MITSUBISHI 16-BIT SINGLE-CHIP MICROCOMPUTER
M16C FAMILY

M16C/60
M16C/20
SERIES

<Sample program>

Application note

MITSUBISHI
ELECTRIC

keep safety first in your circuit designs !

- Mitsubishi Electric Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of non-flammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Mitsubishi semiconductor product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Mitsubishi Electric Corporation or a third party.

- Mitsubishi Electric Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts or circuit application examples contained in these materials.

- All information contained in these materials, including product data, diagrams and charts, represent information on products at the time of publication of these materials, and are subject to change by Mitsubishi Electric Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor for the latest product information before purchasing a product listed herein.

- Mitsubishi Electric Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.

- The prior written approval of Mitsubishi Electric Corporation is necessary to reprint or reproduce in whole or in part these materials.

- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
  Any diversion or reexport contrary to the export control laws and regulations of JAPAN and/or the country of destination is prohibited.

- Please contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor for further details on these materials or the products contained therein.

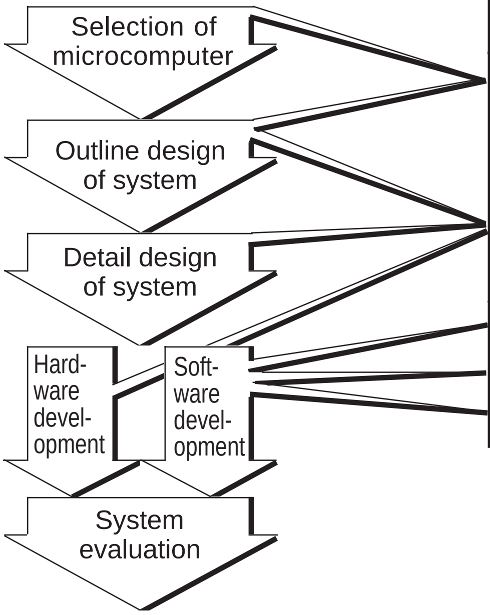
Preface

This collection of reference programs relate to the M16C/60, M16C/20 series of Mitsubishi 16-bit single-chip microcomputers. It contains sample programs and arithmetic libraries that have been prepared in an attempt to provide a useful means of understanding the instruction set available for the M16C/60, M16C/20 series and materials that can be referenced when actually developing your applications software.

For details about the M16C/60, M16C/20 series instruction set, please refer to the “M16C/60, M16C/20 series software manual”.

M16C Family-related document list

Usages
(Microcomputer development flow)



	Type of document	Contents
Hardware	Data sheet and data book	Hardware specifications (pin assignment, memory map, specifications of peripheral functions, electrical characteristics, timing charts)
Hardware	User's manual	Detailed description about hardware specifications, operation, and application examples (connection with peripherals, relationship with software)
Software	Programming manual	Method for creating programs using assembly and C languages
Software	Software manual	Detailed description about operation of each instruction (assembly language)

M16C Family Line-up

- M16C Family
  - M16C/80 Series
    - M16C/80 Group
  - M16C/60 Series
    - M16C/60 Group
    - M16C/61 Group
    - M16C/62 Group
  - M16C/20 Series
    - M16C/20 Group
    - M16C/21 Group

Table of contents

Chapter 3 Program Collection of Mathematic/Trigonometric Functions

Index

Chapter 1

Guide to Using This Manual

1. Guide to Using This Manual

This manual contains sample programs in Chapter 2, "Collection of General-purpose Programs," and arithmetic libraries in Chapter 3, "Collection of Mathematic/Trigonometric Programs." These programs are expected to provide you with useful materials that can be referenced when developing M16C/60, M16C/20 series programs. When actually using the sample programs or arithmetic libraries contained in this manual, please be sure to verify the operation of your program before putting it to work in your application.

1.1 Program Configuration

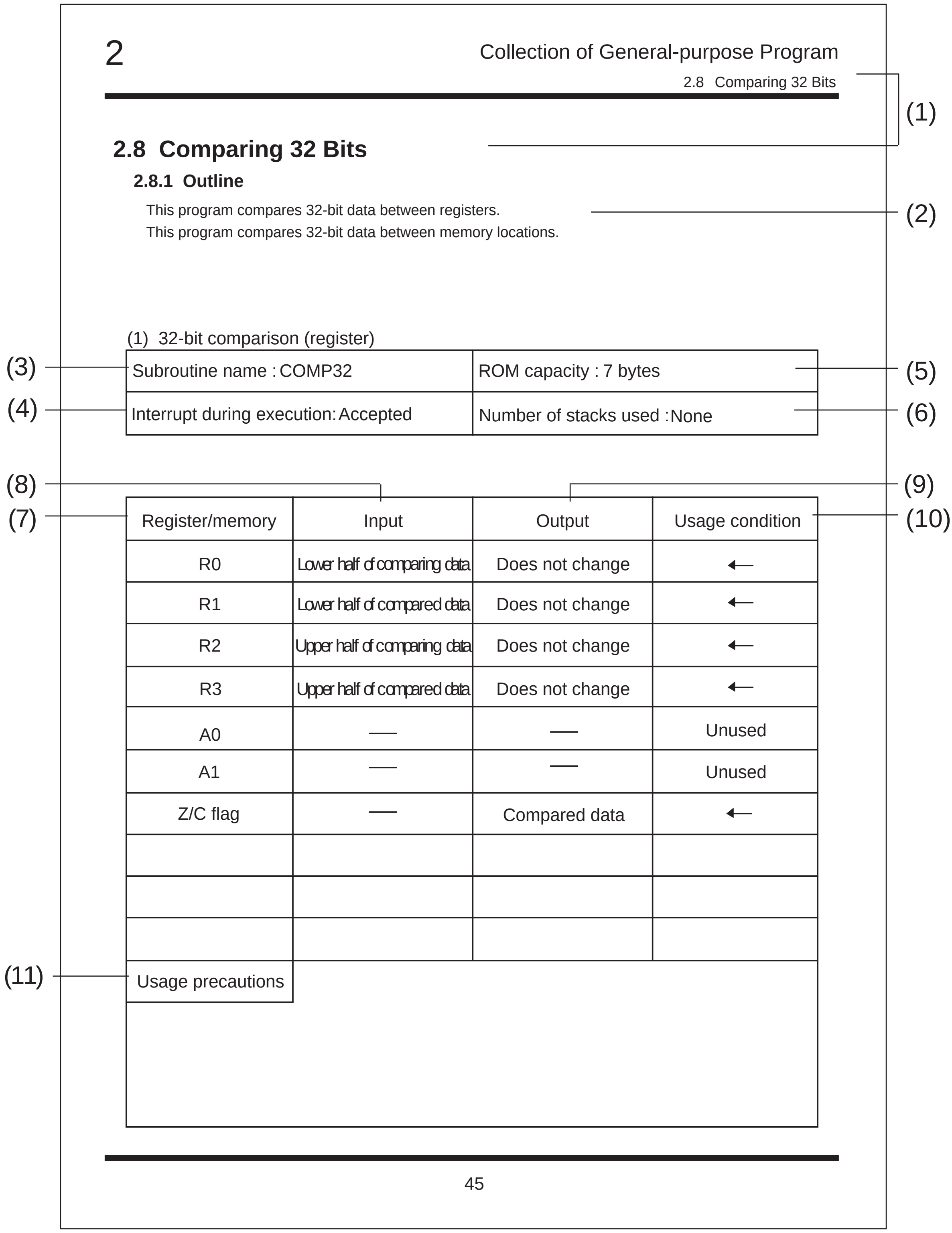
Each sample program contained in this manual consists of the following four items:

Sample program
- (1) Outline
- (2) Explanation
- (3) Flowchart
- (4) Program list

The arithmetic libraries each consist of items (1) and (2) above.
The next pages show you how to read each item (1) through (4).

1.1.1 Outline

The following shows the format of the item “Outline” and how to read it.



2 Collection of General-purpose Program
2.8 Comparing 32 Bits

2.8 Comparing 32 Bits

2.8.1 Outline

This program compares 32-bit data between registers.
This program compares 32-bit data between memory locations.

(1) 32-bit comparison (register)

Subroutine name : COMP32	ROM capacity : 7 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of comparing data	Does not change	←
R1	Lower half of compared data	Does not change	←
R2	Upper half of comparing data	Does not change	←
R3	Upper half of compared data	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
Z/C flag	—	Compared data	←

Usage precautions

45

(1) Function name

It indicates the name of the function performed.

(2) Outline

It indicates the outline function of the program.

(3) Number of execution cycles

It indicates the number of execution cycles required when the program is executed.

(4) Interrupt during execution

It indicates whether an interrupt will be accepted during program execution. If it indicates "Unaccepted," be sure to disable interrupts before you start executing the program.

(5) ROM capacity

It indicates the ROM capacity required for the program.

(6) Number of stacks used

It indicates the number of stacks required for the program. It does not include the stack capacity necessary to call the program as a subroutine.

Allocate the stack capacity shown below before executing the program.

Examples: (3), (4), (5), and (6)

Subroutine name : COMP32	ROM capacity : 7 bytes
Interrupt during execution: Accepted	Number of stacks used : None

(7) Register/memory

It indicates the registers and memory locations used in the program. Memory locations are allocated by the names shown here.

(8) Input

It indicates the input arguments required when executing the program. If any input argument is required, store the data in the register or memory location to be operated on before executing the program. If there is no input argument required, a dash “-” will be indicated here.

(9) Output

It indicates the register and memory status after executing the program.

“-”: No register or memory is used.

“Does not change”: The input data stored before executing the program is retained.

“Indeterminate”: The register or memory content is destroyed by executing the program.

(Returned value): The output return value (result) is stored by executing the program.

(10) Usage condition

It indicates the purpose of use for which a register or memory is used. If an arrow “←” is shown here, see the input and output columns.

(11) Usage precautions

It indicates the precautions to be observed for the purposes of data processing.

Examples: (7), (8), (9), (10), and (11)

Register/memory	Input	Output	Usage condition
R0	Lower half of comparing data	Does not change	←
R1	Lower half of compared data	Does not change	←
R2	Upper half of comparing data	Does not change	←
R3	Upper half of compared data	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
Z/C flag	—	Compared data	←
Usage precautions			

1.1.2 Explanation

The following shows the format of the item "Explanation".

2 Collection of General-purpose Program

2.8 Comparing 32 Bits (1)

2.8.2 Explanation (2)

This program compares 32-bit data between registers. Set the comparing data in R2 and R0 and the compared data in R3 and R1 beginning with the upper half, respectively. The comparison result is output to the Z and C flags.
This program compares 32-bit data between memory locations. Set the least significant memory address of the comparing data and that of the compared data in the address registers. The comparison result is output to the Z and C flags.

C	Z	Meaning
1	0	Comparing data < compared data
1	1	Comparing data = compared data
0	0	Comparing data > compared data

(1) Function name

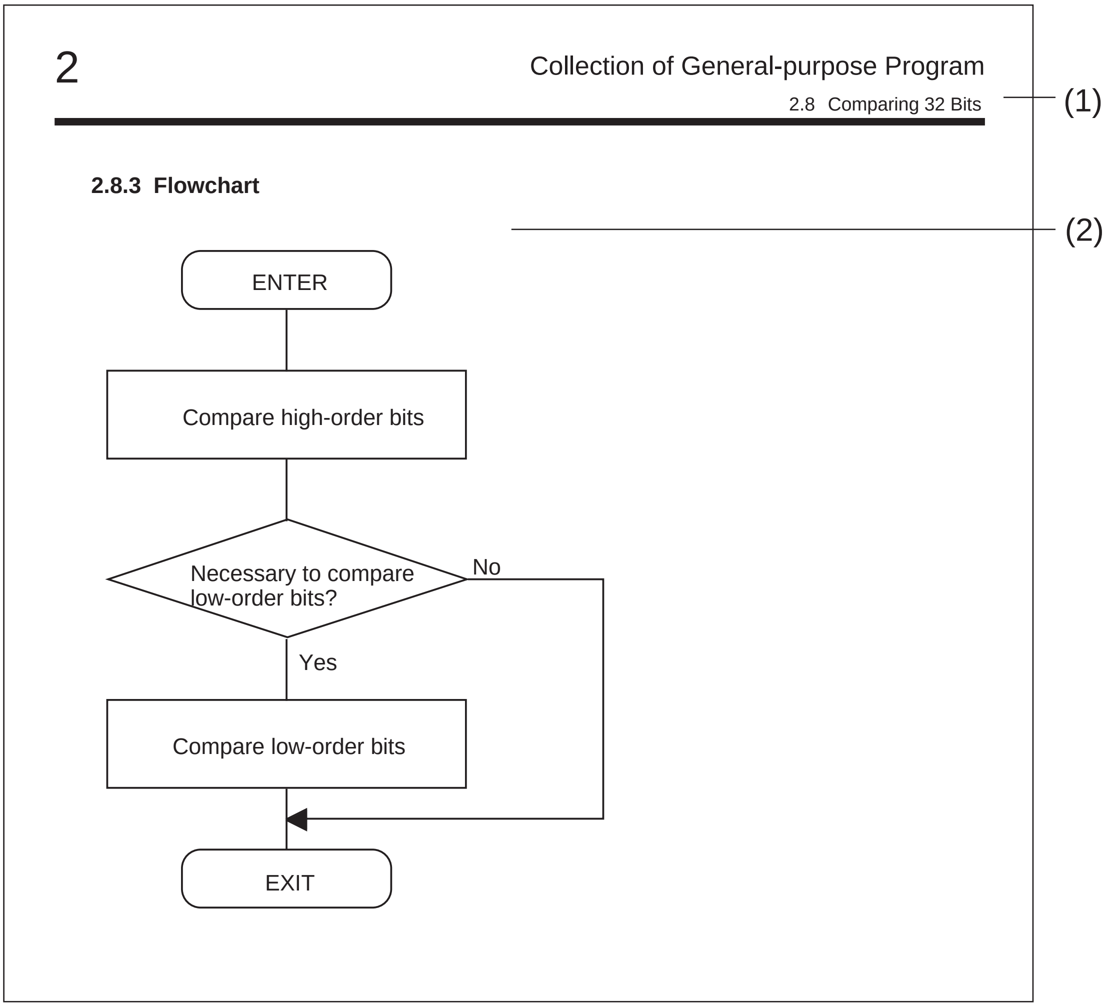
It indicates the name of the function performed.

(2) Explanation

It indicates how the program operates.

1.1.3 Flowchart

The following shows the format of the item “Flowchart”.



(1) Function name

It indicates the name of the function performed.

(2) Flowchart

It indicates the flowchart of the program.

1.1.4 Program list

The following shows the format of the item “Program list” and how to read it.

2 Collection of General-purpose Program

2.8 Comparing 32 Bits — (1)

2.8.4 Program List

```
;*******************************************************************************************
;                                                                                          *
;          M16C Program Collection No. 8                                                   *
;          CPU                 : M16C                                                      *
;                                                                                          *
;*******************************************************************************************
VromTOP                  .EQU     0F0000H                ; Declares start address of ROM      (2)
;                                                        ;
;=============================================================
;          Title: Comparing 32 bits
;          Outline: Compares 32-bit data between registers
;          Input:    ----------------------------->   Output:
;          R0   (Lower half of comparing data)        R0   (Does not change)
;          R1   (Lower half of compared data)         R1   (Does not change)
;          R2   (Upper half of comparing data)        R2   (Does not change)                 (3)
;          R3   (Upper half of compared data)         R3   (Does not change)
;          A0   ( )                                   A0   (Unused)
;          A1   ( )                                   A1   (Unused)
;          Stack amount used: None
;          Notes: Result is returned by Z and C flags
;=============================================================
                         .SECTION      PROGRAM,CODE
                         .ORG          VromTOP           ; ROM area
COMP32:                                                  ;
          CMP.W          R2,R3                           ; Compares high-order bits
          JNE            COMP32exit                      ; --> Result is output after comparing only high-order bits   (4)
          CMP.W          R0,R1                           ; Compares low-order bits
COMP32exit:                                              ;
          RTS                                            ;
;                                                        ;
;=============================================================
;          Title: Comparing 32 bits
;          Outline: Compares 32 bits between memory locations
;          Input:    ----------------------------->   Output:
;          R0   ( )                                   R0   (Unused)
;          R1   ( )                                   R1   (Unused)
;          R2   ( )                                   R2   (Unused)                          (3)
;          R3   ( )                                   R3   (Unused)
;          A0   (Address of comparing data)           A0   (Does not change)
;          A1   (Address of compared data)            A1   (Does not change)
;          Stack amount used: None
;          Notes: Result is returned by Z and C flags
;=============================================================
COMPmemory32:                                            ;
          CMP.W          2[A0],2[A1]                     ; Compares high-order bits
          JNE            COMPmemory32exit                ; --> Result is output after comparing only high-order bits.
          CMP.W          [A0],[A1]                       ; Compares low-order bits           (4)
COMPmemory32exit:                                        ;
          RTS                                            ;
;                                                        ;
                         .END                            ;
```

49

(1) Function name

It indicates the name of the function performed.

(2) Initial setup section

This is the program's initial setup section. Following settings are made here as necessary:

- Declares the start address of a memory area.
- Declares the start address of the program.
- Defines symbols.
- Allocates the memory area.

(3) Specification explanation section

This is the program's specification explanation section. Program specifications are explained here in order of the following:

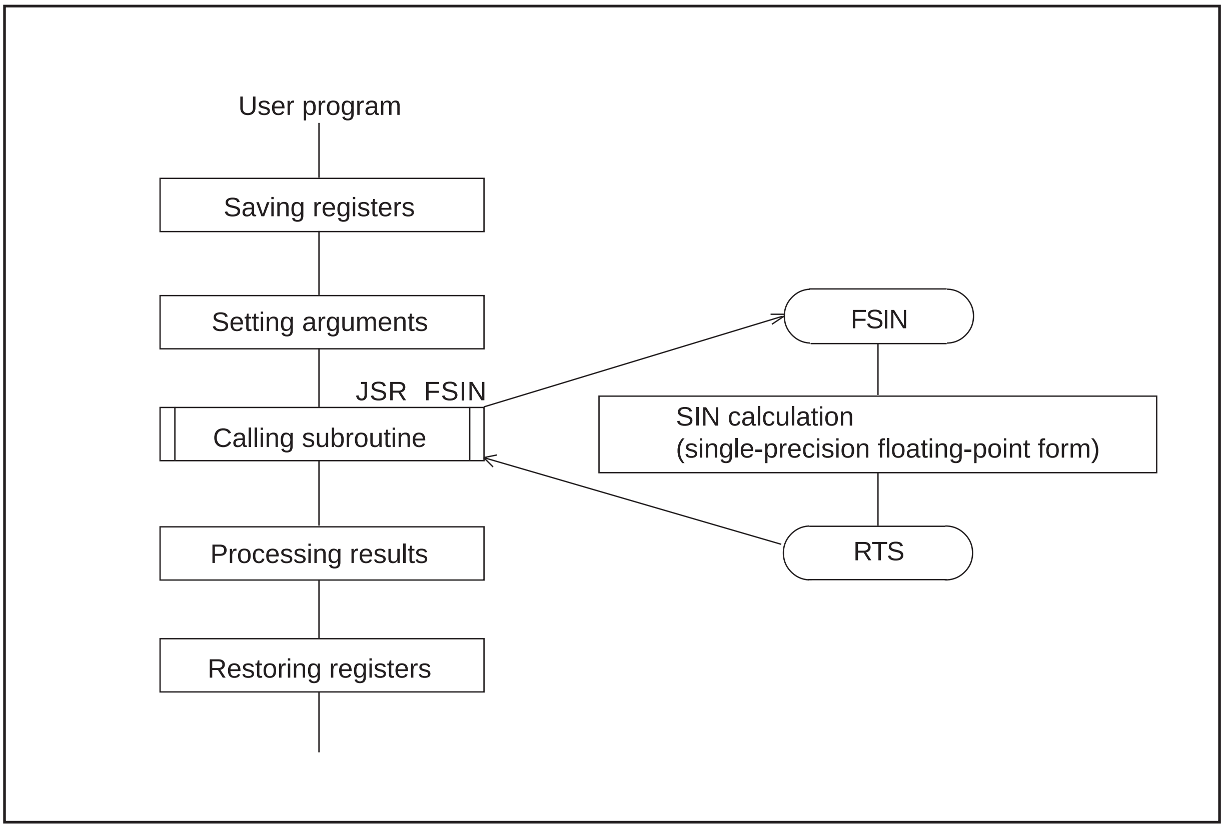
- Title
- Outline
- Storage places and contents of input arguments and output return values
- Stack amount used
- Notes

(4) Program section

Comments about the program are written on the right side of the program list.

1.2 Guide to Using Programs

This manual contains programs in subroutine form and those in routine form. (Refer to Chapter 2, "Function List".) Use the programs in subroutine form by calling them from your application program following the procedure shown below. Use the programs in routine form after incorporating them into your application program.



Procedure for calling a subroutine

```
User program
        :
    MOV.W    WORK1[SB],R0     ........ Sets arguments.
    MOV.W    WORK2[SB],R2

    JSR      FSIN             ........ Calls the program as subroutine.

    JC       SIN_ERR          ........ Result error?  (When in error, jumps to
                                       SIN_ERR.)
    Processing result
        :
SIN_ERR:
    Processing error
```

Example of a subroutine call

MEMO

Chapter 2

Collection of General-purpuse Programs

Function List

2.1 Clearing RAM

2.1.1 Outline

This program initializes memory by using a block constant setup instruction (SSTR).

Subroutine name : —	ROM capacity : 11 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	" 0000_{16} "	Transfer data
R1	—	—	Unused
R2	—	—	Unused
R3	—	" 0000_{16} "	Number of transfers performed
A0	—	—	Unused
A1	—	Last address at destination	Destination address
Specified area	—	Transfer data	←

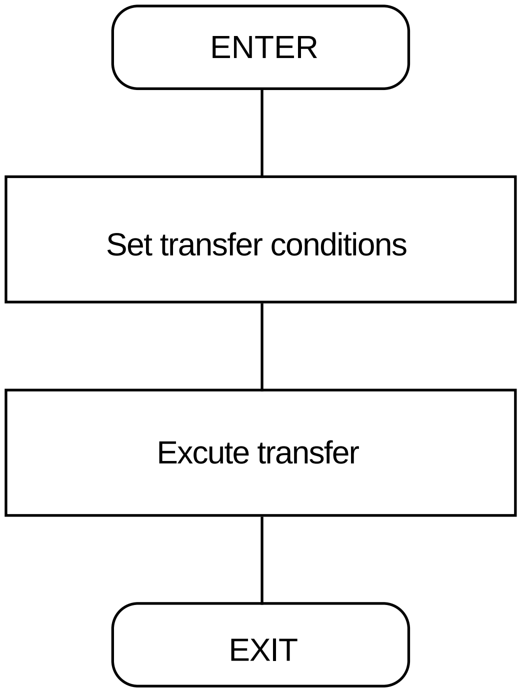
Usage precautions

Memory is initialized in units of words.

2.1.2 Explanation

This program stores 0s in memory in units of words by using a block constant setup instruction (SSTR). The program sets the transfer data (0H) in R0, the number of transfers performed (half the number of bytes of the area to be initialized) in R3, and the start address at destination in A1 before executing the SSTR instruction.

2.1.3 Flowchart



2.1.4 Program List

```
;*************************************************************************************
;                                                                                    *
;          M16C Program Collection No. 1                                             *
;          CPU                  : M16C                                               *
;                                                                                    *
;*************************************************************************************
VramTOP                 .EQU     000400H                    ; Declares start address of RAM
VramEND                 .EQU     002C00H                    ; Declares end address of RAM
VromTOP                 .EQU     0F0000H                    ; Declares start address of ROM
;                                                           ;
;==================================================================
;          Title: Clearing RAM
;          Outline: Clears RAM using block constant setup instruction
;          Input:    ---------------------------->    Output:
;          R0        ( )                              R0        (Transfer data)
;          R1L       ( )                              R1L       (Unused)
;          R1H       ( )                              R1H       (Unused)
;          R2        ( )                              R2        (Unused)
;          R3        ( )                              R3        (Indeterminate)
;          A0        ( )                              A0        (Unused)
;          A1        ( )                              A1        (Indeterminate)
;          Stack amount used: None
;          Notes:
;==================================================================
                        .SECTION         PROGRAM,CODE
                        .ORG             VromTOP            ; ROM area
          MOV.W         #0,R0                               ; Sets transfer data
          MOV.W         #((VramEND+1)-VramTOP)/2,R3         ; Sets number of transfers performed
          MOV.W         #VramTOP,A1                         ; Sets destination address
          SSTR.W                                            ; Executes clearing of RAM
;                                                           ;
                        .END                                ;
```

2.2 Transferring Blocks

2.2.1 Outline

This program transfers memory contents from one location to another by using a block transfer instruction (SMOVF).

Subroutine name : —	ROM capacity : 14 bytes
Interrupt during execution: Accepted	Number of stacks used : None

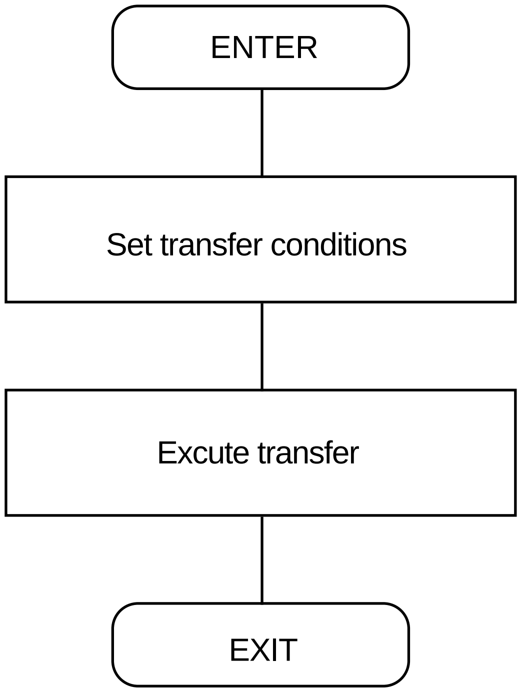
Register/memory	Input	Output	Usage condition
R0	—	—	Unused
R1H	—	High-order 4 bits of last source address	High-order half of source address
R1L	—	—	Unused
R2	—	—	Unused
R3	—	" 0000_{16} "	Number of transfers performed
A0	—	Low-order 16 bits of last source address	Low-order half of source address
A1	—	Last address at destination	Destination address
BLOCK1	Content of BLOCK1	Does not change	←
BLOCK2	Content of BLOCK2	Content of BLOCK1	←
Usage precautions			

2.2.2 Explanation

This program transfers memory contents from one location to another by using a block transfer instruction (SMOVF).

The program sets the number of transfers performed in R3, the high-order 4 bits of the source's start address in R1H, the low-order 16 bits of the source's start address in A0, and the destinations's start address in A1 before executing the SMOVF instruction.

2.2.3 Flowchart



2.2.4 Program List

```
;*******************************************************************************************
;                                                                                         *
;          M16C Program Collection No. 2                                                  *
;          CPU                  : M16C                                                    *
;                                                                                         *
;*******************************************************************************************
VramTOP                  .EQU        000400H          ; Declares start address of RAM
VromTOP                  .EQU        0F0000H          ; Declares start address of ROM
;                                                     ;
                         .SECTION    RAM,DATA
                         .ORG        VramTOP          ; RAM area
LENGTH                   .EQU        10               ; Length of area
BLOCK1:                  .BLKB       LENGTH           ; Source area of transfer
BLOCK2:                  .BLKB       LENGTH           ; Destination area of transfer
;                                                     ;
;===================================================================
;          Title: Transferring blocks
;          Outline: Example for using block transfer instruction
;          Input:     ----------------------------->  Output:
;          R0L        ( )                     R0L       (Unused)
;          R0H        ( )                     R0H       (Unused)
;          R1L        ( )                     R1L       (Unused)
;          R1H        ( )                     R1H       (Indeterminate)
;          R2         ( )                     R2        (Unused)
;          R3         ( )                     R3        (Indeterminate)
;          A0         ( )                     A0        (Indeterminate)
;          A1         ( )                     A1        (Indeterminate)
;          Stack amount used: None
;          Notes:
;===================================================================
                         .SECTION    PROGRAM,CODE
                         .ORG        VromTOP          ; ROM area
          MOV.W          #LENGTH,R3                   ; Sets number of transfers performed
          MOV.W          #BLOCK1 & 0FFFFH,A0          ; Sets low-order half of the source address
          MOV.B          #BLOCK1>>16,R1H              ; Sets high-order half of the source address
          MOV.W          #BLOCK2,A1                   ; Sets destination address
          SMOVF.B                                     ; Executes transfer of blocks
;                                                     ;
                         .END                         ;
```

2.3 Changing Blocks

2.3.1 Outline

This program changes memory contents consisting of the same number of bytes with each other memory location.

Subroutine name : —	ROM capacity : 15 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0L	—	Last data of BLOCK2	Register used for change
R0H	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	" 000_{16} "	Number of transfers performed
A1	—	—	Unused
BLOCK1	Content of BLOCK1	Content of BLOCK2	←
BLOCK2	Content of BLOCK2	Content of BLOCK1	←

Usage precautions

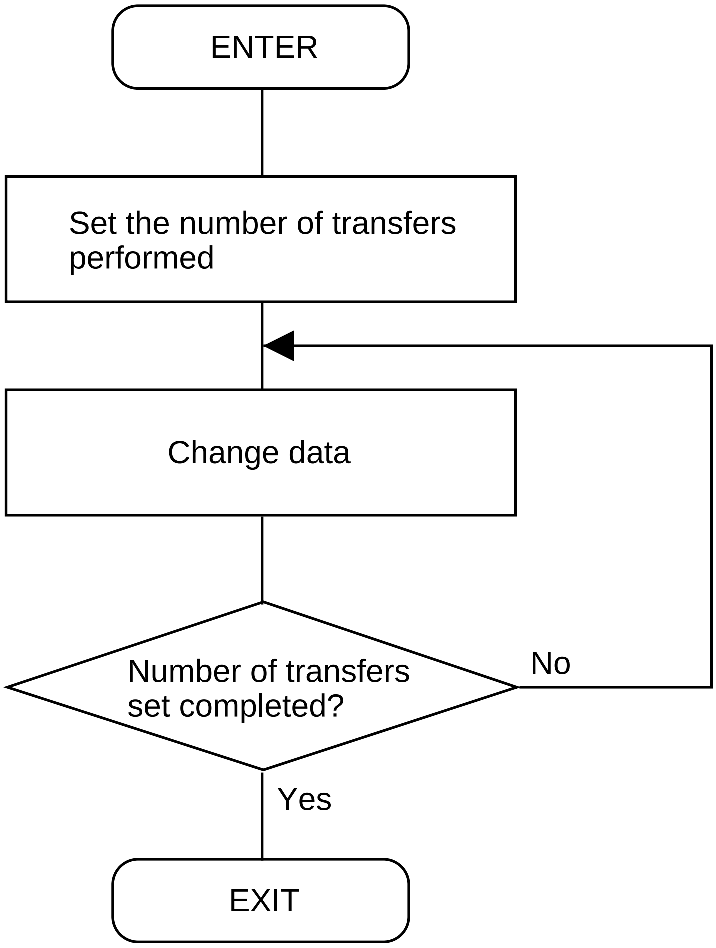
Memory contents are changed in bytes.

2.3.2 Explanation

This program changes memory contents consisting of the same number of bytes with each other memory location. An add and conditional branch instruction (ADJNZ) is used to count the number of transfers performed.

In this program, memory contents basically are changed in bytes. However, if the memory contents to be changed consist of even bytes, they can be changed in words for increased speed of processing.

2.3.3 Flowchart



2.3.4 Program List

```
;*******************************************************************************************
;                                                                                          *
;           M16C Program Collection No.3                                                   *
;           CPU                 : M16C                                                     *
;                                                                                          *
;*******************************************************************************************
VramTOP                 .EQU            000400H         ; Declares start address of RAM
VromTOP                 .EQU            0F0000H         ; Declares start address of ROM
;                                                       ;
                        .SECTION        RAM,DATA
                        .ORG            VramTOP         ; RAM area
LENGTH                  .EQU            10              ; Length of area
BLOCK1:                 .BLKB           LENGTH          ; Area 1
BLOCK2:                 .BLKB           LENGTH          ; Area 2
;                                                       ;
;============================================================================
;           Title: Changing blocks
;           Outline: Changes data in units of blocks.
;           Input:      ----------------------------->  Output:
;           R0L         ( )                             R0L     (Indeterminate)
;           R0H         ( )                             R0H     (Unused)
;           R1L         ( )                             R1L     (Unused)
;           R1H         ( )                             R1H     (Unused)
;           R2          ( )                             R2      (Unused)
;           R3          ( )                             R3      (Unused)
;           A0          ( )                             A0      (Indeterminate)
;           A1          ( )                             A1      (Unused)
;           Stack amount used: None
;           Notes:
;============================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
            MOV.B       #LENGTH,A0                      ; Sets number of transfers performed
LOOP:                                                   ;
            MOV.B       BLOCK1-1[A0],R0L                ;
            XCHG.B      R0L,BLOCK2-1[A0]                ; Changes data
            MOV.B       R0L,BLOCK1-1[A0]                ;
            ADJNZ.W     #-1,A0,LOOP                     ; --> Looped for the number of transfers performed
;                                                       ;
                        .END                            ;
```

2.4 Indirect Subroutine Call

2.4.1 Outline

This program executes an indirect subroutine call instruction after setting the relative jump address for indirect jump. It also executes an indirect subroutine call instruction by using a 20-bit absolute address.

(1) Indirect subroutine call (relative)

Subroutine name : SUBIND_W	ROM capacity : 19 bytes
Interrupt during execution: Accepted	Number of stacks used : 3 bytes

Register/memory	Input	Output	Usage condition
R0	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	Indeterminate	Processing status
A1	—	Indeterminate	Processing relative address
MODE	Current processing status	Next processing status	←

Usage precautions

The indirect jump address set here is a relative address.

(2) Indirect subroutine call (absolute)

Subroutine name : SUBIND_A	ROM capacity : 26 bytes
Interrupt during execution: Accepted	Number of stacks used : 3 bytes

Register/memory	Input	Output	Usage condition
R0	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	Indeterminate	Address pointer
A1	—	—	Unused
MODE	Current processing status	Next processing status	←

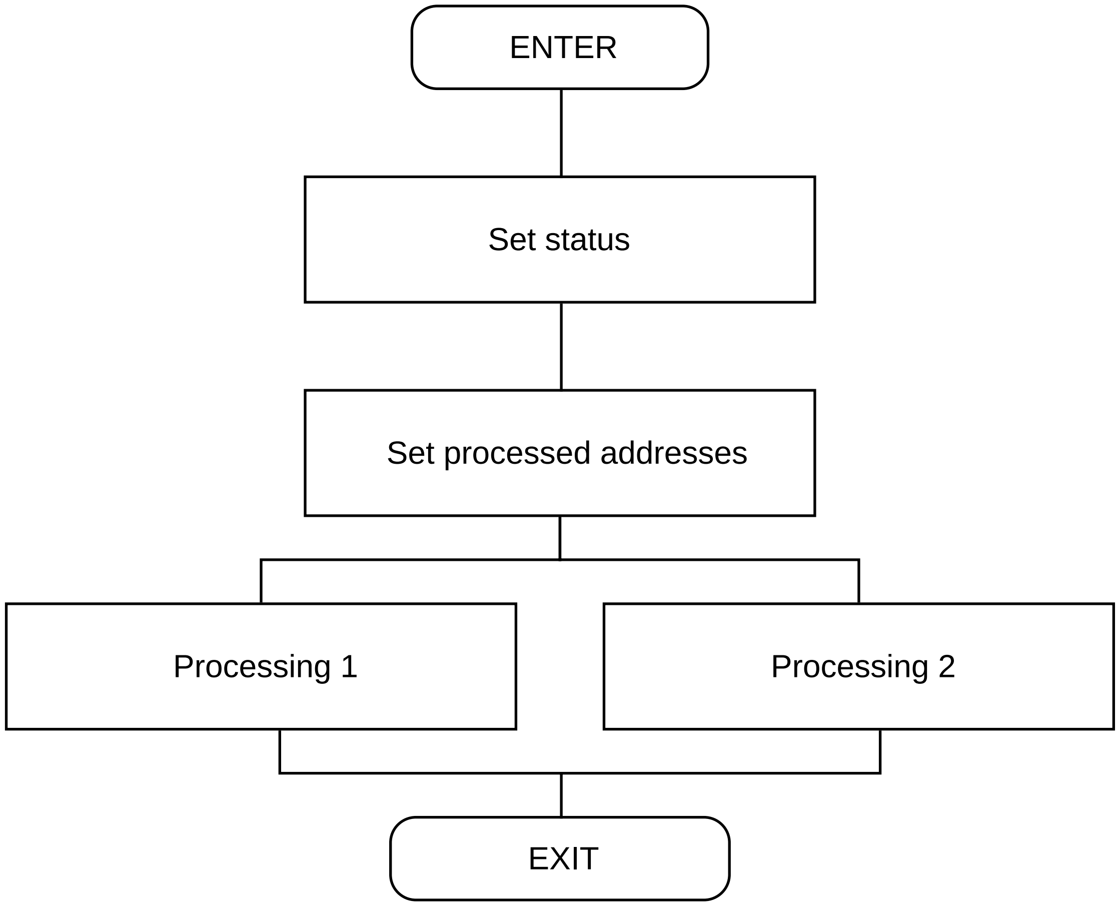
Usage precautions

The indirect jump address set here is a 20-bit absolute address.

2.4.2 Explanation

For indirect jump based on relative addresses, this program uses an extended access instruction (LDE) to set the relative jump address for the indirect jump. In this program, since relative addresses are within the range that can be represented with 8 bits, ".B (byte size)" is used to set the offset data.
For indirect jump based on absolute addresses, this program adds the content of the address register, with its sign ignored, to the start address of the memory area where 20-bit absolute addresses are stored and jumps to the memory location (20-bit absolute address) indicated by the result. The memory area in which to store 20-bit absolute addresses is allocated in units of 3 bytes.

2.4.3 Flowchart



2.4.4 Program List

```
;*******************************************************************************
;           M16C Program Collection No. 4                                     *
;           CPU                 : M16C                                        *
;*******************************************************************************
VramTOP                 .EQU            000400H         ; Declares start address of RAM
VromTOP                 .EQU            0F0000H         ; Declares start address of ROM
Vsb                     .EQU            0400H           ; Sets SB
                        .SECTION        RAM,DATA
                        .ORG            VramTOP         ; RAM area
MODE:                   .BLKB           1               ; Processing status
MD_0                    .EQU            0               ; Status No. 0
MD_1                    .EQU            1               ; Status No. 1
;===============================================================================
;           Title:      Indirect subroutine call
;           Outline:    Branches processing using an indirect subroutine call (relative)
;           Input:      ----------------------------->  Output:
;           R0          ( )                             R0          (Unused)
;           R1          ( )                             R1          (Unused)
;           R2          ( )                             R2          (Unused)
;           R3          ( )                             R3          (Unused)
;           A0          ( )                             A0          (Indeterminate)
;           A1          ( )                             A1          (Indeterminate)
;           Stack amount used: 3 bytes
;===============================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
                        .SB             Vsb             ; Declares SB register value
                        .SBSYM          MODE            ;
SUBIND_W:                                               ;
            MOV.B       MODE,A0                         ;
            LDE.B       JUMPaddress[A0],A1              ; Sets jump address
JUMP_offset:                                            ;
            JSRI.W      A1                              ; Jumps to each processing
            RTS                                         ;
MODE_0:                                                 ;
            MOV.B       #MD_1,MODE                      ;
            RTS                                         ;
MODE_1:                                                 ;
            MOV.B       #MD_0,MODE                      ;
            RTS                                         ;
JUMPaddress:                                            ;
                        .BYTE   MODE_0-JUMP_offset      ;
                        .BYTE   MODE_1-JUMP_offset      ;
;===============================================================================
;           Title:      Indirect subroutine call
;           Outline:    Branches processing using an indirect subroutine call (absolute).
;           Input:      ----------------------------->  Output:
;           R0          ( )                             R0          (Unused)
;           R1          ( )                             R1          (Unused)
;           R2          ( )                             R2          (Unused)
;           R3          ( )                             R3          (Unused)
;           A0          ( )                             A0          (Indeterminate)
;           A1          ( )                             A1          (Unused)
;           Stack amount used: 3 bytes
;===============================================================================
SUBIND_A:
            MOV.B       MODE,A0                         ;
            SHL.W       #1,A0                           ;
            ADD.B       MODE,A0                         ; Sets jump pointer
            JSRI.A      JSRaddress[A0]                  ; Jumps to each processing
            RTS                                         ;
JSR_0:                                                  ;
            MOV.B       #MD_1,MODE                      ;
            RTS                                         ;
JSR_1:                                                  ;
            MOV.B       #MD_0,MODE                      ;
            RTS                                         ;
JSRaddress:                                             ;
                        .ADDR   JSR_0                   ;
                        .ADDR   JSR_1                   ;
                        .END                            ;
```

2.5 Compressing BCD

2.5.1 Outline

This program converts 2-digit unpacked BCD data into 1-digit packed BCD.

Subroutine name : —	ROM capacity : 8 bytes
Interrupt during execution: Accepted	Number of stacks used : None

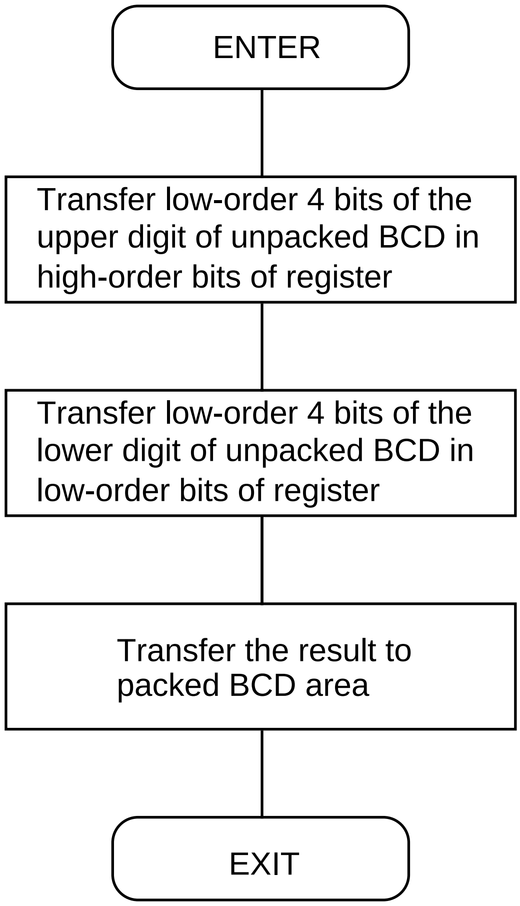
Register/memory	Input	Output	Usage condition
R0L	—	Packed BCD	Used to create data
R1H	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused
UNPACK_BCDhi	Upper half of unpacked BCD	Does not change	←
UNPACK_BCDlow	Lower half of unpacked BCD	Does not change	←
PACK_BCD	—	Packed BCD	←
Usage precautions			

2.5.2 Explanation

This program converts 2-digit unpacked BCD data into 1-digit packed BCD. Set the 2-digit unpacked BCD data in a variable area (UNPACK_BCDhi, UNPACK_BCDlow). When the program is executed, 1-digit packed BCD data is output to a variable area (PACK_BCD).

The program transfers the low-order 4 bits of the upper digit and the low-order 4 bits of the lower digit of the unpacked BCD in the high-order and the low-order bits of a data creation register by using a 4-bit manipulating instruction as it creates packed BCD.

2.5.3 Flowchart



2.5.4 Program List

```
;*******************************************************************************************
;                                                                                          *
;           M16C Program Collection No. 5                                                  *
;           CPU                 : M16C                                                     *
;                                                                                          *
;*******************************************************************************************
VramTOP             .EQU        000400H         ;  Declares start address of RAM
VromTOP             .EQU        0F0000H         ;  Declares start address of ROM
Vsb                 .EQU        0400H           ;  Sets SB
;                                               ;
                    .SECTION    RAM,DATA
                    .ORG        VramTOP         ; RAM area
UNPACK_BCDhi:       .BLKB       1               ; Upper digit of unpacked BCD
UNPACK_BCDlow:      .BLKB       1               ; Lower digit of unpacked BCD
PACK_BCD:           .BLKB       1               ; Packed BCD
;                                               ;
;=============================================================
;           Title: Compressing BCD
;           Outline: Converts 2-digit unpacked BCD to 1-digit packed BCD.
;           Input:      ----------------------------->  Output:
;           R0L         ( )                     R0L     (Packed BCD)
;           R0H         ( )                     R0H     (Unused)
;           R1L         ( )                     R1L     (Unused)
;           R1H         ( )                     R1H     (Unused)
;           R2          ( )                     R2      (Unused)
;           R3          ( )                     R3      (Unused)
;           A0          ( )                     A0      (Unused)
;           A1          ( )                     A1      (Unused)
;           Stack amount used: None
;           Notes:
;=============================================================
                    .SECTION    PROGRAM,CODE
                    .ORG        VromTOP         ; ROM area
                    .SB         Vsb             ; Declares SB register value
                    .SBSYM      UNPACK_BCDhi    ;
                    .SBSYM      UNPACK_BCDlow   ;
                    .SBSYM      PACK_BCD        ;
          MOVLH     UNPACK_BCDhi,R0L            ;
          MOVLL     UNPACK_BCDlow,R0L           ;
          MOV.B     R0L,PACK_BCD                ;
;                                               ;
                    .END                        ;
```

2.6 Calculating Sum-of-Products

2.6.1 Outline

This program calculates a sum of products using a sum-of-products calculating instruction (RMPA).

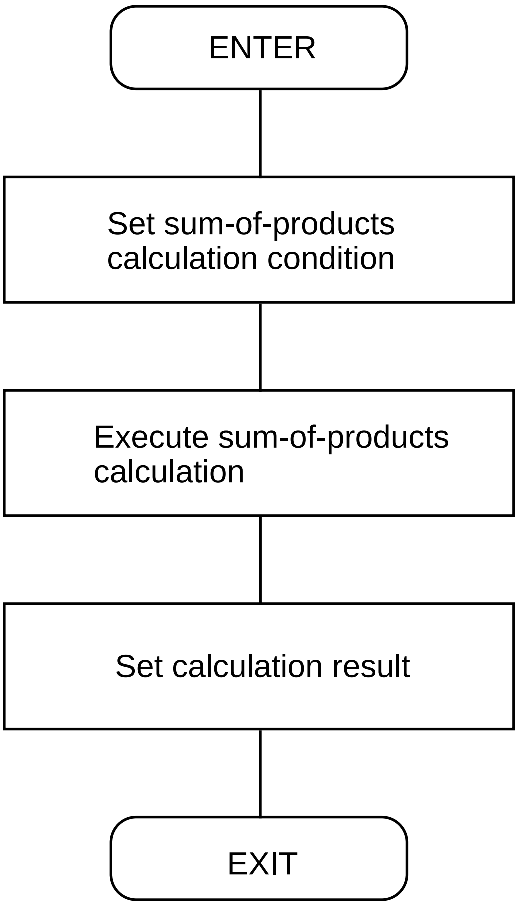
Subroutine name : —	ROM capacity : 15 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	Result of sum-of-products calculation	Used for calculation
R1	—	—	Unused
R2	—	—	Unused
R3	—	" 000_{16} "	Number of some-of-products
A0	—	Last address of multiplicand	Multiplicand address
A1	—	Last address of multiplier	Multiplier address
DATA11 to 13	Multiplicand	Does not change	←
DATA21 to 23	Multiplier	Does not change	←
ANS	—	Result of sum-of-products calculation	←
Usage precautions			

2.6.2 Explanation

This program calculates a sum of products using a sum-of-products calculating instruction (RMPA). Set the multiplier in a variable area (DATA11-13) and the multiplicand in a variable area (DATA21-23). The result of sum-of-products calculation is output to a variable area (ANS).
The program sets the number of sum-of-products in R3, the multiplicand address in A0, and the multiplier address in A1 before executing the RMPA instruction.

