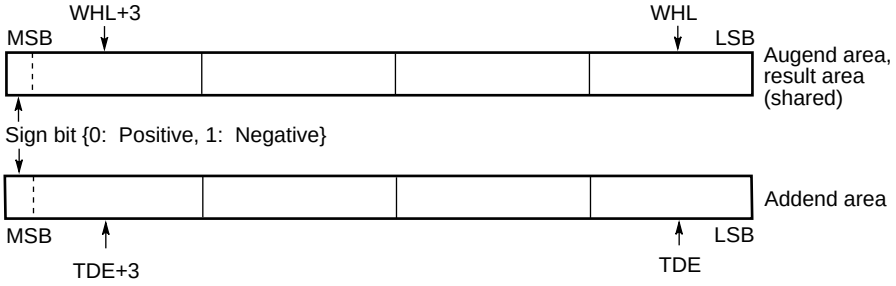
2.1 BINARY ADDITION OF SIGNED 32 BITS + 32 BITS

(1) Outline of Processing

This section presents an example program that adds a signed 32-bit augend to a 32-bit addend, then stores the result into a 32-bit result area (shared with the augend area).



(2) RAM area



When the sign bit is (0): Positive (00000000H through 7FFFFFFFH)
When the sign bit is (1): Negative (FFFFFFFFH through 80000000H)

(3) Registers

AX, C, VP, TDE, and WHL registers

(4) Input

Set the following addresses in the WHL and TDE registers.

WHL: Lowest address of the RAM area containing the 32-bit augend
TDE : Lowest address of the RAM area containing the 32-bit addend

(5) Output

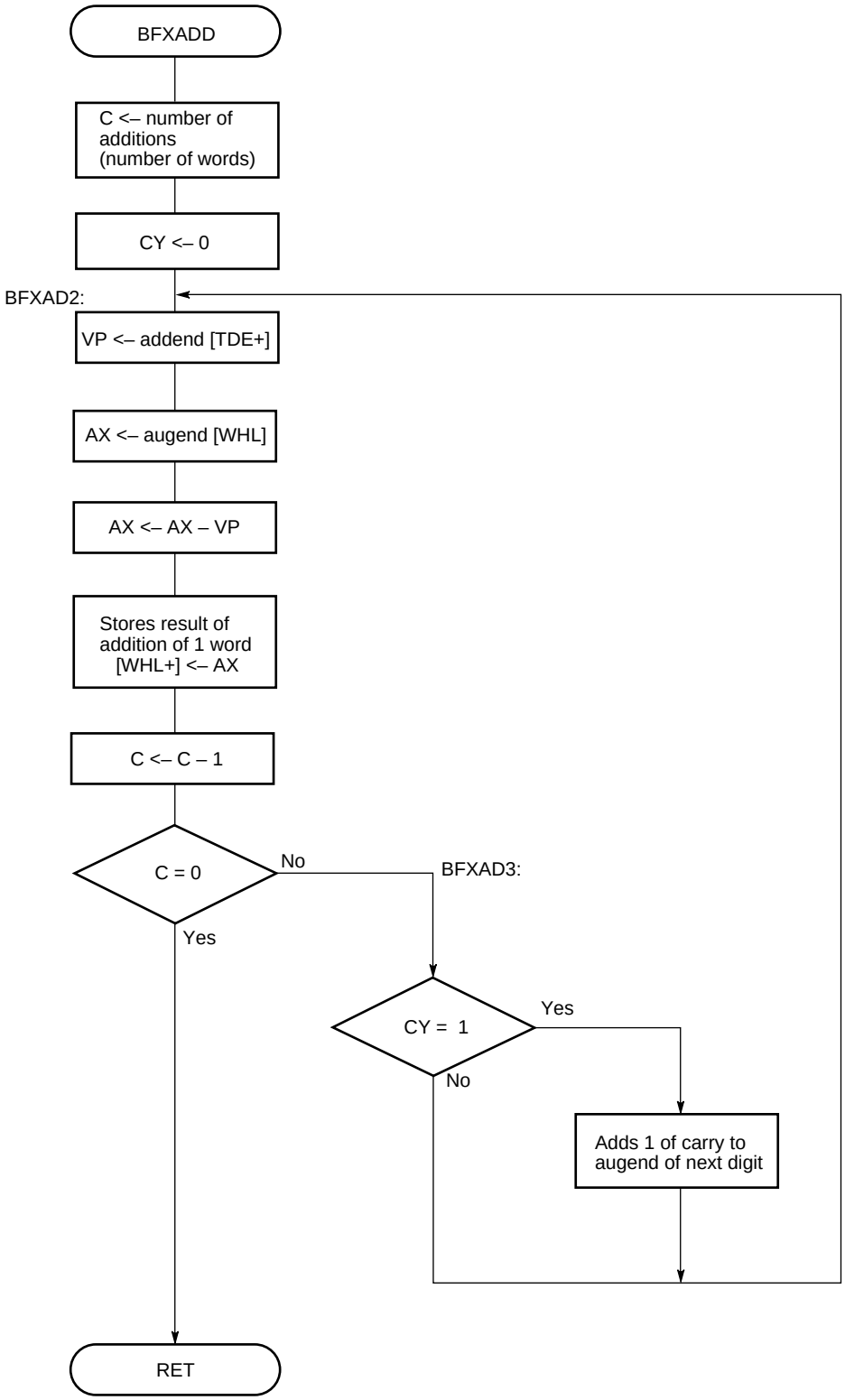
The following contents are stored into the 4-byte RAM area indicated by the WHL register.

WHL to WHL+3: Stores the result of addition.

(6) Program description

(a) Sets the number of words to be manipulated in a counter (C register).
(b) Clears the carry flag (CY) to 0 in advance.
(c) Reads the 2 bytes from the addend area indicated by the addend address (TDE register) into the AX register, and increments the added register (TDE register).
(d) Adds the 2 bytes in the augend area, indicated by the augend register (WHL register), into the AX register.
(e) Stores the value in the AX register into the RAM area indicated by the augend address (WHL register), and increments the augend address (WHL register).
(f) Decrements the counter (C register). When the value of the counter reaches 0, ends addition processing.
(g) If a carry has occurred as a result of the addition, reads the data of the high-order 2 bytes in the result area indicated by the next augend address (WHL register) into the AX register, and adds the 1 of the carry.
(h) Stores the value in the AX register into the result area indicated by the augend address (WHL register), then continues processing from step (c).

(7) Flowchart



(8) Program listing

- **Description of the label used for executing of application routine**

  AUGNE: Lowest address of the RAM area containing the 32-bit augend that stores the 32-bit result (shared)

  ADDEN: Lowest address of the RAM area containing the 32-bit addend

- **Example program listing for main routine**

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

CALL    !BFXADD         ; data ”A+B” subrutin
      .
      .
```

Remark Set the WHL and TDE registers as shown above, then call the subroutine.

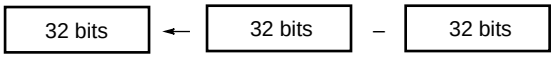
- **Program listing for this application routine**

```
        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                     ;
```

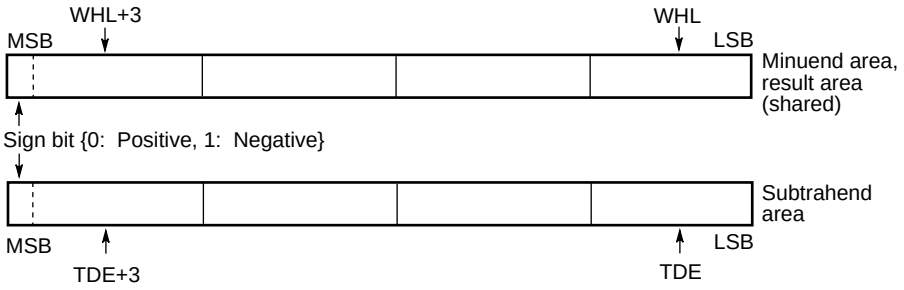
2.2 BINARY SUBTRACTION OF SIGNED 32 BITS – 32 BITS

(1) Outline of processing

This section presents an example program that subtracts a 32-bit subtrahend from a signed 32-bit minuend and stores the result into a 32-bit result area (shared with the minuend area).



(2) RAM area



When the sign bit is (0): Positive (00000000H through 7FFFFFFFH)
When the sign bit is (1): Negative (FFFFFFFFH through 80000000H)

(3) Registers

AX, C, VP, TDE, and WHL registers

(4) Input

Set the following addresses in the WHL and TDE registers.

WHL: Lowest address of the RAM area containing the 32-bit minuend
TDE : Lowest address of the RAM area containing the 32-bit subtrahend

(5) Output

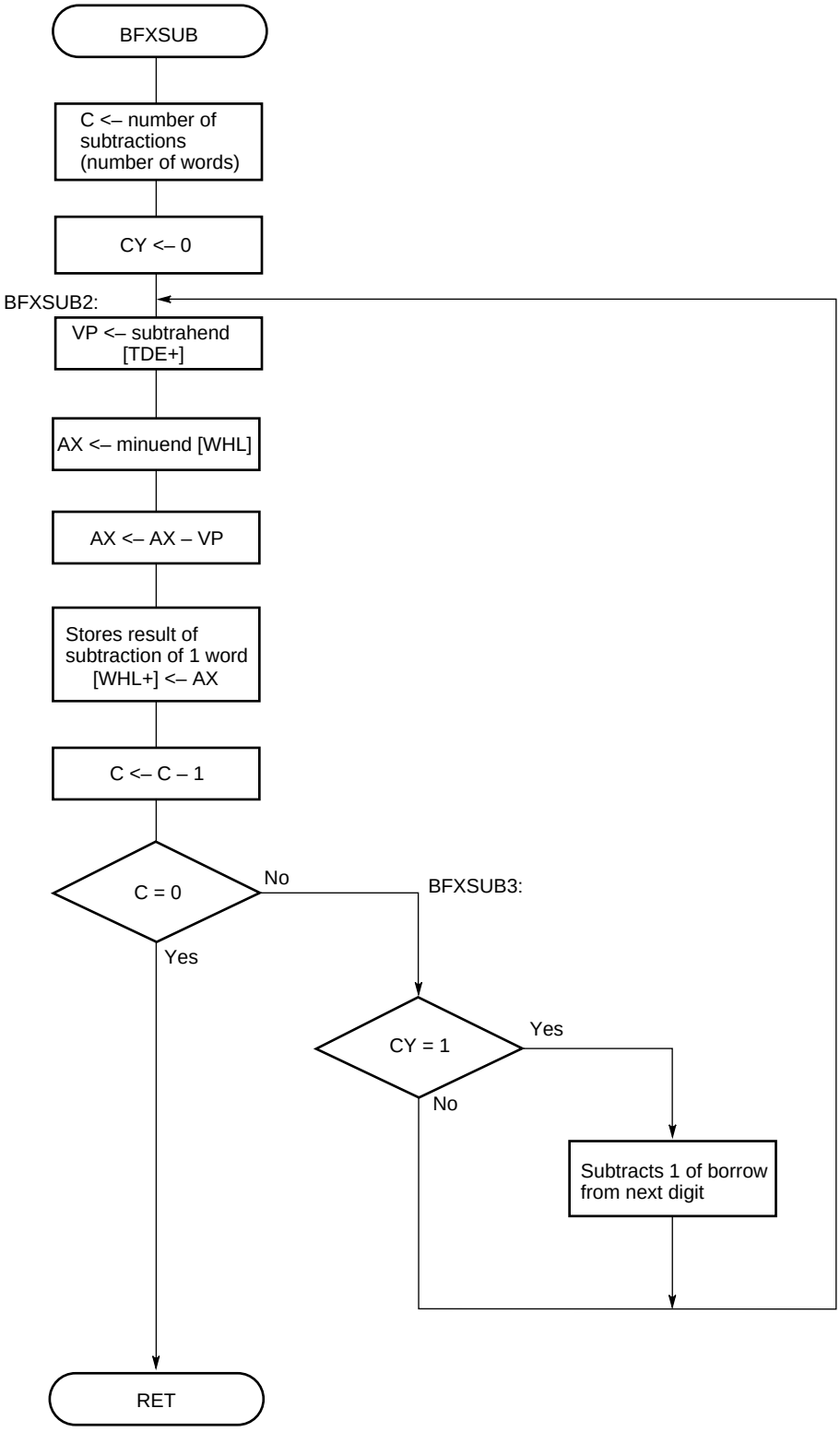
The following contents are stored into the 4-byte RAM area indicated by the WHL register.

WHL to WHL+3: Stores the result of the subtraction.

(6) Program description

(a) Sets the number of words to be manipulated in a counter (C register).
(b) Sets the data for subtraction in the subtrahend and minuend areas.
(c) Clears the carry flag (CY) to 0 in advance.
(d) Reads the 2 bytes from the subtrahend area indicated by the subtrahend address (TDE register) into the AX register, and increments the subtrahend address (TDE register).
(e) Subtracts the 2 bytes in the minuend area, indicated by the minuend address (WHL register), from the AX register.
(f) Stores the value of the AX register into the result area indicated by the minuend address (WHL register), and increments the minuend address (WHL register).
(g) Decrements the counter (C register). When the value of the counter reaches 0, ends subtraction processing.
(h) If a borrow has occurred as a result of the subtraction, reads the data of the high-order 2 bytes indicated by the next minuend address (WHL register) into the AX register, and subtracts the 1 of the borrow.
(i) Stores the value in the AX register into the result area indicated by the minuend address (WHL register), then continues processing from step (d).

(7) Flowchart



(8) Program listing

- **Description of the label used for executing of application routine**

  MINU: Lowest address of the RAM area containing the 32-bit minuend that stores the 32-bit result (shared)

  SUBT: Lowest address of the RAM area containing the 32-bit subtrahend

- **Example of program listing for main routine**

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

CALL     !BFXSUB         ;  data ”A–B” subrutin
      .
      .
```

Remark Set the WHL and TDE registers as shown above, then call the subroutine.

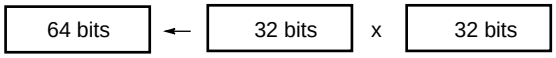
- **Program listing for this application routine**

```
        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 value 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                     ;
```

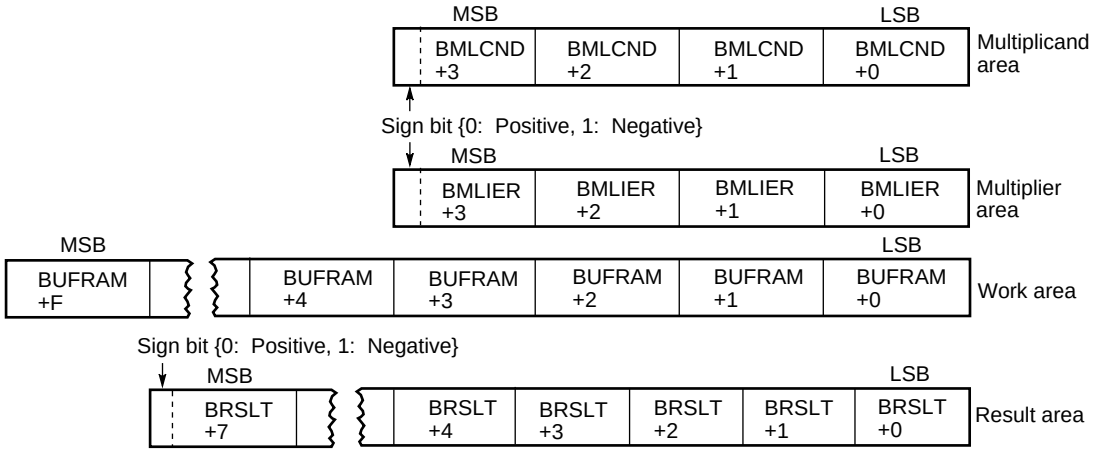
2.3 BINARY MULTIPLICATION OF SIGNED 32 BITS x 32 BITS

(1) Outline of processing

This section presents an example program that multiplies a signed 32-bit multiplicand by a 32-bit multiplier, and stores the result into a 64-bit result area.



(2) RAM area



When the sign bit is (0): Positive (00000000H through 7FFFFFFFH)
When the sign bit is (1): Negative (FFFFFFFFH through 80000000H)

(3) Registers

A, X, B, C, VP, TDE, WHL, RP2, and R4 registers

(4) Input

Set the data necessary for the operation into the following 4-byte RAM areas.

