To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology 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 nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation 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 Renesas Technology Corporation or a third party.
2. Renesas Technology 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, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology 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 Renesas Technology Corporation or an authorized Renesas Technology Corporation 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.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. 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.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.

H8/300 Series, H8/300L Series

Application Note - Software

ADE-502-052

3/5/03
Hitachi Ltd.

Notice

When using this document, keep the following in mind:

1. This document may, wholly or partially, be subject to change without notice.
2. All rights are reserved: No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without Hitachi's permission.
3. Hitachi will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's unit according to this document.
4. Circuitry and other examples described herein are meant merely to indicate the characteristics and performance of Hitachi's semiconductor products. Hitachi assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.
5. No license is granted by implication or otherwise under any patents or other rights of any third party or Hitachi, Ltd.
6. MEDICAL APPLICATIONS: Hitachi's products are not authorized for use in MEDICAL APPLICATIONS without the written consent of the appropriate officer of Hitachi's sales company. Such use includes, but is not limited to, use in life support systems. Buyers of Hitachi's products are requested to notify the relevant Hitachi sales offices when planning to use the products in MEDICAL APPLICATIONS.

Contents <Table of Contents>

Section 1 Move Data

1.1 Set Constants

MCU: H8/300 Series
H8/300L Series

Label name: FILL

1.1.1 Function

1. The software FILL places 1-byte constants in the data memory area.
2. The data memory area can have a free size.
3. Constants can have any length within the range 1 to 255 bytes.
4. This function is useful in initializing a RAM area.

1.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	Byte count (number of bytes)	R0L	1
	Constants	R0H	1
	Start address	R1	2
Output	—	—	—

1.1.3 Internal Register and Flag Changes

R0H	R0L	R1	R2	R3	R4	R5	R6	R7
×	×	×	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	•	•	×	×	×	•

× : Unchanged
• : Indeterminate
↕ : Result

1.1.4 Specifications

Program memory (bytes)
10
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
3068
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

1.1.5 Note

The specified clock cycle count (3068) is for 255 bytes of constants.

1.1.6 Description

1. Details of functions
   a. The following arguments are used with the software FILL:
      R0L: Contains, as an input argument, the number of bytes to be placed in the data memory area holding constants.
      R0H: Contains, as an input argument, 1-byte constants to be placed in the data memory area.
      R1: Contains, as an input argument, the start address of the data memory area holding constants.

b. Figure 1.1 shows an example of the software FILL being executed.

When the input arguments are set as shown in (1), the constant H'34 set in R0H is placed in the data memory area as shown in (2).

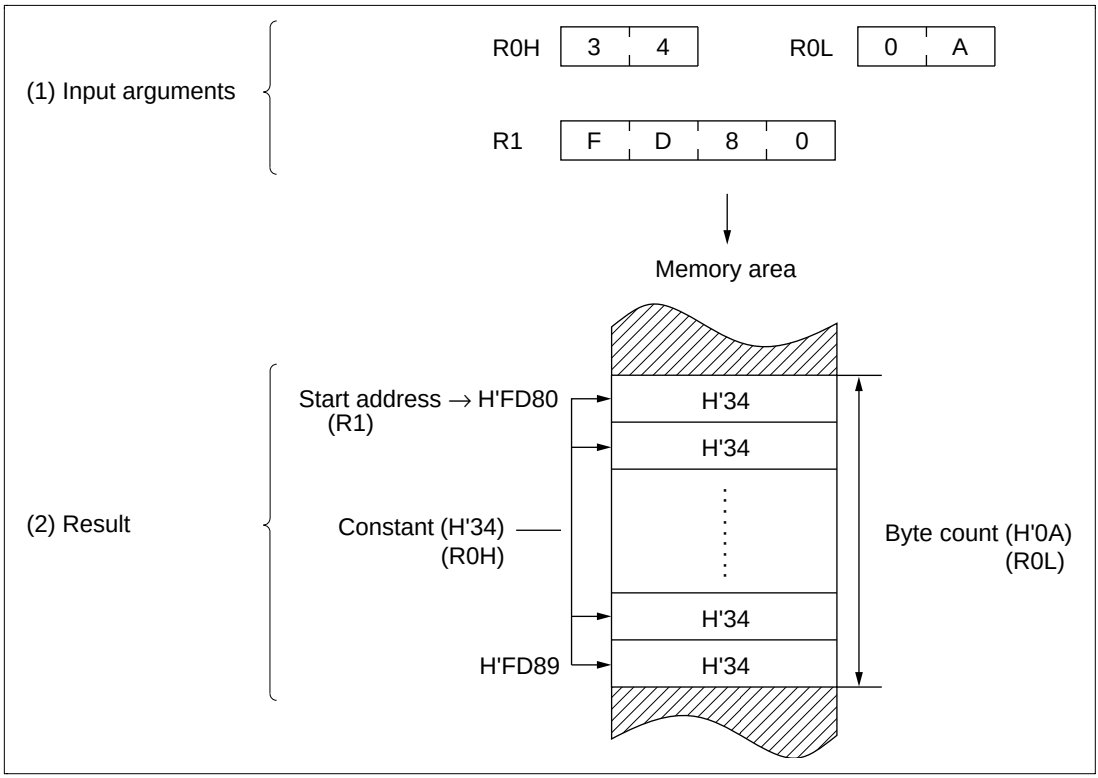


Figure 1.1 Example of Software FILL Execution

2. Notes on usage
   a. R0L is one byte long and should satisfy the relation H'01 ≤ R0L ≤ H'FF.
   b. Do not set "0" in R0L; otherwise, the software FILL can no longer be terminated.
3. Data memory

   The software FILL does not use the data memory.

4. Example of use

   Set a constant, a byte count, and a start address in the arguments and call the software FILL as a subroutine.

```
WORK1    . DATA. B   0          ·····  Reserves a data memory area (1 byte:
                                       contents=H'00) in which the user program
                                       places the number of bytes to be moved.
WORK2    . DATA. B   0          ·····  Reserves a data memory area (1 byte:
                                       contents=H'00) in which the user program
                                       places constants.
WORK3    . RES. B    10         ·····  Reserves a data memory area (10 bytes)
                                       that is set by the software FILL.
         MOV. B      @WORK1, R0L ····  Places the number of bytes set by the user
                                       program in the R0L input argument.
         MOV. B      @WORK2, R0H ····  Places the constants set by the user
                                       program in the R0H argument.
         MOV. W      #WORK3, R1  ····  Places the start address of the data
                                       memory area allocated by the user
                                       program in the R1 argument.
         JSR         @FILL       ····  Calls the software FILL as a subroutine.
```

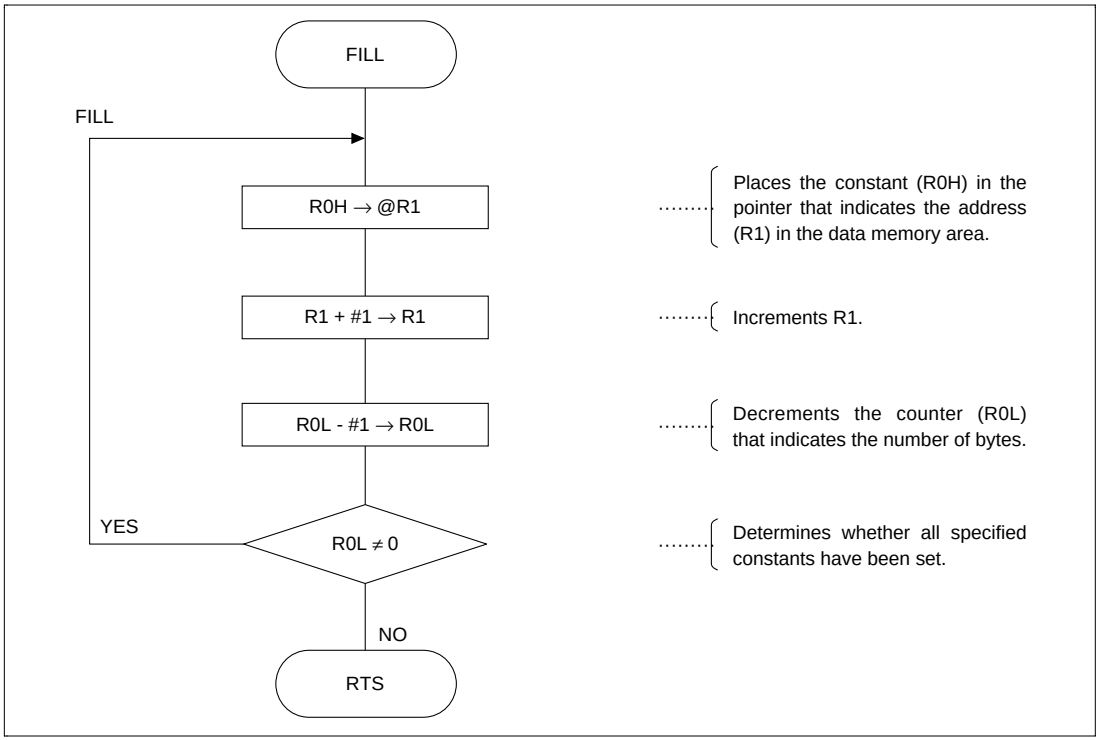
5. Operation

   a. R1 is used as the pointer that indicates the address of the data memory area in which constants are placed.

   b. The constants set in R0H in 16-bit absolute addressing mode are stored sequentially in the data memory area.

   c. R0L is used as the pointer that indicates the number of bytes in the data memory area in which constants are placed. R0L is decremented each time a constant is placed in the data memory area until it reaches 0.

1.1.7 Flowchart



1.1.8 Program List

```
*** H8/300 ASSEMBLER                VER 1.0B **   08/18/92 11:04:12
PROGRAM NAME =
    1                                       ;*************************************************************
    2                                       ;*
    3                                       ;*   00 - NAME              :FILL OF CONSTANT DATA (FILL)
    4                                       ;*
    5                                       ;*************************************************************
    6                                       ;*
    7                                       ;*   ENTRY           :R0L   (Byte counter)
    8                                       ;*                    R0H   (Constant data)
    9                                       ;*                    R1    (Start address)
   10                                       ;*
   11                                       ;*   RETURN          :NOTHING
   12                                       ;*
   13                                       ;*************************************************************
   14                                       ;
   15 FILL_cod C 0000                            .SECTION        FILL_code,CODE,ALIGN=2
   16                                            .EXPORT         FILL
   17                                       ;
   18 FILL_cod C      00000000              FILL .EQU    $               ;Entry Point
   19 FILL_cod C 0000 6890                       MOV.B   R0H.@R1         ;Store constant data
   20 FILL_cod C 0002 0B01                       ADDS.W  #1,R1           ;Increment address pointer
   21 FILL_cod C 0004 1A08                       DEC.B   R0L             ;Decrement byte counter
   22 FILL_cod C 0006 46F8                       BNE     FILL            ;Branch if Z flag = 0
   23                                       ;
   24 FILL_cod C 0008 5470                       RTS
   25                                       ;
   26                                            .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

1.2 Move Block 1

MCU: H8/300 Series
H8/300L Series

Label name: MOVE1

1.2.1 Function

1. The software MOVE1 moves block data from one data memory area to another.
2. The source and destination data memory areas can have a free size.
3. The block data can have any length within the range 1 to 255 bytes.

1.2.2 Arguments

Description		Memory area	Data length (bytes)
Input	Byte count (number of bytes)	R0L	1
	Start address of source area	R1	2
	Start address of destination area	R2	2
Output	—	—	—

1.2.3 Internal Register and Flag Changes

R0H	R0L	R1	R2	R3	R4	R5	R6	R7
×	×	×	×	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	•	•	×	×	×	•

× : Unchanged
• : Indeterminate
↕ : Result

1.2.4 Specifications

Program memory (bytes)
14
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
4598
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

1.2.5 Note

The specified clock cycle count (4598) is for 255 bytes of block data.

1.2.6 Description

1. Details of functions
   a. The following arguments are used with the software MOVE1:
      R0L: Contains, as an input argument, the number of bytes of block data.
      R1: Contains, as an input argument, the start address of the source data memory area.
      R2: Contains, as an input argument, the start address of the destination data memory area.

b. Figure 1.2 shows an example of the software MOVE1 being executed.

When the input arguments are set as shown in (1), the data is moved block by block from the source (H'FD80 to H'FD89) to the destination (H'FE80 to H'FE89) as shown in (2).

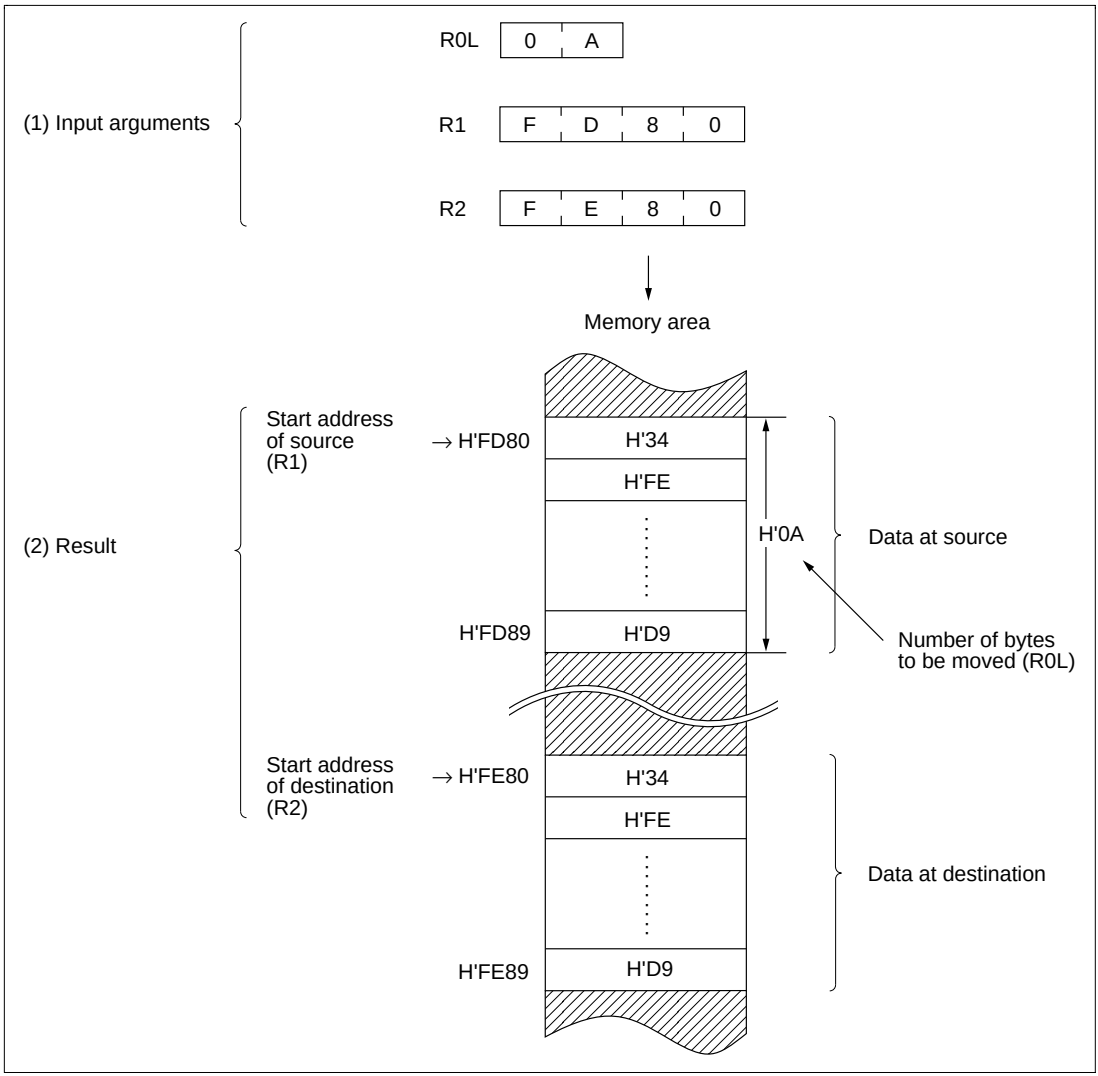


Figure 1.2 Example of Software MOVE1 Execution

2. Notes on usage
   a. R0L is one byte long and should satisfy the relation H'01 ≤ R0L ≤ H'FF.
   b. Do not set "0" in R0L; otherwise, the software MOVE1 can no longer be terminated.
   c. Set the input arguments, ensuring that the source data memory area (A) does not overlap the destination data memory area (C) as shown in figure 1.3. In the case of figure1.3, the overlapped block data (B) at the source is destroyed.

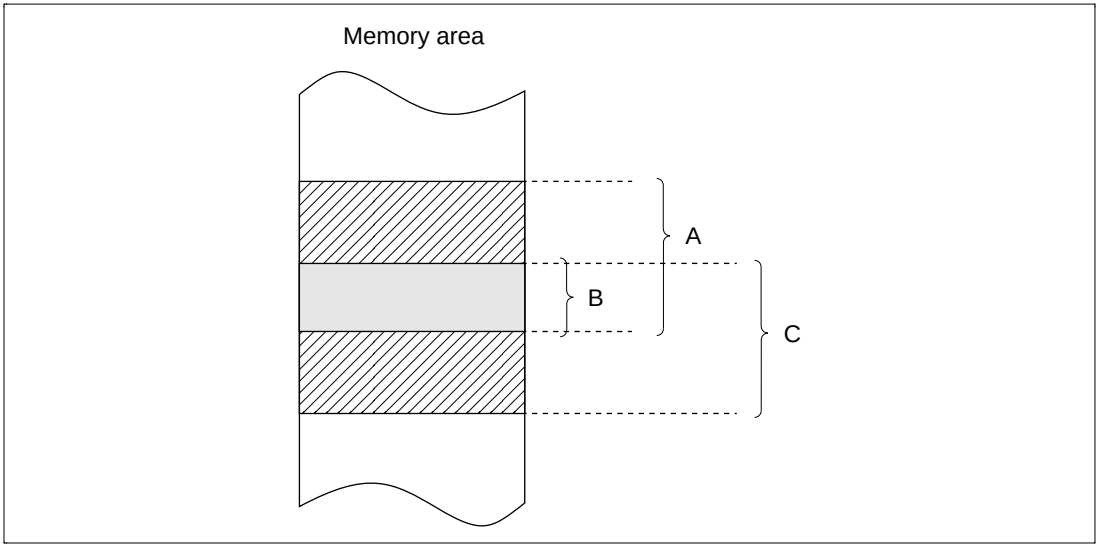


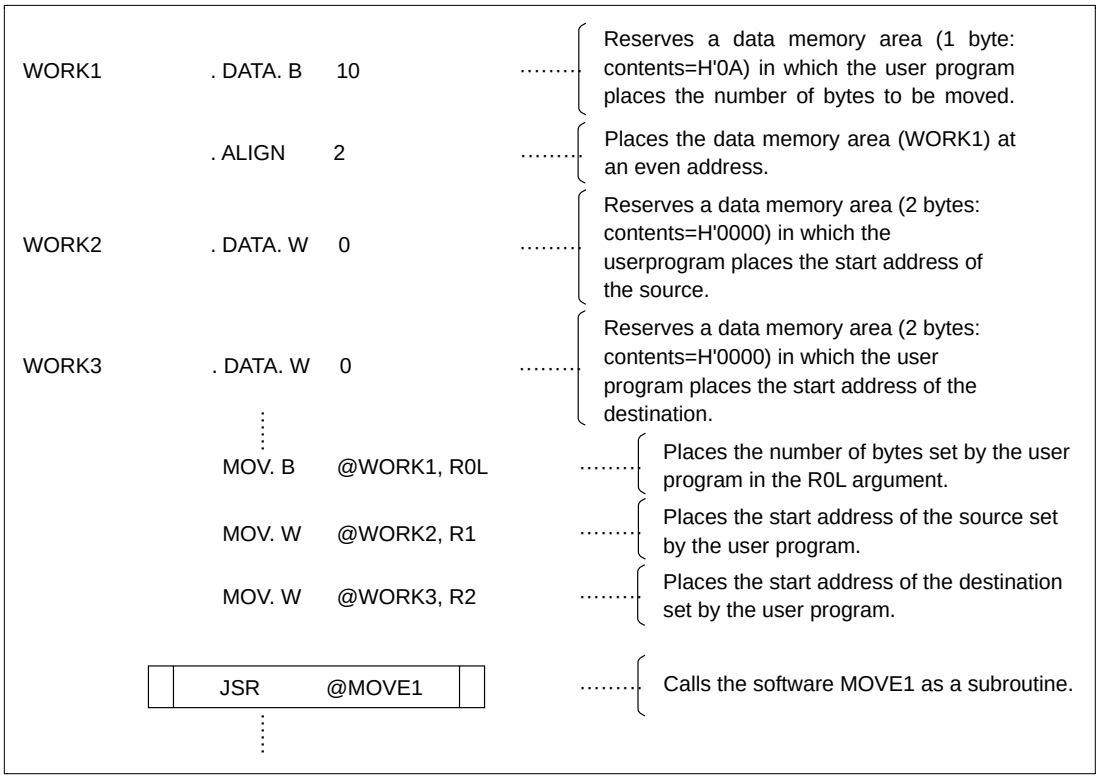
Figure 1.3 Moving Block Data with Overlapped Data Memory Areas

3. Data memory

   The software MOVE1 does not use the data memory.

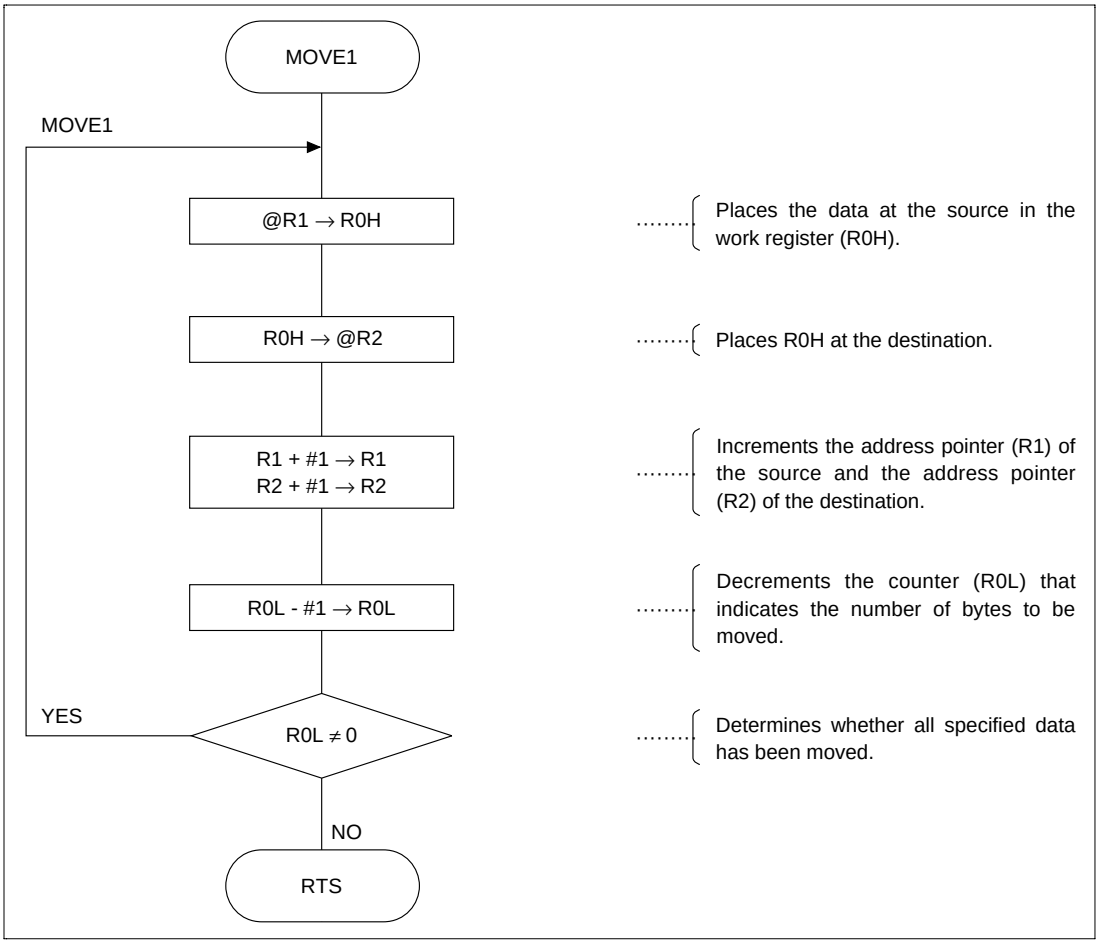
4. Example of use

Set the start address of a source, the start address of a destination, and the number of bytes to be moved in the arguments and call the software MOVE1 as a subroutine.



5. Operation
   a. R1 is used as the pointer that indicates the address of the source and R2 the pointer that indicates the address of the destination.
   b. The cycle of storing the data at the source in the work register (R0H) and then at the destination is repeated in 16-bit absolute addressing mode.
   c. R0L is used as the counter that indicates the number of bytes moved. It is decremented each time 1-byte data is moved until it reaches 0.

1.2.7 Flowchart



1.2.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:45:34
PROGRAM NAME =
    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00-NAME         :BROCK DATA TRANSFER (MOVE1)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0L    (Byte counter)
    8                                   ;*                    R1     (Source data start address)
    9                                   ;*                    R2     (Destination data start address)
   10                                   ;*
   11                                   ;*   RETURN          :NOTHING
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 MOVE1_co C 0000                        .SECTION        MOVE1_code,CODE,ALIGN=2
   16                                        .EXPORT         MOVE1
   17                                   ;
   18 MOVE1_co C      00000000          MOVE1        .EQU    $               ;Entry point
   19 MOVE1_co C 0000 6810                   MOV.B   @R1,R0H         ;Load source address data to R0H
   20 MOVE1_co C 0002 68A0                   MOV.B   R0H,@R2         ;Store R0H to destination address
   21 MOVE1_co C 0004 0B01                   ADDS.W  #1,R1           ;Increment source address pointer
   22 MOVE1_co C 0006 0B02                   ADDS.W  #1,R2           ;Increment destination address pointer
   23 MOVE1_co C 0008 1A08                   DEC     R0L             ;Decrement byte counter
   24 MOVE1_co C 000A 46F4                   BNE     MOVE1           ;Branch if byte counter = 0
   25                                   ;
   26 MOVE1_co C 000C 5470                   RTS
   27                                   ;
   28                                        .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

1.3 Move Block 2 (Example of the EEPMOV Instruction)

MCU: H8/300 Series
H8/300L Series

Label name: MOVE2

1.3.1 Function

1. The software MOVE2 moves block data from one data memory area to another.
2. The source and destination data memory areas can have a free size.
3. Data can be moved even where the source data memory area overlaps the destination data memory area.
4. This is an example of the application software EEPMOV (move block instruction).

1.3.2 Arguments

Description		Memory area	Data length (bytes)
Input	Byte count (number of bytes)	R4L	1
	Start address of source area	R5	2
	Start address of destination area	R6	2
Output	Error	C flag (CCR)	

1.3.3 Internal Register and Flag Changes

R0H	R0L	R1	R2	R3	R4H	R4L	R5	R6	R7
×	×	•	×	×	×	×	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

1.3.4 Specifications

Program memory (bytes)
58
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
1083
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

1.3.5 Note

The specified clock cycle count (1083) is for 255 bytes of block data.

1.3.6 Description

1. Details of functions
   a. The following arguments are used with the software MOVE2:
      R4L: Contains, as an input argument, the number of bytes of block data.
      R5: Contains, as an input argument, the start address of the source data memory area.
      R6: Contains, as an input argument, the start address of the destination data memory area.
      C flag (CCR): Determines the presence or absence of an error in the data length or address of the software MOVE2.
      C=0: All data has been moved.
      C=1: An input argument has an error.

b. Figure 1.4 shows an example of the software MOVE2 being executed.
When the input arguments are set as shown in (1), the data is moved block by block from the source (H'FD80 to H'FD89) to the destination (H'FE80 to H'FE89) as shown in (2).

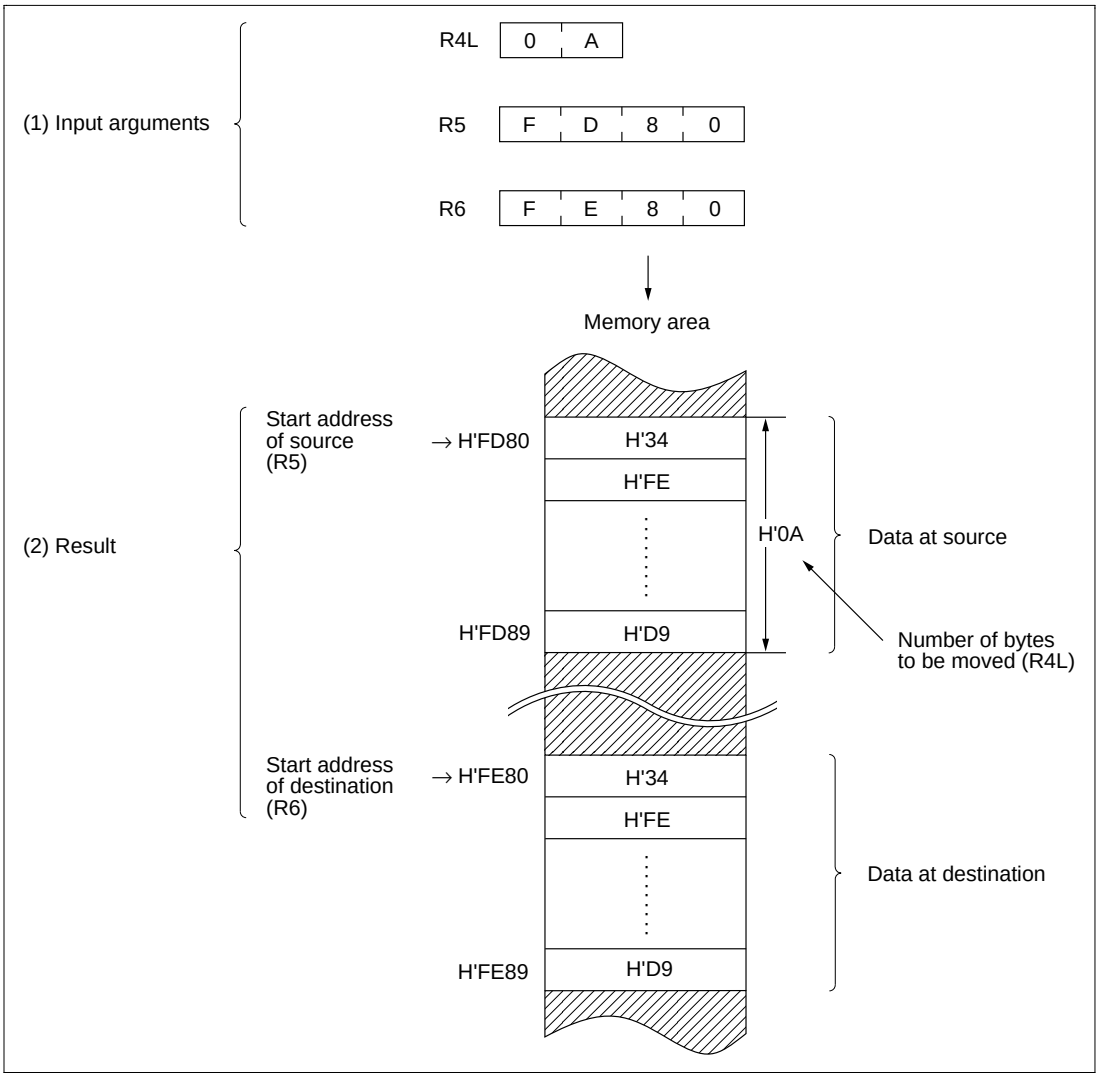


Figure 1.4 Example of Software MOVE2 Execution

2. Notes on usage
   a. R4L is one byte long and should satisfy the relation H'01 ≤ R4L ≤ H'FF.
   b. The source or destination data memory area must not extend over the end address (H'FFFF) to the start address (H'0000) as shown in figure 1.5; otherwise, the software MOVE2 fails.

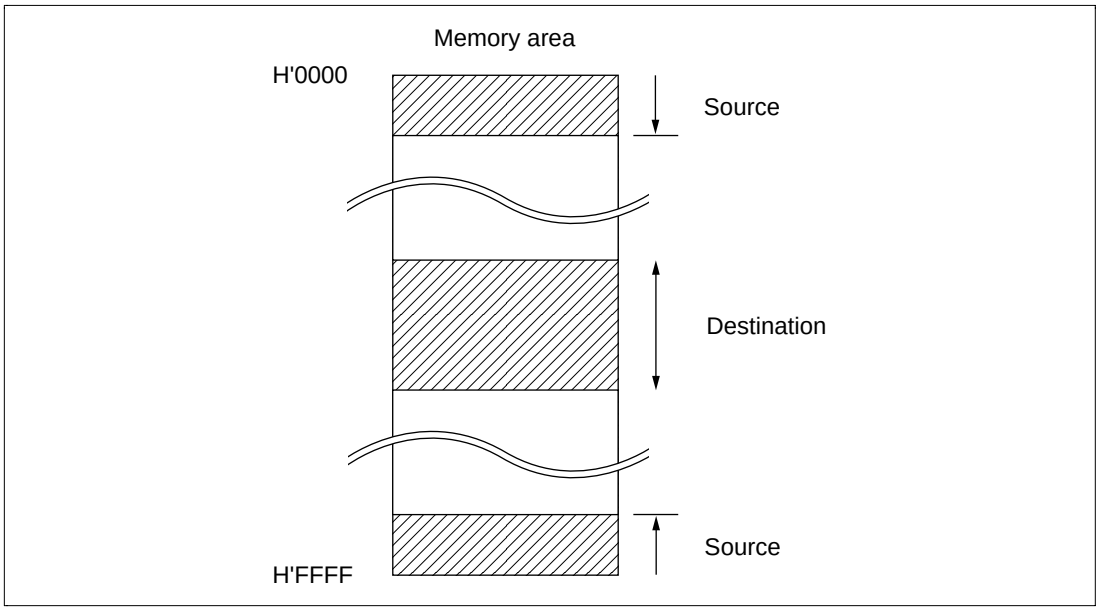


Figure 1.5 Moving Block Data with Data Memory Area Extending over the Higher to Lower Addresses

3. Data memory
   The software MOVE2 does not use the data memory.

4. Example of use

Set the start address of a source, the start address of a destination, and the number of bytes to be moved in the arguments and call the software MOVE2 as a subroutine.