2.6.3 Flowchart



2.6.4 Program List

```
;*********************************************************************************************
;                                                                                            *
;           M16C Program Collection No. 6                                                    *
;           CPU                 : M16C                                                       *
;                                                                                            *
;*********************************************************************************************
VramTOP              .EQU          000400H        ; Declares start address of RAM
VromTOP              .EQU          0F0000H        ; Declares start address of ROM
Vsb                  .EQU          0400H          ; Sets SB
;                                                 ;
                     .SECTION      RAM,DATA
                     .ORG          VramTOP        ; RAM area
DATA11:              .BLKB         1              ; Multiplicand 1
DATA12:              .BLKB         1              ; Multiplicand 2
DATA13:              .BLKB         1              ; Multiplicand 3
DATA21:              .BLKB         1              ; Multiplier 1
DATA22:              .BLKB         1              ; Multiplier 2
DATA23:              .BLKB         1              ; Multiplier 3
ANS:                 .BLKB         2              ; Result of sum-of-products calculation
;                                                 ;
;===============================================================
;           Title: Calculating sum-of-products
;           Outline: Calculates a sum of products.
;           Input:    ----------------------------->  Output:
;           R0        ( )                          R0       (Calculation result)
;           R1L       ( )                          R1L      (Unused)
;           R1H       ( )                          R1H      (Unused)
;           R2        ( )                          R2       (Unused)
;           R3        ( )                          R3       (Indeterminate)
;           A0        ( )                          A0       (Indeterminate)
;           A1        ( )                          A1       (Indeterminate)
;           Stack amount used: None
;           Notes:
;===============================================================
                     .SECTION      PROGRAM,CODE
                     .ORG          VromTOP        ; ROM area
                     .SB           Vsb            ; Declares SB register value
                     .SBSYM        ANS            ;
            MOV.W    #0,R0                        ; Initializes calculation area
            MOV.W    #3,R3                        ; Sets number of sum-of-products
            MOV.W    #DATA11,A0                   ; Multiplicand address
            MOV.W    #DATA21,A1                   ; Multiplier address
            RMPA.B                                ; Executes sum-of-products calculation
            MOV.W    R0,ANS                       ; Sets calculation result
;                                                 ;
                     .END                         ;
```

2.7 Processing Bits

2.7.1 Outline

This program processes bits.

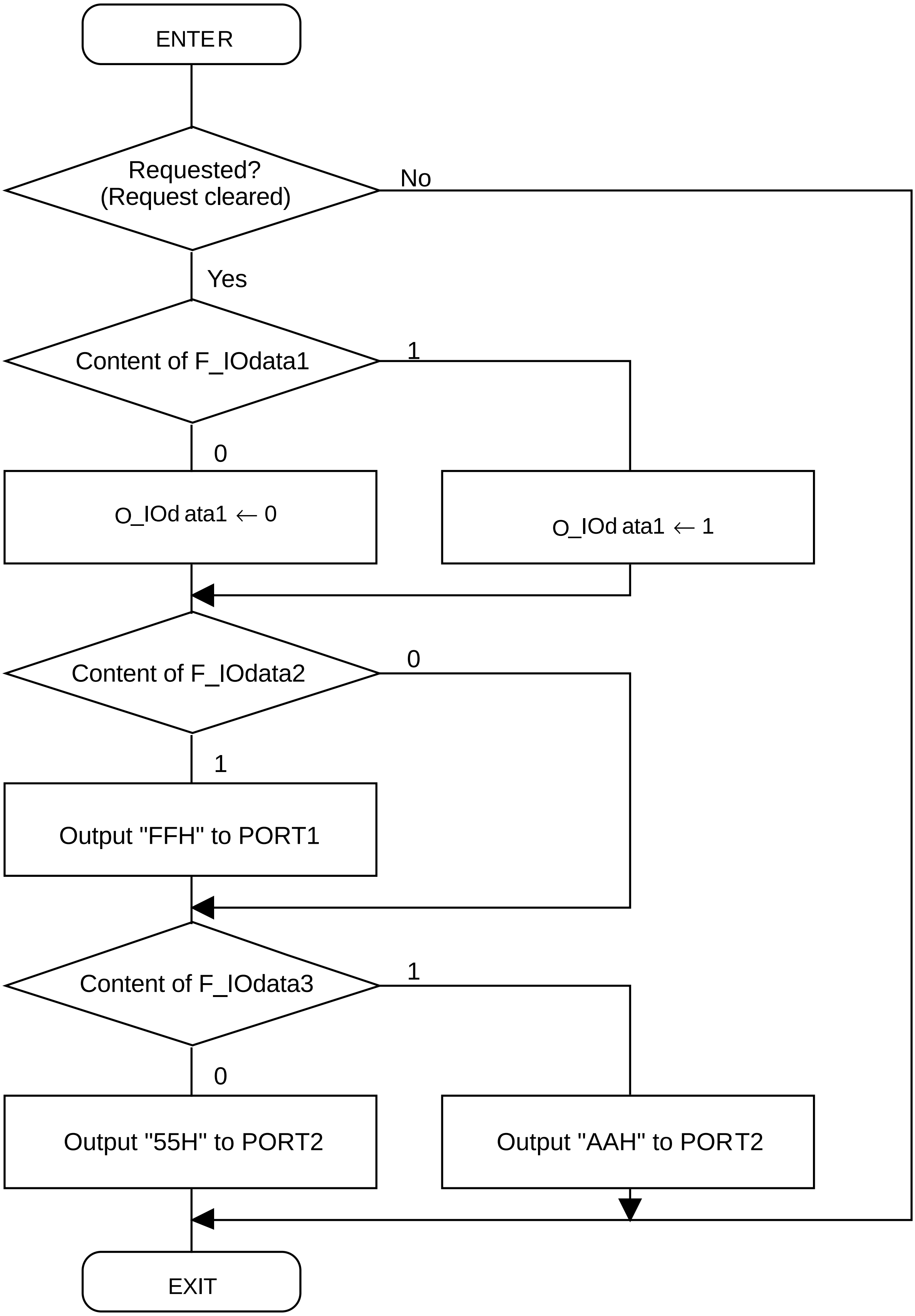
Subroutine name : —	ROM capacity : 32 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused
Usage precautions			

2.7.2 Explanation

This program uses bit processing instructions (BTSTC, BTST, BNTST) and condition store instructions (STZ, STZX) to perform its function. When it is executed, a value is output to PORT1, PORT2, or PORT3 that corresponds to the bit content of a variable area (FLAG1).

2.7.3 Flowchart



2.7.4 Program List

```
;*******************************************************************************************
;                                                                                         *
;          M16C Program Collection No. 7                                                  *
;          CPU                : M16C                                                      *
;                                                                                         *
;*******************************************************************************************
VramTOP               .EQU          000400H        ; Declares start address of RAM
VromTOP               .EQU          0F0000H        ; Declares start address of ROM
;                                                  ;
                                                   ; SFR area
PORT0                 .EQU          003E0H         ; PORT0
O_IOdata1             .BTEQU        0,PORT0        ;
PORT1                 .EQU          003E1H         ; PORT1
PORT2                 .EQU          003E4H         ; PORT2
;                                                  ;
                      .SECTION      RAM,DATA
                      .ORG          VramTOP        ; RAM area
FLAG1:                .BLKB         1              ;
F_REQ                 .BTEQU        0,FLAG1        ; Output request flag
F_IOdata1             .BTEQU        1,FLAG1        ;
F_IOdata2             .BTEQU        2,FLAG1        ;
F_IOdata3             .BTEQU        3,FLAG1        ;
;                                                  ;
;==============================================================================
;          Title: Setting bit after accepting event
;          Outline: Outputs memory content only when requested by other process
;          Input:     ----------------------------->  Output:
;          R0L        ( )                             R0L        (Unused)
;          R0H        ( )                             R0H        (Unused)
;          R1L        ( )                             R1L        (Unused)
;          R1H        ( )                             R1H        (Unused)
;          R2         ( )                             R2         (Unused)
;          R3         ( )                             R3         (Unused)
;          A0         ( )                             A0         (Unused)
;          A1         ( )                             A1         (Unused)
;          Stack amount used: None
;          Notes:
;==================================================
                      .SECTION      PROGRAM,CODE
                      .ORG          VromTOP        ; ROM area
          BTSTC       F_REQ                        ; Confirms and clears request
          JEQ         BITsetEXIT                   ; --> No request
;
          BTST        F_IOdata1                    ; Checks memory content
          BMC         O_IOdata1                    ; Outputs memory content
;
          BNTST       F_IOdata2                    ; Checks memory content
          STZ         #0FFH,PORT1                  ; Outputs "FF" if memory content = 1
;
          BTST        F_IOdata3                    ; Checks memory content
          STZX        #055H,#0AAH,PORT2            ; Outputs "55" : memory content = 0 , "AA" : memory content = 1
BITsetEXIT:
;                                                  ;
                      .END                         ;
```

2.8 Comparing 32 Bits

2.8.1 Outline

This program compares 32-bit data between registers.
This program compares 32-bit data between memory locations.

(1) 32-bit comparison (register)

Subroutine name : COMP32	ROM capacity : 7 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of comparing data	Does not change	←
R1	Lower half of compared data	Does not change	←
R2	Upper half of comparing data	Does not change	←
R3	Upper half of compared data	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
Z/C flag	—	Compared data	←
Usage precautions			

(2) 32-bit comparison (memory)

Subroutine name : COMPmemory32	ROM capacity : 9 bytes
Interrupt during execution:Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	Address of comparing data	Does not change	←
A1	Address of compared data	Does not change	←
Memory indicated by A0	Comparing	Does not change	←
Memory indicated by A1	Compared	Does not change	←
Z/C flag	—	Comparison result	←
Usage precautions			

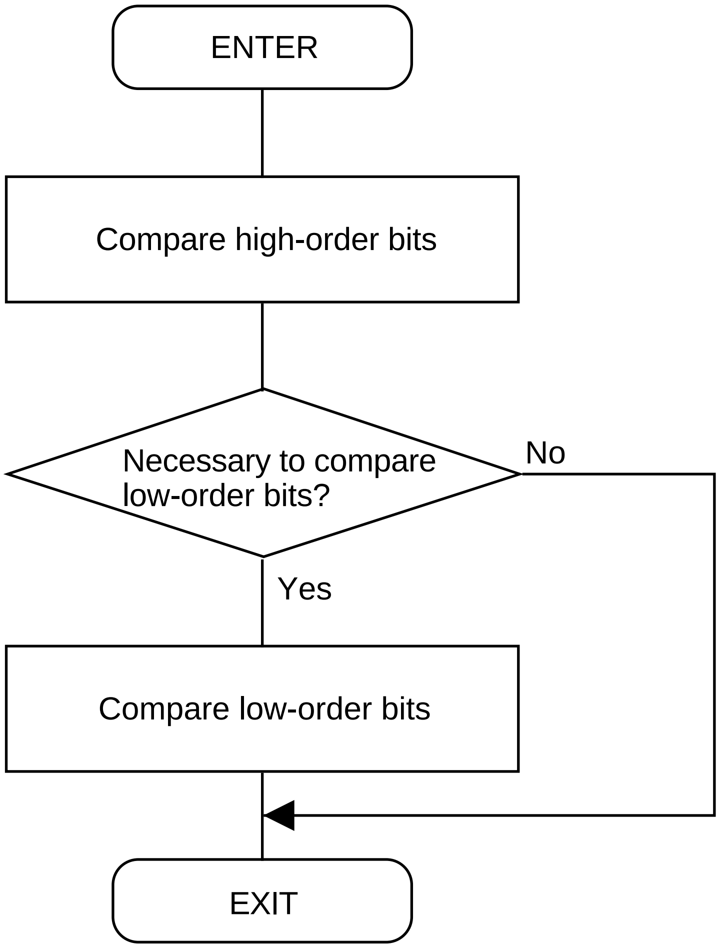
2.8.2 Explanation

This program compares 32-bit data between registers. Set the comparing data in R2 and R0 and the compared data in R3 and R1 beginning with the upper half, respectively. The comparison result is output to the Z and C flags.

This program compares 32-bit data between memory locations. Set the least significant memory address of the comparing data and that of the compared data in the address registers. The comparison result is output to the Z and C flags.

C	Z	Meaning
1	0	Comparing data < compared data
1	1	Comparing data = compared data
0	0	Comparing data > compared data

2.8.3 Flowchart



2.8.4 Program List

```
;*****************************************************************************************
;                                                                                       *
;           M16C Program Collection No. 8                                               *
;           CPU                 : M16C                                                  *
;                                                                                       *
;*****************************************************************************************
VromTOP                         .EQU     0F0000H          ; Declares start address of ROM
;                                                         ;
;================================================================
;           Title: Comparing 32 bits
;           Outline: Compares 32-bit data between registers.
;           Input:     ----------------------------->   Output:
;           R0   (Lower half of comparing data)        R0   (Does not change)
;           R1   (Lower half of compared data)         R1   (Does not change)
;           R2   (Upper half of comparing data)        R2   (Does not change)
;           R3   (Upper half of compared data)         R3   (Does not change)
;           A0   ( )                                   A0   (Unused)
;           A1   ( )                                   A1   (Unused)
;           Stack amount used: None
;           Notes: Result is returned by Z and C flags.
;================================================================
            .SECTION        PROGRAM,CODE
            .ORG            VromTOP                   ; ROM area
COMP32:                                               ;
            CMP.W           R2,R3                     ; Compares high-order bits
            JNE             COMP32exit                ; --> Result is output after comparing only high-order bits
            CMP.W           R0,R1                     ; Compares low-order bits
COMP32exit:                                           ;
            RTS                                       ;
;                                                     ;
;================================================================
;           Title: Comparing 32 bits
;           Outline: Compares 32 bits between memory locations.
;           Input:     ----------------------------->   Output:
;           R0   ( )                                   R0   (Unused)
;           R1   ( )                                   R1   (Unused)
;           R2   ( )                                   R2   (Unused)
;           R3   ( )                                   R3   (Unused)
;           A0   (Address of comparing data)           A0   (Does not change)
;           A1   (Address of compared data)            A1   (Does not change)
;           Stack amount used: None
;           Notes: Result is returned by Z and C flags.
;================================================================
COMPmemory32:                                         ;
            CMP.W           2[A0],2[A1]               ; Compares high-order bits
            JNE             COMPmemory32exit          ; --> Result is output after comparing only high-order bits
            CMP.W           [A0],[A1]                 ; Compares low-order bits
COMPmemory32exit:                                     ;
            RTS                                       ;
;                                                     ;
                            .END                      ;
```

2.9 Adding 32 Bits

2.9.1 Outline

This program performs a 32-bit unsigned addition using registers.
This program performs a 32-bit unsigned addition between memory locations.

(1) 32-bit addition (register)

Subroutine name : ADDITION32	ROM capacity : 5 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of augend	Lower half of addition result	←
R1	Lower half of addend	Does not change	←
R2	Upper half of augend	Upper half of addition result	←
R3	Lower half of addend	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Carry information	←

Usage precautions

The augend is destroyed as a result of program execution.

(2) 32-bit addition (memory)

Subroutine name : ADDITIONmemory32	ROM capacity : 7 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	Augend address	Does not change	←
A1	Addend address	Does not change	←
Memory indicated by A0	Augend	Result of addition	←
Memory indicated by A1	Addend	Does not change	←
C flag	—	Carry information	←

Usage precautions

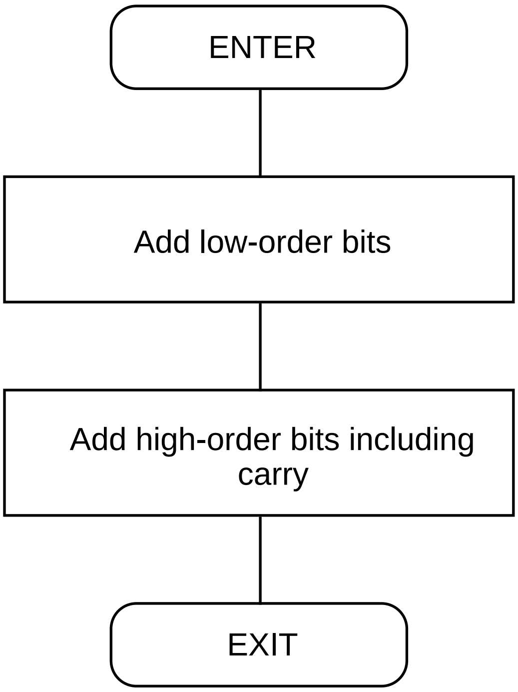
The augend is destroyed as a result of program execution.

2.9.2 Explanation

This program performs a 32-bit unsigned addition using registers. Set the augend in R2 and R0 and the addend in R3 and R1 beginning with the upper half, respectively. The addition result is output to R2 and R0 beginning with the upper half and carry information to the C flag, respectively.
This program performs a 32-bit unsigned addition between memory locations. Set the least significant memory address of the augend and that of the addend in the address registers. The addition result is output to the augend's memory location and carry information to the C flag, respectively.

C	Meaning
0	Without carry
1	With carry

2.9.3 Flowchart



2.9.4 Program List

```
;****************************************************************************************
;                                                                                      *
;          M16C Program Collection No. 9                                               *
;          CPU                   : M16C                                                *
;                                                                                      *
;****************************************************************************************
VromTOP                 .EQU     0F0000H                ; Declares start address of ROM
;                                                       ;
;=================================================================
;          Title: Adding 32 bits
;          Outline: Adds 32-bit data using registers.
;          Input:     ----------------------------->    Output:
;          R0   (Lower half of augend)          R0   (Lower half of addition result)
;          R1   (Lower half of addend)          R1   (Does not change)
;          R2   (Upper half of augend)          R2   (Upper half of addition result)
;          R3   (Upper half of addend)          R3   (Does not change)
;          A0   ( )                             A0   (Unused)
;          A1   ( )                             A1   (Unused)
;          Stack amount used: None
;          Notes:    Carry information in C flag
;                    R2R0 + R3R1
;=================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
ADDITION32:                                             ;
          ADD.W         R1,R0                           ; Adds low-order bits
          ADC.W         R3,R2                           ; Adds high-order bits
          RTS                                           ;
;                                                       ;
;=================================================================
;          Title: Adding 32 bits
;          Outline: Adds 32-bit data between memory locations
;          Input:     ----------------------------->    Output:
;          R0   ( )                   R0   (Unused)
;          R1   ( )                   R1   (Unused)
;          R2   ( )                   R2   (Unused)
;          R3   ( )                   R3   (Unused)
;          A0   (Augend address)      A0   (Does not change)
;          A1   (Addend address)      A1   (Does not change)
;          Stack amount used: None
;          Notes:    Carry information in C flag
;                    (A0) + (A1)
;=================================================================
ADDITIONmemory32:                                       ;
          ADD.W         [A1],[A0]                       ; Adds low-order bits
          ADC.W         2[A1],2[A0]                     ; Adds high-order bits
          RTS                                           ;
;                                                       ;
                        .END                            ;
```

2.10 Subtracting 32 Bits

2.10.1 Outline

This program performs a 32-bit unsigned subtraction using registers.
This program performs a 32-bit unsigned subtraction between memory locations.

(1) 32-bit subtraction (register)

Subroutine name : SUBTRACT32	ROM capacity : 5 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of minuend	Lower half of subtraction result	←
R1	Lower half of subtrahend	Does not change	←
R2	Upper half of minuend	Upper half of subtraction result	←
R3	Upper half of subtrahend	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Borrow information	←

Usage precautions

The minuend is destroyed as a result of program execution.

(2) 32-bit subtraction (memory)

Subroutine name : SUBTRACTmemory32	ROM capacity : 7 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	Minuend address	Does not change	←
A1	Subtrahend address	Does not change	←
Memory indicated by A0	Minuend	Subtraction result	←
Memory indicated by A1	Subtrahend	Does not change	←
C flag	—	Borrow information	←

Usage precautions

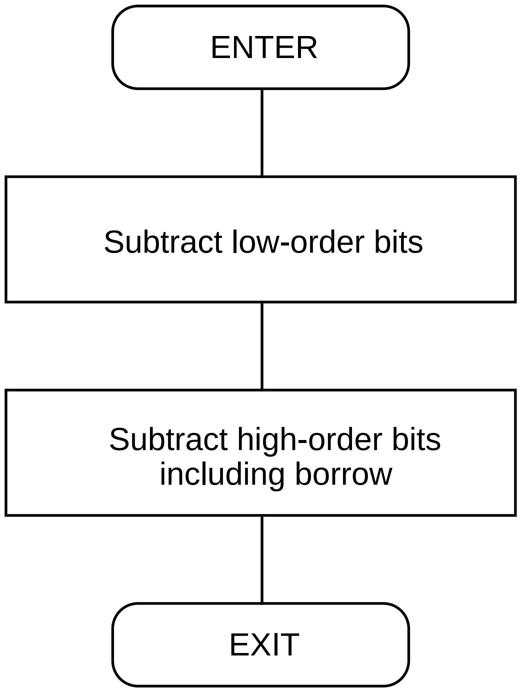
The minuend is destroyed as a result of program execution.

2.10.2 Explanation

This program performs a 32-bit unsigned subtraction using registers. Set the minuend in R2 and R0 and the subtrahend in R3 and R1 beginning with the upper half, respectively. The subtraction result is output to R2 and R0 beginning with the upper half and borrow information to the C flag, respectively.
This program performs a 32-bit unsigned subtraction between memory locations. Set the least significant memory address of the minuend and that of the subtrahend in the address registers. The subtraction result is output to the minuend's memory location and borrow information to the C flag, respectively.

C	Meaning
0	With borrow
1	Without borrow

2.10.3 Flowchart



2.10.4 Program List

```
;*****************************************************************************************
;                                                                                       *
;          M16C Program Collection No. 10                                               *
;          CPU                : M16C                                                    *
;                                                                                       *
;*****************************************************************************************
VromTOP               .EQU          0F0000H        ; Declares start address of ROM
;                                                  ;
;================================================================
;          Title: Subtracting 32 bits
;          Outline: Subtracts 32-bit data using registers.
;          Input:     ----------------------------->   Output:
;          R0   (Lower half of minuend)          R0   (Lower half of subtraction result)
;          R1   (Lower half of subtrahend)       R1   (Does not change)
;          R2   (Upper half of minuend)          R2   (Upper half of addition result)
;          R3   (Upper half of subtrahend)       R3   (Does not change)
;          A0   ( )                              A0   (Unused)
;          A1   ( )                              A1   (Unused)
;          Stack amount used: None
;          Notes:   Borrow information in C flag
;                   R2R0 - R3R1
;================================================================
                      .SECTION      PROGRAM,CODE
                      .ORG          VromTOP        ; ROM area
SUBTRUCT32:                                        ;
          SUB.W       R1,R0                        ; Subtracts low-order bits
          SUB.W       R3,R2                        ; Subtracts high-order bits
          RTS                                      ;
;                                                  ;
;================================================================
;          Title: Subtracting 32 bits
;          Outline: Subtracts 32-bit data between memory locations
;          Input:     ----------------------------->   Output:
;          R0   ( )                              R0   (Unused)
;          R1   ( )                              R1   (Unused)
;          R2   ( )                              R2   (Unused)
;          R3   ( )                              R3   (Unused)
;          A0   (Minuend address)                A0   (Does not change)
;          A1   (Subtrahend address)A1    (Does not change)
;          Stack amount used: None
;          Notes: Borrow information in C flag
;                   (A0) - (A1)
;================================================================
SUBTRACTmemory32:                                  ;
          SUB.W       [A1],[A0]                    ; Subtracts low-order bits
          SUB.W       2[A1],2[A0]                  ; Subtracts high-order bits
          RTS                                      ;
;                                                  ;
                      .END                         ;
```

2.11 Multiplying 32 Bits

2.11.1 Outline

This program performs a 32-bit unsigned multiplication using registers.

Subroutine name : MULTIPLE32	ROM capacity : 37 bytes
Interrupt during execution: Accepted	Number of stacks used : 6 bytes

Register/memory	Input	Output	Usage condition
R0	Lower half of multiplicand	Lower part of multiplication result	←
R1	Lower half of multiplier	Upper part of multiplication result	←
R2	Upper half of multiplicand	Middle part of multiplication result	←
R3	Upper half of multiplier	Most significant part of multiplication result	←
A0	—	Indeterminate	Used for storing data
A1	—	Indeterminate	Used for storing data

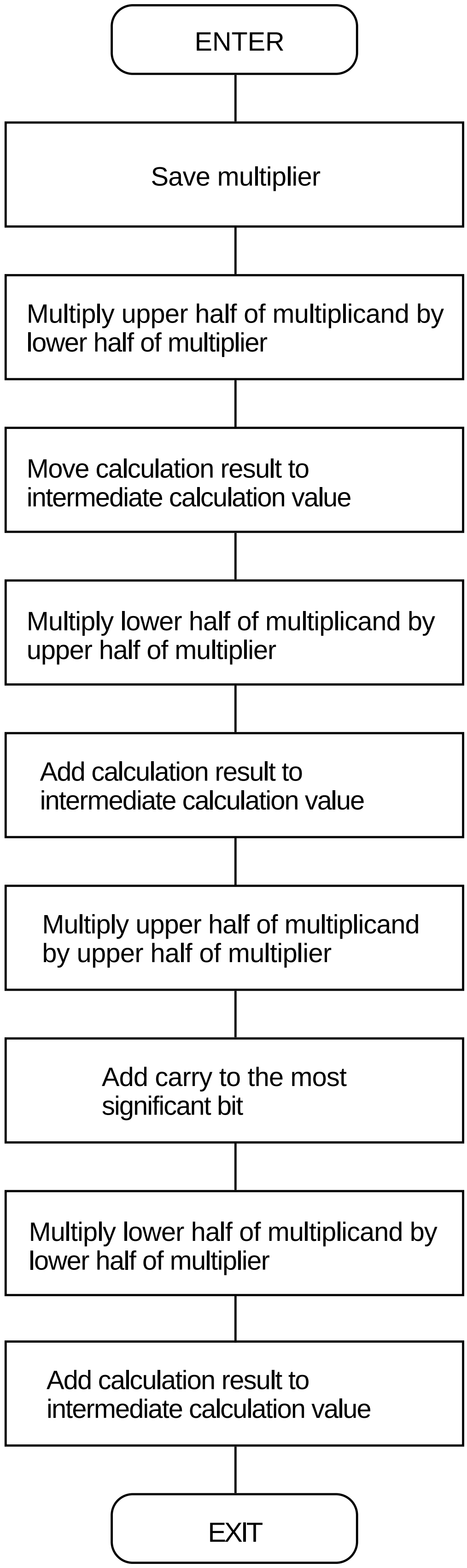
Usage precautions

The multiplication result is output to R3, R1, R2, and R0 beginning with its most significant part.
Both multiplier and multiplicand are destroyed as a result of program execution.

2.11.2 Explanation

This program performs a 32-bit unsigned multiplication using registers. Set the multiplicand in R2 and R0 beginning with the upper half and the multiplier in R3 and R1, respectively. The multiplication result is output to R3, R1, R2, and R0 beginning with its most significant part.
In this program, both multiplier and multiplicand are divided into the upper and lower halves (16 bits each) as they are multiplied. The results are added to produce a 64-bit calculation result.

2.11.3 Flowchart



2.11.4 Program List

```
;*******************************************************************************************
;                                                                                         *
;          M16C Program Collection No. 11                                                 *
;          CPU                : M16C                                                      *
;                                                                                         *
;*******************************************************************************************
VromTOP         .EQU    0F0000H                     ; Declares start address of ROM
;                                                   ;
;==================================================================
;          Title: Multiplying 32 bits
;          Outline: Multiplies 32-bit data together using registers
;          Input:     ----------------------------->    Output:
;          R0   (Lower half of multiplicand)      R0   (Lower part of multiplication result)
;          R1   (Lower half of multiplier)        R1   (Upper part of multiplication result)
;          R2   (Upper half of multiplicand)      R2   (Middle part of multiplication result)
;          R3   (Upper half of multiplier)        R3   (Most significant part of multiplication result)
;          A0   ( )                               A0   (Indeterminate)
;          A1   ( )                               A1   (Indeterminate)
;          Stack amount used: 6 bytes
;          Notes:    R2R0 X R3R1
;                    Calculation result is output in order of R3, R1, R2, and R0 beginning with the most
                     significant bits.
;==================================================================
                .SECTION        PROGRAM,CODE
                .org            VromTOP             ; ROM area
MULTIPLE32:                                         ;
        PUSH.W          R1                          ; Saves lower half of multiplier
        PUSH.W          R3                          ; Saves upper half of multiplier
        PUSH.W          R3                          ; Saves upper half of multiplier
        MULU.W          R2,R1                       ; Multiplies upper half of multiplicand by lower half of multiplier
        MOV.W           R3,A1                       ; Saves calculation result
        MOV.W           R1,A0                       ;
        POP.W           R1                          ; Restores upper half of multiplier
        MULU.W          R0,R1                       ; Multiplies lower half of multiplicand by upper half of multiplier
        ADD.W           R1,A0                       ; Adds to intermediate calculation value and saves result
        ADC.W           R3,A1                       ; Holds carry until next addition is made
        POP.W           R1                          ; Restores upper half of multiplier
        MULU.W          R2,R1                       ; Multiplies upper half of multiplicand by upper half of multiplier
        ADCF.W          R3                          ; Adds carry to the most significant bit
        POP.W           R2                          ; Restores lower half of multiplier
        MULU.W          R2,R0                       ; Multiplies lower half of multiplicand by lower half of multiplier
        ADD.W           A0,R2                       ; Adds intermediate value to middle part
        ADC.W           A1,R1                       ; Adds intermediate value to upper part
        ADCF.W          R3                          ; Adds carry to the most significant bit
        RTS                                         ;
;                                                   ;
                        .END                        ;
```

2.12 Dividing 32 Bits

2.12.1 Outline

This program performs a 32-bit unsigned division using registers.

Subroutine name : DIVIDE32	ROM capacity : 48 bytes
Interrupt during execution: Accepted	Number of stacks used : 3 bytes

Register/memory	Input	Output	Usage condition
R0	Lower half of dividend	Lower half of quotient	←
R1	Lower half of divisor	Does not change	←
R2	Upper half of dividend	Upper half of quotient	←
R3	Upper half of divisor	Does not change	←
A0	—	Lower half of remainder	←
A1	—	Upper half of remainder	←
CNT	—	Indeterminate	Number of shifts performed
Z flag	—	Zero divide information	←

Usage precautions

CNT is allocated in a stack area by configuring a stack frame as a temporary variable area in the program. Therefore, the value of CNT when program execution is completed is indeterminate.
The dividend is destroyed as a result of program execution.

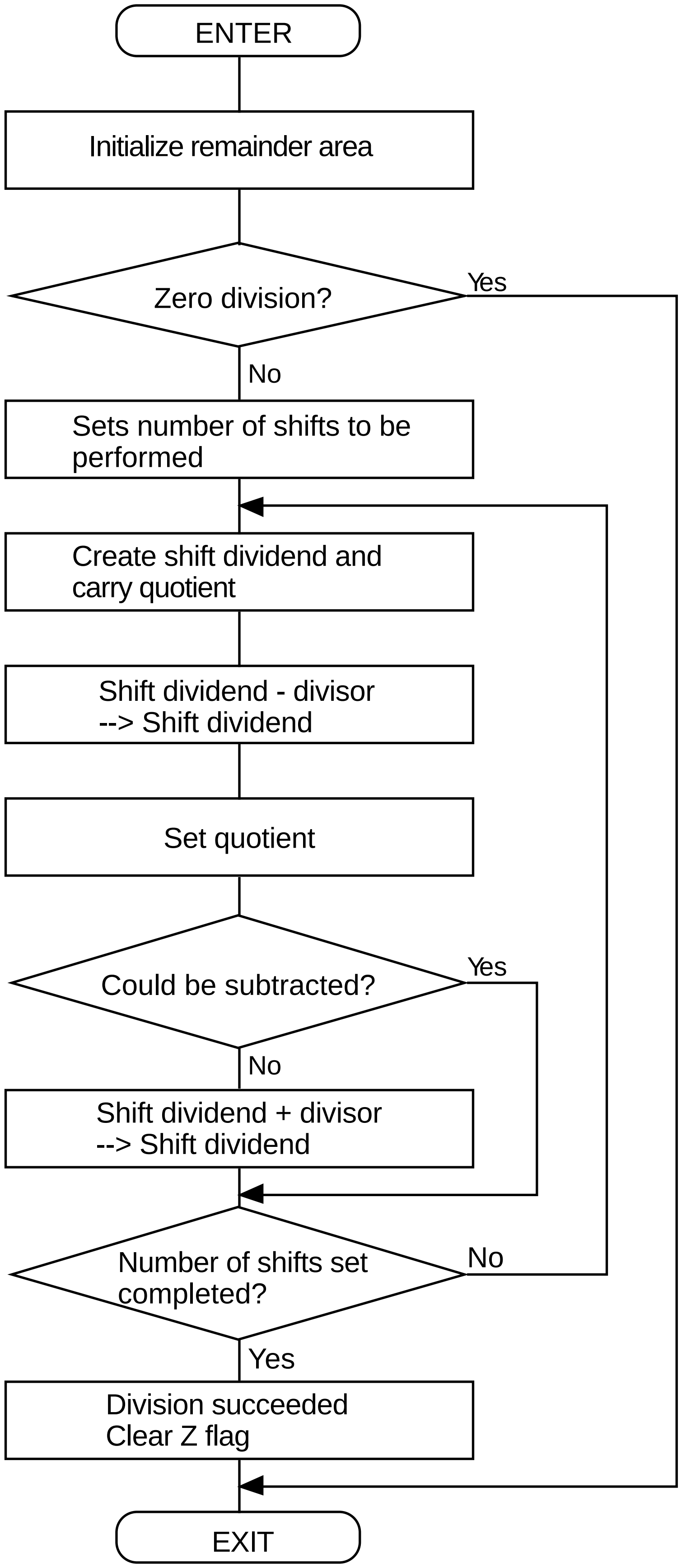
2.12.2 Explanation

This program performs a 32-bit unsigned division using registers. Set the dividend in R2 and R0 and the divisor in R3 and R1 beginning with the upper half, respectively. The quotient and the remainder are output to R2 and R3, and to A1 and A0 beginning with the upper half, respectively. The zero divide information is output to the Z flag.

In this program, the dividend is pushed out one bit at a time beginning with the most significant bit as the program creates a dividend for calculation purposes and the divisor is subtracted from that data to get the quotient beginning with the most significant bit. The quotient and the remainder are obtained by repeating this operation as many times as the number of bits in the dividend.

Z	Meaning
0	Quotient and remainder are valid.
1	Quotient and remainder are invalid because division by zero is attempted.

2.12.3 Flowchart



2.12.4 Program List

```
;*****************************************************************************************
;                                                                                       *
;           M16C Program Collection No. 12                                              *
;           CPU                   : M16C                                                *
;                                                                                       *
;*****************************************************************************************
VromTOP                     .EQU    0F0000H             ; Declares start address of ROM
FBcnst                      .EQU    001000H             ; Assumed FB register value
;===================================================================
;           Title: Dividing 32 bits
;           Outline: Divides 32-bit data together using registers
;           Input:     ----------------------------->   Output:
;           R0   (Lower half of dividend)           R0   (Lower half of quotient)
;           R1   (Lower half of divisor)            R1   (Lower half of divisor)
;           R2   (Upper half of dividend)           R2   (Upper half of quotient)
;           R3   (Upper half of divisor)            R3   (Upper half of divisor)
;           A0   ( )                                A0   (Lower half of remainder)
;           A1   ( )                                A1   (Upper half of remainder)
;           Stack amount used: 3 bytes
;           Notes:    R2R0 ÷ R3R1
;                     Division by zero is returned by Z flag.
;===================================================================
                            .SECTION        PROGRAM,CODE
                            .ORG            VromTOP         ; ROM area
                            .FB             FBcnst          ; Assumes FB register value
DIVIDE32:                                                   ;
;-------------------------------------------------------------------------------------;
;           Declaration of temporary variable                                          ;
;-------------------------------------------------------------------------------------;
CNT                         .EQU    -1                      ; Shift count counter
            ENTER           #1                              ; Sets stack frame
            MOV.B           #0,A0                           ; Initializes remainder area
            MOV.B           #0,A1                           ;
            CMP.W           #0,R1                           ;
            JNE             DIVIDE32_10                     ;
            CMP.W           #0,R3                           ;
            JEQ             DIVIDE32exit                    ; --> Division by zero
DIVIDE32_10:                                                ;
            MOV.B           #32,CNT[FB]                     ; Sets number of shifts performed (32 times)
DIVIDE32_20:                                                ;
            SHL.W           #1,R0                           ; Pushes dividend and carry quotient
            ROLC.W          R2                              ;
            ROLC.W          A0                              ; Creates dividend
            ROLC.W          A1                              ;
            SUB.W           R1,A0                           ; Subtracts divisor
            SBB.W           R3,A1                           ;
            BMC             0,R0                            ; Sets quotient
            JC              DIVIDE32_30                     ; --> Subtraction of divisor succeeded
            ADD.W           R1,A0                           ; Restored to original data because
                                                            ; subtraction of divisor failed
            ADC.W           R3,A1                           ;
DIVIDE32_30:                                                ;
            ADJNZ.B         #-1,CNT[FB],DIVIDE32_20         ; --> Executes next digit
            FCLR            Z                               ; Division succeeded
DIVIDE32exit:                                               ;
            EXITD                                           ; Clears stack frame
;                                                           ;
                            .END                            ;
```

2.13 Dividing 64 Bits

2.13.1 Outline

This program performs an unsigned division on a 64-bit dividend and a 32-bit divisor using registers.

Subroutine name : DIVIDE64	ROM capacity : 78 bytes
Interrupt during execution: Accepted	Number of stacks used : 8 bytes

Register/memory	Input	Output	Usage condition
R0	Lower part of dividend	Lower part of quotient	←
R1	Upper part of dividend	Upper part of quotient	←
R2	Middle part of dividend	Middle part of quotient	←
R3	Most significant part of dividend	Most significant part of quotient	←
A0	Lower half of divisor	Lower half of remainder	←
A1	Upper half of divisor	Upper half of remainder	←
JYOUYO	—	Indeterminate	Shift dividend used for calculation
CNT	—	Indeterminate	Number of shifts performed
Z flag	—	Zero divide information	←

Usage precautions

CNT and JYOUYO are allocated in a stack area by configuring stack frames as temporary variable areas in the program. Therefore, the values of CNT and JYOUYO when program execution is completed are indeterminate. The dividend is destroyed as a result of program execution.

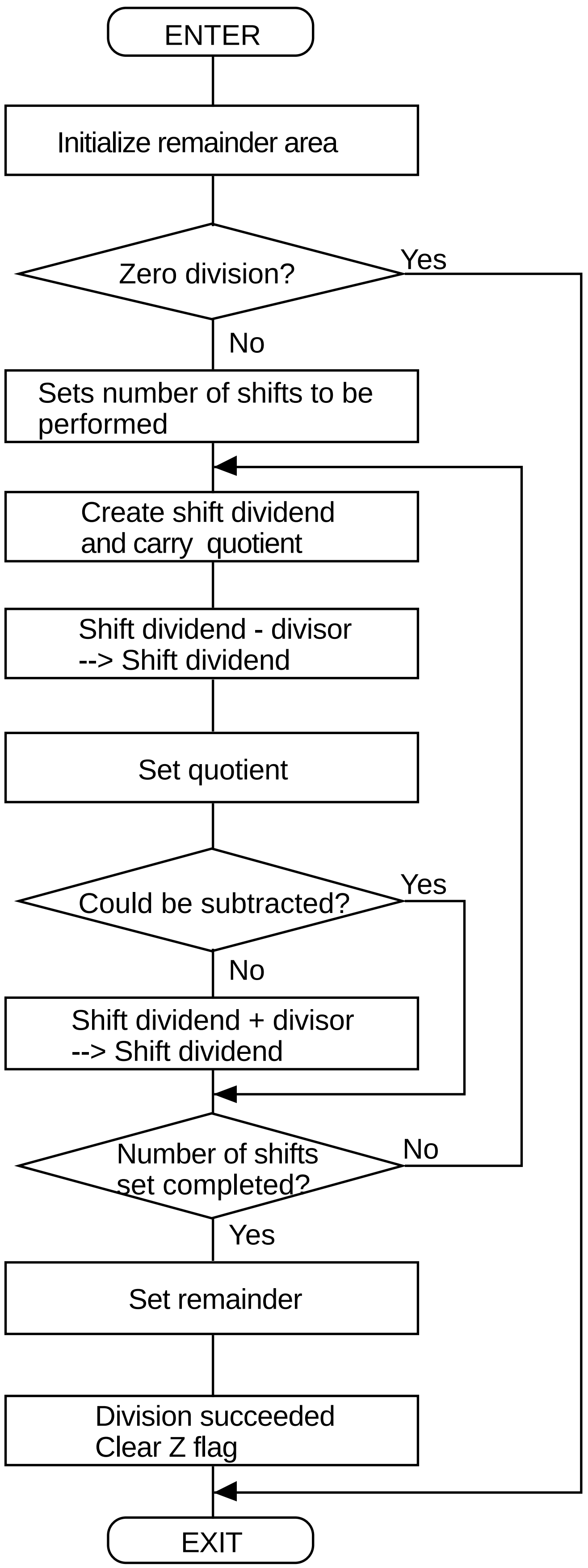
2.13.2 Explanation

This program performs an unsigned division on a 64-bit dividend and a 32-bit divisor using registers. Set the dividend in R3, R1, R2, and R0 beginning with the most significant part, and the divisor in A1 and A0 beginning with the upper half. The quotient and the remainder are output to R3, R1, R2, and R0, and A1 and A0, respectively. The zero divide information is output to the Z flag.

In this program, the dividend is pushed out one bit at a time beginning with the most significant bit as the program creates a dividend for calculation purposes and the divisor is subtracted from that data to get the quotient beginning with the most significant bit. The quotient and the remainder are obtained by repeating this operation as many times as the number of bits in the dividend.

Z	Meaning
0	Quotient and remainder are valid.
1	Quotient and remainder are invalid because division by zero is attempted.

2.13.3 Flowchart



2.13.4 Program List

```
;*******************************************************************************************
;          M16C Program Collection No. 13                                                  *
;          CPU                 : M16C                                                      *
;*******************************************************************************************
VromTOP                        .EQU     0F0000H             ; Declares start address of ROM
FBcnst                         .EQU     001000H             ; Assumed FB register value
;==========================================================================
;          Title: Dividing 64 bits
;          Outline: Divides 64-bit dividend by 32-bit divisor
;          Input:     ----------------------------->    Output:
;          R0   (Lower part of dividend)           R0   (Lower part of quotient)
;          R1   (Upper part of dividend)           R1   (Upper part of quotient)
;          R2   (Middle part of dividend)          R2   (Middle part of quotient)
;          R3   (Most significant part of dividend) R3  (Most significant part of quotient)
;          A0   (Lower half of divisor)            A0   (Lower half of remainder)
;          A1   (Upper half of divisor)            A1   (Upper half of remainder)
;          Stack amount used: 8 bytes
;          Notes:    Division by zero is returned by Z flag.
;                    R3R1R2R0 ÷ A1A0 = R3R1R2R0 remainder A1A0
;======================================================================================
                               .SECTION       PROGRAM,CODE
                               .ORG           VromTOP       ; ROM area
                               .FB            FBcnst        ; Assumes FB register value
DIVIDE64:                                                   ;
;-------------------------------------------------------------------------------------- ;
;          Declaration of temporary variables                                           ;
;-------------------------------------------------------------------------------------- ;
JYOUYO                         .EQU     -6                  ; Used for remainder calculation
CNT                            .EQU     -1                  ; Shift count counter
           ENTER               #6                           ; Sets stack frame
           MOV.W               #0,JYOUYO[FB]                ; Initializes remainder area
           MOV.W               #0,JYOUYO+2[FB]              ;
           MOV.B               #0,JYOUYO+4[FB]              ;
           CMP.W               #0,A0                        ;
           JNE                 DIVIDE64_10                  ;
           CMP.W               #0,A1                        ;
           JEQ                 DIVIDE64exit                 ; --> Division by zero
DIVIDE64_10:                                                ;
           MOV.B               #64,CNT[FB]                  ; Sets number of shifts performed (64 times)
DIVIDE64_20:                                                ;
           SHL.W               #1,R0                        ; Pushes divided and carry quotient
           ROLC.W              R2                           ;
           ROLC.W              R1                           ;
           ROLC.W              R3                           ;
           ROLC.W              JYOUYO[FB]                   ; Creates dividend
           ROLC.W              JYOUYO+2[FB]                 ;
           ROLC.B              JYOUYO+4[FB]                 ;
           SUB.W               A0,JYOUYO[FB]                ; Subtracts divisor
           SBB.W               A1,JYOUYO+2[FB]              ;
           SBB.B               #0,JYOUYO+4[FB]              ;
           BMC                 0,R0                         ; Sets quotient
           JC                  DIVIDE64_30                  ; --> Subtraction of divisor succeeded
           ADD.W               A0,JYOUYO[FB]                ; Restored to original data because
                                                            ; subtraction of divisor failed
           ADC.W               A1,JYOUYO+2[FB]              ;
           ADCF.B              JYOUYO+4[FB]                 ;
DIVIDE64_30:                                                ;
           ADJNZ.B             #-1,CNT[FB],DIVIDE64_20      ; --> Executes next digit
           MOV.W               JYOUYO[FB],A0                ; Sets lower half of remainder
           MOV.W               JYOUYO+2[FB],A1              ; Sets upper half of remainder
           FCLR                Z                            ; Division succeeded
DIVIDE64exit:                                               ;
           EXITD                                            ; Clears stack frame
;                                                           ;
                               .END                         ;
```

2.14 Adding BCD

2.14.1 Outline

This program adds 8 digits of BCD data together by using registers.
This program adds 8 digits of BCD data together between memory locations.

(1) BCD addition (register)

Subroutine name : BCD_ADDITION8	ROM capacity : 13 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of augend	Lower half of addition result	←
R1	Lower half of addend	Does not change	←
R2	Upper half of augend	Upper half of addition result	←
R3	Upper half of addend	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Carry information	←

Usage precautions

The augend is destroyed as a result of program execution.