BMLCND to BMLCND+3: 32-bit multiplicand data
BMLIER to BMLIER+3 : 32-bit multiplier data

(5) Output

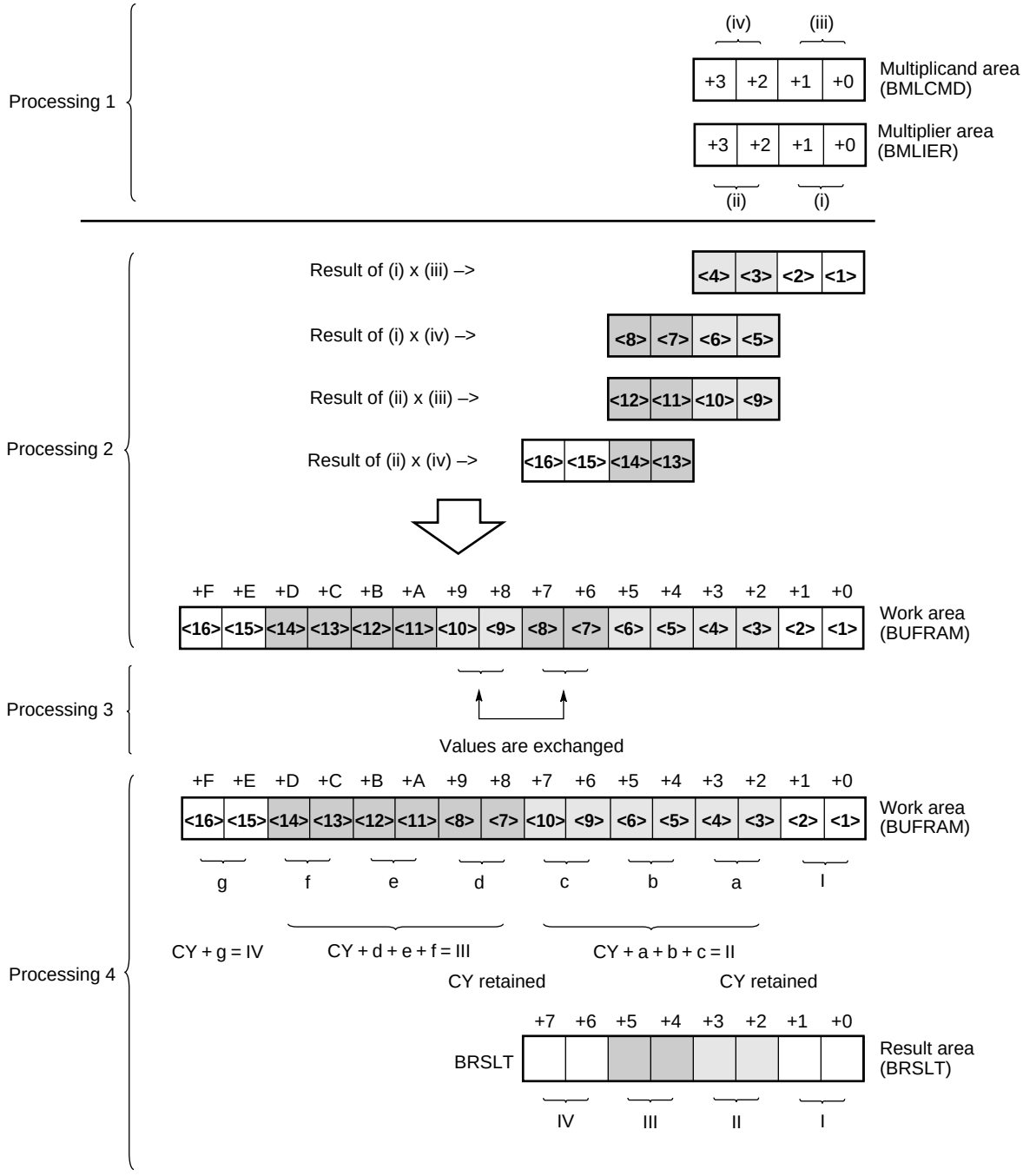
The following contents are stored into the following 8-byte RAM area.

BRSLT to BRSLT+7: Result of multiplication

(6) Program description

Figure 2-2 illustrates the algorithm for binary multiplication using the 16-bit multiplication instruction.

Figure 2-2. Algorithm for Binary Multiplication



Remark CY: Carry flag

Processing 4 adds the carry flag (CY) that has been generated as a result of the operation on the lower digit.

The processing performed by this program is as follows:

(a) Takes the absolute values of the multiplier and multiplicand. If the signs of the multiplier and multiplicand differ, sets the sign identification flag (SFLAG) to 1. If the signs are the same, clears the flag to 0.

(b) Sets the multiplication end flag (LFLAG) to 1.
(LFLAG = 1: Multiplication does not end, LFLAG = 0: Multiplication ends)

(c) Sets 0 to indicate the lowest address by using the B register as the address pointer to the multiplicand area.

Processing 1

(d) Sets the lowest address of the work area in the TDE register and the lowest address of the multiplier area in the WHL register.

(e) Reads the 2-byte multiplier indicated by the WHL register into the VP register, and saves its contents in preparation for the operation performed on the next digit.

Processing 2

(f) Reads the 2-byte multiplicand, indicated by the address pointer (B register) in the multiplicand area, into the AX register, multiplies it by the multiplier stored in the VP register, then stores the result into the work area indicated by the TDE register, 4 bytes at a time.

(g) Adds 2 to the B register, and increments the address pointer of the multiplicand area by 2 bytes.

(h) Compares the value of the address pointer (B register) in the multiplicand area with the number of digits (4) in the multiplicand area to determine whether the two values are the same.
If the values are found to be different, restores the value of the multiplier saved to the VP register in step (e) to multiply the multiplicand of the next digit, and returns to step (g). Steps (g) through (i) are repeated until the values are the same.

(i) Sets 0, to indicate the lowest address, into the address pointer (B register) of the multiplicand area.

(j) Judges whether all multiplications have been completed, by referencing the multiplication end flag (LFLAG). If multiplication has not yet been completed, clears the multiplication end flag (LFLAG) to 0, returns to step (e), then repeats the processing up to step (j).

Processing 3

(k) Exchanges the contents of work areas BUFRAM+6 and BUFRAM+7, and BUFRAM+8 and BUFRAM+9, in which the multiplication result is stored.

Processing 4

(l) Stores the values of BUFRAM+0 and BUFRAM+1 into the first and second bytes of the result area (BRSLT).

(m) Adds the values of BUFRAM+2 through BUFRAM+7 with the carry flag (CY), 2 bytes at a time, then stores the result into the third and fourth bytes of the result area (BRSLT).

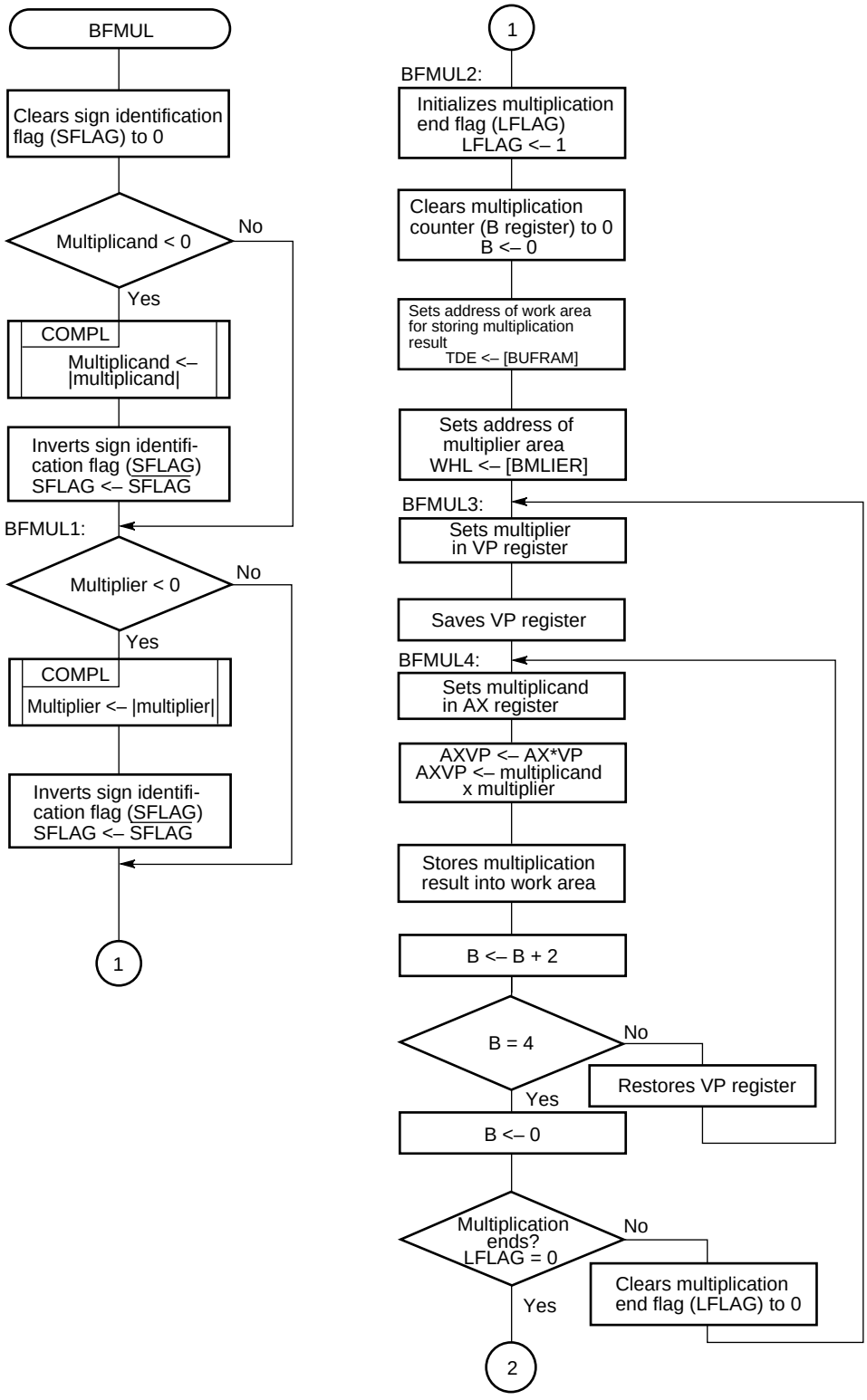
(n) Adds the values of BUFRAM+8 through BUFRAM+0DH with the carry flags (CY), 2 bytes at a time, then stores the result into the fifth and sixth bytes of the result area (BRSLT).

(o) Adds the contents of BUFRAM+0EH and BUFRAM+0FH with the carry flags (CY), then stores the result into the seventh and eighth bytes of the result area (BRSLT).

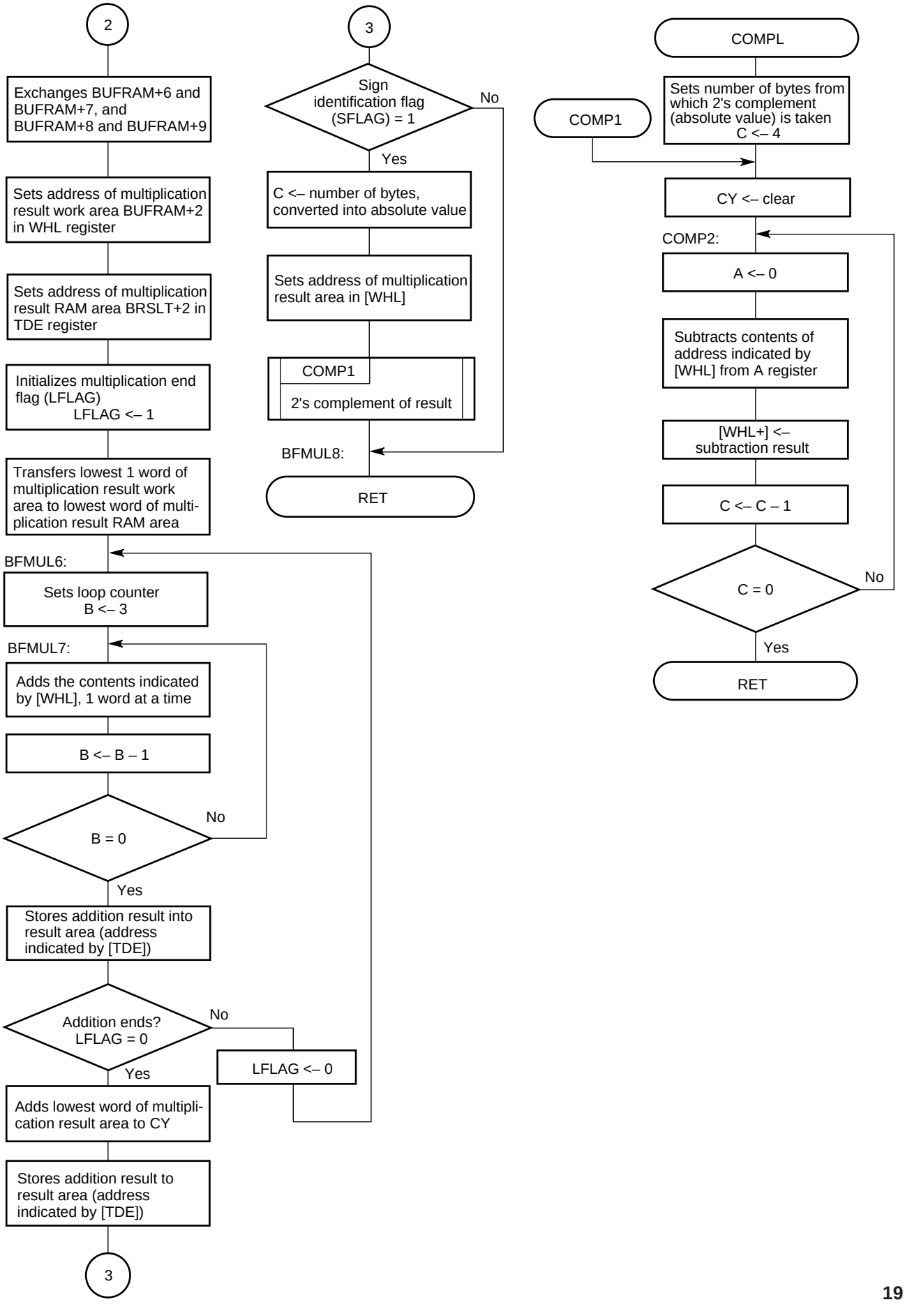
(p) If the sign identification flag (SFLAG) is 1, takes the 2's complement of the multiplication result as the result.

Remark Steps 1 through 4 correspond to the numbers shown in **Figure 2-2**.

(7) Flowchart



Phase-out/Discontinued



(8) Program listing

- **Description of label used for executing the application routine**

BMLCND : Lowest address of the RAM area containing the 32-bit multiplicand
BMLIER : Lowest address of the RAM area containing the 32-bit multiplier
BUFRAM : Lowest address of the work area that temporarily contains the 15-byte multiplication result
BRSLT : Lowest address of the RAM area that contains the final multiplication result
SFLAG : Sign identification flag
SFLAG = 0 ... Same signs
SFLAG = 1 ... Different signs
LFLAG : Multiplication end flag
LFLAG = 0 ... Multiplication has ended
LFLAG = 1 ... Multiplication has not ended

- **Example program listing for main routine**

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

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

Remark Set the multiplicand and multiplier as shown above, then call the subroutine.

- **Program listing for this application routine**

```
        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
;
```

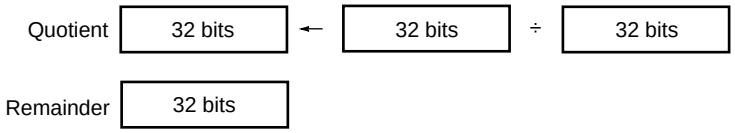
```
;       *** word multiplication process ***
;
BFMUL2:
        SET1    LFLAG                   ;
        MOV     B, #0                   ; MULT loop number clear
        MOVG    TDE, #BUFRAM            ; TDE-reg. <– BUFRAM
        MOVG    WHL, #BMLIER            ; WHL-reg. <– BMLIER
BFMUL3:
        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    LFLAG                   ;
        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
```

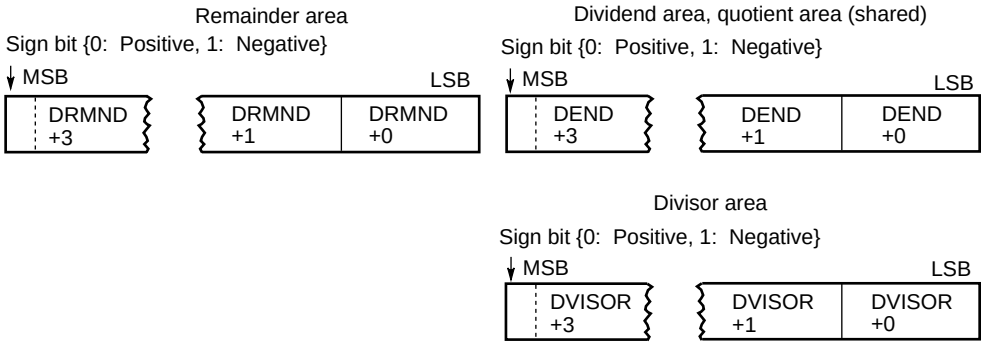
2.4 BINARY DIVISION OF SIGNED 32 BITS/32 BITS

(1) Outline of processing

This section presents an example program that divides a signed 32-bit dividend by a 32-bit divisor and stores the result into a 32-bit result area.