```
WORK1    . DATA. B   10      ..... Reserves a data memory area (1 byte:
                                   contents=H'0A) in which the user program
                                   places the number of bytes to be moved.
         . ALIGN     2       ..... Places the data memory area (WORK1) at
                                   an even address.
WORK2    . DATA. W   0       ..... Reserves a data memory area (2 bytes:
                                   contents=H'0000) in which the user
                                   program places the start address of the
                                   source.
WORK3    . DATA. W   0       ..... Reserves a data memory area (2 bytes:
                                   contents=H'0000) in which the user
                                   program places the start address of the
                                   destination.
           :
         MOV. B    @WORK1, R4L  ..... Places the number of bytes set by the user
                                   program in the R0L argument.
         MOV. W    @WORK2, R5   ..... Places the start address of the source set
                                   by the user program.
         MOV. W    @WORK3, R6   ..... Places the start address of the destination
                                   set by the user program.
       +-----------------------+
       |  JSR      @MOVE2      |  ..... Calls the software MOVE2 as a subroutine.
       +-----------------------+
           :
```

5. Operation

   a. R5 is used as the pointer that indicates the address of the source and R6 the pointer that indicates the address of the destination.

   b. R4L is used as the counter that indicates the number of bytes moved. It is decremented each time 1-byte data is moved until it reaches 0.

   c. When the input argument R4L is 0 or the start address of the source equals that of the destination, the C flag is set to 1 (error indicator) and the software MOVE2 terminates.

d. When the start address (B) of the destination data memory area is between the start address (A) and the end address (A + n – 1) of the source data memory area (A < B < A + n – 1; see figure 1.6), the data is moved sequentially from the higher address of the source in 16-bit absolute addressing mode.

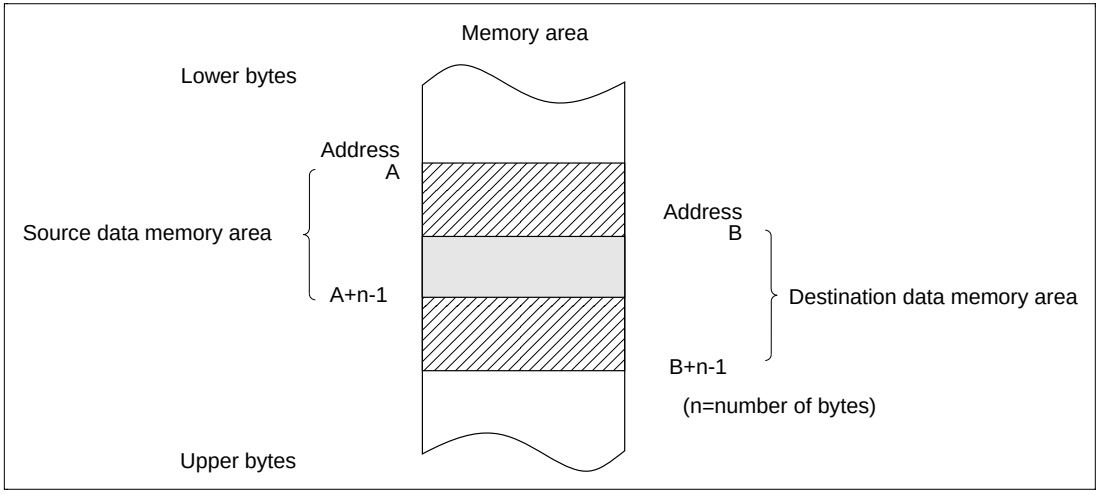
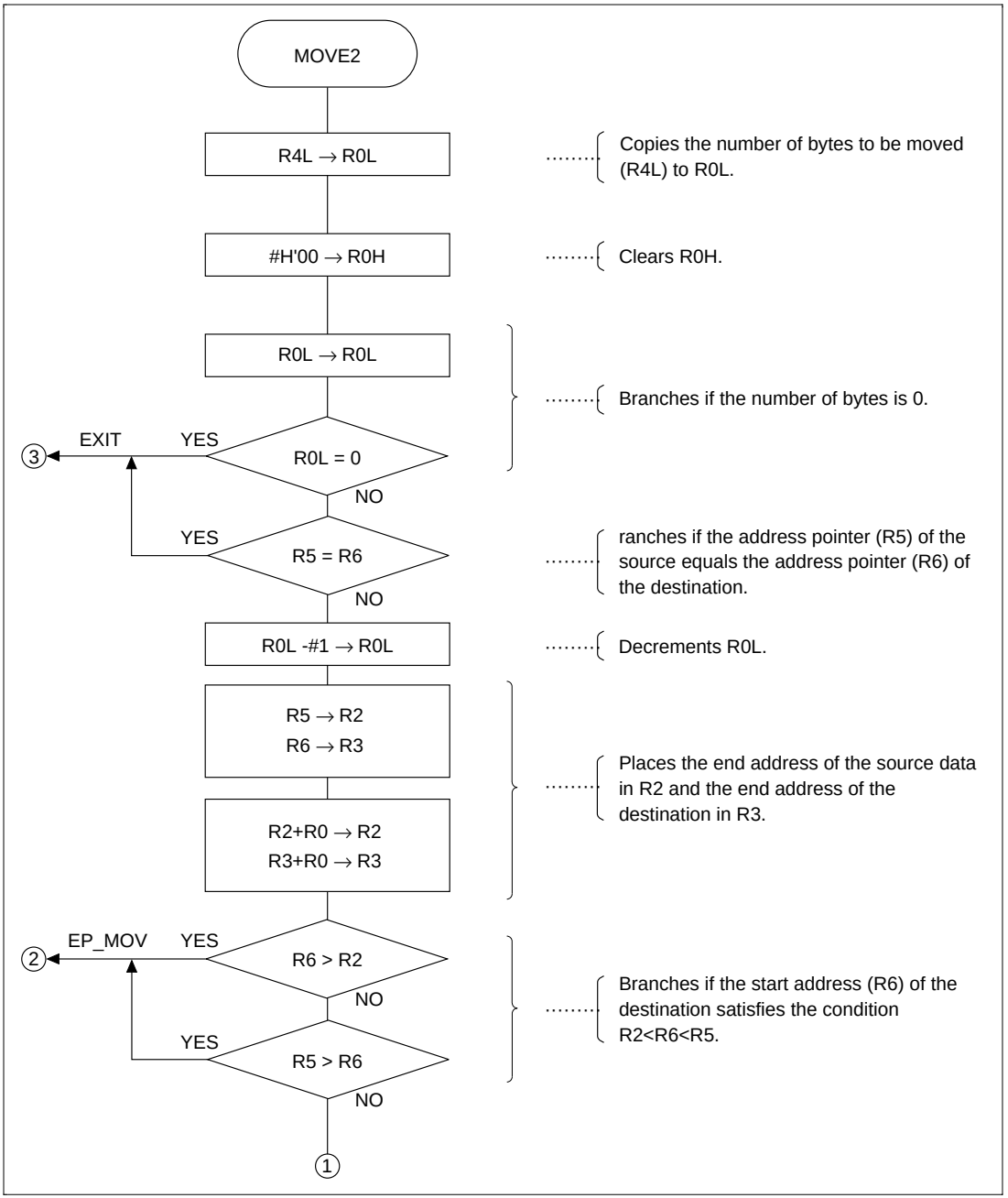
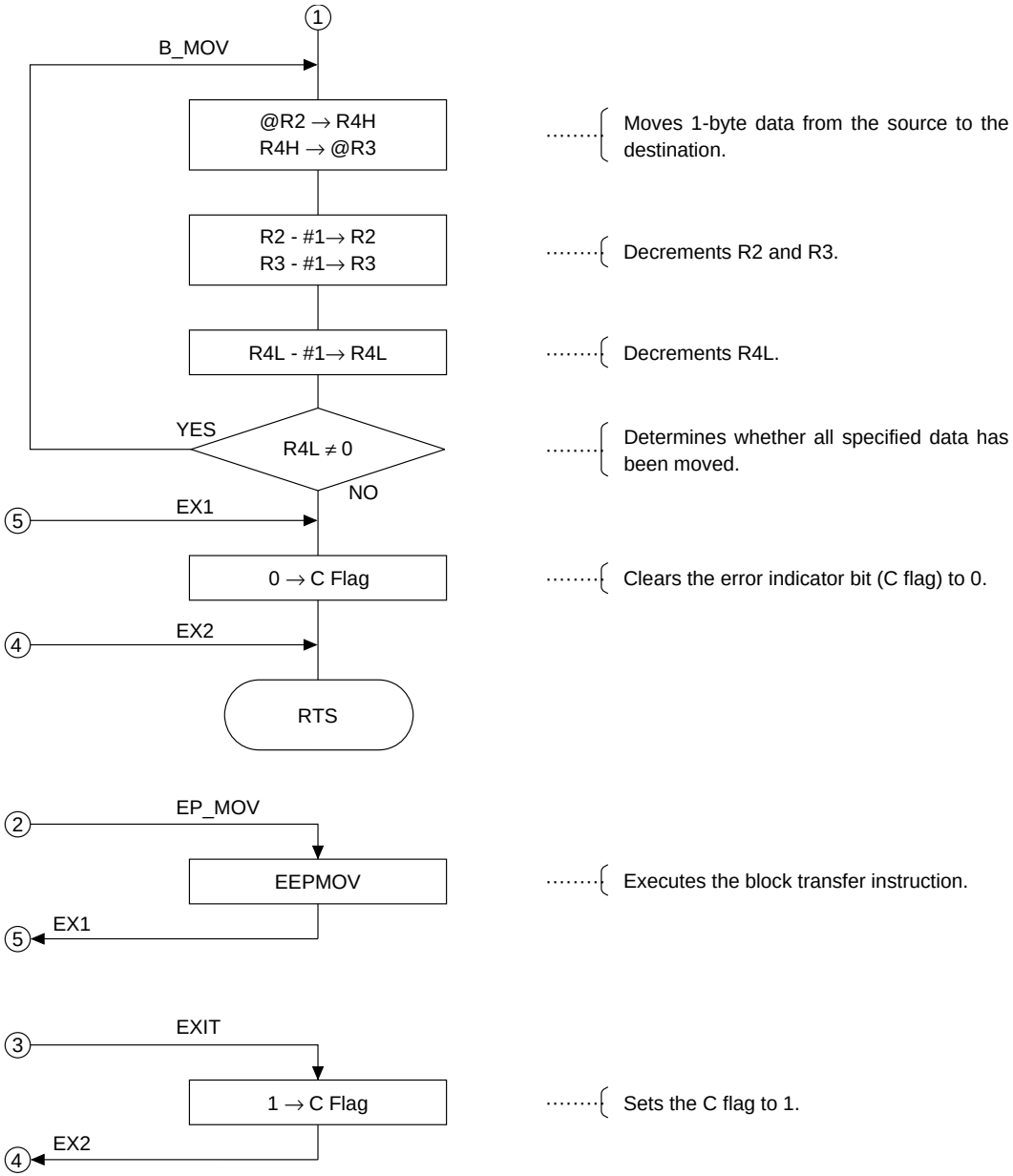


Figure 1.6 Moving Data with Overlapped Data Memory Areas

e. Except in the case of d., the EEPMOV instruction is used to move the data sequentially from the lower address.

1.3.7 Flowchart




1
B_MOV
@R2 → R4H
R4H → @R3
Moves 1-byte data from the source to the destination.
R2 - #1→ R2
R3 - #1→ R3
Decrements R2 and R3.
R4L - #1→ R4L
Decrements R4L.
YES
R4L ≠ 0
NO
Determines whether all specified data has been moved.
5
EX1
0 → C Flag
Clears the error indicator bit (C flag) to 0.
4
EX2
RTS
2
EP_MOV
EEPMOV
Executes the block transfer instruction.
5
EX1
3
EXIT
1 → C Flag
Sets the C flag to 1.
4
EX2

1.3.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:46:07
PROGRAM NAME =
    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :BROCK DATA TRANSFER (MOVE2)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R4L    (Byte counter)
    8                                   ;*                    R5     (Source data start address)
    9                                   ;*                    R6     (Destination data start address)
   10                                   ;*
   11                                   ;*   RETURN          :C bit of CCR (C=0;TRUE , C=1;FALSE)
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 MOVE2_co C 0000                        .SECTION        MOVE2_code,CODE,ALIGN=2
   16                                        .EXPORT         MOVE2
   17                                   ;
   18 MOVE2_co C      00000000          MOVE2        .EQU    $       ;Entry point
   19 MOVE2_co C 0000 0CC8                   MOV.B   R4L,R0L
   20 MOVE2_co C 0002 F000                   MOV.B   #H'00,R0H
   21 MOVE2_co C 0004 0C88                   MOV.B   R0L,R0L
   22 MOVE2_co C 0006 472E                   BEQ     EXIT            ;If byte counter="0" then exit
   23 MOVE2_co C 0008 1D56                   CMP.W   R5,R6
   24 MOVE2_co C 000A 472A                   BEQ     EXIT            ;If R5=R6 then exit
   25 MOVE2_co C 000C 1A08                   DEC.B   R0L
   26 MOVE2_co C 000E 0D52                   MOV.W   R5,R2
   27 MOVE2_co C 0010 0D63                   MOV.W   R6,R3
   28 MOVE2_co C 0012 0902                   ADD.W   R0,R2           ;Set end address of source data
   29 MOVE2_co C 0014 0903                   ADD.W   R0,R3           ;Set end address of destination data
   30 MOVE2_co C 0016 1D26                   CMP.W   R2,R6
   31 MOVE2_co C 0018 4214                   BHI     EP_MOV          ;Branch if R6>R2
   32 MOVE2_co C 001A 1D65                   CMP.W   R6,R5
   33 MOVE2_co C 001C 4210                   BHI     EP_MOV          ;Branch if R5>R6
   34 MOVE2_co C 001E                   B_MOV
   35 MOVE2_co C 001E 6824                   MOV.B   @R2,R4H         ;Load source data to R4H
   36 MOVE2_co C 0020 68B4                   MOV.B   R4H,@R3         ;Store R4H to destination
   37 MOVE2_co C 0022 1B02                   SUBS.W  #1,R2           ;Decrement source data pointer
   38 MOVE2_co C 0024 1B03                   SUBS.W  #1,R3           ;Decrement destination data pointer
   39 MOVE2_co C 0026 1A0C                   DEC.B   R4L
   40 MOVE2_co C 0028 46F4                   BNE     B_MOV           ;Branch if R4L=0
   41 MOVE2_co C 002A                   EX1
   42 MOVE2_co C 002A 06FE                   ANDC.B  #H'FE,CCR       ;Clear C flag of CCR
   43 MOVE2_co C 002C                   EX2
   44 MOVE2_co C 002C 5470                   RTS
   45 MOVE2_co C 002E                   EP_MOV
   46 MOVE2_co C 002E 7B5C598F               EEPMOV
   47 MOVE2_co C 0032 06FE                   ANDC.B  #H'FE,CCR       ;Clear C flag of CCR
   48 MOVE2_co C 0034 40F4                   BRA     EX1
   49 MOVE2_co C 0036                   EXIT
   50 MOVE2_co C 0036 0401                   ORC.B   #H'01,CCR       ;Set c flag for false
   51 MOVE2_co C 0038 40F2                   BRA     EX2
   52                                   ;
   53                                        .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

1.4 Move Character Strings

MCU: H8/300 Series
H8/300L Series

Label name: MOVES

1.4.1 Function

1. The software MOVES moves character string block data from one data memory area to another.
2. When the delimiting H'00 appears in the block data, the software MOVES terminates.
3. The source or destination data memory area can have a free size.

1.4.2 Arguments

Description		Memory area	Data length (bytes)
Input	Start address of source area	R1	2
	Start address of destination area	R2	2
Output	—	—	—

1.4.3 Internal Register and Flag Changes

R0H	R0L	R1	R2	R3	R4	R5	R6	R7
•	×	×	×	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	•	•	×	×	×	•

× : Unchanged
• : Indeterminate
↕ : Result

1.4.4 Specifications

Program memory (bytes)
14
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
5116
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

1.4.5 Note

The specified clock cycle count (4598) is for 255 bytes of character string block data.

1.4.6 Description

1. Details of functions
   a. The following arguments are used with the software MOVES:
      R1: Contains, as an input argument, the start address of the source data memory area.
      R2: Contains, as an input argument, the start address of the destination data memory area.
   b. Figure 1.7 shows an example of the software MOVES being executed.
      When the input arguments are set as shown in (1), the data is moved block by block from the source (H'A000 to H'A009) to the destination (H'B000 to H'B009) as shown in (2).

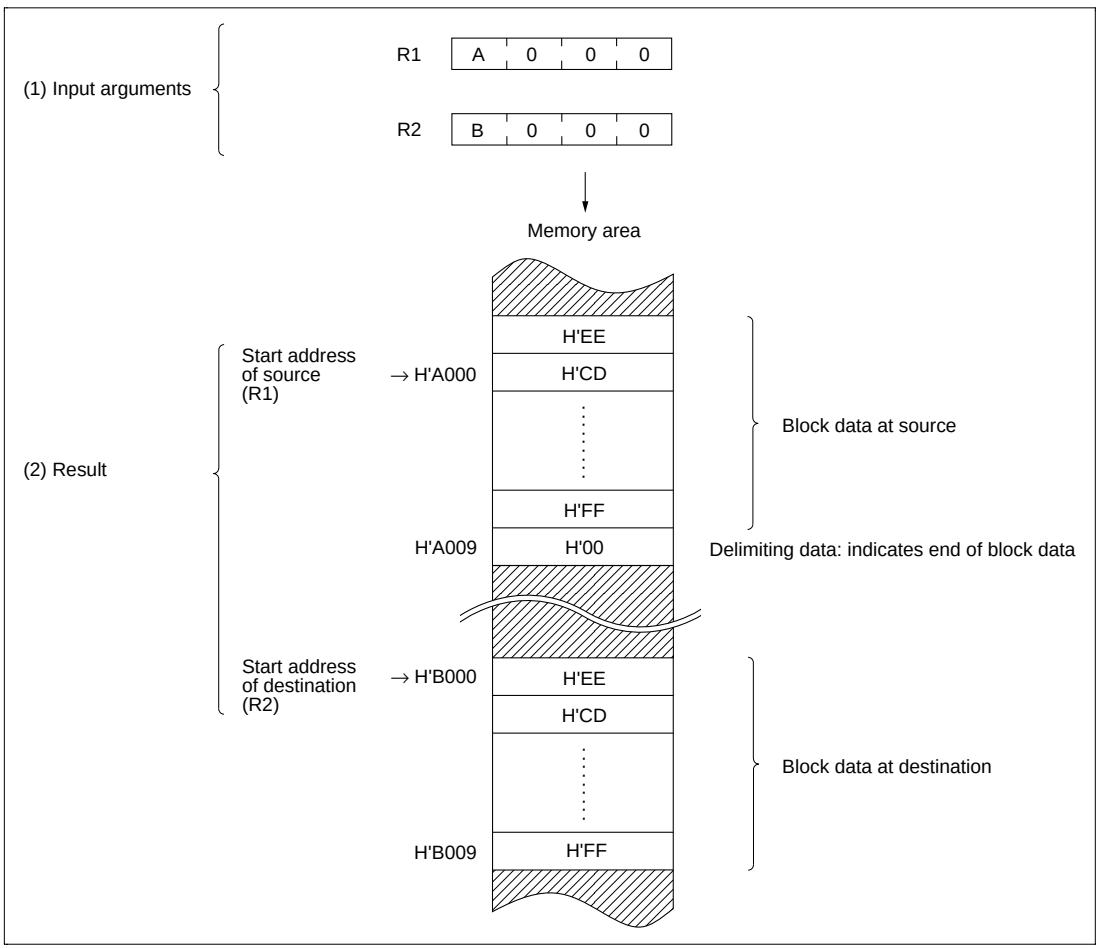


Figure 1.7 Example of Software MOVES Execution

2. Notes on usage
   a. Do not set H'00 in the source block data because H'00 is used as delimiting data; otherwise, the software MOVES terminates.
   b. Set input arguments, ensuring that the source data memory area (A) does not overlap the destination data memory area (C) as shown in figure 1.8. In the case of figure 1.8, the overlapped block data (B) at the source is destroyed.

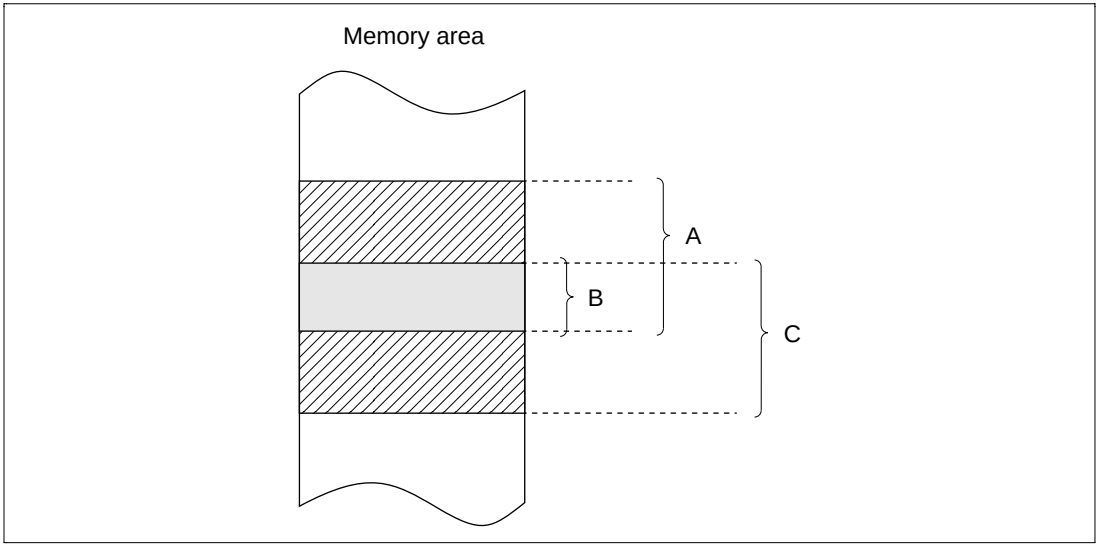
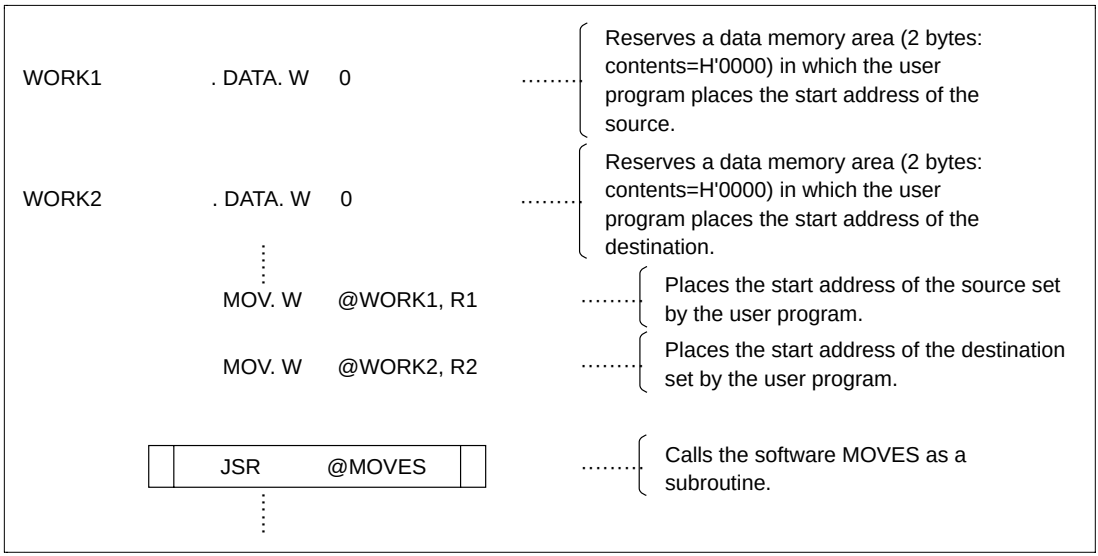


Figure 1.8 Moving Block Data with Overlapped Data Memory Areas

3. Data memory
   The software MOVES does not use the data memory.

4. Example of use

   Set the start address of a source and the start address of a destination in the arguments and call the software MOVES as a subroutine.



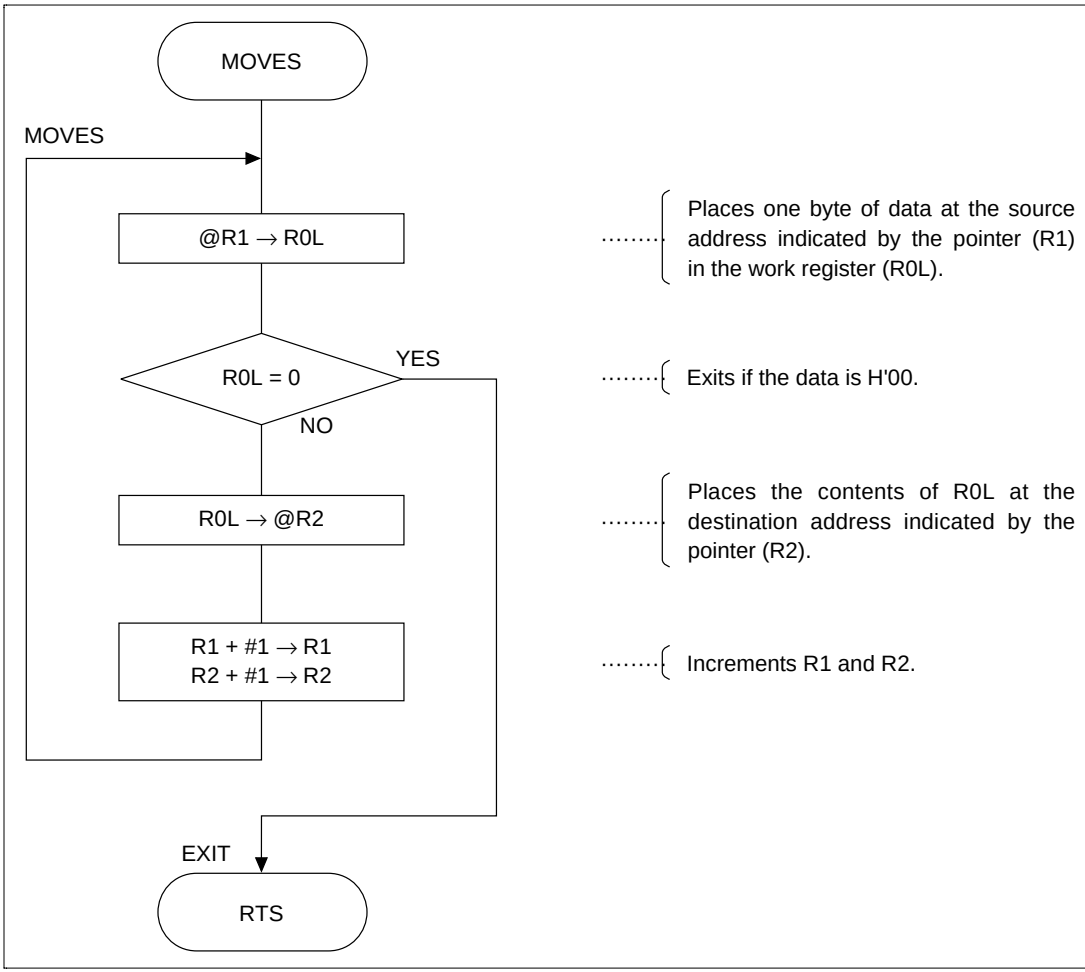
5. Operation

   a. R1 is used as the pointer that indicates the address of the source and R2 the pointer that indicates the address of the destination.

   b. The cycle of storing the data at the source in the work register (R0L) and then at the destination is repeated in 16-bit absolute addressing mode.

   c. During the cycle, whether the R0L data is delimiting data is determined. If it is delimiting data (H'00), the software MOVES terminates; if not, moving the data continues.

1.4.7 Flowchart



1.4.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:46:36
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :TRANSFER OF STRING (MOVES)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R1     (Source address)
    8                                   ;*                    R2     (Destination address)
    9                                   ;*
   10                                   ;*   RETURN          :NOTHING
   11                                   ;*
   12                                   ;*********************************************************************
   13                                   ;
   14 MOVES_co C 0000                        .SECTION        MOVES_code,CODE,ALIGN=2
   15                                        .EXPORT         MOVES
   16                                   ;
   17 MOVES_co C      00000000           MOVES               .EQU    $       ;Entry point
   18 MOVES_co C 0000 6818                   MOV.B   @R1,R0L         ;Load source address data to R0L
   19 MOVES_co C 0002 4708                   BEQ     EXIT            ;Branch if R0H = R0L
   20 MOVES_co C 0004 68A8                   MOV.B   R0L,@R2         ;Store R0H to destination address
   21 MOVES_co C 0006 0B01                   ADDS.W  #1,R1           ;Increment source address pointer
   22 MOVES_co C 0008 0B02                   ADDS.W  #1,R2           ;Increment destination address pointer
   23 MOVES_co C 000A 40F4                   BRA     MOVES           ;Branch always
   24                                   ;
   25 MOVES_co C 000C                   EXIT
   26 MOVES_co C 000C 5470                   RTS
   27                                   ;
   28                                        .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

Section 2 Branch by Table

2.1 Branch by Table

MCU: H8/300 Series
H8/300L Series

Label name: CCASE

2.1.1 Function

1. The software CCASE determines the start address of a processing routine for a 1-word (2-byte) command.
2. This function is useful in decoding data input from the keyboard or performing a process appropriate for input data.

2.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	Command	R0	2
	Start address of data table	R1	2
Output	Start address of processing routine	R4	2
	Command	C flag (CCR)	

2.1.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	×	•	↕	•	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

2.1.4 Specifications

Program memory (bytes)
28
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
74
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

2.1.5 Note

The specified clock cycle count (74) is for the example of figure 2.1 being executed.

2.1.6 Description

1. Details of functions
   a. The following arguments are used with the software CCASE:
      R0: Contains, as an input argument, 2-byte commands.
      R1: Contains, as an input argument, the start address of a data table storing the command (R0) and the start address of a processing routine.
      R4: Contains, as an output argument, the start address (2 bytes) of the processing routine for the command (R0).
      C flag (CCR): Determines the state of data after the software CCASE has been executed.
      C flag = 1: The data matching the command set in R0 was on the data table.
      C flag = 0: The data matching the command set in R0 was not on the data table.

b. Figure 2.1 shows an example of the software CCASE being executed.

When the input arguments are set as shown in (1), the program refers to the data table (see figure 2.1 and places the start address of the processing routine in R4 as shown in (2).

c. Executing the software CCASE requires a data table as shown in figure 2.1. The data table is as follows:

(i) The table contains data groups each consisting of 4 bytes (2 words) beginning with address H'FD80 and delimiting data H'0000 indicating the end of the table.

(ii) The first word of each data group (2 words) contains a command and the second word contains the start address of the processing routine in the order of the upper bytes followed by the lower bytes.

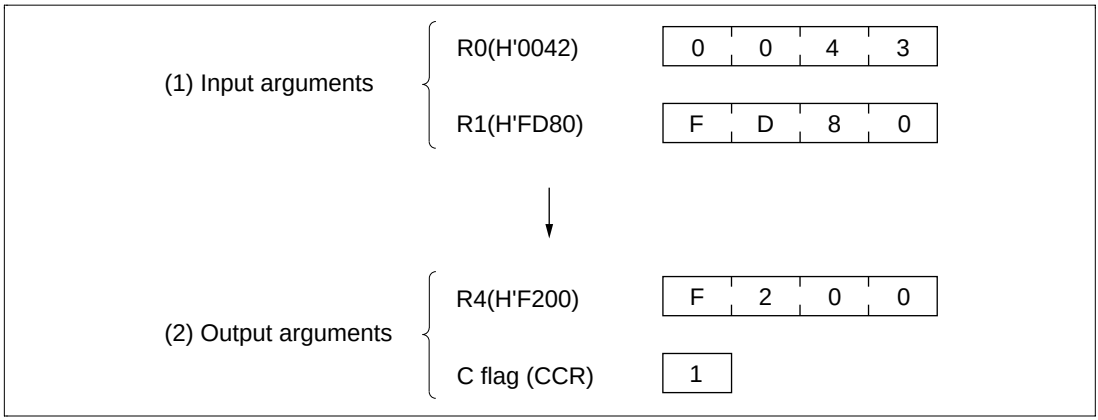


Figure 2.1 Example of Software CCASE Execution

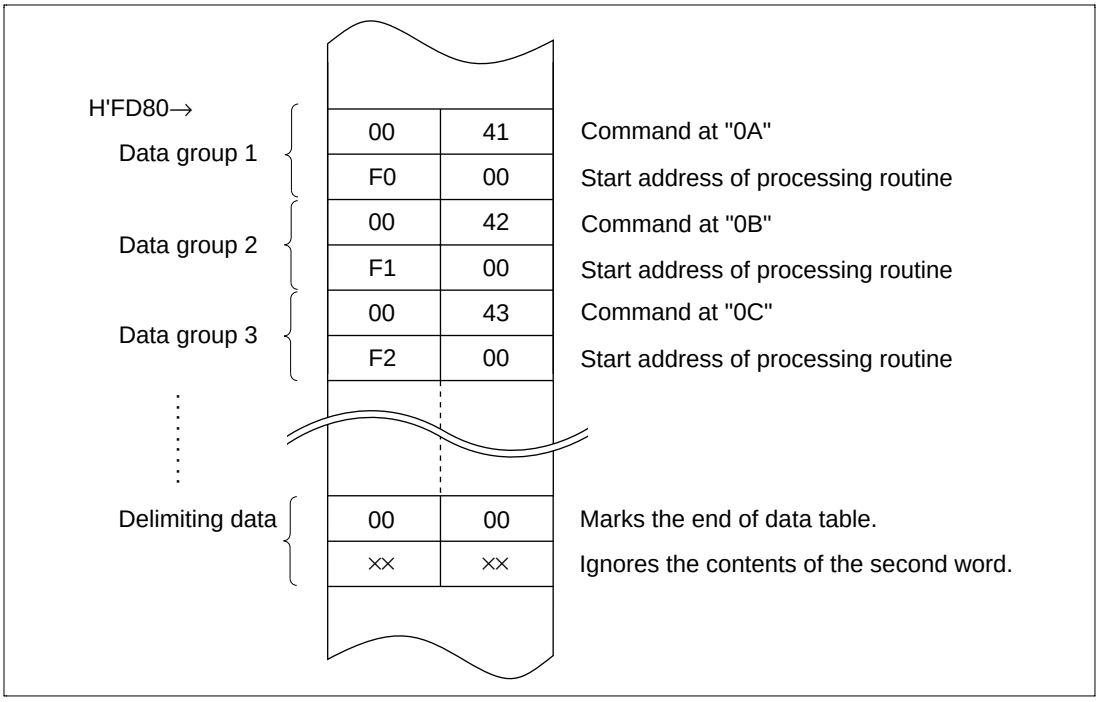


Figure 2.2 Example of Data Table

2. Notes on usage

   Do not use H'0000 as a command in the data table because H'0000 is used as delimiting data.

3. Data memory

   The software CCASE does not use the data memory.

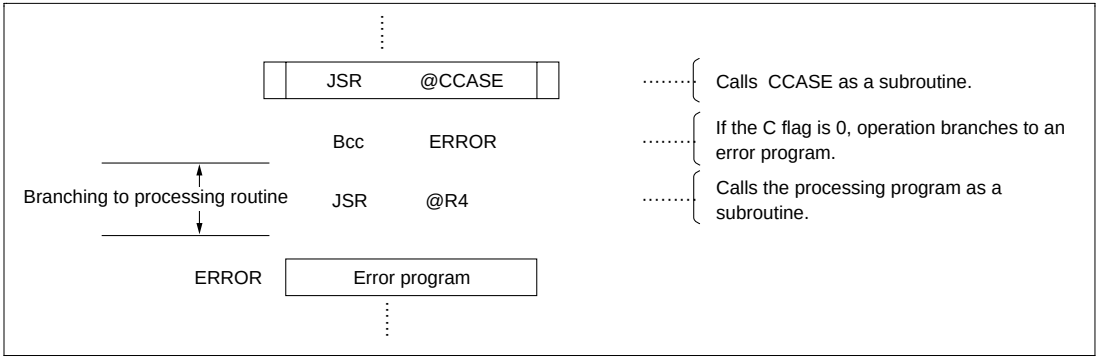
4. Example of use

Set commands and the start address of the data table in the arguments and call.the software CCASE as a subroutine.

```
WORK1     .RES.B    1                ..... Reserves a data memory area in which the
                                           user program places a command.
          :
          MOV. W    #DTABLE, R1      ..... Places in the input argument the start
                                           address of the data table set by the user
                                           program.
          MOV. B    @WORK1, R0L      ..... Places in the argument the command
                                           stored by the user program.
          JSR       @CCASE           ..... Calls the software CCASE as a subroutine.
          Bcc       ERROR            ..... Branches when there is no data that
                                           matches the command on the data table.
          Program to be branched to processing routine
          :
ERROR     Error program              ..... Executes error program.
          :
          .SECTION  D-TABLE,  DATA,  ALIGN=2
DTABLE    .DATA.W   H'0041           Command at "0A"
          .DATA.W   H'F000           Start address of processing routine for command at "0A"
          .DATA.W   H'0042           Command at "0B"
          .DATA.W   H'F100           Start address of processing routine for command at "0B"
          .DATA.W   H'0043           Command at "0C"
          .DATA.W   H'F200           Start address of processing routine for command at "0C2"
          :
          .DATA.W   H'0000           Delimiting data
```

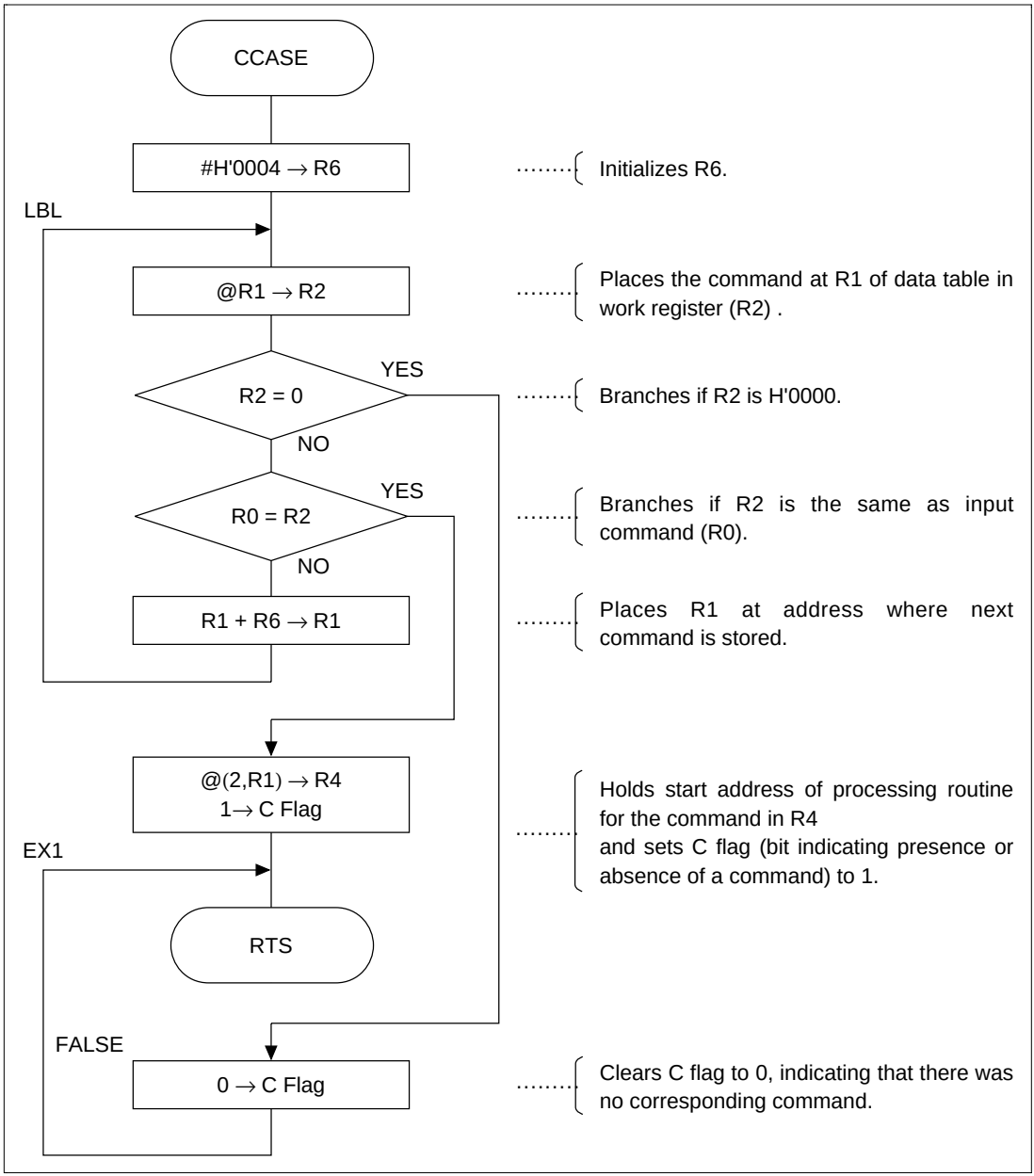
Note Example of program to be branched to processing routine

The software CCASE merely places the start address of a processing routine in R4. Branching to a processing routine requires the following program:



5. Operation
   a. R1 is used as the pointer that indicates the address of the data table.
   b. The commands are read sequentially from the start address of the data table in register indirect addressing mode. Then the contents of each command on the data table are compared with those of each input command (R0).
   c. When a command on the table matches R0, the start address of the processing routine placed at the address next to the command is set in R4. Then the C flag is set to 1 and the software CCASE terminates.
   d. When the command on the data table is H'0000, the C flag is cleared to 0 and the software CCASE terminates.

2.1.7 Flowchart



2.1.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:47:08
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :TABLE BRANCH (CCASE)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     COMMAND
    8                                   ;*                    R1     DATA TABLE START ADDRESS
    9                                   ;*
   10                                   ;*   RETURN          :R4     MODULE START ADDRESS
   11                                   ;*                    C bit of CCR    C=1;TRUE , C=0;FALSE
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 CCASE_co C 0000                        .SECTION        CCASE_code,CODE,ALIGN=2
   16                                        .EXPORT         CCASE
   17                                   ;
   18 CCASE_co C      00000000          CCASE        .EQU    $               ;Entry point
   19 CCASE_co C 0000 79060004               MOV.W   #H'0004,R6
   20 CCASE_co C 0004                   LBL
   21 CCASE_co C 0004 6912                   MOV.W   @R1,R2
   22 CCASE_co C 0006 4710                   BEQ     FALSE           ;If table "END" then exit
   23 CCASE_co C 0008 1D02                   CMP.W   R0,R2
   24 CCASE_co C 000A 4704                   BEQ     TRUE            ;Branch if command find
   25 CCASE_co C 000C 0961                   ADD.W   R6,R1           ;Increment table address
   26 CCASE_co C 000E 40F4                   BRA     LBL             ;Branch always
   27 CCASE_co C 0010                   TRUE
   28 CCASE_co C 0010 6F140002               MOV.W   @(H'2,R1),R4    ;Load module start address
   29 CCASE_co C 0014 0401                   ORC     #H'01,CCR       ;Set C flag for true
   30 CCASE_co C 0016                   EX1
   31 CCASE_co C 0016 5470                   RTS
   32 CCASE_co C 0018                   FALSE
   33 CCASE_co C 0018 06FE                   ANDC    #H'FE,CCR       ;Clear C flag for false
   34 CCASE_co C 001A 40FA                   BRA     EX1
   35                                   ;
   36                                        .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

Section 3 ASCII CODE PROCESSING

3.1 Change ASCII Code from Lowercase to Uppercase

MCU: H8/300 Series
H8/300L Series

Label name: TPR

3.1.1 Function

1. The software TPR changes a lowercase ASCII code to a corresponding uppercase ASCII code.
2. All data used with the software TPR is ASCII code.

3.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	Lowercase ASCII code	R0L	1
Output	Uppercase ASCII code	R0L	1

3.1.3 Internal Register and Flag Changes

R0H	R0L	R1	R2	R3	R4	R5	R6	R7
×	↕	•	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

3.1.4 Specifications

Program memory (bytes)
14
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
24
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

3.1.5 Description

1. Details of functions
   a. The following argument is used with the software TPR:
      R0L: Contains, as an input argument, a lowercase ASCII code. After execution of the software TPR, the corresponding uppercase ASCII code is placed in R0L.
   b. Figure 3.1 shows an example of the software TPR being executed. When the lowercase ASCII code 'a' (H'61) is set as shown in (1), it is converted to the uppercase ASCII code 'A' (H'41) , which then is placed in R0L as shown in (2).

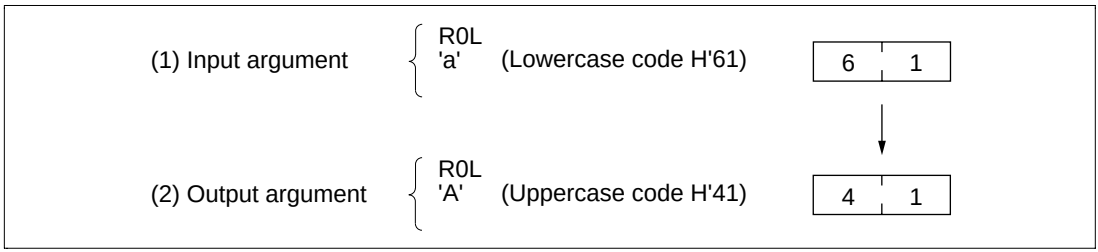


Figure 3.1 Example of Software TPR Execution

2. Notes on usage
   R0L must contain a lowercase ASCII code. If any other code is placed in R0L, the input data is retained in R0L.
3. Data memory
   The software TPR does not use the data memory.

4. Example of use

Set a lowercase ASCII code in the input argument and call the software TPR as a subroutine.

```
WORK1    .RES. B    1          ..... Reserves a data memory area in which the
                                      user program places a lowercase ASCII
                                      code.
WORK2    .RES. B    1          ..... Reserves a data memory area in which a
                                      corresponding uppercase ASCII code is
                                      placed in the user program.
         MOV. B     @WORK1, R0L ..... Places the lowercase ASCII code set by
                                      the user program in the input argument.
         JSR        @TPR        ..... Calls the software TPR as a subroutine.
         MOV. B     R0,  @WORK2 ..... Places the uppercase ASCII code set in
                                      the output argument in the data memory
                                      area of the user program.
```

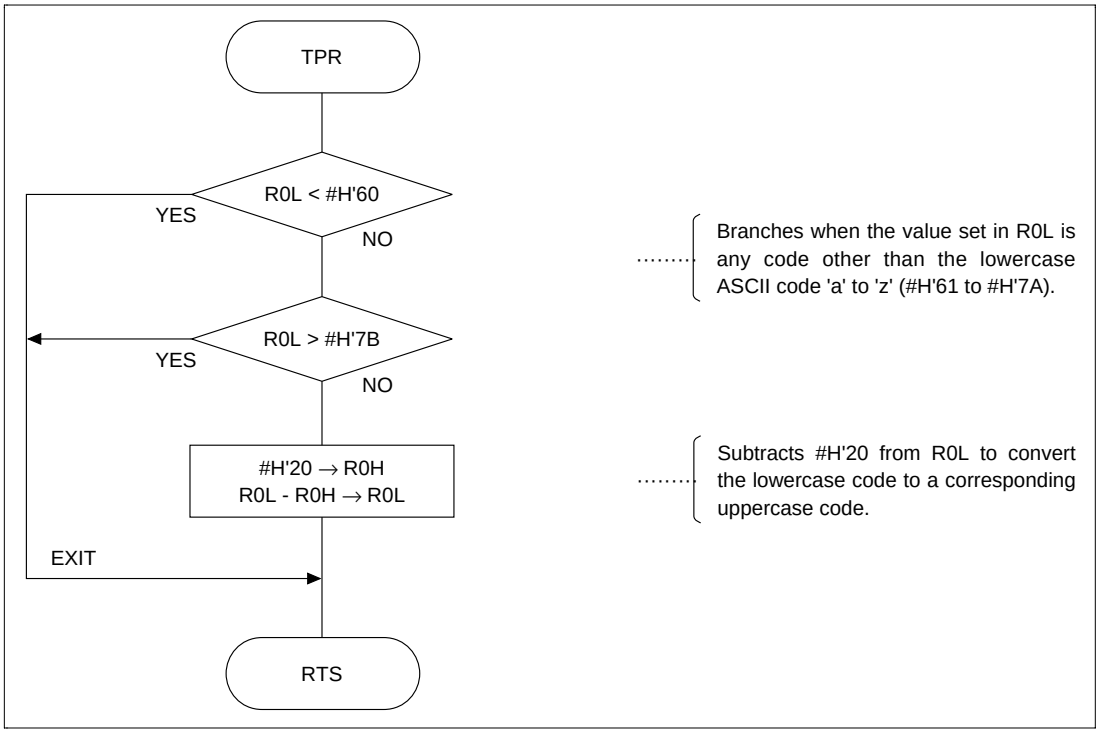
5. Operation

   a. compare instruction (CMP.B) is used to determine whether the input data set in R0L is a lowercase ASCII code.

   b. H'20 is subtracted from the lowercase ASCII code to obtain a corresponding uppercase ASCII code.

   c. If the input data is not a lowercase ASCII code, the program retains the input data and terminates processing.

3.1.6 Flowchart



3.1.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:47:44
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :CHANGE ASCII CODE LOWERCASE
    4                                   ;*                            TO UPPERCASE (TPR)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L    (ASCII CODE LOWERCASE)
    9                                   ;*
   10                                   ;*   RETURN          :R0L    (ASCII CODE UPPERCASE)
   11                                   ;*
   12                                   ;*********************************************************************
   13                                   ;
   14 TPR_code C 0000                        .SECTION        TPR_code,CODE,ALIGN=2
   15                                        .EXPORT         TPR
   16                                   ;
   17 TPR_code C      00000000       TPR  .EQU    $               ;Entry point
   18 TPR_code C 0000 A861                   CMP.B   #H'61,R0L
   19 TPR_code C 0002 4508                   BCS     EXIT            ;Branch if R0L<#H'60
   20 TPR_code C 0004 A87A                   CMP.B   #H'7A,R0L
   21 TPR_code C 0006 4204                   BHI     EXIT            ;Branch  if R0L>#H'7B
   22 TPR_code C 0008 F020                   MOV.B   #H'20,R0H
   23 TPR_code C 000A 1808                   SUB.B   R0H,R0L         ;Lowercase - #H'20 -> Uppercase
   24 TPR_code C 000C                   EXIT
   25 TPR_code C 000C 5470                   RTS
   26                                   ;
   27                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

3.2 Change an ASCII Code to a 1-Byte Hexadecimal Number

MCU: H8/300 Series
H8/300L Series

Label name: NIBBLE

3.2.1 Function

1. The software NIBBLE changes an ASCII code ('0' to '9' and 'A' to 'F') to a corresponding 1-byte hexadecimal number.
2. All data used with the software NIBBLE is ASCII code.

3.2.2 Arguments

Description		Memory area	Data length (bytes)
Input	ASCII code	R0L	1
Output	1-byte hexadecimal number	R0L	1
	Convert or not	C flag (CCR)	

3.2.3 Internal Register and Flag Changes

R0H	R0L	R1	R2	R3	R4	R5	R6	R7
•	↕	•	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

3.2.4 Specifications

Program memory (bytes)
24
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
38
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

3.2.5 Description

1. Details of functions
   a. The following arguments are used with the software NIBBLE:
      R0L: Contains, as an input argument, an ASCII code. After execution of the software NIBBLE, the corresponding 1-byte hexadecimal number is placed in R0L.
      C flag (CCR): Holds, as an output argument, the state of ASCII code after execution of the software NIBBLE.
      C flag = 1: The input ASCII code is any other than '0' to '9' or 'A' to 'F'.
      C flag = 0: The input ASCII code is '0' to '9' or 'A' to 'F'.
   b. Figure 3.2 shows an example of the software NIBBLE being executed. When the input argument is set as shown in (1), a corresponding 1-byte hexadecimal number (H'0F) is placed in R0L as shown in (2).

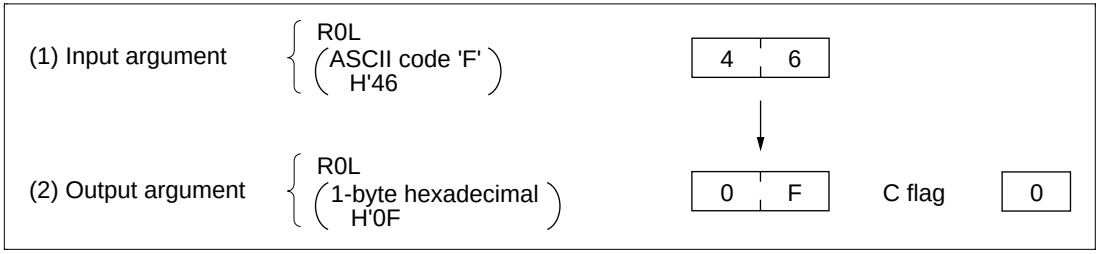
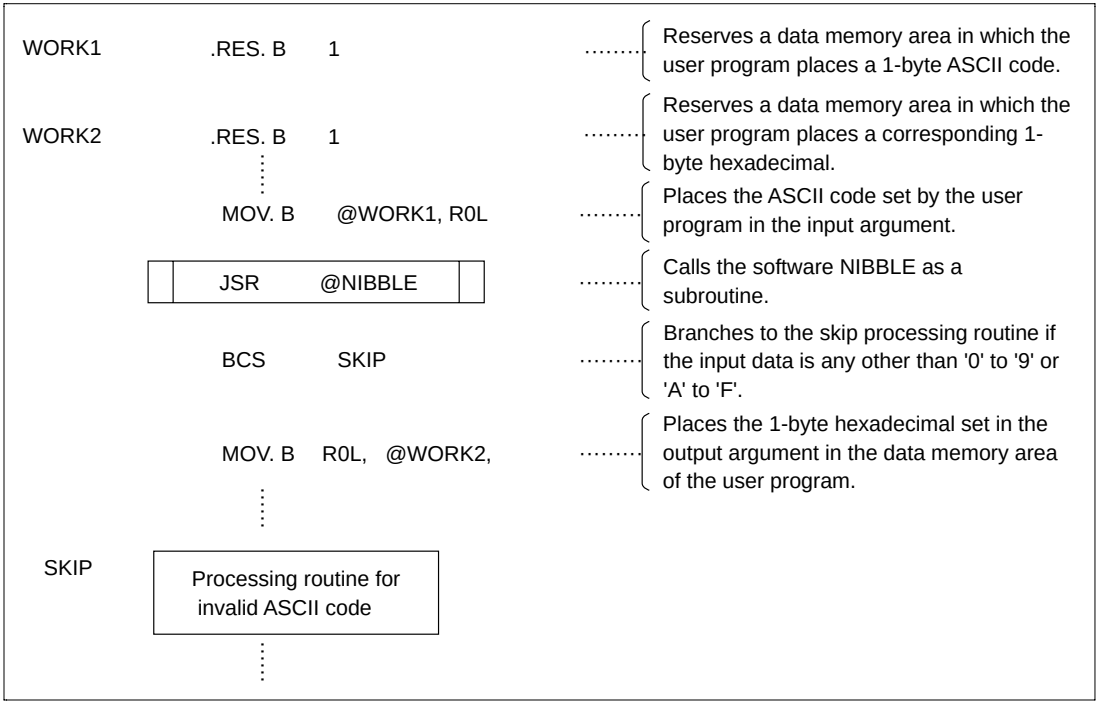


Figure 3.2 Example of Software NIBBLE Execution

2. Notes on usage
   If any data other than ASCII code '0' to '9' or 'A' to 'F' is set in R0L, the data is destroyed after execution of the software NIBBLE.
3. Data memory
   The software NIBBLE does not use the data memory.

4. Example of use

Set an ASCII code in the input argument and call the software NIBBLE as a subroutine.

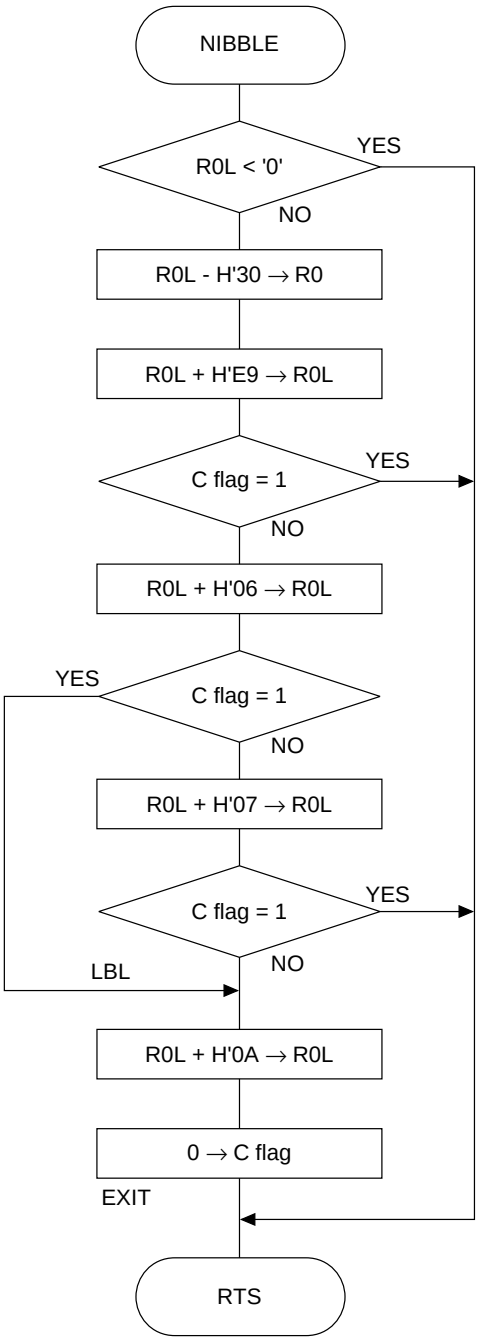


5. Operation
   a. On the basis of the status of the C flag showing the result of operations, the software NIBBLE determines whether the data set in R0L falls in the '0' to 'F' range of the ASCII code table ([ ] in table 3.1).
   b. The software further perform operations to delete the ':' to '@' range ([ ] in table 3.1).
   c. If the input data is outside the '0' to '9' and 'A' to 'F' ranges, 1 is set in the C flag in the processes a. and b..

Table 3.1 ASCII Code Table

MSD / LSD	0 0 0 0	1 0 0 1	2 0 1 0	3 0 1 1	4 1 0 0	5 1 0 1	6 1 1 0	7 1 1 1
0 0000	NUL	DLE	SP	0	@	P	`	p
1 0001	SOH	DC_1	!	1	A	Q	a	q
2 0010	STX	DC_2	"	2	B	R	b	r
3 0011	ETX	DC_3	#	3	C	S	c	s
4 0100	EOT	DC_4	$	4	D	T	d	t
5 0101	ENG	NAK	%	5	E	U	e	u
6 0110	ACK	SYN	&	6	F	V	f	v
7 0111	BEL	ETB	'	7	G	W	g	w
8 1000	BS	CAN	(	8	H	X	h	x
9 1001	HT	EM	)	9	I	Y	i	y
A 1010	LF	SUB	*	:	J	Z	j	z
B 1011	VT	ESC	+	;	K	[	k	{
C 1100	FF	FS	,	<	L	\	l	|
D 1101	CR	GS	-	=	M	]	m	}
E 1110	SO	RS	•	>	N	↑	n	~
F 1111	SI	VS	/	?	O	←	o	DEL

3.2.6 Flowchart



Branches to the processing routine if the input argument (R0L) is less than '0' of ASCII code.

Branches to the processing routine, with the C flag set, if R0L is not less than 'G' of ASCII code.

Branches to the processing routine, with the C flag set, if R0L is not less than 'A' of ASCII code.

Branches to the processing routine, with the C flag set, if R0L is between ':' and '@'.

Changes the value of R0L to a hexadecimal (H'00 to H'0F).

Clears the bit indicating the state of end (the C flag) to 0, indicating the input data has been changed.

3.2.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 20:08:15
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :CHANGE 1 BYTE ASCII CODE
    4                                   ;*                            TO 4 BIT HEXAGON (NIBBLE)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L             (1 BYTE ASCII CODE)
    9                                   ;*
   10                                   ;*   RETURN          :R0L             (4 BIT HEXADECIMAL)
   11                                   ;*                     C flag of CCR   (C=0;FALSE , C=1;TRUE)
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 NIBBLE_c C 0000                        .SECTION        NIBBLE_code,CODE,ALIGN=2
   16                                        .EXPORT         NIBBLE
   17                                   ;
   18 NIBBLE_c C 0000                   NIBBLE
   19 NIBBLE_c C 0000 F030                   MOV.B   #H'30,R0H
   20 NIBBLE_c C 0002 1808                   SUB.B   R0H,R0L         ;R0L - #H'30 -> R0L
   21 NIBBLE_c C 0004 4510                   BCS     EXIT            ;Branch if R0L<'0'
   22 NIBBLE_c C 0006 88E9                   ADD.B   #H'E9,R0L
   23 NIBBLE_c C 0008 450C                   BCS     EXIT            ;Branch if R0L<'F'
   24 NIBBLE_c C 000A 8806                   ADD.B   #H'06,R0L
   25 NIBBLE_c C 000C 4504                   BCS     LBL             ;Branch if R0L<=H'FF
   26 NIBBLE_c C 000E 8807                   ADD.B   #H'07,R0L
   27 NIBBLE_c C 0010 4504                   BCS     EXIT            ;Branch if R0L<=H'FF
   28 NIBBLE_c C 0012                   LBL
   29 NIBBLE_c C 0012 880A                   ADD.B   #H'0A,R0L       ;Change R0L to ASCII CODE
   30 NIBBLE_c C 0014 06FE                   ANDC    #H'FE,CCR       ;Clear C flag of CCR
   31 NIBBLE_c C 0016                   EXIT
   32 NIBBLE_c C 0016 5470                   RTS
   33                                   ;
   34                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

3.3 Change an 8-Bit Binary Number to a 2-Byte ASCII Code

MCU: H8/300 Series
H8/300L Series

Label name: COBYTE

3.3.1 Function

1. The software COBYTE changes an 8-bit binary number to a corresponding 2-byte ASCII code.
2. All data used with the software COBYTE is ASCII code.

3.3.2 Arguments

Description		Memory area	Data length (bytes)
Input	8-bit binary number	R0L	1
Output	2-byte ASCII code	R1	2

3.3.3 Internal Register and Flag Changes

R0H	R0L	R1	R2H	R2L	R3	R4	R5	R6	R7
×	×	↕	•	×	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

3.3.4 Specifications

Program memory (bytes)
38
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
72
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

3.3.5 Description

1. Details of functions
   a. The following arguments are used with the software COBYTE:
      R0L: Contains, as an input argument, an 8-bit binary number to be changed to a corresponding ASCII code.
      R1: Contains, as an output argument, 1-byte ASCII code data in the upper 4 bits and the lower 4 bits each of the 8-bit binary number.
   b. Figure 8-1 shows an example of the software COBYTE being executed. When the input argument is set as shown in á@, 2-byte ASCII code data is placed in R1 as shown in áA.

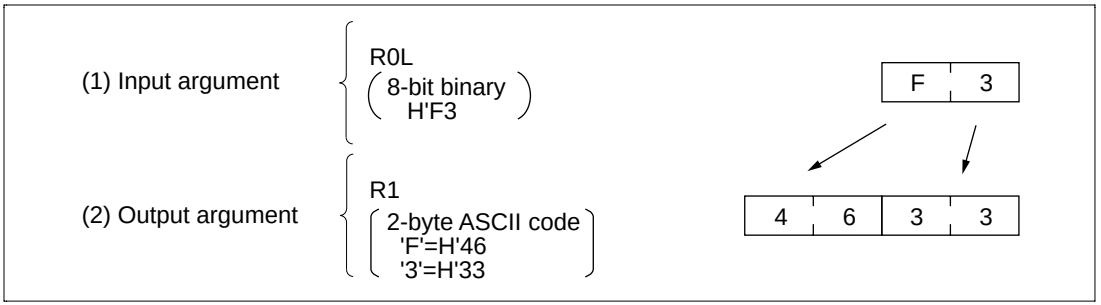


Figure 3.3 Example of Software COBYTE Execution

2. Notes on usage

   The 8-bit binary number set in R0L is destroyed after execution of the software COBYTE.

3. Data memory

   The software COBYTE does not use the data memory.

4. Example of use

   Set an 8-bit binary number in the input argument and call the software COBYTE as a subroutine.

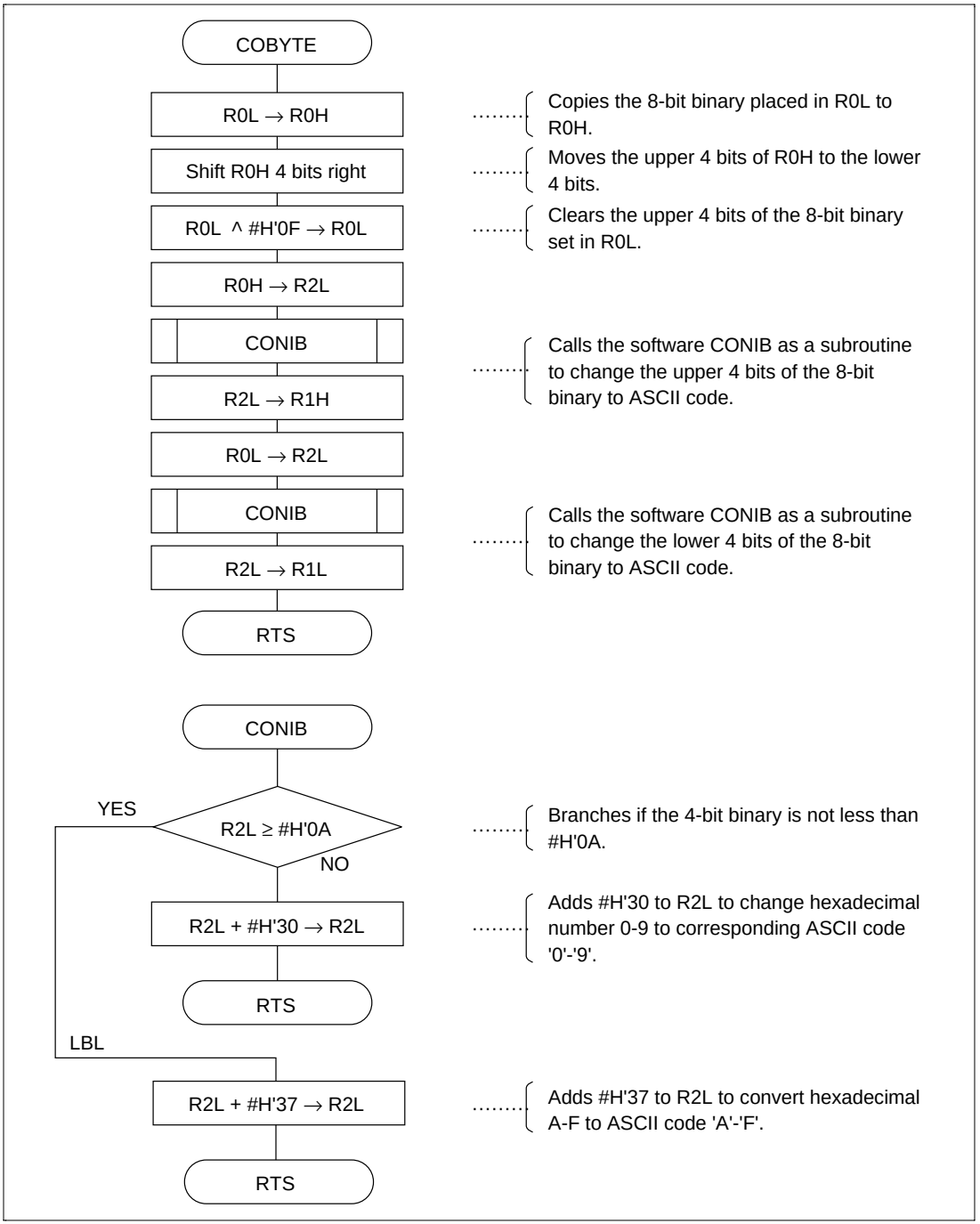
```
WORK1    .RES. B    1          ..... Reserves a data memory area in which the
                                      user program places an 8-bit binary
                                      number.
         .ALIGN
WORK2    .RES. W    1          ..... Reserves a data memory area in which the
                                      user program places a corresponding 2-
                                      byte ASCII code.
          :
         MOV. B     @WORK1, R0L ..... Places the 8-bit binary number set by the
                                      user program in R0L.
         JSR        @COBYTE     ..... Calls the software COBYTE as a
                                      subroutine.
         MOV. W     R1, @WORK2  ..... Places the 2-byte ASCII code set in the
                                      output argument in the data memory area
                                      of the user program.
```

5. Operation
   a. The 8-bit binary number is separated into two bit groups, the upper 4 bits and the lower 4 bits.
   b. A compare instruction is used to determine whether the data (the upper 4 bits + the lower 4 bits) is in the H'00 to H'09 range ([ ] in table 3.2) or in the H'0A to H'0F range ([ ] in table 3.2). H'30 is added to the data if it falls in the H'00 to H'09 range and H'37 to the data if it falls in the H'0A to H'0F range for change to a corresponding ASCII code.

Table 3.2 ASCII Code Table

MSD / LSD	0 0 0 0	1 0 0 1	2 0 1 0	3 0 1 1	4 1 0 0	5 1 0 1	6 1 1 0	7 1 1 1
0 0000	NUL	DLE	SP	0	@	P	`	p
1 0001	SOH	DC1	!	1	A	Q	a	q
2 0010	STX	DC2	"	2	B	R	b	r
3 0011	ETX	DC3	#	3	C	S	c	s
4 0100	EOT	DC4	$	4	D	T	d	t
5 0101	ENG	NAK	%	5	E	U	e	u
6 0110	ACK	SYN	&	6	F	V	f	v
7 0111	BEL	ETB	'	7	G	W	g	w
8 1000	BS	CAN	(	8	H	X	h	x
9 1001	HT	EM	)	9	I	Y	i	y
A 1010	LF	SUB	*	:	J	Z	j	z
B 1011	VT	ESC	+	;	K	[	k	{
C 1100	FF	FS	,	<	L	\	l	|
D 1101	CR	GS	-	=	M	]	m	}
E 1110	SO	RS	•	>	N	↑	n	~
F 1111	SI	VS	/	?	O	←	o	DEL

3.3.6 Flowchart



3.3.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:49:53
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :CHANGE 1 BYTE HEXADECIMAL
    4                                   ;*                            TO 2 BYTE ASCII CODE (COBYTE)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L    (1 BYTE HEXADECIMAL)
    9                                   ;*
   10                                   ;*   RETURN          :R1     (2 BYTE ASCII CODE)
   11                                   ;*
   12                                   ;*********************************************************************
   13                                   ;
   14 COBYTE_c C 0000                        .SECTION        COBYTE_code,CODE,ALIGN=2
   15                                        .EXPORT         COBYTE
   16                                   ;
   17 COBYTE_c C      00000000           COBYTE       .EQU    $               ;Entry Point
   18 COBYTE_c C 0000 0C80                   MOV.B   R0L,R0H
   19                                   ;
   20 COBYTE_c C 0002 1100                   SHLR    R0H
   21 COBYTE_c C 0004 1100                   SHLR    R0H
   22 COBYTE_c C 0006 1100                   SHLR    R0H
   23 COBYTE_c C 0008 1100                   SHLR    R0H             ;Select upper 4 bit hexadecimal(R0H)
   24                                   ;
   25 COBYTE_c C 000A E80F                   AND.B   #H'0F,R0L       ;Select lower 4 bit hexadecimal(R0L)
   26                                   ;
   27 COBYTE_c C 000C 0C0A                   MOV.B   R0H,R2L
   28 COBYTE_c C 000E 550A                   BSR     CONIB           ;Branch subroutine CONIB
   29 COBYTE_c C 0010 0CA1                   MOV.B   R2L,R1H         ;Set 1st ASCII code to R1H
   30                                   ;
   31 COBYTE_c C 0012 0C8A                   MOV.B   R0L,R2L
   32 COBYTE_c C 0014 5504                   BSR     CONIB           ;Branch subroutine CONIB
   33 COBYTE_c C 0016 0CA9                   MOV.B   R2L,R1L         ;Set 2nd ASCII code to R1L
   34                                   ;
   35 COBYTE_c C 0018 5470                   RTS
   36                                   ;---------------------------------------------------------------------
   37 COBYTE_c C 001A                   CONIB                               ;Change R2L to ASCII code
   38 COBYTE_c C 001A AA0A                   CMP.B   #H'0A,R2L
   39 COBYTE_c C 001C 4404                   BCC     LBL             ;Branch if R2L ASCII "A"-"F"
   40 COBYTE_c C 001E 8A30                   ADD.B   #H'30,R2L       ;Reshape R2L ASCII "0"- "9"
   41 COBYTE_c C 0020 5470                   RTS
   42 COBYTE_c C 0022                   LBL
   43 COBYTE_c C 0022 8A37                   ADD.B   #H'37,R2L
   44 COBYTE_c C 0024 5470                   RTS
   45                                   ;
   46                                        .END
*****TOTAL ERRORS       0

*****TOTAL WARNINGS     0
```