(2) BCD addition (memory)

Subroutine name : BCD_ADDITIONmemory8	ROM capacity : 20 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	Indeterminate	Used for calculation
R1	—	Indeterminate	Used for calculation
R2	—	—	Unused
R3	—	—	Unused
A0	Augend address	Does not change	←
A1	Addend address	Does not change	←
Memory indicated by A0	Augend	Result of addition	←
Memory indicated by A1	Addend	Does not change	←
C flag	—	Carry information	←

Usage precautions

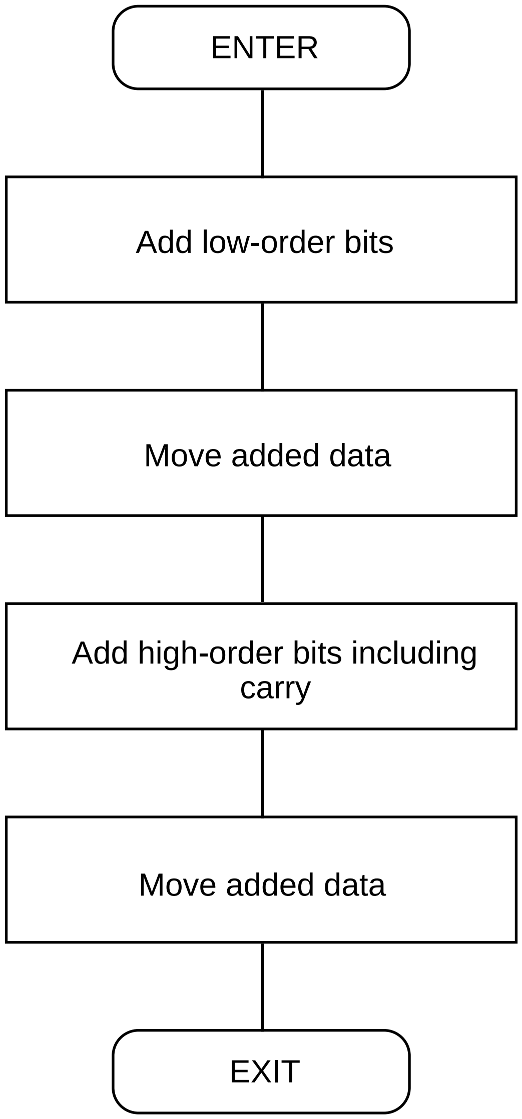
The augend is destroyed as a result of program execution.

2.14.2 Explanation

This program adds 8 digits of BCD data between registers by using a decimal add instruction (DADD). Set the augend in R2 and R0 and the addend in R3 and R1 beginning with the upper half, respectively. The addition result is output to R2 and R0 beginning with the upper half. The carry information is output to the C flag.
This program adds 8 digits of BCD data between memory locations by using a decimal add instruction (DADD). Set the least significant memory address of the augend and that of the addend in the address registers. The addition result is output to the augend's memory location. The carry information is output to the C flag.

C	Meaning
0	Without carry
1	With carry

2.14.3 Flowchart



2.14.4 Program List

```
;************************************************************************************
;                                                                                   *
;           M16C Program Collection No. 14                                          *
;           CPU                  : M16C                                             *
;                                                                                   *
;************************************************************************************
VromTOP                     .EQU     0F0000H              ; Declares start address of ROM
;                                                         ;
;===================================================================
;           Title: Adding 8-digit BCD.
;           Outline: Adds 8-digit BCD together using registers.
;           Input:     ----------------------------->    Output:
;           R0   (Lower half of augend)          R0   (Lower half of addition result)
;           R1   (Lower half of addend)          R1   (Does not change)
;           R2   (Upper half of augend)          R2   (Upper half of addition result)
;           R3   (Upper half of augend)          R3   (Does not change)
;           A0   ( )                             A0   (Unused)
;           A1   ( )                             A1   (Unused)
;           Stack amount used: None
;           Notes:    Result is returned by C flag
;===================================================================
                            .SECTION         PROGRAM,CODE
                            .ORG             VromTOP      ; ROM area
BCD_ADDITION8:                                            ;
            DADD.W          R1,R0                         ; Adds low-order bits
            XCHG.W          R2,R0                         ; Moves added data
            XCHG.W          R3,R1                         ;
            DADC.W          R1,R0                         ; Adds high-order bits
            XCHG.W          R2,R0                         ; Moves added data
            XCHG.W          R3,R1                         ;
            RTS                                           ;
;                                                         ;
;===================================================================
;           Title: Adding 8-bit BCD
;           Outline: Adds 8-bit BCD between memory locations
;           Input:     ----------------------------->    Output:
;           R0   ( )                             R0   (Indeterminate)
;           R1   ( )                             R1   (Indeterminate)
;           R2   ( )                             R2   (Unused)
;           R3   ( )                             R3   (Unused)
;           A0   (Augend address)                A0   (Does not change)
;           A1   (Addend address)                A1   (Does not change)
;           Stack amount used: None
;           Notes:    Result is returned by C flag
;===================================================================
BCD_ADDITIONmemory8:                                      ;
            MOV.W           [A0],R0                       ;
            MOV.W           [A1],R1                       ;
            DADD.W          R1,R0                         ; Adds low-order bits
            MOV.W           R0,[A0]                       ;
            MOV.W           2[A0],R0                      ;
            MOV.W           2[A1],R1                      ;
            DADC.W          R1,R0                         ; Adds high-order bits
            MOV.W           R0,2[A0]                      ;
            RTS                                           ;
;                                                         ;
                            .END                          ;
```

2.15 Subtracting BCD

2.15.1 Outline

This program subtracts 8-digit BCD data using registers.
This program subtracts 8-digit BCD data between memory locations.

(1) BCD subtraction (register)

Subroutine name : BCD_SUBTRACT8	ROM capacity : 13 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of minuend	Lower half of subtraction result	←
R1	Lower half of subtrahend	Does not change	←
R2	Upper half of minuend	Upper half of subtraction result	←
R3	Upper half of subtrahend	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Borrow information	←

Usage precautions

The minuend is destroyed as a result of program execution.

(2) BCD subtraction (memory)

Subroutine name : BCD_SUBTRACTmemory8	ROM capacity : 20 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	Indeterminate	Used for calculation
R1	—	Indeterminate	Used for calculation
R2	—	—	Unused
R3	—	—	Unused
A0	Minuend address	Does not change	←
A1	Subtrahend address	Does not change	←
Memory indicated by A0	Minuend data	Subtraction result	←
Memory indicated by A1	Subtrahend data	Does not change	←
C flag	—	Borrow information	←

Usage precautions

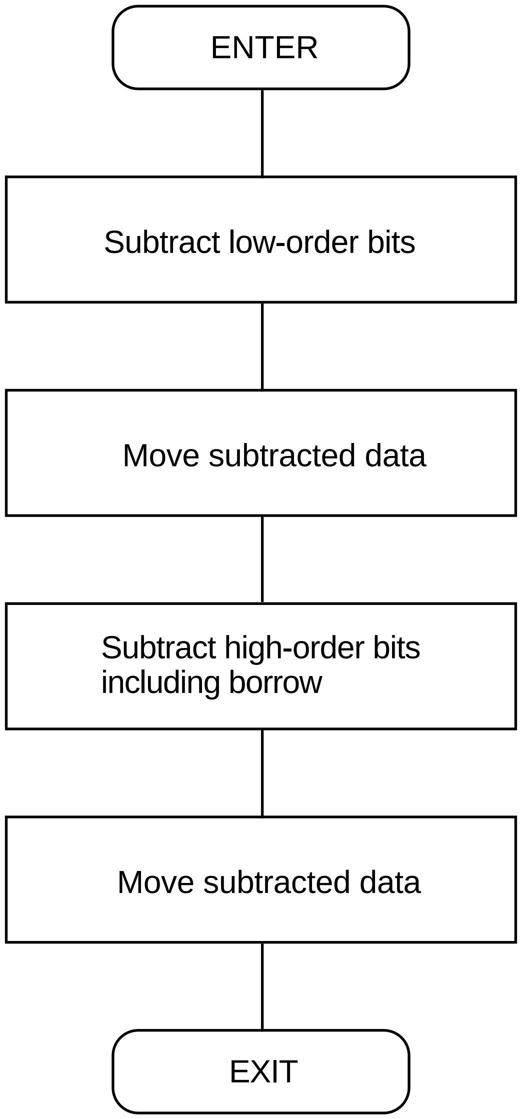
The minuend is destroyed as a result of program execution.

2.15.2 Explanation

This program subtracts 8-digit BCD data between registers by using a decimal subtract instruction (DSUB). Set the minuend in R2 and R0 and the subtrahend in R3 and R1 beginning with the upper half, respectively. The subtraction result is output to R2 and R0 beginning with the upper half. The borrow information is output to the C flag.
This program subtracts 8-digit BCD data between memory locations by using a decimal subtract instruction (DSUB). Set the least significant memory address of the minuend and that of the subtrahend in the address registers. The subtraction result is output to the minuend's memory location. The borrow information is output to the C flag.

C	Meaning
0	With borrow
1	Without borrow

2.15.3 Flowchart



2.15.4 Program List

```
;*********************************************************************************
;                                                                               *
;         M16C Program Collection No. 15                                        *
;         CPU                  : M16C                                           *
;                                                                               *
;*********************************************************************************
VromTOP                  .EQU     0F0000H               ; Declares start address of ROM
;                                                       ;
;==================================================================
;         Title: Subtracting 8-digit BCD
;         Outline: Subtracts 8-digit BCD using registers
;         Input:     ----------------------------->    Output:
;         R0   (Lower half of minuend)        R0   (Lower half of subtraction result)
;         R1   (Lower half of subtrahend)     R1   (Does not change)
;         R2   (Upper half of minuend)        R2   (Upper half of addition result)
;         R3   (Upper half of subtrahend)     R3   (Does not change)
;         A0   ( )                            A0   (Unused)
;         A1   ( )                            A1   (Unused)
;         Stack amount used: None
;         Notes:    Borrow information in C flag
;==================================================================
                         .SECTION        PROGRAM,CODE
                         .ORG            VromTOP        ; ROM area
BCD_SUBTRACT8:                                          ;
          DSUB.W         R1,R0                          ; Subtracts low-order bits
          XCHG.W         R2,R0                          ; Moves subtracted data
          XCHG.W         R3,R1                          ;
          DSBB.W         R1,R0                          ; Subtracts high-order bits
          XCHG.W         R2,R0                          ; Moves subtracted data
          XCHG.W         R3,R1                          ;
          RTS                                           ;
;                                                       ;
;==================================================================
;         Title: Subtracting 8-digit BCD
;         Outline: Subtracts 8-digit BCD between memory locations
;         Input:     ----------------------------->    Output:
;         R0   ( )                            R0   (Indeterminate)
;         R1   ( )                            R1   (Indeterminate)
;         R2   ( )                            R2   (Unused)
;         R3   ( )                            R3   (Unused)
;         A0   (Minuend address)              A0   (Does not change)
;         A1   (Subtrahend address)           A1   (Does not change)
;         Stack amount used: None
;         Notes: Borrow information in C flag
;==================================================================
BCD_SUBTRACTmemory8:                                    ;
          MOV.W          [A0],R0                        ;
          MOV.W          [A1],R1                        ;
          DSUB.W         R1,R0                          ; Subtracts low-order bits
          MOV.W          R0,[A0]                        ;
          MOV.W          2[A0],R0                       ;
          MOV.W          2[A1],R1                       ;
          DSBB.W         R1,R0                          ; Subtracts high-order bits
          MOV.W          R0,2[A0]                       ;
          RTS                                           ;
;                                                       ;
                         .END                           ;
```

2.16 Multiplying BCD

2.16.1 Outline

This program multiplies 4-digit BCD using registers.

Subroutine name : BCD_MULTIPLE4	ROM capacity : 35 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	Lower part of multiplication result	←
R1	Multiplicand	Does not change	←
R2	—	Upper part of multiplication result	←
R3	Multiplier	Indeterminate	←
A0	—	" 0000_{16} "	Number of digits counter
A1	—	" 0000_{16} "	Addition count

Usage precautions

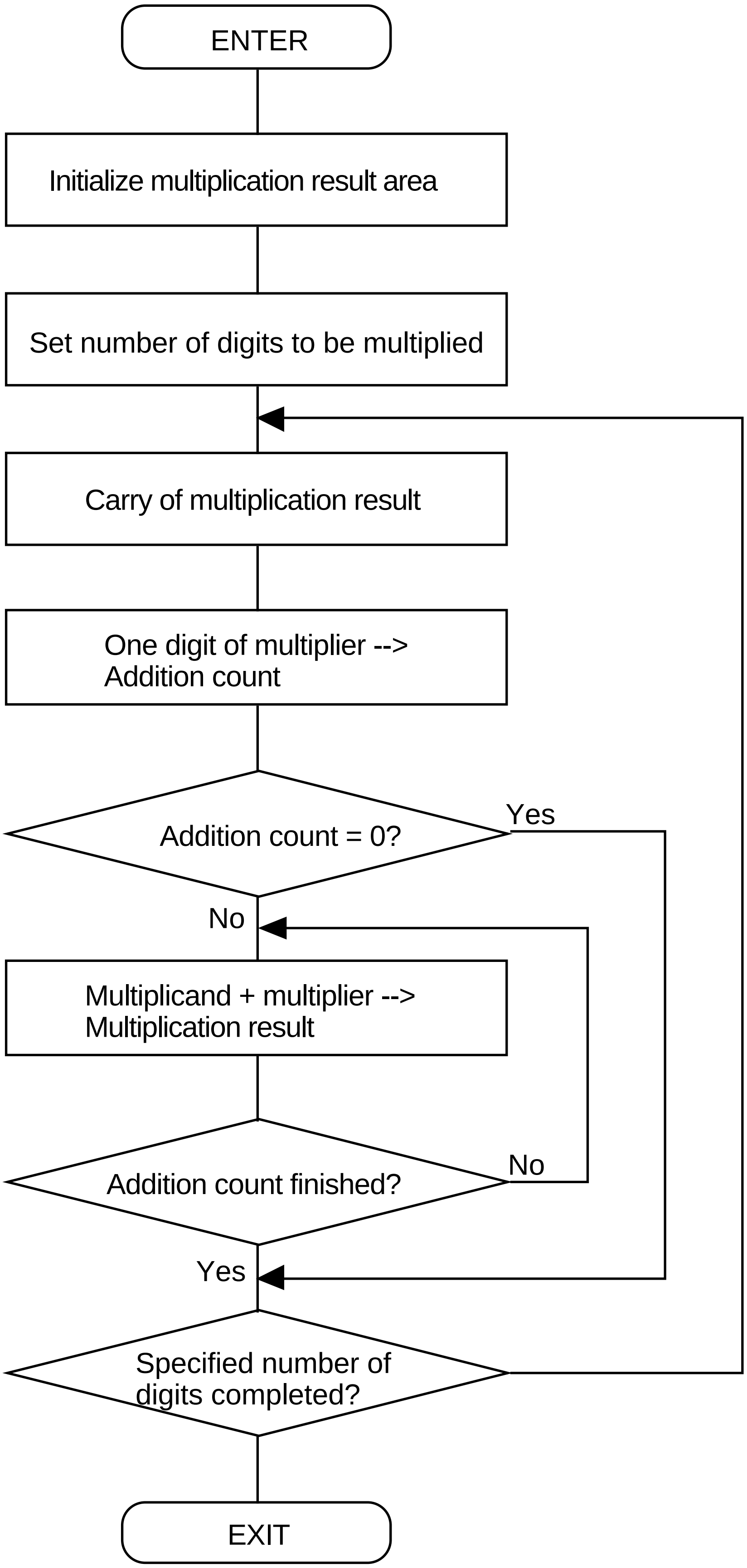
The multiplier is destroyed as a result of program execution.

2.16.2 Explanation

This program multiplies 4-digit BCD together by using registers. Set the multiplicand in R1 and the multiplier in R3, respectively. The multiplication result is output to R2 and R0 beginning with the upper half.

In this program, data for BCD calculation is loaded from the multiplier 4 high-order bits at a time to set an addition count and the multiplicand is added to the multiplication result. The carry deriving from multiplication is shifted in units of 4 bits to the next high-order digit.

2.16.3 Flowchart



2.16.4 Program List

```
;*******************************************************************************************
;                                                                                          *
;           M16C Program Collection No. 16                                                 *
;           CPU                 : M16C                                                     *
;                                                                                          *
;*******************************************************************************************
VromTOP                         .EQU    0F0000H                 ; Declares start address of ROM
;                                                               ;
;=================================================================
;           Title: Multiplying 4-digit BCD
;           Outline: Multiplies 4-digit BCD using registers.
;           Input:      ----------------------------->  Output:
;           R0          ( )                             R0      (Lower half of multiplication result)
;           R1          (Multiplicand)                  R1      (Does not change)
;           R2          ( )                             R2      (Upper half of multiplication result)
;           R3          (Multiplier)                    R3      (Indeterminate)
;           A0          ( )                             A0      (Indeterminate)
;           A1          ( )                             A1      (Indeterminate)
;           Stack amount used: None
;           Notes:
;=================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP                 ; ROM area
BCD_MULTIPLE4:                                                  ;
;                                                               ;
            MOV.W       #0,R0                                   ; Clears multiplication result area
            MOV.W       #0,R2                                   ;
            MOV.B       #4,A0                                   ; Sets number of digits to be multiplied
BCD_MULTIPLE4_10:                                               ;
            SHL.L       #4,R2R0                                 ; Carry processing
            MOV.W       #0001000000000000B,A1                   ; Specifies for 4 bits to be loaded
BCD_MULTIPLE4_20:                                               ;
            SHL.W       #1,R3                                   ; Loads 4 bits
            ROLC.W      A1                                      ; Loads addition count
            JNC         BCD_MULTIPLE4_20                        ; --> Taking 4 bits  not completed
            JEQ         BCD_MULTIPLE4_40                        ; --> Zero (no addition)
BCD_MULTIPLE4_30:                                               ;
            DADD.W      R1,R0                                   ;
            XCHG.W      R2,R0                                   ; Moves high-order data
            DADC.W      #0,R0                                   ; Adds C flag to next high-order digit for carry
            XCHG.W      R2,R0                                   ; Moves high-order data
            ADJNZ.W     #-1,A1,BCD_MULTIPLE4_30                 ; --> Specified addition count not completed
BCD_MULTIPLE4_40:                                               ;
            ADJNZ.W     #-1,A0,BCD_MULTIPLE4_10                 ; --> Specified digit count to be multiplied not completed
            RTS                                                 ;
;                                                               ;
                        .END                                    ;
```

2.17 Dividing BCD

2.17.1 Outline

This program divides 8-digit BCD by using registers.

Subroutine name : BCD_DIVIDE8	ROM capacity : 67 bytes
Interrupt during execution: Accepted	Number of stacks used : 3 bytes

Register/memory	Input	Output	Usage condition
R0	—	Lower half of remainder	←
R1	Lower half of divisor	Does not change	←
R2	—	Upper half of remainder	←
R3	Upper half of divisor	Does not change	←
A0	Lower half of dividend	Lower half of quotient	←
A1	Upper half of dividend	Upper half of quotient	←
CNT	—	Indeterminate	Shift count
Z flag	—	Zero divide information	←

Usage precautions

CNT is allocated in a stack area by configuring a stack frame as a temporary variable area in the program. Therefore, the value of CNT when program execution is completed is indeterminate.
The dividend is destroyed as a result of program execution.

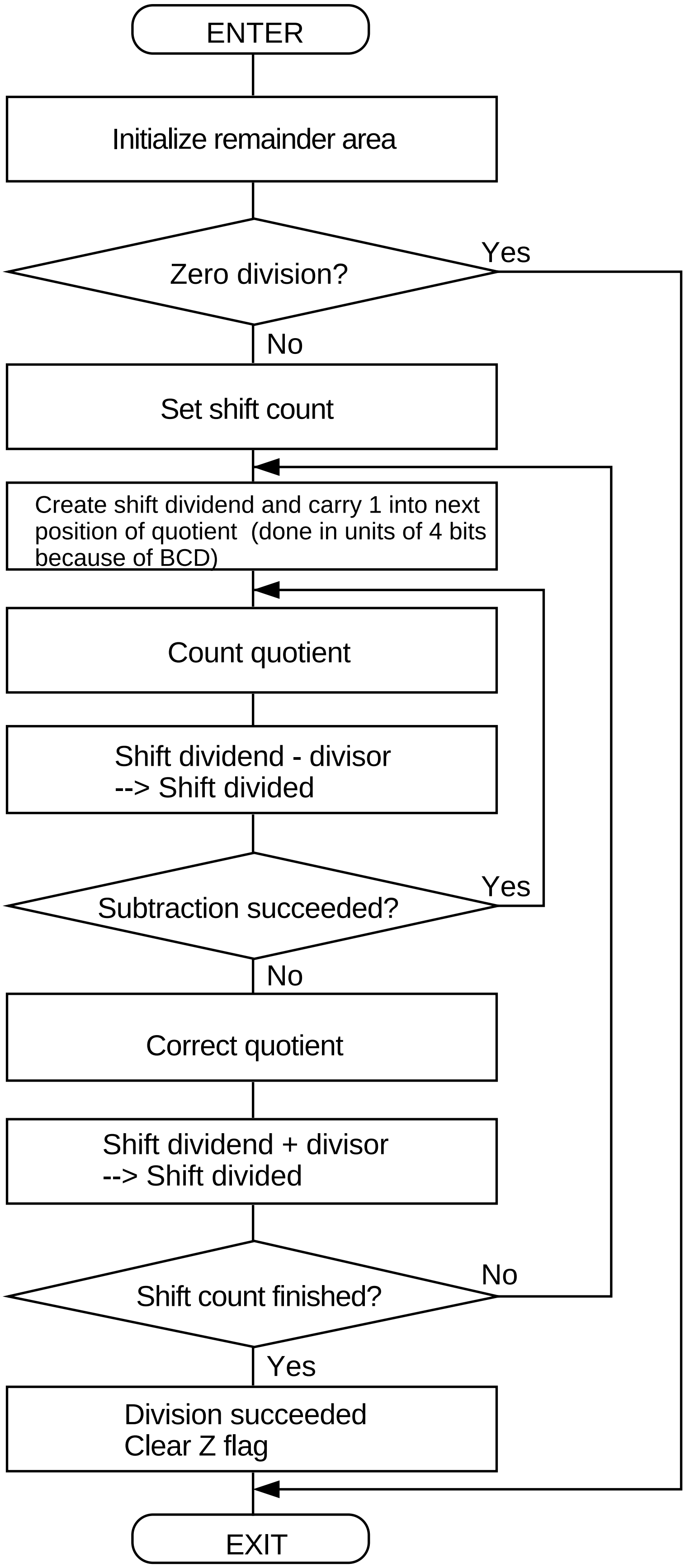
2.17.2 Explanation

This program divides 8-digit BCD together by using registers. Set the dividend in A1 and A0 and the divisor in R3 and R1 beginning with the upper half, respectively. The quotient and the remainder are output to A1 and A0, and to R2 and R0, beginning with the upper half, respectively. The zero divide information is output to the Z flag.

In this program, data for BCD calculation is loaded from the dividend 4 high-order bits at a time to create the dividend to be operated on and the divisor count can be subtracted is counted to obtain the quotient. A carry deriving from the divide operation is shifted in units of 4 bits to the next high-order digit.

Z	Meaning
0	Quotient and remainder are valid.
1	Quotient and remainder are invalid because division by zero is attempted.

2.17.3 Flowchart



2.17.4 Program List

```
;*******************************************************************************************
;           M16C Program Collection No. 17                                                 *
;           CPU                 : M16C                                                     *
;*******************************************************************************************
VromTOP                 .EQU    0F0000H                 ; Declares start address of ROM
FBcnst                  .EQU    001000H                 ; Assumed FB register value
;==========================================================================================
;           Title: Dividing 8-digit BCD
;           Outline: Divides 8-digit BCD using registers
;           Input:     ----------------------------->  Output:
;           R0   ( )                                   R0   (Lower half of remainder)
;           R1   (Lower half of divisor)               R1   (Lower half of divisor)
;           R2   ( )                                   R2   (Upper half of remainder)
;           R3   (Upper half of divisor)               R3   (Upper half of divisor)
;           A0   (Lower half of dividend)              A0   (Lower half of quotient)
;           A1   (Upper half of dividend)              A1   (Upper half of quotient)
;           Stack amount used: 3 bytes
;           Notes:    A1A0 ÷ R3R1
;                     Zero division is returned by Z flag
;==========================================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
                        .FB             FBcnst          ; Sets provisional FB register value
BCD_DIVIDE8:                                            ;
;--------------------------------------------------------------                             ;
;           Declaration of temporary variables          ;
;--------------------------------------------------------------                             ;
CNT                     .EQU    -1                      ; Shift count counter
            ENTER       #1                              ; Sets stack frame
            MOV.W       #0,R0                           ; Initializes remainder area
            MOV.W       #0,R2                           ;
            CMP.W       #0,R1                           ;
            JNE         BCD_DIVIDE8_10                  ;
            CMP.W       #0,R3                           ;
            JEQ         BCD_DIVIDE8exit                 ; --> Zero division
BCD_DIVIDE8_10:                                         ;
            MOV.B       #8,CNT[FB]                      ; Sets number of digits to be divided
BCD_DIVIDE8_20:                                         ;
            BSET        12,R2                           ; Specifies 4-bit carry
BCD_DIVIDE8_30:                                         ;
            SHL.W       #1,A0                           ; Pushes dividend and carries 1 in quotient
            ROLC.W      A1                              ; Pushes dividend and carries 1 in quotient
            ROLC.W      R0                              ; Creates dividend
            ROLC.W      R2                              ;
            JNC         BCD_DIVIDE8_30                  ; --> 4-bit carry not completed
BCD_DIVIDE8_40:                                         ;
            INC.W       A0                              ; Quotient + 1
            DSUB.W      R1,R0                           ; Subtraction by divisor
            XCHG.W      R2,R0                           ; Moves data
            XCHG.W      R3,R1                           ;
            DSBB.W      R1,R0                           ;
            XCHG.W      R2,R0                           ; Moves data
            XCHG.W      R3,R1                           ;
            JGEU        BCD_DIVIDE8_40                  ; --> Subtraction by divisor succeeded
            DEC.W       A0                              ; Quotient corrected
            DADD.W      R1,R0                           ; Restored to original data because divisor subtraction failed
            XCHG.W      R2,R0                           ; Moves data
            XCHG.W      R3,R1                           ;
            DADC.W      R1,R0                           ;
            XCHG.W      R2,R0                           ; Moves data
            XCHG.W      R3,R1                           ;
            ADJNZ.B     #-1,CNT[FB],BCD_DIVIDE8_20      ; --> Executes next digit
            FCLR        Z                               ; Division succeeded
BCD_DIVIDE8exit:                                        ;
            EXITD                                       ; Clears stack frame
;                                                       ;
                        .END                            ;
```

2.18 Converting from HEX Code to BCD Code

2.18.1 Outline

This program converts 1-byte HEX code into 2-byte BCD code.

Subroutine name : HEXtoBCD_1byte	ROM capacity : 19 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	—	BCD code	←
R1H	—	" 00_{16} "	Loop count
R1L	HEX code	Indeterminate	←
R2	—	Indeterminate	Used to save data
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused

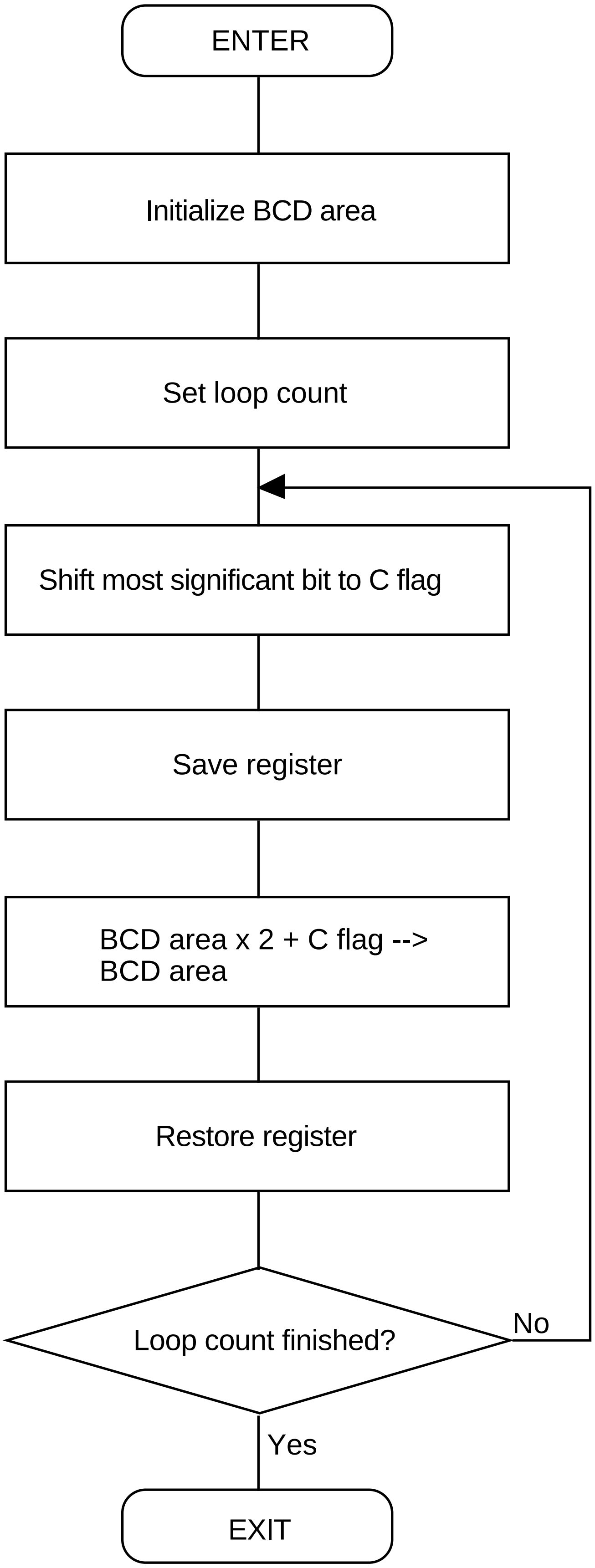
Usage precautions

HEX code is destroyed as a result of program execution.

2.18.2 Explanation

This program converts 1-byte HEX code into 2-byte BCD code. Set the HEX code in R1L. The BCD code is output to R0.
In this program, the HEX code is doubled by decimal calculation sequentially beginning with the most significant bit and the results are added. This operation is repeated by a specified number of bits as the HEX code is converted into BCD code.

2.18.3 Flowchart



2.18.4 Program List

```
;*************************************************************************************************
;                                                                                               *
;           M16C Program Collection No. 18                                                      *
;           CPU                 : M16C                                                          *
;                                                                                               *
;*************************************************************************************************
VromTOP                 .EQU     0F0000H                ; Declares start address of ROM
;                                                       ;
;==============================================================================
;           Title: Converting from HEX code to BCD code
;           Outline: Converts 1-byte HEX code into 2-byte BCD code
;           Input:    ----------------------------->    Output:
;           R0L       ( )                              R0         (BCD code)
;           R0H       ( )
;           R1H       (HEX code)                       R1L        (Indeterminate)
;           R1H       ( )                              R1H        (Indeterminate)
;           R2        ( )                              R2         (Indeterminate)
;           R3        ( )                              R3         (Unused)
;           A0        ( )                              A0         (Unused)
;           A1        ( )                              A1         (Unused)
;           Stack amount used: None
;           Notes:
;==============================================================================
                        SECTION         PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
BINtoBCD_1byte:                                         ;
            MOV.W       #0,R0                           ; Initializes BCD area
            MOV.B       #8,R1H                          ; Sets loop count
BINtoBCD_1byte_10:                                      ;
            SHL.L       #1,R1L                          ; Shifts most significant bit to C flag
            XCHG.W      R1,R2                           ; Saves register
            MOV.W       R0,R1                           ;
            DADC.W      R1,R0                           ; Doubled by decimal calculation + C flag
            XCHG.W      R1,R2                           ; Restores register
            ADJNZ.W     #-1,R1H,BINtoBCD_1byte_10       ; --> Executes next digit
            RTS                                         ;
;                                                       ;
                        .END                            ;
```

2.19 Converting from HEX Code to BCD Code

2.19.1 Outline

This program converts 4-byte HEX code into 5-byte BCD code.

Subroutine name : HEXtoBCD_4byte	ROM capacity : 38 bytes
Interrupt during execution: Accepted	Number of stacks used : 2 bytes

Register/memory	Input	Output	Usage condition
R0	—	Lower part of BCD code	←
R1	Lower half of HEX code	Indeterminate	←
R2	—	Middle part of BCD code	←
R3	Upper half of HEX code	Indeterminate	←
A0	—	" 0000_{16} "	Number of digits counter
A1	—	Upper part of BCD code	←

Usage precautions

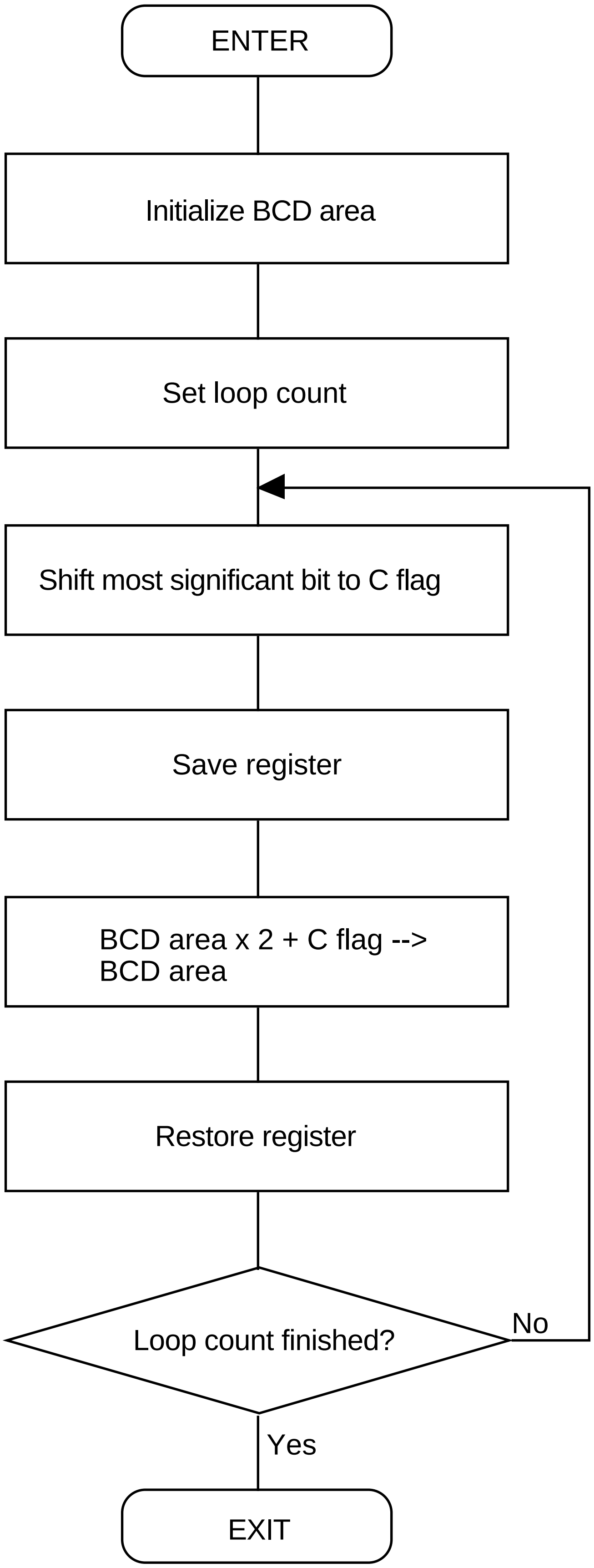
The HEX code is destroyed as a result of program execution.

2.19.2 Explanation

This program converts 4-byte HEX code into 5-byte BCD code. Set the HEX code in R3 and R1 beginning with the upper half. The BCD code is output to A1, R2, and R0 beginning with the most significant part.

In this program, the HEX code is doubled by decimal calculation sequentially beginning with the most significant bit and the results are added. This operation is repeated by a specified number of bits as the HEX code is converted into BCD code.

2.19.3 Flowchart



2.19.4 Program List

```
;*****************************************************************************************
;                                                                                       *
;           M16C Program Collection No. 19                                              *
;           CPU                 : M16C                                                  *
;                                                                                       *
;*****************************************************************************************
VromTOP                     .EQU     0F0000H                 ; Declares start address of ROM
;                                                            ;
;=================================================================
;           Title: Converting from HEX code to BCD code
;           Outline: Converts 4-byte HEX code into 5-byte BCD code
;           Input:    ------------------------------>    Output:
;           R0   ( )                                      R0   (Lower part of BCD)
;           R1   (Lower half of HEX code)                 R1   (Indeterminate)
;           R2   ( )                                      R2   (Middle part of BCD)
;           R3   (Upper half of HEX code)                 R3   (Indeterminate)
;           A0   ( )                                      A0   (Indeterminate)
;           A1   ( )                                      A1   (Upper part of BCD)
;           Stack amount used: 2bytes
;           Notes:
;=================================================================
                            SECTION     PROGRAM,CODE
                            .ORG        VromTOP              ; ROM area
BINtoBCD_4byte:                                              ;
            MOV.W           #0,R0                            ; Initializes BCD area
            MOV.W           #0,R2                            ;
            MOV.W           #0,A1                            ;
            MOV.B           #32,A0                           ; Sets loop count
BINtoBCD_4byte_10:                                           ;
            SHL.L           #1,R3R1                          ; Shifts most significant bit to C flag
            PUSH.W          R1                               ; Saves register
            MOV.W           R0,R1                            ;
            DADC.W          R1,R0                            ; Doubled by decimal calculation + C flag
            XCHG.W          R2,R0                            ;
            MOV.W           R0,R1                            ;
            DADC.W          R1,R0                            ; Doubled by decimal calculation + carry
            XCHG.W          R0,A1                            ;
            MOV.W           R0,R1                            ;
            DADC.W          R1,R0                            ; Doubled by decimal calculation + carry
            XCHG.W          R0,A1                            ;
            XCHG.W          R2,R0                            ;
            POP.W           R1                               ; Restores register
            ADJNZ.W         #-1,A0,BINtoBCD_4byte_10         ; --> Executes next digit
            RTS                                              ;
;                                                            ;
                            .END                             ;
```

2.20 Converting from BCD Code to HEX Code

2.20.1 Outline

This program converts 1-byte BCD code into 1-byte HEX code.

Subroutine name : BCDtoHEX_1byte	ROM capacity : 19 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0L	—	HEX code	←
R0H	BCD code	Indeterminate	←
R1L	—	" 00_{16} "	Loop count
R1H	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused

Usage precautions

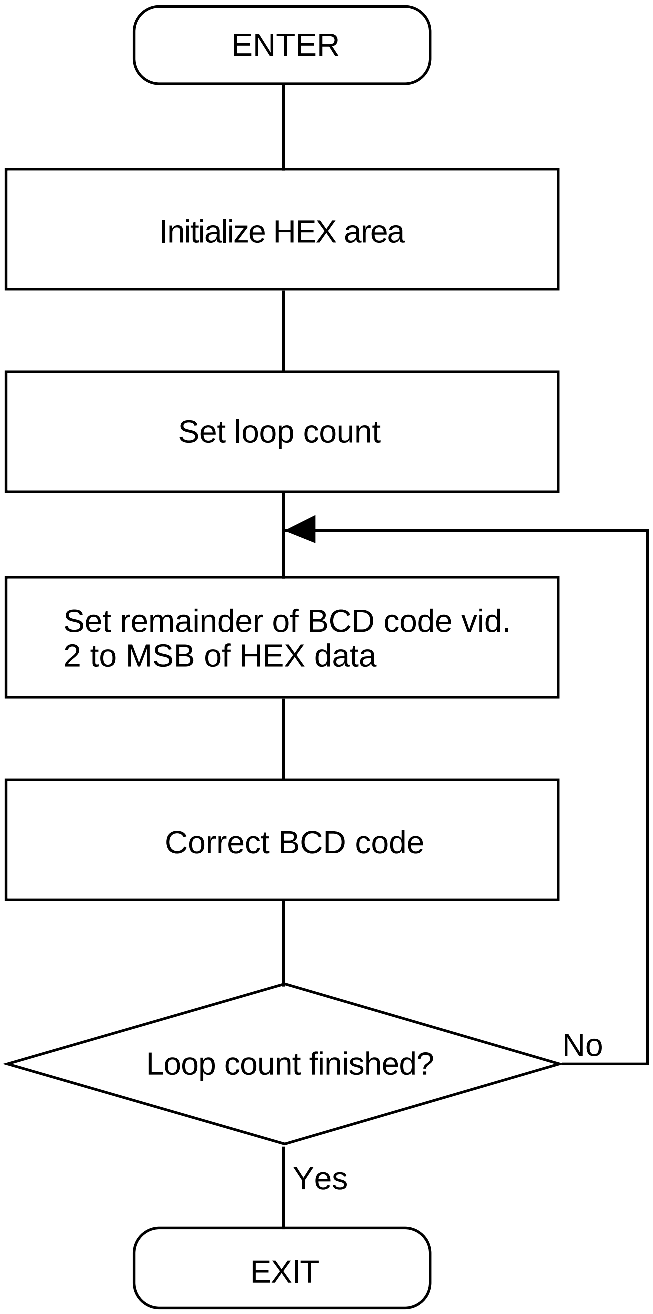
The BCD code is destroyed as a result of program execution.

2.20.2 Explanation

This program converts 1-byte BCD code into 1-byte HEX code. Set the BCD code in R0H. The HEX code is output to R0L.

In this program, the BCD code is divided by 2 (shifted right) and the remainder is loaded into the register as HEX code. If a significant bit is transferred from the BCD's high-order digit to the low-order digit, numeric correction is applied.

2.20.3 Flowchart



2.20.4 Program List

```
;*******************************************************************************************
;                                                                                          *
;           M16C Program Collection No. 20                                                 *
;           CPU                 : M16C                                                     *
;                                                                                          *
;*******************************************************************************************
VromTOP                 .EQU    0F0000H                 ; Declares start address of ROM
;                                                       ;
;===========================================================================================
;           Title: Converting from BCD code to HEX code
;           Outline: Converts 1-byte BCD code into 1-byte HEX code
;           Input:      ----------------------------->  Output:
;           R0L         ( )                             R0L     (HEX code)
;           R0H         (BCD code)                      R0H     (Indeterminate)
;           R1L         ( )                             R1L     (Indeterminate)
;           R1H         ( )                             R1H     (Unused)
;           R2          ( )                             R2      (Unused)
;           R3          ( )                             R3      (Unused)
;           A0          ( )                             A0      (Unused)
;           A1          ( )                             A1      (Unused)
;           Stack amount used: None
;           Notes:
;===========================================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
BCDtoBIN_1byte:                                         ;
            MOV.B       #0,R0L                          ; Initializes HEX area
            MOV.B       #8,R1L                          ; Sets loop count
BCDtoBIN_1byte_10:                                      ;
            SHL.B       #-1,R0H                         ; Shifts most significant bit
            RORC.B      R0L                             ;
            BTST        3+8,R0                          ;
            JEQ         BCDtoBIN_1byte_20               ;
            SUB.B       #3,R0H                          ;
BCDtoBIN_1byte_20:                                      ;
            ADJNZ.B     #-1,R1L,BCDtoBIN_1byte_10       ; --> Executes next BCD digit
            RTS                                         ;
;                                                       ;
                        .END                            ;
```

2.21 Converting from BCD Code to HEX Code

2.21.1 Outline

This program converts 4-byte BCD code into 4-byte HEX code.

Subroutine name : BCDtoHEX_4byte	ROM capacity : 42 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of BCD code	Indeterminate	←
R1	—	Lower part of HEX code	←
R2	Upper half of BCD code	Indeterminate	←
R3	—	Upper part of HEX code	←
A0	—	" 0000_{16} "	Loop count
A1	—	" 0000_{16} "	Number of digits counter

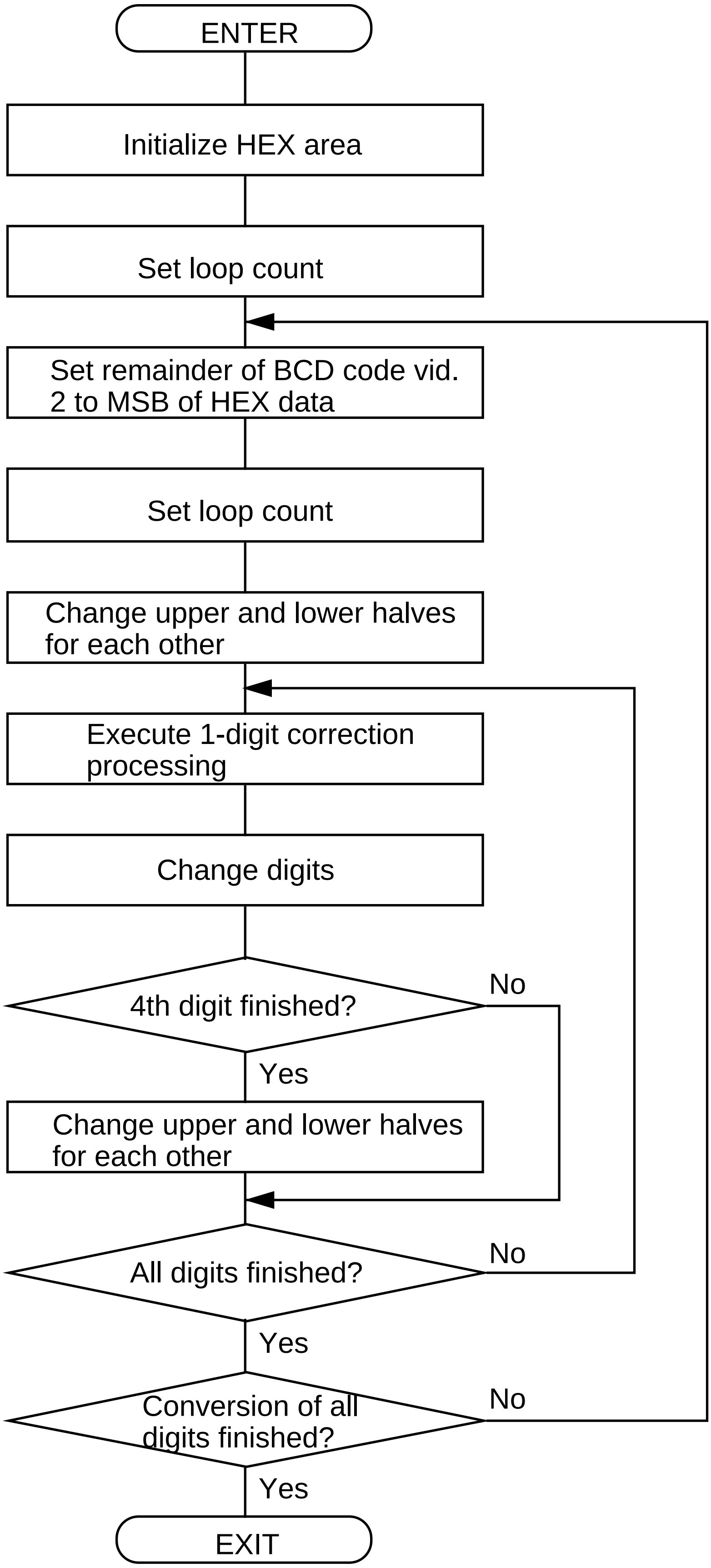
Usage precautions

The BCD code is destroyed as a result of program execution.