(2) RAM area



When the sign bit is (0): Positive (00000000H through 7FFFFFFFH)
When the sign bit is (1): Negative (FFFFFFFFH through 80000000H)

Caution The dividend and quotient (DEND+3, ..., DEND), and remainder (DRMND+3, ..., DRMND) areas must all be 8-byte contiguous RAM areas.

(3) Registers

A, X, B, C, TDE, and WHL registers

(4) Input

Set the data necessary for the operation into the following 4-byte RAM areas.

DEND to DEND+3 : 32-bit dividend data
DVISOR to DVISOR+3 : 32-bit divisor data

(5) Output

The status of the division processing is indicated by setting the following flag.

ERRFLAG: Error flag
ERRFLAG = 0 ... No error has occurred (division has been completed normally)
ERRFLAG = 1 ... An error has occurred (division cannot be executed because the divisor is 0)

The 4-byte RAM areas are used to store the results shown.

DEND to DEND+3 : Stores the quotient resulting from the division**Note**
DRMND to DRMND+3: Stores the remainder resulting from the division**Note**

Note The values before the operation are retained as these values if the error flag (ERRFLAG) = 1.

Remark The main routine checks the error flag (ERRFLAG). Add error processing as necessary.

(6) Program description

This program uses a subtract and return method as the algorithm for binary division. This algorithm is illustrated below.

• Algorithm of division by subtract and return method

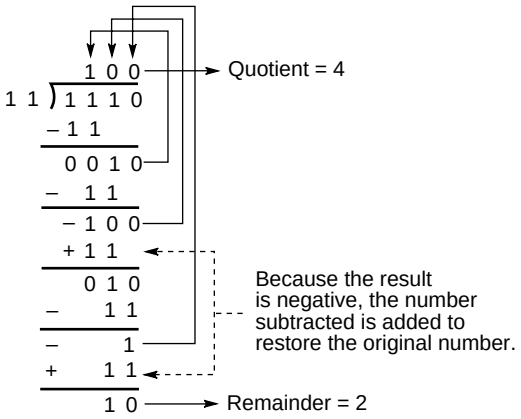
<1> Q <– 0
<2> Y <– Y x 2^{m-n}
<3> { X <– X – Y
<4> if X ≥ 0 then Q <– Q + 1
else X <– X + Y
<5> Q <– Q x 2
<6> Y <– Y/2 }
<7> { } in **<3>** through **<6>** is repeated n times.
<8> As a result, the quotient is stored into Q and the remainder is stored into X.

Remark The meanings of the above symbols are as follows.

Q: Quotient area
X : Dividend area
Y : Divisor area
m: Number of digits in dividend
n : Number of digits in divisor

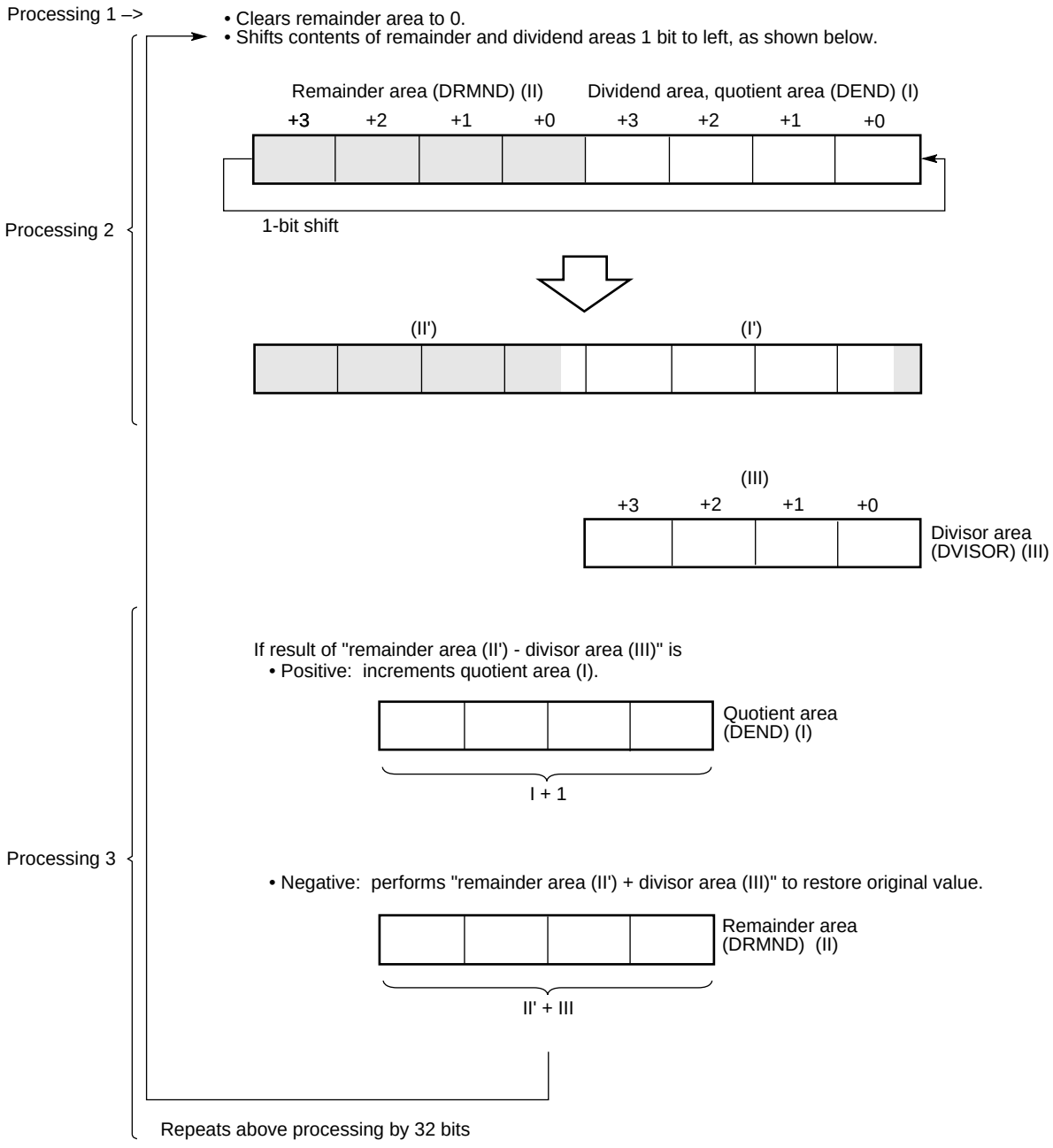
- **Example of binary division (4 bits/4 bits) using the subtract and return method**

Example 14/3 = quotient: 4, remainder: 2 (The following expression is in binary.)



- **Algorithm for binary division (32 bits/32 bits) performed by this program**

Figure 2-3. Algorithm for Binary Division



The processing performed by this programs is explained below.

Processing 1 –> (a) Determines whether the value of the divisor area is 0. If 0, sets the error flag (ERRFLAG) to 1, then ends the operation, with only information indicating the occurrence of an error remaining.
(ERRFLAG = 0 ... No error has occurred, ERRFLAG = 1 ... An error has occurred)

(b) Clears the remainder area to 0.

(c) Takes the absolute values of the dividend and divisor areas. If either of the values in the dividend and divisor areas is negative, sets the quotient sign flag (QUOFLAG) to 1.
(QUOFLAG = 0 ... Sign of quotient is positive, QUOFLAG = 1 ... Sign of quotient is negative)
Sets the remainder sign flag (REMFLAG) to 1 if the value of the dividend area is negative.
(REMFLAG = 0 ... Sign of remainder is positive, REMFLAG = 1 ... Sign of remainder is negative)

(d) Uses the B register as a bit counter that counts the number of bits in the dividend area, then sets the number of bits (32) in the dividend area in this counter.

Processing 2 –> (e) Shifts the remainder area and dividend area (8-byte contiguous area) 1 bit to the left.

Processing 3 –> (f) Executes “remainder area <– remainder area – divisor area”.
If the result is negative, jumps to (h).

Processing 4
(g) Increments the quotient area (DEND). Jumps to (i).

(h) Because too great a value has been subtracted, executes “remainder area <– remainder area + divisor area” to restore the original value of the remainder area.

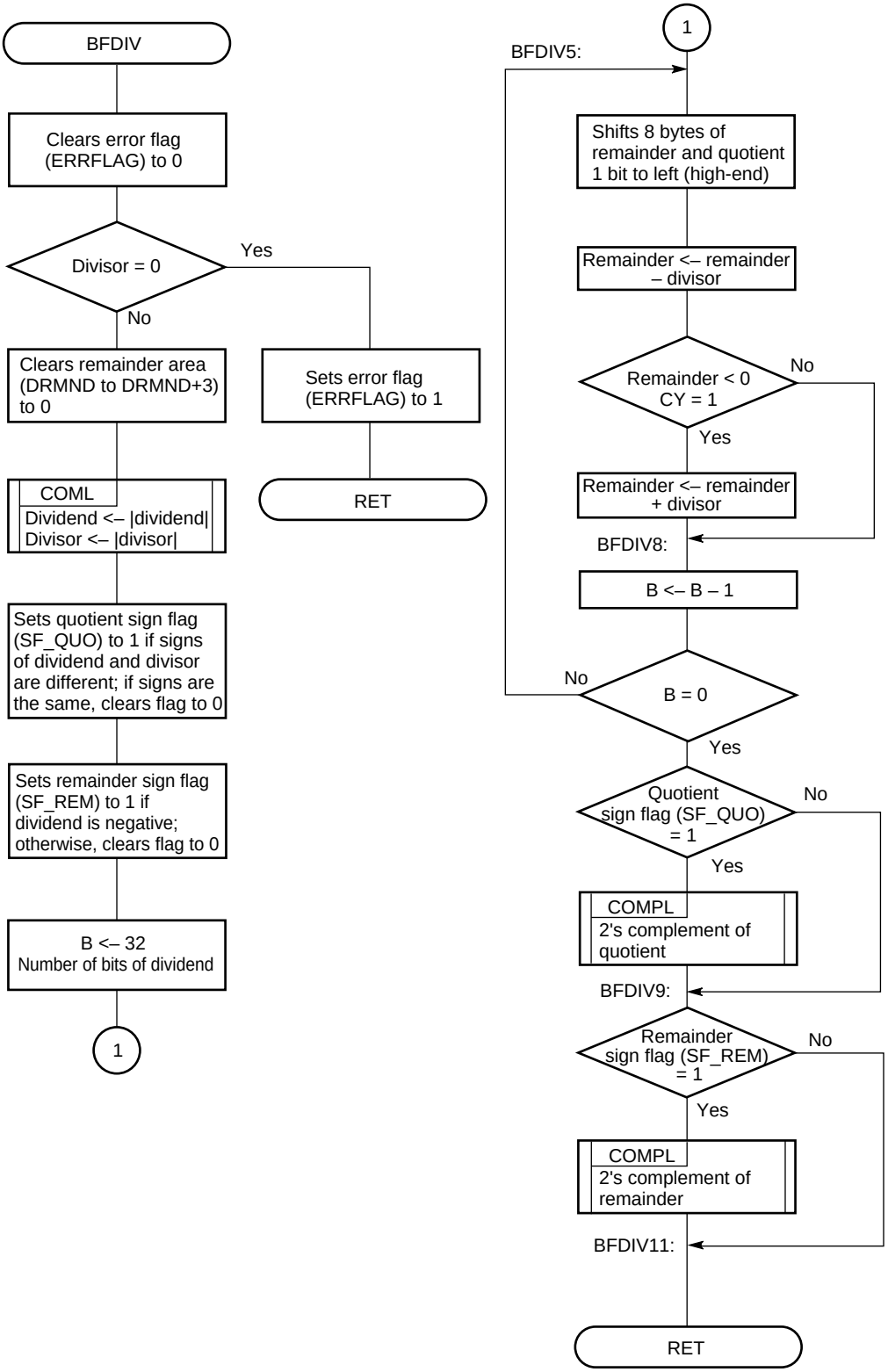
(i) Decrements the bit counter (B register) of the dividend area, and repeats steps (e) through (h) until the counter reaches 0.

(j) Checks the quotient sign flag (QUOFLAG), and takes the 2’s complement of the quotient if the flag is set to 1.
Checks the remainder sign flag (REMFLAG), and takes the 2’s complement of the remainder if the flag is set to 1.

Remarks 1. For details of the COMPL subroutine, see **Section 2.3**.

2. Processing 1 through 4 corresponds to the numbers shown in **Figure 2-3**.

(7) Flowchart



(8) Program listing

- **Description of label used for execution of application routine**

DEND : Lowest address of the RAM area containing the 32-bit dividend and 32-bit quotient (shared)
DRMND : Lowest address of the RAM area containing the 32-bit remainder that results from division
DIVSOR : Lowest address of the RAM area containing the 32-bit divisor
BYTNUM : Number of bytes in the remainder area (used to clear the remainder area to 0)
QUOFLAG : Quotient sign flag
QUOFLAG = 0 ... Sign of quotient is positive
QUOFLAG = 1 ... Sign of quotient is negative
REMFLAG : Remainder flag
REMFLAG = 0 ... Sign of remainder is positive
REMFLAG = 1 ... Sign of remainder is negative
ERRFLAG : Error flag
ERRFLAG = 0 ... An error has not occurred
ERRFLAG = 1 ... An error has occurred

- **Example of program listing for main routine**

```
        .
        .
        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
        .
        .
```

Remark Set the dividend and divisor as shown above, then call the subroutine.

- **Program listing for this application routine**

```
        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, ..., 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          ;
```

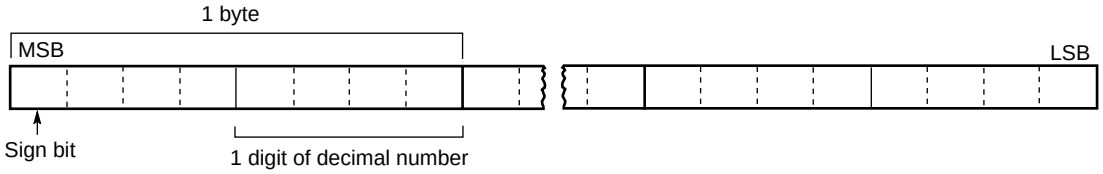
```
;
;       **** if borrow divisor + dividend ****
;
BFDIV7:
        MOVG    TDE, #DVISOR    ; TDE <– DVISOR

        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
;
```

[MEMO]

CHAPTER 3 DECIMAL OPERATIONS

In decimal operations, the most significant bit is used as a sign bit, the remaining bits expressing a numeric value, as shown in Figure 3-1. Decimal numbers are expressed as BCD codes.

Figure 3-1. Expressing Decimal Numbers

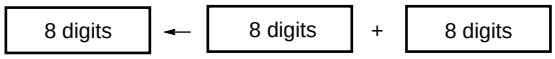


For decimal operations, the data storage area used for the operation and the area storing the result of the operation are located in RAM.

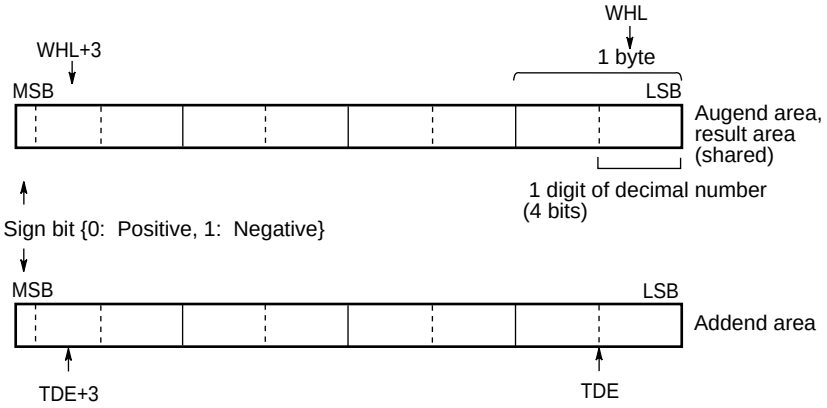
3.1 DECIMAL ADDITION OF SIGNED 8 DIGITS + 8 DIGITS

(1) Outline of Processing

This section presents an example program that adds a signed 8-digit augend to an 8-digit addend, then stores the result into an 8-digit result area (shared with the augend area).