Section 4 BIT PROCESSING

4.1 Count the Number of Logic 1 Bits in 8-Bit Data (HCNT)

MCU: H8/300 Series
H8/300L Series

Label name: HCNT

4.1.1 Function

1. The software HCNT counts how many logic-1 bits exist in 8 bits of data.
2. This function is useful in performing parity check.

4.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	8-bit data	R0L	1
Output	Number of logic-1 bits	R1L	1

4.1.3 Internal Register and Flag Changes

R0H	R0L	R1H	R1L	R2	R3	R4	R5	R6	R7
•	×	×	↕	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	•	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

4.1.4 Specifications

Program memory (bytes)
18
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
162
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

4.1.5 Notes

The specified clock cycle count (162) is for 8-bit data = "FF".

4.1.6 Description

1. Details of functions
   a. The following arguments are used with the software HCNT:
      R0L: Contains, as an input argument, 8-bit data that may have logic-1 bits to be counted.
      R1: Contains, as an output argument, the number of logic-1 bits that have been found and counted in the 8-bit data.
   b. Figure 4.1 shows an example of the software HCNT being executed. When the input argument is set as shown in (1), the number of logic-1 bits that have been found in the 8-bit data is placed in R1L as shown in (2).
   c. The contents of ROL are retained after execution of the software HCNT.

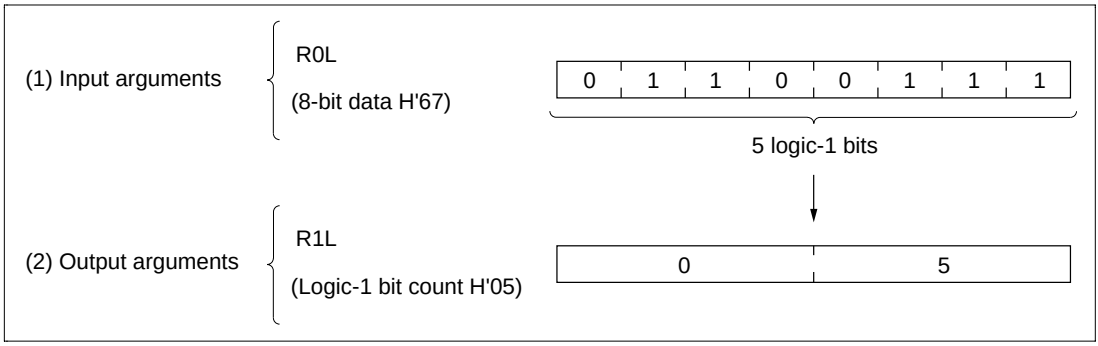


Figure 4.1 Example of Software HCNT Execution

2. Notes on usage

   To count the logic-0 bits, complement the logic 1 in R0L (by using the NOT instruction) before executing the software HCNT.

3. Data memory

   The software HCNT does not use the data memory.

4. Example of use

   Set 8-bit data in the input argument and call the software HCNT as a subroutine.

```
WORK1       . RES. B    1       ..... Reserves a data memory area in which the
                                      user program places 8-bit data.
            MOV. B      @WORK1, R0L ..... Places in the input argument the 8-bit data
                                          set by the user program.
            JSR         @HCNT       ..... Calls the software HCNT as a subroutine.
```

5. Operation

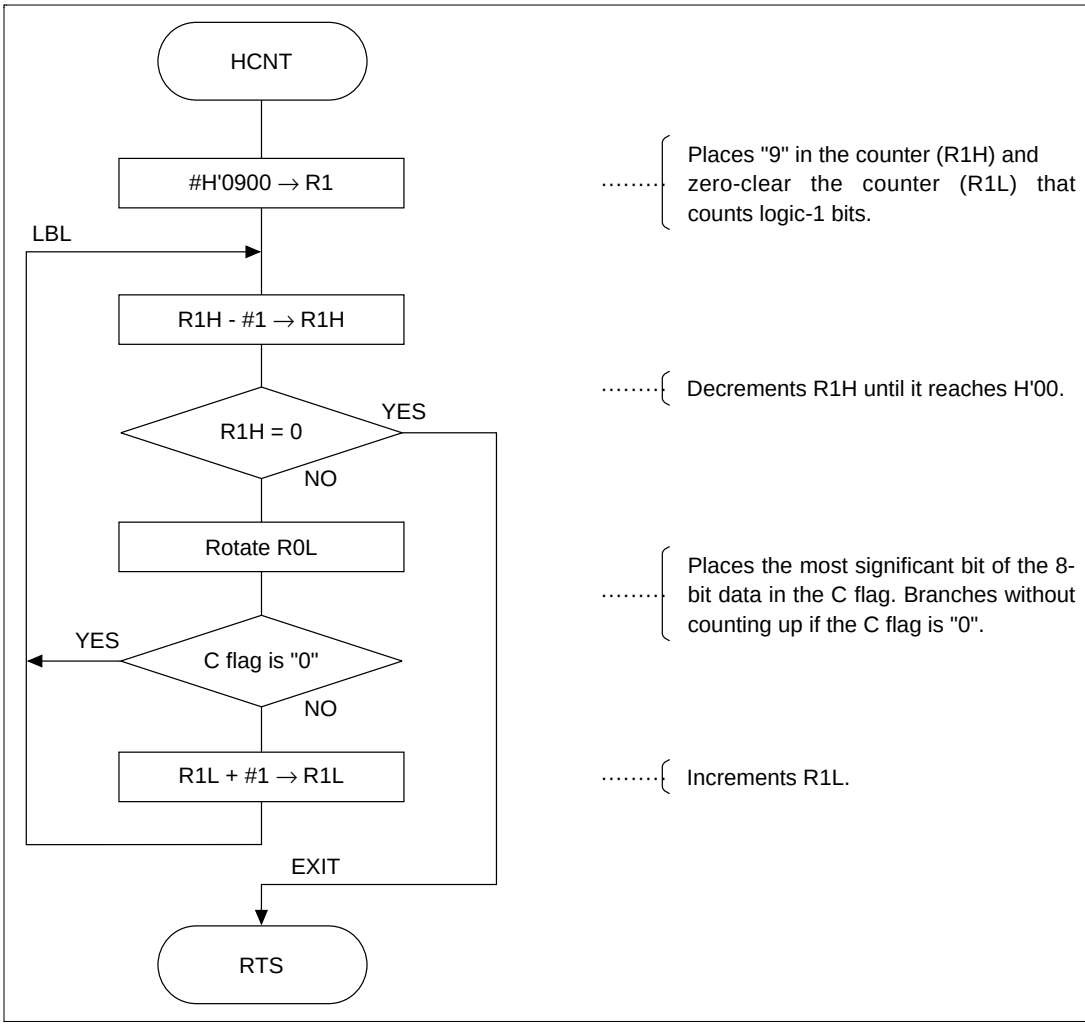
   a. R1H is used as the counter that indicates an 8-bit data rotation count.

   b. The ROTXL instruction is used to set data (R0L) bit by bit in the C flag.

   c. R1L is incremented when the C flag is 1. No operation occurs when the C flag is 0.

   d. R1H is decremented each time the b.-c. process is performed. The process is repeated until R1H reaches 0.

4.1.7 Flowchart



4.1.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:51:00
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :HIGH LEVEL BIT COUNT (HCNT)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0L    (8 BIT DATA)
    8                                   ;*
    9                                   ;*   RETURN          :R1L    (HIGH LEVEL BIT COUNTER)
   10                                   ;*
   11                                   ;*********************************************************************
   12                                   ;
   13 HCNT_cod C 0000                        .SECTION        HCNT_code,CODE,ALIGN=2
   14                                        .EXPORT         HCNT
   15                                   ;
   16 HCNT_cod C      00000000          HCNT .EQU    $       ;Entry point
   17 HCNT_cod C 0000 79010900               MOV.W   #H'0900,R1
   18 HCNT_cod C 0004                   LBL
   19 HCNT_cod C 0004 1A01                   DEC     R1H
   20 HCNT_cod C 0006 4708                   BEQ     EXIT    ;If R1H = 0 then exit
   21 HCNT_cod C 0008 1288                   ROTL    R0L
   22 HCNT_cod C 000A 44F8                   BCC     LBL     ;Branch if C flag = 0
   23 HCNT_cod C 000C 0A09                   INC     R1L
   24 HCNT_cod C 000E 40F4                   BRA     LBL     ;Branch always
   25 HCNT_cod C 0010                   EXIT
   26 HCNT_cod C 0010 5470                   RTS
   27                                   ;
   28                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

4.2 Shift 16-Bit Binary to Right (SHR)

MCU: H8/300 Series
H8/300L Series

Label name: SHR

4.2.1 Function

1. The software SHR shifts a 16-bit binary number to the right.
2. Shift count: 1 to 16.
3 This function is useful in multiplying a 16-bit binary number by 2-n (n=shift count).

4.2.2 Arguments

Description		Memory area	Data length (bytes)
Input	16-bit binary to be shifted right	R0	2
	Shift count	R1L	1
Output	Result of shift	R0	2

4.2.3 Internal Register and Flag Changes

R0	R1H	R1L	R2	R3	R4	R5	R6	R7
↕	•	×	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	•	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

4.2.4 Specifications

Program memory (bytes)
10
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
168
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

4.2.5 Notes

The specified clock cycle count (162) is for shifting right 16 bits.

4.2.6 Description

1. Details of functions
   a. The following arguments are used with the software SHR:
      R0: Contains, as an input argument, a 16-bit binary number to be right-shifted. The result of shift is placed in R0 after execution of the software SHR.
      R1L: Contains, as an input argument, the right-shift count of a 16-bit binary number.
   b. Figure 4.2 shows an example of the software SHR being executed. When the arguments are set as shown in (1), the 16-bit binary number is shifted right as shown in (2). 0's are placed in the remaining upper bits.

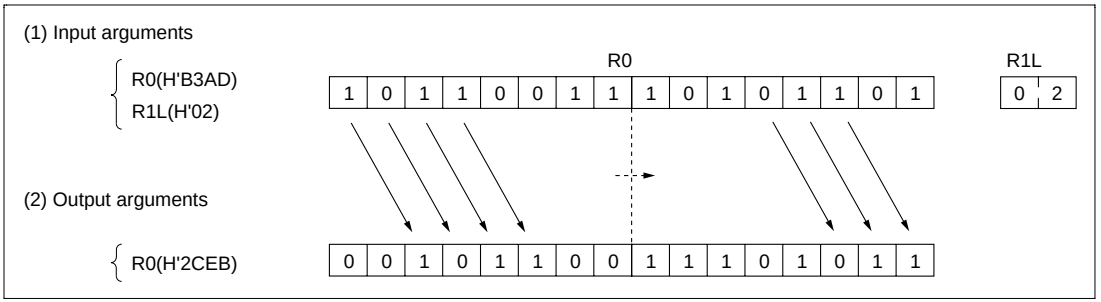


Figure 4.2 Example of Software SHR Execution

2. Notes on usage

   R1L must satisfy the condition H'01≤R1L≤H'0F; otherwise, R0 contains all 0's.

3. Data memory

   The software SHR does not use the data memory.

4. Example of use

   Set a 16-bit binary number and a shift count in the input arguments and call the software SHR as a subroutine.

```
WORK1   .RES. W   1          ..... Reserves a data memory area in which the
                                   user program places the 16-bit binary
                                   number.
WORK2   .RES. B   1          ..... Reserves a data memory area in which the
                                   user program places the 16-bit binary
                                   number.
        MOV. W    @WORK1, R0 ..... Places the 16-bit binary number set by the
                                   user program in the input argument.
        MOV. B    @WORK2, R1L ..... Places the shift count set by the user
                                   program in the input argument.
        JSR       @SHR       ..... Calls the software SHR as a subroutine.
```

5. Operation

a. The upper 8 bits of a 16-bit binary number is shifted right and the least significant bit in the C flag. Then the lower 8 bits are rotated right. This causes the least significant bit (in the C flag) moves to the most significant bit of the lower 8 bits.

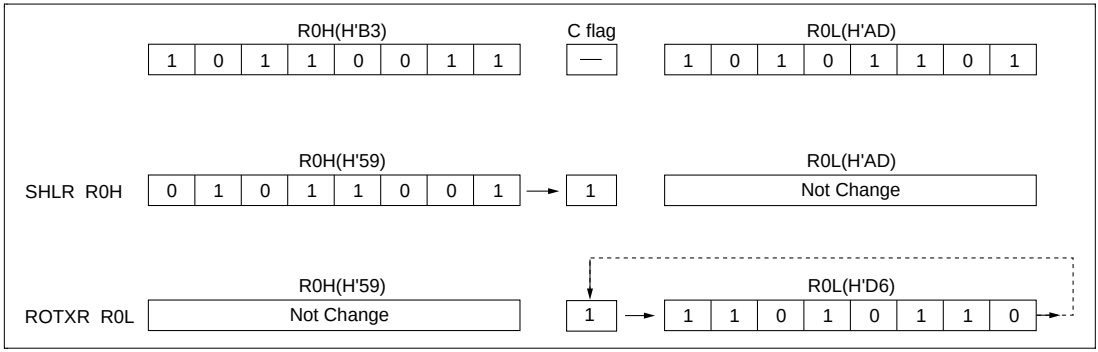
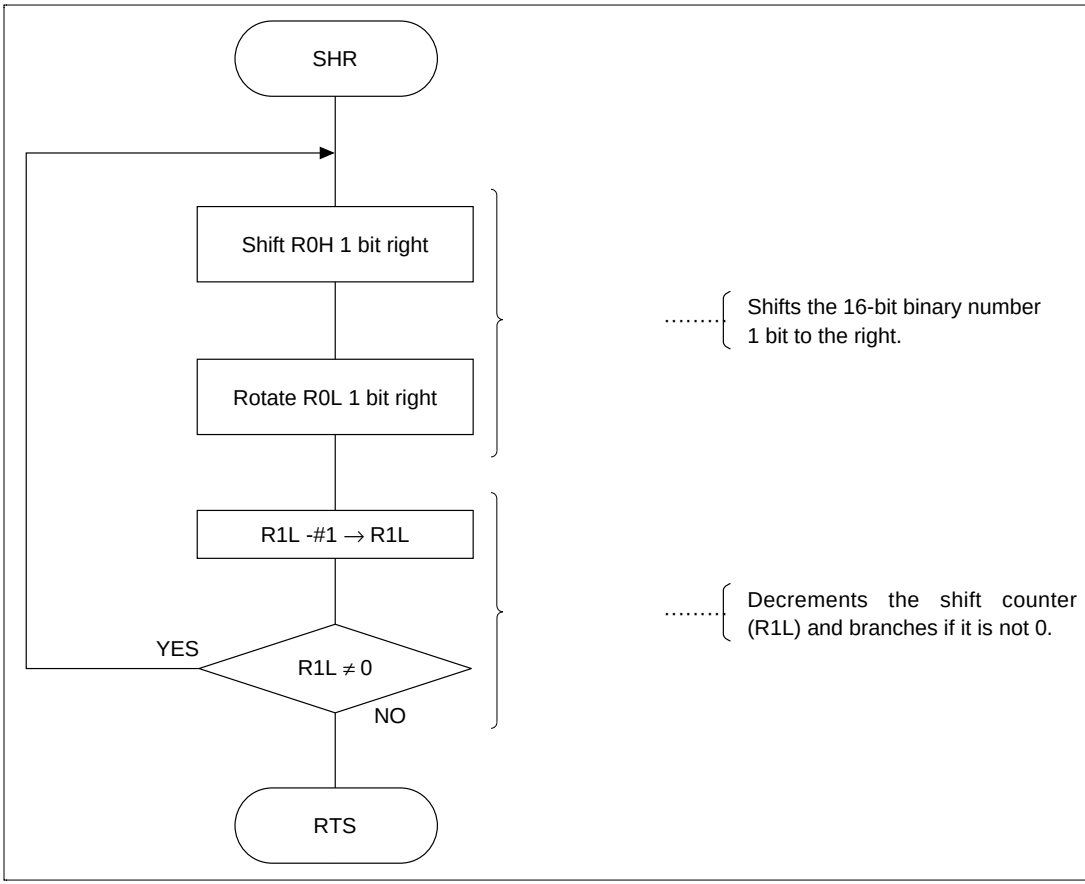


Figure 4.3 Example of Register Changes

b. R1L is used as the counter that indicates the shift count.

R1L is decremented each time the process a. is executed. This process is repeated until R1L reaches 0.

4.2.7 Flowchart



4.2.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:51:29
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME       :SHIFT OF 16 BIT DATA (SHR)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     (16 BIT BINARY DATA)
    8                                   ;*                    R1L    (SHIFT COUNTER)
    9                                   ;*
   10                                   ;*   RETURN          :R0     (16 BIT BINARY DATA)
   11                                   ;*
   12                                   ;*********************************************************************
   13                                   ;
   14 SHR_code C 0000                        .SECTION        SHR_code,CODE,ALIGN=2
   15                                        .EXPORT         SHR
   16                                   ;
   17 SHR_code C      00000000       SHR  .EQU    $       ;Entry point
   18 SHR_code C 0000 1100                SHLR    R0H     ;Shift 16 bit binary 1 bit right
   19 SHR_code C 0002 1308                ROTXR   R0L
   20 SHR_code C 0004 1A09                DEC     R1L     ;Decrement Shift counter
   21 SHR_code C 0006 46F8                BNE     SHR     ;Branch if not R1L=0
   22 SHR_code C 0008 5470                RTS
   23                                   ;
   24                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

Section 5 COUNTER

5.1 4-Digit Decimal Counter

MCU: H8/300 Series
H8/300L Series

Label name: DECNT

5.1.1 Function

1. The software DCNT increments a 4-digit binary-coded decimal (BCD) counter by 1.
2. This function is useful in counting interrupts (external interrupts, timer interrupts, etc.) .

5.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	—	—	—
Output	4-digit BCD counter	DCNTR (RAM)	2
	Counter overflow	C flag (CCR)	1

5.1.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	•	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

5.1.4 Specifications

Program memory (bytes)
18
Data memory (bytes)
2
Stack (bytes)
0
Clock cycle count
28
Reentrant
Impossible
Relocation
Possible
Interrupt
Possible

5.1.5 Description

1. Details of functions
   a. The following arguments are used with the software DECNT:

      DCNT: Used as a 4-digit BCD counter that is incremented by 1 each time the software DECNT is executed.

      C flag (CCR): Indicates the state of the counter after execution of the software DECNT.

      C flag = 1: The counter overflowed. (See figure 5.2.)

      C flag = 0: The counter was incremented normally.

   b. Figure 5.1 shows an example of the software DECNT being executed. When the software DECNT is executed, the 4-digit BCD counter is incremented as shown in (2).

2. Notes on usage

Figure 5.2 Example of Software DECNT Execution

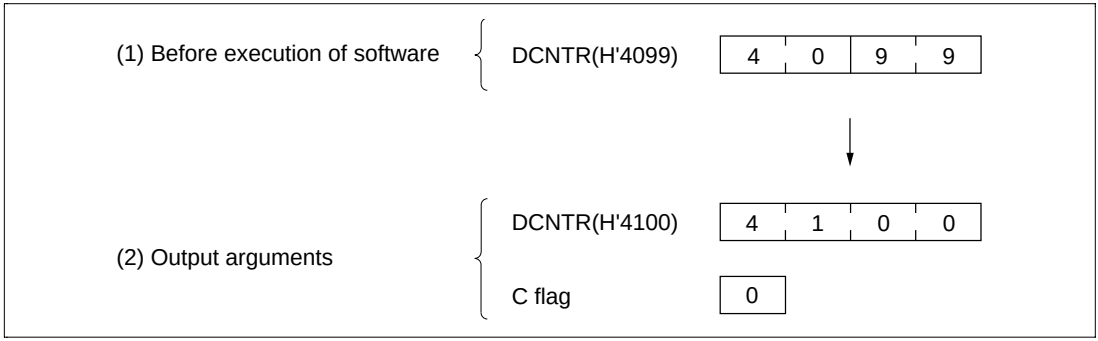


Figure 5.1 Example of Software DECNT Execution

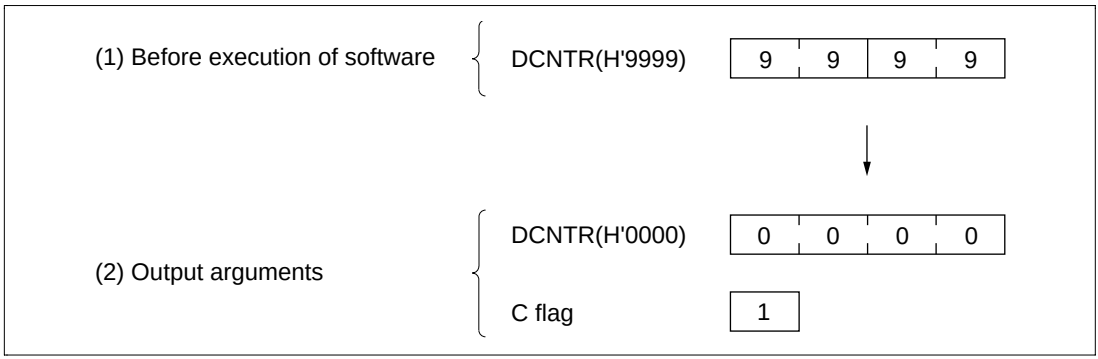
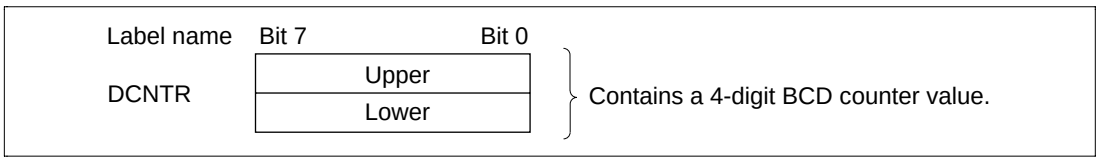
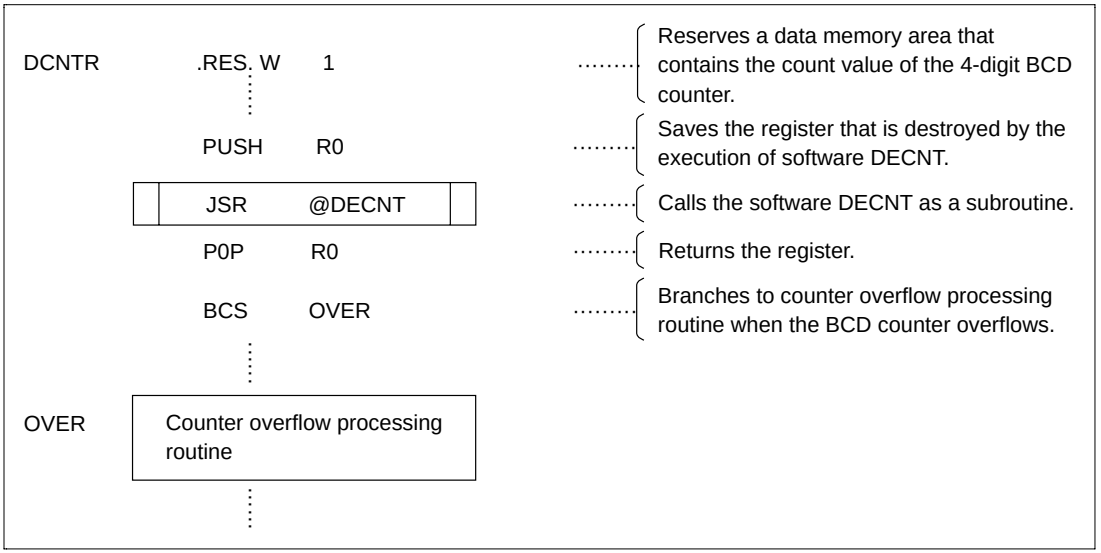


Figure 5.2 Example of Software DECNT Execution

3. Data memory

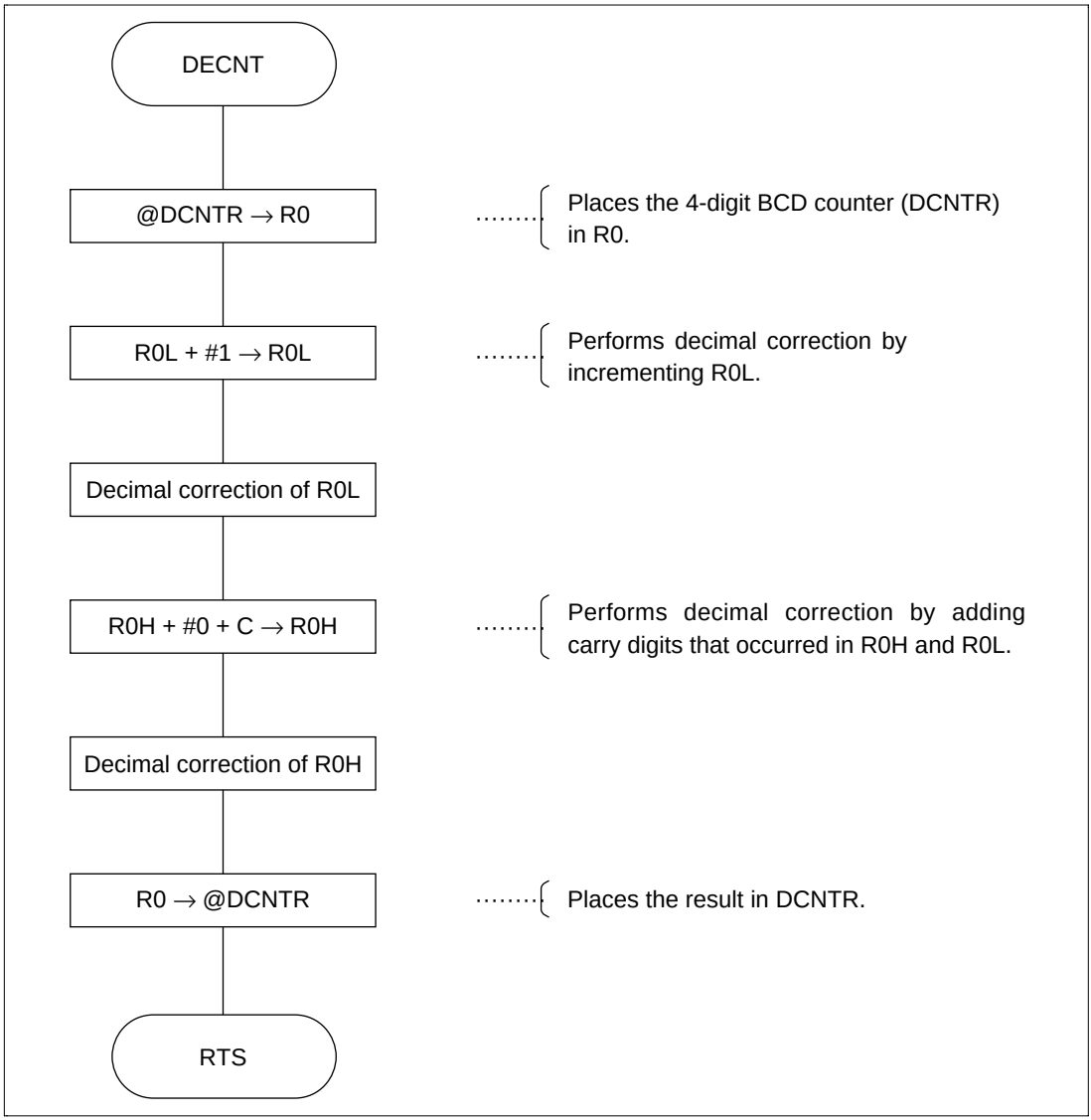


4. Example of use



5. Operation
   a. The software DECNT uses data memory (DCNTR) as a 4-digit BCD counter.
   b. Each time the software DECNT is executed as a subroutine, DCNTR is incremented to repeat decimal correction.

5.1.6 Flowchart



5.1.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:52:01
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00-NAME         :4 FIGURE BCD COUNTER(DECNT)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :NOTHING
    8                                   ;*
    9                                   ;*   RETURNS         :DCNTR  (BCD COUNTER)
   10                                   ;*                    C flag OF CCR (C=0 IS TRUE/C=1 IS OVER FLOW)
   11                                   ;*
   12                                   ;*********************************************************************
   13                                   ;
   14 DECNT_co C 0000                        .SECTION        DECNT_code,CODE,ALIGN=2
   15                                        .EXPORT         DECNT
   16                                   ;
   17 DECNT_co C      00000000           DECNT               .EQU    $       ;Entry point
   18 DECNT_co C 0000 6B000000               MOV.W   @DCNTR,R0       ;Load DCNTR to R0
   19 DECNT_co C 0004 8801                   ADD.B   #H'01,R0L       ;R0L + #H'01 -> R0L
   20 DECNT_co C 0006 0F08                   DAA     R0L             ;Decimal adjust R0L
   21 DECNT_co C 0008 9000                   ADDX.B  #H'00,R0H       ;R0H + #H'00 + C -> R0H
   22 DECNT_co C 000A 0F00                   DAA     R0H             ;Decimal adjust R0H
   23 DECNT_co C 000C 6B800000               MOV.W   R0,@DCNTR       ;Store R0 to DCNTR
   24 DECNT_co C 0010 5470                   RTS
   25                                   ;
   26                                   ;---------------------------------------------------------------------
   27                                   ;
   28 DATA_dat D 0000                        .SECTION        DATA_data,DATA,ALIGN=2
   29                                        .EXPORT         DCNTR
   30                                   ;
   31 DATA_dat D 0000 0000              DCNTR        .DATA.W         H'0000
   32                                   ;
   33                                   ;---------------------------------------------------------------------
   34                                   ;
   35                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

Section 6 COMPARISO

6.1 Compare 32-Bit Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: COMP

6.1.1 Function

1. The software COMP compares two 32-bit binary numbers and sets the result (>, =, <) in the C and Z flags (CCR).
2. All arguments are unsigned integers.

6.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	Comparand	R0, R1	4
	Source number	R2, R3	4
Output	Result of comparison	C flag, Z flag (CCR)	

6.1.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
•	•	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	↕	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

6.1.4 Specifications

Program memory (bytes)
8
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
16
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

6.1.5 Description

1. Details of functions
   a. The following arguments are used with the software COMP:
      R0, R1: Contain a 32-bit binary comparand as input arguments. (See figure 6.1.)
      R2, R3: Contain a source 32-bit binary number as input arguments. (See figure 6.1.)

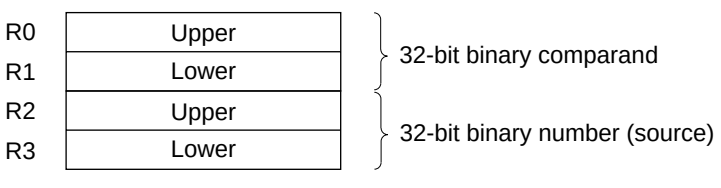


Figure 6.1 Input Argument Setting

b. Table 6.1 shows an example of the software COMP being executed.
The C and Z flags are set according to the input arguments.

Table 6.1 Example of Software COMP Execution

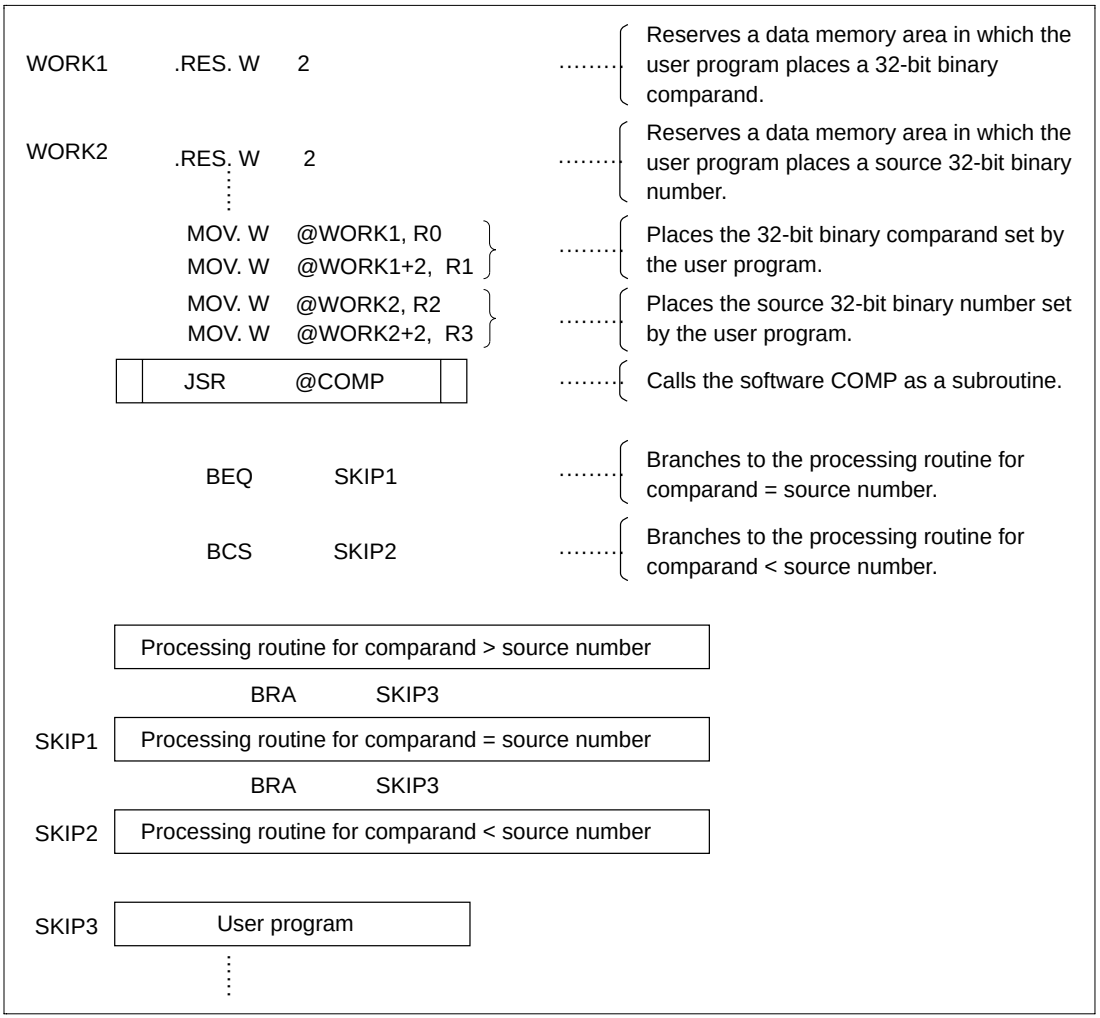
Input arguments					**Output arguments**	
Comparand		Large	Sign		CCR	
R0	R1	Small	R2	R3	C flag	Z flag
F67D	2001	<	2200	4001	0	0
2010	2020	=	2010	2020	0	1
4001	F000	>	A000	BB00	1	0

c. The input arguments are stored even after execution of the software COMP.

2. Notes on usage
When not using upper bits, set 0's in them; otherwise, no correct result of comparison can be obtained because comparison is made on the numbers including indeterminate data set in the upper bits.
3. Data memory
The software COMP does not use the data memory.

4. Example of use

Set a source binary number and a comparand in the input arguments and call the software COMP as a subroutine.



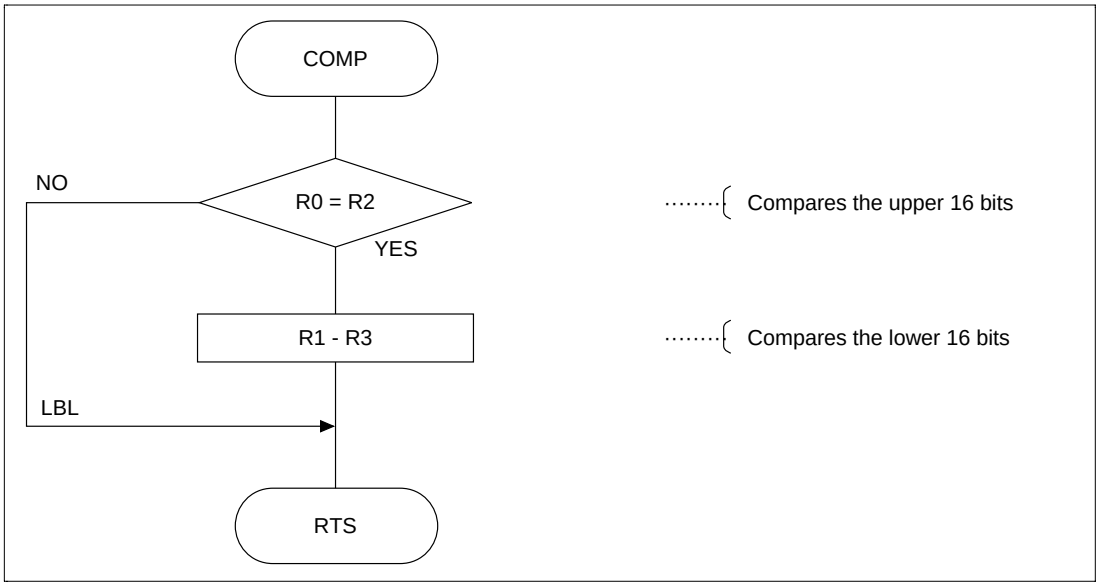
5. Operation

a. Comparison of two or more words can be done by performing a series of 1-word comparisons.

b. The output arguments are the C and Z flags after execution of the compare instruction (CMP.W).

c. The upper words are compared by using the word compare instruction (CMP.W). If the upper words are not equal, the software COMP terminates. If the upper words are equal, then the lower words are compared.

6.1.6 Flowchart



6.1.7 Program List

```
 *** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:52:34
PROGRAM NAME =

   1                                ;*********************************************************************
   2                                ;*
   3                                ;*   00 - NAME              :32 BIT COMPARISON (COMP)
   4                                ;*
   5                                ;*********************************************************************
   6                                ;*
   7                                ;*   ENTRY           :R0     (COMPARAND DATA HIGH)
   8                                ;*                    R1     (COMPARAND DATA LOW)
   9                                ;*                    R2     (COMPARATIVE DATA HIGH)
  10                                ;*                    R3     (COMPARATIVE DATA LOW)
  11                                ;*
  12                                ;*   RETURNS         :C flag & Z flag (COMPARISON RESULT)
  13                                ;*
  14                                ;*********************************************************************
  15                                ;
  16 COMP_cod C 0000                     .SECTION        COMP_code,CODE,ALIGN=2
  17                                     .EXPORT         COMP
  18                                ;
  19 COMP_cod C      00000000       COMP .EQU    $              ;Entry point
  20 COMP_cod C 0000 1D20                CMP.W   R2,R0
  21 COMP_cod C 0002 4602                BNE     LBL            ;Branch if Z=0
  22 COMP_cod C 0004 1D31                CMP.W   R3,R1
  23 COMP_cod C 0006                LBL
  24 COMP_cod C 0006 5470                RTS
  25                                ;
  26                                     .END
 *****TOTAL ERRORS       0

 *****TOTAL WARNINGS     0
```

Section 7 ARITHMETIC OPERATION

7.1 Addition of 32-Bit Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: ADD1

7.1.1 Function

1. The software ADD1 adds a 32-bit binary number to another 32-bit binary number and places the result (a 32-bit binary number) in a general-purpose register.
2. The arguments used with the software ADD1 are unsigned integers.
3. All data is manipulated on general-purpose registers.

7.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	Augend	R0, R1	4
	Addend	R2, R3	4
Output	Result of addition	RO, R1	4
	Carry	C flag (CCR)	

7.1.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.1.4 Specifications

Program memory (bytes)
8
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
14
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.1.5 Description

1. Details of functions
   a. The following arguments are used with the software ADD1:

      R0, R1: Contain a 32-bit binary augend as an input argument. The result of addition is placed in these registers after execution of the software ADD1.

      R2, R3: Contain a 32-bit binary addend as an input argument.

      C flag (CCR): Determines the presence or absence of a carry as an output argument after execution of the software ADD1.

      C flag = 1: A carry occurred in the result. (See figure 7.1)

      C flag = 0: No carry occurred in the result.

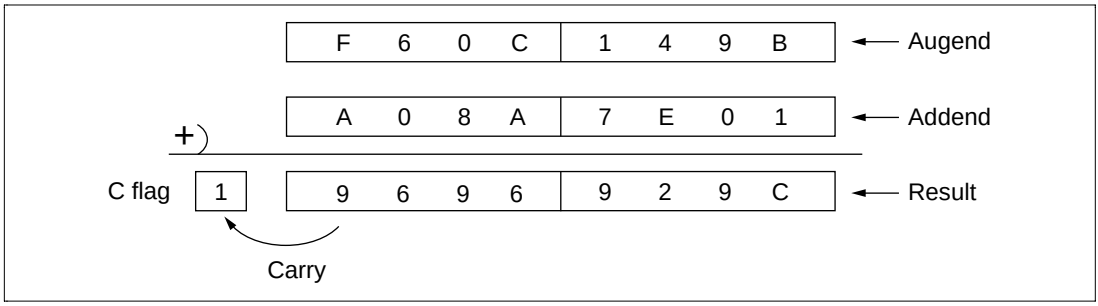


Figure 7.1 Example of Addition with a Carry

b. Figure 7.2 shows an example of the software ADD1 being executed. When the input arguments are set as shown in (1), the result of addition is placed in R0 and R1 as shown in (2).