2.21.2 Explanation

This program converts 4-byte BCD code into 4-byte HEX code. Set the BCD code in R2 and R0 beginning with the upper half. The HEX code is output to R3 and R1 beginning with the upper half.
In this program, the BCD code is divided by 2 (shifted right) and the remainder is loaded into the register as HEX code. If a significant bit is transferred from the BCD's high-order digit to the low-order digit, numeric correction is applied.

2.21.3 Flowchart



2.21.4 Program List

```
;*******************************************************************************************
;                                                                                          *
;          M16C Program Collection No. 21                                                  *
;          CPU                : M16C                                                       *
;                                                                                          *
;*******************************************************************************************
VromTOP                  .EQU    0F0000H                   ; Declares start address of ROM
;                                                          ;
;=================================================================
;          Title: Converting from BCD code to HEX code
;          Outline: Converts 4-byte BCD code into 4-byte HEX code
;          Input:    ----------------------------->   Output:
;          R0  (Lower half of BCD code)       R0  (Indeterminate)
;          R1  ( )                            R1  (Lower part of HEX)
;          R2  (Upper half of HEX code)       R2  (Indeterminate)
;          R3  ( )                            R3  (Upper part of HEX)
;          A0  ( )                            A0  (Indeterminate)
;          A1  ( )                            A1  (Indeterminate)
;          Stack amount used: None
;          Notes:
;=================================================================
                         .SECTION      PROGRAM,CODE
                         .ORG          VromTOP             ; ROM area
BCDtoBIN_1byte:                                            ;
          MOV.W          #0,R1                             ; Initializes HEX area
          MOV.W          #0,R3                             ;
          MOV.B          #32,A0                            ; Sets loop count
BCDtoBIN_1byte_10:                                         ;
          SHL.W          #-1,R2                            ; Shifts most significant bit
          RORC.W         R0                                ;
          RORC.W         R3                                ;
          RORC.W         R1                                ;
          MOV.B          #8,A1                             ; Sets loop count
          XCHG.W         R2,R0                             ; Changes upper/lower halves for each other
BCDtoBIN_1byte_20:                                         ;
          BTST           3,R0                              ;
          JEQ            BCDtoBIN_1byte_30                 ; --> Correction not required
          SUB.W          #3,R0                             ; Executes correction
BCDtoBIN_1byte_30:                                         ;
          ROT.W          #-4,R0                            ; Changes digits
          CMP.B          #5,A1                             ; Determines whether high-order correction is completed
          JNE            BCDtoBIN_1byte_40                 ; --> Change of upper/lower halves not required
          XCHG.W         R2,R0                             ; Changes upper/lower halves for each other
BCDtoBIN_1byte_40:                                         ;
          ADJNZ.W        #-1,A1,BCDtoBIN_1byte_20          ; --> Processes next digit correction
          ADJNZ.W        #-1,A0,BCDtoBIN_1byte_10          ; --> Executes next digit
          RTS                                              ;
;                                                          ;
                         .END                              ;
```

2.22 Converting from Floating-point Number to Binary Number

2.22.1 Outline

This program converts a single-precision, floating- point number into a 32-bit singed binary number.

Subroutine name : FLOATINGtoBIN	ROM capacity : 72 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Mid and lower parts of mantissa	Indeterminate	←
R1	—	Lower half of signed binary	←
R2	Exponent, upper part of mantissa	Indeterminate	←
R3	—	Upper half of signed binary	←
A0	—	Indeterminate	Used to save sign bit
A1	—	—	Unused

Usage precautions

If the magnitude of a single-precision, floating-point number is equal to or greater than "2^{31}", the program outputs the maximum value of the same sign; if less than "1", the program outputs a "0". The floating-point data is destroyed as a result of program execution.

2.22.2 Explanation

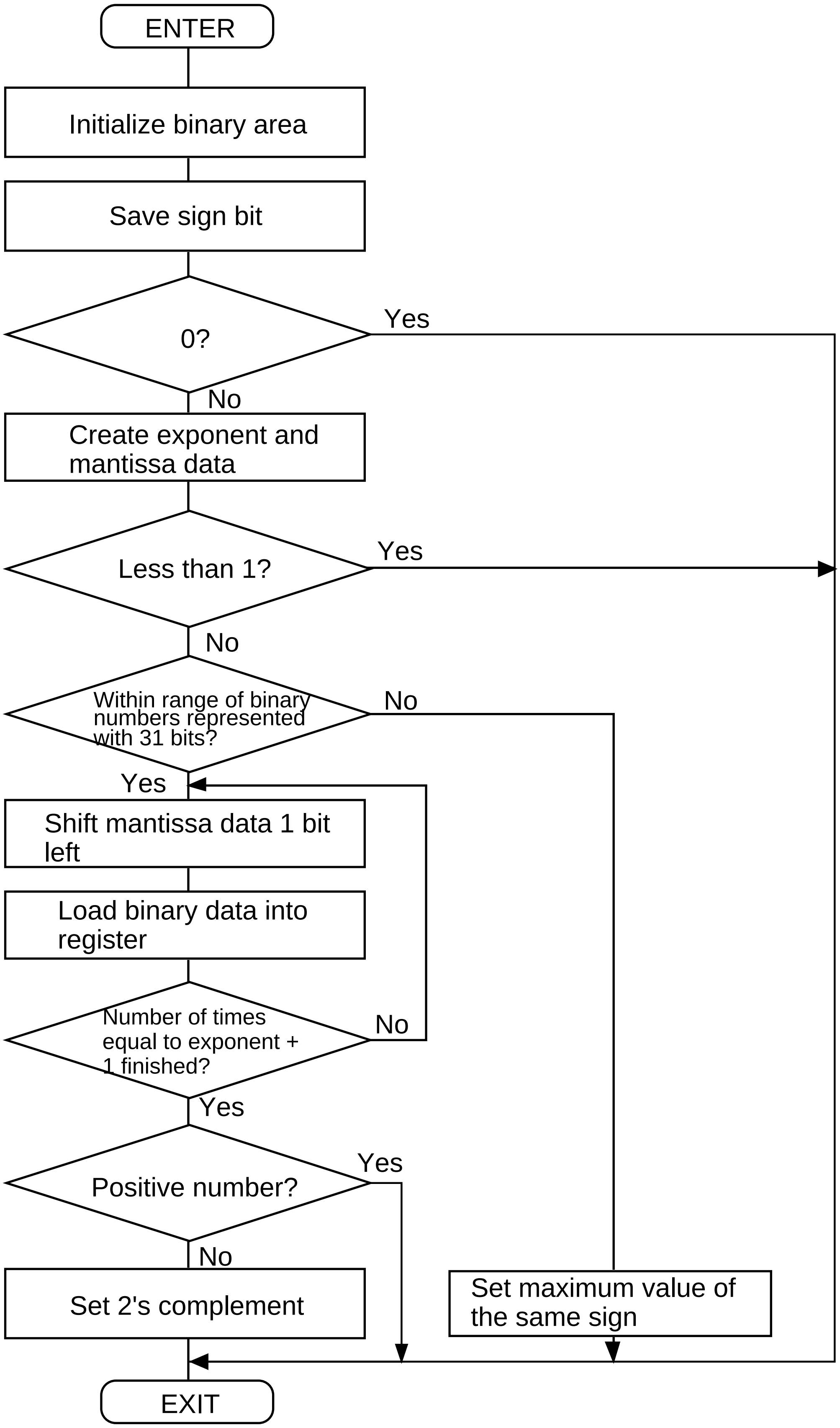
This program converts a single-precision, floating- point number into a 32-bit singed binary number. Set the single-precision, floating-point number in R2 and R0. A signed binary number is output to R3 and R1 beginning with the upper half.

In this program, after confirming that the single- precision, floating-point number is convertible, the data is loaded into the registers while shifting the mantissa data left, and this operation is repeated as many times as dictated by the exponent to create a binary number. Finally, the resulting data is adjusted to make it matched to the sign bit of the input data.

If the magnitude of a single-precision, floating-point number is equal to or greater than "2^{31}", the program outputs the maximum value of the same sign; if less than "1", the program outputs a "0". In either case, the result is output to R3 and R1.

R3 , R1	Meaning
7FFFFFFFH	Magnitude of a single-precision, floating-point number is equal to or greater than "2^{31}" (sign +)
80000000H	Magnitude of a single-precision, floating-point number is equal to or greater than "2^{31}" (sign -)
00000000H	Magnitude of a single-precision, floating-point number is less than "1"

2.22.3 Flowchart



2.22.4 Program List

```
;*************************************************************************************
;                                                                                    *
;        M16C Program Collection No. 22                                              *
;        CPU               : M16C                                                    *
;                                                                                    *
;*************************************************************************************
VromTOP                 .EQU     0F0000H                ; Declares start address of ROM
;                                                       ;
;====================================================================
;        Title: Converting from single-precision, floating-point number to binary number
;        Outline: Converts single-precision, floating-point number into 32-bit signed binary number
;        Input:    ----------------------------->  Output:
;        R0   (Mid and lower parts of mantissa)    R0   (Indeterminate)
;        R1   ( )                                  R1   (Lower half of signed binary)
;        R2   (Exponent, upper part of mantissa)   R2   (Indeterminate)
;        R3   ( )                                  R3   (Upper half of signed binary)
;        A0   ( )                                  A0   (Indeterminate)
;        A1   ( )                                  A1   (Unused)
;        Stack amount used: None
;        Notes:
;====================================================================
                        .SECTION      PROGRAM,CODE
                        .ORG          VromTOP           ; ROM area
FLOATINGtoBIN:                                          ;
        XCHG.W          R0,R2                           ; Changes registers
        MOV.W           #0,R1                           ; Initializes binary area
        MOV.W           #0,R3                           ;
        MOV.W           R0,A0                           ; Saves sign bit
        BCLR            15,R0                           ; Clears sign
        CMP.W           #0,R0                           ;
        JNE             FLOATINGtoBIN_10                ;
        CMP.W           #0,R2                           ;
        JEQ             FLOATINGtoBIN_EXIT              ; --> Zero
FLOATINGtoBIN_10:                                       ;
        BTSTS           7,R0                            ; Sets LSB of exponent to C flag
                                                        ; and adds 1.0 to mantissa
        ROLC.B          R0H                             ; Creates exponent
        SUB.B           #7FH,R0H                        ; Determines whether magnitude is less than 1
        JNC             FLOATINGtoBIN_EXIT              ; --> Sets 0 because magnitude is less than 1
        CMP.B           #31,R0H                         ; Determines whether number is within representation range
        JLTU            FLOATINGtoBIN_20                ; --> Number is within binary representation range
        BSET            15,R3                           ; Initial sets maximum value of the same sign
        BTST            15,A0                           ; Checks sign bit
        JNE             FLOATINGtoBIN_EXIT              ; --> Negative number (80000000)
        NOT.W           R1                              ; Positive number (7FFFFFFF)
        NOT.W           R3                              ;
        JMP.B           FLOATINGtoBIN_EXIT              ;
FLOATINGtoBIN_20:                                       ;
        INC.B           R0H                             ; Adjusts loop count
FLOATINGtoBIN_30:                                       ;
        SHL.W           #1,R2                           ; Pushes mantissa data
        ROLC.B          R0L                             ;
        ROLC.W          R1                              ; Loads result into register
        ROLC.W          R3                              ;
        ADJNZ.B         #-1,R0H,FLOATINGtoBIN_30        ; --> Conversion loop
        BTST            15,A0                           ; Checks sign bit
        JEQ             FLOATINGtoBIN_EXIT              ; --> Positive number
        NOT.W           R1                              ; Takes 2's complement
        NOT.W           R3                              ;
        ADD.W           #1,R1                           ;
        ADCF.W          R3                              ;
FLOATINGtoBIN_EXIT:                                     ;
        RTS                                             ;
;                                                       ;
                        .END                            ;
```

2.23 Converting from Binary Number to Floating-point Number

2.23.1 Outline

This program converts a 32-bit signed binary number into a single-precision, floating-point number.

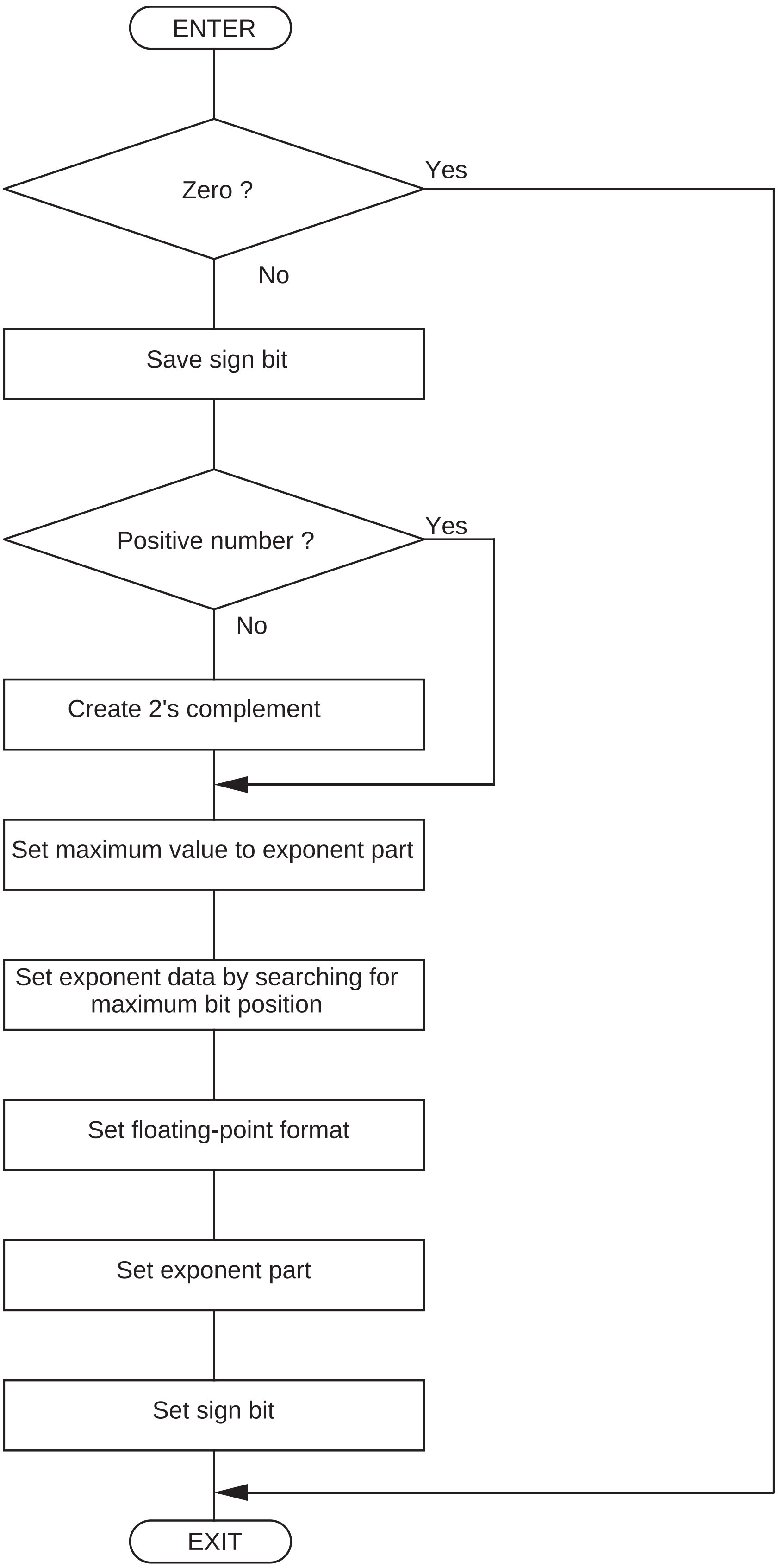
Subroutine name : BINtoFLOATING	ROM capacity : 67 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0	Lower half of signed binary	Mid and lower parts of mantissa	←
R1	—	Indeterminate	Used for format conversion
R2	Upper half of signed binary	Exponent, upper part of mantissa	←
R3	—	Indeterminate	Used to save sign bit
A0	—	—	Unused
A1	—	—	Unused
Usage precautions			

2.23.2 Explanation

This program converts a 32-bit signed binary number into a single-precision, floating-point number. Set the 32-bit signed binary number in R2 and R0 beginning with the upper half. A single-precision, floating-point number is output to R2 and R0.
In this program, after confirming whether the input data is "0" and adjusting the data by the sign, a maximum value is set to the exponent part that can be represented by a 32-bit signed binary number. Next, the input data is shifted left while calculating (subtracting) the exponent part to create mantissa data. Finally, the resulting data is adjusted to suit the format of single-precision, floating-point numbers.

2.23.3 Flowchart



2.23.4 Program List

```
;*************************************************************************************
;                                                                                    *
;        M16C Program Collection No. 23                                              *
;        CPU              : M16C                                                     *
;                                                                                    *
;*************************************************************************************
VromTOP             .EQU    0F0000H                 ; Declares start address of ROM
;                                                   ;
;=====================================================================
;        Title: Converting from binary number to single-precision, floating-point number
;        Outline: Converts 32-bit signed binary number into single-precision, floating-point number
;        Input:   ----------------------------->   Output:
;        R0  (Lower half of signed binary)          R0  (Mid and lower parts of mantissa)
;        R1  ( )                                    R1  (Indeterminate)
;        R2  (Upper half of signed binary)          R2  (Exponent, upper part of mantissa)
;        R3  ( )                                    R3  (Indeterminate)
;        A0  ( )                                    A0  (Unused)
;        A1  ( )                                    A1  (Unused)
;        Stack amount used: None
;        Notes:
;=====================================================================
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP         ; ROM area
BINtoFLOATING:                                      ;
        XCHG.W      R2,R0                           ; Changes data
        CMP.W       #0,R2                           ;
        JNE         BINtoFLOATING_10                ;
        CMP.W       #0,R0                           ;
        JEQ         BINtoFLOATING_EXIT              ; --> ZERO
BINtoFLOATING_10:                                   ;
        MOV.W       R0,R3                           ; Saves sign bit
        BTST        15,R0                           ; Checks sign
        JEQ         BINtoFLOATING_20                ; --> Positive number
        NOT.W       R2                              ; Takes 2's complement
        NOT.W       R0                              ;
        ADD.W       #1,R2                           ;
        ADCF.W      R0                              ;
BINtoFLOATING_20:                                   ;
        MOV.B       #9DH+1,R1L                      ; Sets maximum value to exponent part
BINtoFLOATING_30:                                   ;
        BTST        15,R0                           ; Search of maximum bit position
        JNE         BINtoFLOATING_40                ; --> Finds maximum bit
        SHL.W       #1,R2                           ; Pushes for search of maximum bit position
        ROLC.W      R0                              ;
        SUB.B       #1,R1L                          ; Counts down exponent
        JMP         BINtoFLOATING_30                ;
BINtoFLOATING_40:                                   ;
        MOV.B       #7,R1H                          ; Number of shifts to adjust mantissa position
BINtoFLOATING_50:                                   ;
        SHL.W       #-1,R0                          ; Adjusts mantissa position
        RORC.W      R2                              ;
        ADJNZ.B     #-1,R1H,BINtoFLOATING_50        ; --> Adjustment not completed
        MOV.B       R1L,R0H                         ; Sets exponent
        SHL.W       #-1,R0                          ; Adjusts format
        RORC.W      R2                              ;
        BTST        15,R3                           ; Sets sign bit
        BMC         15,R0                           ;
BINtoFLOATING_EXIT:                                 ;
        XCHG.W      R2,R0                           ; Changes data
        RTS
;                                                   ;
                    .END                            ;
```

2.24 Sorting

2.24.1 Outline

This program sorts data consisting of a specified number of bytes (sizes in bytes) in ascending order.

Subroutine name : SORT	ROM capacity : 28 bytes
Interrupt during execution: Accepted	Number of stacks used : None

Register/memory	Input	Output	Usage condition
R0L	Number of compare bytes - 1	Indeterminate	Compare bytes counter
R0H	—	Indeterminate	Compare bytes counter
R1L	—	Indeterminate	Register used for change
R1H	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	Start address	Indeterminate	Compared address
A1	—	Indeterminate	Compare address
Z flag	—	Sorting succeeded/failed	←

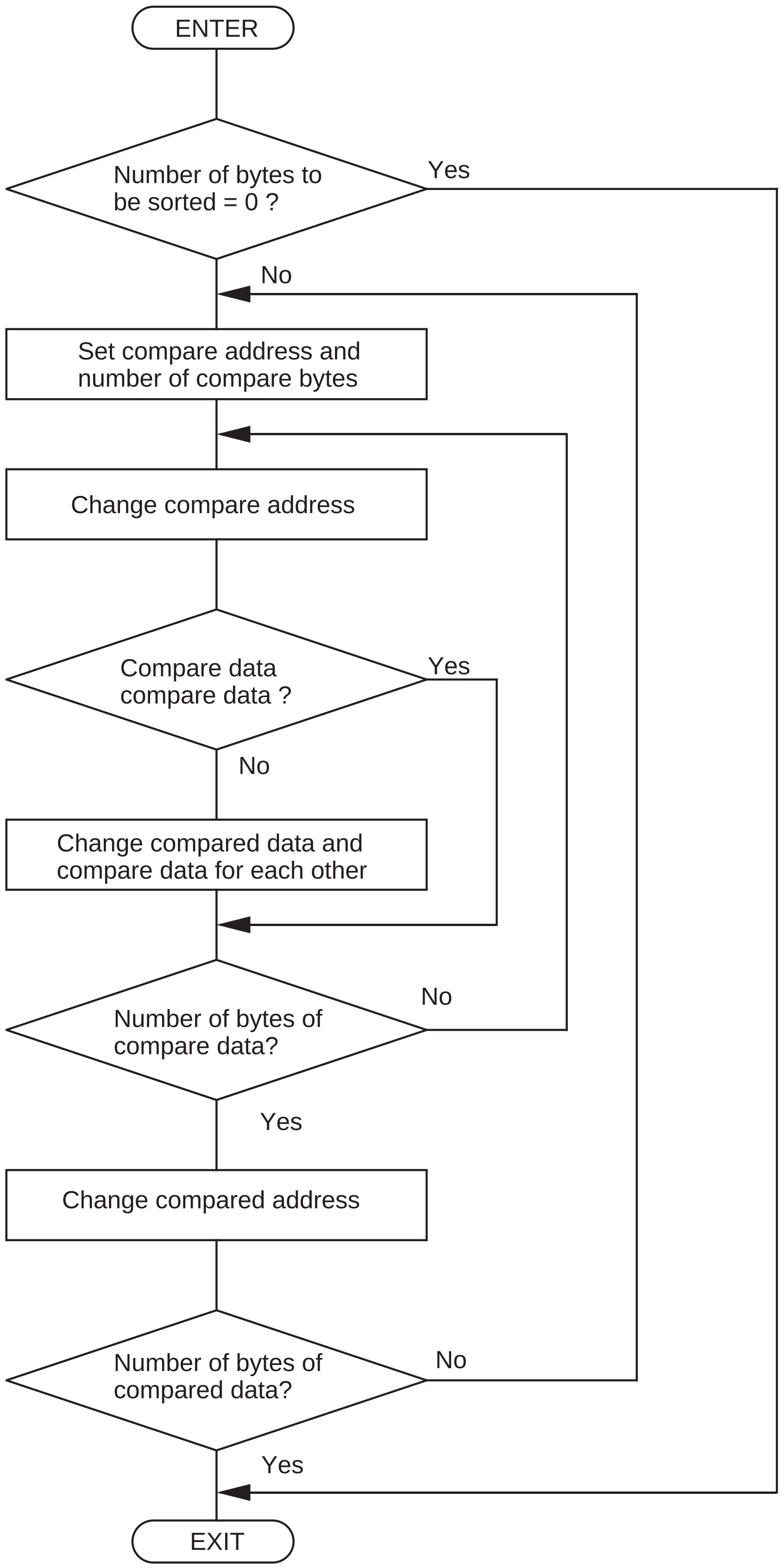
Usage precautions

The number of bytes that can be specified is 2 to 256 bytes.

2.24.2 Explanation

This program sorts data consisting of a specified number of bytes (sizes in bytes) in ascending order beginning with a specified address. Set the “number of bytes to be compared - 1” in R0L and the start address of the data in A0.

2.24.3 Flowchart



2.24.4 Program List

```
;*****************************************************************************************
;                                                                                       *
;          M16C Program Collection No. 24                                               *
;          CPU                 : M16C                                                   *
;                                                                                       *
;*****************************************************************************************
VromTOP         .EQU    0F0000H                   ; Declares start address of ROM
;                                                 ;
;==========================================================================
;          Title: Sorting
;          Outline: Sorts given data (2 to 256 bytes) in ascending order
;          Input:   ----------------------------->  Output:
;          R0L      (Compare bytes - 1)             R0L      (Indeterminate)
;          R0H      ( )                             R0H      (Indeterminate)
;          R1L      ( )                             R1L      (Indeterminate)
;          R1H      ( )                             R1H      (Unused)
;          R2       ( )                             R2       (Unused)
;          R3       ( )                             R3       (Unused)
;          A0       (Start address)                 A0       (Indeterminate)
;          A1       ( )                             A1       (Indeterminate)
;          Stack amount used: None
;          Notes: Success or failure of sorting is returned by Z flag
;==========================================================================
                .SECTION        PROGRAM,CODE
                .ORG            VromTOP           ; ROM area
SORT:                                             ;
          CMP.B         #0,R0L                    ;
          JEQ           SORT_EXIT                 ; --> Number of compare bytes not set
SORT_10:                                          ;
          MOV.B         R0L,R0H                   ; Sets number of compare bytes
          MOV.W         A0,A1                     ; Sets compare address
SORT_20:                                          ;
          INC.W         A1                        ; Changes compare address
          CMP.B         [A0],[A1]                 ; Compare data to see if large or small
          JGEU          SORT_30                   ; --> Sorting unnecessary
          MOV.B         [A0],R1L                  ; Changes compared and compare data for each other
          XCHG.B        R1L,[A1]                  ;
          MOV.B         R1L,[A0]                  ;
SORT_30:                                          ;
          ADJNZ.B       #-1,R0H,SORT_20           ; --> Looped for compare data
          INC.W         A0                        ; Changes compared address
          ADJNZ.B       #-1,R0L,SORT_10           ; --> Looped for compared data
          FCLR          Z                         ; Sorting completed
SORT_EXIT:                                        ;
          RTS                                     ;
;                                                 ;
                .END                              ;
```

2.25 Searching Array

2.25.1 Outline

This program searches for specified data from a two-dimensional array of a given size (maximum 255 x 255 bytes).

Subroutine name : ARRANGE	ROM capacity : 37 bytes
Interrupt during execution: Accepted	Number of stacks used : 2 bytes

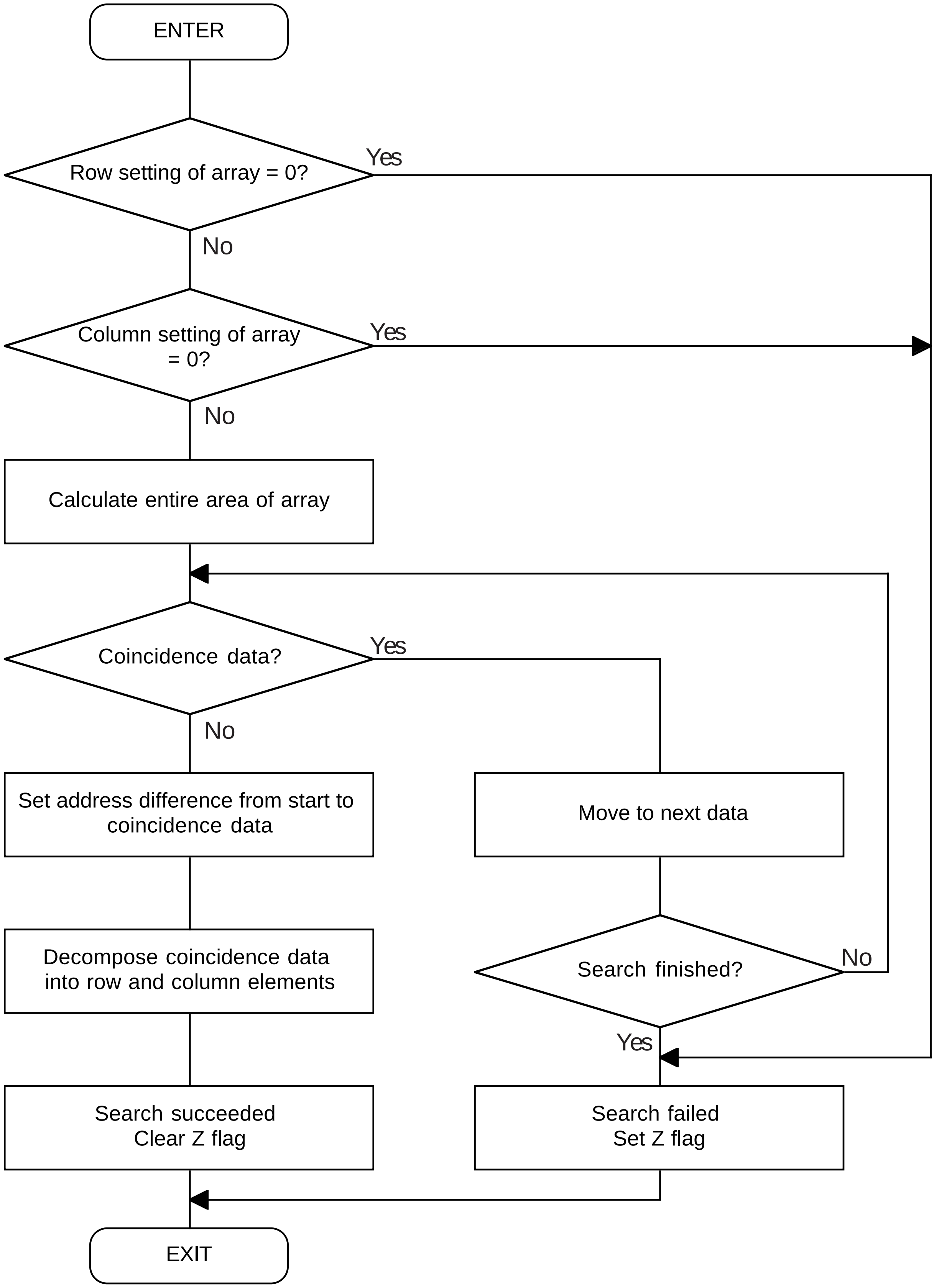
Register/memory	Input	Output	Usage condition
R0L	Row size of array	Row element of coincidence data	←
R0H	Column size of array	Column element of coincidence data	←
R1L	Search data	Does not change	←
R1H	—	Indeterminate	Used to save column size
R2	—	—	Unused
R3	—	—	Unused
A0	Start address of array	Address of coincidence data	←
A1	—	Indeterminate	Used to save start address
Z flag	—	Sorting succeeded/failed	←
Usage precautions			

2.25.2 Explanation

This program searches for specified data from a two-dimensional array of a given size (maximum 255 x 255 bytes). Set the start address of the array in A0, the row size of the array in R0L, the column size of the array in R0H, and the search data in R1L. The address, the row element, and the column element of the coincidence data are output to A0, R0L, and R0H, respectively. Information on whether the search has succeeded or failed is output to the Z flag.
In this program, the overall size of the array is calculated, the specified data is searched from the entire array region, and a difference from the start address to the search address is obtained before decomposing the coincidence data into row and column elements.

Z	Meaning
0	Search succeeded
1	Search failed (no coincidence data found, row setting of array = 0, or column setting of array = 0)

2.25.3 Flowchart



2.25.4 Program List

```
;*******************************************************************************
;                                                                             *
;           M16C Program Collection No. 25                                    *
;           CPU                 : M16C                                        *
;                                                                             *
;*******************************************************************************
VromTOP                 .EQU    0F0000H                 ; Declares start address of ROM
;                                                       ;
;===============================================================================
;           Title: Searching array
;           Outline: Searches for data from two-dimensional array of given size (within 255 x 255 bytes)
;           Input:  ----------------------------->  Output:
;           R0L     (Row size of array)          R0L     (Row element of coincidence data)
;           R0H     (Column size of array)       R0H     (Column element of coincidence data)
;           R1L     (Search data)                R1L     (Does not change)
;           R1H     ( )                          R1H     (Indeterminate)
;           R2      ( )                          R2      (Unused)
;           R3      ( )                          R3      (Unused)
;           A0      (Start address of array)     A0      (Address of coincidence data)
;           A1      ( )                          A1      (Indeterminate)
;           Stack amount used: 2 bytes
;           Notes: Success or failure of search is returned by Z flag
;===============================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
ARRANGE:                                                ;
            CMP.B       #0,R0L                          ;
            JEQ         ARRANGE_NG                      ; --> No rows of array are set
            MOV.B       R0H,R1H                         ; Saves columns
            JEQ         ARRANGE_NG                      ; --> No columns of array are set
            MOV.W       A0,A1                           ;
            MULU.B      R0H,R0L                         ; Calculates array size
ARRANGE_10:                                             ;
            CMP.B       R1L,[A0]                        ;
            JEQ         ARRANGE_20                      ; --> Coincidence data found
            INC.W       A0                              ;
            ADJNZ.W     #-1,R0,ARRANGE_10               ; --> Checks next data
ARRANGE_NG:                                             ;
            FSET        Z                               ; Search failed
            JMP         ARRANGE_EXIT                    ;
ARRANGE_20:                                             ;
            PUSH.W      A0                              ; Saves address of coincidence data
            SUB.W       A1,A0                           ; Creates address difference from start
                                                        ;  to coincidence data
            MOV.W       A0,R0                           ;
            DIVU.B      R1H                             ; Decomposes coincidence data into
                                                        ; row and column elements
            INC.B       R0L                             ; Corrects rows
            INC.B       R0H                             ; Corrects columns
            POP.W       A0                              ; Restores address of coincidence data
            FCLR        Z                               ; Search succeeded
ARRANGE_EXIT:                                           ;
            RTS                                         ;
;                                                       ;
                        .END                            ;
```

2.26 Converting from Lowercase Alphabet to Uppercase Alphabet

2.26.1 Outline

This program converts a lowercase English alphabet in ASCII code into an uppercase English alphabet in ASCII code.

Subroutine name : TOUPPER	ROM capacity : 16 bytes
Interrupt during execution: Accepted	Number of stacks used : None

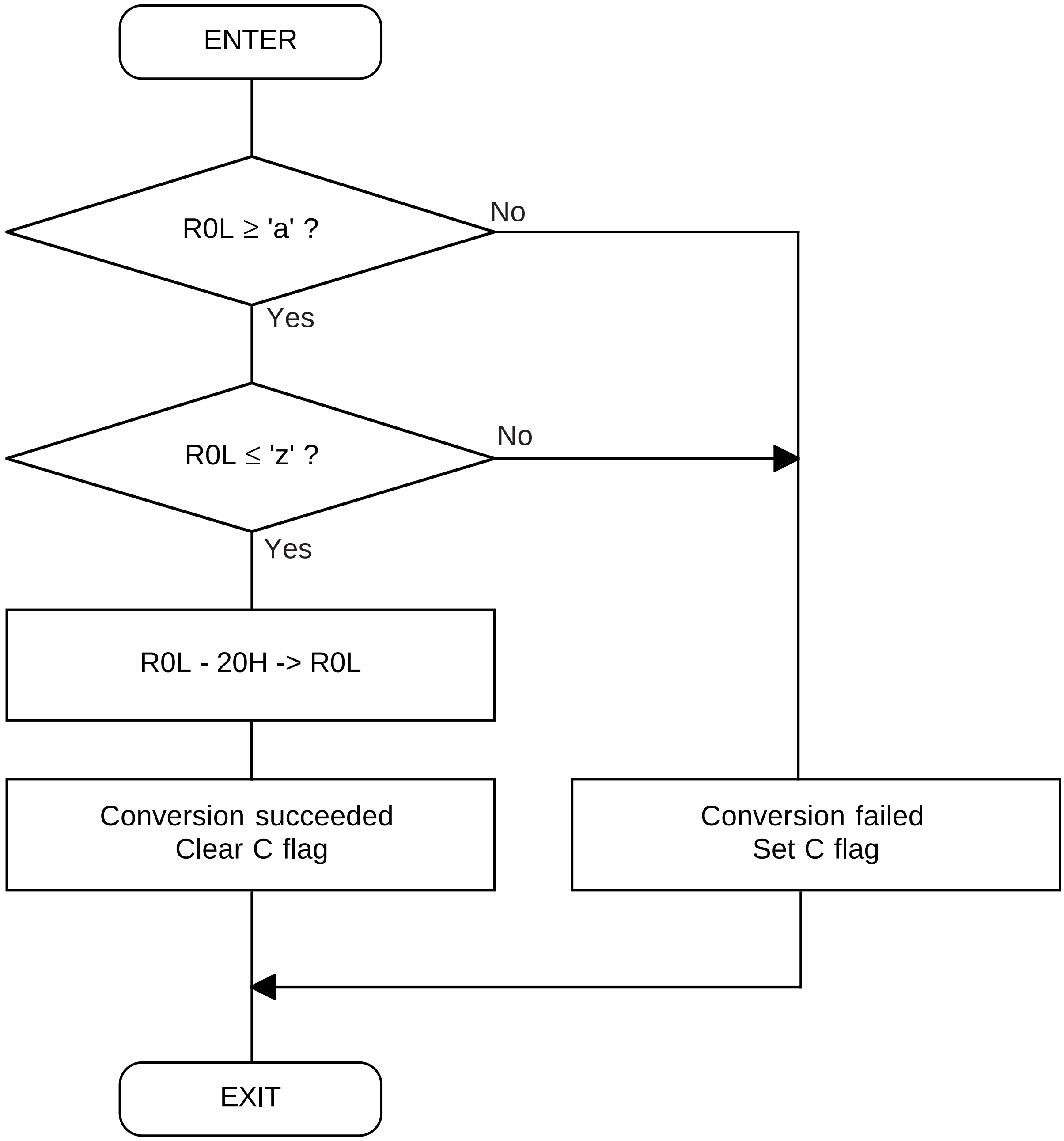
Register/memory	Input	Output	Usage condition
R0L	Lowercase alphabet (ASCII)	Uppercase alphabet (ASCII)	←
R0H	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Conversion information	←
Usage precautions			

2.26.2 Explanation

This program converts a lowercase English alphabet in ASCII code into an uppercase English alphabet in ASCII code. Set the lowercase English alphabet in ASCII code in R0L. The converted uppercase English alphabet in ASCII code is output to R0L. Conversion information is output to the C flag.

C	Meaning
0	Lowercase alphabet converted into uppercase alphabet
1	No converted because inconvertible code was input

2.26.3 Flowchart



2.26.4 Program List

```
;*****************************************************************************************
;                                                                                       *
;           M16C Program Collection No. 26                                              *
;           CPU                 : M16C                                                  *
;                                                                                       *
;*****************************************************************************************
VromTOP                 .EQU            0F0000H         ;  Declares start address of ROM

;================================================================
;           Title: Converting ASCII code lowercase alphabet into uppercase alphabet
;           Contents of processing:
;                   The ASCII code input in R0L is converted from a lowercase English alphabet into an
                    uppercase English alphabet and the result is returned to R0L. No conversion is
                    performed if any code is input in R0L that is not a lowercase English alphabet.
;           Procedure: (1) Input ASCII code in R0L.
;                      (2) Call the subroutine.
;                      (3) Converted ASCII code is loaded into R0L.
;           Result:   The C flag is cleared to 0 when the code was converted from a lowercase alphabet
                      into an uppercase alphabet.  The C flag is set to 1 when the code was not converted.
;           Input:    ------------------------------>  Output:
;           R0L       (ASCII code)                   R0L       (ASCII code)
;           R0H       ( )                            R0H       (Unused)
;           R1        ( )                            R1        (Unused)
;           R2        ( )                            R2        (Unused)
;           R3        ( )                            R3        (Unused)
;           A0        ( )                            A0        (Unused)
;           A1        ( )                            A1        (Unused)
;           Stack amount used: None
;================================================================
                        .SECTION        PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
TOUPPER:                                                ;
            CMP.B       #'a',R0L                        ; Lowercase alphabet ‘a’ or above?
            JLTU        TOUPNON                         ; --> no (not converted)
            CMP.B       #'z',R0L                        ; Lowercase alphabet ‘z’ or below?
            JGTU        TOUPNON                         ; --> no (not converted)
            SUB.B       #20H,R0L                        ; Converts from lowercase alphabet into
                                                        ; uppercase alphabet
            FCLR        C                               ; Sets “converted” information
            RTS                                         ;
TOUPNON:                                                ;
            FSET        C                               ; Sets “not-converted” information
            RTS                                         ;
;                                                       ;
            .END                                        ;
```

2.27 Converting from Uppercase Alphabet to Lowercase Alphabet

2.27.1 Outline

This program converts an uppercase English alphabet in ASCII code into a lowercase English alphabet in ASCII code.

Subroutine name : TOLOWER	ROM capacity : 16 bytes
Interrupt during execution: Accepted	Number of stacks used : None

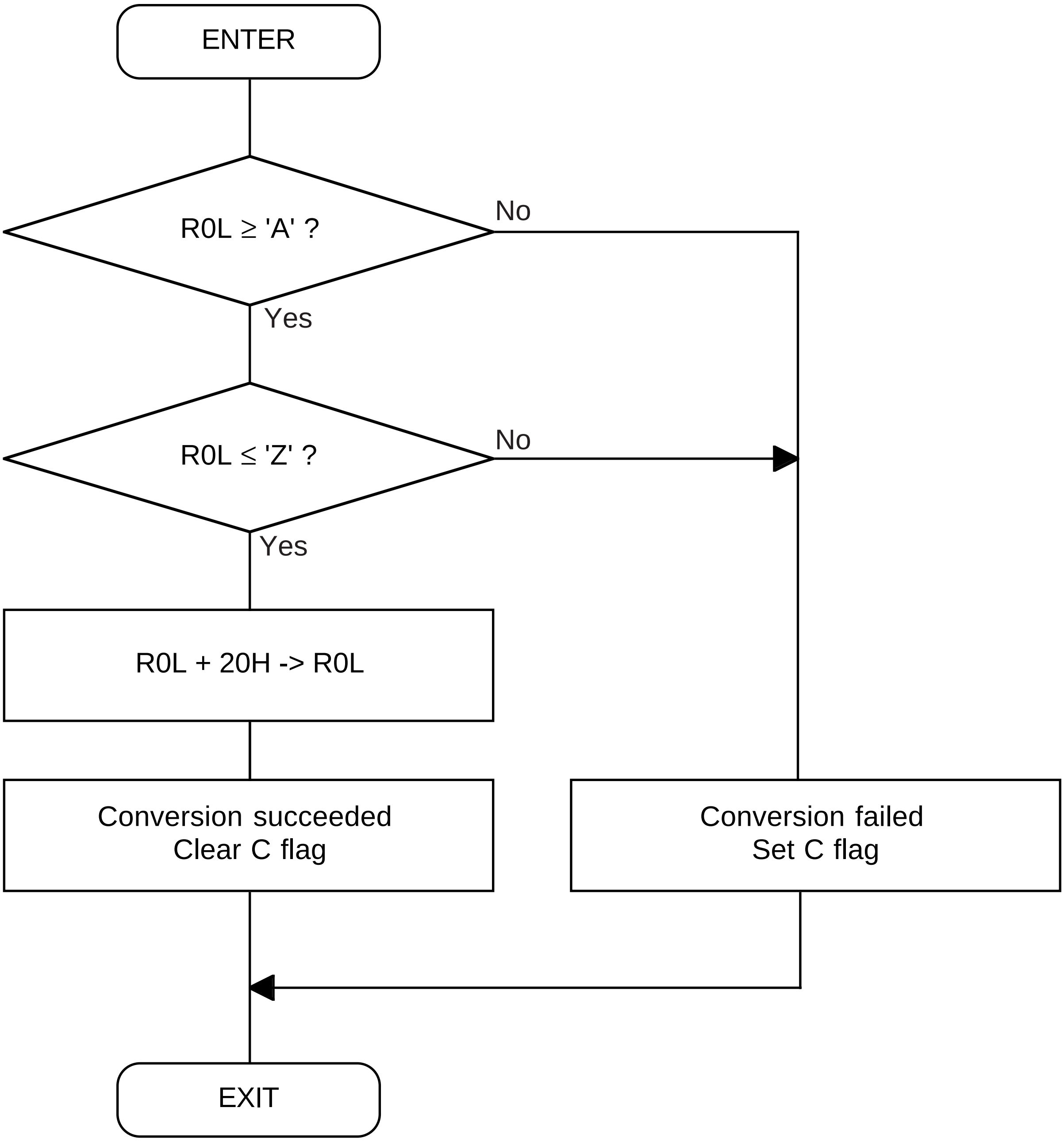
Register/memory	Input	Output	Usage condition
R0L	Uppercase alphabet (ASCII)	Lowercase alphabet (ASCII)	←
R0H	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Conversion information	←
Usage precautions			

2.27.2 Explanation

This program converts an uppercase English alphabet in ASCII code into a lowercase English alphabet in ASCII code. Set the uppercase English alphabet in ASCII code in R0L. The converted lowercase English alphabet in ASCII code is output to R0L. Conversion information is output to the C flag.

C	Meaning
0	Uppercase alphabet converted into lowercase alphabet
1	No converted because inconvertible code was input

2.27.3 Flowchart



2.27.4 Program List

```
;*********************************************************************************************
;                                                                                            *
;        M16C Program Collection No. 27                                                      *
;        CPU              : M16C                                                             *
;                                                                                            *
;*********************************************************************************************
VromTOP              .EQU          0F0000H        ;  Declares start address of ROM
;
;===============================================================
;        Title: Converting ASCII code uppercase alphabet into lowercase alphabet
;        Contents of processing:
;                 The ASCII code input in R0L is converted from an uppercase English alphabet into
;                 a lowercase English alphabet and the result is returned to R0L. No conversion is
;                 performed if any code is input in R0L that is not an uppercase English alphabet.
;        Procedure: (1) Input ASCII code in R0L.
;                   (2) Call the subroutine.
;                   (3) Converted ASCII code is loaded into R0L.
;        Result:  The C flag is cleare d to 0 when the code was converted from a uppercase alphabet
;        ;                 into an lowercase alphabet.  The C flag is set to 1 when the code was not
converted.
;        Input:   ------------------------------>  Output:
;        R0L      (ASCII code)                     R0L      (ASCII code)
;        R0H      ( )                              R0H      (Unused)
;        R1       ( )                              R1       (Unused)
;        R2       ( )                              R2       (Unused)
;        R3       ( )                              R3       (Unused)
;        A0       ( )                              A0       (Unused)
;        A1       ( )                              A1       (Unused)
;        Stack amount used: None
;===============================================================
                     .SECTION      PROGRAM,CODE
                     .ORG          VromTOP        ; ROM area
TOLOWER:                                          ;
         CMP.B       #'A',R0L                     ; Uppercase alphabet 'A' or above?
         JLTU        TOLOWNON                     ; --> no (not converted)
         CMP.B       #'Z',R0L                     ; Uppercase alphabet 'Z' or below?
         JGTU        TOLOWNON                     ; --> no (not converted)
         ADD.B       #20H,R0L                     ; Converts from uppercase alphabet
                                                  ; into lowercase alphabet
         FCLR        C                            ; Sets "converted" information
         RTS                                      ;
TOLOWNON:                                         ;
         FSET        C                            ; Sets "not-converted" information
         RTS                                      ;
;                                                 ;
         .END                                     ;
```

2.28 Converting from ASCII to Hexadecimal Data

2.28.1 Outline

This program converts ASCII code into hexadecimal data.