(2) RAM area



When the sign bit is (0): Positive (0 through 79999999)
When the sign bit is (1): Negative (–1 through –79999999)

(3) Registers

A, C, B, TDE, and WHL registers

(4) Input

Set the following addresses in the WHL and TDE registers.

WHL: Lowest address of the RAM area containing the 8-digit (4-byte) augend
TDE : Lowest address of the RAM area containing the 8-digit (4-byte) addend

(5) Output

The status of the division processing is indicated by the following flag.

ERRFLAG: Error flag
ERRFLAG = 0 ... An error has not occurred (addition was completed normally)
ERRFLAG = 1 ... An error has occurred (addition cannot be executed because an overflow or underflow occurred)

The following contents are stored into the 4-byte RAM area indicated by the WHL register.

WHL to WHL+3: Stores the result of the addition[Note]

Note When the error flag (ERRFLAG) = 1, the 4-byte value in the WHL register will be undefined.

Remarks 1. The operation range is –79999999 to 79999999.

2. The error flag (ERRFLAG) is identified by the main routine. Add error processing as necessary.

(6) Program description

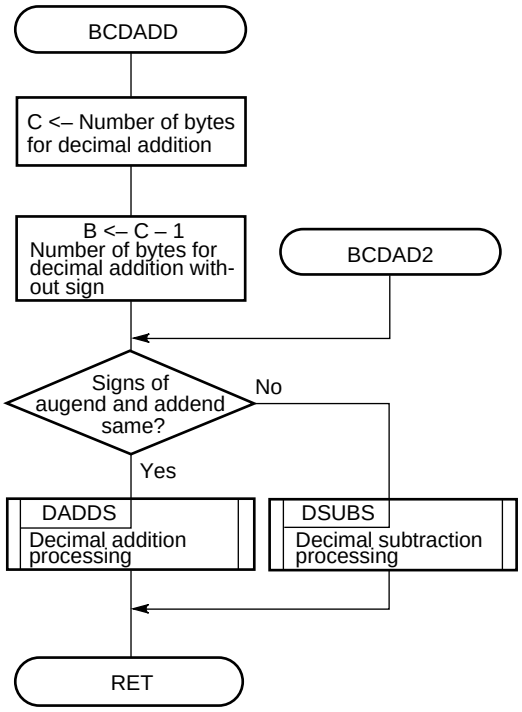
This program performs addition if the signs of the addend and augend are the same; if not, it performs subtraction.

(a) Sets the number of bytes for decimal addition in the C counter (C register).
(b) If the signs of the addend and augend are different, jumps to step (o).
(c) Clears the carry flag (CY) and sign identification flag (SFLAG) to 0.
(d) Reads 1 byte of the augend area indicated by the augend address (WHL register) into the A register.
(e) Adds the 1 byte of the addend area, indicated by the addend address (TDE register), to the A register with the carry flag (CY), and increments the addend address (TDE register).
Converts the result of the operation to a decimal number, stores it into the result area indicated by the augend address (WHL register), then increments the augend address (WHL register).
(f) Decrements the counter (B register), then repeats steps (d) through (e) until the value of the counter reaches 0.
(g) Reads 1 byte from the augend area indicated by the augend address (WHL register) into the A register.
(h) Adds 1 byte of the addend area, indicated by the addend address (TDE register) into the A register with the carry flag (CY).
(i) Jumps to step (k) if the carry flag (CY) is “0”.
(j) Sets the sign identification flag (SFLAG) to 1, then clears the carry flag (CY) to 0.
(k) Converts the value of the A register to a decimal number.
(l) If the carry flag (CY) is “1”, or if the seventh bit of the A register is “1”, an overflow occurs. In this case, sets the error flag (ERRFLAG) to 1 and terminates the operation.
(m) Sets the seventh bit of the A register into 1 if the sign identification flag (SFLAG) is “1”.
(n) Stores the contents of the A register into the result area indicated by the augend address (WHL register), then terminates the operation.
(o) Makes the subtrahend positive, then clears the sign identification flag (SFLAG) to 0.
(p) If the minuend is negative, makes the minuend positive, then sets the sign identification flag to 1.
(q) Clears the carry flag (CY) to 0.
(r) Reads 1 byte in the minuend area, indicated by the minuend address (WHL register), into the A register.
(s) Subtracts 1 byte in the subtrahend area, indicated by the subtrahend address (TDE register), from the A register with the carry flag (CY), and increments the subtrahend address (TDE register). Converts the result of the operation to a decimal number, stores it into the result area indicated by the minuend address (WHL register), then increments the minuend address (WHL register).

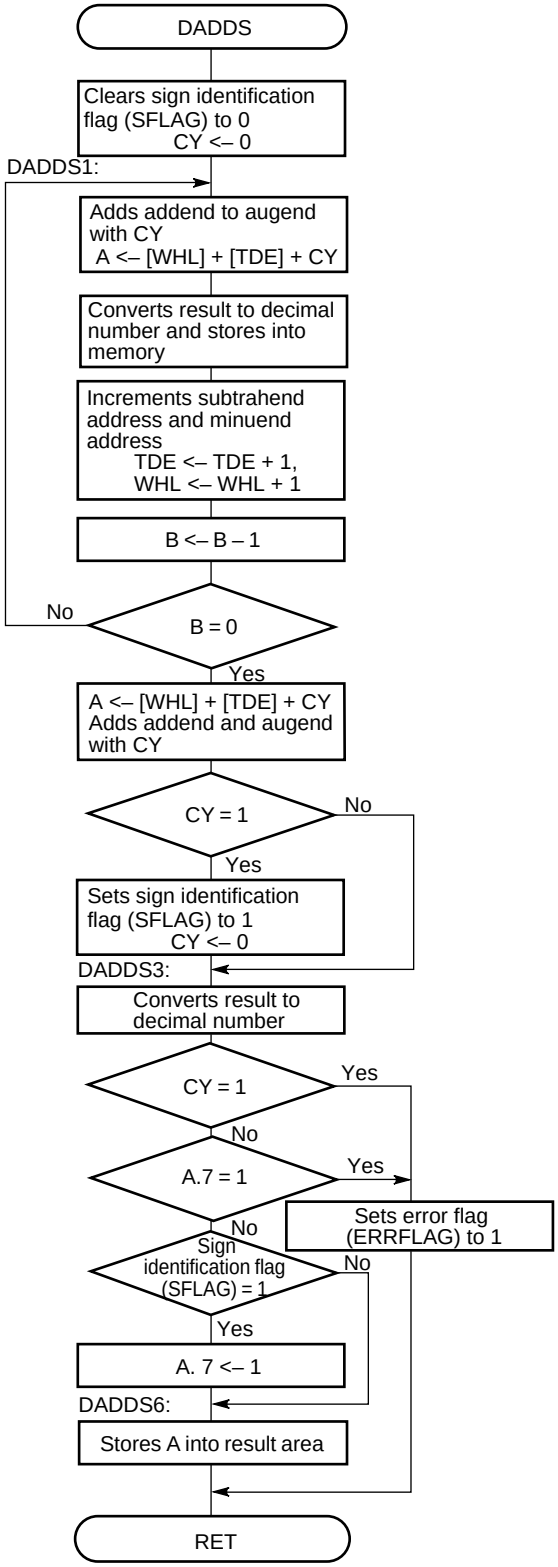
(t) Decrements the counter (C register), then repeats steps (r) and (s) until the value of the counter reaches 0.
(u) Jumps to step (w) if the carry flag (CY) is 0.
(v) Takes the 10's complement of the result and inverts the sign identification flag (SFLAG).
(w) Terminates the operation if the result is 0.
(x) Jumps to step (y) if the sign identification flag (SFLAG) is 1; if the flag is 0, terminates the operation.
(y) Sets the sign bit of the result to 1, then terminates the operation.

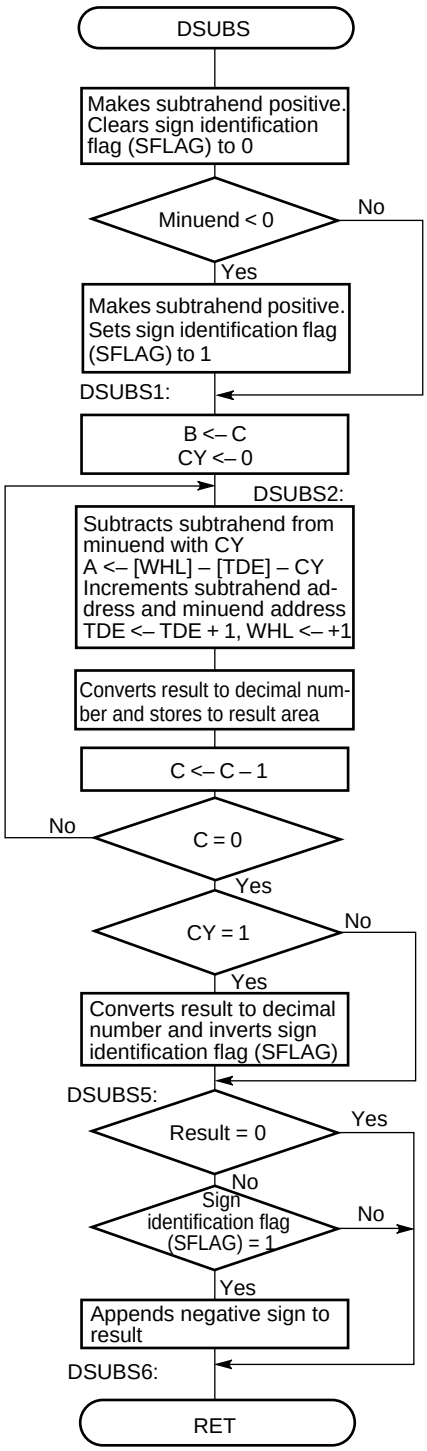
Remark For the decimal subtraction routine (steps (o) through (y)), the augend area is replaced by the minuend area, while the addend area is replaced by the subtrahend area.

(7) Flowchart



Phase-out/Discontinued





(8) Program listing

- **Description of label used for executing the application routine**

  BCDAUG : Lowest address of the RAM area containing the 8-digit (4-byte) augend and 8-digit result (shared)

  BCDADE : Lowest address of the RAM area containing the 8-digit (4-byte) addend

  SFLAG : Sign identification flag

  SFLAG = 0 ... Same signs

  SFLAG = 1 ... Different signs

  ERRFLAG: Error flag

  ERRFLAG = 0 ... No error has occurred

  ERRFLAG = 1 ... An error occurred

- **Example of program listing for main routine**

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

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

Remark Set the WHL and TDE registers as shown above, then call the subroutine. Prepare and add error processing as necessary.

- **Program listing for this application routine**

```
        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
```

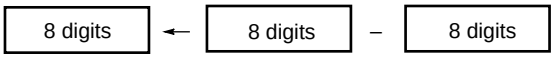

```
;=====================================================
;       ***** decimal subtraction subroutine *****
;=====================================================

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
```

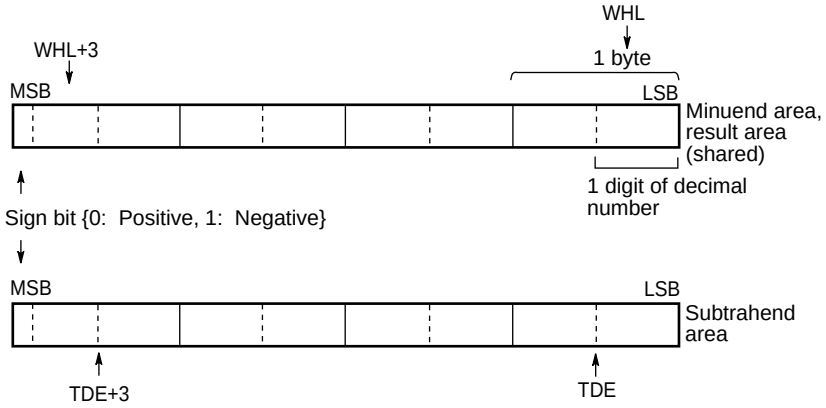
3.2 DECIMAL SUBTRACTION OF SIGNED 8 DIGITS – 8 DIGITS

(1) Outline of Processing

This section presents an example program that subtracts an 8-digit subtrahend from a signed 8-digit minuend then stores the result into an 8-digit result area (shared with the minuend area).



(2) RAM area



When the sign bit is (0): Positive (0 through 79999999)
When the sign bit is (1): Negative (–1 through –79999999)

(3) Registers

A, B, C, TDE, and WHL registers

(4) Input

Set the following addresses in the WHL and TDE registers.

WHL: Lowest address of the RAM area containing the 8-digit (4-byte) minuend
TDE : Lowest address of the RAM area containing the 8-digit (4-byte) subtrahend

(5) Output

The status of the subtraction processing is indicated by the following flag.

ERRFLAG: Error flag
ERRFLAG = 0 ... No error has occurred (subtraction was completed normally)
ERRFLAG = 1 ... An error occurred (subtraction cannot be executed because overflow or underflow occurs)

The following contents are stored into the 4-byte RAM area indicated by the WHL register.

WHL to WHL+3: Stores the result of subtraction**Note**

Note When the error flag (ERRFLAG) = 1, the 4-byte value indicated by the WHL register will be undefined.

Remarks 1. The operation range is –79999999 to 79999999.

2. The error flag (ERRFLAG) is identified by the main routine. Add error processing as necessary.

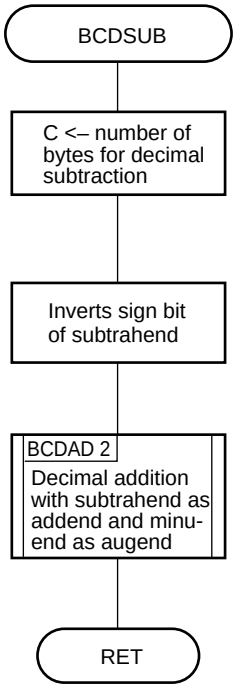
(6) Program description

This program performs the processing of “minuend – subtrahend” by converting it into “minuend + (–subtrahend)”.

(a) Sets the number of bytes for decimal subtraction into the C counter (C register).
(b) Inverts the sign bit of the subtrahend area.
(c) Performs decimal addition by using the minuend and subtrahend areas as the augend and addend areas.

Remark For details of the BCDAD2 subroutine (including the processing for setting the error flag (ERRFLAG) to 1), see **Section 3.1**.
The operation error processing is included in the main routine. Prepare and add error processing as necessary.

(7) Flowchart



(8) Program listing

- **Description of label used for executing the application routine**

  BCDMIN : Lowest address of the RAM area containing the 8-digit (4-byte) minuend and 8-digit result (shared)

  BCDSUT: Lowest address of the RAM area containing the 8-digit (4-byte) subtrahend

- **Example program listing of main routine**

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

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

Remark Set the WHL and TDE registers as shown above, then call the subroutine. Prepare and add error processing as necessary.

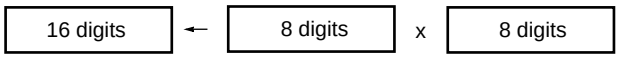
- **Program listing for this application routine**

```
        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
```

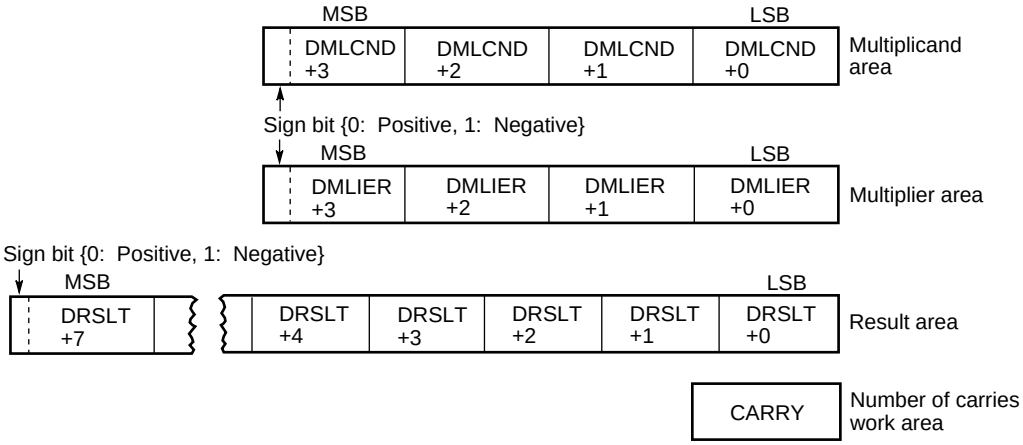
3.3 DECIMAL MULTIPLICATION OF SIGNED 8 DIGITS x 8 DIGITS

(1) Outline of Processing

This section presents an example program that multiplies a signed 8-digit multiplicand by an 8-digit multiplier, then stores the result into a 16-digit result area.