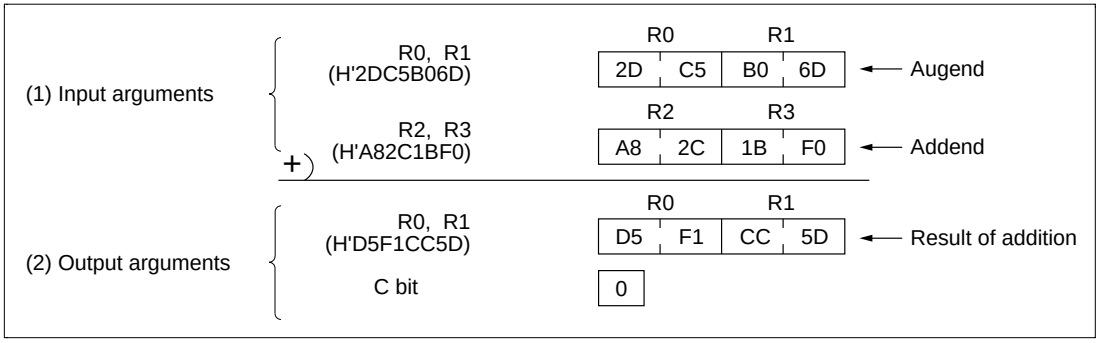


Figure 7.2 Example of Software ADD1 Execution

2. Notes on usage

   When upper bits are not used (see figure 7.3), set 0's in them; otherwise, no correct result can be obtained because addition is done on the numbers including indeterminate data.

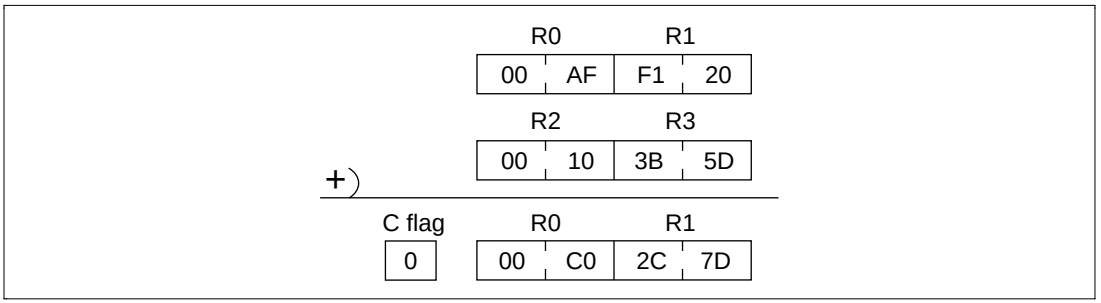


Figure 7.3 Example of Addition with Upper Bits Unused

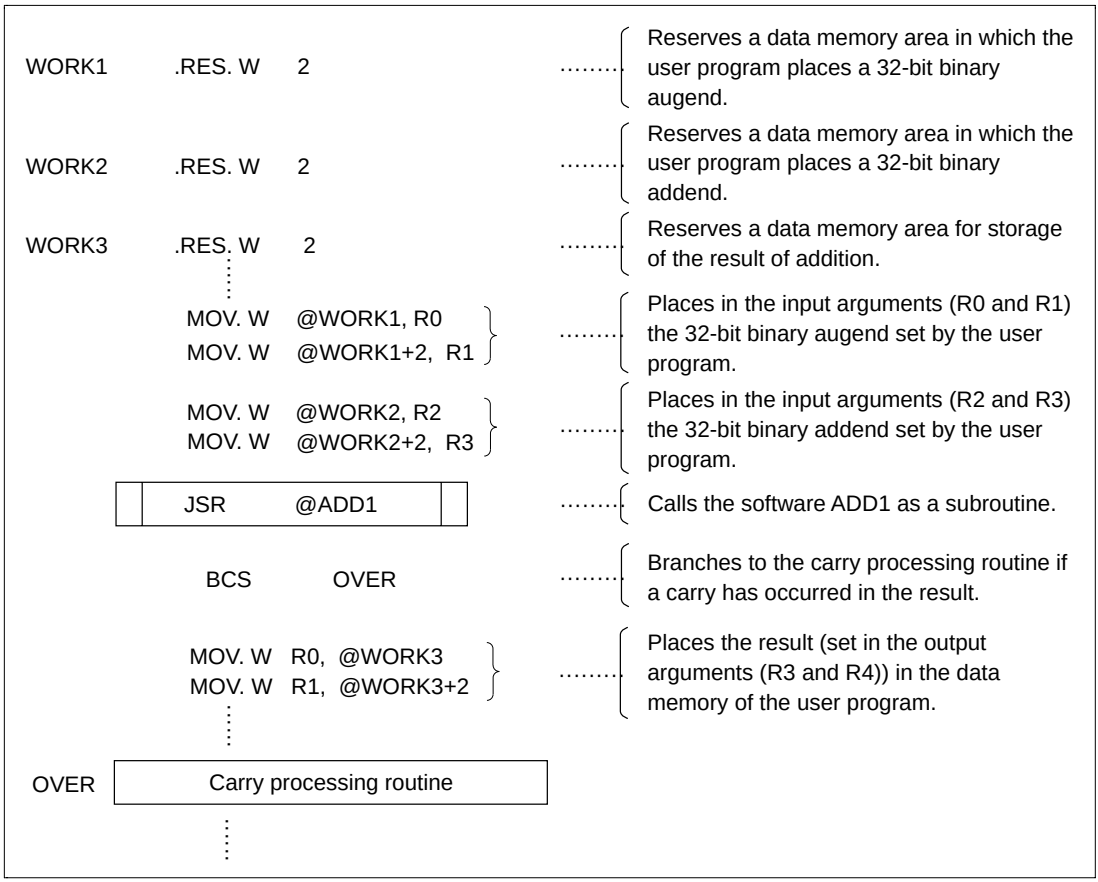
   b. After execution of the software ADD1, the augend is destroyed because the result is placed in R0 and R1. If the augend is necessary after software ADD1 execution, save it on memory.

3. Data memory

   The software ADD1 does not use the data memory.

4. Example of use

   Set an augend and an addend in the input arguments and call the software ADD1 as a subroutine.



5. Operation

   a. Addition of 3 bytes or more can be done by repeating 1-byte additions.

   b. A 1-word add instruction (ADD.W), which does not involve the state of the C flag, is used to add the lower word shown by equation 1. The C flag is set if a carry occurs after execution of the equation.

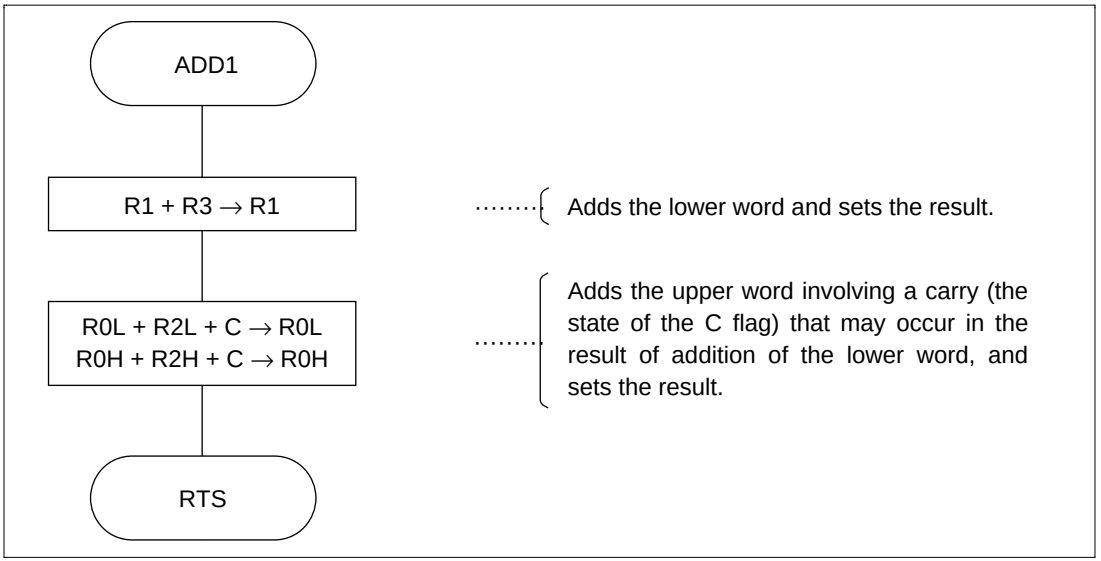
      $R1 + R3 \rightarrow R1$ ..... equation 1

c. A 1-byte add instruction (ADDX.B), which involves the state of the C flag, is used twice to add the upper word shown by equation 2.

$$\left.\begin{array}{l} R0L + R2L + C \rightarrow R0L \\ R0H + R2H + C \rightarrow R0H \end{array}\right\} \cdots\cdots\cdots \text{equation 1}$$

The C flag indicates a carry that may occur in the result of addition of the lower word executed in b. and the lower bytes of the upper word.

7.1.6 Flowchart



7.1.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:53:09
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00-NAME         :32 BIT ADDITION (ADD1)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0,R1  (SUMMAND)
    8                                   ;*                    R2,R3  (ADDEND)
    9                                   ;*
   10                                   ;*   RETURNS         :R0,R1  (SUM)
   11                                   ;*                     C flag OF CCR (C=0;TRUE , C=1;OVERFLOW)
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 ADD1_cod C 0000                        .SECTION        ADD1_code,CODE,ALIGN=2
   16                                        .EXPORT         ADD1
   17                                   ;
   18 ADD1_cod C      00000000          ADD1 .EQU    $               ;Entry point
   19 ADD1_cod C 0000 0931                   ADD.W   R3,R1           ;Adjust lower word
   20 ADD1_cod C 0002 0EA8                   ADDX.B  R2L,R0L         ;Adjust upper word
   21 ADD1_cod C 0004 0E20                   ADDX.B  R2H,R0H         ;
   22 ADD1_cod C 0006 5470                   RTS
   23                                   ;
   24                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.2 Subtraction of 32-Bit Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: SUB1

7.2.1 Function

1. The software SUB1 subtracts a 32-bit binary number from another 32-bit binary number and places the result (a 32-bit binary number) in a general-purpose register.
2. The arguments used with the software SUB1 are unsigned integers.
3. All data is manipulated on general-purpose registers.

7.2.2 Arguments

Description		Memory area	Data length (bytes)
Input	Minuend	R0, R1	4
	Subtrahend	R2, R3	4
Output	Result of subtraction	R0, R1	4
	Borrow	C flag (CCR)	

7.2.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	×	×	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.2.4 Specifications

Program memory (bytes)
8
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
14
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.2.5 Description

1. Details of functions
   a. The following arguments are used with the software SUB1:

   R0, R1: Contain a 32-bit binary minuend as an input argument. The result of subtraction is placed in these registers after execution of the software SUB1.

   R2, R3: Contain a 32-bit binary subtrahend as an input argument.

   C flag (CCR): Determines the presence or absence of a borrow as an output argument after execution of the software SUB1.

   C flag = 1: A borrow occurred in the result. (See figure 7.4)

   C flag = 0: No borrow occurred in the result.

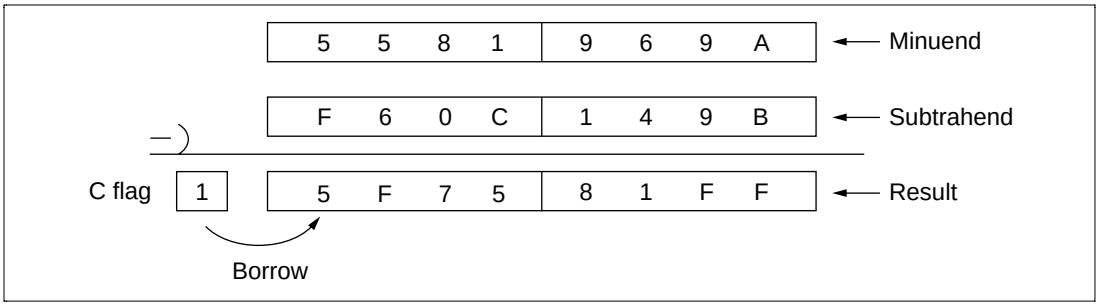


Figure 7.4 Example of Subtraction with a Borrow

b. Figure 7.5 shows an example of the software SUB1 being executed. When the input arguments are set as shown in (1), the result of subtraction is placed in R0 and R1 as shown in (2).

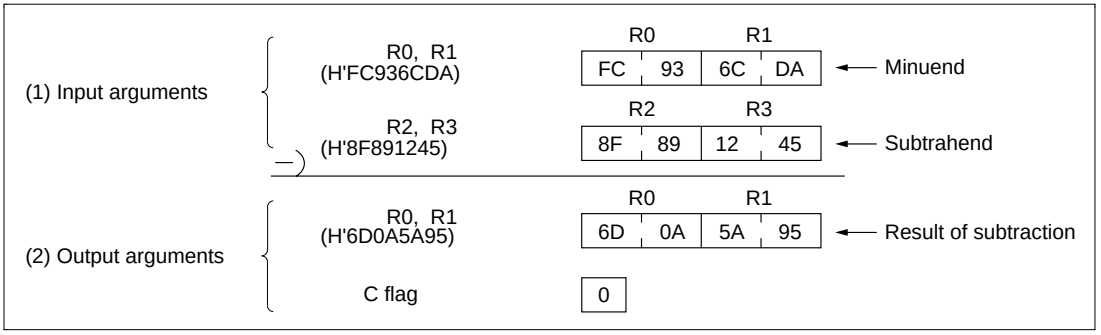


Figure 7.5 Example of Software SUB1 Execution

2. Notes on usage
   a. When upper bits are not used (see figure 7.6), set 0's in them; otherwise, no correct result can be obtained because subtraction is done on the numbers including indeterminate data.

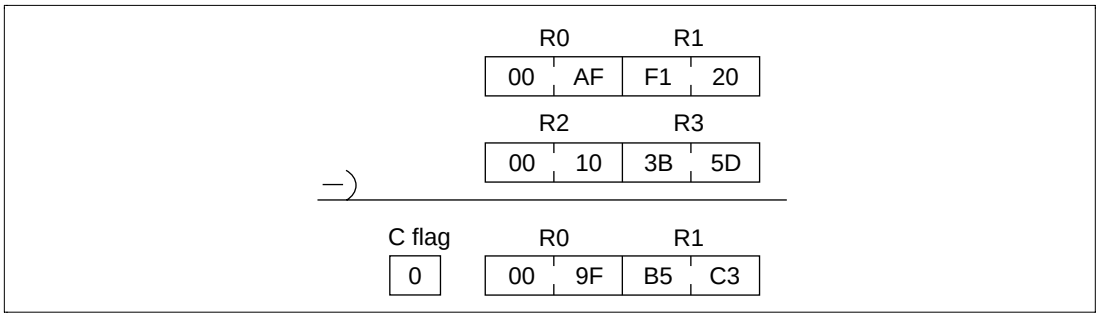


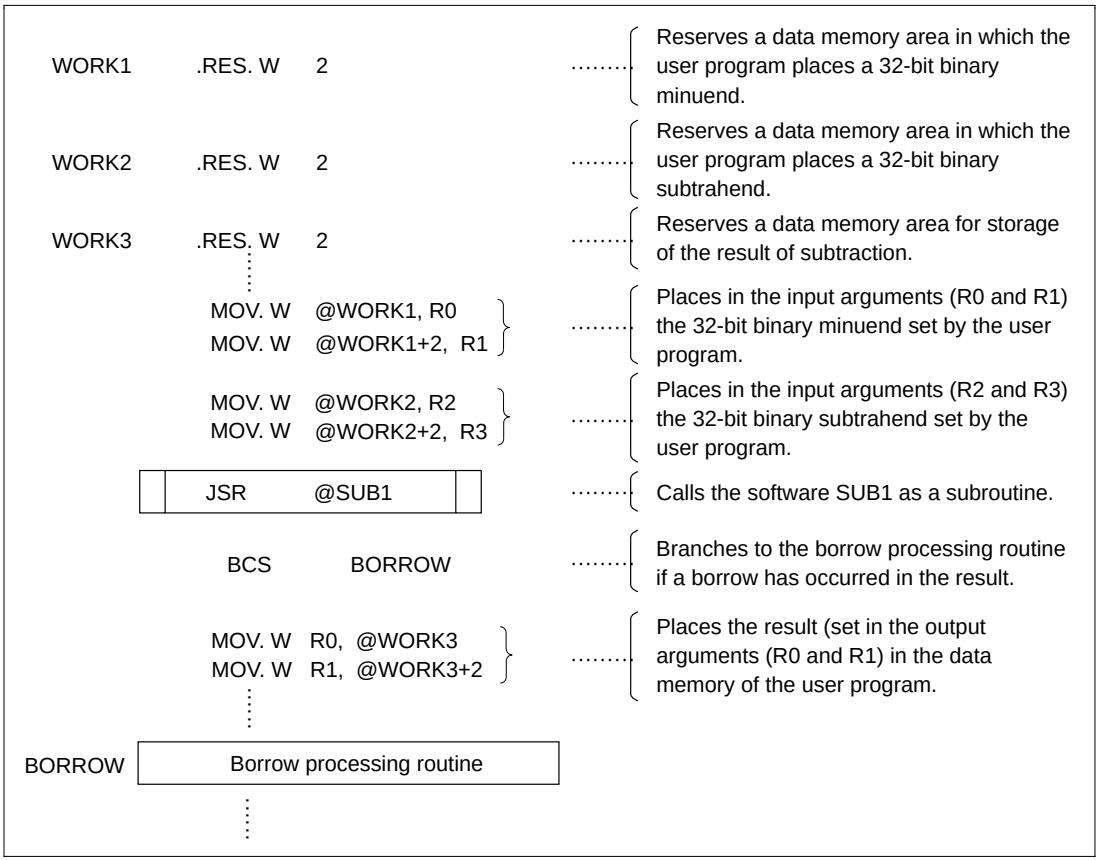
Figure 7.6 Example of Subtraction with Upper Bits Unused

   b. After execution of the software SUB1, the minuend is destroyed because the result is contained in R0 and R1. If the minuend is necessary after software SUB1 execution, save it on memory.
3. Data memory

   The software SUB1 does not use the data memory.

4. Example of use

Set a minuend and a subtrahend in the input arguments and call the software SUB1 as a subroutine.



5. Operation

a. Subtraction of 3 bytes or more can be done by repeating 1-byte subtractions.

b. A 1-word subtract instruction (SUB.W), which does not involve the state of the C flag, is used to subtract the lower word shown by equation 1. The C flag is set if a borrow occurs after execution of the equation.

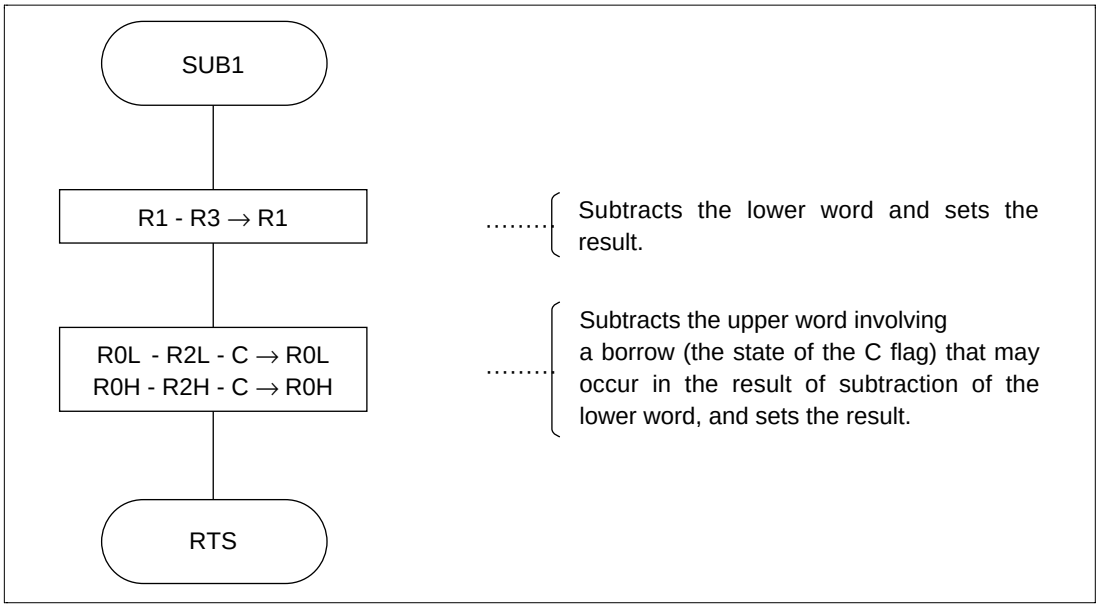
$R1 - R3 \rightarrow R1$ ..... equation 1

c. A 1-byte subtract instruction (SUBX.B), which involves the state of the C flag, is used twice to subtract the upper word shown by equation 2.

$$\left.\begin{array}{l} R0L - R2L - C \rightarrow R0 \\ R0H - R2H - C \rightarrow R0 \end{array}\right\} \cdots\cdots \text{ equation 2}$$

The C flag indicates a borrow that may occur in the result of subtraction of the lower word executed in b. and the lower bytes of the upper word.

7.2.6 Flowchart



7.2.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:53:37
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME       :32 BIT BINARY SUBTRUCTION (SUB1)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0,R1  (MINUEND)
    8                                   ;*                    R2,R3  (SUBTRAHEND)
    9                                   ;*
   10                                   ;*   RETURNS         :R0,R1  (RESULT)
   11                                   ;*                    C flag OF CCR (C=0;TRUE,C=1;BORROW)
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 SUB1_cod C 0000                        .SECTION        SUB1_code,CODE,ALIGN=2
   16                                        .EXPORT         SUB1
   17                                   ;
   18 SUB1_cod C      00000000       SUB1 .EQU    $               ;Entry point
   19 SUB1_cod C 0000 1931                   SUB.W   R3,R1           ;Subtruct lower word
   20 SUB1_cod C 0002 1EA8                   SUBX.B  R2L,R0L         ;Subtruct upper word
   21 SUB1_cod C 0004 1E20                   SUBX.B  R2H,R0H
   22                                   ;
   23 SUB1_cod C 0006 5470                   RTS
   24                                   ;
   25                                        .END
*****TOTAL ERRORS       0

*****TOTAL WARNINGS     0
```

7.3 Multiplication of 16-Bit Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: MUL

7.3.1 Function

1. The software MUL multiplies a 16-bit binary number by another 16-bit binary number and places the result (a 32-bit binary number) in a general-purpose register.
2. The arguments used with the software MUL are unsigned integers.
3. All data is manipulated on general-purpose registers.

7.3.2 Arguments

Description		Memory area	Data length (bytes)
Input	Multiplicand	R1	2
	Multiplier	R0	2
Output	Result of multiplication	R1, R2	4

7.3.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	↕	↕	×	×	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.3.4 Specifications

Program memory (bytes)
32
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
86
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.3.5 Description

1. Details of functions
   a. The following arguments are used with the software MUL:
      R0: Contains a 16-bit binary multiplier as an input argument.
      R1: Contains a 16-bit binary multiplicand as an input argument. The upper 2 bytes of the result are placed in this register after execution of the software MUL.
      R2: Contains the lower 2 bytes of the result as an output argument.

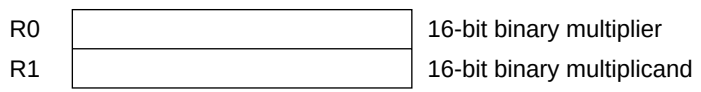


Figure 7.7 Input Argument Setting

b. Figure 7.8 shows an example of the software MUL being executed. When the input arguments are set as shown in (1), the result of multiplication is placed in R1 and R2 as shown in (2).

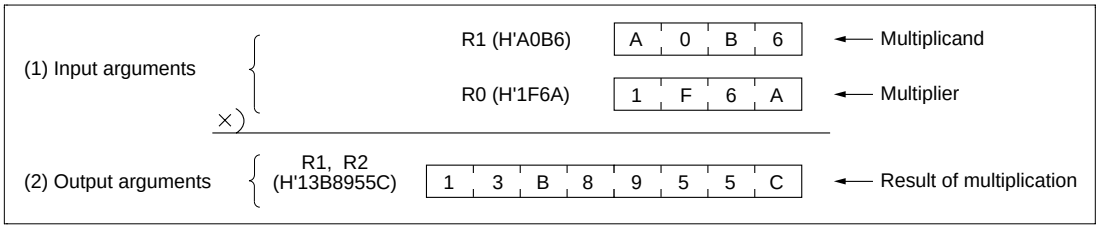


Figure 7.8 Example of Software MUL Execution

c. Table 7.1 lists the results of multiplication with 0's placed in the input arguments.

Table 7.1 Results of Multiplication with 0's Placed in Input Arguments

Input argument		Output argument
Multiplicand (R1)	Multiplier (R0)	Product (R1, R2)
H'****	H'0000	H'00000000
H'0000	H'****	H'00000000
H'0000	H'0000	H'00000000

Note: H'**** is a hexadecimal number.

2. Notes on usage

a. When upper bits are not used (see figure 7.9), set 0's in them; otherwise, no correct result can be obtained because multiplication is made on the numbers including indeterminate data.

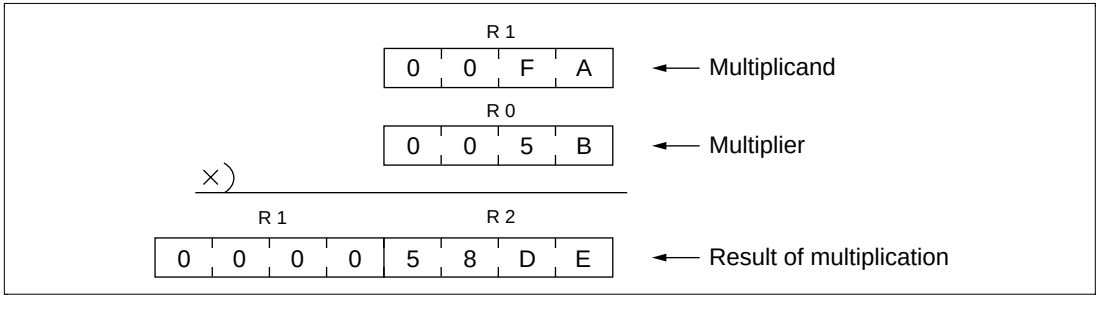


Figure 7.9 Example of Multiplication with Upper Bits Unused

b. After execution of the software MUL, the multiplicand is destroyed because the result is placed in R1. If the multiplicand is necessary after software MUL execution, save it on memory.

3. Data memory

   The software MUL does not use the data memory.

4. Example of use

   Set a multiplicand and a multiplier in the input arguments and call the software MUL as a subroutine.

```
WORK1    .RES. W    1          ..... Reserves a data memory area in which the
                                      user program places a 16-bit binary
                                      multiplicand.
WORK2    .RES. W    1          ..... Reserves a data memory area in which the
                                      user program places a 16-bit binary
                                      multiplier.
WORK3    .RES. W    2          ..... Reserves a data memory area for storage
                                      of the result of multiplication (a 32-bit
                                      binary number).
           :
         MOV. W   @WORK1,  R1  ..... Places in the input arguments (R1) the 16-
                                      bit binary multiplicand set by the user
                                      program.
         MOV. W   @WORK2,  R0  ..... Places in the input arguments (R0) the 16-
                                      bit binary multiplier set by the user
                                      program.
       +-------------------+
       |  JSR       @MUL   |  ..... Calls the software MUL as a subroutine.
       +-------------------+
         MOV. W   R1,  @WORK3  }
         MOV. W   R2,  @WORK3+2}  ..... Places the result (the 32-bit binary number,
                                        set in the output arguments) in the data
                                        memory of the user program.
           :
```

5. Operation
   a. Figure 7.10 shows an example of multiplying 16-bit binary numbers.

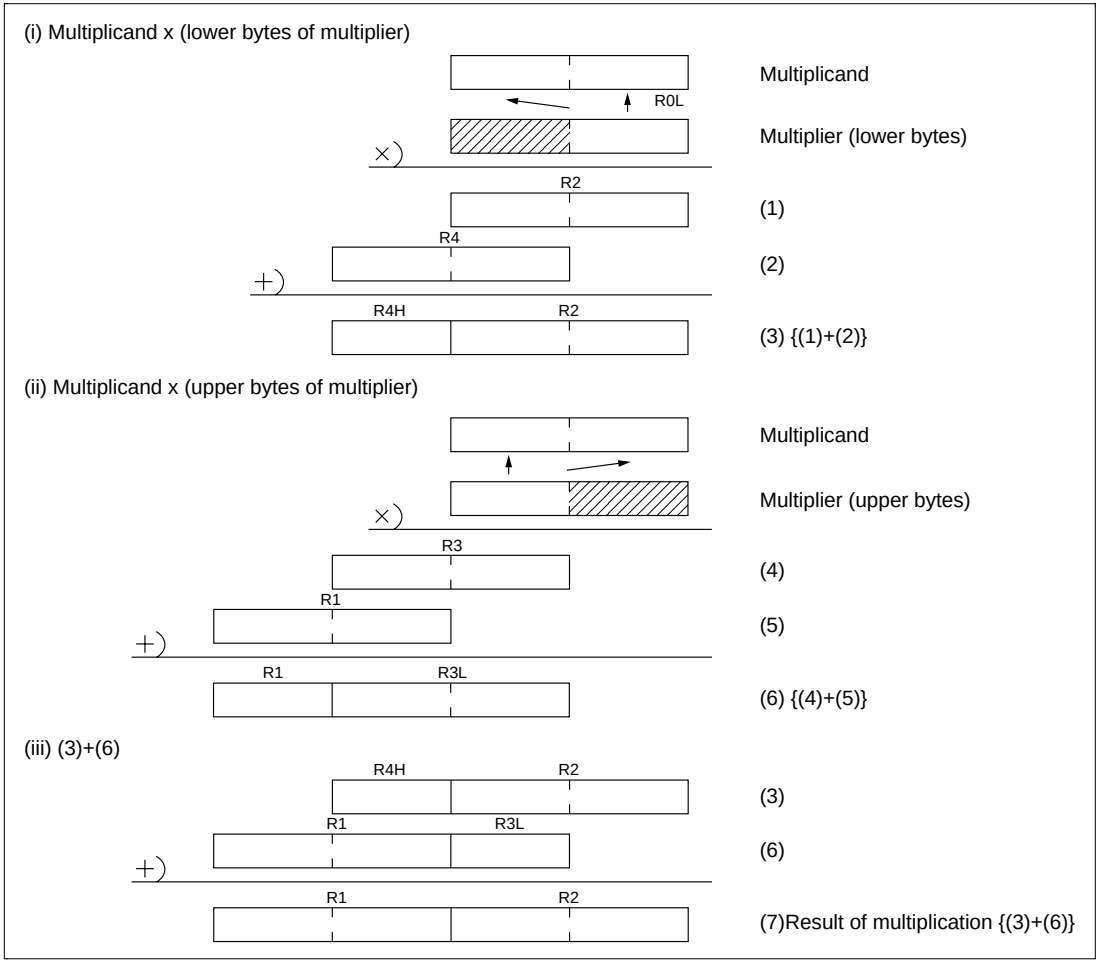


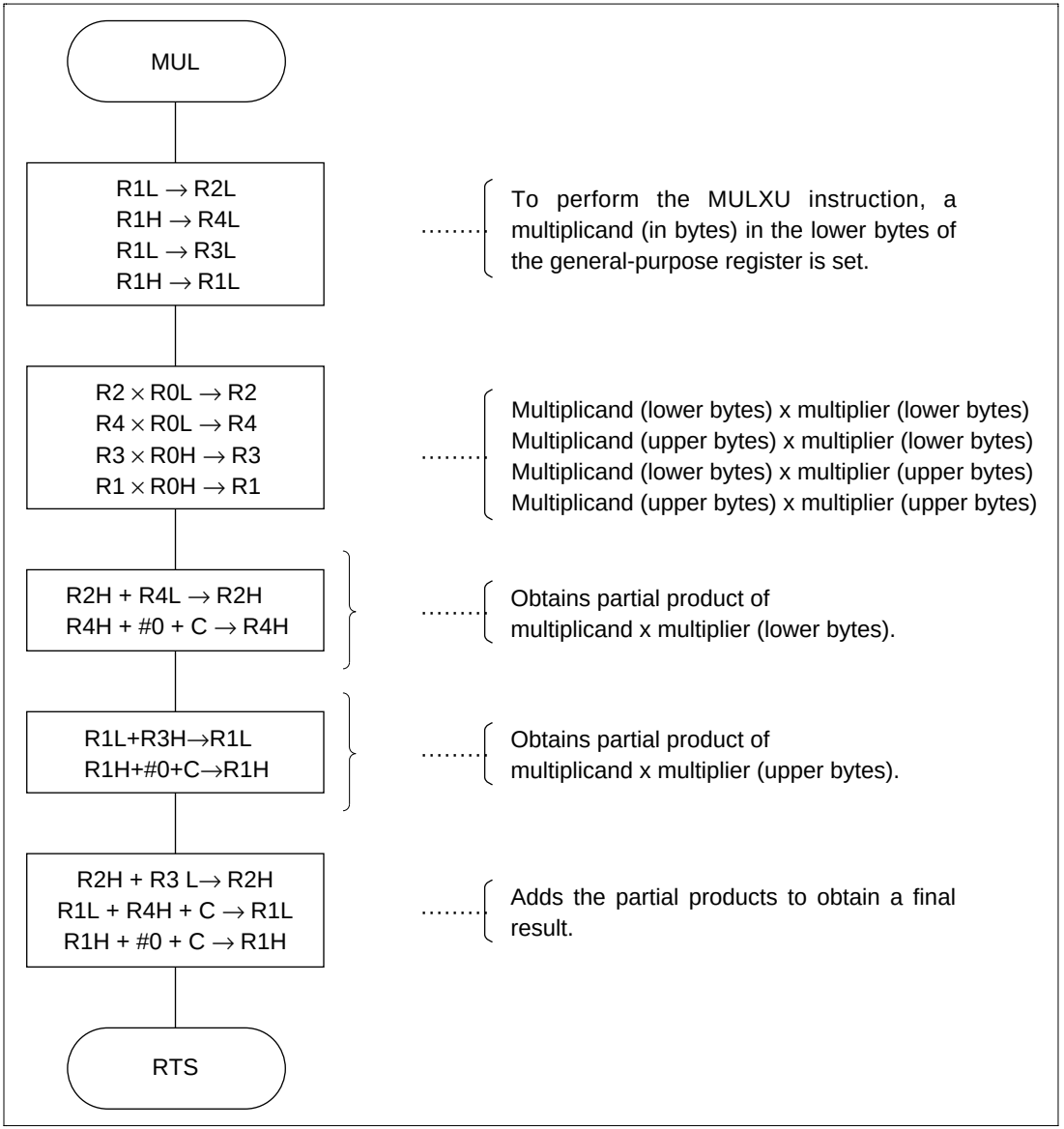
Figure 7.10 Example of Software MUL Execution

Multiplication of 16-bit binary numbers consists of two stages as shown in figure 7.10: (3) finding two partial products with the MULXU instruction and adding the two results (6).

   b. The program runs in the following steps:
      (i) The MULXU instruction is used to obtain the result of the multiplicand (lower bytes) × the multiplier (lower bytes) ((1) in figure 7.10) and the result of the multiplicand (upper bytes) × the multiplier (lower bytes) ((2) in figure 7.10). Then these two results are added to obtain a partial product, that is, the multiplicand x the multiplier (lower bytes) ((3) in figure 7.10).
      (ii) The MULXU instruction is used to obtain another partial product, that is, the multiplicand x the multiplier (upper bytes) ((6) in figure 7.10).

(iii) The two partial products (i) and (ii) are added to obtain the final product ((4) in figure 7.10).

7.3.6 Flowchart



7.3.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:54:03
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :16 BIT MULTIPLICATION (MUL)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     (MULTIPLIER)
    8                                   ;*                    R1     (MULTIPLICAND)
    9                                   ;*
   10                                   ;*   RETURNS         :R1     (UPPER WORD OF RESULT)
   11                                   ;*                    R2     (LOWER WORD OF RESULT)
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 MUL_code C 0000                        .SECTION        MUL_code,CODE,ALIGN=2
   16                                        .EXPORT         MUL
   17                                   ;
   18 MUL_code C      00000000          MUL  .EQU    $               ;Entry point
   19 MUL_code C 0000 0C9A                   MOV.B   R1L,R2L         ;R1L -> R2L
   20 MUL_code C 0002 0C1C                   MOV.B   R1H,R4L         ;R1H -> R4L
   21 MUL_code C 0004 0C9B                   MOV.B   R1L,R3L         ;R1L -> R3L
   22 MUL_code C 0006 0C19                   MOV.B   R1H,R1L         ;R1H -> R1L
   23                                   ;
   24 MUL_code C 0008 5082                   MULXU   R0L,R2          ;R0L * R2L -> R2
   25 MUL_code C 000A 5084                   MULXU   R0L,R4          ;R0L * R4L -> R4
   26 MUL_code C 000C 5003                   MULXU   R0H,R3          ;R0H * R3L -> R3
   27 MUL_code C 000E 5001                   MULXU   R0H,R1          ;R0H * R1L -> R1
   28                                   ;
   29 MUL_code C 0010 08C2                   ADD.B   R4L,R2H         ;R2H + R4L       -> R2H
   30 MUL_code C 0012 9400                   ADDX.B  #H'00,R4H       ;R4H + #H'00 + C -> R4H
   31 MUL_code C 0014 0839                   ADD.B   R3H,R1L         ;R3H + R1L       -> R1L
   32 MUL_code C 0016 9100                   ADDX.B  #H'00,R1H       ;R1H + #H'00 + C -> R1H
   33                                   ;
   34 MUL_code C 0018 08B2                   ADD.B   R3L,R2H         ;R3L + R2H       -> R2H
   35 MUL_code C 001A 0E49                   ADDX.B  R4H,R1L         ;R4H + R1L   + C -> R1L
   36 MUL_code C 001C 9100                   ADDX.B  #H'00,R1H       ;R1H + #H'00 + C -> R1H
   37                                   ;
   38 MUL_code C 001E 5470                   RTS
   39                                   ;
   40                                        .END
*****TOTAL ERRORS       0

*****TOTAL WARNINGS     0
```

7.4 Division of 32-Bit Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: DIV

7.4.1 Function

1. The software DIV divides a 32-bit binary number by another 32-bit binary number and places the result (a 32-bit binary number) in a general-purpose register.
2. The arguments used with the software DIV are unsigned integers.
3. All data is manipulated on general-purpose registers.

7.4.2 Arguments

Description		Memory area	Data length (bytes)
Input	Dividend	R0, R1	4
	Divisor	R2, R3	4
Output	Result of division (Quotient)	R0, R1	4
	Result of division (Remainder)	R4, R5	4
	Errors	Z flag (CCR)	

7.4.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	×	×	↕	↕	•	•

I	U	H	U	N	Z	V	C
•	×	•	×	×	↕	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.4.4 Specifications

Program memory (bytes)
58
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
1374
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.4.5 Description

1. Details of functions
   a. The following arguments are used with the software DIV:
      R0: Contains the upper 2 bytes of a 32-bit binary dividend. The upper 2 bytes of the result of division (quotient) are placed in this register after execution of the software DIV.
      R1: Contains the lower 2 bytes of the 32-bit binary dividend. The lower 2 bytes of the result of division (quotient) are placed in this register after execution of the software DIV.
      R2: Contains the upper 2 bytes of a 32-bit binary divisor as an input argument.
      R3: Contains the lower 2 bytes of the 32-bit binary divisor as an input argument.
      R4: The upper 2 bytes of the result of division (remainder) are placed in this register as an output argument.
      R5: The lower 2 bytes of the result of division (remainder) are placed in this register as an output argument.

Z flag (CCR): Determines the presence or absence of an error (division by 0) with the software DIV as an output argument.

Z flag = 1: The divisor was 0.

Z flag = 0: The divisor was not 0.

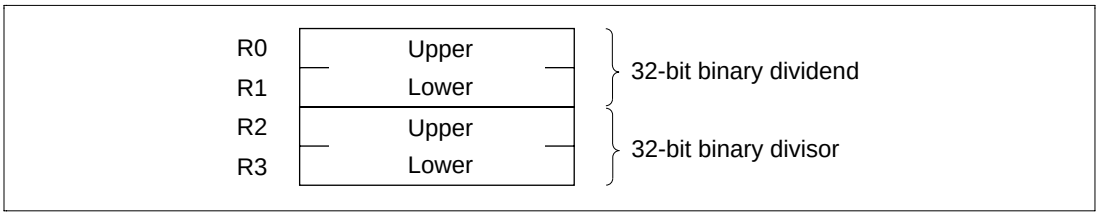


Figure 7.11 Input Argument Setting

b. Figure 7.12 shows an example of the software DIV being executed. When the input arguments are set as shown in (1), the result of division is placed as shown in (2).

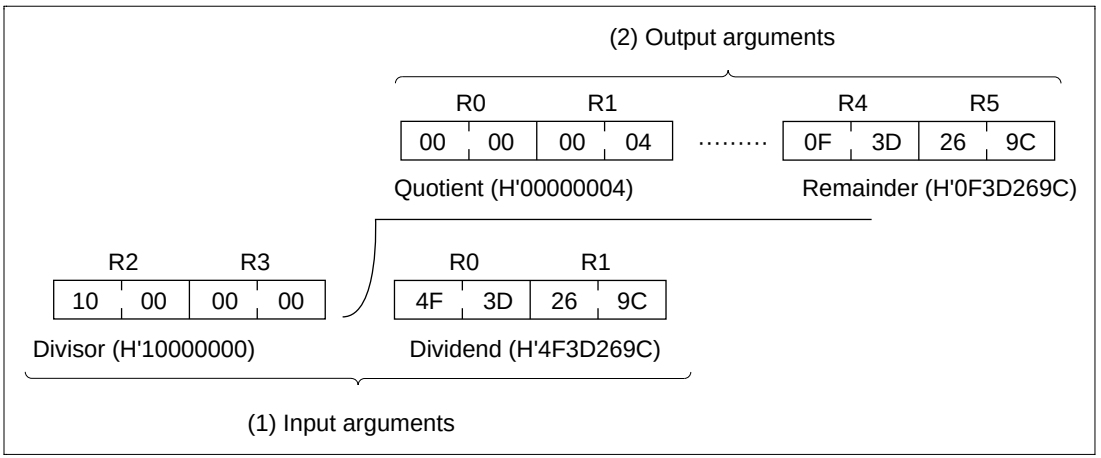


Figure 7.12 Example of Software DIV Execution

c. Table 7.2 lists the results of division with 0's placed in the input arguments.

Table 7.2 Results of Division with 0's Placed in Input Arguments

Input argument		**Output argument**		
Dividend (R0, R1)	Divisor (R2, R3)	Quotient (R0, R1)	Remainder (R4, R5)	Error (Z)
H'********	H'00000000	H'********	H'00000000	1
H'00000000	H'********	H'00000000	H'00000000	0
H'00000000	H'00000000	H'00000000	H'00000000	1

Note: H'**** is a hexadecimal number.

2. Notes on usage

   When upper bits are not used (see figure 7.13), set 0's in them; otherwise, no correct result can be obtained because division is done on the numbers including indeterminate data.

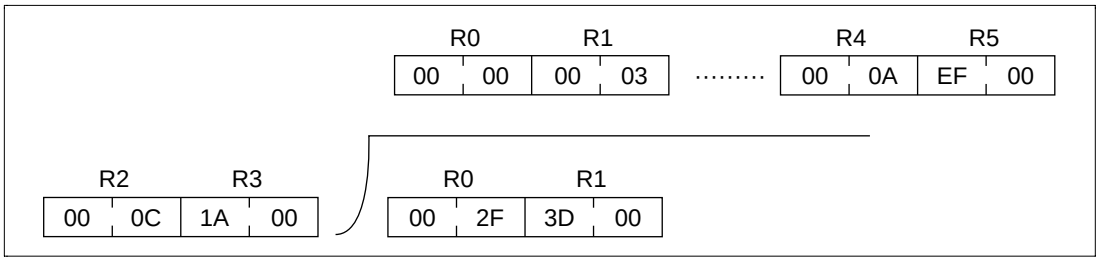


Figure 7.13 Example of Division with Upper Bits Unused

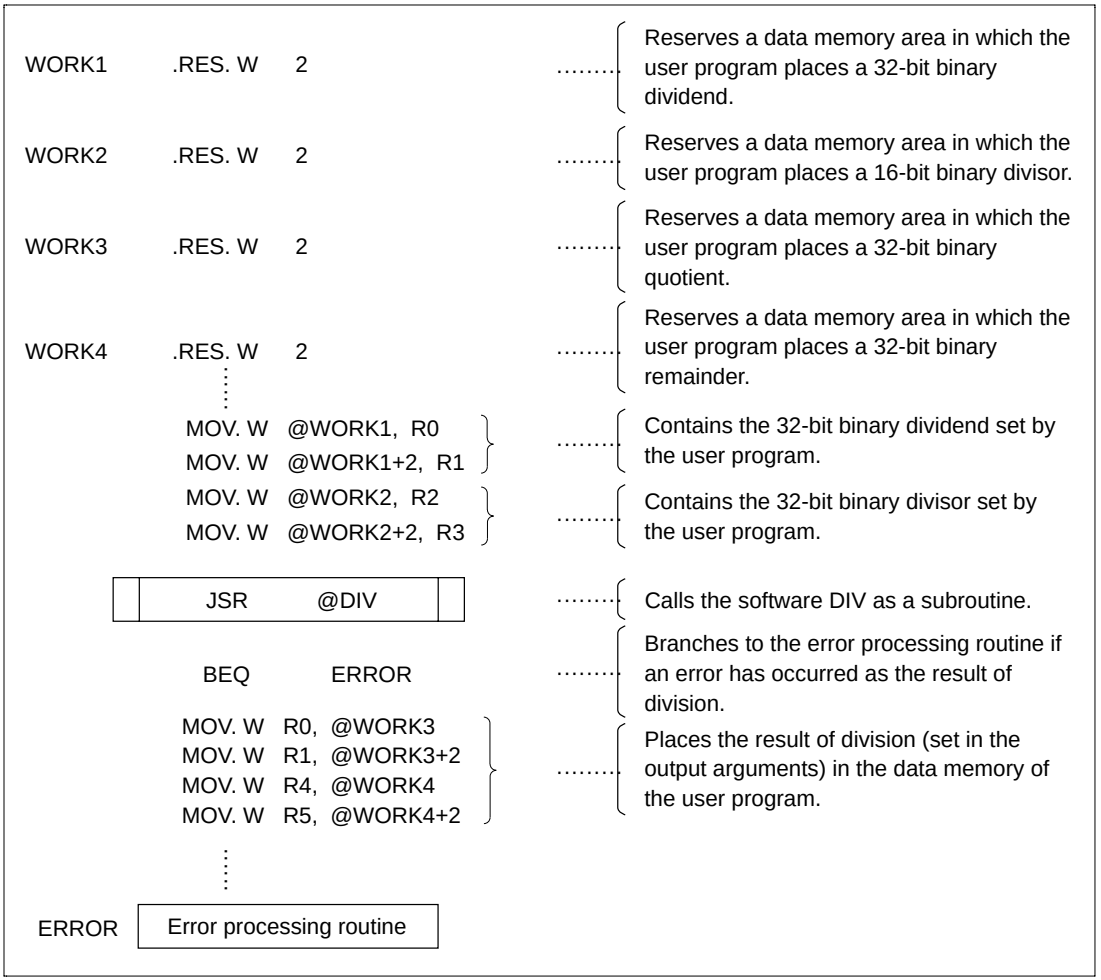
   b. After execution of the software DIV, the dividend is destroyed because the quotient is placed in R0 and R1. If the dividend is necessary after software DIV execution, save it on memory.

3. Data memory

   The software DIV does not use the data memory.

4. Example of use

   Set a dividend and a divisor in the input arguments and call the software DIV as a subroutine.


WORK1 .RES. W 2
Reserves a data memory area in which the user program places a 32-bit binary dividend.
WORK2 .RES. W 2
Reserves a data memory area in which the user program places a 16-bit binary divisor.
WORK3 .RES. W 2
Reserves a data memory area in which the user program places a 32-bit binary quotient.
WORK4 .RES. W 2
Reserves a data memory area in which the user program places a 32-bit binary remainder.
MOV. W @WORK1, R0
MOV. W @WORK1+2, R1
Contains the 32-bit binary dividend set by the user program.
MOV. W @WORK2, R2
MOV. W @WORK2+2, R3
Contains the 32-bit binary divisor set by the user program.
JSR @DIV
Calls the software DIV as a subroutine.
BEQ ERROR
Branches to the error processing routine if an error has occurred as the result of division.
MOV. W R0, @WORK3
MOV. W R1, @WORK3+2
MOV. W R4, @WORK4
MOV. W R5, @WORK4+2
Places the result of division (set in the output arguments) in the data memory of the user program.
ERROR
Error processing routine

5. Operation

a. A binary division can be done by performing a series of subtractions. figure 7.14 shows an example of division (H'0 ÷ DH'03).