Subroutine name : ATOH	ROM capacity : 42 bytes
Interrupt during execution: Accepted	Number of stacks used : None

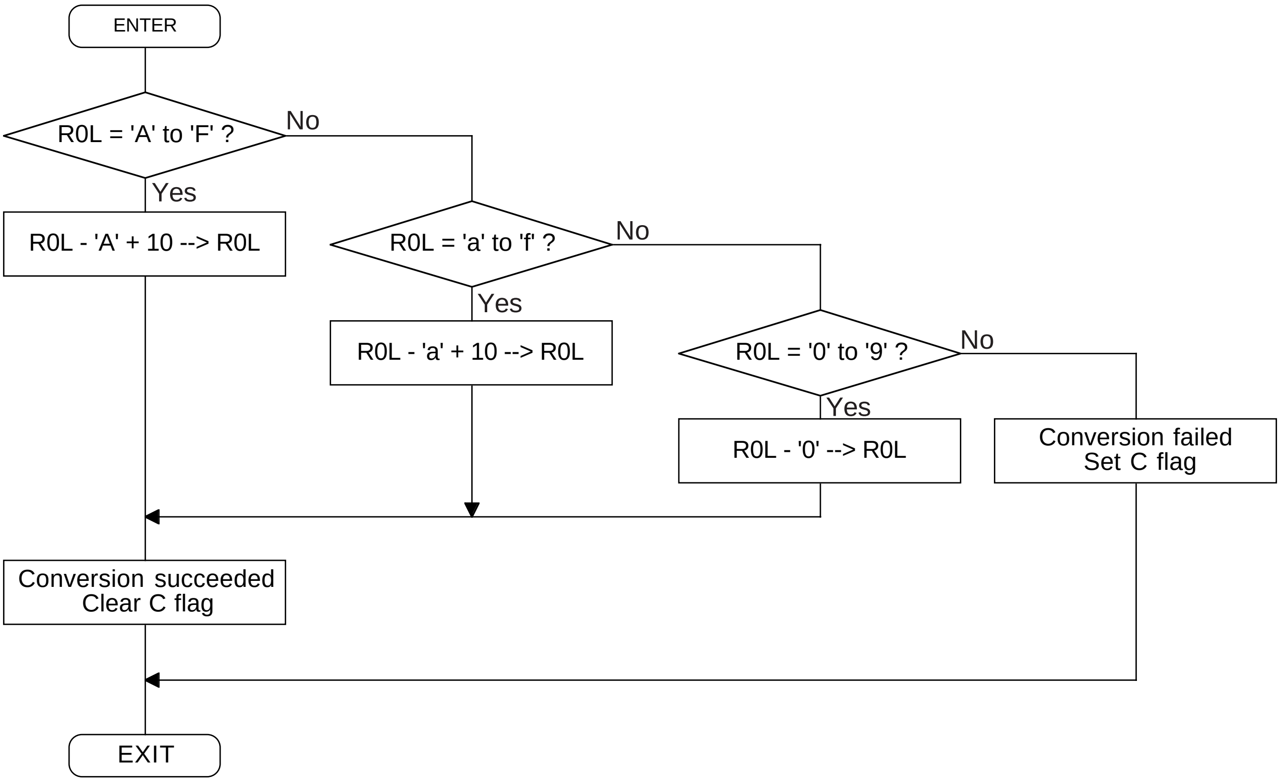
Register/memory	Input	Output	Usage condition
R0L	ASCII code	Hexadecimal	←
R0H	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Conversion information	←
Usage precautions			

2.28.2 Explanation

This program converts ASCII code into hexadecimal data. The ASCII code that can be converted are numbers from ‘0’ to ‘9’ and alphabets from ‘a’ to ‘f’ and ‘A’ to ‘F’. Set ASCII code in R0L. The converted hexadecimal data is output to R0L. Conversion information is output to the C flag.

C	Meaning
0	ASCII converted into hexadecimal
1	Not converted because inconvertible code was input

2.28.3 Flowchart



2.28.4 Program List

```
;****************************************************************************
;                                                                          *
;           M16C Program Collection No. 28                                 *
;           CPU              : M16C                                        *
;                                                                          *
;****************************************************************************
VromTOP              .EQU          0F0000H        ;  Declares start address of ROM

;==========================================================================
;           Title: Converting ASCII code into hexadecimal
;           Contents of processing:
;                      The ASCII code input in R0L is converted into hexadecimal data, which is returned
;                      to R0L. The valid ASCII code are 0 to 9, A to F, and a to f. No conversion is per-
;                      formed if invalid code is input.
;           Procedure: (1) Input ASCII code in R0L.
;                         (2) Call the subroutine.
;                         (3)  The converted hexadecimal data is loaded into R0L.
;           Result:    When converted into hexadecimal data, the C flag is cleared to 0.  If not converted
;                      into hexadecimal data, i.e., if any code other than 0 to 9, A to F, or a to f was input,
;                      the C flag is set to 1.
;           Input:     ----------------------------->   Output:
;           R0L        (ASCII code)                     R0L      (Hexadecimal)
;           R0H        ( )                              R0H      (Unused)
;           R1         ( )                              R1       (Unused)
;           R2         ( )                              R2       (Unused)
;           R3         ( )                              R3       (Unused)
;           A0         ( )                              A0       (Unused)
;           A1         ( )                              A1       (Unused)
;           Stack amount used: None
;==========================================================================
                     .SECTION      PROGRAM,CODE
                     .ORG          VromTOP        ; ROM area
ATOH:                                             ;
          CMP.B      #'a',R0L                     ; ‘a’ or above?
          JLTU       ATOH10                       ; --> no
          CMP.B      #'f',R0L                     ;  ‘f’ or below?
          JGTU       ATOH_ERR                     ; --> no (not converted)
          SUB.B      #(61H-10),R0L                ; SUB.B  #'a'-10,R0L
          FCLR       C                            ; Sets “converted” information
          RTS                                     ;
ATOH10:                                           ;
          CMP.B      #'A',R0L                     ; ‘A’ or above?
          JLTU       ATOH20                       ; --> no
          CMP.B      #'F',R0L                     ; ‘F’ or below?
          JGTU       ATOH_ERR                     ; --> no (not converted)
          SUB.B      #(41H-10),R0L                ; SUB.B  #'A'-10,R0L
          FCLR       C                            ; Sets “converted” information
          RTS                                     ;
ATOH20:                                           ;
          CMP.B      #'0',R0L                     ; ‘0’ or above?
          JLTU       ATOH_ERR                     ; --> no (not converted)
          CMP.B      #'9',R0L                     ; ‘9’ or below?
          JGTU       ATOH_ERR                     ; --> no (not converted)
          AND.B      #0FH,R0L                     ;
          FCLR       C                            ; Sets “converted” information
          RTS                                     ;
ATOH_ERR:                                         ;
          FSET       C                            ; Sets “not-converted” information
          RTS                                     ;
;                                                 ;
          .END                                    ;
```

2.29 Converting from Hexadecimal Data to ASCII Code

2.29.1 Outline

This program converts hexadecimal data into ASCII code.

Subroutine name : HTOA	ROM capacity : 21 bytes
Interrupt during execution: Accepted	Number of stacks used : None

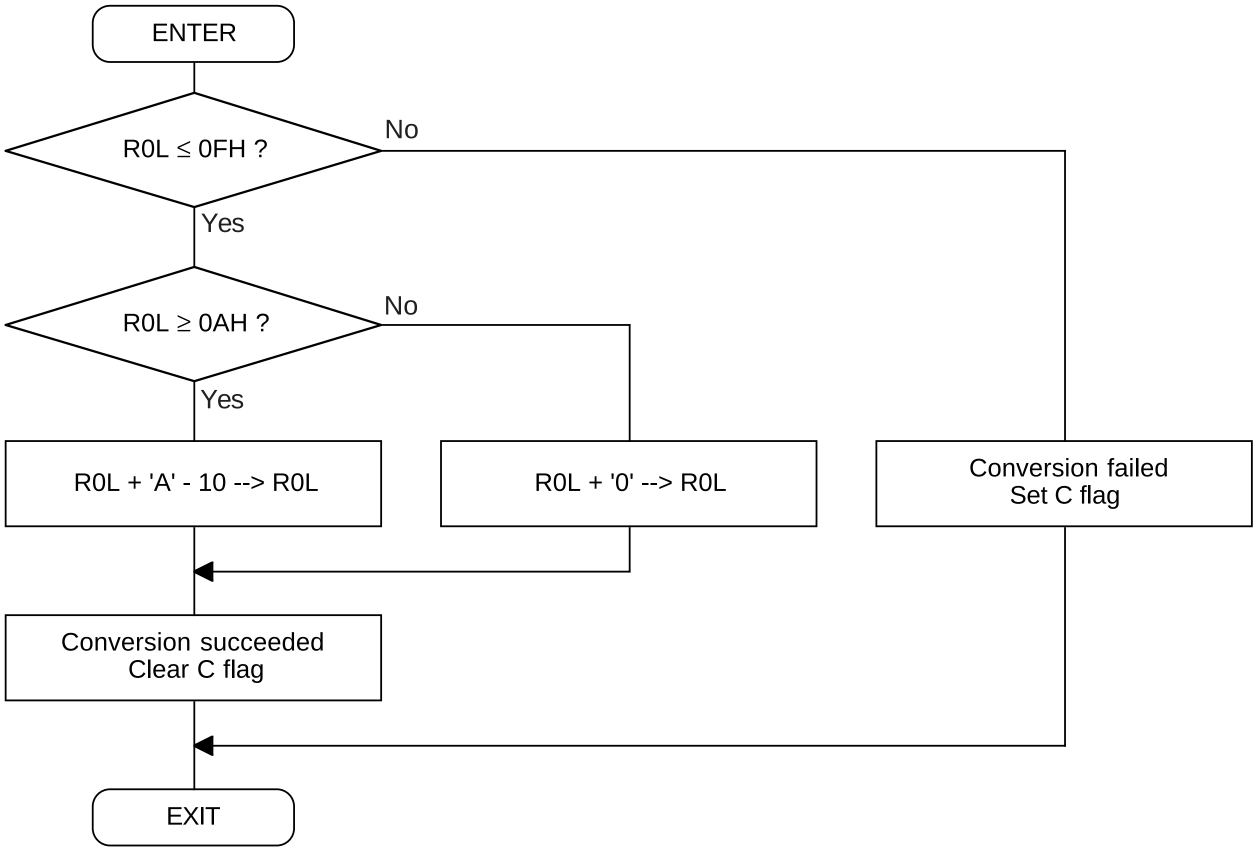
Register/memory	Input	Output	Usage condition
R0L	Hexadecimal	ASCII code	←
R0H	—	—	Unused
R1	—	—	Unused
R2	—	—	Unused
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused
C flag	—	Converted or not	←
Usage precautions			

2.29.2 Explanation

This program converts hexadecimal data into ASCII code. The hexadecimal data that can be converted are from “00H” to “0FH.” The converted ASCII code are numbers from ‘0’ to ‘9’ and alphabets from ‘A’ to ‘F’. Set the hexadecimal data in R0L. The converted ASCII code is output to R0L. Conversion information is output to the C flag.

C	Meaning
0	Hexadecimal converted into ASCII code
1	Not converted because inconvertible code was input

2.29.3 Flowchart



2.29.4 Program List

```
;*************************************************************************************
;                                                                                   *
;       M16C Program Collection No. 29                                              *
;       CPU             : M16C                                                      *
;                                                                                   *
;*************************************************************************************
VromTOP                 .EQU            0F0000H         ; Declares start address of ROM

;==============================================================
;       Title: Converting hexadecimal into ASCII code
;       Contents of processing:
;               The hexadecimal data input in R0L is converted into ASCII code, which is returned
;               to R0L. The valid hexadecimal data are 00 to 0F. 0A to 0F are converted into 'A' to
;               'F.'  No conversion is performed if invalid code is input.
;       Procedure: (1) Input hexadecimal data in R0L.
;                  (2) Call the subroutine.
;                  (3) The converted hexadecimal data is loaded into R0L.
;       Result:  When converted into ASCII code, the C flag is cleared to 0.  If not converted into
;               ASCII code, i.e., if any hexadecimal data other than 00 to 0F was input, the C flag is
;               set to 1.
;       Input:  ------------------------------>  Output:
;       R0L     (Hexadecimal)                   R0L     (ASCII code)
;       R0H     ( )                             R0H     (Unused)
;       R1      ( )                             R1      (Unused)
;       R2      ( )                             R2      (Unused)
;       R3      ( )                             R3      (Unused)
;       A0      ( )                             A0      (Unused)
;       A1      ( )                             A1      (Unused)
;       Stack amount used: None
;==============================================================
                        SECTION         PROGRAM,CODE
                        .ORG            VromTOP         ; ROM area
HTOA:                                                   ;
        CMP.B           #0FH,R0L                        ; 0F or below?
        JGTU            HTOA_ERR                        ; --> No(not converted)
        CMP.B           #0AH,R0L                        ; 0A or above?
        JGEU            HTOA10                          ; --> Yes (A to F set)
        OR.B            #'0',R0L                        ;
        FCLR            C                               ; Sets "converted" information
        RTS                                             ;
HTOA10:                                                 ;
        ADD.B           #(41H-10),R0L                   ; ADD.B  #'A'-10,R0L
        FCLR            C                               ; Sets "converted" information
        RTS                                             ;
HTOA_ERR:                                               ;
        FSET            C                               ; Sets "not-converted" information
        RTS                                             ;
;                                                       ;
        .END                                            ;
```

2.30 Example for Initial Setting Assembler

2.30.1 Outline

This program is an example of initial settings accomplished by using the directive commands of the assembler.

2.30.2 Explanation

The program shown here consists of the following:

(1) Map file information output
(2) Global symbol name specification
(3) Numeric symbol definition
(4) RAM area allocation
(5) Bit symbol definition
(6) Initial setup program
- Interrupt stack pointer setting
- FB register setting
- SB register setting
- INTB register setting
- RAM clear

(7) Main program
(8) Peripheral I/O interrupt vector table
(9) Nonmaskable interrupt fixed vector table

The following shows the range of the FB and SB relative addresses in this program.

FB	380H to 47FH - 128 ↑ 400H ↓ + 127
SB	480H to 57FH 400H ↓ + 255

2.30.3 Program List

```
;*********************************************************************************************
;                                                                                           *
;       M16C Program Collection No. 30                                                      *
;       CPU             : M16C                                                              *
;                                                                                           *
;*********************************************************************************************
;=================================================================================
;       Title: Initial settings using assembler's directive commands
;       Outline:
;               (1)  Assemble control
;               (2)  Address control
;               (3)  Link control
;               (4)  List control
;               (5)  Branch instruction optimization control
;       Notes:
;=================================================================================
;/////////////////////////////////////////////////////////////////////////////
;       Map file information output
;/////////////////////////////////////////////////////////////////////////////
                .VER    'Ver1.02'               ; 'Ver1.02' is output when generating map file
;
;/////////////////////////////////////////////////////////////////////////////
;       Global symbol name specification
;/////////////////////////////////////////////////////////////////////////////
                                                ; [Global symbol specification]
                .GLB    RUTINE                  ; Externlly referenced symbol
                .GLB    MAIN                    ; Public symbol
;
                                                ; [Global bit symbol specification]
                .BTGLB  P2_4                    ; Externally referenced symbol
                .BTGLB  P0_7                    ; Public symbol
;
;/////////////////////////////////////////////////////////////////////////////
;       Numeric symbol definition
;/////////////////////////////////////////////////////////////////////////////
VramTOP         .EQU    000400H                 ; Declares start address of RAM
VramEND         .EQU    002BFFH                 ; Declares last address of RAM
VIstack         .EQU    002C00H                 ; Interrupt stack pointer
VproTOP         .EQU    0F0000H                 ; Declares start address of program
Vintbase.EQU    .EQU    0FFD00H                 ; Declares start address of variable vector table
Vvector         .EQU    0FFFDCH                 ; Declares fixed interrupt vector address
;
CNT125ms        .EQU    125                     ; Sets 125 in CNT125ms
;
AUTOchar        .EQU    -8                      ; Sets -8 in AUTOchar
;
                .FORM   45,160                  ; [List output control instruction]
                                                ; Specifies 45 lines, 160 columns per page of list file
                .LIST   ON                      ; [List output control]
                                                ; Outputs assembler list
                .PAGE   'RAM'                   ; [List page break and title specification]
                .SECTION        MEMORY,DATA     ; [Section name specification]
                                                ; Declares DATA attribute section of section name "MEMORY"
                .ORG            VramTOP         ; [Absolute address setting]
                                                ; Sets location to 100H
```

```
;////////////////////////////////////////////////////////////////////////////////
;           RAM area allocation
;////////////////////////////////////////////////////////////////////////////////
                                                   ; [RAM area 1-byte allocation]
CHAR:           .BLKB           10                 ; Allocates 10-byte area
;
                                                   ; [RAM area 2-byte allocation]
SHORT:          .BLKW           10                 ; Allocates 20-byte area
;
                                                   ; [RAM area 3-byte allocation]
ADDR:           .BLKA           10                 ; Allocates 30-byte area
;
                                                   ; [RAM area 4-byte allocation]
LONG:           .BLKL           10                 ; Allocates 40-byte area
;
                                                   ; [Single-precision, floating-point RAM area allocation]
SFLOAT:         .BLKF           10                 ; Allocates 40-byte area
;
                                                   ; [Double-precision, floating-point RAM area allocation]
DFLOAT:         .BLKD           10                 ; Allocates 80-byte area
;
CHECK:          .BLKW           10
;
;////////////////////////////////////////////////////////////////////////////////
;           Bit symbol definition
;////////////////////////////////////////////////////////////////////////////////
BIT4            .BTEQU          4,CHAR             ; Sets bit 4 of displacement CHAR to BIT4
MSB             .BTEQU          15,SHORT           ; Sets bit 15 of displacement SHORT to MSB
P0_7            .BTEQU          7,3E0H             ; Sets bit 7 at address 3E0 to P0_7
;
                .SECTION        PROG,CODE          ; Declares CODE attribute section of section name “PROG”
                .ORG            VproTOP            ; Sets location to F0000H
                .OPTJ           OFF                ; [Branch instruction optimize specification]
                                                   ; Does not optimize branch instruction after this line
                .FB             VramTOP            ; [Assumption of FB register value]
                                                   ; Assumes 400H for FB register value
                .SB             VramTOP+80H        ; [Assumption of SB register value]
                                                   ; Assumes 480H for SB register value
                .FBSYM          SHORT              ;
                .SBSYM          CHECK              ;
;=======================================
;           Program start
;=======================================
RESET:
          LDC          #VIstack,ISP                    ; Sets interrupt stack pointer
;
          LDC          #VramTOP,FB                     ; Sets frame base register
          LDC          #VramTOP+80H,SB                 ; Sets static base register
          LDINTB       #Vintbase                       ; Sets interrupt table register
;
          MOV.W        #0,R0                           ; Sets store data (0)
          MOV.W        #((VramEND+1)-VramTOP)/2,R3     ; Sets number of transfers performed
          MOV.W        #VramTOP,A1                     ; Sets address where to start storing
          SSTR.W                                       ; Executes clearing of RAM
;
          FSET         I                               ; Enables interrupt
;
```

```
;===============================================
;           Main program
;===============================================
MAIN:
            MOV.W           #1234H,SHORT
;
;
            MOV.W           #5678H,CHECK
;
;
            JSR             RUTINE
            BSET            P2_4
                            :
                            :

                        .PAGE   'VECTOR'
                        .SECTION        UINTER,ROMDATA          ; Declares FOMDATA attribute section
                                                                ; of section name "UINTER"
                        .ORG            Vintbase                ; Sets location to FFD00H
;===============================================
;           Peripheral I/O interrupt vector table
;===============================================
                        .LWORD          NOTUSE                  ; Software interrupt number 0
                        .LWORD          NOTUSE                  ; Software interrupt number 1


                        .SECTION        INTER,ROMDATA           ; Declares FOMDATA attribute section
                                                                ; of section name "INTER"
                        .ORG            Vvector                 ; Sets location to FFFDCH
;===============================================
;           Nonmaskable interrupt fixed vector table
;===============================================
                        .LWORD          NOTUSE                  ; FFFDC to F Undefined instruction
                        .LWORD          NOTUSE                  ; FFFE0 to 3  Overflow
                        .LWORD          NOTUSE                  ; FFFE4 to 7  BRK instruction
                        .LWORD          NOTUSE                  ; FFFE8 to B  Address coincidence
                        .LWORD          NOTUSE                  ; FFFEC to F  Single stepping
                        .LWORD          NOTUSE                  ; FFFF0 to 3  Watchdog timer
                        .LWORD          NOTUSE                  ; FFFF4 to 7  Debugger
                        .LWORD          NOTUSE                  ; FFFF8 to B  NMI
                        .LWORD          RESET                   ; FFFFC to F  Reset
;///////////////////////////////////////////////////////////////////////////////////////////
;           End of assemble direction
;///////////////////////////////////////////////////////////////////////////////////////////
            .END
;
```

2.31 Special Page Subroutine

2.31.1 Outline

This program is an example for using a special subroutine call.

2.31.2 Explanation

The program branches to a subroutine at an address that is the address set in one of the special page vector tables (in 2 bytes each) plus F0000H. The area in which control can branch to a subroutine is from address F0000H to address FFFFFH.
The special page vector tables are located in an area ranging from address FFE00H to address FFFDBH. The special page number at address FFE00H is 255 and that at address FFFDAH is 18. A label can be used in place of a special page number.
Shown in this program are an example where labels are used for special page numbers 255 and 18 and an example where a special page number (254) is used directly.

2.31.3 Program List

```
;*************************************************************************************
;                                                                                   *
;          M16C Program Collection No. 31                                           *
;          CPU               : M16C                                                 *
;                                                                                   *
;*************************************************************************************
VromTOP              .EQU          0F0000H        ;  Declares start address of ROM
;
;==========================================================================
;          Title: Special page subroutine call
;          Outline: Description example of special page subroutine call
;          Input:    ---------------------------->   Output:
;          R0        ( )                             R0        ( )
;          R1        ( )                             R1        ( )
;          R2        ( )                             R2        ( )
;          R3        ( )                             R3        ( )
;          A0        ( )                             A0        ( )
;          A1        ( )                             A1        ( )
;          Stack amount used: 3 bytes
;==========================================================================
                     .SECTION      PROGRAM,CODE
                     .ORG          VromTOP        ; ROM area
MAIN:
          JSRS       \SUB1                        ; Branches to subroutine at LABEL_1
          JSRS       #254                         ; Branches to subroutine at LABEL_2
          JSRS       \SUB238                      ; Branches to subroutine at LABEL_238

LABEL_1:
                     Processing
          RTS
LABEL_2:
                     Processing
          RTS
LABEL_238:
                     Processing
          RTS
;
                     .SECTION      SPECIAL,ROMDATA
                     .ORG          0FFE00H        ; Special page area
;-----------------------------------------------------------------------------
;          Special page
;-----------------------------------------------------------------------------
SUB1:                .WORD  LABEL_1&0FFFFH        ; Special page number 255
                     .WORD  LABEL_2&0FFFFH        ; Special page number 254
;
                     .ORG          0FFFDAH
SUB238:              .WORD  LABEL_238&0FFFFH      ; Special page number 18
;
          .END                                    ;
```

2.32 Special Page Jump

2.32.1 Outline

This program is an example for using a special page jump.

2.32.2 Explanation

Control jumps to an address that is set in one of the special page vector tables (in 2 bytes each) plus F0000H. The area within which control can jump is from address F0000H to address FFFFFH.
The special page vector tables are located in an area ranging from address FFE00H to address FFFDBH.
The special page number at address FFE00H is 255 and that at address FFFDAH is 18. A label can be used in place of a special page number.
Shown in this program are an example where labels are used for special page numbers 255 and 18 and an example where a special page number (254) is used directly.

2.32.3 Program List

```
;****************************************************************************************
;                                                                                       *
;        M16C Program Collection No. 32                                                 *
;        CPU              : M16C                                                        *
;                                                                                       *
;****************************************************************************************
VromTOP             .EQU        0F0000H         ;  Declares start address of ROM
;
;=====================================================================
;        Title: Special page subroutine call
;        Outline: Description example of special page subroutine call
;        Input:   ----------------------------->  Output:
;        R0       ( )                             R0       ( )
;        R1       ( )                             R1       ( )
;        R2       ( )                             R2       ( )
;        R3       ( )                             R3       ( )
;        A0       ( )                             A0       ( )
;        A1       ( )                             A1       ( )
;        Stack amount used: None
;=====================================================================
                    .SECTION    PROGRAM,CODE
                    .ORG        VromTOP         ;ROM area
MAIN:
         JMPS       \SUB1                       ; Jumps to LABEL_1
         JMPS       #254                        ; Jumps to LABEL_2
         JMPS       \SUB238                     ; Jumps to LABEL_238
                      |
                      |
                      |
                      |
LABEL_1:
              Processing

LABEL_2:
              Processing

LABEL_238:
              Processing

;
                    .SECTION    SPECIAL,ROMDATA
                    .ORG        0FFE00H         ; Special page area
;------------------------------------------------------------------------------
;        Special page area
;------------------------------------------------------------------------------
SUB1:               .WORD  LABEL_1&0FFFFH       ; Special page number 255
                    .WORD  LABEL_2&0FFFFH       ; Special page number 254

                    .ORG        0FFFDAH
SUB238:             .WORD  LABEL_238&0FFFFH     ; Special page number 18
;
         .END                                   ;
```

2.33 Variable Vector Table

2.33.1 Outline

This program shows an example for setting variable vector tables and an example for using software interrupts.

2.33.2 Explanation

A variable vector table is a 256-byte interrupt vector table whose start address (IntBase) is indicated by the content of the interrupt table register (INTB). The variable vector table in this program has its start address at FE000H. The variable vector table has individual vector tables each comprised of 4 bytes, and each vector table contains the start address of an interrupt routine.
There are software interrupt numbers (0 to 63) available for each vector table. The INT instruction uses these software interrupt numbers. No labels can be used in place of the software interrupt numbers.
Peripheral I/O interrupts are assigned software interrupt numbers 0 to 31. In this program, software interrupt number 21 is used for timer A0 and software interrupt number 22 is used for timer A1.
Software interrupt numbers 32 to 63 are used for software interrupts. This type of interrupt is generated by the INT instruction. Therefore, software interrupts are used in the same way as a subroutine by using the INT instruction. The INT instruction is executed even when interrupts are disabled. After interrupts are disabled (FCLR I) in this program, INT#22 and INT#32 are executed regardless of whether or not the interrupt enable flag (I) is set.

2.33.3 Program List

```
;***************************************************************************************************
;                                                                                                 *
;          M16C Program Collection No. 33                                                         *
;          CPU                : M16C                                                              *
;                                                                                                 *
;***************************************************************************************************
VromTOP              .EQU       0F0000H     ; Declares start address of ROM
VIstack              .EQU       002C00H     ; Interrupt stack pointer
Vintbase             .EQU       0FE000H     ; Declares interrupt vector table address
;
;=================================================================
;          Title: Variable vector table
;          Outline: Description example of variable vector table and software interrupt
;=================================================================
                     .SECTION   PROGRAM,CODE
                     .ORG       VromTOP     ; ROM area
MAIN:
          LDC        #VIstack,ISP           ; Sets interrupt stack pointer
          LDINTB     #Vintbase              ; Sets interrupt table register
;
          MOV.W      #100-1,TA0             ; Sets timer A0 counter
          MOV.B      #00000001B,TA0IC       ; Sets interrupt level 1 for timer A0
          MOV.W      #1000-1,TA1            ; Sets timer A1 counter
          MOV.B      #00000010B,TA1IC       ; Sets interrupt level 2 for timer A1
;
          MOV.B      #00000011B,TABSR       ; Timers A0 and A1 start counting
;
          FSET       I                      ; Enables interrupts
;
          INT        #21                    ; Performs timer A0 interrupt processing
                                            ; (TIMER_A0 is executed)
;
          FCLR       I                      ; Disables interrupts
;
          INT        #22                    ; Performs timer A1 interrupt processing
                                            ; (TIMER_A1 is executed)
;
          INT        #32                    ; Performs SOFTINT label interrupt processing
                     .
                     .
                     .
                     .
                     .
                     .
TIMER_A0:
               Processing
          REIT

TIMER_A1:
               Processing
          REIT

SOFTINT:
               Processing
          REIT

NOTUSE:
          REIT

;
                     .SECTION   SPECIAL,ROMDATA
                     .ORG       Vintbase    ; Variable vector table area
```

```
;---------------------------------------------------------------------------------
;           Peripheral I/O interrupt vector table
;---------------------------------------------------------------------------------
                              .LWORD      NOTUSE        ; Software interrupt number 0
                              .LWORD      NOTUSE        ; Software interrupt number 1




;
                              .ORG        Vintbase+84
                              .LWORD      TIMER_A0      ; Software interrupt number 21
                              .LWORD      TIMER_A1      ; Software interrupt number 22




;
                              .ORG        Vintbase+128  ; Software interrupt area
;---------------------------------------------------------------------------------
;            Software interrupt vector table
;---------------------------------------------------------------------------------
                              .LWORD      SOFTINT       ; Software interrupt number 32
                              .LWORD      NOTUSE        ; Software interrupt number 33





;
          .END                                          ;
```

2.34 Saving and Restoring Context

2.34.1 Outline

This program shows a usage example for saving context (STCTX instruction) and restoring context (LDCTX instruction).

2.34.2 Explanation

Tasks are executed in the main routine and context save and restore operations are performed within each task processing.

TASK contains a task's execution number. The content of the table equal to twice the content of TASK in the task execution table is executed (task execution processing). This program has three tasks to execute. Context save and restore operations are performed within each task processing.

Vcontext indicates the table's base address. The data stored at an address apart from the base address by twice the content of TASK contains register information and the next address indicates a stack pointer's correction value.

The following shows the function of register information.

b7	b6	b5	b4	b3	b2	b1	b0
FB	SB	A1	A0	R3	R2	R1	R0

The content of the register whose bit is set (= 1) is saved to or restored from a stack. The stack pointer's correction value is twice the number of registers to be saved and restored.

2.34.3 Program List

```
;*******************************************************************************************
;                                                                                         *
;         M16C Program Collection No. 34                                                  *
;         CPU                : M16C                                                       *
;                                                                                         *
;*******************************************************************************************
VramTOP              .EQU          000400H        ;  Declares start address of RAM
VromTOP              .EQU          0F0000H        ;  Declares start address of ROM
Vcontext             .EQU          0FF800H        ;  Table's base address
Vsubtbl              .EQU          0FFA00H        ;  Declares start address of subroutine table
;
                     .SECTION      RAM,DATA
                     .ORG          VramTOP        ; RAM area
TASK:                .BLKB         1              ; Task number
;
;===============================================================
;         Title: Saving/restoring context
;         Outline: Example for using STCTX/LDCTX instructions
;         Notes:
;===============================================================
                     .SECTION      PROGRAM,CODE
                     .ORG          VromTOP        ; ROM area
MAIN:
          MOV.B      TASK,A0
          SHL.W      #2,A0                        ; Subroutine pointer
;
          JSRI.A     Vsubtbl[A0]                  ; Executes task
;
          INC.B      TASK                         ; Task + 1
          CMP.B      #2,TASK                      ; Greater than number of tasks?
          JLEU       L_1                          ; --> No
          MOV.B      #0,TASK                      ; Sets task = 0
L_1:
          JMP        MAIN
;
;-------------------------------------------------------------------------------
;          Processing of task 0
;-------------------------------------------------------------------------------
TASK_0:
          STCTX      TASK,Vcontext                ; Saves registers in order of R0, R1, R2, R3, SB, and FB
                     Processing

          LDCTX      TASK,Vcontext                ; Restores registers in order of FB, SB, R3, R2, R1, and R0
          RTS
;
;-------------------------------------------------------------------------------
;         Processing of task 1
;-------------------------------------------------------------------------------
TASK_1:
          STCTX      TASK,Vcontext                ; Saves registers in order of R0, R2, SB, and FB

                     Processing

          LDCTX      TASK,Vcontext                ;Restores registers in order of FB, SB, R2, and R0
          RTS
;
```

```
;------------------------------------------------------------------------------------
;           Processing of task 2
;------------------------------------------------------------------------------------
TASK_2:
            STCTX           TASK,Vcontext                   ; Saves registers in order of R1, R3, A1, and SB

                    Processing

            LDCTX           TASK,Vcontext                   ; Restores registers in order of SB, A1, R3, and R1
            RTS
;
                            .SECTION        BASE,ROMDATA
                            .ORG            Vcontext        ; Context save/restore table area
;------------------------------------------------------------------------------------
;           Context information table
;------------------------------------------------------------------------------------
                            .BYTE   11001111B               ; TASK = 0  Register information
                            .BYTE   12                      ;           SP correction value
;
                            .BYTE   10000101B               ; TASK = 1  Register information
                            .BYTE   6                       ;           SP correction value
;
                            .BYTE   01101010B               ; TASK = 2  Register information
                            .BYTE   8                       ;           SP correction value
;
                            .SECTION        TABLE,ROMDATA
                            .ORG            Vsubtbl         ; Subroutine table area
;------------------------------------------------------------------------------------
;           Subroutine table
;------------------------------------------------------------------------------------
                            .LWORD          TASK_0          ; TASK = 0  Subroutine
                            .LWORD          TASK_1          ; TASK = 1  Subroutine
                            .LWORD          TASK_2          ; TASK = 2  Subroutine
;
            .END                                            ;
```

MEMO

Chapter 3

Program Collection of Mathematic/Trigonometric Functions

Function list

Item No.	Function	Format	Page
3.1	Single-precision, floating-point format	–	155
3.2	Addition	Library	158
3.3	Subtraction	Library	160
3.4	Multiplication	Library	162
3.5	Division	Library	164
3.6	Sine function	Library	166
3.7	Cosine function	Library	168
3.8	Tangent function	Library	170
3.9	Inverse sine function	Library	172
3.10	Inverse cosine function	Library	174
3.11	Inverse tangent function	Library	176
3.12	Square root	Library	178
3.13	Power	Library	180
3.14	Exponential function	Library	182
3.15	Natural logarithmic function	Library	184
3.16	Common logarithmic function	Library	186
3.17	Data comparison	Library	188
3.18	Conversion from FLOAT type to WORD type	Library	190
3.19	Conversion from WORD type to FLOAT type	Library	192
3.20	Program list *	–	194

*: This consists of a collection of the arithmetic library’s program lists.

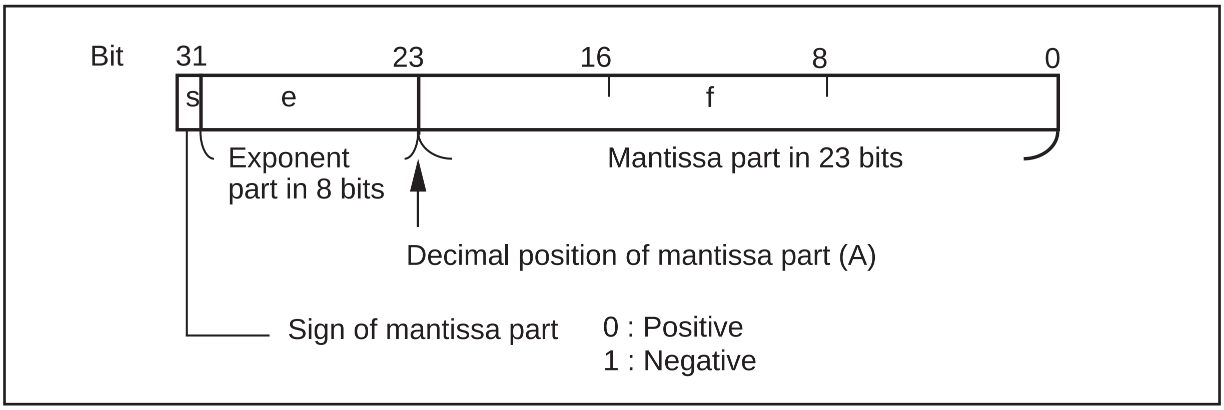
3.1 Single-precision, Floating-point Format

3.1.1 Outline

The floating-point data used in this arithmetic library conforms to the single-precision (4-byte), floating-point format in IEEE standards.
All calculations in this arithmetic library are performed by replacing or referencing register contents. Please be sure to set the necessary data in registers before calling a subroutine. Note also that although each subroutine uses the M16C/60-series' CPU registers to implement its processing, no measures are taken inside the subroutine to protect the registers. Therefore, take protective measures by, for example, saving the registers in a stack area as necessary before calling a subroutine.

3.1.2 Representation of Single-precision, Floating-point Data

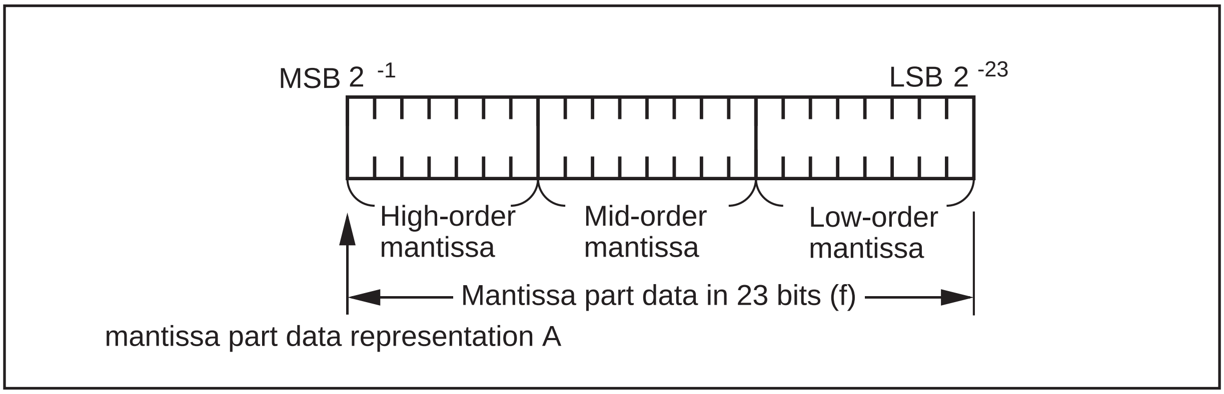
This arithmetic library uses the IEEE standards single-precision data format shown below to represent floating-point binary numbers.



Representation of floating-point data

3.1.3 Mantissa Part

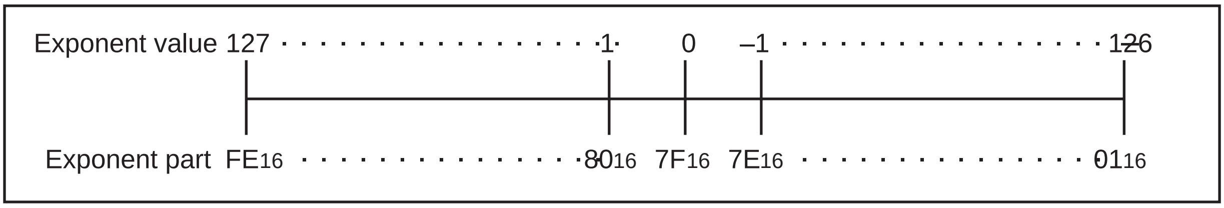
The mantissa part (f) consists of 23 bits of fixed-point real number, with the decimal point placed at position A. Since the floating-point numbers handled in this library are normalized, 1s in the most significant bit are omitted. Consequently, significant digits are always "1 + f". The range of 'f' is $0 \leq f < 1$.



Mantissa part data representation

3.1.4 Exponent Part

The exponent part uses an 8-bit unsigned binary number to express 'e' of 2^{127} to 2^{-126}. The data is expressed by a value that is prebiased by adding $7F_{16}$. (However, e = 0 and e = FF_{16} are used as special numbers.) Consequently, the actual exponent value and the representation of the exponent part have the following relationship.



Relationship between exponent value and representation of exponent part

3.1.5 Sign of Mantissa Part

The sign of the mantissa part (s) is located at the MSB (31st bit) position of the data area. Numeral 0 denotes a positive number and numeral 1 denotes a negative number.

3.1.6 Types and Meanings of Data Representation

The table below shows the values represented by binary floating-point numbers in conformity with IEEE standards.

Values represented by binary floating-point numbers

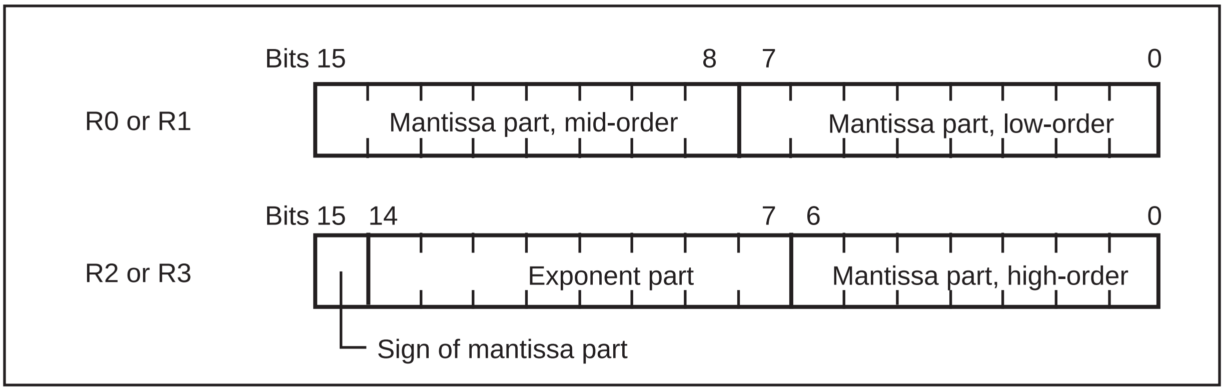
Represented value	Sign 's'	Exponent part 'e'	Mantissa part 'f'	Remarks
Non-numeral	0/1	11111111	11111111 to 00000001	All bits in exponent part are 1s and any bit in mantissa part is not 0.
Infinite	0/1	11111111	00000000	All bits in exponent part are 1s and all bits in mantissa part are 0s.
Normalized number	0/1	11111110 to 00000001	11111111 to 00000000	Maximum value 3.40×10^{38} minimum value 3.40×10^{-38}
Absolute 0	0/1	00000000	00000000	All bits in exponent part and all bits in mantissa part are 0s.

Example of normalization

Sign 's'	Exponent part 'e'	Mantissa part 'f'	Value (decimal)
0	01111111	0000000 00000000 00000000	1
0	01111011	1001100 11001100 11001101	0.1
0	01111110	0000000 00000000 00000000	0.5
1	01111111	0000000 00000000 00000000	–1
1	01111011	1001100 11001100 11001101	–0.1
1	01111110	0000000 00000000 00000000	–0.5

3.1.7 Arguments and Return Values

This section explains the floating-point arguments and return values used in this arithmetic library.
The first operand (or the number to be operated on) of an argument is assigned to registers (R2R0) and the second operand (or the number operating on it) is assigned to registers (R3R1). Set values in these registers before calling a library. The return values from a library are loaded into registers (R2R0). The diagram below shows the structure of an argument and return value.



Structure of argument and return value

3.2 Addition

3.2.1 Outline

This program adds float-point numbers.
The first operand (R2R0) is added to the second operand (R3R1) and the result is stored in (R2R0).
Calculation result (R2R0) = first operand (R2R0) + second operand (R3R1)

Subroutine name: FADD	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 18 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of first operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	Lower half of second operand	Indeterminate	Destroyed during processing
R2	Upper half of first operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	Upper half of second operand	Indeterminate	Destroyed during processing
A0	—	—	Unused
A1	—	—	Unused

Usage precautions

Since the contents of R3 and R1 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
A + B = C A: First operand; B: Second operand; C: Calculation result

3.2.2 Explanation

Procedure:

(1) Store the first operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Store the second operand (normalized single-precision, floating-point number) in R3 and R1.
R3 = sign, exponent, upper part of mantissa
R1 = mid and lower parts of mantissa

(3) Call the subroutine (FADD).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow
Minimum value of normalized number *	Underflow
Non-numeral *	Faulty data
Absolute 0 *	When calculation result = 0
First or second operand whichever larger (not changed)	Exponent underflow

* Refer to Section 3.1.5.

3.3 Subtraction

3.3.1 Outline

This program subtracts floating-point numbers.
The first operand (R2R0) and second operand (R3R1) are subtracted and the result is stored in (R2R0).
Calculation result (R2R0) = first operand (R2R0) – second operand (R3R1)

Subroutine name: FSUB	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 21 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of first operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	Lower half of second operand	Indeterminate	Destroyed during processing
R2	Upper half of first operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	Upper half of second operand	Indeterminate	Destroyed during processing
A0	—	—	Unused
A1	—	—	Unused

Usage precautions

Since the contents of R3 and R1 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
A - B = C A: First operand; B: Second operand; C: Calculation result

3.3.2 Explanation

Procedure:

(1) Store the first operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Store the second operand (normalized single-precision, floating-point number) in R3 and R1.
R3 = sign, exponent, upper part of mantissa
R1 = mid and lower parts of mantissa

(3) Call the subroutine (FSUB).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow
Minimum value of normalized number *	Underflow
Non-numeral *	Faulty data
Absolute 0 *	When calculation result = 0
First or second operand whichever larger (not changed)	Exponent underflow

* Refer to Section 3.1.5.

3.4 Multiplication

3.4.1 Outline

This program multiplies floating-point numbers.
The first operand (R2R0) and second operand (R3R1) are multiplied and the result is stored in (R2R0).
Calculation result (R2R0) = first operand (R2R0) x second operand (R3R1)

Subroutine name: FMUL	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 19 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of first operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	Lower half of second operand	Indeterminate	Destroyed during processing
R2	Upper half of first operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	Upper half of second operand	Indeterminate	Destroyed during processing
A0	—	—	Unused
A1	—	—	Unused

Usage precautions

Since the contents of R3 and R1 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
A x B = C A: First operand; B: Second operand; C: Calculation result

3.4.2 Explanation

Procedure:

(1) Store the first operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Store the second operand (normalized single-precision, floating-point number) in R3 and R1.
R3 = sign, exponent, upper part of mantissa
R1 = mid and lower parts of mantissa

(3) Call the subroutine (FMUL).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow
Minimum value of normalized number *	Underflow
Non-numeral *	Faulty data
Absolute 0 *	When calculation result = 0

* Refer to Section 3.1.5.