(2) RAM area



When the sign bit is (0): Positive (0 through 79999999)
When the sign bit is (1): Negative (–1 through –79999999)

(3) Registers

A, X, B, C, TDE, and WHL registers

(4) Input

Set the data necessary for the operation into the following 4-byte RAM areas.

DMLCND to DMLCND+3: 8-digit multiplicand data
DMLIER to DMLIER+3 : 8-digit multiplier data

(5) Output

The following 8-byte RAM area is used to store the contents shown.

DRSLT to DRSLT+7: Result of multiplication

Remark Both the multiplier and multiplicand can be between –79999999 and 79999999.
The operation result can be between –6399999840000001 and 6399999840000001.

(6) Program description

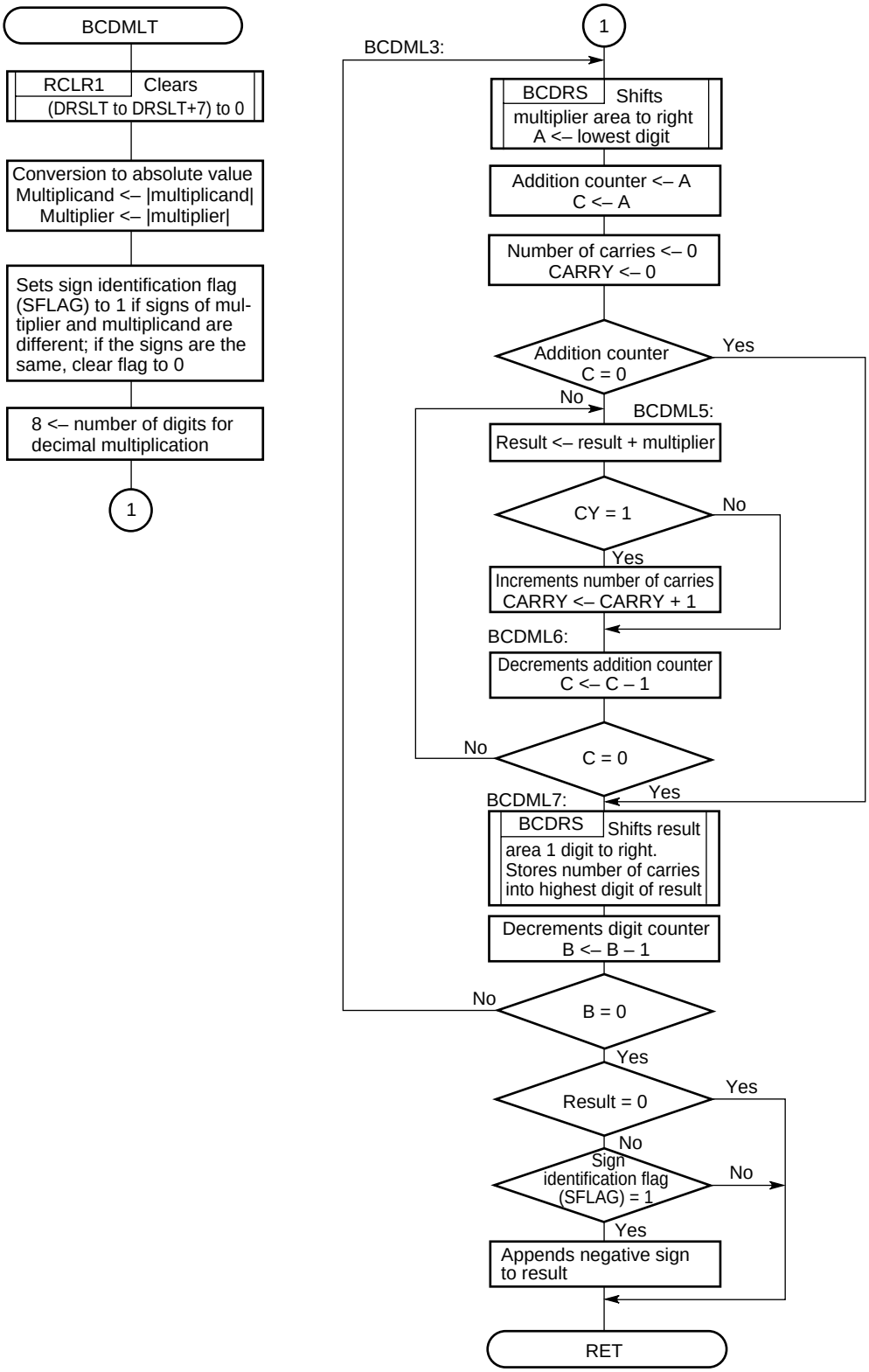
This program shifts the multiplier 1 digit (4 bits) to the right then loads the multiplicand into the addition counter, starting from the lowest digit, 1 digit at a time. Using the addition counter, the "result <– result + multiplicand" operation is repeated.

When addition using the addition counter has ended, addition is performed using the multiplier 1 digit higher; therefore, the result area is shifted 1 digit (4 bits) to the right.

The processing is explained below:

(a) Clears the result area to 0.
(b) Takes the absolute values of the multiplier area and multiplicand area. If the signs of the multiplier and multiplicand areas are different, sets the sign identification flag (SFLAG) to 1; if the signs are the same, clears the flag to 0.
(c) Sets the number of digits (8) for multiplication, using the B register as a digit counter.
(d) Sets the value of the lowest digit (4 bits) of the multiplier area into the addition counter (C register) by shifting the multiplier area 1 digit (4 bits) to the right.
(e) Clears the number of carries work area (CARRY), reserved in advance and which is used to save the number of carries, to 0.
(f) Jumps to step (i) if the value of the addition counter (C register) is 0.
(g) Executes "result area (higher 8 digits) <– result area (higher 8 digits) + multiplicand area" decimal addition processing. If an overflow occurs, increments the value in the number of carries work area (CARRY).
(h) Decrements the addition counter (C register), then repeats step (g) until the value of the counter reaches 0.
(i) Shifts the result area, including the number of carries work area (CARRY), 1 digit (4 bits) to the right. Stores the number of carries into the highest digit of the result area.
(j) Decrements the digit counter (B register), then repeats steps (d) through (i) until the value of the counter reaches 0.
(k) Terminates the operation if the contents of the digit counter (B register) are 0.
(l) Sets the sign bit of the result area to 1 if the sign identification flag (SFLAG) is 1.

(7) Flowchart



(8) Program listing

- **Description of label used for execution of application routine**

  DMLCND: Lowest address of the RAM area containing the 8-digit (4-byte) multiplicand
  DMLIER : Lowest address of the RAM area containing the 8-digit (4-byte) multiplier
  CARRY : Number of carries work area
  DRSLT : Lowest address of the RAM area used to contain the 16-digit (8-byte) result
  SFLAG : Sign identification flag
  SFLAG = 0 ... Same signs
  SFLAG = 1 ... Different signs

- **Example program listing for main routine**

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

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

CALL    !BCDMLT
      .
      .
```

Remark Set the multiplicand and multiplier as shown above, then call the subroutine.

- **Program listing for this application routine**

```
        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 ****
;
```

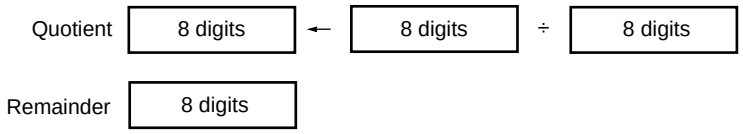
```
BCDML3:
        MOVG    WHL, #DMLIER+3
        MOV     C, #4                   ; C-register <– 4
;
        MOV     A, #0                   ; Acc <– 0
BCDRS:
        ROR4    [WHL]
        DECG    WHL                     ; decrement (WHL)
        DBNZ    C, $BCDRS

        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
```

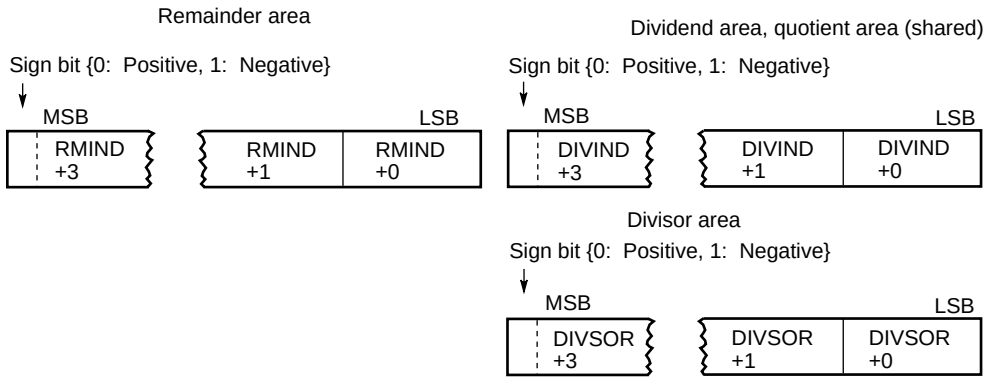
3.4 DECIMAL DIVISION OF SIGNED 8 DIGITS/8 DIGITS

(1) Outline of Processing

This section presents an example of a program that divides a signed 8-digit dividend by an 8-digit divisor, then stores the result into an 8-digit result area, and the remainder into an 8-digit remainder area.



(2) RAM area



Caution The dividend and quotient areas (DIVIND, ..., DIVIND+3) and remainder area (RMIND, ..., RMIND+3) must all be contiguous 8-byte RAM areas.

When the sign bit is (0): Positive (0 through 79999999)
When the sign bit is (1): Negative (–1 through –79999999)

(3) Registers

A, X, B, C, TDE, and WHL registers

Remark When the X register takes the absolute value of the dividend or divisor, the following bits of the register are used as flags.

- QUOFLAG: Bit 0 of X register is used as quotient sign flag
- REMFLAG: Bit 1 of X register is used as remainder sign flag

(4) Input

Set the data necessary for the operation into the following 4-byte RAM areas.

DIVIND to DIVIND+3 : 8-digit dividend data
DIVSOR to DIVSOR+3: 8-digit divisor data

(5) Output

The status of the division processing is indicated by the following flag.

ERRFLAG: Error flag
- ERRFLAG = 0 ... No error has occurred (division was completed normally)
- ERRFLAG = 1 ... An error occurred (division cannot be executed because the divisor is 0)

The following contents are stored into the following 4-byte RAM areas.

DIVIND to DIVIND+3 : Quotient resulting from division operation**Note**
RMIND to RMIND+3 : Remainder resulting from division operation**Note**

Note These values will be the same as those before the operation if the error flag (ERRFLAG) is set to 1.

Remark The value of the error flag (ERRFLAG) is determined by the main routine. Add error processing as necessary.

(6) Program description

This program uses contiguous 8-byte areas as the dividend and quotient (DIVIND, ..., DIVIND+3), and remainder (RMIND, ..., RMIND+3) areas.

The highest digit of the dividend is transferred to the lowest area of the remainder by shifting the dividend and remainder 1 digit to the left, while the quotient is stored into the lowest area of the dividend, 1 digit at a time.

Each digit of the quotient indicates the number of times the operation must be repeated until the result of "remainder – divisor" becomes negative.

(a) Identifies whether the value of the divisor area is 0. If it is 0, sets the error flag (ERRFLAG) to 1, and terminates the operation with only information indicating the occurrence of an error remaining.
(ERRFLAG = 0 ... No error has occurred, ERRFLAG = 1 ... An error occurred)

(b) Clears the remainder area to 0.

(c) Obtains the absolute values of the dividend and divisor areas.
If either the dividend or divisor areas is negative, sets the quotient sign flag (QUOFLAG) to 1.
(QUOFLAG = 0 ... Sign of quotient is positive, QUOFLAG = 1 ... Sign of quotient is negative)
If the dividend is negative, sets the remainder sign flag (REMFLAG) to 1.
(REMFLAG = 0 ... Sign of remainder is positive, REMFLAG = 1 ... Sign of remainder is negative)

(d) Sets the number of bits (8) in the dividend area by using the C register as a bit counter that counts the number of bits in the dividend area.

(e) Shifts the remainder and quotient areas (contiguous 8-byte areas) 4 bits to the left.

(f) Executes the "remainder area <– remainder area – divisor area" operation.
If the result is negative, jumps to (h).

(g) Increments the quotient area (DIVIND). Jumps to (i).

(h) Because too great a value has been subtracted from the remainder area, executes the "remainder area <– remainder area + divisor area" operation to restore the original value of the remainder area.

(i) Decrements the bit counter (C register) of the dividend area, then repeats steps (e) through (h) until the value of the counter reaches 0.

(j) Jumps to step (l) if the quotient is 0.

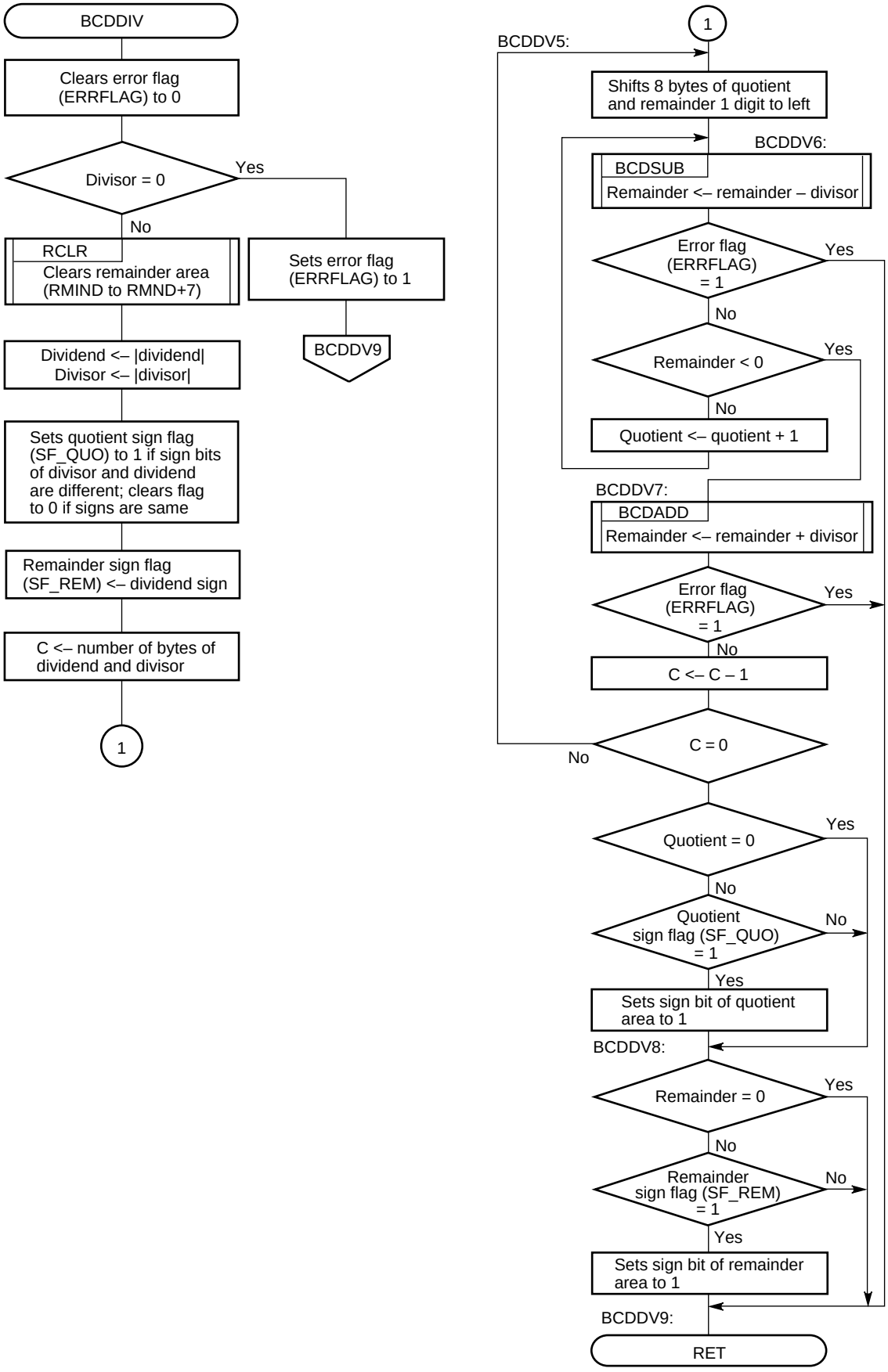
(k) Sets the sign bit of the quotient area to 1 if the quotient sign flag (QUOFLAG) is 1.

(l) Terminates the operation if the remainder is 0.

(m) Sets the sign bit of the remainder area to 1 if the remainder sign flag (REMFLAG) is 1.

Remark For details of the BCDAD2 subroutine, see **Section 3.1**. For details of the BCDSUB subroutine, see **Section 3.2**.
For details of the processing to set the error flag (ERRFLAG) to 1, and of the error processing, see **Section 3.1**.

(7) Flowchart



(8) Program listing

- **Description of label used for executing application routine**

  DIVSOR : Lowest address of the RAM area containing the 8-digit (4-byte) dividend and 8-digit quotient

  DIVIND : Lowest address of the area containing the 8-digit (4-byte) remainder resulting from division

  RMIND : Lowest address of the area containing 8-digit (4-byte) divisor

  QUOFLAG: Quotient sign flag
  QUOFLAG = 0 ... Sign of quotient is positive
  QUOFLAG = 1 ... Sign of quotient is negative

  REMFLAG: Remainder sign flag
  REMFLAG = 0 ... Sign of remainder is positive
  REMFLAG = 1 ... Sign of remainder is negative

  SFLAG : Sign identification flag
  SFLAG = 0 ... Same signs
  SFLAG = 1 ... Different signs

  ERRFLAG : Error flag
  ERRFLAG = 0 ... No error has occurred
  ERRFLAG = 1 ... An error occurred

- **Example of program listing for main routine**