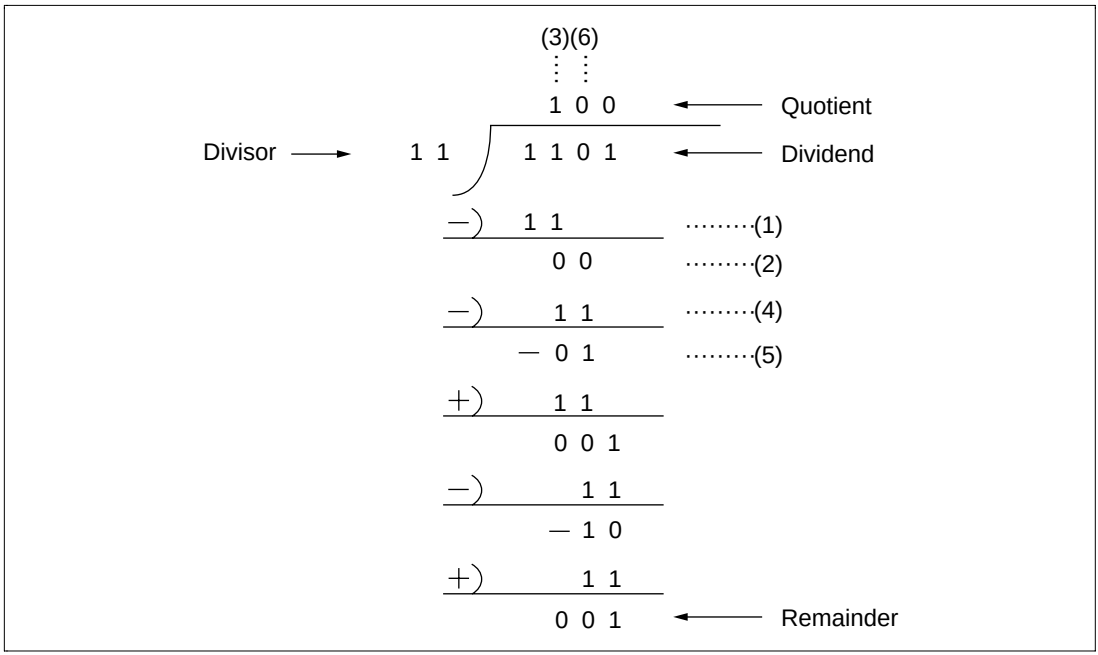


Figure 7.14 Example of Software DIV Execution (H'0D ÷ H'03)

This example indicates that the quotient and remainder are obtained by repeating a process of subtracting the dividend from the divisor. More specifically, the dividend is taken out bit by bit from the upper byte and the divisor is subtracted from the sum of the data and the previous result of subtraction.

b. The program runs in the following steps:

(i) A shift count (D732) is set.

(ii) The dividend is 1 bit shifted to the left to place the most significant bit c. in the least significant bit of the remainder.

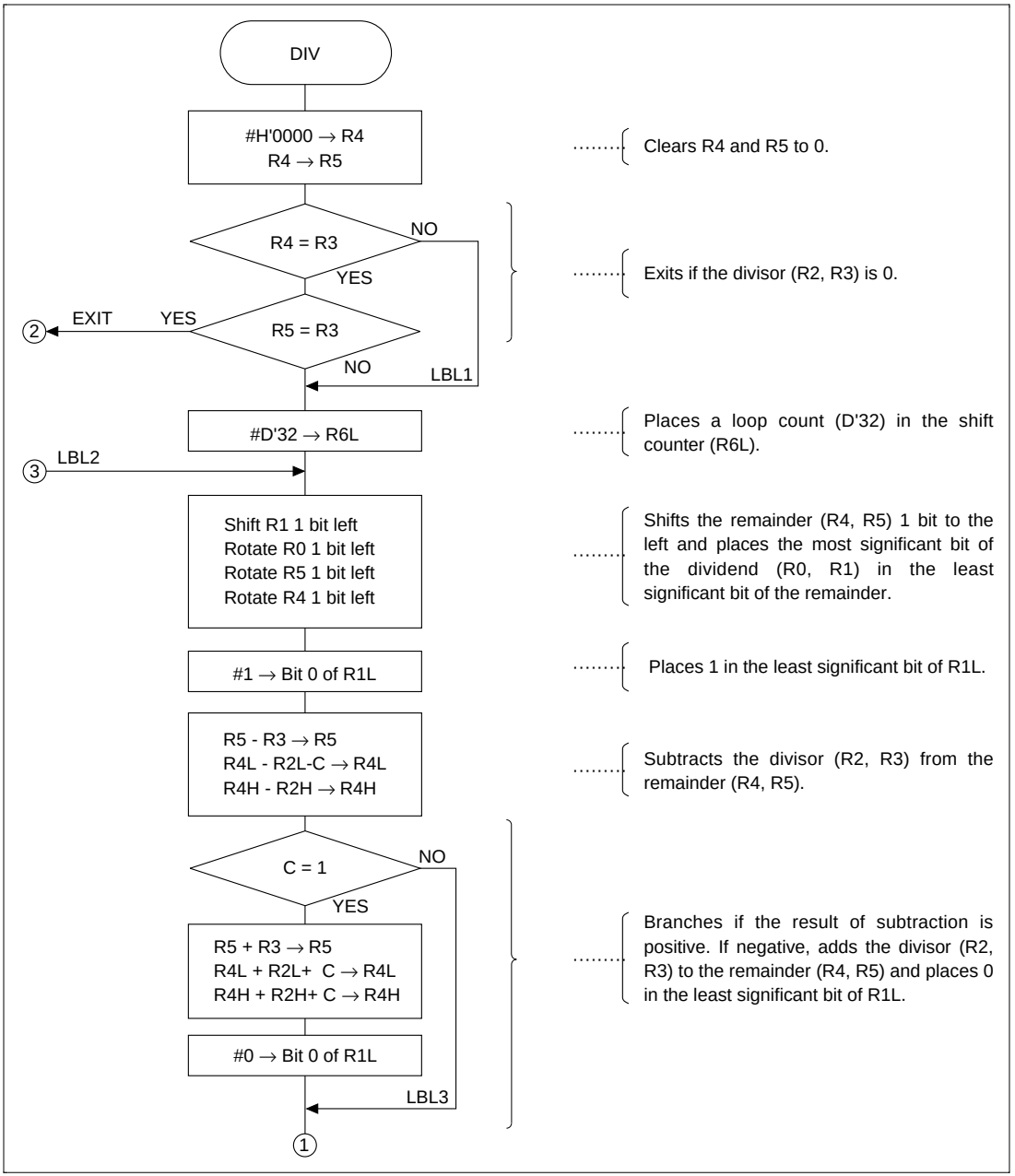
(iii) The divisor is subtracted from the remainder.

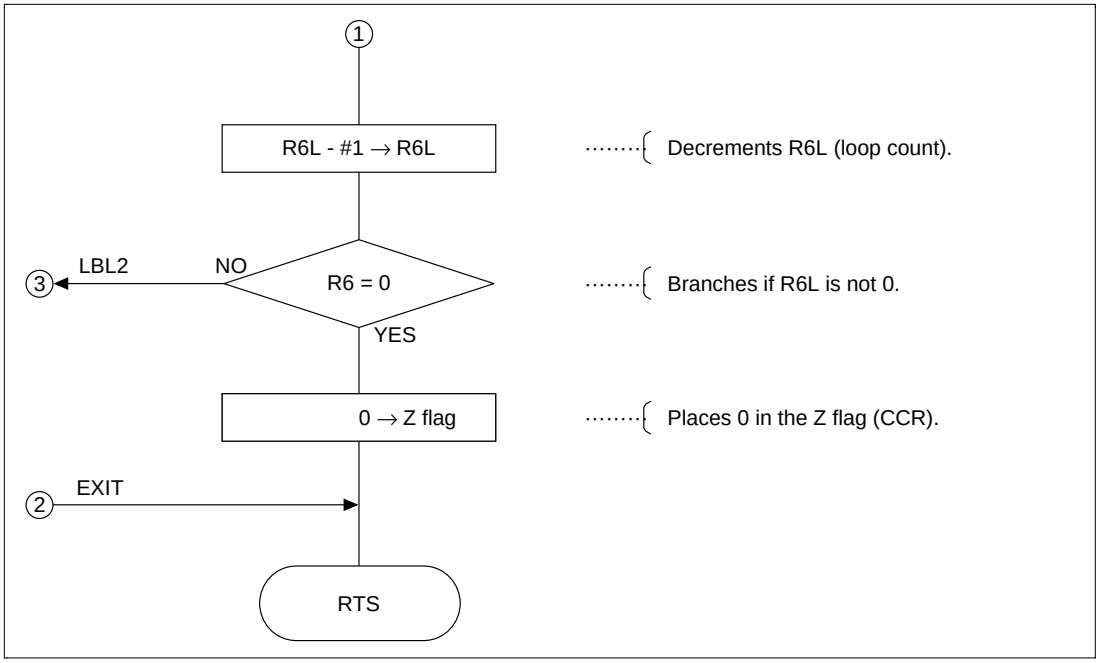
If the result is positive, "1" is placed in the least significant bit of the dividend ((1) → (2) → (3) in figure 7.14). If the result is negative, "0" is placed in the least significant bit of the dividend and the divisor is added to the result to return to the state before the subtraction. ((4) → (5) → (6) in figure 7.14).

(iv) The shift count (set in step (i)) is decremented.

(v) Steps (ii) to (iv) are repeated until the shift count reaches H'00.

7.4.6 Flowchart




1
R6L - #1 → R6L
Decrements R6L (loop count).
3
LBL2
NO
R6 = 0
Branches if R6L is not 0.
YES
0 → Z flag
Places 0 in the Z flag (CCR).
2
EXIT
RTS

7.4.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:58:57
PROGRAM NAME =
    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :32 BIT DIVISION (DIV)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     (UPPER WORD DIVIDEND)
    8                                   ;*                    R1     (LOWER WORD DIVIDEND)
    9                                   ;*                    R2     (UPPER WORD DIVISOR)
   10                                   ;*                    R3     (LOWER WORD DIVISOR)
   11                                   ;*
   12                                   ;*   RETURNS         :R0     (UPPER WORD QUOTIENT)
   13                                   ;*                    R1     (LOWER WORD QUOTIENT)
   14                                   ;*                    R2     (UPPER WORD RESIDUE)
   15                                   ;*                    R3     (LOWER WORD RESIDUE)
   16                                   ;*                    Z flag OF CCR (Z=0;TRUE , Z=1;FALSE)
   17                                   ;*
   18                                   ;*********************************************************************
   19                                   ;
   20 DIV_code C 0000                        .SECTION        DIV_code,CODE,ALIGN=2
   21                                        .EXPORT         DIV
   22                                   ;
   23 DIV_code C      00000000       DIV  .EQU    $               ;Entry point
   24 DIV_code C 0000 79040000            MOV.W   #H'0000,R4      ;Clear R4
   25 DIV_code C 0004 0D45                MOV.W   R4,R5           ;Clear R5
   26 DIV_code C 0006 1D42                CMP.W   R4,R2
   27 DIV_code C 0008 4604                BNE     LBL1            ;Branch if Z flag = 0
   28 DIV_code C 000A 1D43                CMP.W   R4,R3
   29 DIV_code C 000C 472A                BEQ     EXIT            ;Branch if Z flag = 1 then exit
   30 DIV_code C 000E                  LBL1
   31 DIV_code C 000E FE20                MOV.B   #D'32,R6L       ;Set byte counter
   32 DIV_code C 0010                  LBL2
   33 DIV_code C 0010 1009                SHLL    R1L             ;Shift dividend 1 bit left
   34 DIV_code C 0012 1201                ROTXL   R1H
   35 DIV_code C 0014 1208                ROTXL   R0L
   36 DIV_code C 0016 1200                ROTXL   R0H
   37                                   ;
   38 DIV_code C 0018 120D                ROTXL   R5L
   39 DIV_code C 001A 1205                ROTXL   R5H
   40 DIV_code C 001C 120C                ROTXL   R4L
   41 DIV_code C 001E 1204                ROTXL   R4H
   42                                   ;
   43 DIV_code C 0020 7009                BSET    #0,R1L          ;Bit set bit 0 of R1L
   44                                   ;
   45 DIV_code C 0022 1935                SUB.W   R3,R5           ;R5  - R3      -> R5
   46 DIV_code C 0024 1EAC                SUBX.B  R2L,R4L         ;R4L - R2L - C -> R4L
   47 DIV_code C 0026 1E24                SUBX.B  R2H,R4H         ;R4H - R2H - C -> R4H
   48                                   ;
   49 DIV_code C 0028 4408                BCC     LBL3            ;Branch if C=0
   50 DIV_code C 002A 0935                ADD.W   R3,R5           ;R3  + R5      -> R3
   51 DIV_code C 002C 0EAC                ADDX.B  R2L,R4L         ;R2L + R4L + C -> R4L
   52 DIV_code C 002E 0E24                ADDX.B  R2H,R4H         ;R2H + R4H + C -> R4H
   53                                   ;
   54 DIV_code C 0030 7209                BCLR    #0,R1L          ;Bit clear bit 0 of R1L
   55 DIV_code C 0032                  LBL3
   56 DIV_code C 0032 1A0E                DEC.B   R6L             ;Decrement R6L
   57 DIV_code C 0034 46DA                BNE     LBL2            ;Branch if Z=0
   58 DIV_code C 0036 06FB                ANDC    #B'11111011,CCR         ;Clear Z flag
   59 DIV_code C 0038                  EXIT
   60 DIV_code C 0038 5470                RTS
   61                                   ;
   62                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.5 Addition of Multiple-Precision Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: ADD2

7.5.1 Function

1. The software ADD2 adds a multiple-precision binary number to another multiple-precision binary number and places the result in the data memory where the augend was placed.
2. The arguments used with the software ADD2 are unsigned integers, each being up to 255 bytes long.

7.5.2 Arguments

Description		Memory area	Data length (bytes)
Input	Augend and addend byte length	R0L	1
	Start address of augend	R3	2
	Start address of addend	R4	2
Output	Start address of the result of addition	R3	2
	Error	Z flag (CCR)	
	Carry	C flag (CCR)	

7.5.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	×	↕	×	×	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	↕	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.5.4 Specifications

Program memory (bytes)
42
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
7170
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.5.5 Notes

The clock cycle count (7170) in the specifications is for addition of 255 bytes to 255 bytes.

7.5.6 Description

1. Details of functions
   a. The following arguments are used with the software ADD2:
      R0L: Contains, as an input argument, the byte count of an augend and an addend in 2-digit hexadecimals.
      R3: Contains the start address of the augend in the data memory area. The start address of the result of addition is placed in this register after execution of the software ADD2.
      R4: Contains, as an input argument, the start address of the addend in the data memory area.
      Z flag (CCR): Indicates an error in data length as an output argument.

Z flag = 0: The data byte count (R0L) was not 0.

Z flag = 1: The data byte count (R0L) was 0 (indicating an error).

C flag (CCR): Determines the presence or absence of a carry, as an output argument, after execution of the software ADD2.

C flag = 0: No carry occurred in the result of addition.

C flag = 1: A carry occurred in the result of addition (see figure 17-2).

b. Figure 7.15 shows an example of the software ADD2 being executed. When the input arguments are set as shown in (1), the result of addition is placed in the data memory area as shown in (2).

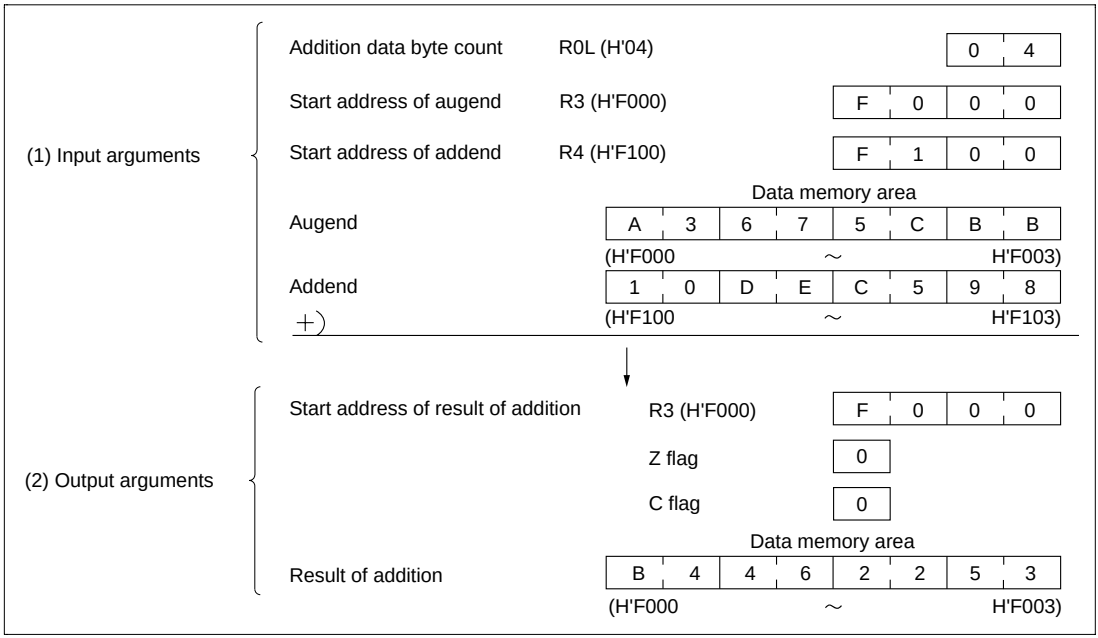


Figure 7.15 Example of Software ADD2 Execution

Figure 7.16 shows an example of addition with a carry that occurred in the result.

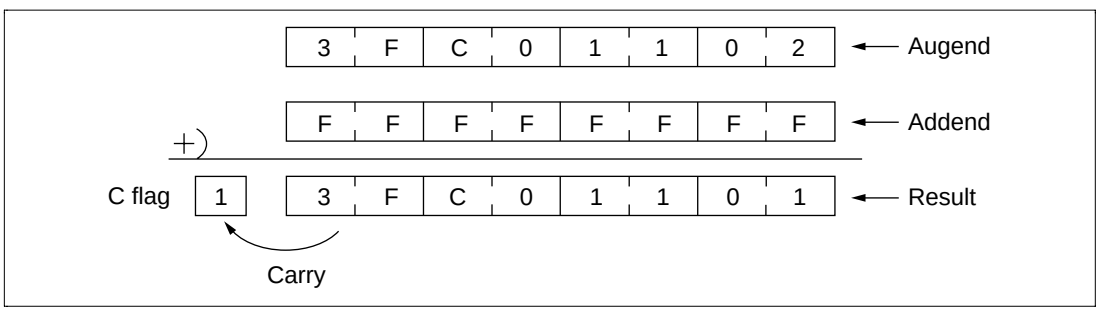


Figure 7.16 Example of Addition with a Carry

2. Notes on usage
   a. When upper bits are not used (see figure 7.17), set 0's in them. The software ADD2 performs byte-based addition; if 0's are not set in the upper bits unused, no correct result can be obtained because the addition is done on the numbers including indeterminate data.

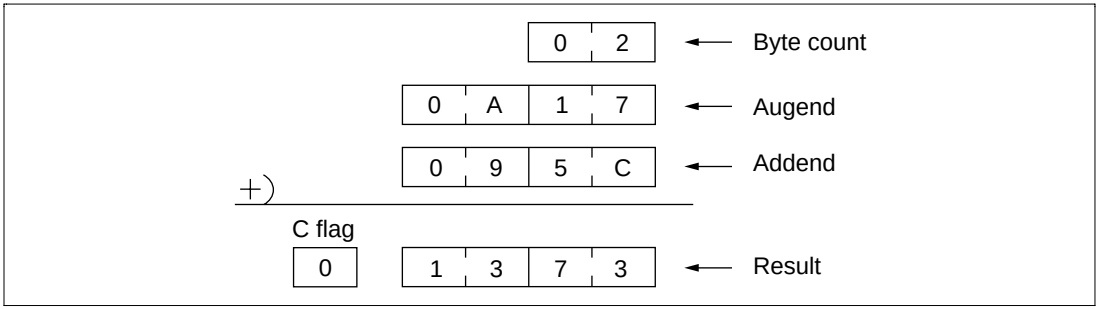


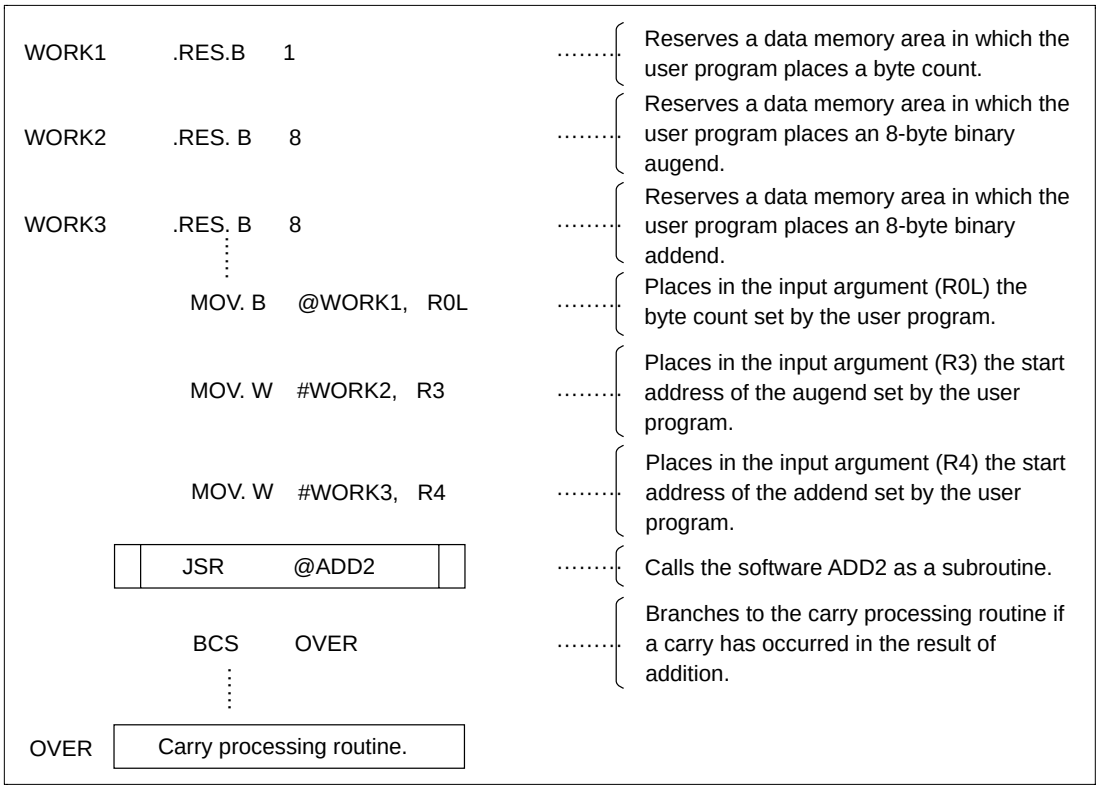
Figure 7.17 Example of Addition with Upper Bits Unused

   b. After execution of the software ADD2, the augend is destroyed because the result is placed in the data memory area where the augend was set. If the augend is necessary after software ADD2 execution, save it on memory.
3. Data memory

   The software ADD2 does not use the data memory.

4. Example of use

This is an example of adding 8 bytes of data. Set the start addresses of a byte count, an augend and an addend in the registers and call the software ADD2 as a subroutine.



5. Operation

a. Addition of multiple-precision binary numbers can be done by performing a series of add instructions with a carry flag (ADDX.B) as the augend and addend data are placed in registers on a byte basis.

b. The end address of the data memory area containing the augend is placed in R3, and the end address of the data memory area containing the addend is placed in R4.

c. R1L is cleared for saving the C flag.

d. The augend and addend are loaded in R2L and R2H respectively, byte by byte, starting at their end address and equation 1 is executed:

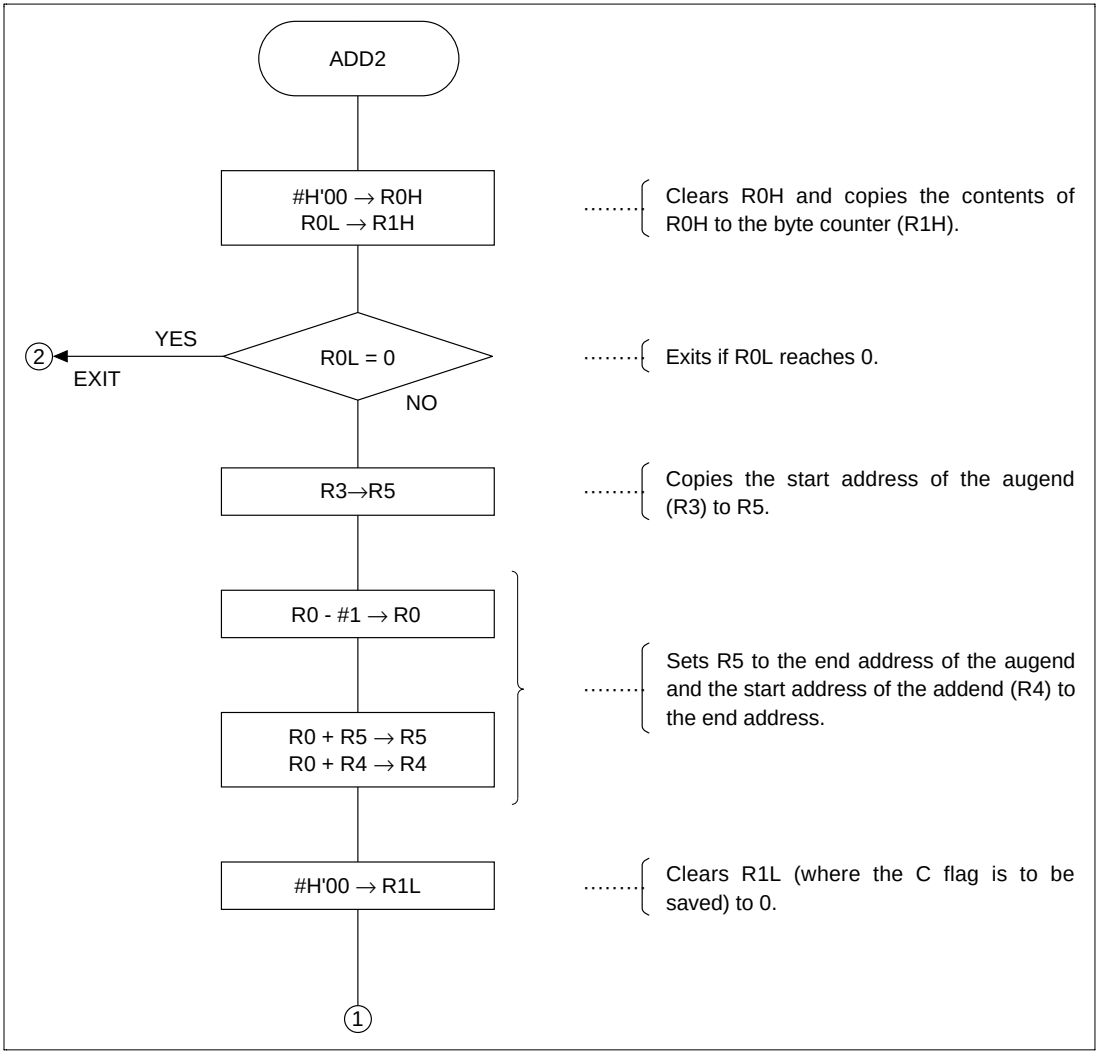
$$\left.\begin{array}{l}\text{Augend} + \text{addend} + \text{C} \rightarrow \text{R2L} \\ \text{R2L} \rightarrow @\text{R3}\end{array}\right\} \cdots\cdots \text{equation 1}$$

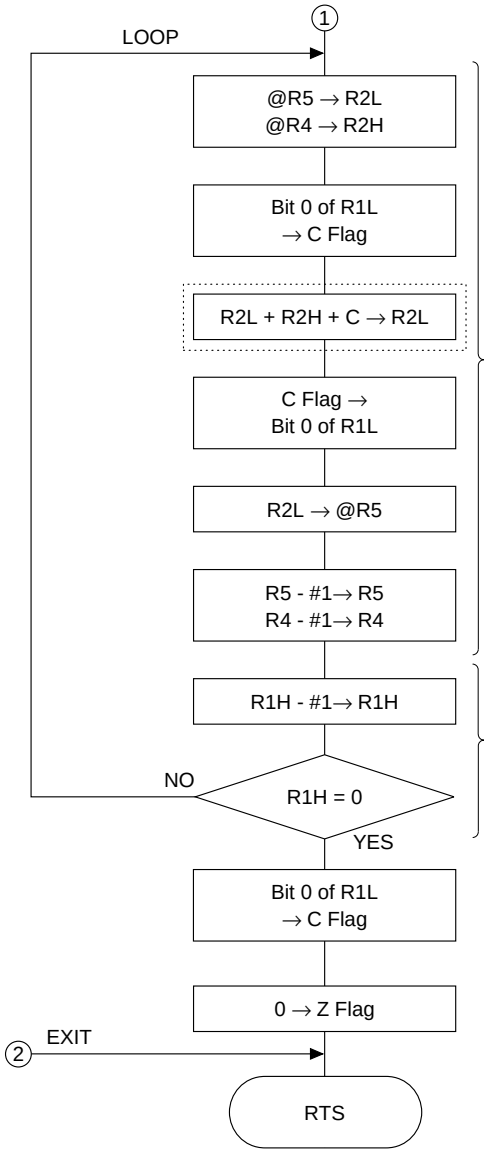
where the C flag indicates a carry that may occur in the result of addition of the lower bytes.

e. The result of d. is placed in the data memory area for the augend.

f. R3, R4, and R0L are decremented each time the process d. to e. terminates. This processing is repeated until R0L reaches 0.

7.5.7 Flowchart





Adds the augend, addend and carry and places the result in the data memory area where the augend was placed.

Repeats this process as many times as the byte count of the addition data.

Sets bit 0 of R1L in the C flag.

Clears the Z flag to 0.

Note: ADD2 is the same as SUB2, ADDD2, and SUBD2 except for the step surrounded by dotted lines.