3.5 Division

3.5.1 Outline

This program divides floating-point numbers.
The first operand (R2R0) and second operand (R3R1) are multiplied and the result is stored in (R2R0).
Calculation result (R2R0) = first operand (R2R0) ÷ second operand (R3R1)

Subroutine name: FDIV	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 18 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of first operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	Lower half of second operand	Indeterminate	Destroyed during processing
R2	Upper half of first operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	Upper half of second operand	Indeterminate	Destroyed during processing
A0	—	—	Unused
A1	—	—	Unused

Usage precautions

Since the contents of R3 and R1 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
A ÷ B = C A: First operand; B: Second operand; C: Calculation result

3.5.2 Explanation

Procedure:

(1) Store the first operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Store the second operand (normalized single-precision, floating-point number) in R3 and R1.
R3 = sign, exponent, upper part of mantissa
R1 = mid and lower parts of mantissa

(3) Call the subroutine (FDIV).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow
Minimum value of normalized number *	Underflow
Infinite *	Division by zero
Non-numeral *	Faulty data
Absolute 0 *	When calculation result = 0
First or second operand whichever larger (not changed)	Exponent underflow

* Refer to Section 3.1.5.

3.6 Sine Function

3.6.1 Outline

This program finds a sine of the operand (R2R0) comprised of a single-precision, floating-point number and stores the result in (R2R0).

(R2R0) = SIN(R2R0)

The unit is radian.

Make sure the operand is smaller than 2π.

Subroutine name: FSIN	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 34 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1 and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.

Supplementary explanation

C = SIN(A)   A: Operand; C: Calculation result

3.6.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FSIN).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.7 Cosine Function

3.7.1 Outline

This program finds a cosine of the operand (R2R0) comprised of a single-precision, floating-point number and stores the result in (R2R0).
(R2R0) = COS(R2R0)
The unit is radian.
Make sure the operand is smaller than 2π.

Subroutine name: FCOS	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 34 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
C = COS(A)   A: Operand; C: Calculation result

3.7.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FCOS).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.8 Tangent Function

3.8.1 Outline

This program finds a tangent of the operand (R2R0) comprised of a single-precision, floating-point number and stores the result in (R2R0).
(R2R0) = TAN(R2R0)
The unit is radian.
Make sure the operand is smaller than 2π.

Subroutine name: FTAN	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 41 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
C = TAN(A) A: Operand; C: Calculation result

3.8.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FTAN).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.9 Inverse Sine Function

3.9.1 Outline

This program finds an inverse sine of the operand (R2R0) comprised of a single-precision, floating-point number and stores the result in (R2R0).

(R2R0) = SIN^{-1}(R2R0)

The unit is radian.

Make sure the operand is smaller than 2π.

Subroutine name: FASN	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 60 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.

Supplementary explanation

C = SIN^{-1}(A)   A: Operand; C: Calculation result

3.9.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FASN).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow
Non-numeral *	Argument error

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.10 Inverse Cosine Function

3.10.1 Outline

This program finds an inverse cosine of the operand (R2R0) consisting of a single-precision, floating-point number and stores the result in (R2R0).

(R2R0) = COS^{-1}(R2R0)

The unit is radian.

Make sure the operand is smaller than 2π.

Subroutine name: FACN	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 60 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.

Supplementary explanation

C = COS^{-1}(A)   A: Operand; C: Calculation result

3.10.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FACN).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow
Non-numeral *	Argument error

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.11 Inverse Tangent Function

3.11.1 Outline

This program finds an inverse tangent of the operand (R2R0) consisting of a single-precision, floating-point number and stores the result in (R2R0).

(R2R0) = TAN^{-1}(R2R0)

The unit is radian.

Make sure the operand is smaller than 2π.

Subroutine name: FATN	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 34 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.

Supplementary explanation

C = TAN^{-1}(A)   A: Operand; C: Calculation result

3.11.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FATN).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.12 Square Root

3.12.1 Outline

This program finds a square root of the operand (R2R0) consisting of a single-precision, floating-point number and stores the result in (R2R0).

(R2R0) = $\sqrt{\text{(R2R0)}}$

Subroutine name: FSQR	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 53 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.

Supplementary explanation

C = $\sqrt{A}$ A: Operand; C: Calculation result

3.12.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FSQR).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Non-numeral *	Calculation error
Maximum value of normalized number *	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.13 Power

3.13.1 Outline

This program finds a product of the operand (R2R0) consisting of a single-precision, floating-point number raised to the power of exponent data (R3R1) and stores the result in (R2R0).
(R2R0) = (R2R0)$^{(R3R1)}$

Subroutine name: FPOW	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 50 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
$C = A^B$ A: Operand; B: Exponent data; C: Calculation result

3.13.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Store the exponent data (normalized single-precision, floating-point number) in R3 and R1.
R3 = sign, exponent, upper part of mantissa
R1 = mid and lower parts of mantissa

(3) Call the subroutine (FPOW).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Non-numeral *	Calculation error
Maximum value of normalized number *	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.14 Exponential Function

3.14.1 Outline

This program finds an exponential function of the operand (R2R0) consisting of a single-precision, floating- point number and stores the result in (R2R0).
(R2R0) = $e^{(R2R0)}$

Subroutine name: FEXP	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 38 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
$C = e^A$ A: Operand; C: Calculation result

3.14.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FEXP).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Maximum value of normalized number *	Overflow or argument exceeds the range of -87.3 to 87.3 including both ends

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.15 Natural Logarithmic Function

3.15.1 Outline

This program finds a natural logarithmic function of the operand (R2R0) consisting of a single-precision, floating-point number and stores the result in (R2R0).
(R2R0) = LN(R2R0)

Subroutine name: FLN	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 41 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
C = LN(A)   A: Operand; C: Calculation result

3.15.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FLN).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Non-numeral *	Calculation error
No change	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.16 Common Logarithmic Function

3.16.1 Outline

This program finds a common logarithmic function of the operand (R2R0) consisting of a single-precision, floating- point number and stores the result in (R2R0).
(R2R0) = LOG(R2R0)

Subroutine name: FLOG	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 33 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Lower half of calculation result	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of operand	Upper half of calculation result	Sign, exponent, upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	Indeterminate	Destroyed during processing
A1	—	—	Unused
C flag	—	0: Normal; 1: Erroneous	Status of calculation result

Usage precautions

Since the contents of R3, R1, and A0 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.
Supplementary explanation
C = LOG(A)   A: Operand; C: Calculation result

3.16.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FLOG).

Calculation result:

The calculation result is placed in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

If the operation resulted in an error, one of the following values is returned.

Contents of R2 and R0	Meaning
Non-numeral *	Calculation error
No change	Overflow

* Refer to Section 3.1.5.

The status of the calculation result is set in the C flag.

Content of C flag	Meaning
1	Operation resulted in error
0	Operation completed normally

3.17 Data Comparison

3.17.1 Outline

This program compares the operand (R2R0) consisting of a single-precision, floating-point number with comparison data (R3R1) and sets the result in flags.
Flag = operand (R2R0): comparison data (R3R1)

Subroutine name: FCMP	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 32 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of operand	Does not change	←
R1	Lower half of comparison data	Does not change	←
R2	Upper half of operand	Does not change	←
R3	Upper half of comparison data	Does not change	←
A0	—	—	Unused
A1	—	—	Unused
C flag	—	1 : (R2R0) ≥ (R3R1)	Large/small result
Z flag	—	1 : (R2R0) = (R3R1)	=/ ≠ result
Usage precautions			

3.17.2 Explanation

Procedure:

(1) Store the operand (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Store the comparison data (normalized single-precision, floating-point number) in R3 and R1.
R3 = sign, exponent, upper part of mantissa
R1 = mid and lower parts of mantissa

(3) Call the subroutine (FCMP).

Calculation result:

The comparison result is placed in flags.

C flag	Z flag	Meaning
1	0	(R2, R0) > (R3, R1)
1	1	(R2, R0) = (R3, R1)
0	0	(R2, R0) < (R3, R1)

3.18 Conversion from FLOAT Type to WORD Type

3.18.1 Outline

This program converts the content of the registers (R2R0) consisting of a single-precision, floating-point number into an integer of the WORD (16-bit) type and stores the result in (R3R1).

Subroutine name: FTOI	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 1 bytes

Register/memory	Input	Output	Remarks
R0	Lower half of FLOAT type	WORD type data	Integer
R1	—	Indeterminate	Destroyed during processing
R2	Upper half of FLOAT type	Does not change	←
R3	—	—	Unused
A0	—	—	Unused
A1	—	—	Unused
C flag	—	1: Overflow or underflow	Result overflowed or underflowed
Z flag	—	1: Result is zero	Result is zero
S flag	—	1: Result is negative	Result is negative

Usage precautions

Since the content of R1 is destroyed as a result of program execution, save the register before calling the subroutine as necessary.

3.18.2 Explanation

Procedure:

(1) Store FLOAT data (normalized single-precision, floating-point number) in R2 and R0.
R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

(2) Call the subroutine (FTOI).

Result:

The result is placed in R0. However, if the operation resulted in overflow or underflow, the content of R0 becomes as shown below.

Condition	Content of R0
Positive overflow	$7FFF_{16}$
Negative overflow	8000_{16}
Underflow	0000_{16}

The status of the result is set in flags.

C flag	Z flag	S flag	Meaning
1	0	0	Positive overflow
1	0	1	Negative overflow
1	1	0	Underflow
0	1	0	Result is zero
0	0	0	Result is positive
0	0	1	Result is negative

3.19 Conversion from WORD Type to FLOAT Type

3.19.1 Outline

This program converts the content of a WORD (16-bit) type integer (R0) into a normalized single-precision, floating-point number and stores the result in (R2R0).

Subroutine name: ITOF	ROM capacity: bytes
Interrupt during execution: Accepted	Number of stacks used: 4 bytes

Register/memory	Input	Output	Remarks
R0	WORD type data	Lower half of FLOAT type	Mid and lower parts of mantissa
R1	—	Indeterminate	Destroyed during processing
R2	—	Upper half of FLOAT type	Sign, exponent, and upper part of mantissa
R3	—	Indeterminate	Destroyed during processing
A0	—	—	Unused
A1	—	—	Unused
Z flag	—	1: Result is zero	Result is zero
S flag	—	1: Result is negative	Result is negative

Usage precautions

Since the contents of R1 and R3 are destroyed as a result of program execution, save the registers before calling the subroutine as necessary.

3.19.2 Explanation

Procedure:

(1) Store a WORD type integer in R0.
(2) Call the subroutine (ITOF).

Result:

The result is placed in R2 and R0.

R2 = sign, exponent, upper part of mantissa
R0 = mid and lower parts of mantissa

The status of the result is set in flags.

Z flag	S flag	Meaning
1	0	When result is 0
0	0	When result is positive
0	1	When result is negative

3.20 Program List

```
;**************************************************************************************************
;         M16C Program Collection of Mathematic/Trigonometric Functions No. 1          *
;                                                                                       *
;         Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                            *
;**************************************************************************************************
                  .GLB          FADD
;
                  .GLB          CHKDATA       ; Checks non-numeral and infinity
;
VromTOP           .EQU          0F0000H       ; Declares start address of ROM
FBcnst            .EQU          001000H       ; Assumed FB register value
;
CALDAT            .EQU          –15           ; Calculation area (4 bytes)
SMALL             .EQU          –11           ; Compares magnitudes of first and second operand data
DEF               .EQU          –10           ; Difference between first and second operand data
SIGN              .EQU          –9            ; Sign of calculation result  0: plus; 1: minus
OPE               .EQU          –8            ; Second operand data (4 bytes)
CO_OPE            .EQU          –4            ; First operand data (4 bytes)
;
;================================================================
;         Title: Addition (single-precision, floating-point)
;
;         Content of processing:
;                   This program adds first operand data (R2R0) and second operand data (R3R1) and
;                   stores the result in R2, R0.
;                   (R2R0) = (first operand data) + (second operand data)
;
;         Procedure:
;                   (1)  First operand data (normalized single-precision, floating-point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R2 and the mantissa (mid, lower) in register R0.
;                   (2)  Second operand data (normalized single-precision, floating-point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R3 and the mantissa (mid, lower) in register R1.
;                   (3)  Call the subroutine.
;                   (4)  The calculation result is placed in R2, R0.
;
;         Result:
;                      R2 (High)                R2 (Low)                  R0H            R0L
;                         ↑                        ↑                        ↑              ↑
;                   Sign, Exponent b7 to b1  Exponent b0, Mantissa (upper)  Mantissa (mid)  Mantissa (lower)
;
;                   If the operation resulted in an error, one of the following values is returned:
;
```

Contents of R2, R0	Meaning
Maximum value	Overflow
Minimum value	Underflow
Non-numeral	Erroneous data
Absolute 0	When result is 0
First or second operand whichever larger (no change)	Underflow in exponent

```
;           Input:     ------------------------------------------->Output:
;
;           R0 (Lower half of first operand data)           R0 (Lower half of calculation result)
;           R1 (Lower half of second operand data)          R1 (Indeterminate)
;           R2 (Upper half of first operand data)           R2 (Upper half of calculation result)
;           R3 (Upper half of second operand data)          R3 (Indeterminate)
;           A0 ( )                                          A0 (Unused)
;           A1 ( )                                          A1 (Unused)
;
;           Stack amount used: 18 bytes
;=====================================================================
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
                    .FB             FBcnst              ; Assumes FB register value
FADD:
            ENTER           #15                         ; Allocates internal variables
            MOV.W           R2,CO_OPE+2[FB]             ; Saves first operand data in variables
            MOV.W           R0,CO_OPE[FB]
            MOV.W           R3,OPE+2[FB]                ; Saves second operand data in variables
            MOV.W           R1,OPE[FB]
;
            JSR             CHKDATA                     ; Checks first operand data for non-numeral and infinity
;
            MOV.W           OPE+2[FB],R2                ; Sets second operand data
            MOV.W           OPE[FB],R0
            JSR             CHKDATA                     ; Checks second operand data for non-numeral and infinity
;
            JSR             CHKZERO                     ; Checks for absolute 0
            JSR             CMPEXP                      ; Compares exponent parts
;
            MOV.W           CO_OPE+2[FB],R0             ; Checks signs of first and second operand data
            XOR.W           OPE+2[FB],R0                ; Signs are same?
            JN              FADDNS                      ; --> Signs are different
            JMP             FADDSAME                    ; --> Signs are same
;
;---------------------------------------------------------------------
;           Processing when signs are different
;---------------------------------------------------------------------
FADDNS:
            CMP.B           #24,DEF[FB]                 ; Exponent parts differ more than 24?
            JGEU            UNDERSET                    ; --> Yes (goes to set exponent part underflow information)
;
            CMP.B           #0,DEF[FB]                  ; No difference in exponent parts?
            JNE             FADDNS10                    ; --> no
;
;  No difference in exponent parts (mantissa parts are compared)
;
            MOV.B           CO_OPE+2[FB],R0H
            AND.B           #7FH,R0H
            MOV.B           OPE+2[FB],R0L
            AND.B           #7FH,R0L
            CMP.B           R0H,R0L                     ; Compares mantissa (upper) parts
            JLTU            FADDNSOP                    ; --> Second operand data is larger
            JGTU            FADDNSCO                    ; --> First operand data is larger
            MOV.W           CO_OPE[FB],R0
            CMP.W           R0,OPE[FB]                  ; Compares mantissa (mid, lower) parts
            JLTU            FADDNSOP                    ; --> Second operand data is larger
            JMP             FADDNSCO                    ; --> First operand data is larger
;
```

```
FADDNS10:
        CMP.B       #0,SMALL[FB]
        JEQ         FADDNSOP            ; --> Exponent part of first operand data is larger
;
; Aligning digits of first operand data
FADDNSCO:
        BTST        7,OPE+3[FB]         ; Checks sign of second operand data
        STZX        #0,#1,SIGN[FB]      ; Sets sign of calculation result
        JSR         CO_OPESHF           ; Aligns digits of first operand data
        JMP         SUBCAL              ; Subtraction
;
; Aligning digits of second operand data
FADDNSOP:
        BTST        7,CO_OPE+3[FB]      ; Checks sign of first operand data
        STZX        #0,#1,SIGN[FB]      ; Sets sign of calculation result
        JSR         OPESHF              ; Aligns digits of second operand data
;
;------------------------------------------------------------------
;           (R1R0) – CALDAT
;------------------------------------------------------------------
SUBCAL:
        SUB.W       CALDAT[FB],R0       ; Subtracts mantissa (mid, lower) parts together
        SBB.B       CALDAT+2[FB],R1L    ; Subtracts mantissa parts together including borrow
        JC          FADDNOR             ; --> No underflow in mantissa (goes to normalization processing)
;
; Setting underflow information (minimum value)
        MOV.W       #0000H,R0           ; Sets minimum value in mantissa (mid, lower) part
        MOV.W       #0100H,R2           ; Sets minimum value in exponent part and mantissa part (upper)
        SHL.B       #-1,SIGN[FB]        ; Places sign in C flag
        RORC.W      R2                  ; Sets sign
        EXITD
;
; Normalization processing
FADDNOR:
        BTST        7,R1
        JNE         CALSET              ; --> Normalization completed
;
        SHL.W       #1,R0               ; Normalizes mantissa (mid, lower) part
        ROLC.B      R1L                 ; Normalizes mantissa (upper) part
        SUB.B       #1,R1H              ; Normalizes exponent part
        JMP         FADDNOR             ; --> Continues normalization processing
;
;------------------------------------------------------------------
;           Processing when signs are same
;------------------------------------------------------------------
FADDSAME:
        CMP.B       #24,DEF[FB]         ; Exponent parts differ more than 24?
        JLTU        FADDSA10            ; --> Difference in exponent parts is 23 or less
;
;Setting exponent part underflow information (no change)
UNDERSET:
        CMP.B       #0,SMALL[FB]        ; Which data, first or second operand, is returned “not changed”?
        JEQ         FADDSACO            ; --> First operand data is returned “not changed”
                                        ; Second operand data is returned “not changed”
        MOV.W       OPE[FB],R0
        MOV.W       OPE+2[FB],R2        ; Sets “no change” for second operand data
        EXITD
```

```
FADDSACO:
        MOV.W       CO_OPE[FB],R0
        MOV.W       CO_OPE+2[FB],R2       ; Sets “no change” for first operand data
        EXITD
;
FADDSA10:
        BTST        7,CO_OPE+3[FB]        ; Checks sign of first operand data
        STZX        #0,#1,SIGN[FB]        ; Sets sign of calculation result
        TST.B       #0FFH,SMALL[FB]
        JEQ         FADDSA100             ; --> Exponent part of first operand data is larger
;
; Aligning digits of first operand data
;
        JSR         CO_OPESHF             ; Aligns digits of first operand data
        JMP         ADDCAL                ; Addition
;
; Aligning digits of second operand data
;
FADDSA100:
        JSR         OPESHF                ; Aligns digits of second operand data

;---------------------------------------------------------------------
;       (R1R0) + CALDAT
;---------------------------------------------------------------------
ADDCAL:
        ADD.W       CALDAT[FB],R0         ; Adds mantissa (mid, lower) parts together
        ADC.B       CALDAT+2[FB],R1L      ; Adds mantissa (upper) parts together including carry
        JNC         CALSET                ; --> No overflow in mantissa part (goes to set
                                          ; calculation result)
;
;Overflow check
;
        ADD.B       #1,R1H                ; Exponent + 1
        CMP.B       #0FFH,R1H             ; Overflow?
        JGEU        OVERSET               ; --> Overflow (goes to set overflow information)
;
; Aligning digits
;
        FSET        C                     ; Sets overflow bit of mantissa
        RORC.B      R1L                   ; Borrows 1 from LSB in mantissa (upper) part
        RORC.W      R0                    ; Borrows 1 from LSB in mantissa (mid, lower) part
;
;Setting calculation result
;
CALSET:
        SHL.B       #1,R1L                ; Discards economized form bit
        SHL.B       #-1,SIGN[FB]          ; Places sign in C flag
        RORC.W      R1                    ; Sets sign
        MOV.W       R1,R2                 ; Sets sign, exponent part, and mantissa (upper) part in R2
        EXITD
;
; Setting overflow information (maximum value)
;
OVERSET:
        MOV.W       #0FFFFH,R0            ; Sets maximum value in mantissa (mid, lower) part
        MOV.W       CO_OPE+2[FB],R2       ; Reads exponent part and mantissa (upper) part
        AND.W       #8000H,R2             ; Clears exponent part and mantissa (upper) part
        OR.W        #7F7FH,R2             ; Sets maximum value in exponent and mantissa (upper) parts
                                          ; (without changing sign)
        EXITD
;
```

```
;/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;           Absolute 0 Check Subroutine
;
;           Function:
;                       When the operation results is zero, this subroutine sets absolute 0 in R2 and R0 before
;                       returning to the previous program location (from which FADD was called).  If the result is
;                       other than the above, the subroutine returns to the program location from which it was called.
;/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
CHKZERO:
        MOV.W       CO_OPE+2[FB],R0         ; Reads exponent and mantissa (upper) parts of first operand data
        OR.W        OPE+2[FB],R0            ; Checks exponent parts of first and second operand data
        AND.W       #7F80H,R0               ; Exponent parts of both are 0?
        JEQ         ZEROSET                 ; --> Sets absolute 0
;
        MOV.W       CO_OPE+2[FB],R0         ; Reads exponent and mantissa (upper) parts of first operand data
        AND.W       #7F80H,R0               ; Exponent part is 0?
        JEQ         OPE_ANS                 ; --> Returns second operand data as answer
;
        MOV.W       OPE+2[FB],R0            ; Reads exponent and mantissa (upper) parts of second operand data
        AND.W       #7F80H,R0               ; Exponent part is 0?
        JEQ         CO_OPE_ANS              ; --> Returns first operand data as answer
;
        CMP.W       OPE[FB],CO_OPE[FB]      ; Compares mantissa parts (mid, lower) of first and second operand data
        JNE         CKZRET                  ; --> Contents are different (not 0)
        MOV.W       CO_OPE+2[FB],R0         ; Reads exponent and mantissa (upper) parts of first operand data
        XOR.W       #8000H,R0               ; Inverts sign (to make it matched to sign of second operand data)
        CMP.W       OPE+2[FB],R0            ; Compares exponent and mantissa (upper) parts
        JEQ         ZEROSET                 ; --> Contents are same (goes to set absolute 0)
CKZRET:
        RTS                                 ; Returns to the program location from which FADD was called
;
;Setting second operand data
;
OPE_ANS:
        MOV.W       OPE[FB],R0
        MOV.W       OPE+2[FB],R2            ; Sets "no change" for second operand data
        JMP         ZERO_EXIT
;
;Setting first operand data
;
CO_OPE_ANS:
        MOV.W       CO_OPE[FB],R0
        MOV.W       CO_OPE+2[FB],R2         ; Sets "no change" for first operand data
        JMP         ZERO_EXIT
;
;Setting absolute 0
;
ZEROSET:
        MOV.W       #0000H,R0               ; Sets absolute 0 in mantissa (mid, upper) part
        MOV.W       #0000H,R2               ; Sets absolute 0 in exponent and mantissa (upper) parts
ZERO_EXIT:
        STC         SP,R3                   ; Reads stack
        ADD.W       #4,R3                   ; Stack + 4 (for 2 returns)
        LDC         R3,SP                   ; Sets stack back again
        EXITD
;
```

```
;////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;           Exponent Part Comparing Subroutine
;
;           Function:
;                     This subroutine subtracts the exponent part of the second operand data from that of
;                     the first operand data and returns the result indicating which operand data is larger.
;                               When SMALL [FB] = 0, the exponent part of the first operand data is larger
;                               When SMALL [FB] = 1, the exponent part of the second operand data is larger
;                     Furthermore, the subroutine returns the difference. The difference is returned by DEF [FB].
;////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
CMPEXP:
          MOV.W       OPE+2[FB],R0          ; Loads exponent part of second operand data into DEF
          SHL.W       #1,R0
          MOV.B       R0H,DEF[FB]
;
          MOV.W       CO_OPE+2[FB],R0       ; Reads exponent part of first operand data
          SHL.W       #1,R0
;
          SUB.B       DEF[FB],R0H           ; Subtracts exponent part of second operand data
                                            ; from that of first operand data
          JPZ         CMPPLUS               ; --> Exponent part of first operand data ≥ exponent
                                            ; part of second operand data
;
; Exponent of first operand data ≤ exponent of second operand data
          MOV.B       #1,SMALL[FB]          ; Sets information that second operand data is larger
          XOR.B       #0FFH,R0H             ; Changes difference in exponent parts to positive
                                            ; number (2's complement)
          INC.B       R0H
          MOV.B       R0H,DEF[FB]           ; Sets difference in exponent parts
          RTS
;
CMPPLUS:
          MOV.B       #0,SMALL[FB]          ; Sets information that first operand data is larger
          MOV.B       R0H,DEF[FB]           ; Sets difference in exponent parts
          RTS
;
;////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;           Second Operand Data Digit Adjusting Subroutine
;
;           Function:
;                     This subroutine adds a economized form bit to the second operand data,
;                     loads the sum into CALDAT to adjust digits, and returns the sum of the first
;                     operand data plus economized form bit placed in R0 and R1.
;////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
OPESHF:
; Converting second operand data into calculation-purpose data and loading it into register
          MOV.W       OPE[FB],R0            ; Mantissa (mid, lower) part of second operand data --> R0
          MOV.W       OPE+2[FB],R1          ; Exponent part of second operand data --> R1H,
                                            ; Mantissa part (upper) --> R1L
          SHL.W       #1,R1                 ; Discards sign and adjusts R1H to exponent part
          FSET        C                     ; Sets economized form bit
          RORC.B      R1L                   ; Sets mantissa (upper) part including economized
                                            ; form bit in R1L
; Digit adjust processing
OPESHT:
          DEC.B       DEF[FB]               ; Difference in exponent part - 1
          JN          OPESHTSET             ; Digit adjustment finished? --> Yes
          ADD.B       #1,R1H                ; Exponent part + 1
          SHL.B       #-1,R1L               ; Shifts mantissa (upper) part down
          RORC.W      R0                    ; Shifts mantissa (mid, lower) part down
          JMP         OPESHT
```

```
; Loading digit-adjusted content into CALDAT
;
OPESHTSET:
          MOV.W     R0,CALDAT[FB]           ; Loads mantissa (mid, lower) part
          MOV.W     R1,CALDAT+2[FB]         ; Loads exponent and mantissa (upper) parts
;
; Converting first operand data into calculation-purpose data and loading it into register
;
          MOV.W     CO_OPE[FB],R0           ; Mantissa (mid, lower) part of first operand data --> R0
          MOV.W     CO_OPE+2[FB],R1         ; Exponent part of first operand data --> R1H,
                                            ; Mantissa (upper) part --> R1L
          SHL.W     #1,R1                   ; Discards sign and adjusts R1H to exponent part
          FSET      C                       ; Sets economized form bit
          RORC.B    R1L                     ; Sets mantissa (upper) part including economized
                                            ; form bit in R1L
          RTS
;
;//////////////////////////////////////////////////////////////////////////////////////////////////////////
;         First Operand Data Digit Adjusting Subroutine
;
;         Function:
;                   This subroutine adds a economized form bit to the first operand data, loads the
;                   sum into CALDAT to adjust digits, and returns the sum of the second operand data
;                   plus economized form bit placed in R0 and R1.
;//////////////////////////////////////////////////////////////////////////////////////////////////////////
CO_OPESHF:
;
; Converting first operand data into calculation-purpose data and loading it into register
;
          MOV.W     CO_OPE[FB],R0           ; Mantissa (mid, lower) part of first operand data --> R0
          MOV.W     CO_OPE+2[FB],R1         ; Exponent part of first operand data --> R1H,
                                            ; Mantissa (upper) part --> R1L
          SHL.W     #1,R1                   ; Discards sign and adjusts R1H to exponent part
          FSET      C                       ; Sets economized form bit
          RORC.B    R1L                     ; Sets mantissa (upper) part including economized
                                            ; form bit in R1L
;
; Digit adjust processing
;
COSHT:
          DEC.B     DEF[FB]                 ; Difference in exponent part - 1
          JN        COSHTSET                ; Digit adjustment finished? --> Yes
          ADD.B     #1,R1H                  ; Exponent part + 1
          SHL.B     #-1,R1L                 ; Shifts mantissa (upper) part down
          RORC.W    R0                      ; Shifts mantissa (mid, lower) part down
          JMP       COSHT                   ;
;
; Loading digit-adjusted content into CALDAT
;
COSHTSET:
          MOV.W     R0,CALDAT[FB]           ; Loads mantissa (mid, lower) part
          MOV.W     R1,CALDAT+2[FB]         ; Loads exponent and mantissa (upper) parts
;
; Converting second operand data into calculation-purpose data and loading it into register

;
          MOV.W     OPE[FB],R0              ; Mantissa (mid, lower) part of second operand data --> R0
          MOV.W     OPE+2[FB],R1            ; Exponent part of second operand data --> R1H,
                                            ; Mantissa (upper) part --> R1L
          SHL.W     #1,R1                   ; Discards sign and adjusts R1H to exponent part
          FSET      C                       ; Sets economized form bit
          RORC.B    R1L                     ; Sets mantissa (upper) part including economized
                                            ; form bit in R1L
          RTS
;
          .END
```

```
;****************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 2          *
;                                                                                       *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                           *
;****************************************************************************************************
                    .GLB          FSUB
;
                    .GLB          FADD
;
VromTOP             .EQU          0F0000H                 ; Declares start address of ROM
;=====================================================================
;          Title: Subtraction (single-precision, floating-point)
;          Content of processing:
;                    This program subtracts first operand data (R2R0) and second operand data (R3R1)
;                    and stores the result in R2, R0.
;                    (R2R0) = (first operand data) – (second operand data)
;          Procedure:
;                    (1)  First operand data (normalized single-precision, floating-point number)
;                         Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                         R2 and the mantissa (mid, lower) in register R0.
;                    (2)  Second operand data (normalized single-precision, floating-point number)
;                         Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                         R3 and the mantissa (mid, lower) in register R1.
;                    (3)  Call the subroutine.
;                    (4)  The calculation result is placed in R2, R0.
;          Result:
;                  R2 (High)              R2 (Low)                     R0H               R0L
;                     ↑                      ↑                          ↑                 ↑
;          Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                    If the operation resulted in an error, one of the following values is returned:
```

Contents of R2, R0	Meaning
Maximum value	Overflow
Minimum value	Underflow
Non-numera	Erroneous data
Absolute 0	When result is 0
First or second operand whichever larger (no change)	Underflow in exponent

```
;
;          Input:    -------------------------------------------> Output:
;
;          R0 (Lower half of first operand data)         R0 (Lower half of calculation result)
;          R1 (Lower half of second operand data)        R1 (Indeterminate)
;          R2 (Upper half of first operand data)         R2 (Upper half of calculation result)
;          R3 (Upper half of second operand data)        R3 (Indeterminate)
;          A0 ( )                                        A0 (Unused)
;          A1 ( )                                        A1 (Unused)
;          Stack amount used: 21 bytes
;=====================================================================
                    .SECTION      PROGRAM,CODE
                    .ORG          VromTOP
FSUB:
          XOR.W          #8000H,R3               ; Inverts sign of second operand data
          JSR            FADD                    ; Then, result is obtained by adding
          RTS
;
          .END
```

```
;*************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 3          *
;                                                                                       *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                           *
;*************************************************************************************************
                  .GLB        FMUL
;
                  .GLB        CHKDATA       ; Checks non-numeral and infinity
;
VromTOP           .EQU        0F0000H       ; Declares start address of ROM
FBcnst            .EQU        001000H       ; Assumed FB register value
;
CALDAT            .EQU        –16           ; Calculation area (6 bytes)
EXP               .EQU        –10           ; Calculation result of exponent part
SIGN              .EQU        –9            ; Sign of calculation result  0: plus; 1: minus
OPE               .EQU        –8            ; Second operand data (4 bytes)
CO_OPE            .EQU        –4            ; First operand data (4 bytes)
;
;=================================================================
;         Title: Multiplication (single-precision, floating-point)
;         Content of processing:
;                   This program multiplies first operand data (R2R0) and second operand data (R3R1)
;                   and stores the result in R2, R0.
;                   (R2R0) = (first operand data) x (second operand data)
;         Procedure:
;                   (1)  First operand data (normalized single-precision, floating-point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R2 and the mantissa (mid, lower) in register R0.
;                   (2)  Second operand data (normalized single-precision, floating-point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R3 and the mantissa (mid, lower) in register R1.
;                   (3)  Call the subroutine.
;                   (4)  The calculation result is placed in R2, R0.
;         Result:
;                      R2 (High)              R2 (Low)                  R0H                R0L
;                        ↑                      ↑                        ↑                  ↑
;         Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                   If the operation resulted in an error, one of the following values is returned:
;
;                   Contents of R2, R0                  Meaning
;
;                   Maximum value                       Overflow
;
;                   Minimum value                       Underflow
;
;                   Non-numeral                         Erroneous data
;
;                   Absolute 0                          When result is 0
;
;         Input:    -------------------------------------------> Output:
;         R0 (Lower half of first operand data)      R0 (Lower half of calculation result)
;         R1 (Lower half of second operand data)     R1 (Indeterminate)
;         R2 (Upper half of first operand data)      R2 (Upper half of calculation result)
;         R3 (Upper half of second operand data)     R3 (Indeterminate)
;         A0 ( )                                     A0 (Unused)
;         A1 ( )                                     A1 (Unused)
;         Stack amount used: 19 bytes
;=================================================================
```

```
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
                    .FB             FBcnst          ; Assumes FB register value
FMUL:
          ENTER           #16                       ; Allocates internal variables
          MOV.W           R2,CO_OPE+2[FB]           ; Saves first operand data in variables
          MOV.W           R0,CO_OPE[FB]
          MOV.W           R3,OPE+2[FB]              ; Saves second operand data in variables
          MOV.W           R1,OPE[FB]
;
          JSR             CHKDATA                   ; Checks first operand data for non-numeral and infinity
;
          MOV.W           OPE+2[FB],R2              ; Sets second operand data
          MOV.W           OPE[FB],R0
          JSR             CHKDATA                   ; Checks second operand data for non-numeral and infinity
;
          MOV.W           CO_OPE+2[FB],R0           ; Checks signs of first and second operand data
          XOR.W           OPE+2[FB],R0              ; Signs are same?
          JN              FMUL1                     ; --> Signs are different
;
; Signs are same (signs are made positive)
;
          MOV.B           #0,SIGN[FB]               ; Turns signs positive
          JMP             FMUL10
;
; Signs are different (signs are made negative)
;
FMUL1:
          MOV.B           #1,SIGN[FB]               ; Turns signs negative
;
;--------------------------------------------------------------------
;         Absolute 0 check
;--------------------------------------------------------------------
FMUL10:
          MOV.W           CO_OPE+2[FB],R0           ; Reads exponent part of first operand data
          AND.W           #7F80H,R0                 ; Clears all but exponent part
          JEQ             FMULZERO                  ; --> Sets absolute 0
          MOV.W           OPE+2[FB],R0              ; Reads exponent part of second operand data
          AND.W           #7F80H,R0                 ; Clears all but exponent part
          JNZ             FMUL20                    ; --> Not absolute 0
;
; Setting absolute 0
;
FMULZERO:
          MOV.W           #0,R0                     ; Sets absolute 0 in return value
          MOV.W           #0,R2
          SHL.B           #-1,SIGN[FB]              ; Loads sign into C flag
          RORC.W          R2                        ; Sets sign
          EXITD
;
;--------------------------------------------------------------------
;         Adding exponent part
;--------------------------------------------------------------------
FMUL20:
          MOV.W           CO_OPE+2[FB],R0           ; Reads exponent part of first operand data
          SHL.W           #-7,R0                    ; Adjusts exponent part to low-order bits
          AND.W           #00FFH,R0                 ; Clears all but exponent part
```

```
        MOV.W       OPE+2[FB],R1        ; Reads exponent part of second operand data
        SHL.W       #-7,R1              ; Adjusts exponent part to low-order bits
        AND.W       #00FFH,R1           ; Clears all but exponent part
;
        ADD.W       R1,R0               ; Adds exponent part
        SUB.W       #7FH-1,R0           ; Subtracts 7F from addition result
                                        ; (in effect, subtracted by 7E to adjust digits)
;
        JC          FMUL30              ; --> Overflow check
;
; Setting underflow information (minimum value)
;
        MOV.W       #0,R0               ; Sets minimum value in mantissa (mid, lower) part
        MOV.W       #0100H,R2           ; Sets minimum value in mantissa (upper) part and
                                        ; LSB of exponent part
        SHL.B       #-1,SIGN[FB]        ; Checks signs
        RORC.W      R2                  ; Sets minimum value in exponent part and sign
        EXITD
;
Overflow check
;
FMUL30:
        CMP.W       #00FFH,R0           ; Overflow?
        JLTU        FMUL40              ; --> No overflow
;
; Setting overflow information (maximum value)
;
        MOV.W       #0FFFFH,R0          ; Sets maximum value in mantissa (mid, lower) part
        MOV.W       #0FEFEH,R2          ; Sets maximum value in mantissa (upper) part and
                                        ; LSB of exponent part
        SHL.B       #-1,SIGN[FB]        ; Checks signs
        RORC.W      R2                  ; Sets maximum value in exponent part and sign
        EXITD
;
;-----------------------------------------------------------------------
;       Multiplication of mantissa part
;-----------------------------------------------------------------------
FMUL40:
        MOV.B       R0L,EXP[FB]         ; Stores calculation result of exponent part
;
        MOV.W       CO_OPE+2[FB],R0     ; Reads mantissa (upper) part
        AND.W       #007FH,R0           ; Clears exponent part
        BSET        7,R0                ; Sets economized form bit
        MOV.W       R0,CO_OPE+2[FB]     ; Loads only mantissa (upper) part into first operand data
;
        MOV.W       OPE[FB],R0          ; Reads mantissa (mid, lower) part of second operand data
        MULU.W      CO_OPE[FB],R0       ; Multiplies mantissa (mid, lower) part
        MOV.W       R0,CALDAT[FB]       ; Stores lower half of calculation result
        MOV.W       R2,CALDAT+2[FB]     ; Stores upper half of calculation result
;
        MOV.W       OPE[FB],R0          ; Reads mantissa (mid, lower) part of second operand data
        MULU.W      CO_OPE+2[FB],R0     ; Multiplies mantissa (mid, lower) and (upper) parts
        ADD.W       R0,CALDAT+2[FB]     ; Adds and stores lower half of calculation result
        ADCF.W      R2                  ; Adds upper half of calculation result
        MOV.W       R2,CALDAT+4[FB]     ; Stores upper half of calculation result
;
```

```
        MOV.W       OPE+2[FB],R0          ; Reads mantissa (upper) part of second operand data
        AND.W       #007FH,R0             ; Clears exponent part and sign
        BSET        7,R0                  ; Sets economized form bit
        MULU.W      CO_OPE[FB],R0         ; Multiplies mantissa (upper) and (mid, lower) parts
        ADD.W       R0,CALDAT+2[FB]       ; Adds and stores lower half of calculation result
        ADC.W       R2,CALDAT+4[FB]       ; Adds and stores upper half of calculation result
;
        MOV.W       OPE+2[FB],R0          ; Reads mantissa (upper) part of second operand data
        AND.W       #007FH,R0             ; Clears exponent part and sign
        BSET        7,R0                  ; Sets economized form bit
        MULU.W      CO_OPE+2[FB],R0       ; Multiplies mantissa (upper) parts
        ADD.W       R0,CALDAT+4[FB]       ; Adds and stores upper half of calculation result
;
;-------------------------------------------------------------------
;       Adjusting digits
;-------------------------------------------------------------------
        BTST        7,CALDAT+5[FB]        ; Digit adjustment finished?
        JNZ         FMULSET               ; --> Finished
        SHL.W       #1,CALDAT+2[FB]       ; Adjusts digits of calculation data
        ROLC.W      CALDAT+4[FB]
        DEC.B       EXP[FB]               ; Adjusts exponent (exponent part - 1)
;
; Setting calculation result in return value
;
FMULSET:
        MOV.W       CALDAT+3[FB],R0       ; Sets calculation result of mantissa (mid, lower) part
        MOV.B       EXP[FB],R1H           ; Reads calculation result of exponent part
        MOV.B       CALDAT+5[FB],R1L      ; Reads calculation result of mantissa (upper) part
        SHL.B       #1,R1L                ; Discards economized form bit
        SHL.B       #-1,SIGN[FB]          ; Loads sign into C flag
        RORC.W      R1                    ; Sets sign
        MOV.W       R1,R2                 ; Sets sign, exponent part, and mantissa (upper)
                                          ; calculation result
        EXITD
;
        .END
```

```
;*****************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 4             *
;                                                                                           *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                              *
;*****************************************************************************************************
                .GLB            FDIV
;
                .GLB            CHKDATA         ; Checks non-numeral and infinity
;
VromTOP         .EQU            0F0000H         ; Declares start address of ROM
FBcnst          .EQU            001000H         ; Assumed FB register value
;
CALBUF          .EQU            –15             ; Calculation buffer
COUNT           .EQU            –11             ; Counter
EXP             .EQU            –10             ; Calculation result of exponent part
SIGN            .EQU            –9              ; Sign of calculation result  0: plus; 1: minus
OPE             .EQU            –8              ; Second operand data (4 bytes)
CO_OPE          .EQU            –4              ; First operand data (4 bytes)
;
;==================================================================
;           Title: Division (single-precision, floating-point)
;
;           Content of processing:
;                   This program divides first operand data (R2R0) and second operand data (R3R1) and
;                   stores the result in R2, R0.
;                   (R2R0) = (first operand data) ÷ (second operand data)
;
;           Procedure:
;                   (1)  First operand data (normalized single-precision, floating-point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R2 and the mantissa (mid, lower) in register R0.
;                   (2)  Second operand data (normalized single-precision, floating-point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R3 and the mantissa (mid, lower) in register R1.
;                   (3)  Call the subroutine.
;                   (4)  The calculation result is placed in R2, R0.
;
;           Result:
;                      R2 (High)              R2 (Low)                 R0H             R0L
;                         ↑                      ↑                       ↑               ↑
;           Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                   If the operation resulted in an error, one of the following values is returned:
```