```
            .
            .
        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
            .
            .
```

Remark Set the dividend and divisor as shown, then call the subroutine.

- **Program listing for this application routine**

```
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
```

[MEMO]

CHAPTER 4 SHIFT PROCESSING

The 78K/IV Series devices support instructions that shift general registers (X, A, C, B, E, D, L, H, and R4 through R11) and general register pairs (AX, BC, DE, HL, VP, UP, RP2, and RP3) in 1-bit units, as well as 4-bit shift instructions ROR4 and ROL4.

This chapter presents example programs featuring the following two types of shift instructions.

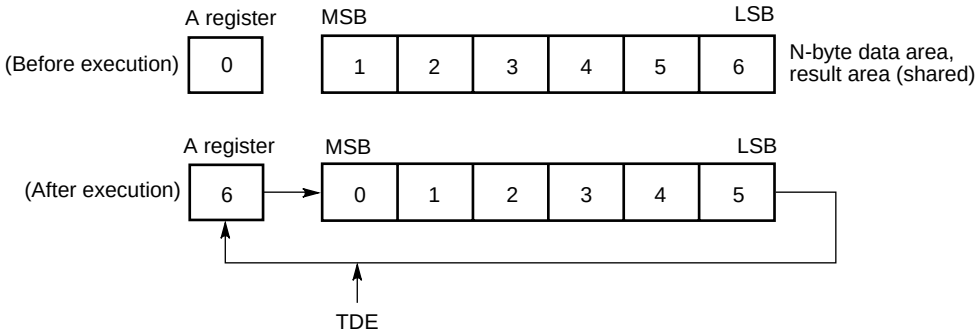
(i) Shifting in 1-byte units [XCHM [TDE+], A, XCHM [TDE–], and A instructions]
(ii) Shifting in 4-bit units [ROR4 mem and ROL4 mem instructions]

4.1 SHIFTING N-BYTE DATA 1 BYTE TO THE RIGHT

(1) Outline of processing

This section presents an example program that shifts N-byte data 1 byte to the right by using the XCHM instruction. When shifting N-byte data 1 byte to the right as shown below, set the values described in **(4)** in the appropriate registers, then execute this subroutine.

(2) RAM area



(3) Registers

A, C, and TDE registers

(4) Input

Load the following data into the A, C, and TDE registers.

A : Value to be transferred to the highest address of the N-byte data
C : Number of bytes to be shifted (N)
TDE : Highest address of the N-byte data to be shifted

(5) Output

The following contents are stored into the N-byte RAM area indicated by the TDE register.

TDE: Contents of N-byte data, shifted 1 byte to the right (The value of the A register is loaded into the most significant byte position of the N-byte data.)

(6) Program description

When the program is executed, the contents of the A register, specified as the input conditions, are exchanged with the contents of the RAM area indicated by the TDE register, after which the contents of the TDE register are decremented. Subsequently, the contents of the C register are decremented. This procedure is repeated until the contents of the C register are decremented to 0. As a result, the value set in the A register is stored into the most significant byte, and the contents of the least significant byte are output to the A register, shifting the 6-byte data 1 byte to the right.

(7) Flowchart

None

(8) Program listing

- **Labels used for execution of application routine**

  AREGDT: Value to be transferred to the most significant address of the N-byte data
  BYTNUM: Number of bytes to be shifted (N)
  R6SIFT : Least significant address of the N-byte data

- **Example program listing for main routine**

  The following shows an example of the setting necessary when 6-byte data is shifted.

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

          CALL    !BYTRST             ;right shift
              .
              .
```

Remark Set the A, C, and TDE registers as shown above, then call the subroutine.

- **Program listing for this application routine**

```
          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
```

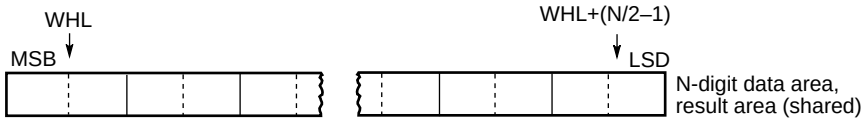
4.2 SHIFTING N-DIGIT DATA 1 BYTE TO THE RIGHT (DECIMAL 1/10 PROCESSING)

(1) Outline of processing

This section presents an example program that shifts N-digit data 1 digit to the right by using a shift instruction in 4-bit units (ROR4 mem or ROL4 mem).

When shifting N-digit data 1 digit (4 bits) to the right, set the values described in **(4)** into the appropriate registers, then execute this subroutine.

(2) RAM area



(3) Registers

A, C, and WHL registers

(4) Input

Set the following data into the WHL and C registers.

WHL : Highest address of the RAM area into which N-digit data is to be stored
C : Number of bytes to be shifted (N/2)

(5) Output

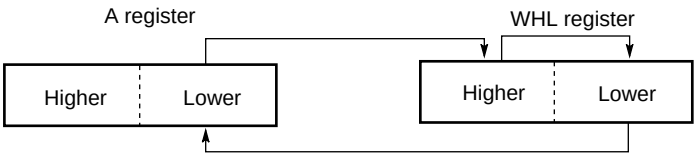
The following contents are stored into the N/2-byte RAM area indicated by the TDE register.

TDE : Contents of N-digit data, shifted 1 digit (4 bits) to the right (0 is loaded into the most significant digit of the N-digit data.)

(6) Program description

(a) Set 0 into the A register.

(b) Rotate the contents of the RAM area, indicated by the low-order 4 bits of the A register and WHL register, to the right.



(c) The C register is used as a counter that counts the number of times that data is shifted. The value of this counter is decremented.

When the value of the shift counter (C register) reaches 0, the processing has been completed. Otherwise, the processing returns to step (b).

(7) Flowchart

None

(8) Program listing

- **Labels used for execution of application routine**

  BYTNUM : Number of bytes to be shifted (N/2)

  R4SIFT : Lowest address of RAM area containing N-digit data

- **Example program listing for main routine**

  The following shows an example of setting 8-digit data.

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

Remark Set the C and WHL registers as shown above, then call the subroutine.

- **Program listing for this application routine**

```
          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                 ;
```

[MEMO]

CHAPTER 5 BLOCK TRANSFER PROCESSING

This chapter presents block transfer programs that use the string instructions that are unique to the 78K/IV Series.

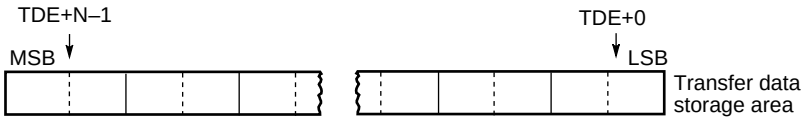
5.1 BLOCK TRANSFER PROCESSING OF FIXED BYTE DATA

(1) Outline of processing

This section presents an example program that stores 1-byte data into a specified N-byte RAM area by using the MOVM instruction.

This processing is effective for initializing a specific RAM area.

(2) RAM area



(3) Registers

A, C, and TDE registers

(4) Input

Set the following data in the A, C, and TDE registers.

A : Data to be transferred
C : Number of bytes to be transferred
TDE : Lowest address of RAM area containing N-byte transferred data

(5) Output

The following contents are stored into the RAM area, starting from the address indicated by the TDE register.

TDE: The contents of the A register are stored into an N-byte RAM area.

(6) Program description

(a) Sets the lowest address of the RAM area containing N-byte transferred data into the TDE register.
(b) Using the C register as a counter for counting the number of bytes transferred, sets the number of bytes transferred (BYTNUM).
(c) Writes the data to be transferred into the A register.
(d) Transfers the contents of the A register to the RAM area specified by the TDE register, then decrements the transfer counter (C register).
When the value of the transfer counter (C register) reaches 0, the processing is terminated.

(7) Flowchart

None

(8) Program listing

- **Labels used for execution of application routine**

  BYTNUM : Number of bytes subject to block transfer (N)

  DATASET: Lowest address of RAM area to which N-byte data is to be transferred

- **Program listing for this application routine**

  The following shows an example of writing 0 to an entire 8-byte RAM area.

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

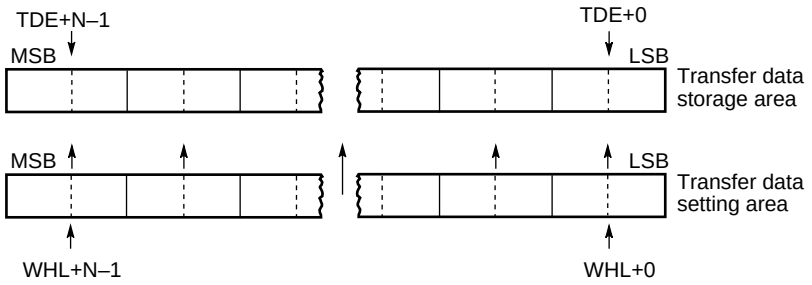
Remark When the program is developed as shown above, the value of register A is stored into the RAM area addressed by the TDE register, the number of times specified in the C register.

5.2 BLOCK TRANSFER PROCESSING OF BYTE DATA

(1) Outline of processing

This section presents an example program that transfers the contents of an N-byte RAM area indicated by the WHL register to an N-byte RAM area indicated by the TDE register by using the MOVBK instruction (MOVBK [TDE+], [WHL+]).

(2) RAM area



(3) Registers

A, C, TDE, and WHL registers

(4) Input

Set the following data in the TDE, WHL, and C registers.

TDE : Lowest address of the RAM area containing the transferred N-byte data
WHL : Lowest address of the RAM area containing the N-byte data to be transferred
C : Number of bytes to be transferred

(5) Output

The following contents are stored into the RAM area, starting from the address indicated by the TDE register.

TDE: Contents of an N-byte RAM area indicated by the WHL register

(6) Program description

(a) Sets the lowest address of the RAM area containing the N-byte data to be transferred, into the WHL register.
(b) Sets the lowest address of the RAM area to which the N-byte data will be transferred, into the TDE register.
(c) By using the C register as a transfer counter that counts the number of bytes to be transferred, sets the number of bytes to be transferred (BYTNUM).
(d) Transfers the data in the RAM area specified by the WHL register into the RAM area specified by the TDE register, then decrements the value of the transfer counter (C register).
When the value of the transfer counter (C register) reaches 0, processing ends.

(7) Flowchart

None

(8) Program listing

- **Labels used for execution of application routine**

  BYTNUM : Number of bytes subject to block transfer (N)
  TENSODT : Lowest address of the RAM area containing the N-byte data to be transferred
  KAKUNOU: Lowest address of the RAM area to which the N-byte data will be transferred

- **Program listing for this application routine**

  The following shows an example of transferring the contents of an 8-byte RAM area.

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

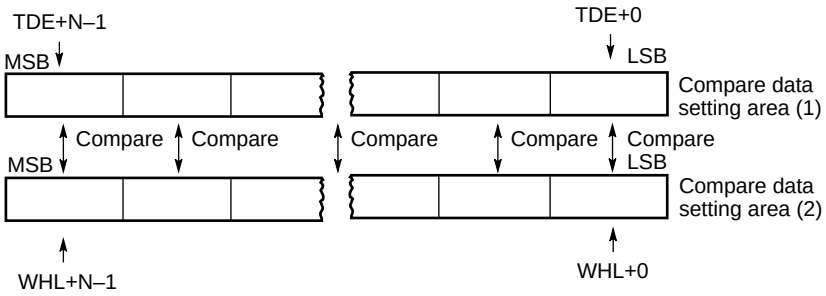
Remark When the program is developed as shown above, the value in the RAM area addressed by the WHL register is transferred to the area addressed by the TDE register, the number of times set in the C register.

5.3 BLOCK COMPARISON (COINCIDENCE DETECTION) OF BYTE DATA

(1) Outline of processing

This section presents an example program that compares (detects any coincidence between) the contents of the N-byte RAM area indicated by the WHL register with the contents of the N-byte RAM area indicated by the TDE register by using the CMPBKE instruction (CMPBKE [TDE+], [WHL+]).

(2) RAM area



(3) Registers

A, C, TDE, and WHL registers

(4) Input

Set the following data in the TDE, WHL, and C registers.

TDE : Lowest address of the RAM area (1) containing the N-byte data to be compared
WHL : Lowest address of the RAM area (2) containing the N-byte data to be compared
C : Number of bytes to be compared

(5) Output

The result of the comparison (coincidence detection) is indicated by the Z (zero) flag.

Z (zero flag): Z (zero flag) of PSW (program status word)
Z = 0 … No coincidence was detected
Z = 1 … Coincidence was detected

(6) Program description

(a) Sets the lowest address of RAM area (1) containing the N-byte data to be compared into the WHL register.

(b) Sets the lowest address of RAM area (2) containing the N-byte data to be compared into the TDE register.

(c) By using the C register as a comparison counter to count the number of bytes to be compared, sets the number of bytes to be compared (BYTNUM).

(d) Compares the data in the RAM area specified by the WHL register with the data in the RAM area specified by the TDE register, and decrements the value of the comparison counter (C register). If the data in the two RAM areas does not coincide, or if the value of the comparison counter (C register) reaches 0, processing ends.

(7) Flowchart

Omitted because the number of program steps is minimal.

(8) Program listing

- **Labels used for execution of application routine**

  COMPDTA: Lowest address of the RAM area (1) containing the N-byte data to be compared
  COMPDTB: Lowest address of the RAM area (2) containing the N-byte data to be compared
  BYTNUM : Number of bytes to be compared (N)

- **Program listing for this application routine**

  The following shows an example of comparing the contents of 8-byte RAM areas.

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

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

Remark When the program is developed as shown above, the value in the area addressed by the TDE register is compared with that in the area addressed by the WHL register, the number of times specified by the C register.

CHAPTER 6 DATA EXCHANGE PROCESSING

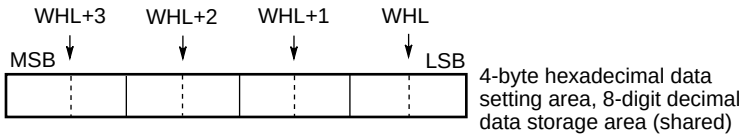
This chapter presents example programs that convert the format of numeric data from hexadecimal to decimal and vice versa, as well as from hexadecimal to ASCII and vice versa.

6.1 CONVERTING A HEXADECIMAL NUMBER (HEX) TO A DECIMAL NUMBER (BCD)

(1) Outline of processing

This section presents an example program that converts 4-byte hexadecimal data into 8-digit decimal data.

(2) RAM area



(3) Registers

A, X, D, E, UP, and WHL registers

(4) Input

Set the following address in the WHL register.

WHL: Lowest address of the RAM area containing the 4-byte hexadecimal number to be converted

(5) Output

The following flag indicates the status of the conversion processing.

ERRFLAG: Error flag
ERRFLAG = 0 … No error occurred (data conversion was completed normally)
ERRFLAG = 1 … An error occurred (data cannot be converted because it is outside the valid range)

The following contents are stored into the 4-byte area indicated by the WHL register.