7.5.8 Program List

```
*** H8/300 ASSEMBLER              VER 1.0B **   08/18/92 09:59:33
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :MULTIPLE PRECISION BINARY ADDITION
    4                                   ;*                            (ADD2)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L    (BYTE LENGTH OF ADDTION DATA)
    9                                   ;*                    R3     (START ADDRESS OF SUMMAND)
   10                                   ;*                    R4     (START ADDRESS OF ADDEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R3     (START ADDRESS OF RESULT)
   13                                   ;*                    Z flag OF CCR (Z=0;TRUE , Z=1;FALSE)
   14                                   ;*                    C flag OF CCR (C=0;TRUE , C=1;OVER FLOW)
   15                                   ;*
   16                                   ;*********************************************************************
   17                                   ;
   18 ADD2_cod C 0000                        .SECTION        ADD2_code,CODE,ALIGN=2
   19                                        .EXPORT         ADD2
   20                                   ;
   21 ADD2_cod C      00000000           ADD2 .EQU    $               ;Entry point
   22 ADD2_cod C 0000 F000                   MOV.B   #H'00,R0H       ;Clear R0H
   23 ADD2_cod C 0002 0C81                   MOV.B   R0L,R1H         ;Set byte counter(R1H)
   24 ADD2_cod C 0004 4722                   BEQ     EXIT            ;Branch if R0L=0
   25 ADD2_cod C 0006 0D35                   MOV.W   R3,R5           ;R3 -> R5
   26 ADD2_cod C 0008                   MAIN
   27 ADD2_cod C 0008 1B00                   SUBS.W  #1,R0           ;Decrement R0
   28 ADD2_cod C 000A 0905                   ADD.W   R0,R5           ;Set start address of summand(R5)
   29 ADD2_cod C 000C 0904                   ADD.W   R0,R4           ;Set start address of addend(R4)
   30 ADD2_cod C 000E F900                   MOV.B   #H'00,R1L       ;Clear R1L
   31 ADD2_cod C 0010                   LOOP
   32 ADD2_cod C 0010 685A                   MOV.B   @R5,R2L         ;Load summand to R2L
   33 ADD2_cod C 0012 6842                   MOV.B   @R4,R2H         ;Load addend to R2H
   34 ADD2_cod C 0014 7709                   BLD     #0,R1L          ;Load bit 0 of R1L to C flag
   35 ADD2_cod C 0016 0E2A                   ADDX.B  R2H,R2L         ;Addition
   36 ADD2_cod C 0018 6709                   BST     #0,R1L          ;Store C flag to bit 0 of R1L
   37 ADD2_cod C 001A 68DA                   MOV.B   R2L,@R5         ;Store result
   38 ADD2_cod C 001C 1B05                   SUBS.W  #1,R5           ;Decrement summand address(R5)
   39 ADD2_cod C 001E 1B04                   SUBS.W  #1,R4           ;Decrement addend address(R4)
   40 ADD2_cod C 0020 1A01                   DEC.B   R1H             ;Decrement byte counter(R1H)
   41 ADD2_cod C 0022 46EC                   BNE     LOOP            ;Branch if Z=0
   42                                   ;
   43 ADD2_cod C 0024 7709                   BLD     #0,R1L          ;Load bit 0 of R1L to c flag
   44 ADD2_cod C 0026 06FB                   ANDC.B  #H'FB,CCR       ;Clear Z flag
   45 ADD2_cod C 0028                   EXIT
   46 ADD2_cod C 0028 5470                   RTS
   47                                   ;
   48                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.6 Subtraction of Multiple-Precision Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: SUB2

7.6.1 Function

1. The software SUB2 subtracts a multiple-precision binary number from another multiple-precision binary number and places the result in the data memory where the minuend was set.
2. The arguments used with the software SUB2 are unsigned integers, each being up to 255 bytes long.

7.6.2 Arguments

Description		Memory area	Data length (bytes)
Input	Minuend and subtrahend byte count	R0L	1
	Start address of minuend	R3	2
	Start address of subtrahend	R4	2
Output	Start address of result	R3	2
	Error	Z flag (CCR)	
	Borrow	C flag (CCR)	

7.6.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	×	↕	×	×	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	↕	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.6.4 Specifications

Program memory (bytes)
42
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
7170
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.6.5 Notes

The clock cycle count (7170) in the specifications for subtraction of 255 bytes from 255 bytes.

7.6.6 Description

1. Details of functions
   a. The following arguments are used with the software SUB2:
      R0L: Contains, as an input argument, the byte count of a minuend and the byte count of a subtrahend in 2-digit hexadecimals.
      R3: Contains, as an input argument, the start address of the data memory area where the minuend is placed. After execution of the software SUB2, the start address of the result is placed in this register.
      R4: Contains, as an input argument, the start address of the data memory area where the subtrahend is placed.
      Z flag (CCR): Indicates an error in data length as an output argument.

Z flag = 0: The data byte count (R0L) was not 0.

Z flag = 1: The data byte count (R0L) was 0, indicating an error.

C flag (CCR): Determines the presence or absence of a borrow after software SUB2 execution as an output argument.

C flag = 0: No borrow occurred in the result.

C flag = 1: A borrow occurred in the result. (See figure 7.19)

b. Figure 7.18 shows an example of the software SUB2 being executed. When the input arguments are set as shown in (1), the result of subtraction is placed in the data memory area as shown in (2).

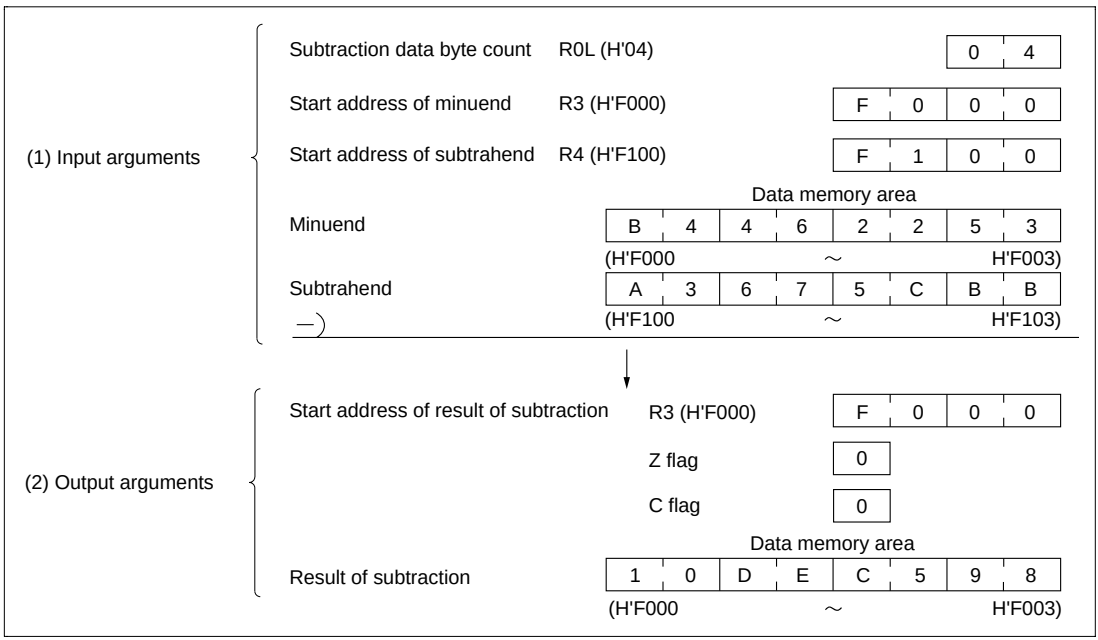


Figure 7.18 Example of Software SUB2 Execution

Figure 7.19 shows an example of subtraction with a borrow that has occurred in the result.

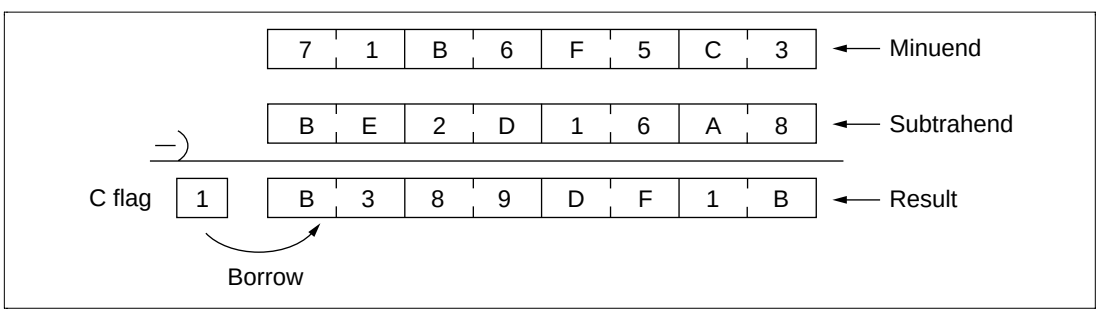


Figure 7.19 Example of Subtraction with a Borrow

2. Notes on usage
   a. When upper bits are not used (see figure 7.20), set 0's in them. The software SUB2 performs byte-based subtraction; if 0's are not set in the upper bits unused, no correct result can be obtained because the subtraction is done on the numbers including indeterminate data.

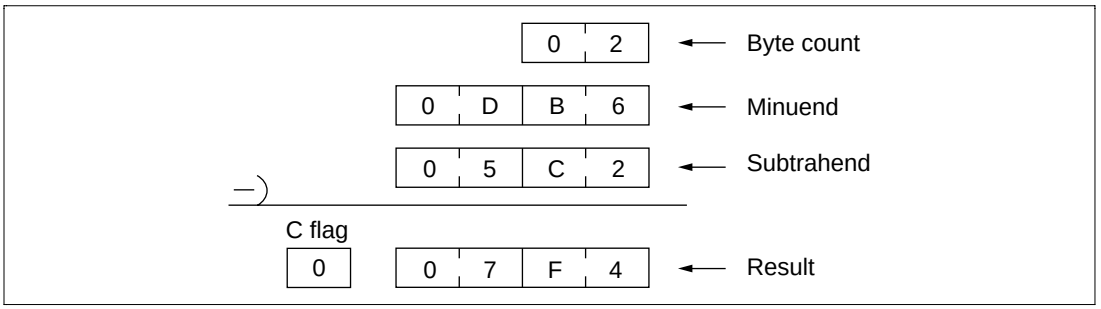
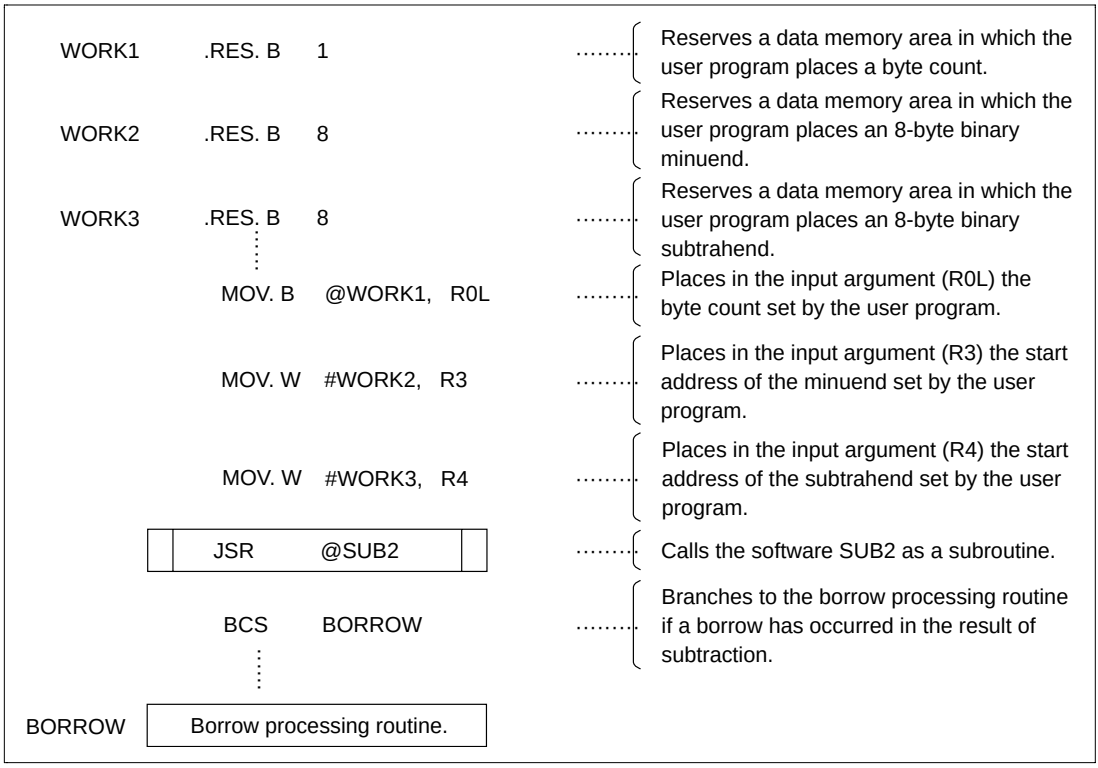


Figure 7.20 Example of Subtraction with Upper Bits Unused

   b. After execution of the software SUB2, the minuend is destroyed because the result is placed in the data memory area where the minuend was set. If the minuend is necessary after software SUB2 execution, save it on memory.
3. Data memory

   The software SUB2 does not use the data memory.
4. Example of use

   This is an example of subtracting 8 bytes of data. Set the start addresses of a byte count, a minuend and a subtrahend in the registers and call the software SUB2 as a subroutine.

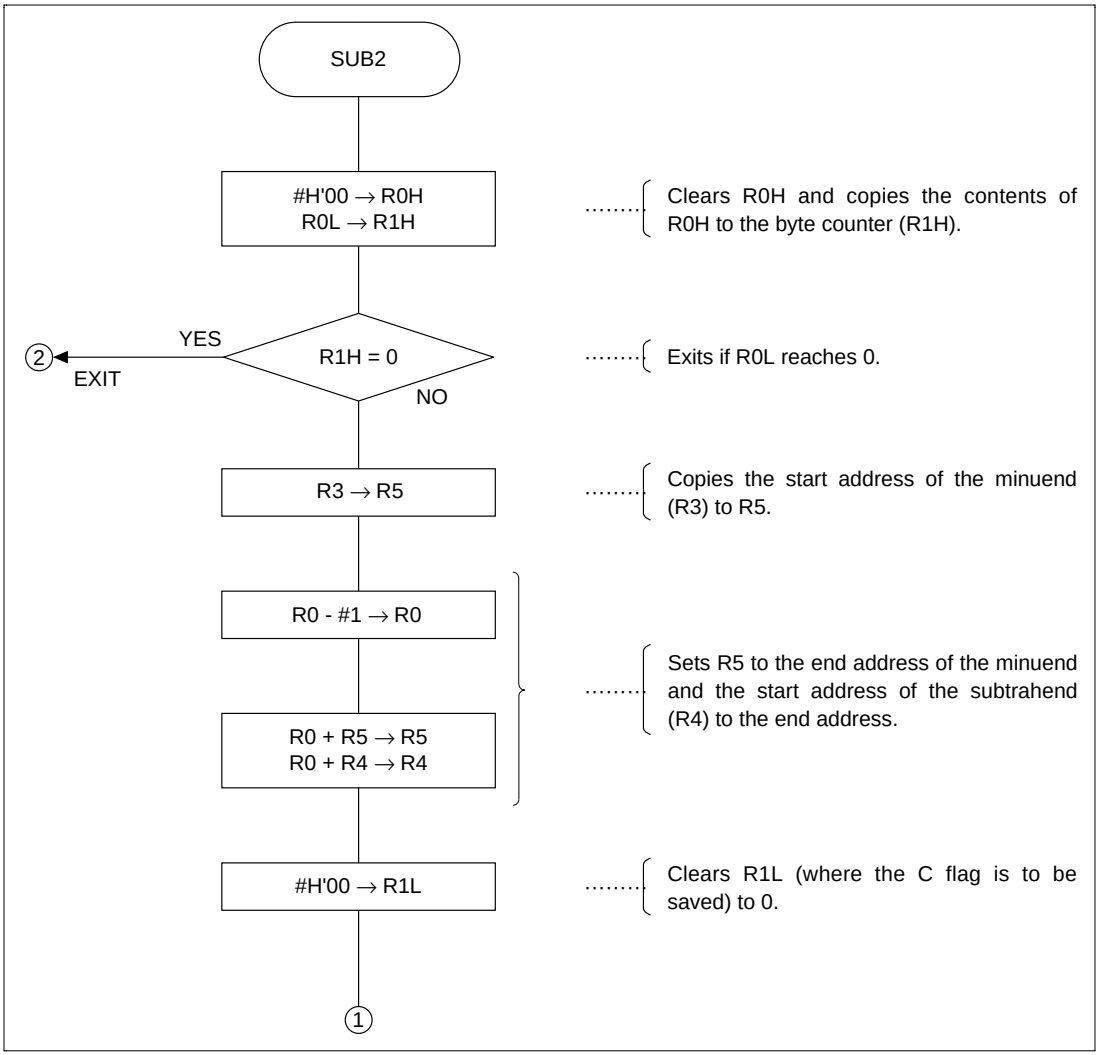


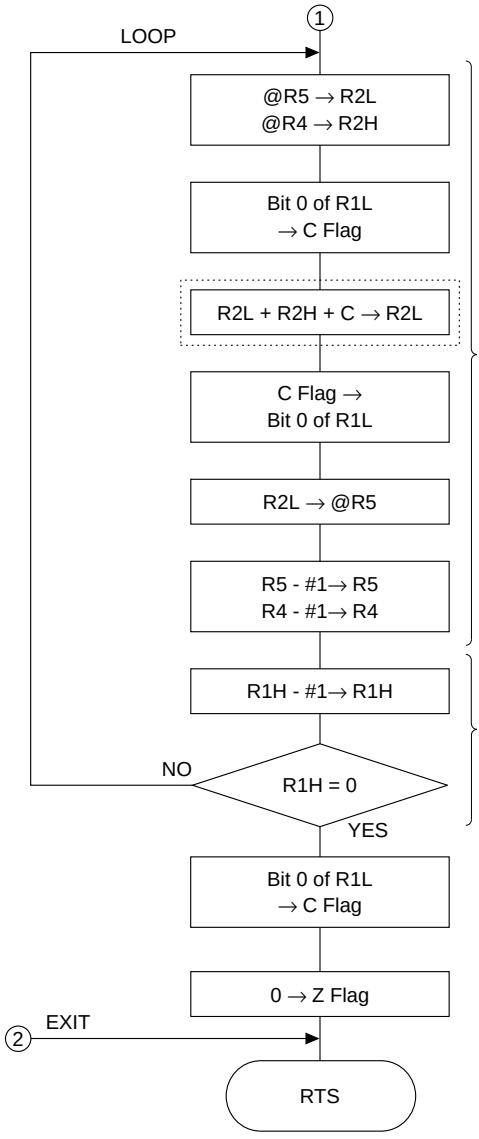
5. Operation
   a. Subtraction of multiple-precision binary numbers can be done by repeating a subtract instruction with a carry flag (SUBX.B) as the minuend and subtrahend data are placed in registers on a byte basis.
   b. The end address of the data memory area containing the minuend is placed in R3, and the end address of the data memory area containing the subtrahend is placed in R4.
   c. R1L is cleared for saving the C flag.
   d. The minuend and subtrahend are loaded in R2L and R2H respectively, byte by byte, starting at their end address and equation 1 is executed:

$$\left.\begin{array}{l}\text{Minuend} - \text{subtrahend} - \text{C} \rightarrow \text{R2L} \\ \text{R2L} \rightarrow @\text{R3}\end{array}\right\} \cdots\cdots \text{equation 1}$$

   where the C flag indicates a carry that may occur in the result of subtraction of the lower bytes.
   e. The result of d. is placed in the data memory area for the minuend.
   f. R3, R4, and R0L are decremented each time the process d. to e. terminates. This processing is repeated until R0L reaches 0.

7.6.7 Flowchart





Subtracts the subtrahend and borrow from the minuend and places the result in the data memory area where the minuend is placed.

Repeats this process as many times as the byte count of the subtraction data.

Sets bit 0 of R1L in the C flag.

Clears the Z flag to 0.

Note: SUB2 is the same as ADD2, ADDD2, and SUBD2 except for the step surrounded by dotted lines.

7.6.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:00:06
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :MULTIPLE-PRECISION BINARY SUBTRACTION
    4                                   ;*                            (SUB2)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L    (BYTE LENGTH OF DATA)
    9                                   ;*                    R3     (START ADDRESS OF MINUEND)
   10                                   ;*                    R4     (START ADDRESS OF SUBTRAHEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R3     (START ADDRESS OF RESULT)
   13                                   ;*                    Z flag OF CCR (Z=0;TRUE , Z=1;FALSE)
   14                                   ;*                    C flag OF CCR (C=0;TRUE , C=1;BORROW)
   15                                   ;*
   16                                   ;*********************************************************************
   17                                   ;
   18 SUB2_cod C 0000                        .SECTION        SUB2_code,CODE,ALIGN=2
   19                                        .EXPORT         SUB2
   20                                   ;
   21 SUB2_cod C      00000000          SUB2 .EQU    $               ;Entry point
   22 SUB2_cod C 0000 F000                   MOV.B   #H'00,R0H       ;Clear R0H
   23 SUB2_cod C 0002 0C81                   MOV.B   R0L,R1H         ;Set byte counter(R1H)
   24 SUB2_cod C 0004 4722                   BEQ     EXIT            ;Branch if R0L=0
   25 SUB2_cod C 0006 0D35                   MOV.W   R3,R5           ;R3 -> R5
   26 SUB2_cod C 0008                   MAIN
   27 SUB2_cod C 0008 1B00                   SUBS.W  #1,R0           ;Decrement R0
   28 SUB2_cod C 000A 0905                   ADD.W   R0,R5           ;Adjust minuend start address(R5)
   29 SUB2_cod C 000C 0904                   ADD.W   R0,R4           ;Adjust subtrahend start address(R4)
   30 SUB2_cod C 000E F900                   MOV.B   #H'00,R1L       ;Clear R1L
   31 SUB2_cod C 0010                   LOOP
   32 SUB2_cod C 0010 685A                   MOV.B   @R5,R2L         ;Load minuend
   33 SUB2_cod C 0012 6842                   MOV.B   @R4,R2H         ;Load subtrahend
   34 SUB2_cod C 0014 7709                   BLD     #0,R1L          ;Load bit 0 of R1L to C flag
   35 SUB2_cod C 0016 1E2A                   SUBX.B  R2H,R2L         ;Subtruction
   36 SUB2_cod C 0018 6709                   BST     #0,R1L          ;Store C flag to bit 0 of R1L
   37 SUB2_cod C 001A 68DA                   MOV.B   R2L,@R5         ;Store result
   38 SUB2_cod C 001C 1B05                   SUBS.W  #1,R5           ;Decrement minuend address
   39 SUB2_cod C 001E 1B04                   SUBS.W  #1,R4           ;Decrement subtrahend address
   40 SUB2_cod C 0020 1A01                   DEC.B   R1H             ;Decrement byte counter
   41 SUB2_cod C 0022 46EC                   BNE     LOOP            ;Branch if not R0L=0
   42                                   ;
   43 SUB2_cod C 0024 7709                   BLD     #0,R1L          ;Load bit 0 of R1L to C flag
   44 SUB2_cod C 0026 06FB                   ANDC    #H'FB,CCR       ;Clear Z flag
   45 SUB2_cod C 0028                   EXIT
   46 SUB2_cod C 0028 5470                   RTS
   47                                   ;
   48                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.7 Addition of 8-Digit BCD Numbers

MCU: H8/300 Series
H8/300L Series

Label name: ADDD1

7.7.1 Function

1. The software ADDD1 adds an 8-digit binary-coded decimal (BCD) number to another 8-digit BCD number and places the result (an 8-digit BCD number) in a general-purpose register.
2. The arguments used with the software ADDD1 are unsigned integers.
3. All data is manipulated in general-purpose registers.

7.7.2 Arguments

Description		Memory area	Data length (bytes)
Input	Augend	R0, R1	4
	Addend	R2, R3	4
Output	Result of addition	R0, R1	4
	Carry	C flag (CCR)	

7.7.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.7.4 Specifications

Program memory (bytes)
18
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
24
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.7.5 Description

1. Details of functions
   a. The following arguments are used with the software ADDD1:

   R0, R1: Contain an 8-digit BCD augend (32 bits long). After execution of the software ADDD1, the result of addition (an 8-digit BCD number, 32 bits long) is placed in this register.

   R2, R3: Contain an 8-digit BCD addend (32 bits long) as an input argument.

   C flag (CCR): Determines the presence or absence of a carry, as an output argument, after execution of the software ADDD1.

   C flag = 1: A carry occurred in the result of addition. (See figure 7.21.)

   C flag = 0: No carry occurred in the result of addition.

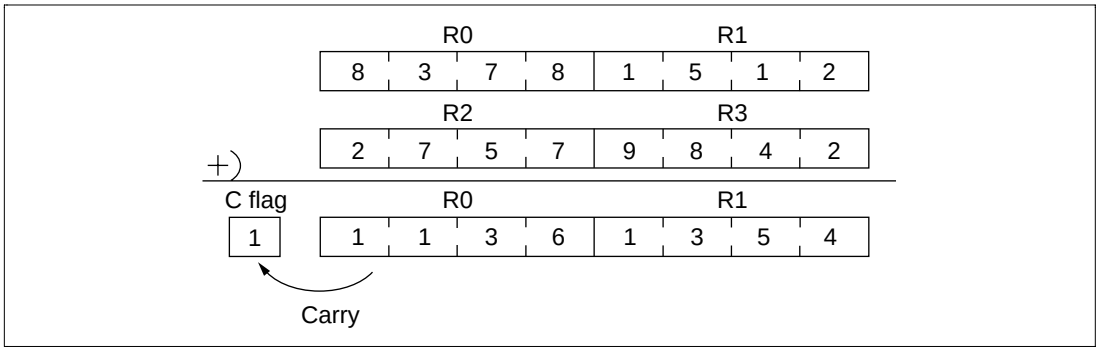


Figure 7.21 Example of Addition with a Carry

b. Figure 7.22 shows an example of the software ADDD1 being executed. When the input arguments are set as shown in (1), the result of addition is placed in R0 and R1 as shown in (2).

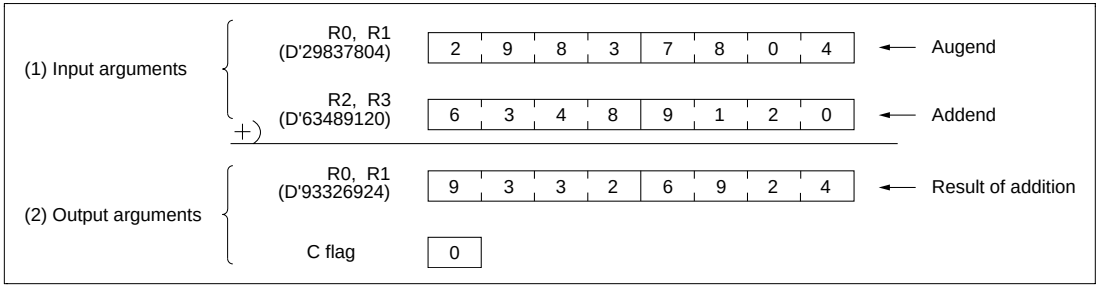


Figure 7.22 Example of Software ADDD1 Execution

2. Notes on usage

   a. When upper bits are not used (see figure 7.23), set 0's in them; otherwise, no correct result can be obtained because addition is done on the numbers including indeterminate data.

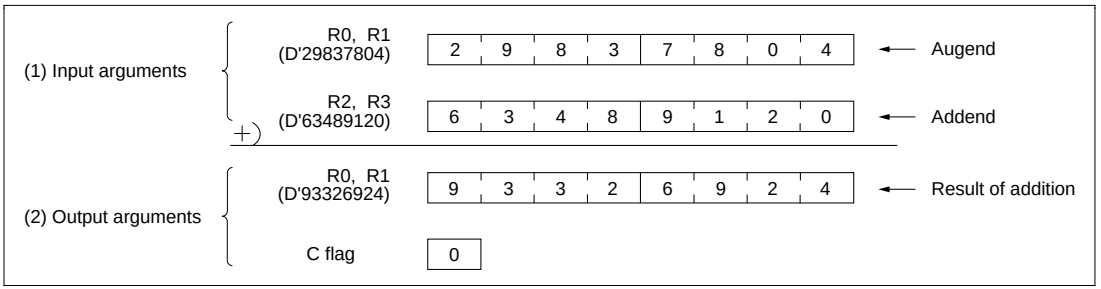


Figure 7.23 Example of Addition with Upper Bits Unused

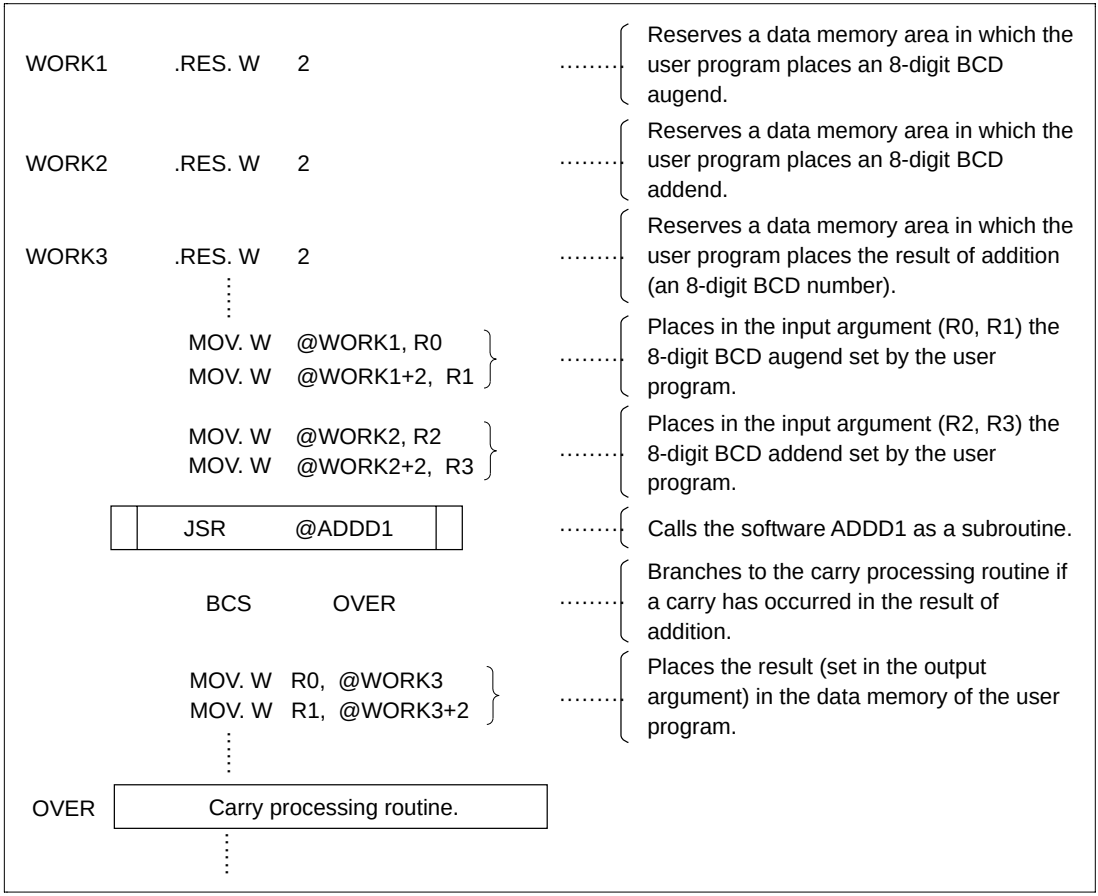
   b. After execution of the software ADDD1, the augend is destroyed because the result is placed in R1 and R2. If the augend is necessary after software ADDD1 execution, save it on memory.

3. Data memory

   The software ADDD1 does not use the data memory.

4. Example of use

Set an augend and an addend in the registers and call the software ADDD1 as a subroutine.



5. Operation

a. Addition of 2 bytes or more of BCD numbers can be done by performing a series of 1-byte additions with decimal correction.

b. A 1-byte add instruction (ADD.B), which does not involve the state of the C flag, is used to add the most significant byte given in equation 1. If a carry occurs after execution of equation 1, the C flag is set. Then a decimal correct instruction (DAA) is used to perform decimal correction.

R1L + R3L → R1L ····· equation 1

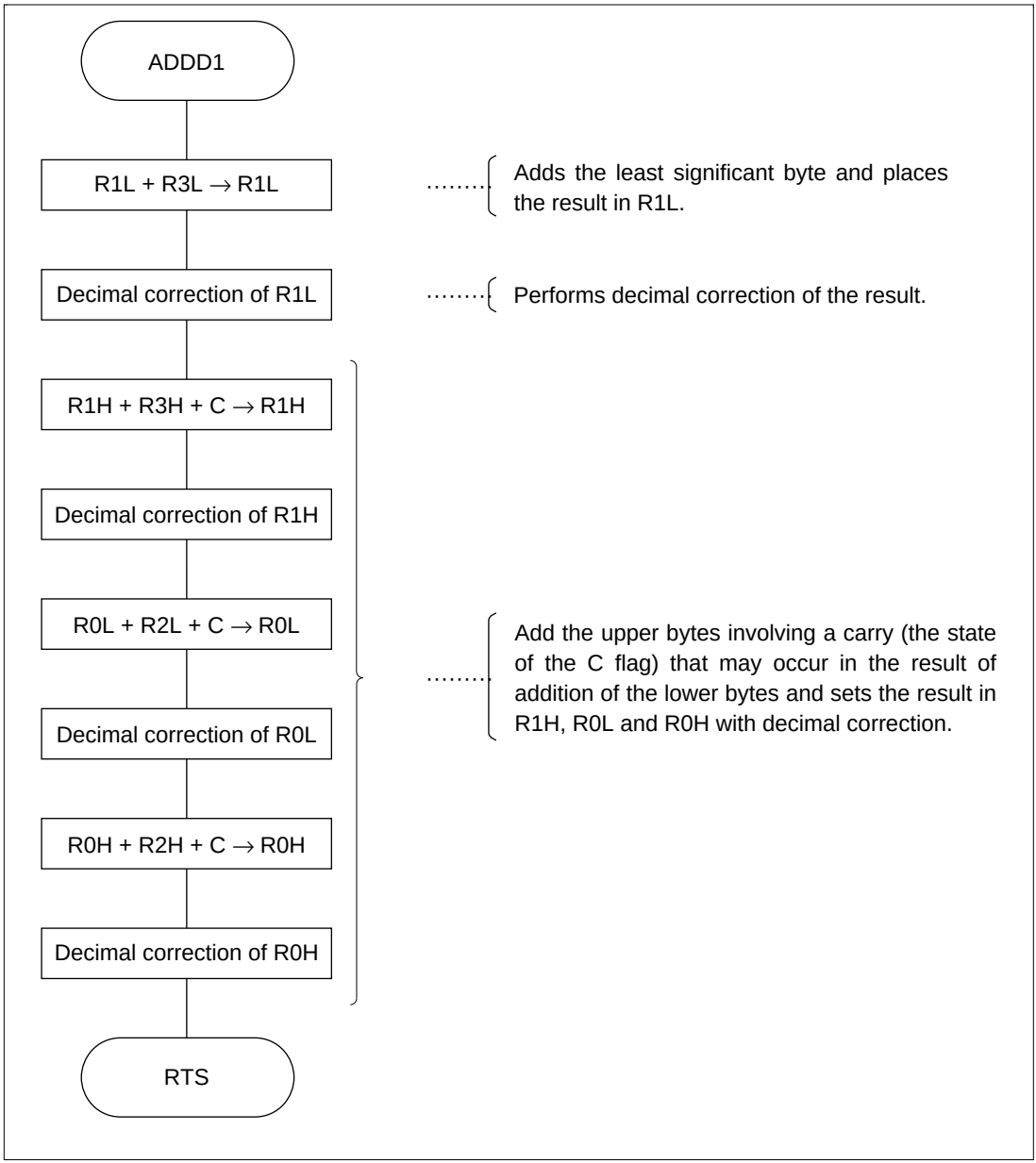
Decimal correction of R1L → R1L

c. A 1-byte add instruction (ADDX.B), which involves the state of the C flag, and a decimal-correct instruction are executed three times to add the upper bytes given in equation 2.

$$\left.\begin{array}{l} R1H + R3H + C \rightarrow R1H \quad \text{Decimal correction of } R1H \rightarrow R1H \\ R0H + R2L + C \rightarrow R0L \quad \text{Decimal correction of } R0L \rightarrow R0L \\ R0H + R2H + C \rightarrow R0H \quad \text{Decimal correction of } R0H \rightarrow R0H \end{array}\right\} \cdots\cdots\cdots \text{equation 2}$$

The C flag indicates a carry that may occur in the result of addition of the least significant byte, the upper bytes of the lower word, and the lower bytes of the upper word that was executed in step (b).

7.7.6 Flowchart



7.7.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:00:37
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :DECIMAL ADDITION (ADDD1)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     (UPPER WORD SUMMAND)
    8                                   ;*                    R1     (LOWER WORD SUMMAND)
    9                                   ;*                    R2     (UPPER WORD ADDEND)
   10                                   ;*                    R3     (LOWER WORD ADDEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R0     (UPPER WORD RESULT)
   13                                   ;*                    R1     (LOWER WORD RESULT)
   14                                   ;*                    C flag OF CCR   (C=0;TRUE , C=1;OVERFLOW)
   15                                   ;*
   16                                   ;*********************************************************************
   17                                   ;
   18 ADDD1_co C 0000                        .SECTION        ADDD1_code,CODE,ALIGN=2
   19                                        .EXPORT         ADDD1
   20                                   ;
   21 ADDD1_co C      00000000          ADDD1        .EQU    $               ;Entry point
   22 ADDD1_co C 0000 08B9                   ADD.B   R3L,R1L         ;R3L + R1L     -> R1L
   23 ADDD1_co C 0002 0F09                   DAA     R1L             ;Decimal adjust R1L
   24 ADDD1_co C 0004 0E31                   ADDX.B  R3H,R1H         ;R3H + R1H + C -> R1H
   25 ADDD1_co C 0006 0F01                   DAA     R1H             ;Decimal adjuxt R1H
   26 ADDD1_co C 0008 0EA8                   ADDX.B  R2L,R0L         ;R2L + R0L + C -> R0L
   27 ADDD1_co C 000A 0F08                   DAA     R0L             ;Decimal adjust R0H
   28 ADDD1_co C 000C 0E20                   ADDX.B  R2H,R0H         ;R2H + R0H + C -> R0H
   29 ADDD1_co C 000E 0F00                   DAA     R0H             ;Decimal adjust R0H
   30 ADDD1_co C 0010 5470                   RTS
   31                                   ;
   32                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.8 Subtraction of 8-Digit BCD Numbers

MCU: H8/300 Series
H8/300L Series

Label name: SUBD1

7.8.1 Function

1. The software SUBD1 subtracts an 8-digit binary-coded decimal (BCD) number from another 8-digit BCD number and places the result (an 8-digit BCD number) in a general-purpose register.
2. The arguments used with the software SUBD1 are unsigned integers.
3. All data is manipulated in general-purpose registers.

7.8.2 Arguments

Description		Memory area	Data length (bytes)
Input	Minuend	R0, R1	4
	Subtrahend	R2, R3	4
Output	Result of subtraction	R0, R1	4
	Borrow	C flag (CCR)	

7.8.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	•	•	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.8.4 Specifications

Program memory (bytes)
18
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
24
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.8.5 Description

1. Details of functions
   a. The following arguments are used with the software SUBD1:

   R0, R1: Contain an 8-digit BCD minuend (32 bits long). After execution of the software SUBD1, the result of subtraction (an 8-digit BCD number, 32 bits long) is placed in this register.

   R2, R3: Contain an 8-digit BCD subtrahend (32 bits long) as an input argument.

   C flag (CCR): Determines the presence or absence of a borrow, as an output argument, after execution of the software SUBD1.

   C flag = 1: A borrow occurred in the result of subtraction. (See figure 7.24.)

   C flag = 0: No borrow occurred in the result of subtraction.

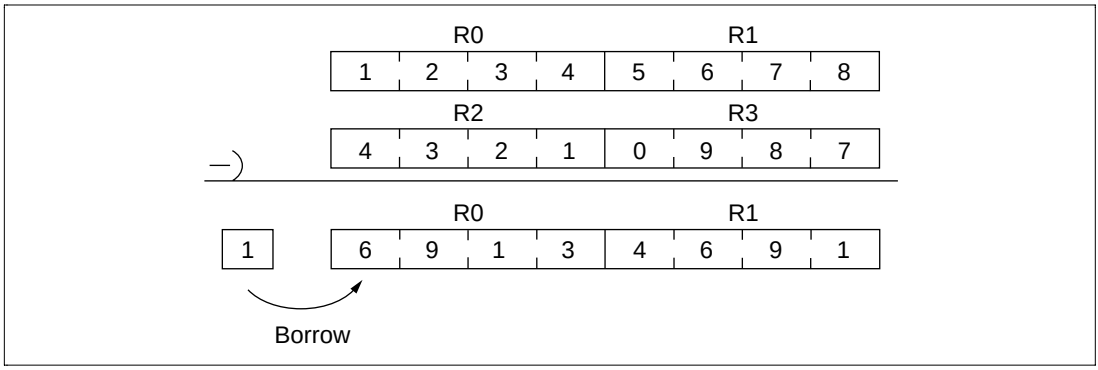


Figure 7.24 Example of Subtraction with a Borrow

b. Figure 7.25 shows an example of the software SUBD1 being executed. When the input arguments are set as shown in (1), the result of subtraction is placed in R0 and R1 as shown in (2).

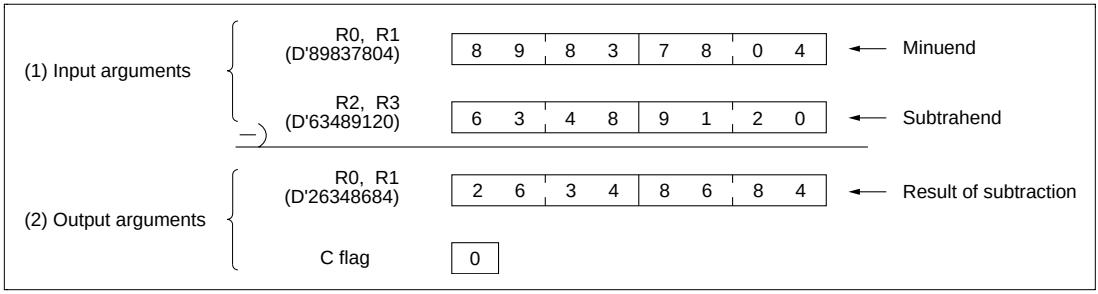


Figure 7.25 Example of Software SUBD1 Execution

2. Notes on usage
   a. When upper bits are not used (see figure 7.26), set 0's in them; otherwise, no correct result can be obtained because subtraction is done on the numbers including indeterminate data.

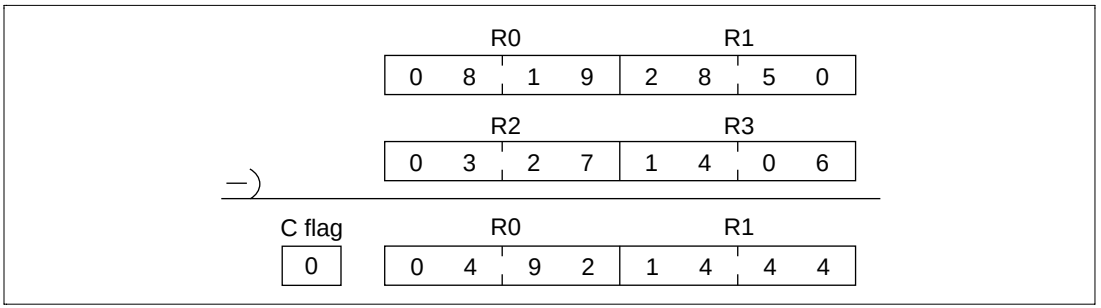


Figure 7.26 Example of Subtraction with Upper Bits Unused

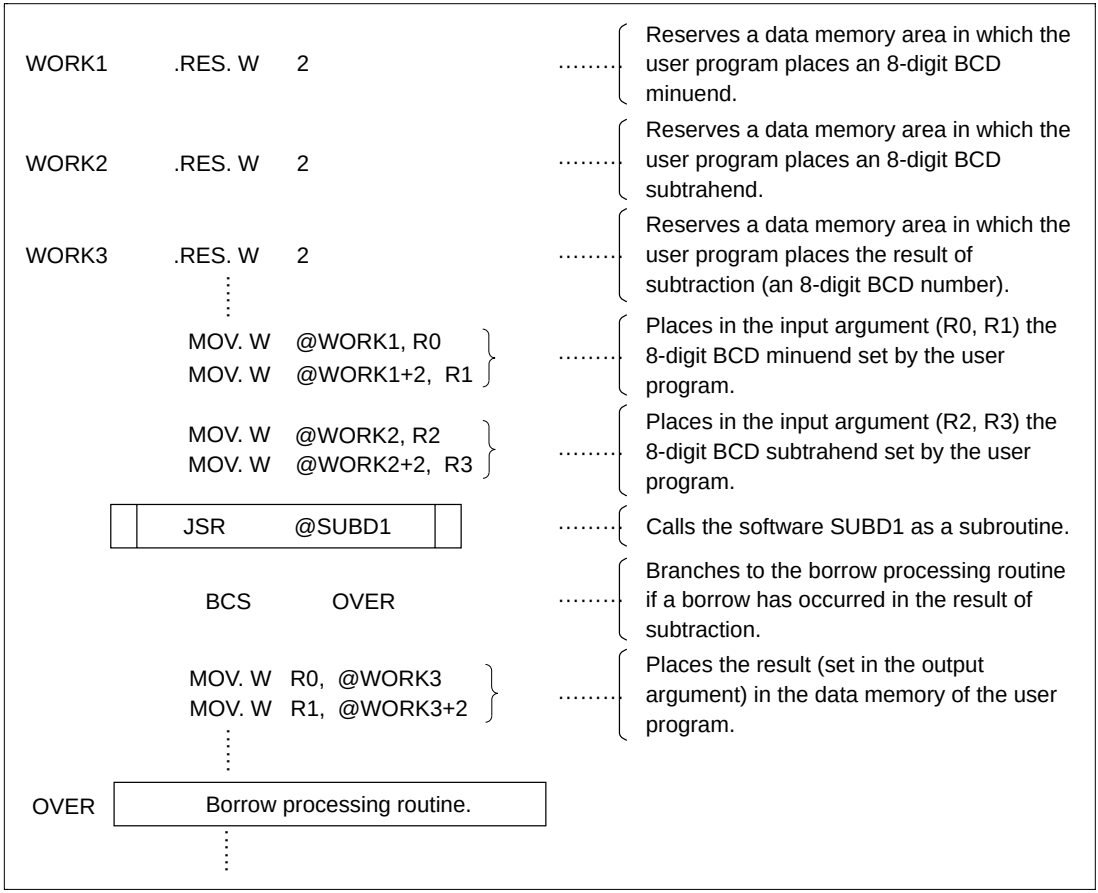
   b. After execution of the software SUBD1, the minuend is destroyed because the result is placed in R1 and R2. If the minuend is necessary after software SUBD1 execution, save it on memory.

3. Data memory

   The software SUBD1 does not use the data memory.

4. Example of use

Set a minuend and a subtrahend in the registers and call the software SUBD1 as a subroutine.



5. Operation

a. Subtraction of 2 bytes or more of BCD numbers can be done by performing a series of 1-byte subtractions with decimal correction.

b. A 1-byte subtract instruction (SUB.B), which does not involve the state of the C flag, is used to add the most significant byte given in equation 1. If a borrow occurs after execution of equation 1, the C flag is set. Then a decimal correct instruction (DAS) is used to perform decimal correction.

R1L – R3L → R1L .... equation 1

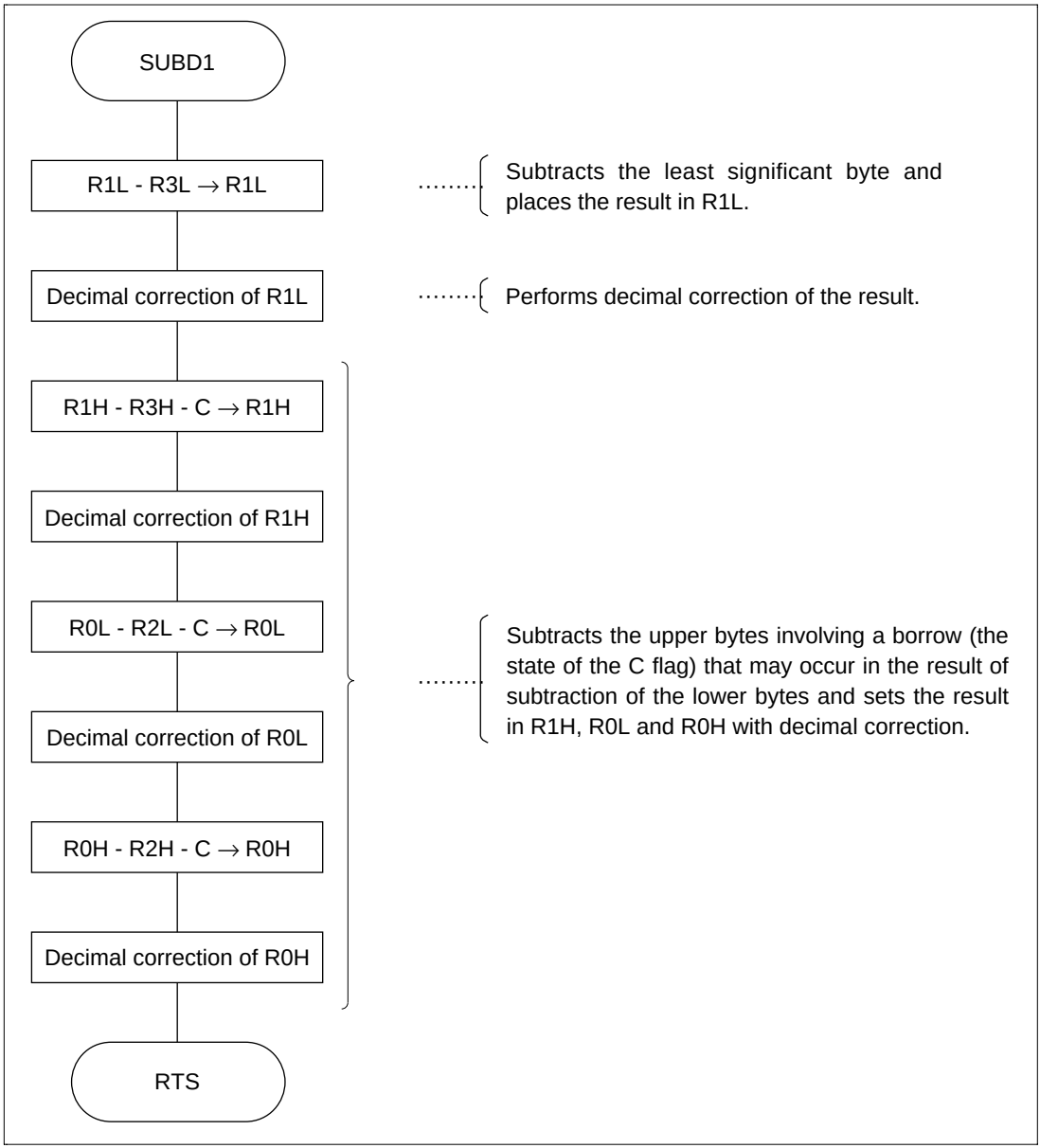
Decimal correction of R1L → R1L

c. A 1-byte subtract instruction (SUBX.B), which involves the state of the C flag, and a decimal-correct instruction (DAS) are executed three times to add the upper bytes given in equation 2.

$$\left.\begin{array}{l} R1H - R3H - C \rightarrow R1H \quad \text{Decimal correction of } R1H \rightarrow R1H \\ R0H - R2L - C \rightarrow R0L \quad \text{Decimal correction of } R0L \rightarrow R0L \\ R0H - R2H - C \rightarrow R0H \quad \text{Decimal correction of } R0H \rightarrow R0H \end{array}\right\} \cdots\cdots\cdots \text{equation 2}$$

The C flag indicates a borrow that may occur in the result of subtraction of the least significant byte, the upper bytes of the lower word, and the lower bytes of the upper word that was executed in step b..

7.8.6 Flowchart



7.8.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:01:03
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :DECIMAL SUBTRUCTION (SUBD1)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     (UPPER WORD MINUEND)
    8                                   ;*                    R1     (LOWER WORD MINUEND)
    9                                   ;*                    R2     (UPPER WORD SUBTRAHEND)
   10                                   ;*                    R3     (LOWER WORD SUBTRAHEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R0     (UPPER WORD RESULT)
   13                                   ;*                    R1     (LOWER WORD RESULT)
   14                                   ;*                    C flag OF CCR   (C=0;TRUE,C=1;UNDER FLOW)
   15                                   ;*
   16                                   ;*********************************************************************
   17                                   ;
   18 SUBD1_co C 0000                        .SECTION        SUBD1_code,CODE,ALIGN=2
   19                                        .EXPORT         SUBD1
   20                                   ;
   21 SUBD1_co C      00000000          SUBD1        .EQU    $               ;Entry point
   22 SUBD1_co C 0000 18B9                   SUB.B   R3L,R1L         ;R1L - R3L     -> R1L
   23 SUBD1_co C 0002 1F09                   DAS     R1L             ;Decimal adjust R1L
   24 SUBD1_co C 0004 1E31                   SUBX.B  R3H,R1H         ;R1H - R3H - C -> R1H
   25 SUBD1_co C 0006 1F01                   DAS     R1H             ;Decimal adjust R1H
   26 SUBD1_co C 0008 1EA8                   SUBX.B  R2L,R0L         ;R0L - R2L - C -> R0L
   27 SUBD1_co C 000A 1F08                   DAS     R0L             ;Decimal adjust R0L
   28 SUBD1_co C 000C 1E20                   SUBX.B  R2H,R0H         ;R0H - R2H - C -> R0H
   29 SUBD1_co C 000E 1F00                   DAS     R0H             ;Decimal adjust R0H
   30 SUBD1_co C 0010 5470                   RTS
   31                                   ;
   32                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.9 Multiplication of 4-Digit BCD Numbers

MCU: H8/300 Series
H8/300L Series

Label name: MULD

7.9.1 Function

1. The software MULD multiplies a 4-digit binary-coded decimal (BCD) number by another 4-digit BCD number and places the result (an 8-digit BCD number) in a general-purpose register.
2. The arguments used with the software MULD are unsigned integers.
3. All data is manipulated in general-purpose registers.

7.9.2 Arguments

Description		Memory area	Data length (bytes)
Input	Multiplicand	R1	2
	Multiplier	R0	2
Output	Result of multiplication	R2, R3	4

7.9.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5H	R5L	R6H	R6L	R7
×	•	↕	↕	•	•	×	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.9.4 Specifications

Program memory (bytes)
62
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
1192
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.9.5 Notes

The clock cycle count (1192) in the specifications is for multiplication of 9999 by 9999.

7.9.6 Description

1. Details of functions
   a. The following arguments are used with the software MULD:
      R0: Contains a 4-digit BCD multiplier (16 bits long) as an input argument.
      R1: Contains a 4-digit BCD multiplicand (16 bits long) as an input argument.
      R2: Contains the upper 4 digits of the result (an 8-digit BCD, 32 bits long) as an output argument.
      R3: Contains the lower 4 digits of the result (an 8-digit BCD, 32 bits long) as an output argument.

b. Figure 7.27 shows an example of the software MULD being executed. When the input arguments are set as shown in (1), the result of multiplication is places in R2 and R3 as shown in (2).

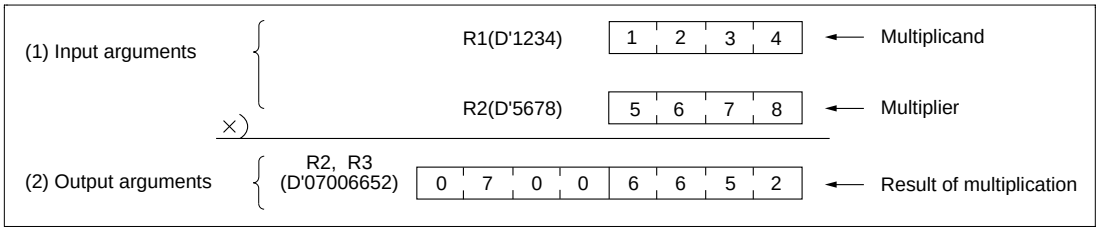


Figure 7.27 Example of Software MULD Execution

c. Table 7.3 lists the result of multiplication with 0's placed in input arguments.

Table 7.3 Result of Multiplication with 0's Placed in Input Arguments

Input arguments		**Output arguments**
Multiplicand	Multiplier	Product
H'****	H'0000	H'0000
H'0000	H'****	H'0000
H'0000	H'0000	H'0000

Note: H'**** is a hexadecimal number.

2. Notes on usage
   a. When upper bits are not used (see figure 7.28), set 0's in them; otherwise, no correct result can be obtained because multiplication is done on the numbers including indeterminate data placed in the upper bytes.

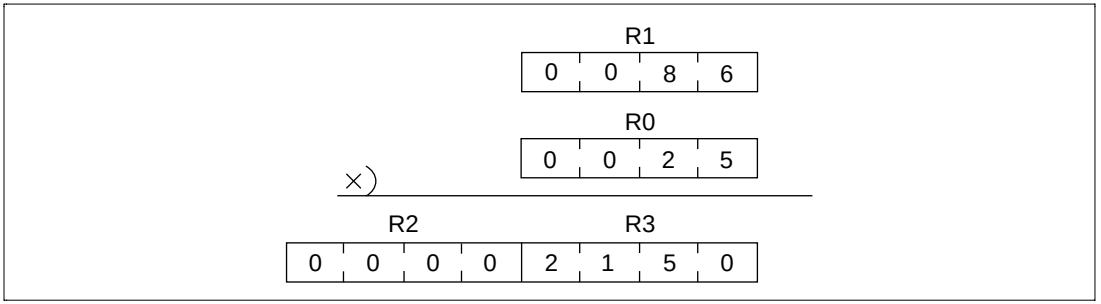


Figure 7.28 Example of Multiplication with Upper Bits Unused

   b. After execution of the software MULD, the multiplier is destroyed. If the multiplier is necessary after software MULD execution, save it on memory.
3. Data memory

   The software MULD does not use the data memory.

4. Example of use

Set a multiplicand and a multiplier in the registers and call the software MULD as a subroutine.