Contents of R2, R0	Meaning
Maximum value	Overflow
Minimum value	Underflow
Infinite	Zero division
Non-numeral	Erroneous data
Absolute 0	When result is 0
First or second operand whichever larger (no change)	Underflow in exponent part

```
;           Input:    ----------------------------------------->  Output:
;
;           R0 (Lower half of first operand data)            R0 (Lower half of calculation result)
;           R1 (Lower half of second operand data)           R1 (Indeterminate)
;           R2 (Upper half of first operand data)            R2 (Upper half of calculation result)
;           R3 (Upper half of second operand data)           R3 (Indeterminate)
;           A0 ( )                                           A0 (Unused)
;           A1 ( )                                           A1 (Unused)
;
;           Stack amount used: 18 bytes
;==================================================================
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
                    .FB             FBcnst              ; Assumes FB register value
FDIV:
          ENTER           #15                  ; Allocates internal variables
          MOV.W           R2,CO_OPE+2[FB]      ; Saves first operand data in variables
          MOV.W           R0,CO_OPE[FB]
          MOV.W           R3,OPE+2[FB]         ; Saves second operand data in variables
          MOV.W           R1,OPE[FB]
;
          JSR             CHKDATA              ; Checks first operand data for non-numeral and infinity
;
          MOV.W           OPE+2[FB],R2         ; Sets second operand data
          MOV.W           OPE[FB],R0
          JSR             CHKDATA              ; Checks second operand data for non-numeral and infinity
;
          MOV.B           CO_OPE+3[FB],R0H     ; Checks signs of first and second operand data
          XOR.B           OPE+3[FB],R0H        ; Signs are same?
          JN              FDIV1                ; --> Signs are different
;
; Signs are same (signs are made positive)
;
          MOV.B           #0,SIGN[FB]          ; Turns signs positive
          JMP             FDIV10
;
; Signs are different (signs are made negative)
;
FDIV1:
          MOV.B           #1,SIGN[FB]          ; Turns signs negative
;
```

```
;------------------------------------------------------------------
;           Zero division check
;------------------------------------------------------------------
FDIV10:
        MOV.W       OPE+2[FB],R0        ; Reads exponent and mantissa (upper) parts of
                                        ; second operand data
        BCLR        15,R0               ; Clears sign
        OR.W        OPE[FB],R0          ; All bits in exponent and mantissa parts of second
                                        ; operand data are 0? (zero division?)
        JNE         FDIV20              ; --> No (not zero division)
;
; Setting zero division (infinite value)
;
        MOV.W       #0,R0               ; Sets infinite value in mantissa (mid, lower) part
        MOV.W       #0FF00H,R2          ; Sets infinite value in exponent and mantissa
                                        ; (upper) parts
        SHL.B       #-1,SIGN[FB]        ; Loads sign into C flag
        RORC.W      R2                  ; Sets sign
        EXITD
;
;------------------------------------------------------------------
;           Absolute 0 check
;------------------------------------------------------------------
FDIV20:
        MOV.W       CO_OPE+2[FB],R0     ; Reads exponent and mantissa (upper) parts of
                                        ; first operand data
        BCLR        15,R0               ; Clears sign
        OR.W        CO_OPE[FB],R0       ; All bits in exponent and mantissa parts of first
                                        ; operand data are 0? (zero division?)
        JNE         FDIV30              ; --> No (not absolute 0)
;
; Setting absolute 0
;
        MOV.W       #0,R0               ; Sets absolute 0
        MOV.W       #0,R2
        SHL.B       #-1,SIGN[FB]        ; Loads sign into C flag
        RORC.W      R2                  ; Sets sign
        EXITD
;
```

```
;---------------------------------------------------------------------
;           Checking first operand data = second operand data
;---------------------------------------------------------------------
FDIV30:
            MOV.W       OPE[FB],R0
            CMP.W       CO_OPE[FB],R0       ; Mantissa (mid, lower) parts of first and second
                                            ; operand data are same?
            JNE         FDIV40              ; --> No
            MOV.W       OPE+2[FB],R0
            BCLR        15,R0               ; Clears sign of second operand data
            BCLR        7,CO_OPE+3[FB]      ; Clears sign of first operand data
            CMP.W       CO_OPE+2[FB],R0     ; Exponent and mantissa (upper) parts of first and
                                            ; second operand data are same?
            JNE         FDIV40              ; --> No
;
; Setting calculation result 1
;
            MOV.W       #0,R0               ; Sets mantissa (mid, lower) part
            MOV.W       #7F00H,R2           ; Sets exponent and mantissa (upper) parts
            SHL.B       #-1,SIGN[FB]        ; Loads sign into C flag
            RORC.W      R2                  ; Sets sign
            EXITD
;
;---------------------------------------------------------------------
;           Subtracting exponent parts
;---------------------------------------------------------------------
FDIV40:
            MOV.W       CO_OPE+2[FB],R0     ; Reads exponent part of first operand data
            SHL.W       #-7,R0              ; Adjusts exponent part to low-order bits
            AND.W       #00FFH,R0           ; Clears all but exponent part
            MOV.W       OPE+2[FB],R1        ; Reads exponent part of second operand data
            SHL.W       #-7,R1              ; Adjusts exponent part to low-order bits
            AND.W       #00FFH,R1           ; Clears all but exponent part
            SUB.W       R1,R0               ; Subtracts exponent parts
;
```

```
;-------------------------------------------------------------------
;           Checking underflow and overflow
;-------------------------------------------------------------------
            JC              FDIV41                  ; --> First operand ≥ second operand
            CMP.B           #83H,R0L                ; Underflow occurred?
            JC              FDIV50                  ; --> No underflow
;
; Setting underflow information (minimum value)
;
            MOV.W           #0,R0                   ; Sets minimum value in mantissa (mid, lower) part
            MOV.W           #0100H,R2               ; Sets minimum value in exponent and mantissa
                                                    ; (upper) parts
            SHL.B           #-1,SIGN[FB]            ; Loads sign into C flag
            RORC.W          R2                      ; Sets sign
            EXITD
;
FDIV41:
            CMP.B           #80H,R0L                ; Overflow occurred?
            JNC             FDIV50                  ; --> No overflow
;
; Setting overflow information (maximum value)
;
            MOV.W           #0FFFFH,R0              ; Sets maximum value in mantissa (mid, lower) part
            MOV.W           #0FEFFH,R2              ; Sets maximum value in exponent and mantissa
                                                    ; (upper) parts
            SHL.B           #-1,SIGN[FB]            ; Loads sign into C flag
            RORC.W          R2                      ; Sets sign
            EXITD
;
; Storing calculation result of exponent part
;
FDIV50:
            ADD.B           #80H-1,R0L              ; Adds 80H from subtraction result
                                                    ; (in effect, added by 7F for digit adjustment)
            MOV.B           R0L,EXP[FB]             ; Stores calculation result of exponent part
;
```

```
;---------------------------------------------------------------------
;           Converting first/second operand data into calculation-purpose data
;           4 bytes = mantissa + economized form  bit + 8 low-order bits
;---------------------------------------------------------------------
            MOV.W           CO_OPE[FB],R0           ; Reads mantissa (mid, lower) part of first operand data
            MOV.W           CO_OPE+2[FB],R2         ; Reads mantissa (upper) part of first operand data
            AND.W           #007FH,R2               ; Clears exponent and sign parts
            BSET            7,R2                    ; Adds economized form bit
;
            MOV.W           OPE[FB],R1              ; Reads mantissa (mid, lower) part of second operand data
            MOV.W           OPE+2[FB],R3            ; Reads mantissa (upper) part of second operand data
            AND.W           #007FH,R3               ; Clears exponent and sign parts
            BSET            7,R3                    ; Adds economized form bit
;
            MOV.W           #0,CALBUF[FB]           ; Clears calculation result
            MOV.W           #0,CALBUF+2[FB]
;
;---------------------------------------------------------------------
;           First operand data ÷ second operand data
;---------------------------------------------------------------------
            MOV.B           #24,COUNT[FB]           ; Number of shifts performed
;
DIVCALC:
            SHL.W           #1,CALBUF[FB]           ; Shifts calculation result
            ROLC.W          CALBUF+2[FB]
;
            CMP.W           R3,R2
            JLTU            DIVCALC2                ; --> First operand data is small
            JGTU            DIVCALC1                ; --> First operand data is large
            CMP.W           R1,R0
            JLTU            DIVCALC2                ; --> Second operand data is small
DIVCALC1:
            SUB.W           R1,R0
            SBB.W           R3,R2
            BSET            0,CALBUF[FB]            ; Sets bit of calculation result
DIVCALC2:
            SHL.W           #1,R0                   ; Shifts first operand
            ROLC.W          R2
            ADJNZ.B         #-1,COUNT[FB],DIVCALC   ; --> During calculation
;
```

```
;-------------------------------------------------------------------------
;           Adjusting digits
;-------------------------------------------------------------------------
            BTST        7,CALBUF+2[FB]
            JNE         FDIVSET              ; --> Digit adjustment finished
            SHL.W       #1,CALBUF[FB]        ; Adjusts digits of calculation data
            ROLC.W      CALBUF+2[FB]
            DEC.B       EXP[FB]              ; Adjusts exponent (exponent part – 1)
;
; Setting calculation result in return value
;
FDIVSET:
            MOV.W       CALBUF[FB],R0        ; Sets calculation result in mantissa (mid, lower)
            MOV.B       EXP[FB],R1H          ; Calculation result of exponent
            MOV.B       CALBUF+2[FB],R1L     ; Mantissa (upper)
            SHL.B       #1,R1L               ; Discards economized form bit
            SHL.B       #-1,SIGN[FB]         ; Sets sign in C flag
            RORC.W      R1                   ; Sets sign
            MOV.W       R1,R2                ; Sets sign, exponent part, and mantissa (upper)
                                             ; part calculation result
            EXITD
;
            .END
```

```
;*****************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 5          *
;                                                                                       *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                           *
;*****************************************************************************************************
                .GLB        FSIN
;
                .GLB        FADD            ; Floating-point addition
                .GLB        FSUB            ; Floating-point subtraction
                .GLB        FMUL            ; Floating-point multiplication
                .GLB        FDIV            ; Floating-point division
                .GLB        FCMP            ; Data comparison
                .GLB        FCAL            ; Table data calculation
                .GLB        FOVERCHK        ; Checks for overflow
;
VromTOP         .EQU        0F0000H         ; Declares start address of ROM
FBcnst          .EQU        001000H         ; Assumed FB register value
;
F2PI_H          .EQU        40C9H           ; 2π upper 2-byte value
F2PI_L          .EQU        0FDBH           ;      lower 2-byte value
FPAI_H          .EQU        4049H           ; π upper 2-byte value
FPAI_L          .EQU        0FDBH           ;    lower 2-byte value
FPI2_H          .EQU        3FC9H           ; π/2 upper 2-byte value
FPI2_L          .EQU        0FDBH           ;       lower 2-byte value
FOVER_H         .EQU        07F7FH          ; Overflow upper 2-byte value
FOVER_L         .EQU        0FFFFH          ;              lower 2-byte value
FUNDER_H        .EQU        0080H           ; Underflow upper 2-byte value
FUNDER_L        .EQU        0000H           ;                lower 2-byte value
;
SIGN            .EQU        –5              ; Sign of calculation result 0: plus; 1: minus
CO_OPE          .EQU        –4              ; Operand data (4 bytes)
;
                .SECTION    PROGRAM,ROMDATA
                .ORG        VromTOP
FSIT:
                .FLOAT      1.5148419E–4    ; 0.00015148419
                .FLOAT      –4.6737656E–3   ;–0.00467376557
                .FLOAT      7.9689679E–2    ; 0.07968967928
                .FLOAT      –6.4596371E–1   ;–0.64596371106
                .FLOAT      1.5707963       ; 1.57079631847
;
;=============================================================
;          Title: Sine function [SIN] (single-precision, floating-point)
;
;          Content of processing:
;                This program finds a sine of operand data (R2R0) and stores the result in R2, R0.
;                (R2R0) = SIN (R2R0)
;                The unit is radian.
;                Make sure the contents of R2 and R0 are smaller than 2π.
;
```

```
;           Procedure:
;                       (1)  Operand data (normalized single-precision, floating- point number)
;                            Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                            R2 and the mantissa (mid, lower) in register R0.
;                       (2)  Call the subroutine.
;                       (3)  The calculation result is placed in R2, R0.
;
;           Result:
;                       Result normal:
;                                   The C flag is reset to "0".
;                                   The calculation result is stored in R2, R0.
;                          R2 (High)                R2 (Low)                    R0H                 R0L
;                             ↑                        ↑                         ↑                   ↑
;           Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                       Result erratic:
;                                   The C flag is set to "1".
;                                   The following value is returned in R2, R0.
;                       -------------------------------------------------------
;                        Contents of R2, R0                     Meaning
;                       -------------------------------------------------------
;                        Maximum value                          Overflow
;                       -------------------------------------------------------
;
;           Input:      -------------------------------> Output:
;
;           R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;           R1 ( )                              R1 (Indeterminate)
;           R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;           R3 ( )                              R3 (Indeterminate)
;           A0 ( )                              A0 (Indeterminate)
;           A1 ( )                              A1 (Indeterminate)
;
;           Stack amount used: 34 bytes
;==================================================================
                        .SECTION        PROGRAM,CODE
                        .FB             FBcnst              ; Assumes FB register value
FSIN:
            ENTER           #6                              ; Allocates internal variables
            MOV.W           R2,CO_OPE+2[FB]                 ; Saves operand data in variables
            MOV.W           R0,CO_OPE[FB]
;
; Checking overflow
;
            MOV.W           CO_OPE+2[FB],R0                 ; Reads exponent part
            SHL.W           #1,R0                           ; Discards sign and align to R0H
            CMP.B           #98H,R0H                        ; Overflow?
            JGEU            SINOVER                         ; --> Yes
;
; Checking sign
;
            BTSTC           7,CO_OPE+3[FB]                  ; Sign is positive? (sign cleared)
            STZX            #0,#1,SIGN[FB]                  ; Sets sign
;
```

```
; Adjusting data to 2π or less
;
          MOV.W      CO_OPE+2[FB],R2      ; Reads exponent and mantissa (upper) parts of
                                          ; operand data
          MOV.W      CO_OPE[FB],R0        ; Reads mantissa (mid, lower) part of operand data
F2P_LOOP:
          MOV.W      #F2PI_H,R3           ; Sets 2π
          MOV.W      #F2PI_L,R1
          JSR        FCMP                 ; Operand data ≥ 2π?
          JNC        FSIN10               ; --> Operand data < 2π
          JNE        F2P_OVER             ; --> Operand data > 2π
          MOV.B      #0,SIGN[FB]          ; Sets sign positive
F2P_OVER:
          JSR        FSUB                 ; (R2 R0) ← operand data – 2π
          JSR        FOVERCHK             ; Checks for overflow
          JNC        F2P_LOOP             ; Looped until 2π or less
;
; Setting overflow information (maximum) value
;
SINOVER:
          MOV.W      #FOVER_H,R2          ; Sets maximum value in return value
          MOV.W      #FOVER_L,R0
          FSET       C                    ; Sets “result erratic” information
          EXITD
;
; Inverting sign of π to 2π and reducing it to below π
;
FSIN10:
          MOV.W      #FPAI_H,R3           ; Sets π
          MOV.W      #FPAI_L,R1
          JSR        FCMP                 ; Operand data ≥ π?
          JNC        FSIN20               ; --> Operand < π
          JNE        FSIN15               ; --> Operand > π
          MOV.B      #01H,SIGN[FB]        ; Changes sign negative (to make it positive)
FSIN15:
          XOR.B      #01H,SIGN[FB]        ; Inverts sign
          JSR        FSUB                 ; (R2 R0) ← operand data – π
          JSR        FOVERCHK             ; Checks for overflow
          JC         SINOVER              ; --> Overflow
;
; Converting π/2 to π into data π/2 or less
;
FSIN20:
          MOV.W      #FPI2_H,R3           ; Sets π/2
          MOV.W      #FPI2_L,R1
          JSR        FCMP                 ; Operand data ≥ π?
          JNC        FSIN30               ; --> Operand < π/2
;
          BSET       15,R2                ; Changes data negative
          MOV.W      #FPAI_H,R3           ; Sets π
          MOV.W      #FPAI_L,R1
          JSR        FADD                 ; Adds π to get 0 to π/2
          JSR        FOVERCHK             ; Checks for overflow
          JC         SINOVER              ; --> Overflow
;
```

```
; Operand data ÷ π/2
;
FSIN30:
        MOV.W       #FPI2_H,R3          ; Sets π/2
        MOV.W       #FPI2_L,R1
        JSR         FDIV                ; Data ÷ π/2
        JSR         FOVERCHK            ; Checks for overflow
        JC          SINOVER             ; --> Overflow
        MOV.W       R2,CO_OPE+2[FB]     ; Saves calculation data
        MOV.W       R0,CO_OPE[FB]
;
        MOV.W       R2,R3               ; Sets data
        MOV.W       R0,R1
        JSR         FMUL                ; Squares data
        JSR         FOVERCHK            ; Checks for overflow
        JC          SINOVER             ; --> Overflow
;
        MOV.W       #FSIT&0FFFFH,A0     ; Sets data table address
        MOV.W       #FSIT>>16,A1
        MOV.B       #5-1,R1L            ; Sets number of tables
        JSR         FCAL                ; Calculates table data
        JC          SINOVER             ; --> Overflow
;
        MOV.W       CO_OPE+2[FB],R3     ; Restores calculation data
        MOV.W       CO_OPE[FB],R1
        JSR         FMUL                ; Table calculation data x calculation data
        JSR         FOVERCHK            ; Checks for overflow
        JC          SINOVER             ; --> Overflow
;
        SHL.W       #1,R2
        RORC.B      SIGN[FB]            ; Sign inverting information → C flag
        RORC.W      R2                  ; Sets sign
        FCLR        C                   ; Sets “result normal” information
        EXITD
;
        .END
```

```
;*************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 6          *
;                                                                                          *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                            *
;*************************************************************************************************
                .GLB            FCOS
;
                .GLB            FSIN            ; Sine function [SIN]
                .GLB            FADD            ; Floating-point addition
                .GLB            FOVERCHK        ; Checks for overflow
;
VromTOP         .EQU            0F0000H         ; Declares start address of ROM
;
FPI2_H          .EQU            3FC9H           ; π/2 upper 2-byte value
FPI2_L          .EQU            0FDBH           ;      lower 2-byte value
FOVER_H         .EQU            07F7FH          ; Overflow upper-2 byte value
FOVER_L         .EQU            0FFFFH          ;            lower-2 byte value
;
;============================================================================
;           Title: Cosine [COS] (single-precision, floating-point)
;
;           Content of processing:
;                   This program finds a cosine of operand data (R2R0) and stores the result in R2, R0.
;                   (R2R0) = COS (R2R0)
;                   The unit is radian.
;                   Make sure the contents of R2 and R0 are smaller than 2π
;
;           Procedure:
;                   (1)  Operand data (normalized single-precision, floating- point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R2 and the mantissa (mid, lower) in register R0.
;                   (2)  Call the subroutine.
;                   (3)  The calculation result is placed in R2, R0.
;
;           Result:
;                   Result normal:
;                           The C flag is reset to "0".
;                           The calculation result is stored in R2, R0.
;                     R2 (High)                R2 (Low)                     R0H              R0L
;                        ↑                        ↑                            ↑                ↑
;           Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                   Result erratic:
;                           The C flag is set to "1".
;                           The following value is returned in R2, R0.
;                   ---------------------------------------------
;                    Contents of R2, R0              Meaning
;                   ---------------------------------------------
;                    Maximum value                   Overflow
;                   ---------------------------------------------
;
;
```

```
;           Input:    --------------------------------> Output:
;
;           R0 (Lower half of operand data)      R0 (Lower half of calculation result)
;           R1 ( )                               R1 (Indeterminate)
;           R2 (Upper half of operand data)      R2 (Upper half of calculation result)
;           R3 ( )                               R3 (Indeterminate)
;           A0 ( )                               A0 (Indeterminate)
;           A1 ( )                               A1 (Indeterminate)
;
;           Stack amount used: 34 bytes
;==============================================================
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
FCOS:
            MOV.W           #FPI2_H,R3              ; Sets π/2
            MOV.W           #FPI2_L,R1
            JSR             FADD                    ; Data (R2 R0) + π/2
            JSR             FOVERCHK                ; Checks for overflow
            JC              COSOVER                 ; --> Overflow
            JMP             FSIN                    ; Calculates SIN by advancing π/2 from COS
                                                    ; Calculated value is returned as COS data
;
; Setting overflow information (maximum value)
;
COSOVER:
            MOV.W           #FOVER_H,R2             ; Sets maximum value in return value
            MOV.W           #FOVER_L,R0
            FSET            C                       ; Sets “result erratic” information
            RTS
;
            .END
```

```
;****************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 7            *
;                                                                                          *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                             *
;****************************************************************************************************
                .GLB        FTAN
;
                .GLB        FDIV            ; Floating-point division
                .GLB        FSIN            ; Sine function [SIN]
                .GLB        FCOS            ; Cosine function [COS]
                .GLB        FOVERCHK        ; Checks for overflow
;
VromTOP         .EQU        0F0000H         ; Declares start address of ROM
FBcnst          .EQU        001000H         ; Assumed FB register value
;
FOVER_H         .EQU        07F7FH          ; Overflow upper-2 byte value
FOVER_L         .EQU        0FFFFH          ;           lower-2 byte value
;
COSDAT          .EQU        –8              ; COS calculation result
CO_OPE          .EQU        –4              ; Operand data (4 bytes)
;
;==================================================================
;           Title: Tangent [TAN] (single-precision, floating-point)
;
;           Content of processing:
;                       This program finds a tangent of operand data (R2R0) and stores the result in R2, R0.
;                       (R2R0) = TAN (R2R0)
;                       The unit is radian.
;                       Make sure the contents of R2 and R0 are smaller than 2π.
;
;           Procedure:
;                       (1)  Operand data (normalized single-precision, floating- point number)
;                            Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                            R2 and the mantissa (mid, lower) in register R0.
;                       (2)  Call the subroutine.
;                       (3)  The calculation result is placed in R2, R0.
;
;           Result:
;                       Result normal:
;                                   The C flag is reset to “0”.
;                                   The calculation result is stored in R2, R0.
;                         R2 (High)                R2 (Low)                R0H              R0L
;                            ↑                         ↑                     ↑                ↑
;           Sign, Exponent b7 to b1     Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;
;                       Result erratic:
;                                   The C flag is set to “1”.
;                                   The following value is returned in R2, R0.
;                       ---------------------------------------------------------
;                       Contents of R2, R0                    Meaning
;                       ---------------------------------------------------------
;                       Maximum value                         Overflow
;                       ---------------------------------------------------------
;
```

```
;
;           Input:      ---------------------------------> Output:
;
;           R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;           R1 ( )                              R1 (Indeterminate)
;           R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;           R3 ( )                              R3 (Indeterminate)
;           A0 ( )                              A0 (Indeterminate)
;           A1 ( )                              A1 (Indeterminate)
;
;           Stack amount used: 41 bytes
;================================================================
                    .SECTION    PROGRAM,CODE
                    .ORG        VromTOP
                    .FB         FBcnst          ; Assumes FB register value
FTAN:
        ENTER       #8                          ; Allocates internal variables
        MOV.W       R2,CO_OPE+2[FB]             ; Saves operand data in variables
        MOV.W       R0,CO_OPE[FB]
;
        JSR         FCOS                        ; COS calculation
        JC          TANERR                      ; --> Overflow
;
        MOV.W       R2,COSDAT+2[FB]             ; Stores COS calculation result
        MOV.W       R0,COSDAT[FB]
;
        MOV.W       CO_OPE+2[FB],R2             ; Sets operand data
        MOV.W       CO_OPE[FB],R0
;
        JSR         FSIN                        ; SIN calculation
        JC          TANERR                      ; --> Overflow
;
        MOV.W       COSDAT+2[FB],R3             ; Sets COS calculation result in operand data
        MOV.W       COSDAT[FB],R1
;
        JSR         FDIV                        ; TAN = SIN/COS
        JSR         FOVERCHK                    ; Overflow check
        JC          TANERR                      ; --> Overflow
;
        FCLR        C                           ; Sets “result normal” information
        EXITD
;
; Setting overflow information (maximum value)
;
TANERR:
        MOV.W       #FOVER_H,R2                 ; Sets maximum value in return value
        MOV.W       #FOVER_L,R0
        FSET        C                           ; Sets “result erratic” information
        EXITD
;
        .END
```

```
;*************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 8            *
;                                                                                          *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                              *
;*************************************************************************************************
                    .GLB          FASN
;
                    .GLB          FOVERCHK      ; Overflow check
                    .GLB          FADD          ; Floating-point addition
                    .GLB          FMUL          ; Floating-point multiplication
                    .GLB          FDIV          ; Floating-point division
                    .GLB          FSQR          ; Square root
                    .GLB          FATN          ; Inverse tangent
;
VromTOP             .EQU          0F0000H       ; Declares start address of ROM
FBcnst              .EQU          001000H       ; Assumed FB register value
;
FNO1_H              .EQU          3F80H         ; Numeral 1 upper 2-byte value
FNO1_L              .EQU          0000H         ;           lower 2-byte value
FPI2_H              .EQU          3FC9H         ; π/2 upper 2-byte value
FPI2_L              .EQU          0FDBH         ;      lower 2-byte value
FOVER_H             .EQU          07F7FH        ; Overflow upper-2 byte value
FOVER_L             .EQU          0FFFFH        ;          lower-2 byte value
;
CO_OPE              .EQU          –4            ; Operand data (4 bytes)
;==============================================================
;          Title: Inverse sine function [SIN (raised to power of -1)  (single-precision, floating-point)
;          Content of processing:
;                    This program finds an inverse sine of operand data (R2R0) and stores the result in R2, R0.
;                    (R2R0) = SIN –1 (R2R0)
;          Procedure:
;                    (1) Operand data (normalized single-precision, floating- point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R2 and the mantissa (mid, lower) in register R0.
;                    (2)  Call the subroutine.
;                    (3)  The calculation result is placed in R2, R0.
;          Result:
;                    The unit is radian.
;                    Result normal:
;                              The C flag is reset to “0”.
;                              The calculation result is stored in R2, R0.
;                      R2 (High)              R2 (Low)                      R0H              R0L
;                          ↑                      ↑                            ↑                ↑
;          Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                    Result erratic:
;                              The C flag is set to “1”.
;                              The following value is returned in R2, R0.
;                    ---------------------------------------------------
;                    Contents of R2, R0             Meaning
;                    ---------------------------------------------------
;                    Maximum value                  Overflow
;                    ---------------------------------------------------
;                    Non-numeral                    Argument error
;                    ---------------------------------------------------
```

```
;           Input:     --------------------------------> Output:
;
;           R0 (Lower half of operand data)    R0 (Lower half of calculation result)
;           R1 ( )                             R1 (Indeterminate)
;           R2 (Upper half of operand data)    R2 (Upper half of calculation result)
;           R3 ( )                             R3 (Indeterminate)
;           A0 ( )                             A0 (Indeterminate)
;           A1 ( )                             A1 (Indeterminate)
;
;           Stack amount used: 60 bytes
;==================================================================
                .SECTION        PROGRAM,CODE
                .ORG            VromTOP
                .FB             FBcnst          ; Assumes FB register value
FASN:
        ENTER           #4                      ; Allocates internal variables
        MOV.W           R2,CO_OPE+2[FB]         ; Saves operand data in variables
        MOV.W           R0,CO_OPE[FB]
;
; Checking argument error (check of 1 or less)
        BCLR            15,R2                   ; Clears sign
        CMP.W           #3F80H,R2               ; Operand data less than 1?
        JLTU            FASN10                  ; --> Less than 1 (no error)
        JGTU            FASNERR                 ; --> Larger than 1 (error)
        CMP.W           #0,R0                   ; Exactly 1?
        JEQ             FASN1SET                ; --> Yes (no error)
;
; Setting argument error information (non-numeral)
FASNERR:
        MOV.W           CO_OPE+2[FB],R2         ; Sets overflow in return value
        OR.W            #7FFFH,R2               ; Returns same sign as that of argument
        MOV.W           #0FFFFH,R0
        FSET            C                       ; Sets “result erratic” information
        EXITD
;
;------------------------------------------------------------------
;           Setting π/2
;------------------------------------------------------------------
FASN1SET:
        MOV.W           #FPI2_L,R0              ; Sets π/2 lower 2-byte value
        MOV.W           #FPI2_H,R2              ; Sets π/2 upper 2-byte value
        SHL.W           #1,R2
        MOV.B           CO_OPE+3[FB],R1L
        SHL.B           #1,R1L                  ; Sign →C flag
        RORC.W          R2                      ; Sets sign
        FCLR            C                       ; Sets “result normal” information
        EXITD
;
; Calculation formula  Operand data ÷ (1 – square of operand data)
FASN10:
        MOV.W           R2,R3                   ; Operand data →calculation data
        MOV.W           R0,R1
        JSR             FMUL                    ; Squares operand data
        JSR             FOVERCHK                ; Checks for overflow
        JC              ASNOVER                 ; --> Overflow
;
```

```
        BSET        15,R2               ; Changes sign negative
        MOV.W       #FNO1_H,R3          ; Sets numeral 1 in operand data
        MOV.W       #FNO1_L,R1
        JSR         FADD                ; R2, R0 = 1 – (square of operand data)
        JSR         FOVERCHK            ; Checks for overflow
        JC          ASNOVER             ; --> Overflow
        JSR         FSQR                ; Square root of calculation result
        JC          ASNOVER             ; --> Overflow
;
        MOV.W       R2,R3               ; Calculation result → operand data
        MOV.W       R0,R1
        MOV.W       CO_OPE[FB],R0       ; Reads operand data
        MOV.W       CO_OPE+2[FB],R2
        JSR         FDIV                ; Divides operand data by calculation result
        JSR         FOVERCHK            ; Checks for overflow
        JC          ASNOVER             ; --> Overflow
        JSR         FATN                ; Inverse tangent of calculation result
        JSR         FOVERCHK            ; Checks for overflow
        JC          ASNOVER             ; --> Overflow
        FCLR        C                   ; Sets “result normal” information
        EXITD
;
; Setting overflow information (maximum)
;
ASNOVER:
        MOV.W       CO_OPE[FB],R2
        AND.W       #8000H,R2           ; Clears all but sign
        OR.W        #FOVER_H,R2         ; Sets maximum data in return value
        MOV.W       #FOVER_L,R0
        FSET        C                   ; Sets “result erratic” information
        EXITD
;
        .END
```

```
;*****************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 9              *
;                                                                                             *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                                *
;*****************************************************************************************************
                .GLB        FACN
;
                .GLB        FOVERCHK    ; Overflow check
                .GLB        FADD        ; Floating-point addition
                .GLB        FMUL        ; Floating-point multiplication
                .GLB        FDIV        ; Floating-point division
                .GLB        FSQR        ; Square root
                .GLB        FATN        ; Inverse tangent
;
VromTOP         .EQU        0F0000H     ; Declares start address of ROM
FBcnst          .EQU        001000H     ; Assumed FB register value
;
FNO1_H          .EQU        3F80H       ; Numeral 1 upper-2 byte value
FNO1_L          .EQU        0000H       ;           lower-2 byte value
FPAI_H          .EQU        4049H       ; π upper 2-byte value
FPAI_L          .EQU        0FDBH       ;   lower 2-byte value
FPI2_H          .EQU        3FC9H       ; π/2 upper 2-byte value
FPI2_L          .EQU        0FDBH       ;     lower 2-byte value
FOVER_H         .EQU        07F7FH      ; Overflow upper 2-byte value
FOVER_L         .EQU        0FFFFH      ;          lower 2-byte value
;
CO_OPE          .EQU        –4          ; Operand data (4 bytes)
;
;==================================================================
;       Title: Inverse cosine function [COS (raised to power of –1)  (single-precision, floating-point)
;       Content of processing:
;               This program finds an inverse cosine of operand data (R2R0) and stores the result in R2, R0.
;               (R2R0) = COS –1 (R2R0)
;       Procedure:
;               (1)  Operand data (normalized single-precision, floating- point number)
;                    Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                    R2 and the mantissa (mid, lower) in register R0.
;               (2)  Call the subroutine.
;               (3)  The calculation result is placed in R2, R0.
;       Result:
;               The unit is radian.
;               Result normal:
;                       The C flag is reset to “0”.
;                       The calculation result is stored in R2, R0.
;                  R2 (High)              R2 (Low)                 R0H              R0L
;                     ↑                      ↑                      ↑                ↑
;       Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;               Result erratic:
;                       The C flag is set to “1”.
;                       The following value is returned in R2, R0.
;               ----------------------------------------------------
;               Contents of R2, R0            Meaning
;               ----------------------------------------------------
;               Maximum value                 Overflow
;               ----------------------------------------------------
;               Non-numeral                   Argument error
;               ----------------------------------------------------
;
```

```
;           Input:    ------------------------------->  Output:
;
;           R0 (Lower half of operand data)    R0 (Lower half of calculation result)
;           R1 ( )                             R1 (Indeterminate)
;           R2 (Upper half of operand data)    R2 (Upper half of calculation result)
;           R3 ( )                             R3 (Indeterminate)
;           A0 ( )                             A0 (Indeterminate)
;           A1 ( )                             A1 (Indeterminate)
;
;           Stack amount used: 60 bytes
;==================================================================
                 .SECTION         PROGRAM,CODE
                 .ORG             VromTOP
                 .FB              FBcnst          ; Assumes FB register value
FACN:
         ENTER          #4                        ; Allocates internal variables
         MOV.W          R2,CO_OPE+2[FB]           ; Saves operand data in variables
         MOV.W          R0,CO_OPE[FB]
;
; Checking argument error (check of 1 or less)
;
         BCLR           15,R2                     ; Clears sign
         CMP.W          #3F80H,R2                 ; Operand data less than 1?
         JLTU           FACN10                    ; --> Smaller than 1
         JGTU           FACNERR                   ; --> Larger than 1 (error)
         CMP.W          #0,R0                     ; Exactly 1?
         JGTU           FACNERR                   ; --> Larger than 1 (error)
FACN10:
         OR.W           R2,R0                     ; Data 0?
         JNE            FACN20                    ; --> No
;------------------------------------------------------------------
;        Setting π/2
;------------------------------------------------------------------
         MOV.W          #FPI2_L,R0                ; Sets π/2 lower 2-byte value
         MOV.W          #FPI2_H,R2                ; Sets π/2 upper 2-byte value
         BTST           7,CO_OPE+3[FB]            ; Sign is negative?
         BMNZ           15,R2                     ; Changes sign negative
         FCLR           C                         ; Sets “result normal” information
         EXITD
;
; Setting argument error information (non-numeral)
;
FACNERR:
         MOV.W          CO_OPE+2[FB],R2           ; Sets overflow in return value
         OR.W           #7FFFH,R2                 ; Returns same sign as that of argument
         MOV.W          #0FFFFH,R0
         FSET           C                         ; Sets “result erratic” information
         EXITD
;
; Calculation formula →√ ̄(1 – square of operand data) ÷ operand data
;
```

```
FACN20:
        MOV.W       CO_OPE[FB],R0        ; Reads operand data
        MOV.W       CO_OPE+2[FB],R2
;
        MOV.W       R2,R3                ; Operand data →calculation data
        MOV.W       R0,R1
        JSR         FMUL                 ; Squares operand data
        JSR         FOVERCHK             ; Checks for overflow
        JC          ACNOVER              ; --> Overflow
;
        BSET        15,R2                ; Changes sign negative
        MOV.W       #FNO1_H,R3           ; Sets numeral 1 in calculation data
        MOV.W       #FNO1_L,R1
        JSR         FADD                 ; R2, R0 = 1 – (square of operand data)
        JSR         FOVERCHK             ; Checks for overflow
        JC          ACNOVER              ; --> Overflow
        JSR         FSQR                 ; Square root of calculation result
        JC          ACNOVER              ; --> Overflow
;
        MOV.W       CO_OPE[FB],R1        ; Reads operand data
        MOV.W       CO_OPE+2[FB],R3
        JSR         FDIV                 ; Divides calculation result by operand data
        JSR         FOVERCHK             ; Checks for overflow
        JC          ACNOVER              ; --> Overflow
        JSR         FATN                 ; Inverse tangent of calculation result
        JSR         FOVERCHK             ; Checks for overflow
        JC          ACNOVER              ; --> Overflow
;
        BTST        7,CO_OPE+3[FB]       ; Sign is negative?
        JEQ         FACN_OK              ; --> No
;
; Calculation result + π
;
        MOV.W       #FPAI_H,R3           ; Sets π
        MOV.W       #FPAI_L,R1
        JSR         FADD                 ; Calculation result + π
        JSR         FOVERCHK             ; Checks for overflow
        JC          ACNOVER              ; --> Overflow
FACN_OK:
        FCLR        C                    ; Sets “result normal” information
        EXITD
;
; Setting overflow information (maximum value)
;
ACNOVER:
        MOV.W       CO_OPE[FB],R2
        AND.W       #8000H,R2            ; Clears all but sign
        OR.W        #FOVER_H,R2          ; Sets maximum value in return value
        MOV.W       #FOVER_L,R0
        FSET        C                    ; Sets “result erratic” information
        EXITD
;
        .END
```

```
;************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 10          *
;                                                                                          *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                             *
;************************************************************************************************
                .GLB        FATN
;
                .GLB        FCMP            ; Large/small comparison
                .GLB        FOVERCHK        ; Overflow check
                .GLB        FCAL            ; Table data calculation
                .GLB        FADD            ; Floating-point addition
                .GLB        FMUL            ; Floating-point multiplication
                .GLB        FDIV            ; Floating-point division
;
VromTOP         .EQU        0F0000H         ; Declares start address of ROM
FBcnst          .EQU        001000H         ; Assumed FB register value
;
FNO1_H          .EQU        3F80H           ; Numeral 1 upper 2-byte value
FNO1_L          .EQU        0000H           ;           lower 2-byte value
FPI2_H          .EQU        3FC9H           ; π/2 upper 2-byte value
FPI2_L          .EQU        0FDBH           ;     lower 2-byte value
FOVER_H         .EQU        07F7FH          ; Overflow upper 2-byte value
FOVER_L         .EQU        0FFFFH          ;          lower 2-byte value
;
OVER1           .EQU        –6              ; 0: 1 or less ; 1: greater than 1
SIGN            .EQU        –5              ; 0: plus ; 1: minus
CO_OPE          .EQU        –4              ; Operand data (4 bytes)
;
                .SECTION    PROGRAM,ROMDATA
                .ORG        VromTOP
FATT:
                .FLOAT      6.812411E–3     ;  0.006812411   (C13)
                .FLOAT      –3.3606269E–2   ; –0.033606269   (C11)
                .FLOAT      7.9626318E–2    ;  0.079626318   (C9)
                .FLOAT      –1.3233510E–1   ; –0.132335096   (C7)
                .FLOAT      1.9807869E–1    ;  0.198078690   (C5)
                .FLOAT      –3.3317376E–1   ; –0.333173758   (C3)
                .FLOAT      9.9999612E–1    ;  0.999996115   (C1)
;
;               .LWORD      03BDF3AA4H      ;  0.006812411   (C13)
;               .LWORD      0BD09A6BAH      ; –0.033606269   (C11)
;               .LWORD      03DA3131EH      ;  0.079626318   (C9)
;               .LWORD      0BE0782D8H      ; –0.132335096   (C7)
;               .LWORD      03E4AD522H      ;  0.198078690   (C5)
;               .LWORD      0BEAA95C0H      ; –0.333173758   (C3)
;               .LWORD      03F7FFFBEH      ;  0.999996115   (C1)
;
;================================================================
;           Title: Inverse tangent function [TAN (raised to power of –1)  (single-precision, floating-point)
;
;           Content of processing:
;               This program finds an inverse tangent of operand data (R2R0) and stores the result in R2, R0.
;               (R2R0) = TAN–1 (R2R0)
;
```

```
;           Procedure:
;                     (1)  Operand data (normalized single-precision, floating- point number)
;                          Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                          R2 and the mantissa (mid, lower) in register R0.
;                     (2)  Call the subroutine.
;                     (3)  The calculation result is placed in R2, R0.
;
;           Result:
;                     The unit is radian.
;                     Result normal:
;                               The C flag is reset to "0".
;                               The calculation result is stored in R2, R0.
;                        R2 (High)            R2 (Low)                 R0H                R0L
;                           ↑                    ↑                      ↑                  ↑
;           Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)  Mantissa (mid)   Mantissa (lower)
;
;                     Result erratic:
;                               The C flag is set to "1".
;                               The following value is returned in R2, R0.
;
;                     Contents of R2, R0           Meaning
;
;                     Maximum value                Overflow
;
;
;           Input:    -------------------------------> Output:
;
;           R0 (Lower half of operand data)   R0 (Lower half of calculation result)
;           R1 ( )                            R1 (Indeterminate)
;           R2 (Upper half of operand data)   R2 (Upper half of calculation result)
;           R3 ( )                            R3 (Indeterminate)
;           A0 ( )                            A0 (Indeterminate)
;           A1 ( )                            A1 (Indeterminate)
;
;           Stack amount used: 34 bytes
;==============================================================
                    .SECTION      PROGRAM,CODE
                    .FB           FBcnst          ; Assumes FB register value
FATN:
          ENTER          #6                       ; Allocates internal variables
          MOV.W          R2,CO_OPE+2[FB]          ; Saves operand data in variables
          MOV.W          R0,CO_OPE[FB]
;
; Checking sign
;
          BTSTC          15,R2                    ; Checks sign (sign cleared)
          STZX           #0,#1,SIGN[FB]           ; Sets sign information
;
; Checking for unsigned data equal to or less than 1
;
          MOV.B          #0,OVER1[FB]             ; Sets "equal to or less than 1" information
          MOV.W          #FNO1_H,R3               ; Sets floating-point number 1
          MOV.W          #FNO1_L,R1
          JSR            FCMP                     ; Compares
          JLTU           FATN20                   ; --> 1 or less
          INC.B          OVER1[FB]                ; Sets "greater than 1" information
;
```

```
; Checking absolute 0
;
FATN20:
        CMP.W       R2,R0               ; Absolute 0?
        JNE         FATN30              ; --> No
;
; Returning absolute 0 information
;
        MOV.W       CO_OPE[FB],R0       ; Returns data that was input
        MOV.W       CO_OPE+2[FB],R2
        EXITD
;
FATN30:
        MOV.W       R2,CO_OPE+2[FB]     ; Saves unsigned operand data
;
        CMP.B       #0,OVER1[FB]        ; Unsigned data is equal to or less than 1?
        JEQ         FATN40              ; --> Yes
        XCHG.W      R2,R3               ; Floating-point number 1 →(R2, R0)
        XCHG.W      R0,R1               ; Unsigned operand data →(R3, R1)
        JSR         FDIV                ; Divides 1 by unsigned operand data
        JSR         FOVERCHK            ; Checks for overflow
        JC          ATNOVER             ; --> Overflow
;
        MOV.W       R2,CO_OPE+2[FB]     ; Saves calculation result
        MOV.W       R0,CO_OPE[FB]
FATN40:
        MOV.W       R2,R3               ; Calculation result  →(R3, R1)
        MOV.W       R0,R1
        JSR         FMUL                ; Squares calculation result
        JSR         FOVERCHK            ; Checks for overflow
        JC          ATNOVER             ; --> Overflow
;
        MOV.W       #FATT&0FFFFH,A0     ; Sets data table address
        MOV.W       #FATT>>16,A1
        MOV.B       #7-1,R1L            ; Sets number of tables
        JSR         FCAL                ; Calculates table data
        JC          ATNOVER             ; --> Overflow
;
        MOV.W       CO_OPE[FB],R1       ; Reads saved data
        MOV.W       CO_OPE+2[FB],R3
        JSR         FMUL                ; Multiplies result by saved data
        JSR         FOVERCHK            ; Checks for overflow
        JC          ATNOVER             ; --> Overflow
;
        CMP.B       #0,OVER1[FB]        ; "Equal to or less than 1" information?
        JEQ         FATN50              ; --> Yes
;
        BSET        15,R2               ; Changes calculation result negative
        MOV.W       #FPI2_H,R3          ; Sets π/2
        MOV.W       #FPI2_L,R1
        JSR         FADD                ; Subtracts calculation result from (π/2)
        JSR         FOVERCHK            ; Checks for overflow
        JC          ATNOVER             ; --> Overflow
```

```
FATN50:
        SHL.B       #-1,SIGN[FB]        ; Sign information →C flag
        BMC         15,R2               ; Inverts sign if sign information is negative
        FCLR        C                   ; Sets "result normal" information
        EXITD
;
; Setting overflow information (maximum value)
;
ATNOVER:
        MOV.W       CO_OPE[FB],R2
        AND.W       #8000H,R2           ; Clears all but sign
        OR.W        #FOVER_H,R2         ; Sets maximum value in return value
        MOV.W       #FOVER_L,R0
        FSET        C                   ; Sets "result erratic" information
        EXITD
;
        .END
```

```
;*************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 11          *
;                                                                                          *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                              *
;*************************************************************************************************
                    .GLB            FSQR
;
                    .GLB            FPOW            ; Power calculation
;
VromTOP             .EQU            0F0000H         ; Declares start address of ROM
;
FP5_H               .EQU            3F00H           ; 0.5 upper 2-byte value
FP5_L               .EQU            0000H           ;       lower 2-byte value
;
;===============================================================
;          Title: Square root (single-precision, floating-point)
;
;          Content of processing:
;                    This program finds a square root of operand data (R2R0) and stores the result in R2, R0.
;                    (R2R0) = √(R2R0)
;
;          Procedure:
;                    (1)  Operand data (normalized single-precision, floating- point number)
;                         Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                         R2 and the mantissa (mid, lower) in register R0.
;                    (2)  Call the subroutine.
;                    (3)  The calculation result is placed in R2, R0.
;
;          Result:
;                    Result normal:
;                              The C flag is reset to “0”.
;                              The calculation result is stored in R2, R0.
;                       R2 (High)              R2 (Low)                   R0H                R0L
;                          ↑                      ↑                        ↑                  ↑
;          Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                    Result erratic:
;                              The C flag is set to “1”.
;                              The following value is returned in R2, R0.
;                    ---------------------------------------------------
;                    Contents of R2, R0                 Meaning
;                    ---------------------------------------------------
;                    Non-numeral                        Calculation error
;                    ---------------------------------------------------
;                    Maximum value                      Overflow
;                    ---------------------------------------------------
;
```

```
;           Input:     -------------------------------> Output:
;
;           R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;           R1 ( )                               R1 (Indeterminate)
;           R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;           R3 ( )                               R3 (Indeterminate)
;           A0 ( )                               A0 (Indeterminate)
;           A1 ( )                               A1 (Indeterminate)
;
;           Stack amount used: 53 bytes
;===============================================================
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
FSQR:
            MOV.W           #FP5_H,R3               ; Sets 0.5
            MOV.W           #FP5_L,R1
            JSR             FPOW                    ; Calculates a product of the operand data raised to
                                                    ; the power of 0.5
            RTS
;
            .END
```

```
;***********************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 12          *
;                                                                                        *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                            *
;***********************************************************************************************
                    .GLB            FPOW
;
                    .GLB            FOVERCHK        ; Overflow check
                    .GLB            FIXI            ; Floating data →integer conversion processing
                    .GLB            FLN             ; Natural logarithmic calculation
                    .GLB            FEXP            ; Exponential function calculation
                    .GLB            FMUL            ; Floating-point multiplication
;
VromTOP             .EQU            0F0000H         ; Declares start address of ROM
FBcnst              .EQU            001000H         ; Assumed FB register value
;
FP5_H               .EQU            3F00H           ; 0.5 upper 2-byte value
FP5_L               .EQU            0000H           ;       lower 2-byte value
;
SIGN                .EQU            –9              ; Sign of calculation result  0: plus; 1: minus
POWER               .EQU            –8              ; Multiplication data (4 bytes)
CO_OPE              .EQU            –4              ; Operand data (4 bytes)
;
;=================================================================
;          Title: Power (single-precision, floating-point)
;          Content of processing:
;                    This program finds a product of operand data (R2R0) raised to the power (R3R1) and
;                    stores the result in R2, R0.
;                    (R2R0) = (R2R0) (R3R1)
;          Procedure:
;                    (1)  Operand data (normalized single-precision, floating- point number)
;                         Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                         R2 and the mantissa (mid, lower) in register R0.
;                    (2)  Exponent data (normalized single-precision, floating-point number)
;                         Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                         R3 and the mantissa (mid, lower) in register R1.
;                    (3)  Call the subroutine.
;                    (4)  The calculation result is placed in R2, R0.
;          Result:
;                    Result normal:
;                              The C flag is reset to “0”.
;                              The calculation result is stored in R2, R0.
;                    R2 (High)                 R2 (Low)                    R0H              R0L
;                         ↑                         ↑                        ↑                ↑
;          Sign, Exponent b7 to b1    Exponent b0, Mantissa (upper)    Mantissa (mid)    Mantissa (lower)
;
;                    Result erratic:
;                              The C flag is set to “1”.
;                              The following value is returned in R2, R0.
;                    ------------------------------------------------------------
;                    Contents of R2, R0                     Meaning
;                    ------------------------------------------------------------
;                    Non-numeral                            Calculation error
;                    ------------------------------------------------------------
;                    Maximum value                          Overflow
;                    ------------------------------------------------------------
;
```

```
;           Input:     -------------------------------> Output:
;
;           R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;           R1 (Lower half of exponent data)    R1 (Indeterminate)
;           R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;           R3 (Upper half of exponent data)    R3 (Indeterminate)
;           A0 ( )                              A0 (Indeterminate)
;           A1 ( )                              A1 (Indeterminate)
;
;           Stack amount used: 50 bytes
;=================================================================
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
                    .FB             FBcnst              ; Assumes FB register value
FPOW:
          ENTER           #9                            ; Allocates internal variables
          MOV.W           R2,CO_OPE+2[FB]               ; Saves operand data in variables
          MOV.W           R0,CO_OPE[FB]
          MOV.W           R3,POWER+2[FB]                ; Saves exponent data
          MOV.W           R1,POWER[FB]
;
; Checking exponent data = 0
;
          CMP.W           #0,R1                         ; Exponent data is 0?
          JNE             FPOW0                         ; --> No
          AND.W           #7FFFH,R3                     ; Exponent data is 0?
          JNE             FPOW0                         ; --> No
;-------------------------------------------------------------------
;         Setting result = 1
;-------------------------------------------------------------------
          MOV.W           #0,R0                         ; Sets 1 in return value
          MOV.W           #3F80H,R2
          FCLR            C                             ; Sets “result normal” information
          EXITD
;
; Checking error & result = 0
;
FPOW0:
          CMP.W           #0,R0                         ; Operand data is 0?
          JNE             FPOW1                         ; --> No
          AND.W           #7FFFH,R2                     ; Operand data is 0?
          JNE             FPOW1                         ; --> No
          BTST            7,POWER+3[FB]                 ; Power is minus?
          JEQ             POWZERO                       ; --> No (goes to set result = 0)
;
; Setting calculation error information (non-numeral)
;
POW_ERR:
          MOV.W           #0FFFFH,R0                    ; Sets non-numeral in return value
          MOV.W           CO_OPE+2[FB],R2
          OR.W            #7FFFH,R2
          FSET            C                             ; Sets “result erratic” information
          EXITD
```

```
;----------------------------------------------------------------------
;           Setting result = 0
;----------------------------------------------------------------------
POWZERO:
            MOV.W       #0,R0                   ; Sets result = 0
            MOV.W       #0,R2
            FCLR        C                       ; Sets “result normal” information
            EXITD
;
;**********************************************************************************
;
FPOW1:
            BTST        7,CO_OPE+3[FB]          ; Operand data is minus?
            JEQ         FPOW6                   ; --> No
            MOV.W       POWER[FB],R3            ; Reads power
            MOV.W       POWER+2[FB],R1
            SHL.W       #1,R3                   ; Shifts data up (to adjust type of exponent part)
            ROLC.W              R1
            CMP.B       #7FH,R1H                ; Power is less than 1?
            JLTU        POW_ERR                 ; --> Yes (error)
;
FPOW2:
            CMP.B       #7FH,R1H                ; Conversion of power into integer completed?
            JEQ         FPOW3                   ; --> Yes
            SHL.W       #1,R3                   ; Shifts mantissa part data up
            ROLC.B      R1L
            SUB.B       #1,R1H                  ; Subtracts 1 from exponent part
            JMP         FPOW2
FPOW3:
            AND.W       #00FFH,R1               ; Clears exponent part
            OR.W        R3,R1                   ; No decimal fraction?
            JNE         POW_ERR                 ; --> No (error)
FPOW4:
            BTST        7,CO_OPE+3[FB]          ; Operand data is minus?
            JEQ         FPOW6                   ; --> No
FPOW5:
            XOR.B       #80H,CO_OPE+3[FB]       ; Inverts sign of operand data
;
            MOV.W       POWER[FB],R0            ; Reads power
            MOV.W       POWER+2[FB],R2
            JSR         FIXI                    ; Converts floating data of power into integer
            MOV.B       #1,SIGN[FB]             ; Sets minus in sign information
            BTST        0,R0                    ; LSB of integer is 0?
            JNE         FPOW7                   ; --> No
FPOW6:
            MOV.B       #0,SIGN[FB]             ; Sets plus in sign information
```

```
FPOW7:
        MOV.W       CO_OPE[FB],R0       ; Reads operand data (inverted)
        MOV.W       CO_OPE+2[FB],R2
        JSR         FLN                 ; Natural logarithmic calculation
        JC          POWOVER             ; --> Overflow
;
        MOV.W       POWER[FB],R1        ; Reads power
        MOV.W       POWER+2[FB],R3
        JSR         FMUL                ; Multiplies calculation result by power
        JSR         FOVERCHK            ; Checks for overflow
        JC          POWOVER             ; --> Overflow
;
        JSR         FEXP                ; Exponential function calculation
        JSR         FOVERCHK            ; Checks for overflow
        JC          POWOVER             ; --> Overflow
;
        CMP.B       #0,SIGN[FB]         ; Sign inverted?
        JEQ         FPOW_EXT            ; --> No
        XOR.W       #8000H,R2           ; Inverts sign of calculation result
FPOW_EXT:
        FCLR        C                   ; Sets "result normal" information
        EXITD
;
; Setting overflow information (maximum value)
;
POWOVER:
        AND.W       #8000H,R2           ; Clears all but sign
        OR.W        #7F7FH,R2           ; Sets maximum value in return value
        MOV.W       #0FFFFH,R0
        FSET        C                   ; Sets "result erratic" information
        EXITD
;
        .END
```

```
;*************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 13            *
;                                                                                             *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                                *
;*************************************************************************************************
                .GLB        FEXP
;
                .GLB        FOVERCHK    ; Overflow check
                .GLB        FSUB        ; Floating-point addition
                .GLB        FDIV        ; Floating-point division
                .GLB        FCAL        ; Table data calculation
                .GLB        FLOT        ; Integer data → floating data conversion processing
                .GLB        FIXI        ; Floating data →integer conversion processing
;
VromTOP         .EQU        0F0000H     ; Declares start address of ROM
FBcnst          .EQU        001000H     ; Assumed FB register value
;
F87_H           .EQU        042AEH      ; 87.33654475 upper 2-byte value
F87_L           .EQU        0AC50H      ;                   lower 2-byte value
FP5_H           .EQU        03F00H      ; 0.5 upper 2-byte value
FP5_L           .EQU        00000H      ;       lower 2-byte value
FL2C_H          .EQU        03F31H      ; LN(2) upper 2-byte value
FL2C_L          .EQU        07218H      ;           lower 2-byte value
FOVER_H         .EQU        07F7FH      ; Overflow upper 2-byte value
FOVER_L         .EQU        0FFFFH      ;               lower 2-byte value
;
BUFA            .EQU        –9          ; Used for saving Q data
SIGN            .EQU        –5          ; Sign of calculation result  0: plus; 1: minus
CO_OPE          .EQU        –4          ; Operand data (4 bytes)
;
                .SECTION    PROGRAM,ROMDATA
                .ORG        VromTOP
FEXT:
                .FLOAT      1.0939E-4       ;  0.00010939    (C7)
                .FLOAT      9.4755E-4       ;  0.00094755    (C6)
                .FLOAT      6.80097E-3      ;  0.00680097    (C5)
                .FLOAT      3.9246744E-2    ;  0.039246744   (C4)
                .FLOAT      1.6986580E-1    ;  0.169865796   (C3)
                .FLOAT      4.9012909E-1    ;  0.490129090   (C2)
                .FLOAT      7.0710678E-1    ;  0.707106781   (C1)
;
;================================================================
;           Title: Exponential function (single-precision, floating-point)
;
;           Content of processing:
;                       This program finds an exponential function of operand data (R2R0) and stores the
;                       result in R2, R0.
;                       (R2R0) = e^(R2R0)
;
```

```
;           Procedure:
;                     (1)  Operand data (normalized single-precision, floating- point number)
;                          Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                          R2 and the mantissa (mid, lower) in register R0.
;                     (2)  Call the subroutine.
;                     (3)  The calculation result is placed in R2, R0.
;
;           Result:
;                     Result normal:
;                               The C flag is reset to "0".
;                               The calculation result is stored in R2, R0.
;                        R2 (High)              R2 (Low)               R0H              R0L
;                           ↑                      ↑                     ↑                ↑
;           Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                     Result erratic:
;                               The C flag is set to "1".
;                               The following value is returned in R2, R0.
;
```