WHL to WHL+3: 8-digit decimal value resulting from conversion**Note**

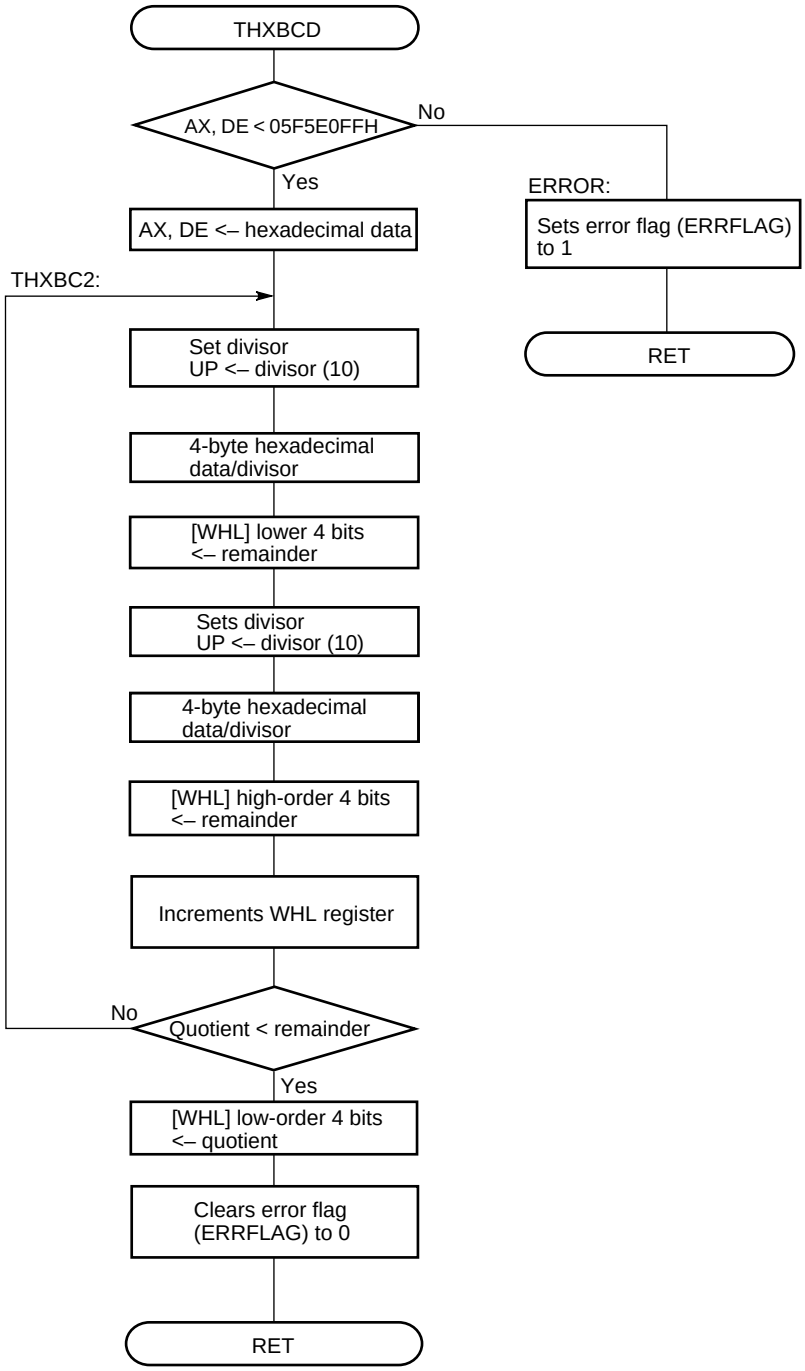
Note The value of the 4-byte data indicated by the WHL register is undefined when the error flag (ERRFLAG) = 1.

Remarks 1. Values between 00000000H (0) and 05F5E0FFH (99999999) can be converted.
2. The error flag (ERRFLAG) is read by the main routine. Add error processing as necessary.

(6) Program description

(a) Compares the 4-byte hexadecimal data indicated by the WHL register with 05F5E0FFH.
(b) If the value of the 4-byte hexadecimal data is greater than 05F5E0FFH, the error flag (ERRFLAG) is set to 1 and conversion ends because this value falls outside the supported conversion range.
(c) Sets the 4-byte hexadecimal data to be converted into the AX and DE registers (the high-order 2 bytes are written into the AX register and the low-order 2 bytes are written into the DE register).
(d) Writes the divisor (10) into the UP register.
(e) Executes “4-byte hexadecimal data/divisor” (the remainder is loaded into the UP register).
(f) Stores the remainder into the low-order 4 bits of the 8-digit decimal data storage area indicated by the WHL register.
(g) Compares the quotient in the AX and DE register with the divisor. If the quotient is greater, jumps to step (h); otherwise, stores the quotient into the high-order 4 bits of the 8-digit decimal data storage area indicated by the WHL register.
(h) Writes the divisor (10) into the UP register.
(i) Executes “input 2-byte hexadecimal data/divisor” (the remainder is loaded into the UP register).
(j) Stores the remainder into the high-order 4 bits of the 8-digit decimal data storage area indicated by the WHL register, then increments the WHL register.
(k) Compares the quotient stored in the AX and DE register with the divisor. If the quotient is greater, returns to step (d); if not, stores the quotient into the low-order 4 bits of the 8-digit decimal data storage area indicated by the WHL register.
(l) Clears the error flag (ERRFLAG) to 0 to indicate that conversion was completed without error, then terminates the conversion processing.

(7) Flowchart



(8) Program listing

- **Labels/flags used for execution of application routine**

  HEXDAT : Lowest address of the RAM area containing the 4-byte hexadecimal number to be converted

  ERRFLAG : Error flag
  ERRFLAG = 0 ... No error occurred
  ERRFLAG = 1 ... An error occurred

- **Example program listing for main routine**

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

Remark Set the WHL register as shown above then call the subroutine. Prepare and add error processing as necessary.

- **Program listing for this application program**

```
                NAME   TRBCDR
;*******************************************************
;*       transform BCD <– HEX                          *
;*          input condition                            *
;*               WHL-register <– HEX-4   byte 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          ;
```

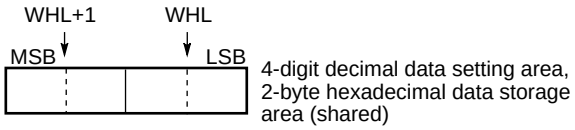
```
;
THXBC3:
        MOVW    UP, #10         ; set divisor
        DIVUX   UP              ; AXDE / UP
        PUSH    AX              ; save AX-register
        MOVW    AX, UP          ;
        ROL     X, 4            ;
        XCH     A, X            ;
        OR      [WHL+], A       ;
        POP     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 CONVERTING A DECIMAL NUMBER (BCD) TO A HEXADECIMAL NUMBER (HEX)

(1) Outline of processing

This section presents an example program that converts a 4-digit decimal number to a 2-byte hexadecimal number.

(2) RAM area



(3) Registers

A, X, B, C, DE, and WHL registers

(4) Input

Set the following address in the WHL register.

WHL: Lowest address of the RAM area containing the 4-digit decimal number to be converted

(5) Output

The following flag indicates the status of the conversion processing.

ERRFLAG: Error flag
- ERRFLAG = 0 … No error occurred (data conversion was completed normally)
- ERRFLAG = 1 … An error occurred (data could not be converted because it is not a decimal number)

The following contents are stored into the 2-byte area indicated by the WHL register.

WHL to WHL+1: 4-digit hexadecimal number resulting from conversion**Note**

Note The value of the 2-byte data indicated by the WHL register is undefined when the error flag (ERRFLAG) = 1.

Remarks 1. Values between 0 and 9999 can be converted.

2. The error flag (ERRFLAG) is read by the main routine. Add error processing as necessary.

(6) Program description

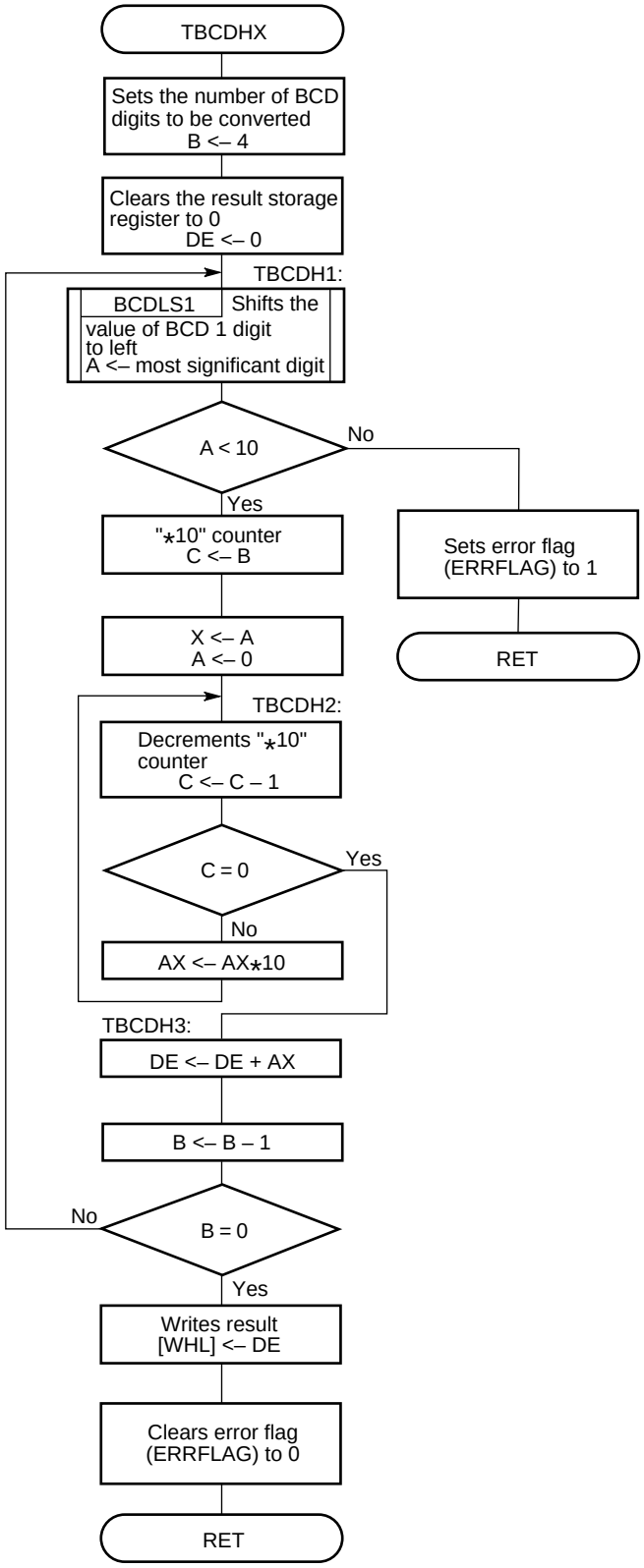
This program shifts 4-digit decimal data 1 digit (4 bits) to the left, starting from the most significant digit. It then transfers the data to the A register, 1 digit at a time, and converts the decimal data to hexadecimal data by repeating the following processing four times:

(Storage area) <– (Storage area) x 10 + A register

The processing procedure is as follows.

(a) Sets the number of digits to be converted (4) by using the B register as an address pointer to the 4-digit decimal data to be converted.
(b) By using the DE register as a result storage register for storing the conversion result, clears the value of the result storage register to 0.
(c) Shifts the value in the 4-digit decimal data setting area 1 digit (4 bits) to the left, then reads the value of the most significant digit in the 4-digit decimal data area into the A register.
(d) Checks whether the value, read from the 4-digit decimal data area into the A register, is decimal data (0 to 9). If the data is other than a decimal number, sets the error flag (ERRFLAG) to 1 to indicate that conversion is impossible.
(e) Executes “result storage register (DE register) <– result storage register (DE register) x 10 + A register.”
(f) Decrements the address pointer (B register) of the 4-digit decimal data. If the value of the address pointer is other than 0, repeats (c) to (e).
(g) Stores the value of the result storage register (DE register) into the 2-byte hexadecimal data storage area indicated by the WHL register.
(h) Clears the error flag (ERRFLAG) to 0 to indicate that the conversion has been completed without any errors, then terminates the conversion processing.

(7) Flowchart



(8) Program listing

- **Labels/flags used for execution of application routine**

  BCDDAT : Lowest address of the RAM area containing the 4-digit decimal number to be converted

  ERRFLAG: Error flag

  ERRFLAG = 0 ... No error occurred

  ERRFLAG = 1 ... An error occurred

- **Example program listing for main routine**

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

Remark Set the WHL register as shown above then call the subroutine. Prepare and add error processing as necessary.

- **Program listing for this application routine**

```
        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 CONVERTING AN ASCII CODE TO A HEXADECIMAL CODE

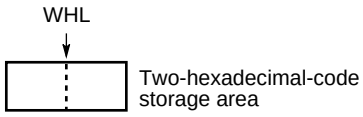
(1) Outline of processing

This section introduces an example program that converts two (2-byte) ASCII codes (30H through 39H and 41H through 46H) to two (1-byte) hexadecimal codes (00H through 0FH).
The ASCII codes and hexadecimal codes correspond as follows:

ASCII code	30H	31H	32H	33H	34H	35H	36H	37H	38H	39H
Hexadecimal code	0H	1H	2H	3H	4H	5H	6H	7H	8H	9H

ASCII code	41H	42H	43H	44H	45H	46H
Hexadecimal code	AH	BH	CH	DH	EH	FH

(2) RAM area



(3) Registers

A, B, C, and WHL registers

(4) Input

Set the following data in the BC and WHL registers.

BC : Two ASCII codes to be converted
WHL: Address of the RAM area into which the two hexadecimal codes resulting from the conversion will be stored

(5) Output

The following flag indicates the status of the conversion processing.

ERRFLAG: Error flag
ERRFLAG = 0 … No error occurred (data conversion was completed normally)
ERRFLAG = 1 … An error occurred (data cannot be converted because it consists of other than ASCII codes)

The following contents are stored into the 1-byte RAM area indicated by the WHL register.

WHL: Two hexadecimal codes resulting from the conversion[Note]

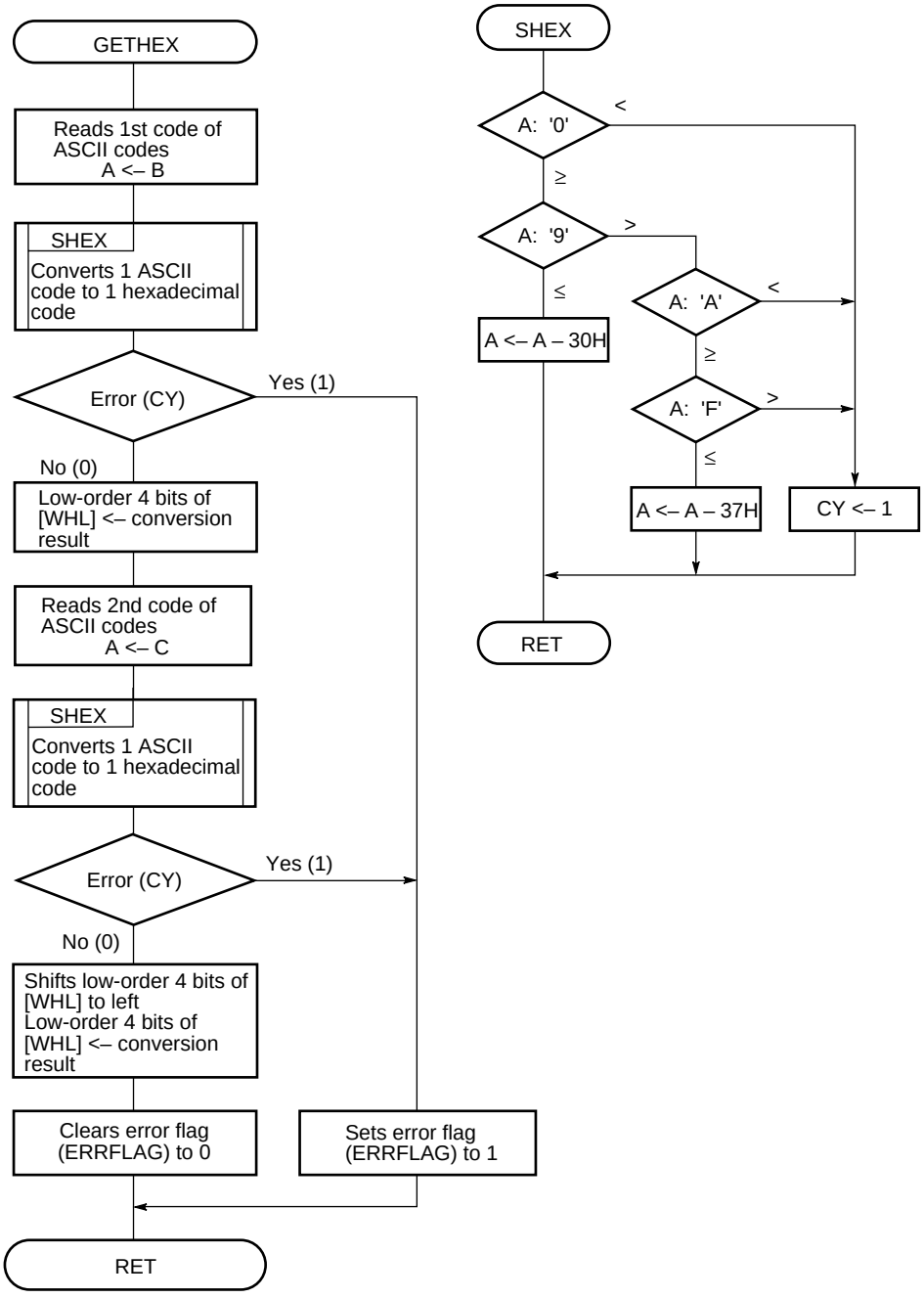
Note The 1-byte value indicated by the WHL register is undefined if the error flag (ERRFLAG) is 1.