```
WORK1   .RES. W   1          ..... Reserves a data memory area in which the user program places a 4-digit BCD multiplicand.
WORK2   .RES. W   1          ..... Reserves a data memory area in which the user program places a 4-digit BCD multiplier.
WORK3   .RES. W   2          ..... Reserves a data memory area in which the user program places the result of multiplication (an 8-digit BCD number).
          :
        MOV. W    @WORK1, R1 ..... Places in the input argument (R1) the 4-digit BCD multiplicand set by the user program.
        MOV. W    @WORK2, R0 ..... Places in the input argument (R0) the 4-digit BCD multiplier set by the user program.
        JSR       @MULD      ..... Calls the software MULD as a subroutine.
        MOV. W    R2, @WORK3 ..... Places the result (set in the output argument) in the data memory of the user program.
        MOV. W    R3, @WORK3+2
          :
```

5. Operation

a. Multiplication of decimal numbers can be can be done by performing a series of additions and shifts. Figure 7.29 shows an example of multiplication (568 × 1234).

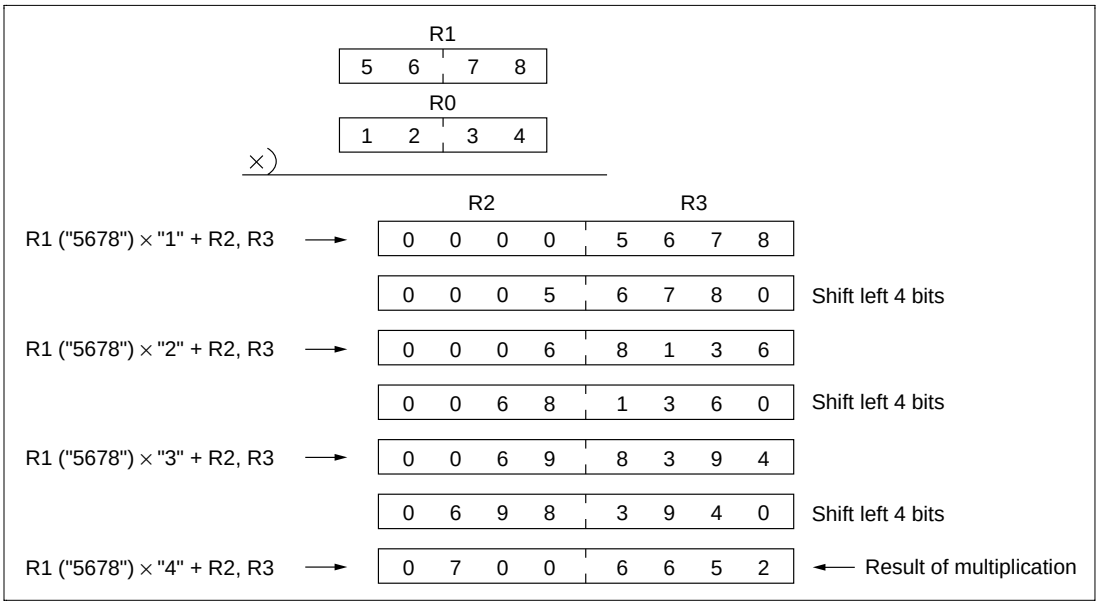


Figure 7.29 Example of Multiplication (5678 × 1234)

Figure 7.29 indicates that a product is obtained by performing a series of shifting the result of multiplication and adding the multiplicand.

First, 4 bits (1 digit of the BCD) are taken out of the most significant byte of the multiplier and the multiplicand is added by that value. The result is shifted 4 bits (1 digit of the BCD). Next, 4 bits are taken out of the upper byte of the multiplier and the multiplicand is added by that value. The result is added to the previously obtained result. By performing this series of operations as many times as the number of digits of the BCD (that is, four times) the final result of multiplication can be obtained.

b. The program runs in the following steps:

(i) H'04 is placed in the H6H counter that indicates the number of digits of data.

(ii) The result of multiplication (R2 and R3) is cleared.

(iii) R2 and R3 are shifted 4 bits (1 digit of the BCD) to the left.

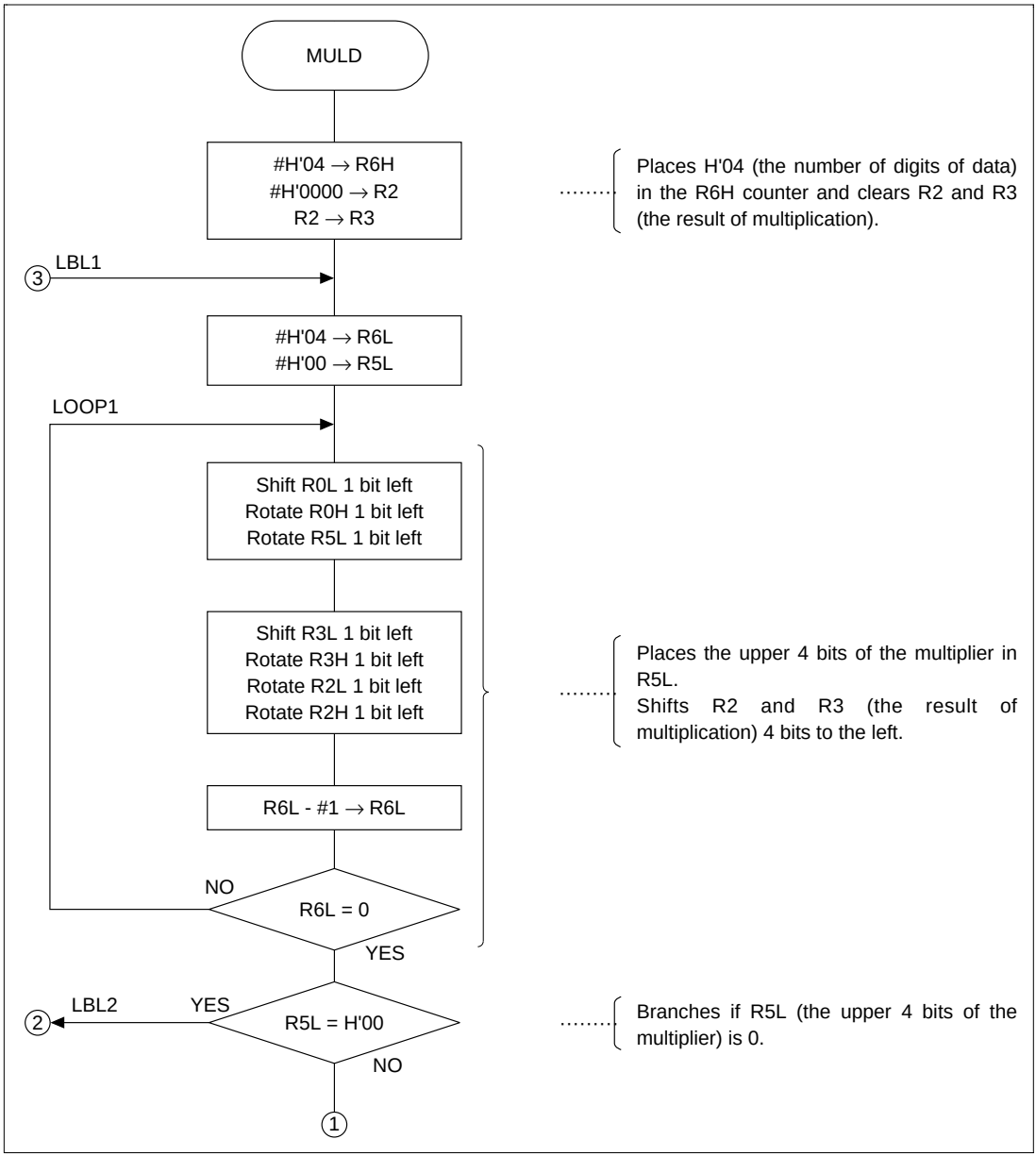
(iv) The multiplier is loaded in units of 4 bits (1 digit of the BCD) to R5, starting at its upper bytes. Branches to step (vi) if R5L is 0.

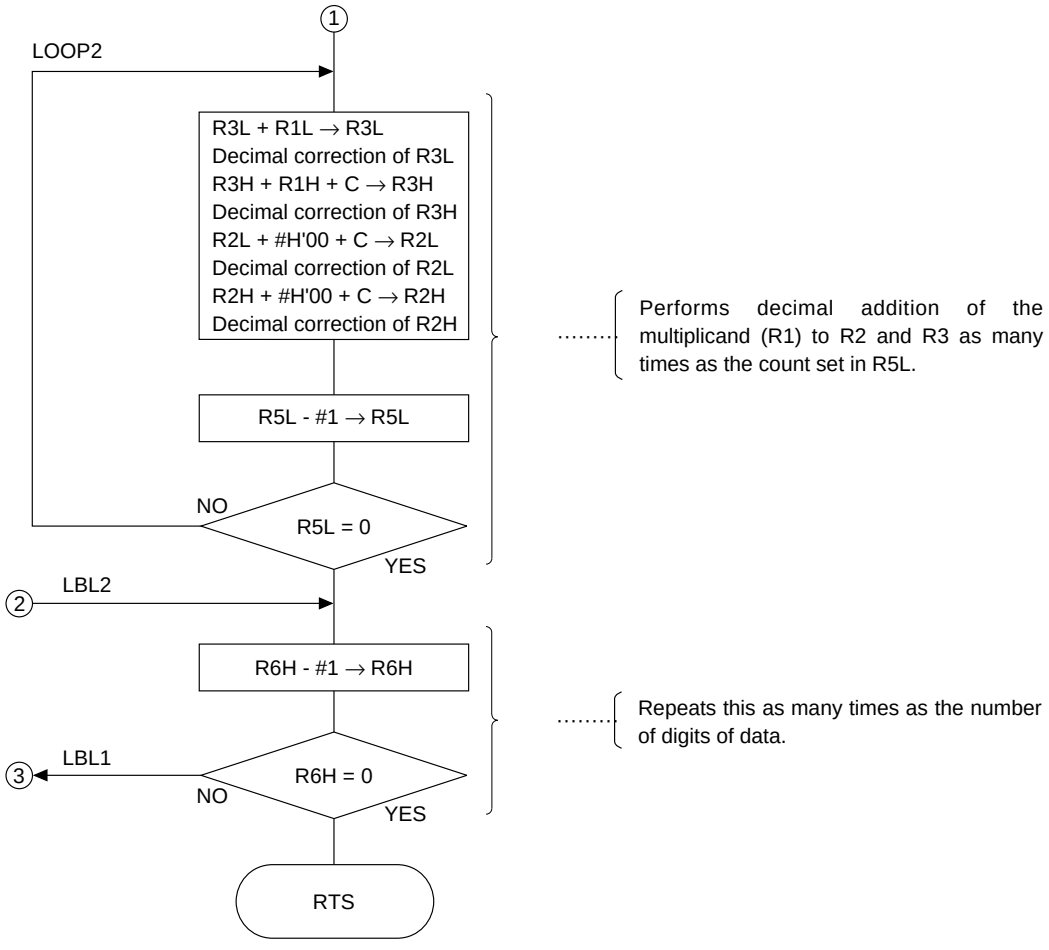
(v) Decimal addition of the multiplicand to R2 and R3 is performed by the value of R5L.

(vi) R6H is decremented.

(vii) Steps (iii) to (vi) are repeated until R6H reaches 0.

7.9.7 Flowchart




1
LOOP2
R3L + R1L → R3L
Decimal correction of R3L
R3H + R1H + C → R3H
Decimal correction of R3H
R2L + #H'00 + C → R2L
Decimal correction of R2L
R2H + #H'00 + C → R2H
Decimal correction of R2H
R5L - #1 → R5L
NO
R5L = 0
YES
LBL2
2
R6H - #1 → R6H
LBL1
3
R6H = 0
NO
YES
RTS
Performs decimal addition of the multiplicand (R1) to R2 and R3 as many times as the count set in R5L.
Repeats this as many times as the number of digits of data.

7.9.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:01:29
PROGRAM NAME =
    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME       :DECIMAL MULTIPLICATION
    4                                   ;*                    (MULD)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY   :R1     (MULTIPLICAND)
    9                                   ;*            R0     (MULTIPLIER)
   10                                   ;*
   11                                   ;*   RETURNS :R2     (UPPER WORD OF RESULT)
   12                                   ;*            R3     (LOWER WORD OF RESULT)
   13                                   ;*
   14                                   ;*********************************************************************
   15                                   ;
   16 MULD_cod C 0000                        .SECTION        MULD_code,CODE,ALIGN=2
   17                                        .EXPORT         MULD
   18                                   ;
   19 MULD_cod C      00000000          MULD .EQU    $               ;Entry point
   20 MULD_cod C 0000 F604                   MOV.B   #H'04,R6H       ;Set bit counter1
   21 MULD_cod C 0002 79020000               MOV.W   #H'0000,R2      ;Clear R2
   22 MULD_cod C 0006 0D23                   MOV.W   R2,R3           ;Clear R3
   23 MULD_cod C 0008                   LBL1
   24 MULD_cod C 0008 FE04                   MOV.B   #H'04,R6L       ;Set bit counter2
   25 MULD_cod C 000A FD00                   MOV.B   #H'00,R5L       ;Clear R5L
   26 MULD_cod C 000C                   LOOP1
   27 MULD_cod C 000C 1008                   SHLL.B  R0L             ;Shift multiplier 1 bit left
   28 MULD_cod C 000E 1200                   ROTXL.B R0H
   29 MULD_cod C 0010 120D                   ROTXL.B R5L
   30 MULD_cod C 0012 100B                   SHLL.B  R3L             ;Shift result 1 bit left
   31 MULD_cod C 0014 1203                   ROTXL.B R3H
   32 MULD_cod C 0016 120A                   ROTXL.B R2L
   33 MULD_cod C 0018 1202                   ROTXL.B R2H
   34 MULD_cod C 001A 1A0E                   DEC.B   R6L             ;Decrement bit counter 2
   35 MULD_cod C 001C 46EE                   BNE     LOOP1           ;Branch if Z=0
   36 MULD_cod C 001E AD00                   CMP.B   #H'00,R5L
   37 MULD_cod C 0020 4714                   BEQ     LBL2            ;Branch if Z=1
   38 MULD_cod C 0022                   LOOP2
   39 MULD_cod C 0022 089B                   ADD.B   R1L,R3L         ;R1L + R3L       -> R1L
   40 MULD_cod C 0024 0F0B                   DAA.B   R3L             ;Decimal adjust R3L
   41 MULD_cod C 0026 0E13                   ADDX.B  R1H,R3H         ;R1H + R3H + C   -> R1H
   42 MULD_cod C 0028 0F03                   DAA.B   R3H             ;Decimal adjust R3H
   43 MULD_cod C 002A 9A00                   ADDX.B  #H'00,R2L       ;R2L + #H'00 + C -> R2L
   44 MULD_cod C 002C 0F0A                   DAA.B   R2L             ;Decimal adjust R2L
   45 MULD_cod C 002E 9200                   ADDX.B  #H'00,R2H       ;R2H + #H'00 + C -> R2H
   46 MULD_cod C 0030 0F02                   DAA.B   R2H             ;Decimal adjust R2H
   47 MULD_cod C 0032 1A0D                   DEC.B   R5L             ;Clear bit 0 of R5L
   48 MULD_cod C 0034 46EC                   BNE     LOOP2           ;Branch if Z=0
   49 MULD_cod C 0036                   LBL2
   50 MULD_cod C 0036 1A06                   DEC.B   R6H             ;Decrement bit counter1
   51 MULD_cod C 0038 46CE                   BNE     LBL1            ;Branch if Z=0
   52                                   ;
   53 MULD_cod C 003A 5470                   RTS
   54                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.10 Division of 8-Digit BCD Numbers

MCU: H8/300 Series
H8/300L Series

Label name: DIVD

7.10.1 Function

1. The software DIVD divides an 8-digit binary-coded decimal (BCD) number by another 8-digit BCD number and places the result (an 8-digit BCD number) in a general-purpose register.
2. The arguments used with the software DIVD are unsigned integers.
3. All data is manipulated in general-purpose registers.

7.10.2 Arguments

Description		Memory area	Data length (bytes)
Input	Dividend	R0, R1	4
	Divisor	R2, R3	4
Output	Result of division (quotient)	R0, R1	4
	Result of division (remainder)	R4, R5	4
	Error	Z flag (CCR)	1

7.10.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	•	•	↕	↕	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	↕	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.10.4 Specifications

Program memory (bytes)
84
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
1162
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.10.5 Notes

The clock cycle count (1162) in the specifications is for division of 99999999 by 9999.

7.10.6 Description

1. Details of functions
   a. The following arguments are used with the software DIVD:
      R0: Contains the upper 4 digits of an 8-digit BCD dividend (32 bits long). After execution of the software DIVD, the upper 4 digits of the result of division (quotient) are placed in this register.
      R1: Contains the lower 4 bits of the 8-digit BCD dividend (32 bits long). After execution of the software DIVD, the lower 4 digits of the result of division (quotient) are placed in this register.
      R2: Contains the upper 4 digits of an 8-digit BCD divisor as an input argument.
      R3: Contains the lower 4 digits of the 8-digit BCD divisor as an input argument.
      R4: The upper 4 digits of an 8-digit BCD remainder are placed in this register as an output argument.
      R5: The lower 4 digits of the 8-digit BCD remainder are placed in this register as an output argument.
      Z flag (CCR): Determines the presence or absence of an error (division by 0) with the software DIVD as an output argument.
      Z flag = 1: The divisor was 0, indicating an error.
      Z flag = 0: The divisor was not 0.
   b. Figure 7.30 shows an example of the software DIVD being executed. When the input arguments are set as shown in (1), the result of division is placed in the registers as shown in (2).

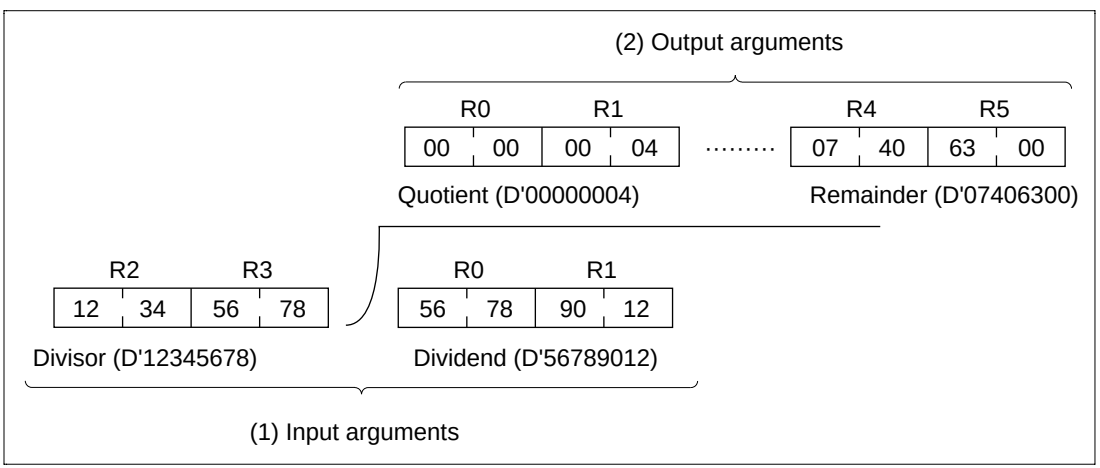


Figure 7.30 Example of Software DIVD Execution

c. Table 7.4 lists the result of division with 0's placed in input arguments.

Table 7.4 Result of Division with 0's Placed in Input Arguments

Input arguments		Output arguments		
Dividend (R0, R1)	Divisor (R2, R3)	Quotient (R0, R1)	Remainder (R4, R5)	Error (Z)
H'********	H'00000000	H'********	H'00000000	1
H'00000000	H'********	H'00000000	H'00000000	0
H'00000000	H'00000000	H'00000000	H'00000000	1

Note: H'**** is a hexadecimal number.

2. Notes on usage
   a. When upper bits are not used (see figure 7.31), set 0's in them; otherwise, no correct result can be obtained because division is done on the numbers including indeterminate data placed in the upper bits.

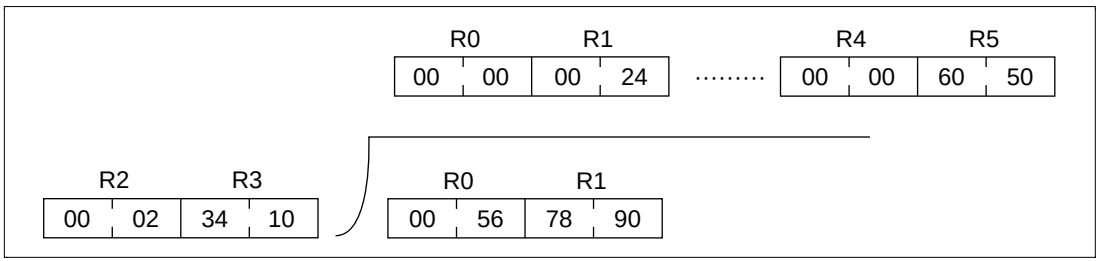


Figure 7.31 Example of Division with Upper Bits Unused

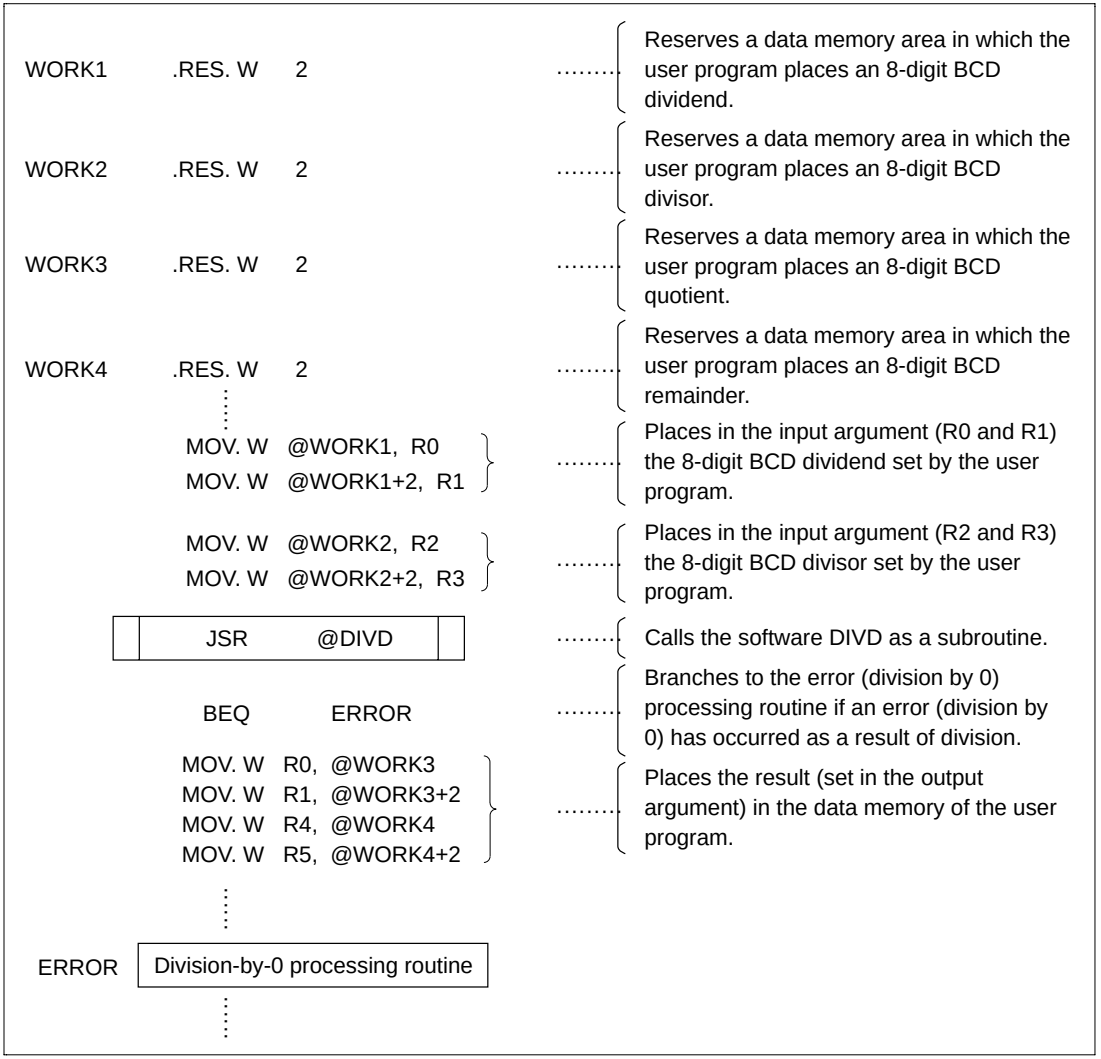
   b. After execution of the software DIVD, the dividend is destroyed because the quotient is placed in R0 and R1. If the dividend is necessary after software DIVD execution, save it on memory.

3. Data memory

   The software DIVD does not use the data memory.

4. Example of use

Set a dividend and a divisor in the registers and call the software DIVD as a subroutine.



5. Operation

a. Division of decimal numbers can be done by performing a series of subtractions. Figure 7.32 shows an example of division (64733088 ÷ 5).

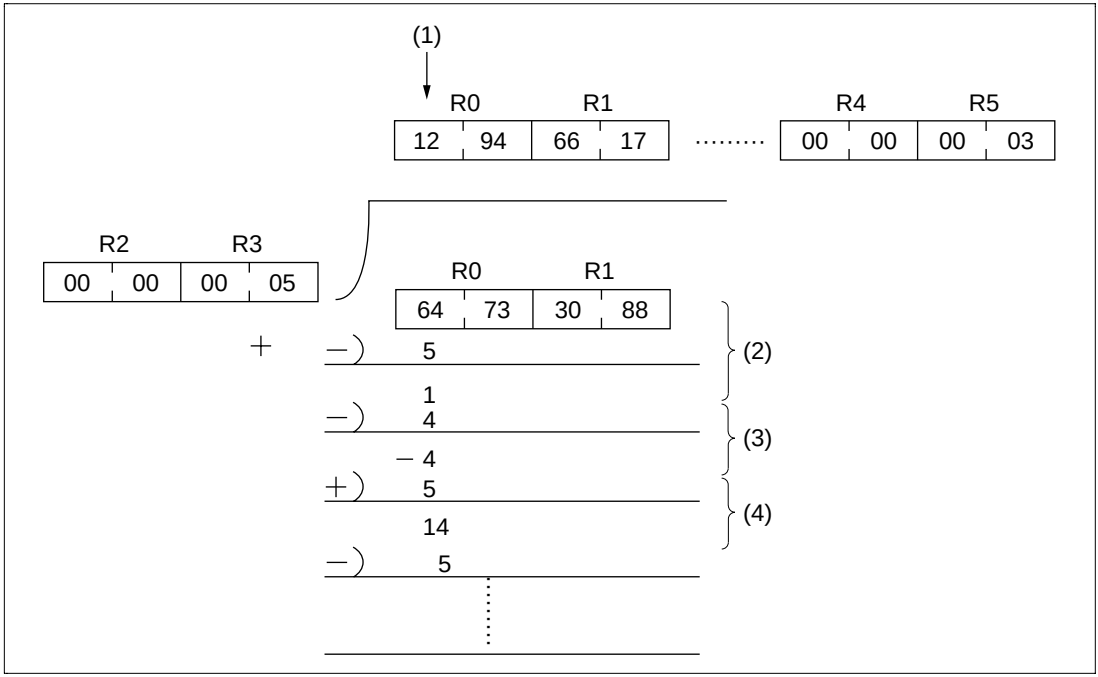


Figure 7.32 Example of Division (64733088 ÷ 5)

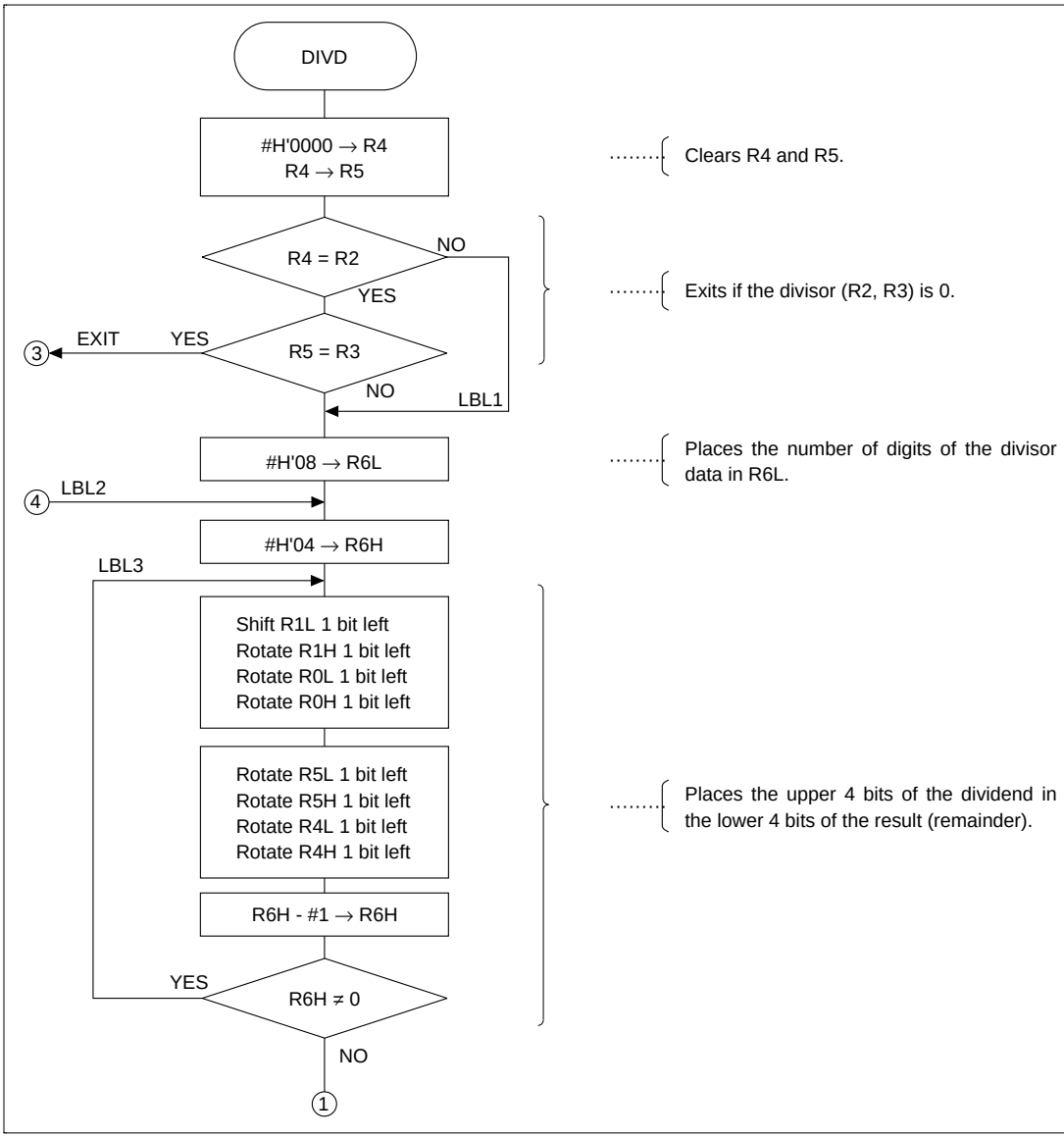
b. The program runs in the following steps:

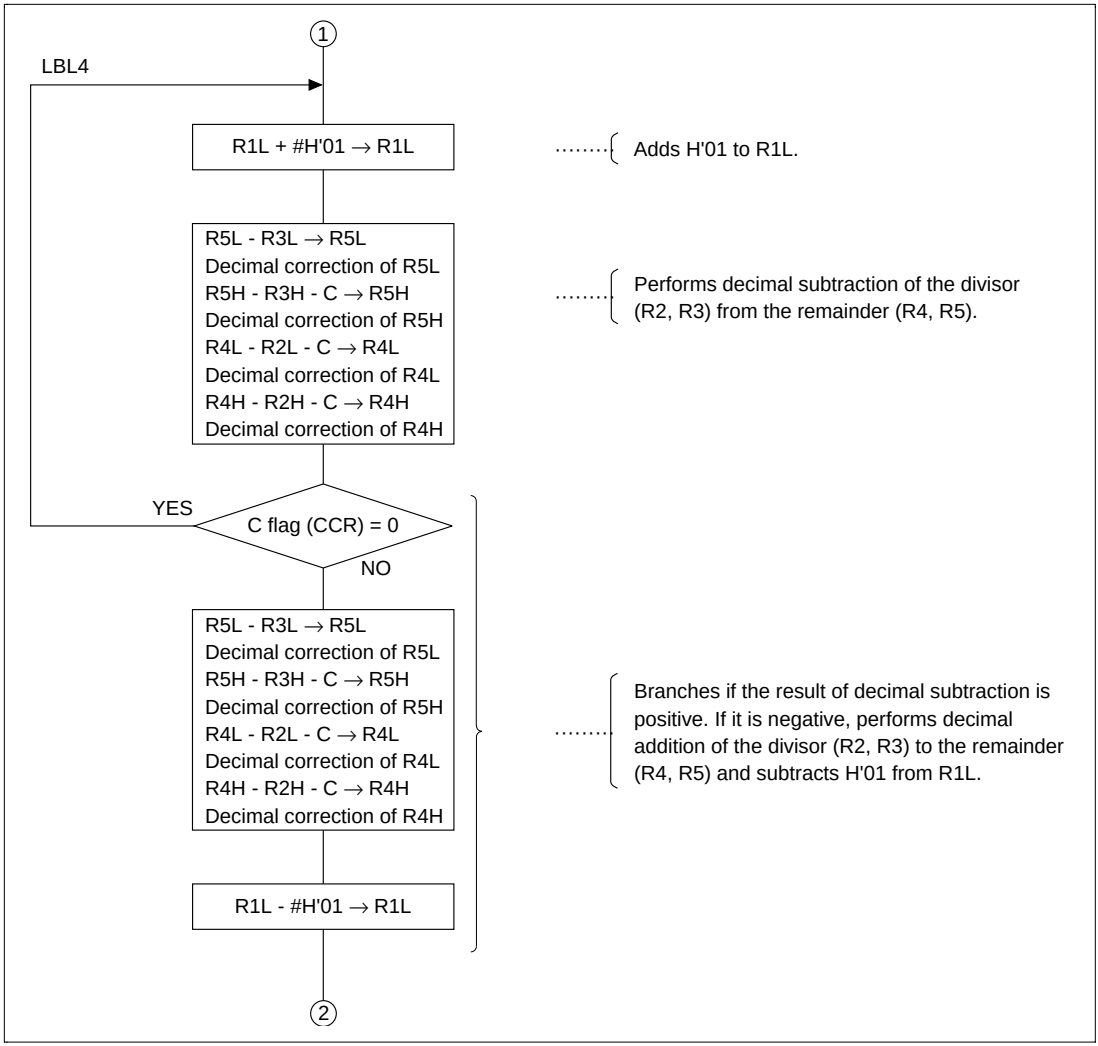
(i) The dividend is shifted 4 bits (1 digit of the BCD) to the left to place the upper 4 bits of the dividend in the lower 4 bits of the result of division (remainder).

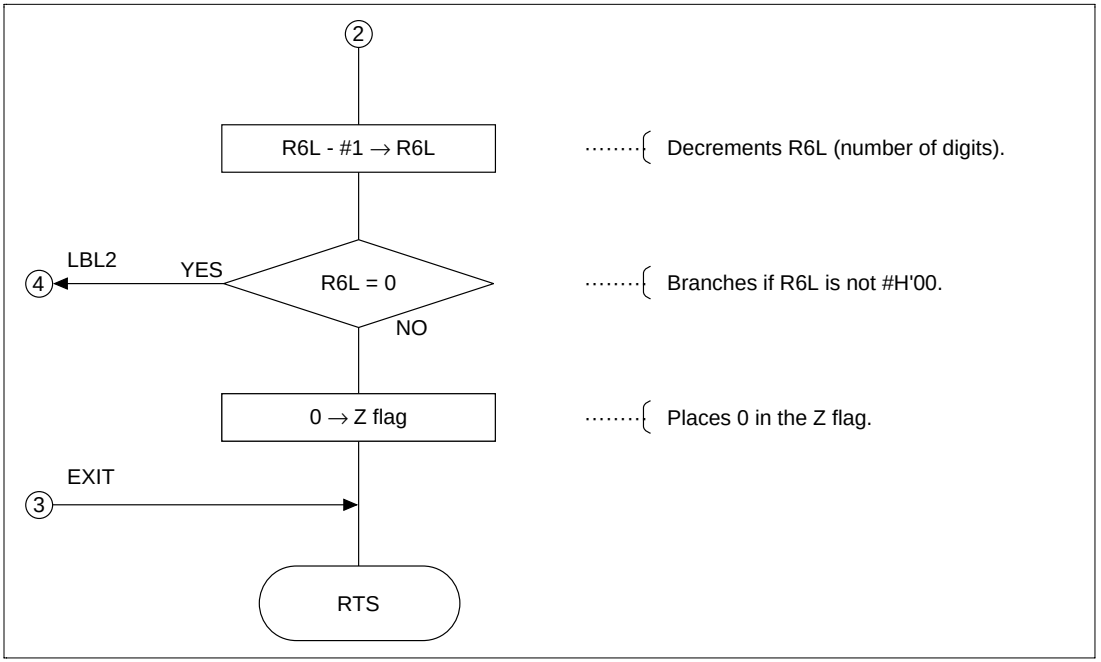
(ii) The divisor is subtracted from the dividend. Subtractions are repeated until the result becomes negative. The number of subtractions thus done is placed in the lower 4 bits (the least significant digit) of the dividend. ((2)→(3)→(1) in figure 7.32) When the result has become negative, the divisor is added to the result (remainder) to return to the value before subtractions. ((4) in figure 7.32)

(iii) The steps (i) to (ii) are repeated as many times as 8 digits.

7.10.7 Flowchart




1
LBL4
R1L + #H'01 → R1L
Adds H'01 to R1L.
R5L - R3L → R5L
Decimal correction of R5L
R5H - R3H - C → R5H
Decimal correction of R5H
R4L - R2L - C → R4L
Decimal correction of R4L
R4H - R2H - C → R4H
Decimal correction of R4H
Performs decimal subtraction of the divisor (R2, R3) from the remainder (R4, R5).
YES
C flag (CCR) = 0
NO
R5L - R3L → R5L
Decimal correction of R5L
R5H - R3H - C → R5H
Decimal correction of R5H
R4L - R2L - C → R4L
Decimal correction of R4L
R4H - R2H - C → R4H
Decimal correction of R4H
Branches if the result of decimal subtraction is positive. If it is negative, performs decimal addition of the divisor (R2, R3) to the remainder (R4, R5) and subtracts H'01 from R1L.
R1L - #H'01 → R1L
2


2
R6L - #1 → R6L
Decrements R6L (number of digits).
LBL2
YES
4
R6L = 0
Branches if R6L is not #H'00.
NO
0 → Z flag
Places 0 in the Z flag.
EXIT
3
RTS

7.10.8 Program List

```
*** H8/300 ASSEMBLER              VER 1.0B **   08/18/92 10:02:05                                    PAGE     1
PROGRAM NAME =

    1                                ;*********************************************************************
    2                                ;*
    3                                ;*   00 - NAME       :MULTIPLE-PRECISION DECIMAL DIVISION (DIVD)
    4                                ;*
    5                                ;*********************************************************************
    6                                ;*
    7                                ;*   ENTRY           :R2,R3           (DIVISOR)
    8                                ;*                    R0,R1           (DIVIDEND)
    9                                ;*
   10                                ;*   RETURNS         :R0,R1           (QUOTIENT)
   11                                ;*                    R4,R5           (RESIDUAL)
   12                                ;*                    Z flag OF CCR (Z=1;FALSE , Z=0;TRUE)
   13                                ;*
   14                                ;*********************************************************************
   15                                ;
   16 DIVD_cod C 0000                     .SECTION        DIVD_code,CODE,ALIGN=2
   17                                     .EXPORT         DIVD
   18                                ;
   19 DIVD_cod C      00000000       DIVD .EQU    $               ;Entry point
   20 DIVD_cod C 0000 79040000            MOV.W   #H'0000,R4      ;Clear R4
   21 DIVD_cod C 0004 0D45                MOV.W   R4,R5           ;Clear R5
   22 DIVD_cod C 0006 1D42                CMP.W   R4,R2
   23 DIVD_cod C 0008 4604                BNE     LBL1            ;Branch if Z=0
   24 DIVD_cod C 000A 1D53                CMP.W   R5,R3
   25 DIVD_cod C 000C 4744                BEQ     EXIT            ;Branch if Z=1 then exit
   26 DIVD_cod C 000E                LBL1
   27 DIVD_cod C 000E FE08                MOV.B   #H'08,R6L       ;Set bit counter1
   28 DIVD_cod C 0010                LBL2
   29 DIVD_cod C 0010 F604                MOV.B   #H'04,R6H       ;Set bit counter2
   30 DIVD_cod C 0012                LBL3
   31 DIVD_cod C 0012 1009                SHLL.B  R1L             ;Shift dividend
   32 DIVD_cod C 0014 1201                ROTXL.B R1H
   33 DIVD_cod C 0016 1208                ROTXL.B R0L
   34 DIVD_cod C 0018 1200                ROTXL.B R0H
   35 DIVD_cod C 001A 120D                ROTXL.B R5L
   36 DIVD_cod C 001C 1205                ROTXL.B R5H
   37 DIVD_cod C 001E 120C                ROTXL.B R4L
   38 DIVD_cod C 0020 1204                ROTXL.B R4H
   39 DIVD_cod C 0022 1A06                DEC.B   R6H             ;Decrement bit counter2
   40 DIVD_cod C 0024 46EC                BNE     LBL3            ;Branch if Z=0
   41 DIVD_cod C 0026                LBL4
   42 DIVD_cod C 0026 0A09                INC.B   R1L             ;Increment R1L
   43 DIVD_cod C 0028 18BD                SUB.B   R3L,R5L         ;R5L - R3L     -> R5L
   44 DIVD_cod C 002A 1F0D                DAS.B   R5L             ;Decimal adjust R5L
   45 DIVD_cod C 002C 1E35                SUBX.B  R3H,R5H         ;R5H - R3H - C -> R3H
   46 DIVD_cod C 002E 1F05                DAS.B   R5H             ;Decimal adjust R5H
   47 DIVD_cod C 0030 1EAC                SUBX.B  R2L,R4L         ;R4L - R2L - C -> R4L
   48 DIVD_cod C 0032 1F0C                DAS.B   R4L             ;Decimal adjust R4L
   49 DIVD_cod C 0034 1E24                SUBX.B  R2H,R4H         ;R4H - R2H - C -> R4H
   50 DIVD_cod C 0036 1F04                DAS.B   R4H             ;Decimal adjust R4H
   51 DIVD_cod C 0038 44EC                BCC     LBL4            ;Branch if C=0
   52                                ;
   53 DIVD_cod C 003A 08BD                ADD.B   R3L,R5L         ;R3L + R5L     -> R5L
   54 DIVD_cod C 003C 0F0D                DAA.B   R5L             ;Decimal adjust R5L
   55 DIVD_cod C 003E 0E35                ADDX.B  R3H,R5H         ;R3H + R5H + C -> R5H
   56 DIVD_cod C 0040 0F05                DAA.B   R5H             ;Decimal adjust R5H
   57 DIVD_cod C 0042 0EAC                ADDX.B  R2L,R4L         ;R2L + R4L + C -> R4L
   58 DIVD_cod C 0044 0F0C                DAA.B   R4L             ;Decimal adjust R4L
   59 DIVD_cod C 0046 0E24                ADDX.B  R2H,R4H         ;R2H + R4H + C -> R4H
   60 DIVD_cod C 0048 0F04                DAA.B   R4H             ;Decimal adjust R4H
   61 DIVD_cod C 004A 1A09                DEC.B   R1L             ;Decrement R1L
   62 DIVD_cod C 004C 1A0E                DEC.B   R6L             ;Decrement bit counter1
   63 DIVD_cod C 004E 46C0                BNE     LBL2
```

```
 64 DIVD_cod C 0050 06FB                    ANDC.B  #B'11111011,CCR ;Clear Z flag of CCR
 65 DIVD_cod C 0052                   EXIT
 66 DIVD_cod C 0052 5470                    RTS
 67                                   ;
 68                                         .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

7.11 Addition of Multiple-Precision BCD Numbers

MCU: H8/300 Series
H8/300L Series

Label name: ADDD2

7.11.1 Function

1. The software ADDD2 adds a multiple-precision binary-coded decimal (BCD) number to another multiple-precision BCD number and places the result in the data memory where the augend was placed.
2. The arguments used with the software ADDD2 are unsigned integers, each being up to 255 bytes long.

7.11.2 Arguments

Description		Memory area	Data length (bytes)
Input	Augend and addend byte count	R0L	1
	Start address of augend	R3	2
	Start address of addend	R4	2
Output	Start address of the result of addition	R3	2
	Error	Z flag (CCR)	1
	Carry	C flag (CCR)	1

7.11.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	×	↕	×	×	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	↕	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.11.4 Specifications

Program memory (bytes)
44
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
7680
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.11.5 Notes

The clock cycle count (7680) in the specifications is for addition of 255 bytes to 255 bytes.

7.11.6 Description

1. Details of functions
   a. The following arguments are used with the software ADDD2:
      R0L: Contains, as an input argument, the byte count of an augend and an addend in 2-digit hexadecimals.
      R3: Contains the start address of the augend in the data memory area. The start address of the result of addition is placed in this register after execution of the software ADDD2.
      R4: Contains, as an input argument, the start address of the addend in the data memory area.

Z flag (CCR): Indicates an error in data length as an output argument.

Z flag = 0: The data byte count (R0L) was not 0.

Z flag = 1: The data byte count (R0L) was 0 (indicating an error).

C flag (CCR): Determines the presence or absence of a carry, as an output argument, after execution of the software ADDD2.

C flag = 0: No carry occurred in the result of addition.

C flag = 1: A carry occurred in the result of addition (see figure 7.28).

b. Figure 7.33 shows an example of the software ADDD2 being executed. When the input arguments are set as shown in 1., the result of addition is placed in the data memory area as shown in 2..

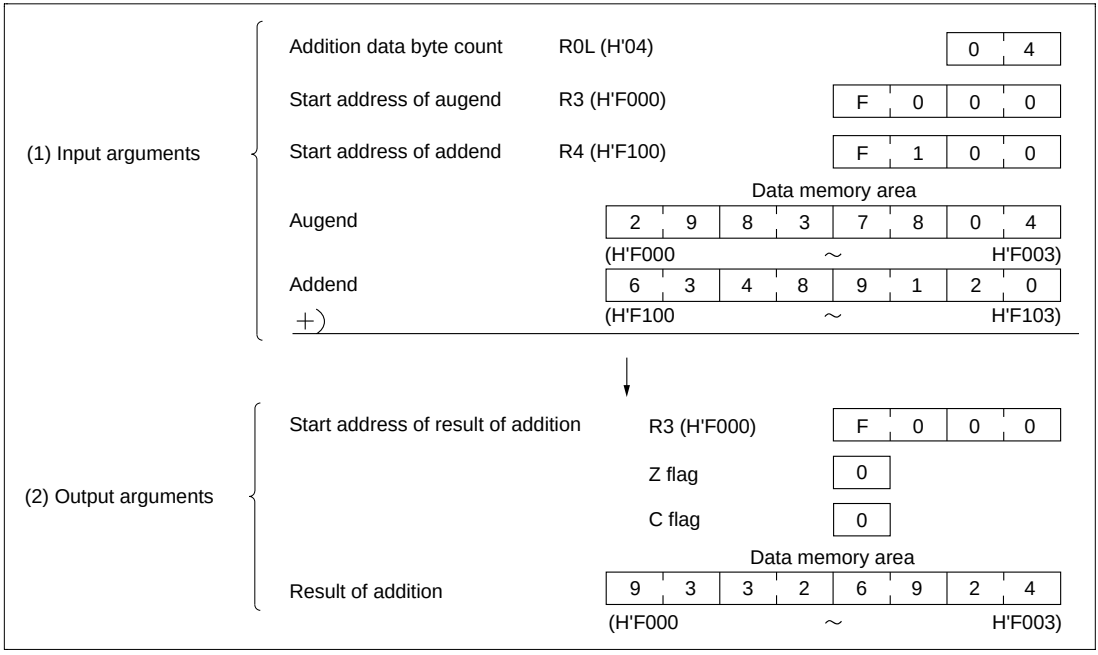


Figure 7.33 Example of Software ADDD2 Execution

Figure 7.34 shows an example of addition with a carry that occurred in the result.

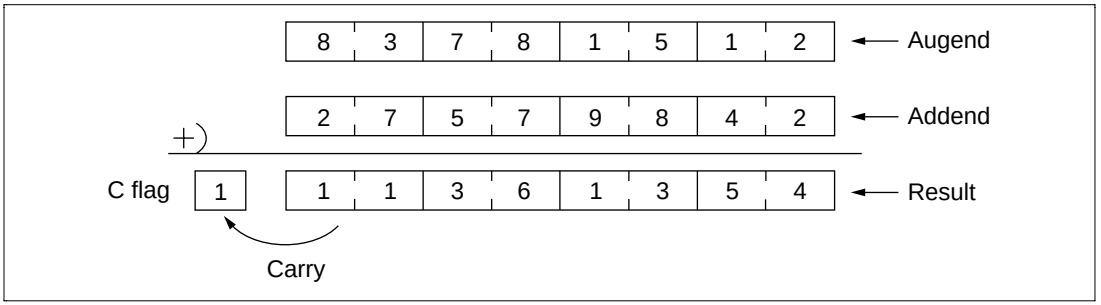


Figure 7.34 Example of Addition with a Carry

2. Notes on usage
   a. When upper bits are not used (see figure 7.35), set 0's in them. The software ADDD2 performs byte-based addition; if 0's are not set in the upper bits unused, no correct result can be obtained because the addition is done on the numbers including indeterminate data.

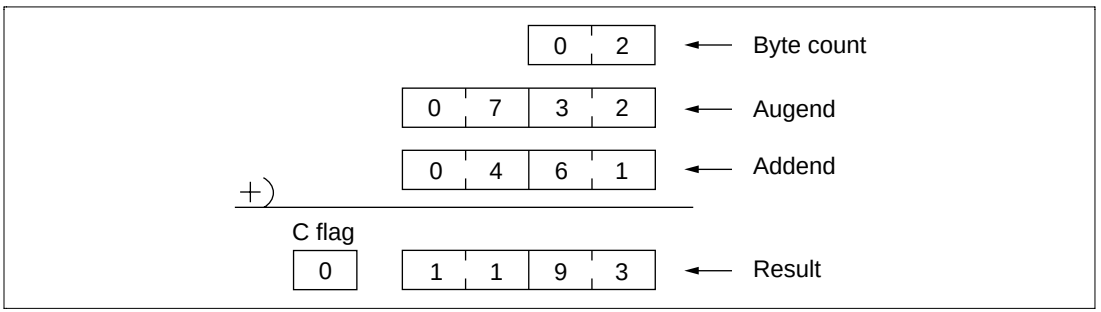


Figure 7.35 Example of Addition with Upper Bits Unused

   b. After execution of the software ADDD2, the augend is destroyed because the result is placed in the data memory area where the augend was set. If the augend is necessary after software ADDD2 execution, save it on memory.
3. Data memory

   The software ADDD2 does not use the data memory.

4. Example of use

This is an example of adding 8 bytes of data. Set the start addresses of a byte count, an augend, and an addend in the registers and call the software ADDD2 as a subroutine.

```
WORK1   .RES. B   1          ..... Reserves a data memory area in which the
                                   user program places a byte count.
WORK2   .RES. B   8          ..... Reserves a data memory area in which the
                                   user program places an 8-byte (16-digit
                                   BCD) augend.
WORK3   .RES. B   8          ..... Reserves a data memory area in which the
                                   user program places an 8-byte (16-digit
                                   BCD) addend.
        :
        MOV. B    @WORK1, R0L ..... Places in the input argument (R0L) the
                                   byte count set by the user program.
        MOV. W    #WORK2, R3  ..... Places in the input argument (R3) the start
                                   address of the augend set by the user
                                   program.
        MOV. W    #WORK3, R4  ..... Places in the input argument (R4) the start
                                   address of the addend set by the user
                                   program.
        JSR       @ADDD2      ..... Call the software ADDD2 as a subroutine.
        BCS       OVER        ..... Branches to the carry processing routine if
                                   a carry has occurred in the result of
                                   addition.
        :
OVER    Carry processing routine.
```

5. Operation
   a. Addition of multiple-precision BCD numbers can be done by performing a series of 1-byte add instructions (ADDX.B) with decimal-correct instructions (DAA) as the augend and addend data are placed in registers, 2 digits in 1 byte.
   b. The address of the least significant byte of the data memory area for the augend is placed in R3, and the address of the least significant byte of the data memory area for the addend in R4.
   c. R1L that is used for saving the C flag is cleared. .

d. The augend and addend are loaded in R2L and R2H respectively, byte by byte, starting at their least significant byte and then equation 1 is executed:

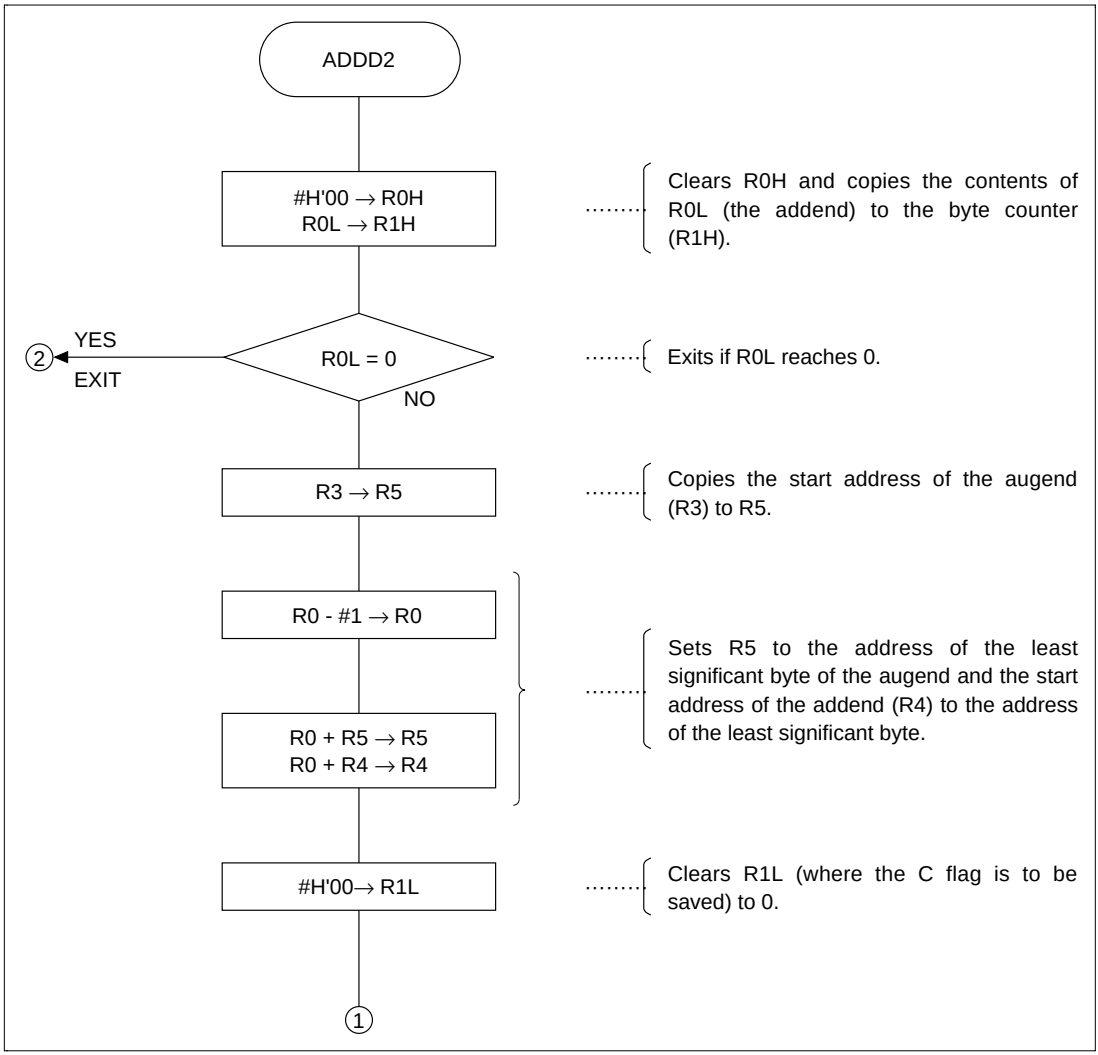
   where the C flag indicates a carry that may occur in the result of addition of the lower bytes.

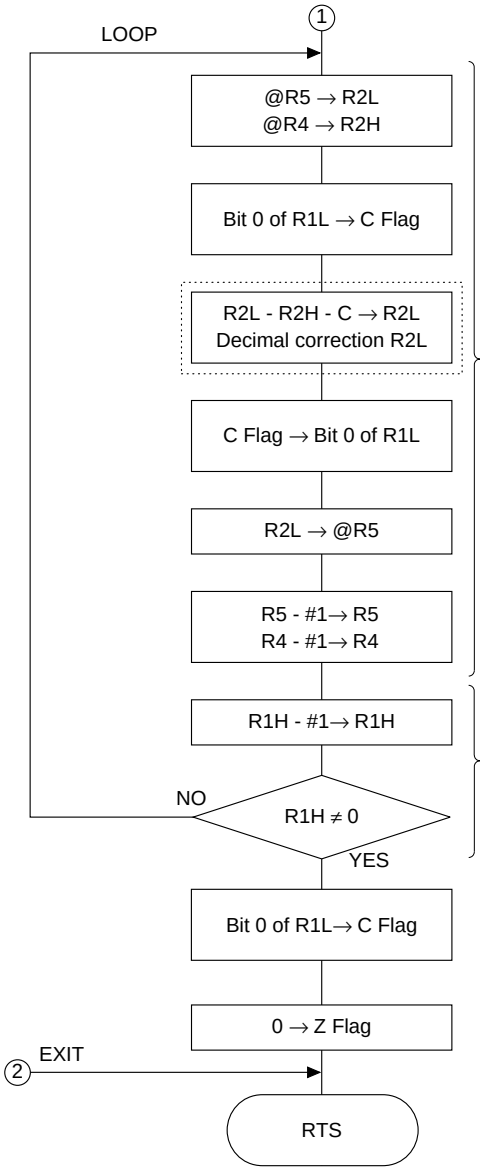
$$\left.\begin{array}{r} \text{R2L (augend)} + \text{R2H (addend)} + \text{C} \rightarrow \text{R2L} \\ \text{Decimal correction of R2L} \rightarrow \text{R2L} \\ \text{R2L} \rightarrow @\text{R3} \end{array}\right\} \cdots\cdots\cdots \text{equation 1}$$

e. The result of (d) is placed in the data memory area for the augend.

f. R3, R4, and R0L are decremented each time the process d. to e. terminates. This processing is repeated until R0L reaches 0.

7.11.7 Flowchart





Adds the augend, addend and carry and places the result (decimally corrected) in the data memory area where the augend was placed.

Repeats this process as many times as the byte count of the addition data.

Sets bit 0 of R1L in the C flag.

Clears the Z flag to 0.

Note: ADDD2 is the same as ADD2, SUB2, and SUBD2 except for the step surrounded by dotted lines.

7.11.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:02:42
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :MULTIPLE-PRECISION DECIMAL ADDITION
    4                                   ;*                            (ADDD2)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L    (BYTE COUNTER OF ADDTION DATA)
    9                                   ;*                    R3     (START ADDRESS OF AUGEND)
   10                                   ;*                    R4     (START ADDRESS OF ADDEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R3     (START ADDRESS OF RESULT)
   13                                   ;*                    Z flag OF CCR   (Z=0;TRUE,Z=1;FALSE)
   14                                   ;*                    C flag OF CCR   (C=0;TRUE,C=1;OVERFLOW)
   15                                   ;*
   16                                   ;*********************************************************************
   17                                   ;
   18 ADDD2_co C 0000                        .SECTION        ADDD2_code,CODE,ALIGN=2
   19                                        .EXPORT         ADDD2
   20                                   ;
   21 ADDD2_co C      00000000          ADDD2        .EQU    $               ;Entry point
   22 ADDD2_co C 0000 F000                   MOV.B   #H'00,R0H       ;Clear R0H
   23 ADDD2_co C 0002 0C81                   MOV.B   R0L,R1H         ;Clear R1H
   24 ADDD2_co C 0004 4724                   BEQ     EXIT            ;Branch if Z=1 then exit
   25 ADDD2_co C 0006 0D35                   MOV.W   R3,R5
   26 ADDD2_co C 0008                   MAIN
   27 ADDD2_co C 0008 1B00                   SUBS.W  #1,R0           ;Decrement R0
   28 ADDD2_co C 000A 0905                   ADD.W   R0,R5           ;Set end address to summand pointer
   29 ADDD2_co C 000C 0904                   ADD.W   R0,R4           ;Set end address to addend pointer
   30 ADDD2_co C 000E F900                   MOV.B   #H'00,R1L       ;Clear R1L
   31 ADDD2_co C 0010                   LOOP
   32 ADDD2_co C 0010 685A                   MOV.B   @R5,R2L         ;Load summand data
   33 ADDD2_co C 0012 6842                   MOV.B   @R4,R2H         ;Load addend data
   34 ADDD2_co C 0014 7709                   BLD     #0,R1L          ;Bit load bit 0 of R1L
   35 ADDD2_co C 0016 0E2A                   ADDX.B  R2H,R2L         ;R2H + R2L + C -> R2L
   36 ADDD2_co C 0018 0F0A                   DAA     R2L             ;Decimal adjust R1L
   37 ADDD2_co C 001A 6709                   BST     #0,R1L          ;Store C falg to bit 0 of R1L
   38 ADDD2_co C 001C 68DA                   MOV.B   R2L,@R5         ;Store struct
   39 ADDD2_co C 001E 1B05                   SUBS.W  #1,R5           ;Decrement summand pointer
   40 ADDD2_co C 0020 1B04                   SUBS.W  #1,R4           ;Decrement addend pointer
   41 ADDD2_co C 0022 1A01                   DEC.B   R1H             ;Decrement R1H
   42 ADDD2_co C 0024 46EA                   BNE     LOOP            ;Branch if Z=0
   43                                   ;
   44 ADDD2_co C 0026 7709                   BLD     #0,R1L          ;Load bit 0 of R1L to C flag
   45 ADDD2_co C 0028 06FB                   ANDC.B  #H'FB,CCR       ;Clear Z flag of CCR
   46 ADDD2_co C 002A                   EXIT
   47 ADDD2_co C 002A 5470                   RTS
   48                                   ;
   49                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.12 Subtraction of Multiple-Precision BCD Numbers

MCU: H8/300 Series
H8/300L Series

Label name: SUBD2

7.12.1 Function

1. The software SUBD2 subtracts a multiple-precision binary-coded decimal (BCD) number from another multiple-precision BCD number and places the result in the data memory where the minuend was set.
2. The arguments used with the software SUBD2 are unsigned integers, each being up to 255 bytes long.