Contents of R2, R0	Meaning
Maximum value	Overflow or argument exceeds the range of –87.3 to 87.3 including both ends

```
;
;
;           Input:    ------------------------------> Output:
;
;           R0 (Lower half of operand data)   R0 (Lower half of calculation result)
;           R1 ( )                            R1 (Indeterminate)
;           R2 (Upper half of operand data)   R2 (Upper half of calculation result)
;           R3 ( )                            R3 (Indeterminate)
;           A0 ( )                            A0 (Indeterminate)
;           A1 ( )                            A1 (Indeterminate)
;
;           Stack amount used: 38 bytes
;=============================================================
                    .SECTION     PROGRAM,CODE
                    .FB          FBcnst          ; Assumes FB register value
FEXP:
          ENTER      #10                         ; Allocates internal variables
          MOV.W      R2,CO_OPE+2[FB]             ; Saves operand data in variables
          MOV.W      R0,CO_OPE[FB]
;
; Checking argument = 0
;
          CMP.W      #0,R0                       ; Argument is 0?
          JNE        FEXP1                       ; --> No
          AND.W      #7FFFH,R2                   ; Argument is 0?
          JNE        FEXP1                       ; --> No
;-----------------------------------------------
;         Setting result = 1
;-----------------------------------------------
          MOV.W      #3F80H,R2                   ; Sets 1 in return value
          FCLR       C                           ; Sets "result normal" information
          EXITD
;
```

```
; Checking overflow (exceeding the range of –87.3 to 87.3)
;
FEXP1:
        MOV.W       CO_OPE+2[FB],R2     ; Reads operand data
        BCLR        15,R2               ; Clears sign of operand data
;
        CMP.W       #F87_H,R2           ; Less than –87.3 or greater than 87.3 including both ends?
        JGTU        EXPOVER             ; --> Yes (overflow)
        JLTU        FEXP2               ; --> No
        CMP.W       #F87_L,R0           ; Less than –87.3 or greater than 87.3 including both ends?
        JGEU        EXPOVER             ; --> Yes (overflow)
;
; Calculation processing
;
FEXP2:
        MOV.W       CO_OPE+2[FB],R2     ; Reads operand data
        MOV.W       #FL2C_H,R3          ; Sets LN(2) data
        MOV.W       #FL2C_L,R1
;
        JSR         FDIV                ; Divides operand by LN(2)
        JSR         FOVERCHK            ; Checks for overflow
        JC          EXPOVER             ; --> Overflow
        MOV.W       R0,CO_OPE[FB]       ; Saves calculation result
        MOV.W       R2,CO_OPE+2[FB]
;
        JSR         FIXI                ; Converts data into integer (Q data)
;
        BTST        15,R2               ; Checks sign
        JEQ         FEXP3               ; --> Plus
;
        XOR.W       #0FFFFH,R0          ; Takes 2's complement
        ADD.W       #1,R0
        XOR.W       #7FFFH,R2
        ADCF.W      R2
FEXP3:
        MOV.W       R0,BUFA[FB]         ; Saves Q data
        MOV.W       R2,BUFA+2[FB]
;
        JSR         FLOT                ; Converts Q data into floating data
;
        MOV.W       R2,R3               ; Modifies Q data register
        MOV.W       R0,R1
;
        MOV.W       CO_OPE[FB],R0       ; Reads (operand divided by LN(2))
        MOV.W       CO_OPE+2[FB],R2
;
        JSR         FSUB                ; Divides operand by LN(2) and subtracts Q
        JSR         FOVERCHK            ; Checks for overflow
        JC          EXPOVER             ; --> Overflow
;
        MOV.W       #FP5_H,R3           ; Sets 0.5
        MOV.W       #FP5_L,R1
        JSR         FSUB                ; Subtracts 0.5 from calculation result
        JSR         FOVERCHK            ; Checks for overflow
        JC          EXPOVER             ; --> Overflow
;
```

```
        MOV.W       #FEXT&0FFFFH,A0     ; Sets data table address
        MOV.W       #FEXT>>16,A1
        MOV.B       #7-1,R1L            ; Sets number of tables
        JSR         FCAL                ; Calculates table data
        JC          EXPOVER             ; --> Overflow
;
        MOV.W       R2,R1               ; Modifies calculation result register (exponent part)
;
        MOV.B       #0,SIGN[FB]         ; Initializes sign information
        SHL.W       #1,R1               ; Sign →C flag
        ROLC.B      SIGN[FB]            ; Sets sign information
;
        FSET        C                   ; Sets C flag = 1
        ADC.B       BUFA[FB],R1H        ; Adds exponent + Q + 1
;
        SHL.B       #-1,SIGN[FB]        ; Sign →C flag
        RORC.W      R1                  ; Sets sign
        MOV.W       R1,R2               ; Restores register
        FCLR        C                   ; Sets “result normal” information
        EXITD
;
; Setting overflow information (maximum value)
;
EXPOVER:
        MOV.W       #0FFFFH,R0          ; Sets maximum value in mantissa (mid, lower) part
        MOV.W       #0FEFEH,R2          ; Sets maximum value in mantissa (upper) part and
                                        ; LSB of exponent part
        SHL.B       #-1,SIGN[FB]        ; Checks sign
        RORC.W      R2                  ; Sets maximum value in exponent part and sign
        FSET        C                   ; Sets “result erratic” information
        EXITD
;
        .END
```

```
;*************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 14         *
;                                                                                        *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                            *
;*************************************************************************************************
                    .GLB            FLN
;
                    .GLB            FLN_CAL                 ; Natural logarithmic calculation
;
VromTOP             .EQU            0F0000H                 ; Declares start address of ROM
;
;=================================================================
;          Title: Natural logarithmic calculation (single-precision, floating-point)
;          Content of processing:
;                    This program finds a natural logarithmic of operand data (R2R0) and stores the result in R2, R0.
;                    (R2R0) = LN (R2R0)
;          Procedure:
;                    (1)  Operand data (normalized single-precision, floating- point number)
;                         Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                         R2 and the mantissa (mid, lower) in register R0.
;                    (2)  Call the subroutine.
;                    (3)  The calculation result is placed in R2, R0.
;          Result:
;                    Result normal:
;                              The C flag is reset to "0".
;                              The calculation result is stored in R2, R0.
;                      R2 (High)                 R2 (Low)                 R0H              R0L
;                          ↑                         ↑                      ↑                ↑
;          Sign, Exponent b7 to b1     Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;                    Result erratic:
;                              The C flag is set to "1".
;                              The following value is returned in R2, R0.
;                    ------------------------------------------------------
;                    Contents of R2, R0                 Meaning
;                    ------------------------------------------------------
;                    Non-numeral                        Calculation error
;                    ------------------------------------------------------
;                    No change                          Overflow
;                    ------------------------------------------------------
;
;          Input:    --------------------------------> Output:
;
;          R0 (Lower half of operand data)    R0 (Lower half of calculation result)
;          R1 ( )                             R1 (Indeterminate)
;          R2 (Upper half of operand data)    R2 (Upper half of calculation result)
;          R3 ( )                             R3 (Indeterminate)
;          A0 ( )                             A0 (Indeterminate)
;          A1 ( )                             A1 (Indeterminate)
;
;          Stack amount used: 41 bytes
;=================================================================
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
FLN:
          FSET                Z                             ; Sets LN information
          JSR                 FLN_CAL                       ; Natural logarithmic calculation
          RTS
;
          .END
```

```
;*************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 15          *
;                                                                                         *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                            *
;*************************************************************************************************
                .GLB        FLOG
;
                .GLB        FOVERCHK      ; Overflow check
                .GLB        FCAL          ; Table data calculation
                .GLB        FADD          ; Floating-point addition
                .GLB        FSUB          ; Floating-point subtraction
                .GLB        FMUL          ; Floating-point multiplication
                .GLB        FDIV          ; Floating-point division
                .GLB        FLOT          ; Integer data → floating data conversion processing
;
VromTOP         .EQU        0F0000H       ; Declares start address of ROM
FBcnst          .EQU        001000H       ; Assumed FB register value
;
FNO1_H          .EQU        3F80H         ; Numeral 1 upper 2-byte value
FNO1_L          .EQU        0000H         ;           lower 2-byte value
FL2C_H          .EQU        3F31H         ; LN(2) upper 2-byte value
FL2C_L          .EQU        7218H         ;       lower 2-byte value
FL10_H          .EQU        4013H         ; LN(10) upper 2-byte value
FL10_L          .EQU        5D8EH         ;        lower 2-byte value
;
EXP             .EQU        -10           ; Used for saving exponent part
MODE            .EQU        -9            ; 0: FLOG; 1: FLN
BUFA            .EQU        -8            ; General-purpose buffer
CO_OPE          .EQU        -4            ; Operand data (4 bytes)
;
                .SECTION    PROGRAM,ROMDATA
                .ORG        VromTOP
FLGT:
                .FLOAT      1.0757369E–2   ;  0.010757369   (C7)
                .FLOAT      –5.5119959E–2  ; –0.055119959   (C6)
                .FLOAT      1.3463927E–1   ;  0.134639267   (C5)
                .FLOAT      –2.2587328E–1  ; –0.225873284   (C4)
                .FLOAT      3.2823312E–1   ;  0.328233122   (C3)
                .FLOAT      –4.9947015E–1  ; –0.499470150   (C2)
                .FLOAT      9.9998103E–1   ;  0.999981028   (C1)
;
;===========================================================================
;           Title: Common logarithmic calculation (single-precision, floating-point)
;
;           Content of processing:
;                   This program finds a common logarithmic of operand data (R2R0) and stores the result in R2, R0.
;                   (R2R0) = LOG (R2R0)
;
;           Procedure:
;                   (1)  Operand data (normalized single-precision, floating- point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R2 and the mantissa (mid, lower) in register R0.
;                   (2)  Call the subroutine.
;                   (3)  The calculation result is placed in R2, R0.
;
```

```
;           Result:
;                       Result normal:
;                                   The C flag is reset to "0".
;                                   The calculation result is stored in R2, R0.
;                          R2 (High)                R2 (Low)                   R0H              R0L
;                             ↑                        ↑                        ↑                ↑
;           Sign, Exponent b7 to b1    Exponent b0, Mantissa (upper)    Mantissa (mid)    Mantissa (lower)
;
;                       Result erratic:
;                                   The C flag is set to "1".
;                                   The following value is returned in R2, R0.
;
;                       Contents of R2, R0          Meaning
;
;                       Non-numeral                 Calculation error
;
;                       No change                   Overflow
;
;
;           Input:      ------------------------------->  Output:
;
;           R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;           R1 ( )                              R1 (Indeterminate)
;           R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;           R3 ( )                              R3 (Indeterminate)
;           A0 ( )                              A0 (Indeterminate)
;           A1 ( )                              A1 (Indeterminate)
;
;           Stack amount used: 33 bytes
;=================================================================
                    .SECTION        PROGRAM,CODE
                    .FB             FBcnst              ; Assumes FB register value
FLOG:
            FCLR            C                           ; Sets LOG information
FLN_CAL:
            ENTER           #10                         ; Allocates internal variables
            STZX            #0,#1,MODE[FB]              ; Sets LOG/LN mode
            MOV.W           R2,CO_OPE+2[FB]             ; Saves operand data
            MOV.W           R0,CO_OPE[FB]
            BTSTC           15,R2                       ; Clears sign and checks sign
            JNE             LOG_ERR2                    ; --> Operand data minus (error)
            OR.W            R0,R2                       ; Absolute 0?
            JNE             FLOG2                       ; --> No
LOG_ERR2:
            JMP             LOG_ERR                     ; Sets non-numeral
;
FLOG2:
            MOV.W           CO_OPE+2[FB],R2             ; Reads exponent and mantissa (upper) parts
            BCLR            15,R2                       ; Clears sign
            CMP.W           #FNO1_H,R2                  ; Logic 1?
            JNE             FLOG3                       ; --> No
            MOV.W           CO_OPE[FB],R0               ; Logic 1?
            JNE             FLOG3                       ; --> No
            JMP             LOG_ZERO                    ; --> Yes (returns absolute zero)
;
```

```
FLOG3:
        MOV.W    R2,R1              ; Exponent part →R1H
        SHL.W    #1,R1
        CMP.B    #1,R1H             ; Exponent part is 1?
        JNE      FLOG31             ; --> No
        JMP      LOG_NON            ; --> Yes (conversion unnecessary)
;
FLOG31:
        MOV.B    R1H,EXP[FB]        ; Saves exponent part
        MOV.W    CO_OPE+2[FB],R2    ; Reads exponent and mantissa (upper) parts
        AND.W    #807FH,R2          ; Clears exponent part.
        OR.W     #3F80H,R2          ; Sets 7F in exponent part
;
        MOV.W    #FNO1_H,R3         ; Sets numeral 1
        MOV.W    #FNO1_L,R1
        JSR      FSUB               ; Subtracts 1 from operand
        JSR      FOVERCHK           ; Checks for overflow
        JC       LOGOVER            ; --> Overflow
        MOV.W    R2,BUFA+2[FB]      ; Saves calculation result
        MOV.W    R0,BUFA[FB]
;
        MOV.W    #FLGT&0FFFFH,A0    ; Sets data table address
        MOV.W    #FLGT>>16,A1
        MOV.B    #7-1,R1L           ; Sets number of tables
        JSR      FCAL               ; Calculates table data
        JC       LOGOVER            ; --> Overflow
;
        MOV.W    BUFA+2[FB],R3      ; Restores calculation result
        MOV.W    BUFA[FB],R1
        JSR      FMUL               ; Multiplies table calculation result by restored result
        JSR      FOVERCHK           ; Checks for overflow
        JC       LOGOVER            ; --> Overflow
        MOV.W    R2,BUFA+2[FB]      ; Saves table calculation result
        MOV.W    R0,BUFA[FB]
;
        MOV.B    EXP[FB],R0L        ; Restores exponent part
        SUB.B    #7FH,R0L           ; Subtracts 7F from exponent part
        JNC      FLOG4              ; --> Decimal
;
        MOV.W    #0,R2              ; Sets integer Q
        MOV.B    #0,R0H
        JMP      FLOG5
FLOG4:
        MOV.W    #0FFFFH,R2         ; Sets decimal Q
        MOV.B    #0FFH,R0H
```

```
FLOG5:
        JSR         FLOT                 ; Converts integer data into floating data
;
        MOV.W       #FL2C_H,R3           ; Sets LN(2)
        MOV.W       #FL2C_L,R1
        JSR         FMUL                 ; Multiplies LN(2) by floating data
        JSR         FOVERCHK             ; Checks for overflow
        JC          LOGOVER              ; --> Overflow
;
        MOV.W       BUFA+2[FB],R3        ; Restores table calculation result
        MOV.W       BUFA[FB],R1
        JSR         FADD                 ; Adds calculation result and table calculation result
        JSR         FOVERCHK             ; Checks for overflow
        JC          LOGOVER              ; --> Overflow
;
        MOV.B       MODE[FB],R1L         ; LOG mode?
        JEQ         FLOG_EXT             ; --> No (LN mode)
;
        MOV.W       #FL10_H,R3           ; Sets LN(10)
        MOV.W       #FL10_L,R1
        JSR         FDIV                 ; Divides calculation result by LN(10)
        JSR         FOVERCHK             ; Checks for overflow
        JC          LOGOVER              ; --> Overflow
FLOG_EXT:
        FCLR        C                    ; Sets “result normal” information
        EXITD
;
;---------------------------------------------------------------------
;       Setting calculation error (non-numeral) or overflow (no change)
;---------------------------------------------------------------------
LOG_ERR:
        MOV.W       CO_OPE+2[FB],R2      ; Reads sign
        OR.W        #7FFFH,R2            ; Sets non-numeral in return value
        MOV.W       #0FFFFH,R0
LOGOVER:
        FSET        C                    ; Sets “result erratic” information
        EXITD
;
;---------------------------------------------------------------------
;       Setting absolute 0 (normal)
;---------------------------------------------------------------------
LOG_ZERO:
        MOV.W       #0,R0                ; Sets absolute 0
        MOV.W       #0,R2
;---------------------------------------------------------------------
;       Conversion unnecessary (normal)
;---------------------------------------------------------------------
LOG_NON:
        FCLR        C                    ; Sets “result normal” information
        EXITD
;
        .END
```

```
;*************************************************************************************************
;           M16C Program Collection of Mathematic/Trigonometric Functions No. 16            *
;                                                                                            *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                               *
;*************************************************************************************************
                    .GLB            FCMP
;
                    .GLB            FSUB            ; Floating-point subtraction
;
VromTOP             .EQU            0F0000H         ; Declares start address of ROM
FBcnst              .EQU            001000H         ; Assumed FB register value
;
OPE                 .EQU            –8              ; Comparison data (4 bytes)
CO_OPE              .EQU            –4              ; Operand data (4 bytes)
;
;==========================================================================
;           Title: Data comparison (single-precision, floating-point)
;
;           Content of processing:
;                   This program compares the contents of (R2R0) and (R3R1) and sets the result in FLG bits.
;                   FLG = (R2R0) : (R3R1)
;
;           Procedure:
;                   (1)  Operand data (normalized single-precision, floating- point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R2 and the mantissa (mid, lower) in register R0.
;                   (2)  Comparison data (normalized single-precision, floating-point number)
;                        Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                        R3 and the mantissa (mid, lower) in register R1.
;                   (3)  Call the subroutine.
;                   (4)  The result is placed in FLG bits.
;
;           Result:
;                   ------------------------------------------
;                   C    Z              Meaning
;                   ------------------------------------------
;                   1    0          (R2 R0) > (R3 R1)
;                   ------------------------------------------
;                   1    1          (R2 R0) = (R3 R1)
;                   ------------------------------------------
;                   0    0          (R2 R0) < (R3 R1)
;                   ------------------------------------------
;
;           Input:  ------------------------------------------> Output:
;
;           R0 (Lower half of operand data)             R0 (Does not change)
;           R1 (Lower half of comparison data)          R1 (Does not change)
;           R2 (Upper half of operand data)             R2 (Does not change)
;           R3 (Upper half of comparison data)          R3 (Does not change)
;           A0 ( )                                      A0 (Unused)
;           A1 ( )                                      A1 (Unused)
;
;           Stack amount used: 32 bytes
;==========================================================================
```

```
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
                    .FB             FBcnst              ; Assumes FB register value
FCMP:
          ENTER           #8                          ; Allocates internal variables
          MOV.W           R2,CO_OPE+2[FB]             ; Saves (R2 R0)
          MOV.W           R0,CO_OPE[FB]
          MOV.W           R3,OPE+2[FB]                ; Saves (R3 R1)
          MOV.W           R1,OPE[FB]
;
          JSR             FSUB                        ; (R2,R0) = (R2,R0) – (R3,R1)
;
; Checking absolute 0
;
          MOV.W           R2,R3                       ; Moves result to R3
          SHL.W           #1,R3                       ; Clears sign
          CMP.W           R0,R3                       ; Absolute 0?
          JEQ             FCMP_END                    ; --> Yes (C = 1, Z = 1)
;
          BNTST           15,R2                       ; Sets result in C flag
          FCLR            Z                           ; Clears Z flag
FCMP_END:
          PUSHC           FLG                         ; Saves FLG
          MOV.W           CO_OPE[FB],R0               ; Restores register
          MOV.W           CO_OPE+2[FB],R2
          MOV.W           OPE[FB],R1
          MOV.W           OPE+2[FB],R3
          POPC            FLG                         ; Restores FLG
          EXITD
;
          .END
```

```
;*************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 17          *
;                                                                                         *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                            *
;*************************************************************************************************
                    .GLB            FTOI
;
VromTOP             .EQU            0F0000H         ; Declares start address of ROM
FBcnst              .EQU            001000H         ; Assumed FB register value
;
SIGN                .EQU            –1              ; 0: plus; 1: minus
;
;==========================================================================
;          Title: Conversion from FLOAT type to WORD type
;          Content of processing:
;                    This program converts the content of FLOAT data (R2R0) into WORD (16-bit) type and
;                    stores the result in R0.
;          Procedure:
;                    (1)  FLOAT data (normalized single-precision, floating-point number)
;                         Store the sign, exponent b7 to b1, exponent b0, and mantissas (upper) in register
;                         R2 and the mantissa (mid, lower) in register R0.
;                    (2)  Call the subroutine.
;                    (3)  The result is placed in R0.
;          Result:
;                    The result is placed in R0.
;                    If an overflow occurs, R0 is 7FFFH when positive or 8000H when negative.
;                    If an underflow occurs, R0 is cleared to 0000H.
;                    The following shows the contents of flags.
;
;                    C   Z   S                 Meaning
;
;                    1   0   0          Positive overflow (R0 = 7FFFH)
;
;                    1   0   1          Negative overflow (R0 = 8000H)
;
;                    1   1   0          Underflow (R0 = 0000H)
;
;                    0   1   0          Result is 0
;
;                    0   0   0          Result is positive
;
;                    0   0   1          Result is negative
;
;
;          Input:    -------------------------------------------> Output:
;
;          R0 (Lower half of FLOAT type data)          R0 (WORD type data)
;          R1 ( )                                      R1 (Indeterminate)
;          R2 (Upper half of FLOAT type data)          R2 (Does not change)
;          R3 ( )                                      R3 (Unused)
;          A0 ( )                                      A0 (Unused)
;          A1 ( )                                      A1 (Unused)
;
;          Stack amount used: 1 byte
;==========================================================================
```

```
                .SECTION        PROGRAM,CODE
                .ORG            VromTOP
                .FB             FBcnst          ; Assumes FB register value
FTOI:
        ENTER           #1                      ; Allocates internal variables
;
        XCHG.W          R0,R2                   ; Changes registers
        MOV.W           #0,R1                   ; Initializes WORD type
;
        BTSTC           15,R0                   ; Checks sign (sign cleared)
        STZX            #0,#1,SIGN[FB]          ; Sets sign of calculation result
;
        CMP.W           #0,R0                   ; Input 0?
        JNE             FTOI_10                 ; --> No
        CMP.W           #0,R2                   ; Input 0?
        JNE             FTOI_10                 ; --> No
        FCLR            C                       ; Sets "without flow" information
        JMP             I0SET
I0UNDER:
        FSET            C                       ; Sets "with flow" information
;-------------------------------------------------------------------
;       Setting integer 0
;-------------------------------------------------------------------
I0SET:
        MOV.W           #0,R0                   ; Sets integer 0 in return value
        EXITD
;
FTOI_10:
        BTSTS           7,R0                    ; Sets LSB of exponent part in C flag
                                                ; Adds 1.0 to mantissa part
        ROLC.B          R0H                     ; Creates exponent
        SUB.B           #7FH,R0H                ; Less than 1?
        JNC             I0UNDER                 ; --> Yes (sets 0)
        CMP.B           #15,R0H                 ; Within representation range?
        JLTU            FTOI_20                 ; --> Yes
        BSET            15,R1                   ; Sets maximum value of the same sign
        JNE             FTOI_15                 ; --> Out of representation range
        CMP.B           #0,SIGN[FB]             ; Sign plus?
        JEQ             FTOI_PLS                ; --> Yes (out of range)
;-------------------------------------------------------------------
;       Checking maximum negative value
;-------------------------------------------------------------------
        CMP.W           #0,R2
        JNE             FTOI_MIS                ; --> Out of representation range
        CMP.B           #80H,R0L
        JNE             FTOI_MIS                ; --> Out of representation range
        FCLR            C                       ; Sets "without flow"
        JMP             FTOI_MIMAX              ; --> Sets maximum negative value
;
FTOI_15:
        CMP.B           #0,SIGN[FB]             ; Sign plus?
        JNE             FTOI_MIS                ; --> Negative number (8000H)
;
```

```
; Positive overflow
;
FTOI_PLS:
        NOT.W       R1                      ; Positive number (7FFF)
;
; Negative overflow
;
FTOI_MIS:
        FSET        C                       ; Sets “with flow”
FTOI_MIMAX:
        MOV.W       R1,R0                   ; Sets return value
        EXITD
;
;---------------------------------------------------------------------
;       FLOAT →integer conversion
;---------------------------------------------------------------------
FTOI_20:
        INC.B       R0H                     ; Adjusts loop count
FTOI_LOOP:
        SHL.W       #1,R2                   ; Shifts mantissa data up
        ROLC.B      R0L
        ROLC.W      R1                      ; Gets result
        ADJNZ.B     #-1,R0H,FTOI_LOOP       ; Loop finished? --> No
;
        CMP.B       #0,SIGN[FB]             : Sign plus?
        JEQ         FTOI_30                 ; --> Yes
        NEG.W       R1                      ; Turns data into 2’s complement
FTOI_30:
        MOV.W       R1,R0                   ; Sets return value
        FCLR        C                       ; Sets “without flow”
        EXITD
;
        .END
```

```
;*************************************************************************************************
;          M16C Program Collection of Mathematic/Trigonometric Functions No. 18          *
;                                                                                        *
;          Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                            *
;*************************************************************************************************
                    .GLB            ITOF
;
                    .GLB            FNOR            ; Normalization processing
;
VromTOP             .EQU            0F0000H         ; Declares start address of ROM
;
;===========================================================================
;          Title: Conversion from WORD type to FLOAT type
;
;          Content of processing:
;                    This program converts the content of WORD (16-bit) type (R0) into FLOAT data and
;                    stores the result in R2R0.
;
;          Procedure:
;                    (1)  Store WORD (16-bit) type data in register R0.
;                    (2)  Call the subroutine.
;                    (3)  The result is placed in R2, R0.
;
;          Result:   The result is placed in R2, R0.
;
;                         R2 (High)               R2 (Low)                  R0H              R0L
;                            ↑                       ↑                       ↑                ↑
;          Sign, Exponent b7 to b1   Exponent b0, Mantissa (upper)   Mantissa (mid)   Mantissa (lower)
;
;                    The following shows the contents of flags.
;                    ------------------------------------------
;                     Z   S          Meaning
;                    ------------------------------------------
;                     1   0     When result is 0
;                    ------------------------------------------
;                     0   0     When result is positive
;                    ------------------------------------------
;                     0   1     When result is negative
;                    ------------------------------------------
;
;          Input:    ---------------------> Output:
;
;          R0 (WORD type data)       R0 (Lower half of FLOAT type data)
;          R1 ( )                    R1 (Indeterminate)
;          R2 ( )                    R2 (Upper half of FLOAT type data)
;          R3 ( )                    R3 (Indeterminate)
;          A0 ( )                    A0 (Unused)
;          A1 ( )                    A1 (Unused)
;
;          Stack amount used: 4 bytes
;===========================================================================
```

```
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
ITOF:
          ENTER           #1                    ; Allocates internal variables
          MOV.W           R0,R1                 ; Integer data →R1
;
          MOV.W           #0,R0                 ; Sets 0 in floating-point data
          MOV.W           #0,R2
;
          CMP.W           #0,R1                 ; Integer data is 0?
          JNE             ITOF10                ; --> No
          EXITD
;
ITOF10:
          BTST            15,R1                 ; Sign is minus?
          JEQ             ITOF20                ; --> No (plus)
;
          BSET            15,R2                 ; Changes floating-point sign negative
          CMP.W           #8000H,R1             ; Maximum value?
          JEQ             ITOF_MAX              ; --> Yes
ITOF11:
          NEG.W           R1                    ; Takes 2's complement
ITOF20:
          MOV.B           R1L,R0H               ; Lower half of integer →Mid part of floating-point
                                                ; number
          SHL.W           #-8,R1                ; Upper half of integer →Upper part of floating-
                                                ; point number
          OR.W            #4700H,R1             ; Sets 8E in exponent part
          OR.W            R1,R2                 ; Sets sign
;
          PUSHC           FLG                   ; Saves flags
          JSR             FNOR                  ; Normalization processing
          POPC            FLG                   ; Restores flags
          EXITD
;
ITOF_MAX:
          MOV.W           #0C700H,R2            ; Sets maximum value
          EXITD
;
          .END
```

```
;***********************************************************************************************
;           M16C Floating-point Library Subroutine                                      *
;                                                                                       *
;           Copyright (C) 1995 MITSUBISHI ELECTRIC CORPORATION                          *
;***********************************************************************************************
                    .GLB            CHKDATA
                    .GLB            FOVERCHK
                    .GLB            FCAL
                    .GLB            FLOT
                    .GLB            FIXI
                    .GLB            FNOR
;
                    .GLB            FADD            ; Floating-point addition
                    .GLB            FMUL            ; Floating-point multiplication
;
VromTOP             .EQU            0F0000H         ; Declares start address of ROM
FBcnst              .EQU            001000H         ; Assumed FB register value
;
                    .SECTION        PROGRAM,CODE
                    .ORG            VromTOP
;
;///////////////////////////////////////////////////////////////////////////////////////////////
;           Non-numeral and Infinity Check Subroutine
;
;           Function:
;                   If the data input with (R2R0) is non-numeral or infinite, this subroutine sets
;                   non-numeral and infinite data in R2 and R0 before returning to the previous
;                   program location (e.g., a location from which FADD was called).
;                   If the data is other than the above, it returns to the location from which it was called.
;
;           Input:  --------------------------------> Output:
;
;           R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;           R1 ( )                              R1 (Indeterminate)
;           R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;           R3 ( )                              R3 (Indeterminate)
;           A0 ( )                              A0 (Unused)
;           A1 ( )                              A1 (Unused)
;
;           Stack amount used: None
;///////////////////////////////////////////////////////////////////////////////////////////////
CHKDATA:
            MOV.W           R2,R3                   ; Saves input data
            MOV.W           R0,R1
            XCHG.W          R2,R0                   ; Changes registers
;
; Checking operand data
;
            SHL.W           #1,R0                   ; Places exponent part of operand data in R0H
            CMP.B           #0FFH,R0H               ; Exponent part is non-numeral or infinite data?
            JEQ             CHKDATA10               ; --> Yes
            MOV.W           R3,R2                   ; Sets data that was input
            MOV.W           R1,R0
            RTS
;
```

```
CHKDATA10:
        CMP.B       #0,R0L          ; Mantissa (upper) part is infinite?
        JNE         CKDTNON         ; --> Non-numeral
        CMP.W       #0,R2           ; Mantissa (mid, lower) part is infinite?
        JEQ         CKDTINF         ; --> Infinite
;
; Setting non-numeral value
;
CKDTNON:
        MOV.W       #0FFFFH,R0      ; Sets non-numeral in mantissa (mid, lower) part
        MOV.W       R1,R2           ; Reads operand data sign, exponent, and mantissa (upper)
        OR.W        #7FFFH,R2       ; Sets non-numeral in exponent and mantissa
                                    ; (upper) parts (with sign unchanged)
BASERET:
        STC         SP,R3           ; Reads stack
        ADD.W       #4,R3           ; Stack + 4 (for two returns)
        LDC         R3,SP           ; Sets stack back again
        EXITD
;
; Setting infinite value
;
CKDTINF:
        MOV.W       #0000H,R0       ; Sets infinity in mantissa (mid, lower) part
        MOV.W       R1,R2           ; Reads operand data sign, exponent, and mantissa (upper)
        AND.W       #0FF80H,R2      ; Sets infinity in mantissa (upper) part
        OR.W        #07F80H,R2      ; Sets infinity in exponent part (with sign unchanged)
        JMP         BASERET         ; Returns to location from which FADD was called
;
;//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;       Data Over/Underflow Check Subroutine
;
;       Function:
;               This subroutine checks to see if the data input in (R2R0) is in overflow or underflow or else.
;               If the data is in overflow or underflow,
;                       the C flag is set to 1.
;               Otherwise,
;                       the C flag is reset to 0.
;
;       Input:  -------------------------------> Output:
;
;       R0 (Lower half of operand data)     R0 (Does not change)
;       R1 ( )                              R1 (Unused)
;       R2 (Upper half of operand data)     R2 (Does not change)
;       R3 ( )                              R3 (Unused)
;       A0 ( )                              A0 (Unused)
;       A1 ( )                              A1 (Unused)
;
;       Stack amount used: None
;//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
FOVER_H     .EQU        07F7FH      ; Overflow upper 2-byte value
FOVER_L     .EQU        0FFFFH      ;          lower 2-byte value
FUNDER_H    .EQU        0080H       ; Underflow upper 2-byte value
FUNDER_L    .EQU        0000H       ;           lower 2-byte value
;
FOVERCHK:
;
```

```
; Overflow check
;
          CMP.W       #FOVER_H,R2          ; Overflow value?
          JNE         FUNDERCHK            ; --> No
          CMP.W       #FOVER_L,R0          ; Overflow value?
          JNE         FOVER_0              ; --> No (without flow)
FOVER_1:
          FSET        C                    ; With flow (C flag is set)
          RTS
;
;
; Underflow check
;
FUNDERCHK:
          CMP.W       #FUNDER_H,R2         ; Underflow value?
          JNE         FOVER_0              ; --> No (without flow)
          CMP.W       #FUNDER_L,R0         ; Underflow value?
          JEQ         FOVER_1              ; --> Yes (with flow)
FOVER_0:
          FCLR        C                    ; Without flow (C flag is cleared)
          RTS
;
;//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;         Table Data Calculation Subroutine
;
;         Function:
;                   This subroutine calculates the data input in (R2R0) by the double-word table data at
;                   address indicated by A1A0 as many time as the count of R1L.  The calculation result
;                   is placed in R2, R0 and the C flag is reset to 0.  However, if an overflow occurs, the
;                   C flag is set to 1.
;
;         Input:    ------------------------------->  Output:
;
;         R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;         R1 (R1L = count)                    R1 (Indeterminate)
;         R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;         R3 ( )                              R3 (Indeterminate)
;         A0 (Lower half of table address)    A0 (Indeterminate)
;         A1 (Upper half of table address)    A1 (Indeterminate)
;
;         Stack amount used: 24 bytes
;//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
CO_OPE              .EQU    –5             ; Area for storing floating data when input
COUNT               .EQU    –1             ; Counter (used in internal processing)
                    .FB          FBcnst    ; Assumes FB register value
;
FCAL:
          ENTER       #5                   ; Allocates internal variables
          MOV.B       R1L,COUNT[FB]        ; Calculation count →internal variable
          MOV.W       R2,CO_OPE+2[FB]      ; Saves input data
          MOV.W       R0,CO_OPE[FB]
;
          LDE.W       [A1A0],R1            ; Sets lower half of calculation data
          ADD.W       #2,A0                ; Calculates high-order address
          ADCF.W      A1
          LDE.W       [A1A0],R3            ; Sets upper half of calculation data
```

```
FCAL_LOOP:
        JSR         FMUL              ; Multiplies calculation result by data
        JSR         FOVERCHK          ; Checks for overflow
        JC          FCALOVER          ; --> Overflow
;
        ADD.W       #2,A0             ; Sets next table pointer
        ADCF.W          A1
        LDE.W       [A1A0],R1         ; Sets lower half of calculation data
        ADD.W       #2,A0             ; Calculates high-order address
        ADCF.W          A1
        LDE.W       [A1A0],R3         ; Sets upper half of calculation data
;
        JSR         FADD              ; Adds calculation result and table data
        JSR         FOVERCHK          ; Checks for overflow
        JC          FCALOVER          ; --> Overflow
;
        MOV.W       CO_OPE[FB],R1
        MOV.W       CO_OPE+2[FB],R3
;
        DEC.B       COUNT[FB]         ; Decrements counter by 1
        JNE         FCAL_LOOP         ; --> Continues calculation
;
; Calculation terminated normally
;
        FCLR        C                 ; Without flow (C flag is cleared)
        EXITD
;
 Overflow occurred
;
FCALOVER:
        FSET        C                 ; With flow (C flag is set)
        EXITD
;
;////////////////////////////////////////////////////////////////////////////////////////////
;          Integer Data →Floating Data Conversion Processing
;
;       Function:
;               This program converts the integer data input in (R2R0) into floating-point numbers
;               and returns the converted data placed in R2, R0.
;       Input:  -------------------------------->  Output:
;
;       R0 (Lower half of operand data)    R0 (Lower half of calculation result)
;       R1 ( )                             R1 (Indeterminate)
;       R2 (Upper half of operand data)    R2 (Upper half of calculation result)
;       R3 ( )                             R3 (Unused)
;       A0 ( )                             A0 (Unused)
;       A1 ( )                             A1 (Unused)
;
;       Stack amount used: None
;////////////////////////////////////////////////////////////////////////////////////////////
FLOT:
        MOV.W       R2,R1             ; Changes exponent and mantissa (upper) parts to R1
;
        BTST        15,R1
        JNE         FLOT_MI           ; --> Sign minus
;
```

```
        CMP.W       #0,R0              ; Absolute 0?
        JNE         FLOT1              ; --> No
        CMP.W       #0,R1              ; Absolute 0?
        JNE         FLOT1              ; --> No
;
; Setting absolute 0
;
        MOV.W       #0,R0
        MOV.W       #0,R2
        RTS
;
; Setting 96H in exponent part
;
FLOT1:
        ROLC.W      R1                 ; Exponent part → R1H, sign → C flag
        MOV.B       #96H,R1H           ; Sets 96H in exponent part
        RORC.W      R1                 ; C flag → sign, exponent part position adjusted
        MOV.W       R1,R2              ; Returns exponent and mantissa (upper) parts to R2
        JSR         FNOR               ; Normalization
        RTS
;
FLOT_MI:
        XOR.W       #0FFFFH,R0         ; Inverts data
        ADD.W       #1,R0              ; Takes 2's complement
        XOR.W       #0FFFFH,R1         ; Inverts data
        ADCF.W      R1                 ; Takes 2's complement
        BSET        15,R1              ; Sets negative sign
        JMP         FLOT1
;
;///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;       Floating Data → Integer Conversion Processing
;
;       Function:
;               This program converts the floating data input in (R2R0) into integral numbers and
;               returns the converted data placed in R2, R0.
;
;       Input:  -------------------------------->  Output:
;
;       R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;       R1 ( )                              R1 (Indeterminate)
;       R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;       R3 ( )                              R3 (Indeterminate)
;       A0 ( )                              A0 ( )
;       A1 ( )                              A1 ( )
;
;       Stack amount used: 1 byte
;///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
FIXI:
        MOV.W       R0,R1              ; Changes mantissa (mid, lower) part to R1
        MOV.W       R2,R0              ; Changes exponent and mantissa (upper) parts to R0
;
        SHL.W       #1,R1
        ROLC.W      R0                 ; Adjusts exponent part to high-order bit
        PUSHC       FLG                ; Saves sign (sign = C flag)
;
        CMP.B       #7FH,R0H           ; Data is less than 1?
        JGEU        FIXI10             ; --> No
```

```
; Integer 0 returned when less than 1 & exponent 97H or greater
;
FIXI00:
        POPC        FLG             ; Adjusts stacks
        MOV.W       #0,R0           ; Sets integer 0
        MOV.W       #0,R2
        RTS
;
FIXI10:
        FSET        C               ; Economized form bit
        RORC.B      R0L             ; Shifts mantissa part down
        RORC.W      R1
        ADD.B       #69H,R0H        ; Exponent + 69H
        JGEU        FIXI00          ; --> Exponent 97H or greater (data over)
FIXI20:
        ADD.B       #1,R0H          ; Exponent + 1
        JGEU        FIXI30          ; --> Conversion into integer finished (exponent part 0)
        SHL.B       #-1,R0L         ; Shifts mantissa part down
                                    ; (0s inserted in high-order bits)
        RORC.W      R1
        JMP         FIXI20
;
FIXI30:
        POPC        FLG             ; Restores sign (sign = C flag)
        JNC         FIXI40          ; --> Sign plus
        BSET        15,R0           ; Sets negative sign
FIXI40:
        MOV.W       R0,R2           ; Sets integer-converted data
        MOV.W       R1,R0
        RTS
;
;//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;       Normalization Processing
;
;       Function:
;               This program normalizes the floating-point data input in (R2R0) and returns the result
;               placed in R2, R0.
;       Input:  -------------------------------->  Output:
;
;       R0 (Lower half of operand data)     R0 (Lower half of calculation result)
;       R1 ( )                              R1 (Indeterminate)
;       R2 (Upper half of operand data)     R2 (Upper half of calculation result)
;       R3 ( )                              R3 (Indeterminate)
;       A0 ( )                              A0 (Unused)
;       A1 ( )                              A1 (Unused)
;
;       Stack amount used: None
;//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
FNOR:
        MOV.W       R2,R3           ; Saves operand data in registers
        MOV.W       R0,R1
;
        XCHG.W      R0,R2           ; Changes registers for each other
        SHL.W       #1,R0           ; Discards sign and adjusts exponent
        SHL.B       #-1,R0L         ; Restores mantissa (upper) part
```

```
FNOR0:
          CMP.B       #1,R0H          ; Underflow?
          JEQ         FNOR_SML        ; --> Yes (goes to set minimum value)
;
          BTST        6,R0            ; MSB of mantissa part is 1?
          JNE         FNOR2           ; --> Yes
          CMP.W       #0,R2           ; Mantissa part is 0?
          JNE         FNOR1           ; --> No
          CMP.B       #0,R0L          ; Mantissa part is 0?
          JEQ         FNOR_NON        ; --> Yes ("no change" returned)
FNOR1:
          SHL.W       #1,R2           ; Shifts mantissa part up
          ROLC.B      R0L
          DEC.B       R0H             ; Exponent  – 1
          JMP         FNOR0
;
; Economized form bit processing
;
FNOR2:
          SHL.W       #1,R2           ; Shifts mantissa part up
          ROLC.B      R0L             ; Discards economized form bit
          DEC.B       R0H             ; Sets – 1 in exponent part
          SHL.B       #1,R0L
          SHL.W       #1,R3           ; Sign →C flag
          RORC.W      R0              ; Sets sign (types of exponent and mantissa (upper)
                                      ; parts adjusted)
          XCHG.W      R0,R2           ; Changes registers for each other
          RTS
;
; Setting minimum value
;
FNOR_SML:
          MOV.W       R3,R2
          AND.W       #8000H,R2       ; Clears all but sign
          OR.W        #0080H,R2       ; Sets 1 in exponent part
          MOV.W       #0,R0           ; Sets minimum value in mantissa part
          RTS
;
; Returning "no change"
;
FNOR_NON:
          MOV.W       R3,R2           ; Restores operand data
          MOV.W       R1,R0
          RTS
;
          .END
```

MEMO

Index

Instruction index

MITSUBISHI SINGLE-CHIP MICROCOMPUTERS
M16C/60 ,M16C/20 Series
Sample Programs Collection Rev.A

March. First Edition 1998
Editioned by
Committee of editing of Mitsubishi Semiconductor
Published by
Mitsubishi Electric Corp., Kitaitami Works