Remark The error flag (ERRFLAG) is read by the main routine. Prepare and add error processing as necessary.

(6) Program description

(a) Reads the first code (B register) of the ASCII codes into the A register.
(b) Checks whether the contents of the A register are between 30H and 39H or between 41H and 46H. If not, sets the error flag (ERRFLAG) to 1 to indicate that conversion is impossible, then terminates the conversion processing.
(c) Subtracts 30H from the contents of the A register if the contents are between 30H and 39H. Subtracts 37H from the contents of the A register if the contents are between 41H and 46H.
(d) Shifts the contents of the two-hexadecimal-code storage area, indicated by the WHL register, 4 bits to the left, and stores the contents of the A register into the low-order 4 bits.
(e) Reads the second (C register) of the ASCII codes into the A register, then performs steps (b) through (d).

(7) Flowchart



(8) Program listing

- **Labels/flags used for execution of application routine**

  ASCDAT : Highest address of the RAM area containing the two ASCII codes (2 bytes) to be converted

  HEXDAT : Address of the RAM area containing the two hexadecimal codes (1 byte) resulting from the conversion

  ERRFLAG : Error flag
  ERRFLAG = 0 ... No error occurred
  ERRFLAG = 1 ... An error occurred

- **Example program listing for main routine**

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

Remark Set the WHL register as shown above, then call the subroutine. Prepare and add error processing as necessary.

- **Program listing for this application routine**

```
        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 CONVERTING A HEXADECIMAL CODE TO AN ASCII CODE

(1) Outline of processing

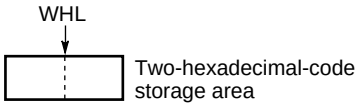
This section presents an example program that converts two (1-byte) hexadecimal codes (00H through 0FH) to two (2-byte) ASCII codes (30H through 39H and 41H through 46H).

The ASCII codes and hexadecimal codes correspond as follows:

Hexadecimal code	0H	1H	2H	3H	4H	5H	6H	7H	8H	9H
ASCII code	30H	31H	32H	33H	34H	35H	36H	37H	38H	39H

Hexadecimal code	AH	BH	CH	DH	EH	FH
ASCII code	41H	42H	43H	44H	45H	46H

(2) RAM area



(3) Registers

A, B, C, and WHL registers

(4) Input

Set the following address in the WHL register.

WHL: Address of the RAM area containing the two hexadecimal codes to be converted

(5) Output

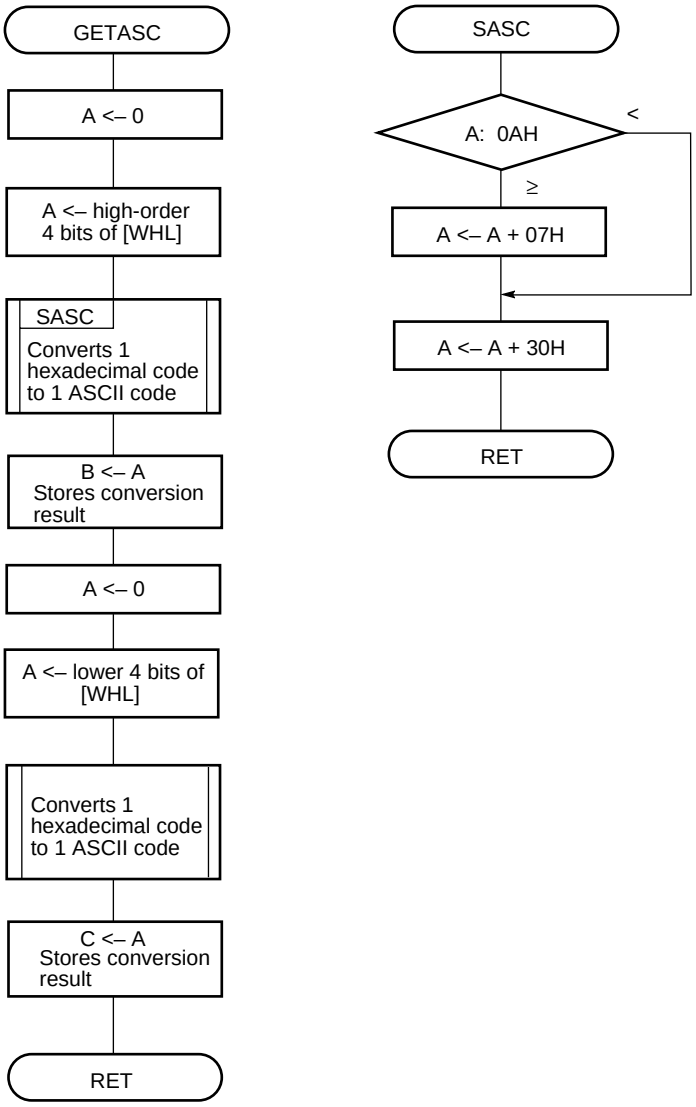
The following contents are stored into the BC register.

BC: Two ASCII codes resulting from the conversion

(6) Program description

(a) Reads the first code (high-order 4 bits) of the two-hexadecimal-code storage area, indicated by register WHL, into the A register.
(b) Checks whether the contents of the A register are greater than 10. If they are less than 10, proceeds to step (d).
(c) Adds 7 to the contents of the A register.
(d) Adds 30H to the contents of the A register.
(e) Stores the contents of the A register into the B register.
(f) Reads the second code (low-order 4 bits) in the two-hexadecimal-code storage area, indicated by the WHL register, into the A register.
(g) Performs steps (b) and (c), then stores the contents of the A register into the C register.

(7) Flowchart



(8) Program listing

- **Labels/flags used for execution of application routine**

  HEXDAT: Address of the RAM area containing the two hexadecimal codes (1-byte) to be converted

  ASCDAT: Highest address of the RAM area containing the two ASCII codes (2-byte) resulting from the conversion

- **Example program listing for main routine**

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

Remark Set the WHL register as shown above then call the subroutine. Prepare and add error processing as necessary.

- **Program listing for this application routine**

```
        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
```

CHAPTER 7 DATA PROCESSING

As an illustration of data processing, this chapter presents example programs that sort and search for data.

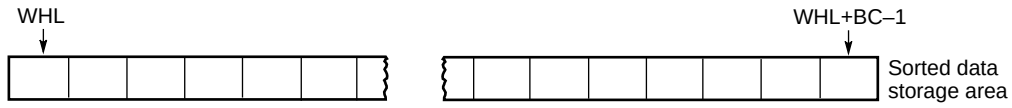
7.1 SORTING 1-BYTE DATA

(1) Outline of processing

This section presents an example program that sorts the 1-byte data in a data file into ascending order. The bubble sort method is used to sort the data.

Bubble sort involves comparing data with the subsequent data in the series, starting from the first item of data, exchanging the two data items if they are not in order. Once the last item of data in the series has been compared, processing returns to the first item of data. The procedure is repeated until no further exchange of data occurs.

(2) RAM area



(3) Registers

A, BC, and WHL registers (However, the WHL register is reserved.)

(4) Input

Set the following data in the WHL and BC registers.

WHL: Address of the RAM area into which the data is to be sorted
BC : Number of items of data (number of bytes) to be sorted

(5) Output

The following contents are stored into the RAM area indicated by the WHL register.

WHL to WHL+BC–1: Data is sorted into ascending order

(6) Program description

This program uses the bubble sort method to sort data.

The processing procedure is as follows.

(a) Clears the exchange flag (CHGFLAG) that indicates that exchange has been executed to 0.

(b) By using the BC register as a byte counter to indicate the number of bytes of data to be sorted, decrements the byte counter. If the value of the byte counter is 0, terminates the sort processing.

(c) Saves the WHL and BC registers.

(d) Compares the value in the sorted data storage area indicated by the WHL register ([WHL]) with the value in the sorted data storage area indicated by WHL register + 1 ([WHL+1]).
If [WHL] < [WHL+1], jumps to step (e).
If [WHL] ≥ [WHL+1], jumps to step (f).

(e) Exchanges the contents of the sorted data storage area indicated by the WHL register with the contents of the sorted data storage area indicated by the WHL register + 1, then sets the exchange flag (CHNGFL) to 1.

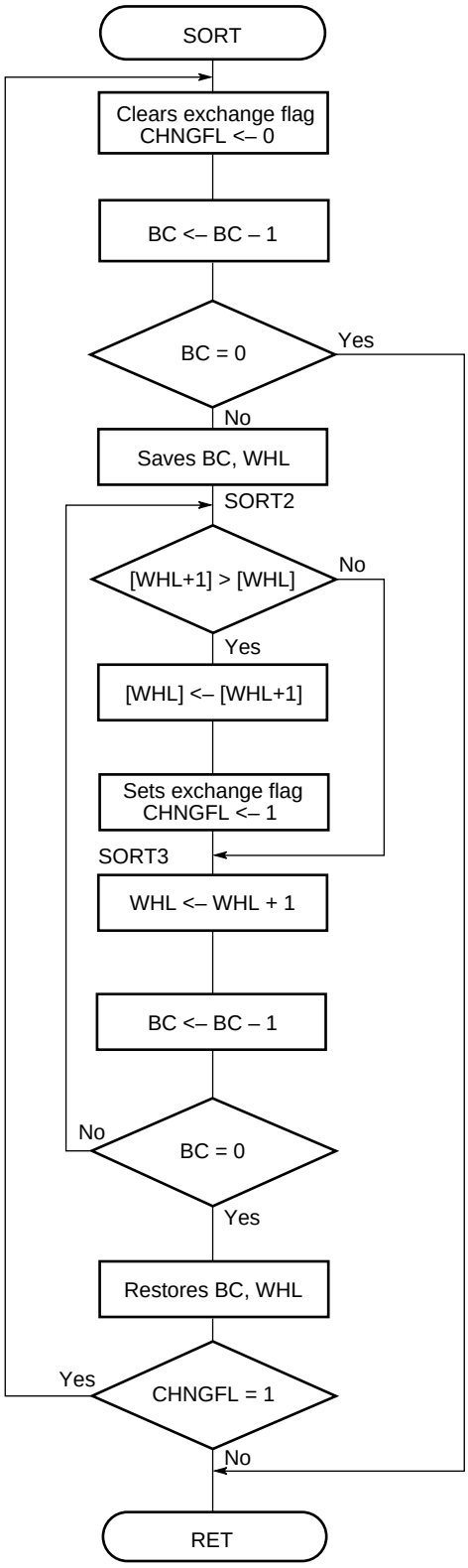
(f) Increments the WHL register indicating the address of the sorted data storage area, then decrements the byte counter (BC register).

(g) If the value of the byte counter (BC register) is other than 0, repeats steps (d) through (f).

(h) Restores the data for sorting, saved in step (c), into the WHL register that indicates the address of the sorted data storage area, and into the BC register indicating the number of bytes in the data arrangement.

(i) Repeats steps (a) through (h) if the exchange flag (CHGFLAG) is set to 1. Otherwise, ends the sorting processing.

(7) Flowchart



(8) Program listing

- **Labels used for execution of application routine**

  SORTDAT : First address of data array to be sorted

  BC register: Number of data items to be sorted

  CHGFLAG : Flag indicating whether data items have been exchanged

  CHGFLAG = 1 ... Data items have been exchanged

  CHGFLAG = 0 ... Data items have not been exchanged

- **Example of program listing for main routine**

```
      .
      .
MOVW    BC, #10H              ;data length is 16 bytes
MOVG    WHL, #SORTDAT
CALL    !SORT
      .
      .
```

Remark Set the BC and WHL register as shown above, then call the subroutine.

- **Program listing for this application routine**

```
        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 SEARCHING FOR DATA

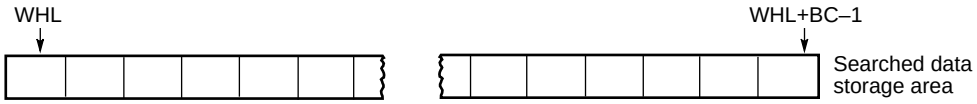
(1) Outline of processing

This section presents an example program that searches for specific data and returns the corresponding storage address once that data has been found. Binary search is used to search for data.

Binary search involves searching for a specified data string in a collection of data.

In this example, a collection of data that has been sorted into ascending order is searched. The data to be searched is compared with the intermediate value for the collection of data, half the collection of data being deleted depending on which of the searched data or intermediate value is greater. By executing this operation repeatedly, the specified data can be located.

(2) RAM area



(3) Registers

A, BC, WHL, UUP, and VVP registers

(4) Input

Set the following data in the A, WHL, and BC registers.

A : Data to be searched
WHL : First address of the searched data storage area (in which data has already been sorted into ascending order)
BC : Amount of data (bytes) to be searched

(5) Output

The following flag indicates the status of the search processing.

CY: Carry flag
CY = 0 ... End of search
CY = 1 ... Specified data could not be found

The following contents are stored into the WHL register.

WHL register: Address of the searched data**Note**

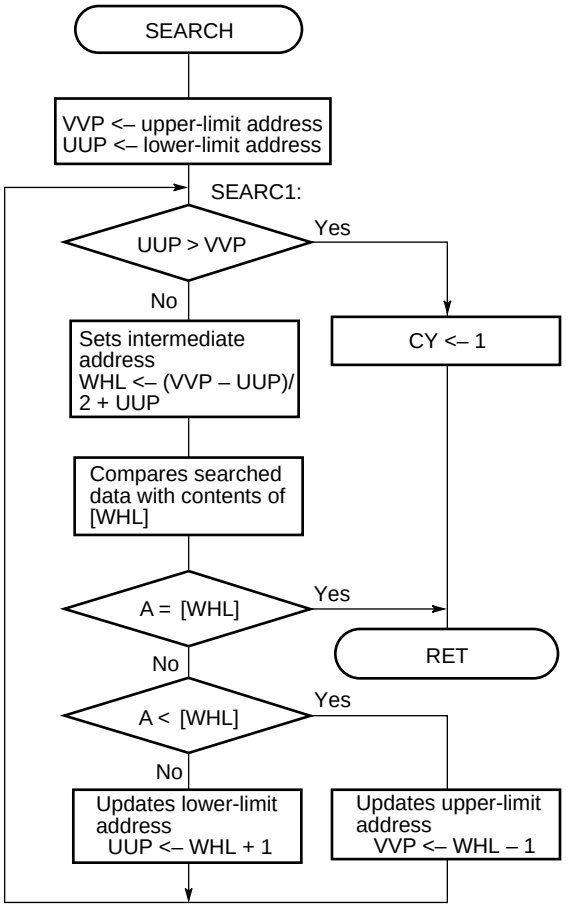
Note If the specified data is not found, the value of the WHL register will be undefined.

(6) Program description

This program uses binary search. The processing procedure is described below.

(a) Writes the first address of the searched data storage area into the UUP register, and the last address of the searched data storage area into the VVP register.
(b) Compares the first address of the searched data storage area (UUP register) with the last address of the searched data storage area (VVP register).
If UUP register ≤ VVP register, proceeds to step (d).
If UUP register > VVP register, proceeds to step (c).
(c) Sets the carry flag (CY) to 1 then terminates the processing.
(d) Sets the intermediate address of the searched data storage area (intermediate address indicated by the UUP and VVP registers) into the WHL register.
(e) Compares the searched data with the contents of the intermediate address of the searched data storage area indicated by the WHL register (intermediate address indicated by the UUP and VVP registers). If the searched data coincides with the contents of the intermediate address (i.e., when the data has been found), ends the search processing.
(f) If the carry flag (CY) is 1, executes “VVP <– WHL – 1” then sets a new last address. If the carry flag is 0, executes “UUP <– WHL + 1”, sets a new first address, then returns to step (c).

(7) Flowchart



(8) Program listing

- **Labels used for execution of application routine**
  SORTDAT: First address of data array to be sorted

- **Example program listing for main routine**

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

Remark Set the BC, WHL, and A registers as shown above, then call the subroutine.

- **Program listing for this application routine**

```
        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
```