7.12.2 Arguments

Description		Memory area	Data length (bytes)
Input	Minuend and subtrahend byte count	R0L	1
	Start address of minuend	R3	2
	Start address of subtrahend	R4	2
Output	Start address of result	R3	2
	Error	Z flag (CCR)	1
	Borrow	C flag (CCR)	1

7.12.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	×	↕	×	×	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	↕	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

7.12.4 Specifications

Program memory (bytes)
44
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
7680
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.12.5 Notes

The clock cycle count (7680) in the specifications is for subtraction of 255 bytes from 255 bytes.

7.12.6 Description

1. Details of functions
   a. The following arguments are used with the software SUBD2:
      R0L: Contains, as an input argument, the byte count of a minuend and the byte count of a subtrahend in 2-digit hexadecimals.
      R3: Contains, as an input argument, the start address of the data memory area where the minuend is placed. After execution of the software SUBD2, the start address of the result is placed in this register.
      R4: Contains, as an input argument, the start address of the data memory area where the subtrahend is placed.
      Z flag (CCR): Indicates an error in data length as an output argument.

Z flag = 0: The data byte count (R0L) was not 0.

Z flag = 1: The data byte count (R0L) was 0, indicating an error.

C flag (CCR): Determines the presence or absence of a borrow after software SUBD2 execution as an output argument.

C flag = 0: No borrow occurred in the result.

C flag = 1: A borrow occurred in the result. (See figure 7.36)

b. Figure 7.36 shows an example of the software SUBD2 being executed. When the input arguments are set as shown in (1), the result of subtraction is placed in the data memory area as shown in (2).

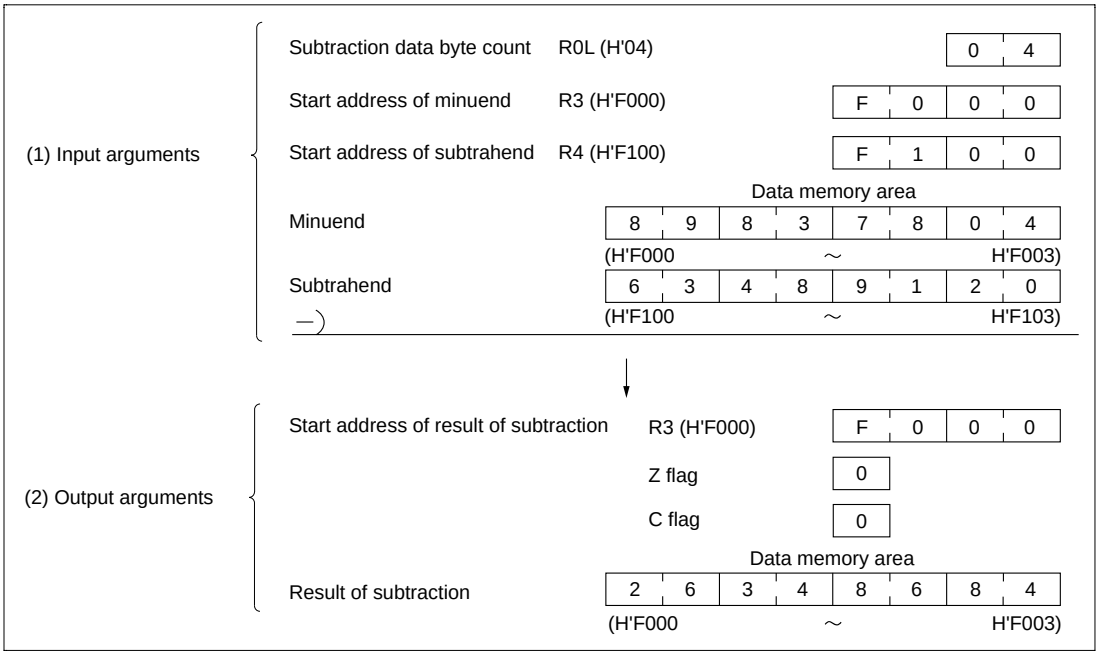


Figure 7.36 Example of Software SUBD2 Execution

Figure 7.37 shows an example of subtraction with a borrow that has occurred in the result.

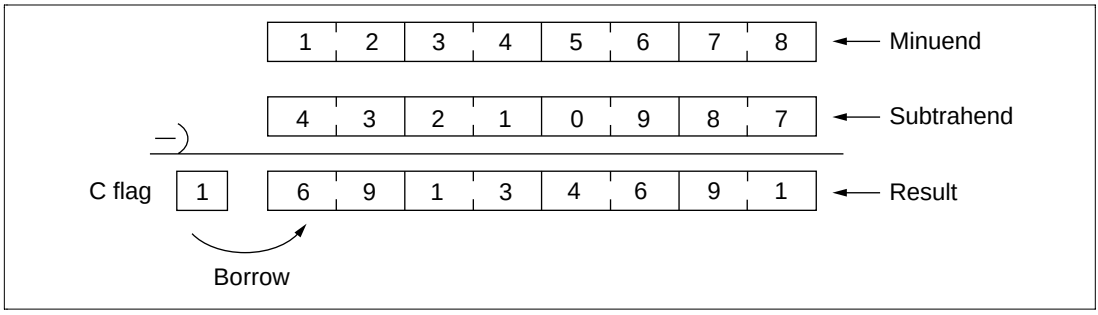


Figure 7.37 Example of Subtraction with a Borrow

2. Notes on usage
   a. When upper bits are not used (see figure 7.38), set 0's in them. The software SUBD2 performs byte-based subtraction; if 0's are not set in the upper bits unused, no correct result can be obtained because the subtraction is done on the numbers including indeterminate data.

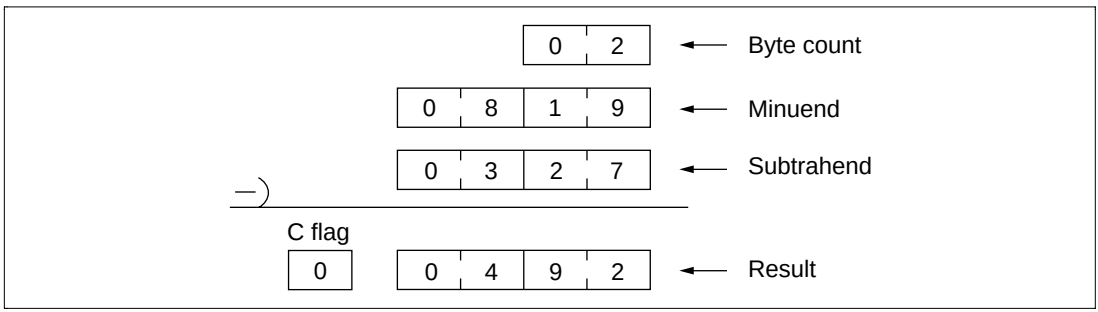
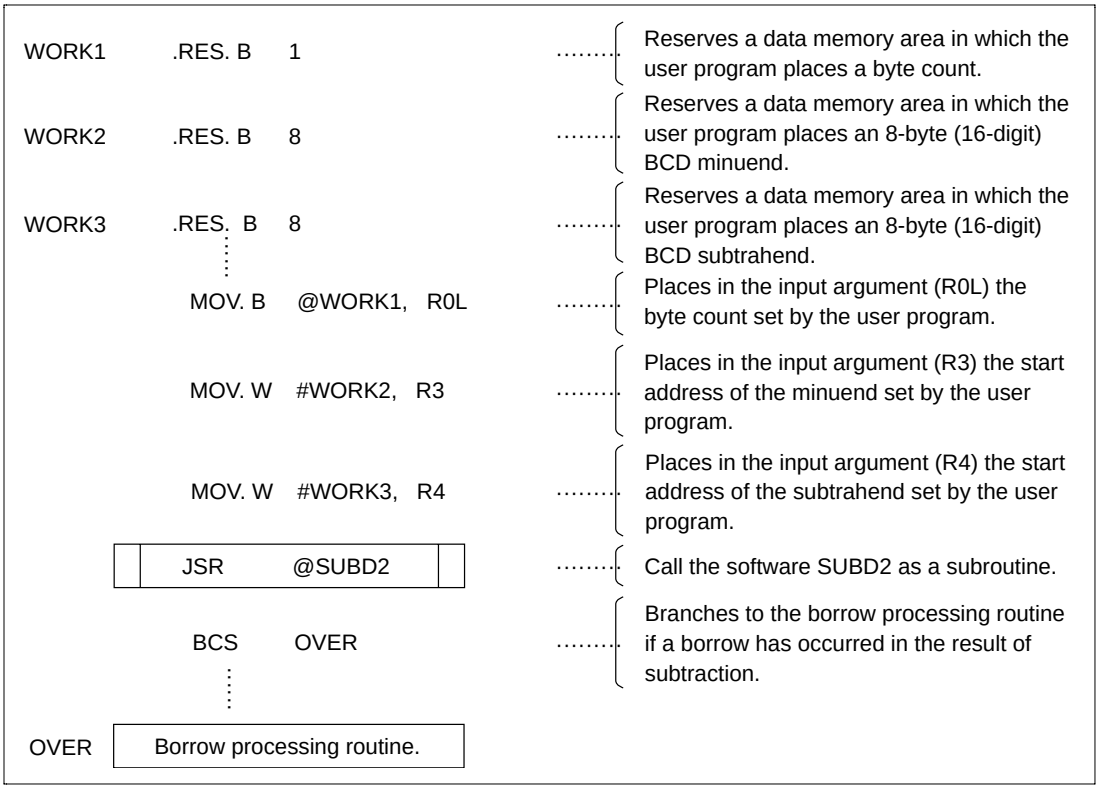


Figure 7.38 Example of Subtraction with Upper Bits Unused

   b. After execution of the software SUBD2, the minuend is destroyed because the result is placed in the data memory area where the minuend was set. If the minuend is necessary after software SUBD2 execution, save it on memory.
3. Data memory

   The software SUBD2 does not use the data memory.
4. Example of use

   This is an example of subtracting 8 bytes of data. Set the start addresses of a byte count, a minuend and a subtrahend in the registers and call the software SUBD2 as a subroutine.

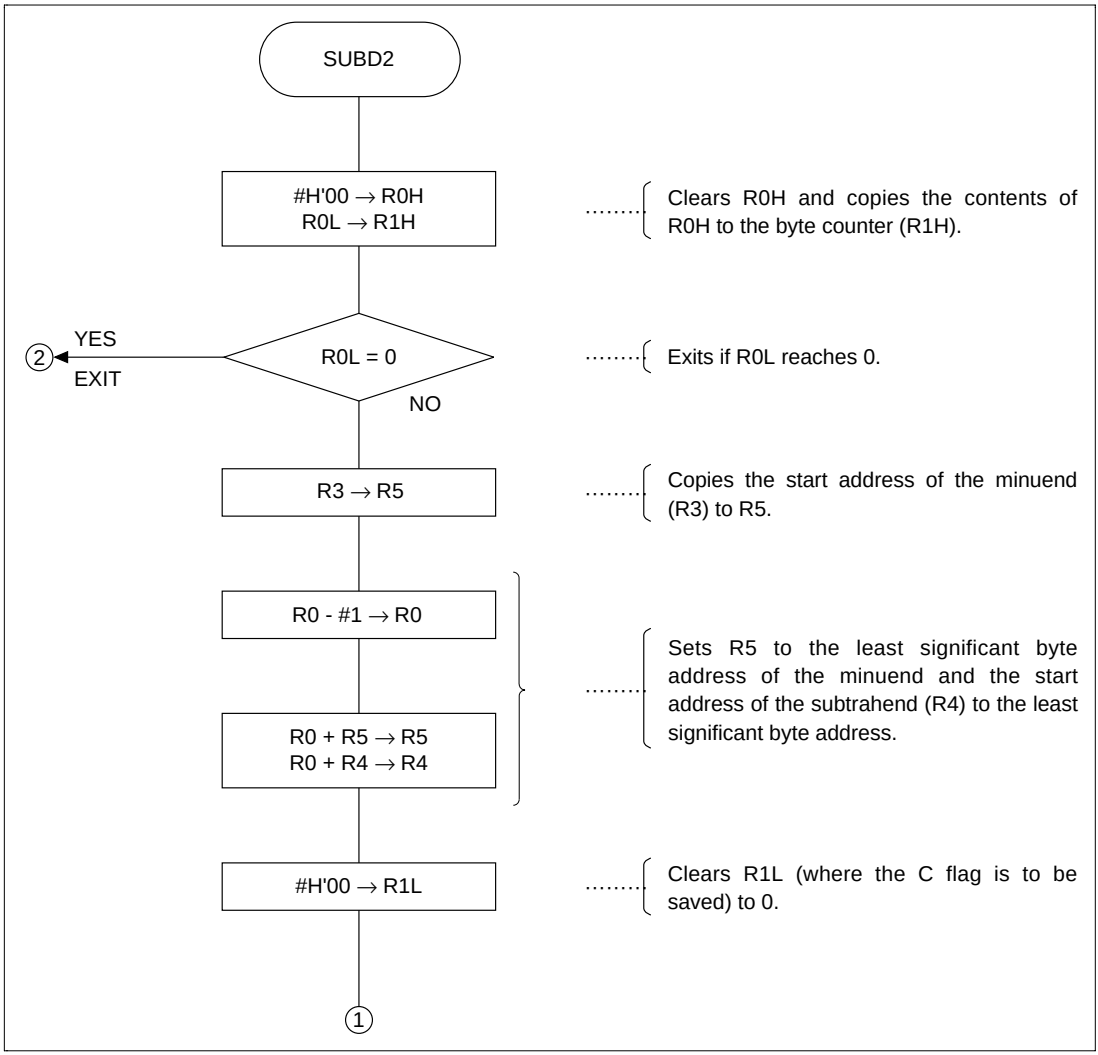


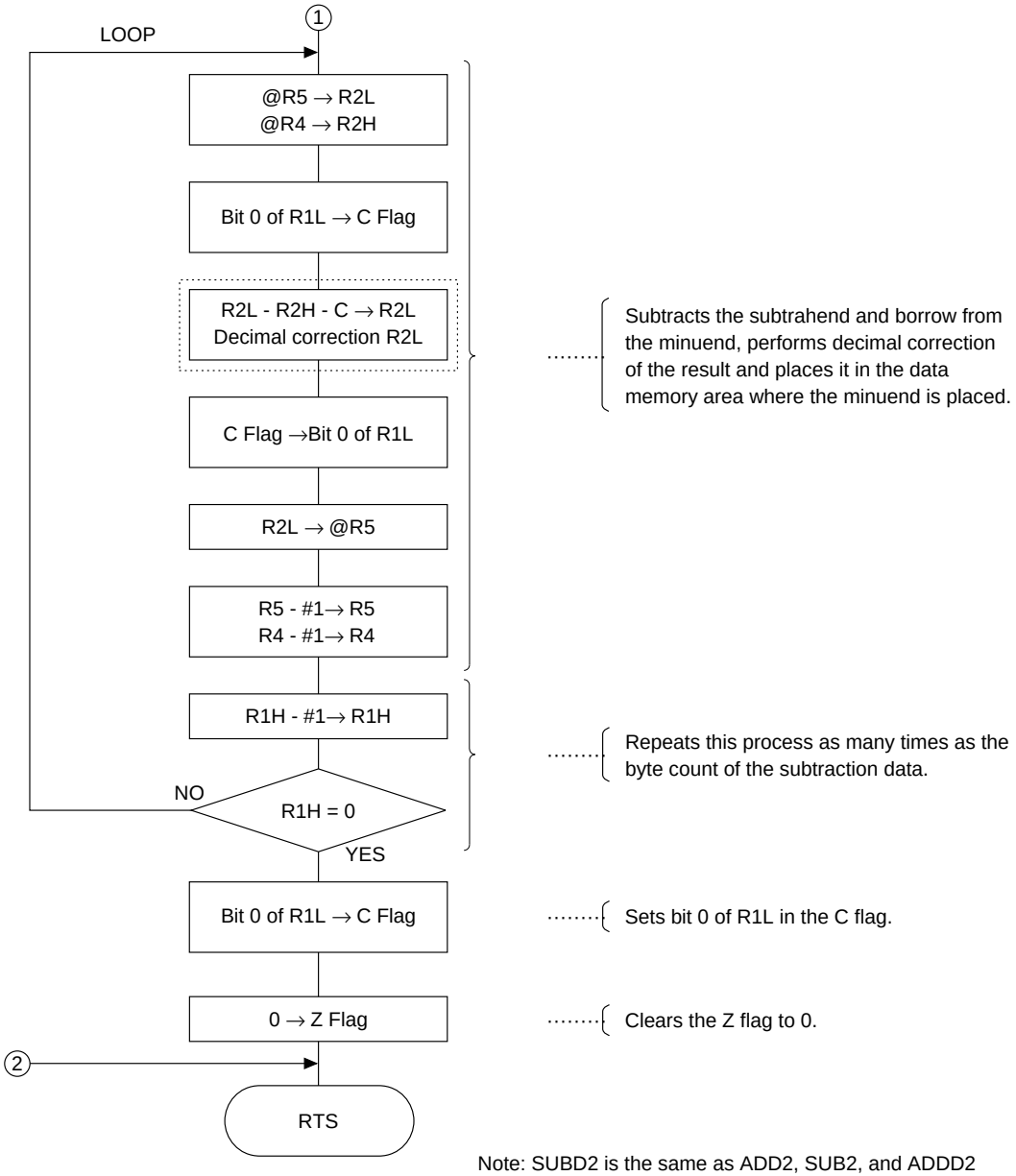
5. Operation
   a. Subtraction of multiple-precision binary numbers can be done by repeating a 1-byte subtract instruction (SUBX.B) and a decimal-correct instruction (DAA) as the minuend and subtrahend data are placed in registers, 2 digits in 1 byte..
   b. The least significant byte of the data memory area for the minuend is placed in R3, and the least significant byte of the data memory area for the subtrahend in R4.
   c. R1L that is used for saving the C flag is cleared.
   d. The minuend and subtrahend are loaded in R2L and R2H respectively, byte by byte, starting at their least significant byte and equation 1 is executed:

$$\left.\begin{array}{r} \text{R2L (minuend)} - \text{R2H (subtrahend)} - \text{C} \rightarrow \text{R2L} \\ \text{Decimal correction of R2L} \rightarrow \text{R2L} \\ \text{R2L} \rightarrow @\text{R3} \end{array}\right\} \quad \cdots\cdots\cdots \text{ equation 1}$$

   where the C flag indicates a borrow that may occur in the result of subtraction of the lower bytes.
   e. The result of d. is placed in the data memory area for the minuend.
   f. R3, R4, and R0L are decremented each time the process d. to e. terminates. This processing is repeated until R0L reaches 0.

7.12.7 Flowchart





Note: SUBD2 is the same as ADD2, SUB2, and ADDD2 except for the step surrounded by dotted lines.

7.12.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:03:31
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :MULTIPLE-PRECISION DECIMAL SUBSTRUCTION
    4                                   ;*                            (SUBD2)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L    (BYTE LENGTH OF DATA)
    9                                   ;*                    R3     (START ADDRESS OF MINUEND)
   10                                   ;*                    R4     (START ADDRESS OF SUBSTRAHEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R3     (START ADDRESS OF RESULT)
   13                                   ;*                    Z BIT OF CCR    (Z=0;TRUE , Z=1;FALSE)
   14                                   ;*                    C BIT OF CCR    (C=0;TRUE , C=1;OVERFLOW)
   15                                   ;*
   16                                   ;*********************************************************************
   17                                   ;
   18 SUBD2_co C 0000                        .SECTION        SUBD2_code,CODE,ALIGN=2
   19                                        .EXPORT         SUBD2
   20                                   ;
   21 SUBD2_co C      00000000          SUBD2        .EQU    $               ;Entry point
   22 SUBD2_co C 0000 F000                   MOV.B   #H'00,R0H       ;Clear R0H
   23 SUBD2_co C 0002 0C81                   MOV.B   R0L,R1H         ;Set byte counter
   24 SUBD2_co C 0004 4724                   BEQ     EXIT            ;Branch if Z=1 then exit
   25 SUBD2_co C 0006 0D35                   MOV.W   R3,R5
   26 SUBD2_co C 0008                   MAIN
   27 SUBD2_co C 0008 1B00                   SUBS.W  #1,R0           ;Decrement byte length
   28 SUBD2_co C 000A 0905                   ADD.W   R0,R5           ;Set end address of minuend
   29 SUBD2_co C 000C 0904                   ADD.W   R0,R4           ;Set end address of substrahend
   30 SUBD2_co C 000E F900                   MOV.B   #H'00,R1L       ;Clear R1L
   31 SUBD2_co C 0010                   LOOP
   32 SUBD2_co C 0010 685A                   MOV.B   @R5,R2L         ;Load minuend data
   33 SUBD2_co C 0012 6842                   MOV.B   @R4,R2H         ;Load substrahend data
   34 SUBD2_co C 0014 7709                   BLD     #0,R1L          ;Bit load bit 0 of R1L
   35 SUBD2_co C 0016 1E2A                   SUBX.B  R2H,R2L         ;R2L - R2H - C -> R2L
   36 SUBD2_co C 0018 1F0A                   DAS     R2L             ;Decimal adjust R2L
   37 SUBD2_co C 001A 6709                   BST     #0,R1L          ;Bit store bit 0 of R1L
   38 SUBD2_co C 001C 68DA                   MOV.B   R2L,@R5         ;Store result
   39 SUBD2_co C 001E 1B05                   SUBS.W  #1,R5           ;Decrement minuend pointer
   40 SUBD2_co C 0020 1B04                   SUBS.W  #1,R4           ;Decrement substrahend pointer
   41 SUBD2_co C 0022 1A01                   DEC.B   R1H             ;Decrement byte counter
   42 SUBD2_co C 0024 46EA                   BNE     LOOP            ;Branch if Z=0
   43                                   ;
   44 SUBD2_co C 0026 7709                   BLD     #0,R1L          ;Bit load bit 0 of R1L
   45 SUBD2_co C 0028 06FB                   ANDC.B  #H'FB,CCR       ;Clear Z bit
   46 SUBD2_co C 002A                   EXIT
   47 SUBD2_co C 002A 5470                   RTS
   48                                   ;
   49                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.13 Addition of Signed 32-Bit Binary Numbers

MCU: H8/300 Series|
H8/300L Series

Label name: SADD

7.13.1 Function

1. The software SADD adds a signed 32-bit binary number to another signed 32-bit binary number and places the result in a general-purpose register.
2. The arguments used with the software SADD are signed integers.
3. All data is manipulated on general-purpose registers.

7.13.2 Arguments

Description		Memory area	Data length (bytes)
Input	Augend	R0, R1	4
	Addend	R2, R3	4
Output	Result of addition	R0, R1	4
	Carry	V flag (CCR)	1

7.13.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	•	•	•	•	×	•

I	U	H	U	N	Z	V	C
•	×	×	×	×	×	↕	×

× : Unchanged
• : Indeterminate
↕ : Result

7.13.4 Specifications

Program memory (bytes)
20
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
44
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.13.5 Description

1. Details of functions
   a. The following arguments are used with the software SADD:
      (i) Input arguments:
         R0, R1: Contain a signed 32-bit binary augend.
         R2, R3: Contain a signed 32-bit binary addend.
      (ii) Output arguments
         R0, R1: Contain the result of addition (a signed 32-bit binary number)
         V flag (CCR): Determines the presence or absence of a carry as a result of addition.
         V flag = 1: A carry occurred in the result.
         V flag = 0: No carry occurred in the result.

b. Figure 7.39 shows an example of the software SADD being executed. When the input arguments are set as shown in 1., the result of addition is placed in R0 and R1 as shown in 2..

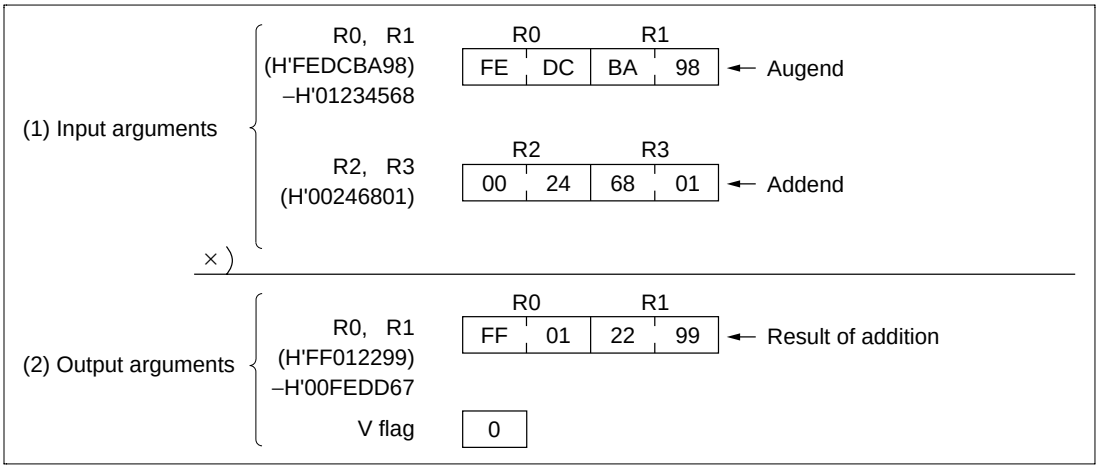


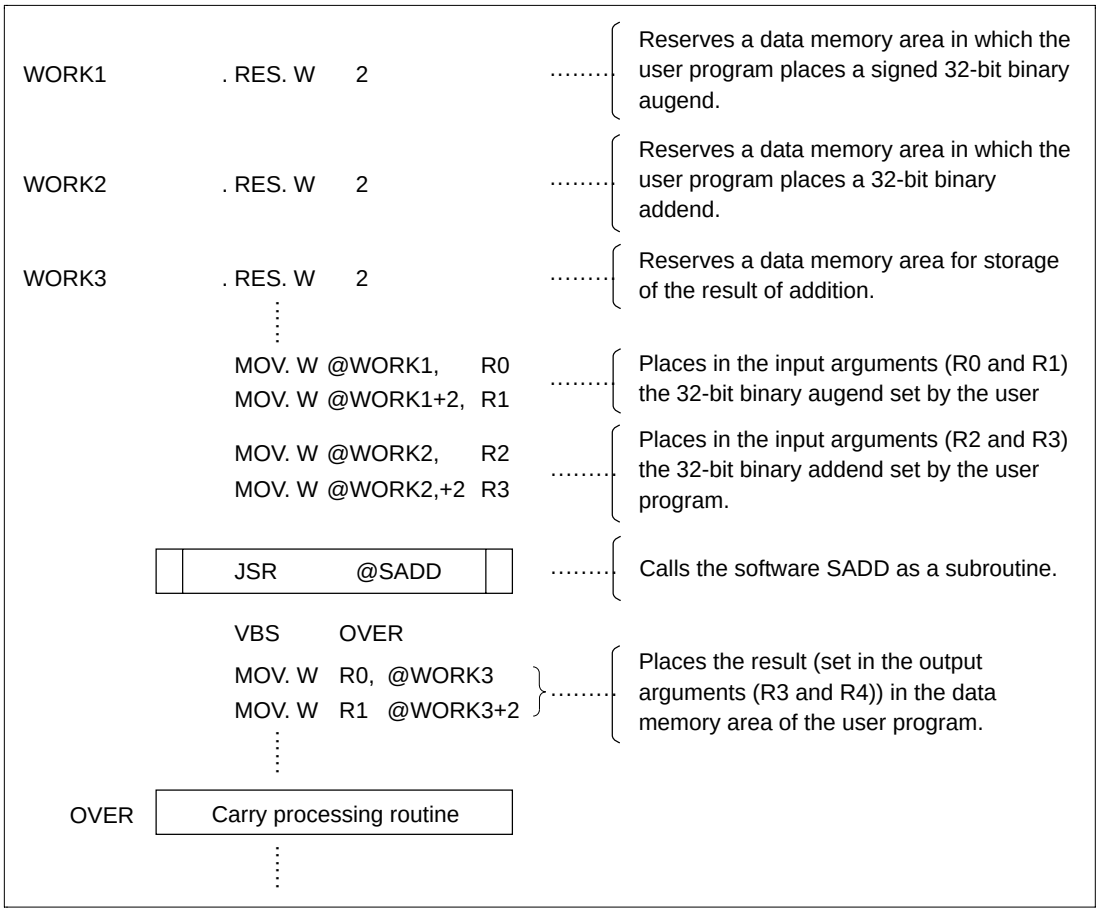
Figure 7.39 Example of Software SADD Execution

2. Notes on usage
   a. After execution of the software SADD, the augend is destroyed because the result is placed in R0 and R1. If the augend is necessary after software SADD execution, save it on memory.
3. Data memory

   The software SADD does not use the data memory.

4. Example of use

Set an augend and an addend in the input arguments and call the software SADD as a subroutine.



5. Operation
   a. Addition of signed 32-bit binary numbers is done by using add instructions (ADD.W and ADDX.B).
   b. The addition steps are as follows:
      (i) An augend is placed in R0 and R1 and an addend in R2 and R3.
      (ii) The user bits (bits 6 and 4) and the overflow flag (bit 2) are cleared.
      (iii) If the augend is negative, sign bit "1" is placed in the user bit (bit 6) of the CCR. If the addend is negative, sign bit "1" is placed in the user bit (bit 4) of the CCR.
      (iv) The augend is added to the addend as follows:

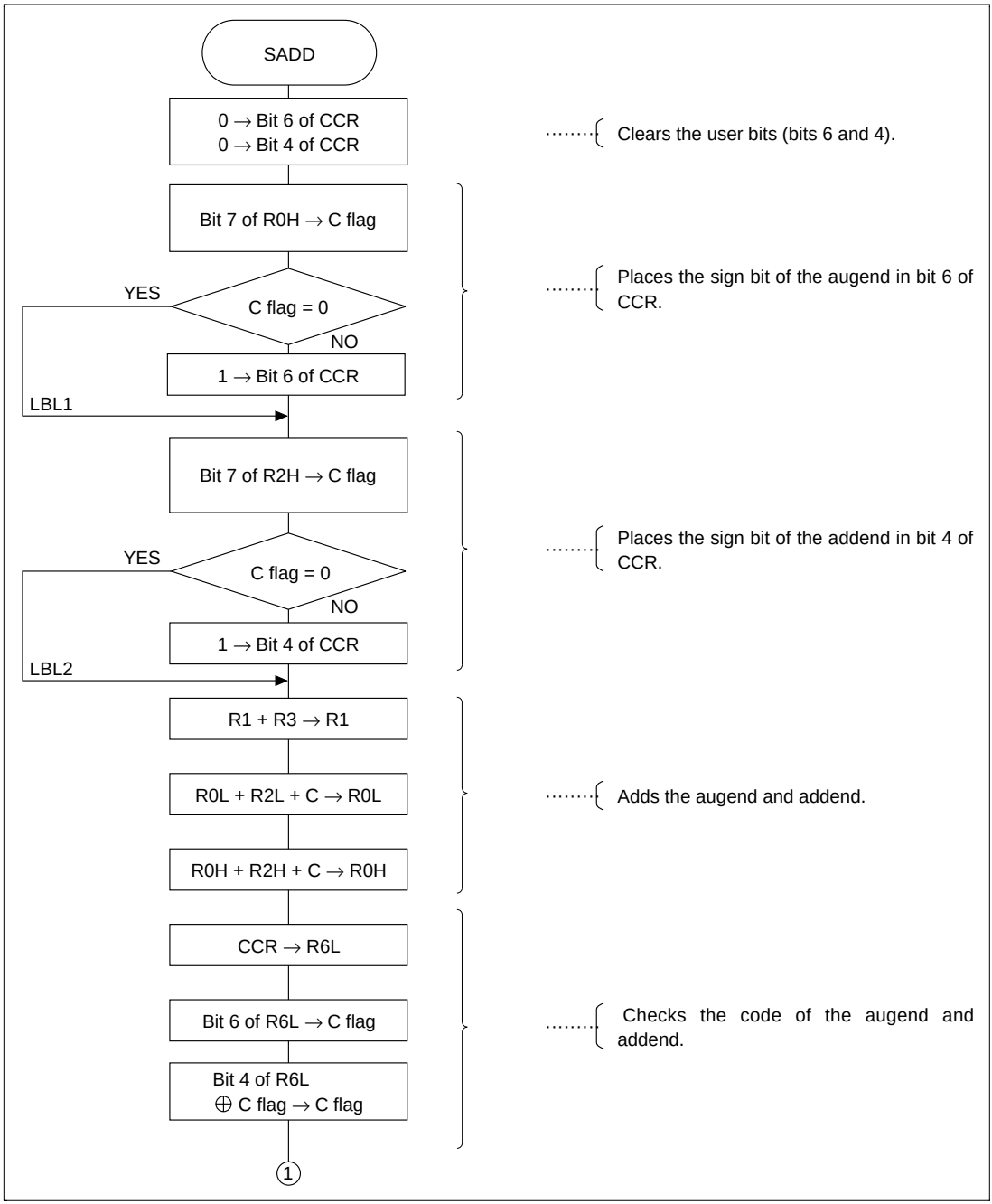
$$\left.\begin{array}{l} R1 + R3 \rightarrow R1 \\ R0L + R2L + C \rightarrow R0L \\ R0H + R2H + C \rightarrow R0H \end{array}\right\} \cdots\cdots\cdots \text{equation 1}$$

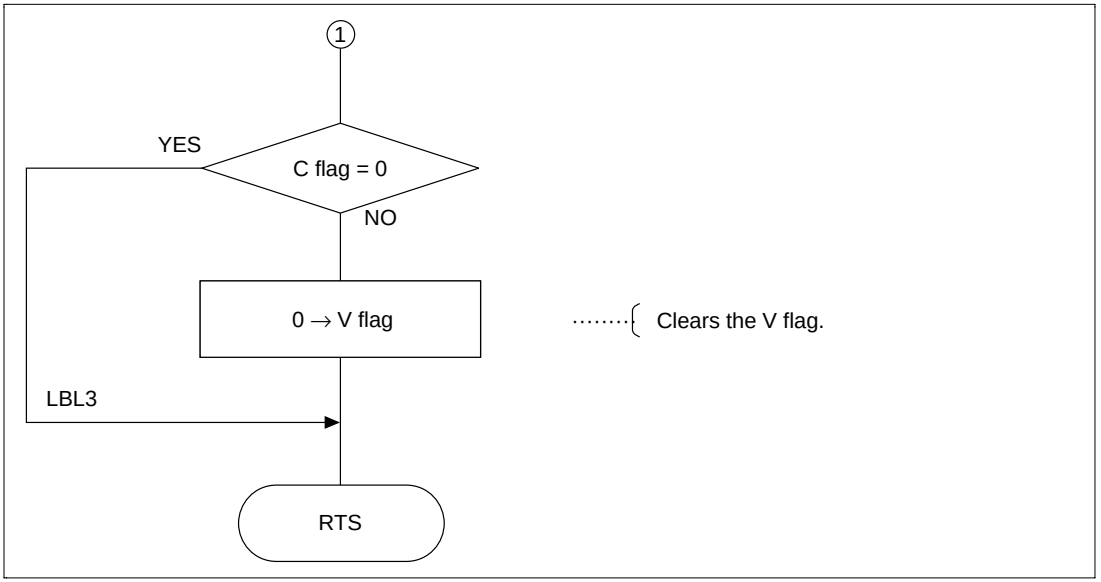
      (v) Whether to continue processing or clear the V flag is determined depending on the state of the sign bit (CCR user bit):

<Sign bit>

Bit 6 of CCR (Augend)	Bit 4 of CCR (Addend)	
0	0	→ Continues processing.
0	1	→ Clears the V flag.
1	0	
1	1	→ Continues processing.

7.13.6 Flowchart




1
YES
C flag = 0
NO
0 → V flag
Clears the V flag.
LBL3
RTS

7.13.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:15:08
PROGRAM NAME =
    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME       :SIGNED 32 BIT BINARY ADDITION (SADD)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     (UPPER WORD OF SUMMAND)
    8                                   ;*                    R1     (LOWER WORD OF SUMMAND)
    9                                   ;*                    R2     (UPPER WORD OF ADDEND)
   10                                   ;*                    R3     (LOWER WORD OF ADDEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R0     (UPPER WORD OF RESULT)
   13                                   ;*                    R1     (LOWER WORD OF RESULT)
   14                                   ;*                    V FLAG OF CCR
   15                                   ;*                           (V=0;TRUE,V=1:OVERFLOW OR UNDERFLOW)
   16                                   ;*
   17                                   ;*********************************************************************
   18                                   ;
   19 SADD_cod C 0000                        .SECTION        SADD_code,CODE,ALIGN=2
   20                                        .EXPORT         SADD
   21                                   ;
   22 SADD_cod C      00000000       SADD .EQU    $               ;Entry point
   23 SADD_cod C 0000 06AD                   ANDC    #H'AD,CCR       ;Clear user bits and V flag of CCR
   24 SADD_cod C 0002 7770                   BLD     #7,R0H          ;Load sign bit of summand
   25 SADD_cod C 0004 4402                   BCC     LBL1            ;Branch if C=0
   26 SADD_cod C 0006 0440                   ORC.B   #H'40,CCR       ;Bit set user bit (bit 6 of CCR)
   27 SADD_cod C 0008                   LBL1
   28 SADD_cod C 0008 7772                   BLD     #7,R2H          ;Load sign bit of addend
   29 SADD_cod C 000A 4402                   BCC     LBL2            ;Branch if C=0
   30 SADD_cod C 000C 0410                   ORC.B   #H'10,CCR       ;Bit set user bit (bit 4 of CCR)
   31 SADD_cod C 000E                   LBL2
   32 SADD_cod C 000E 0931                   ADD.W   R3,R1           ;R3  + R1      -> R1
   33 SADD_cod C 0010 0EA8                   ADDX.B  R2L,R0L         ;R2L + R0L + C -> R0L
   34 SADD_cod C 0012 0E20                   ADDX.B  R2H,R0H         ;R2H + R0H + C -> R0H
   35 SADD_cod C 0014 020E                   STC     CCR,R6L         ;CCR -> R6L
   36 SADD_cod C 0016 776E                   BLD     #6,R6L          ;Bit load bit 4 of R6L
   37 SADD_cod C 0018 754E                   BXOR    #4,R6L          ;Bit exclusive OR sign bits
   38 SADD_cod C 001A 4402                   BCC     LBL3            ;Barnch if C=0
   39 SADD_cod C 001C 06FD                   ANDC.B  #H'FD,CCR       ;Clear V flag
   40 SADD_cod C 001E                   LBL3
   41 SADD_cod C 001E 5470                   RTS
   42                                   ;
   43                                        .END
  *****TOTAL ERRORS       0

  *****TOTAL WARNINGS     0
```

7.14 Multiplication of Signed 16-Bit Binary Numbers

MCU: H8/300 Series
H8/300L Series

Label name: SMUL

7.14.1 Function

1. The software SMUL multiplies a signed 16-bit binary number to another signed 162-bit binary number and places the result in a general-purpose register.
2. The arguments used with the software SMUL are signed integers.
3. All data is manipulated on general-purpose registers.

7.14.2 Arguments

Description		Memory area	Data length (bytes)
Input	Multiplicand	R1	2
	Multiplier	R0	2
Output	Result of multiplication	R1, R2	4

7.14.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6H	R6L	R7
×	↕	↕	×	×	•	•	×	•

I	U	H	U	N	Z	V	C
•	×	×	×	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.14.4 Specifications

Program memory (bytes)
52
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
132
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.14.5 Notes

The clock cycle count (132) in the specifications is a maximum cycle count.

7.14.6 Description

1. Details of functions
   a. The following arguments are used with the software SMUL:
      (i) Input arguments:
         R0: Contains a signed 16-bit binary multiplier.
         R1: Contains a signed 162-bit binary multiplicand.
      (ii) Output arguments
         R1, R2: Contain the result of multiplication (a signed 16-bit binary number)
   b. Figure 7.40 shows an example of the software SMUL being executed. When the input arguments are set as shown in (1), the result of multiplication is placed in R1 and R2 as shown in (2).

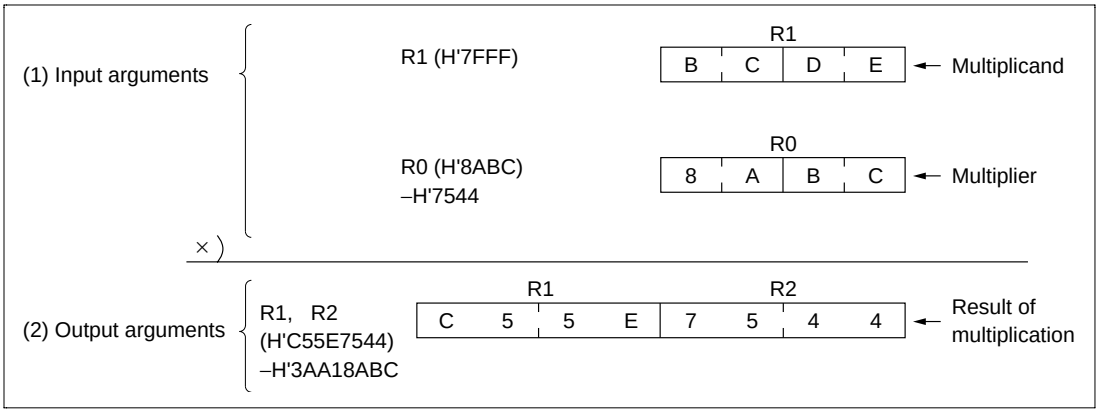


Figure 7.40 Example of Software SMUL Execution

2. Notes on usage
   a. When upper bits are not used (see figure 7.41), set 0's in them; otherwise, no correct result can be obtained because multiplication is done on the numbers including indeterminate data placed in the upper bits. (The upper bits referred to here do not include sign bits.)

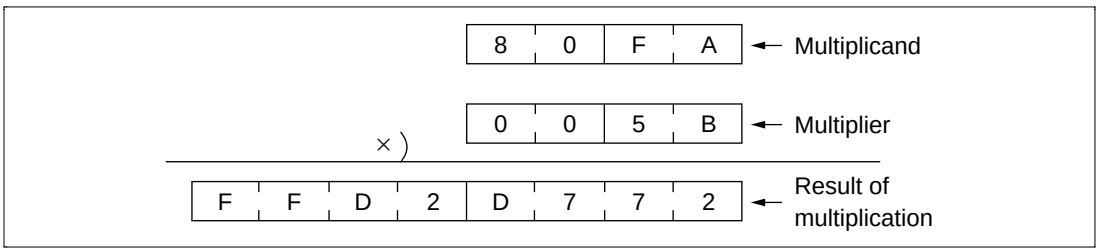


Figure 7.41 Example of Multiplication with Upper Bits Unused

b. After execution of the software SMUL, the multiplicand is destroyed because the upper 2 bytes of the result are placed in R1. If the multiplicand is necessary after software SMUL execution, save it on memory.

3. Data memory

   The software SMUL does not use the data memory.

4. Example of use

   Set a multiplicand and a multiplier in the input arguments and call the software SMUL as a subroutine.

```
WORK1    . RES. W   2          ..... Reserves a data memory area in which the
                                      user program places a signed 16-bit binary
                                      multiplicand.
WORK2    . RES. W   2          ..... Reserves a data memory area in which the
                                      user program places a 16-bit binary
                                      multiplier.
WORK3    . RES. W   2          ..... Reserves a data memory area for storage
                                      of the result of multiplication.
           :
         MOV. W     @WORK1, R1 ..... Places in R1 the 16-bit binary multiplicand
                                      set by the user program.
         MOV. W     @WORK2, R0 ..... Places in R0 the 16-bit binary multiplier set
                                      by the user program.
         JSR        @SMUL      ..... Calls the software SMUL as a subroutine.
         MOV. W     R1, @WORK3 }
         MOV. W     R2, @WORK3+2} ..... Places the result (set in the output
                                        argument) in the data memory area of the
                                        user program.
           :
```

5. Operation

   a. Subtraction of signed 16-bit binary numbers is done in one of the following manners depending on the signs of the multiplicand and multiplier:

(Multiplicand)	(Multiplier)		(Process)
(+)	(+)	→	Multiplied directly.
(+)	(–)	→	Multiplied with the sign of the multiplier inverted.
(–)	(+)	→	Multiplied with the sign of the multiplicand inverted.
(–)	(–)	→	Multiplied with the signs of both multiplicand and multiplie

b. The multiplication steps are as follows:

(i) A multiplicand is placed in R1 and a multiplier in R0.

(ii) The user bit (CCR) is cleared.

(iii) If the multiplicand is negative, its sign is inverted. If the multiplier is negative, its sign bit is inverted. Bits 6 and 4 of the CCR (user bits) are used as the sign bits of the multiplicand and multiplier, respectively. If the multiplicand or multiplier is negative, "1" is set in the corresponding user bit.

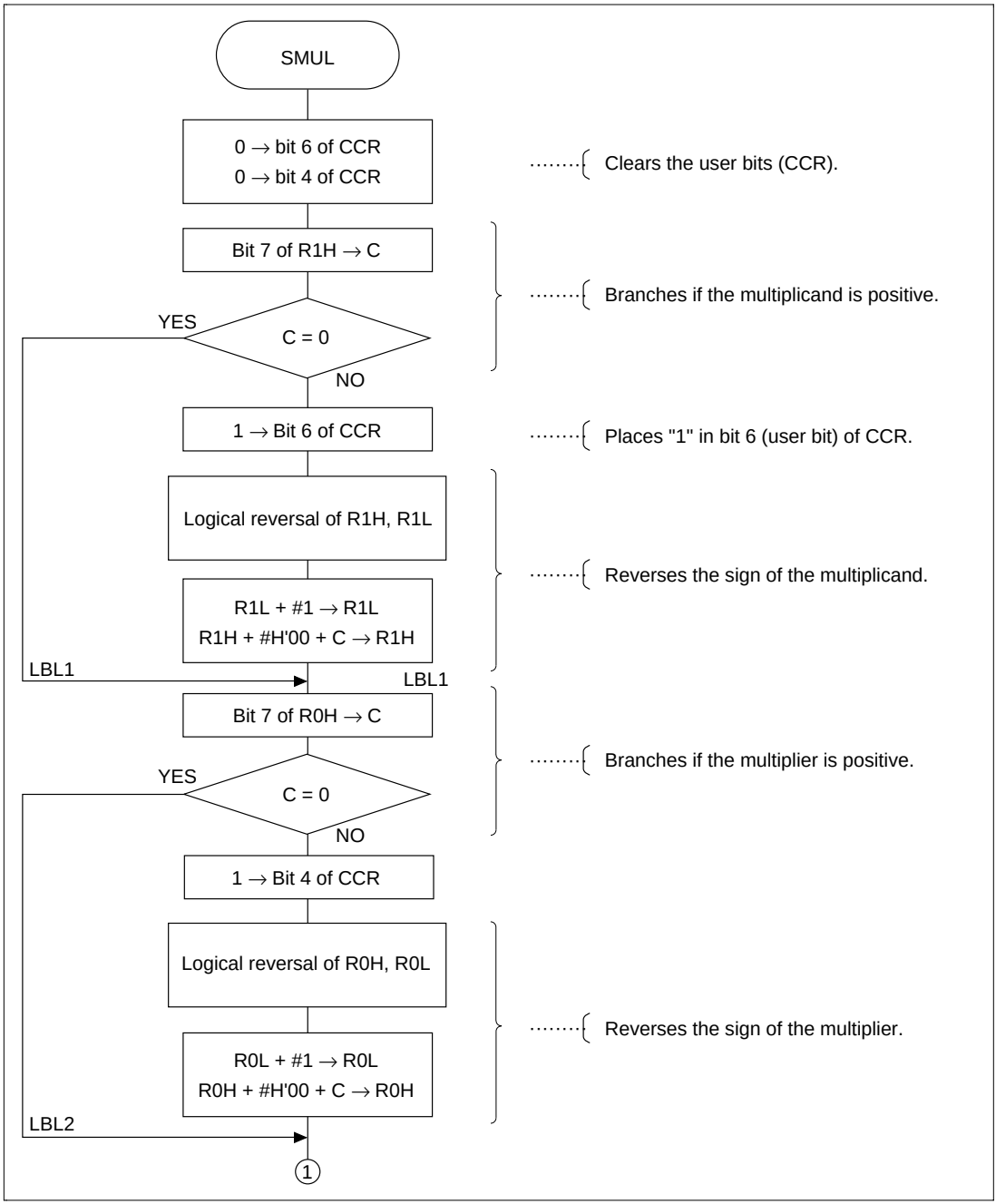
(iv) Multiplication is done with the software MUL.

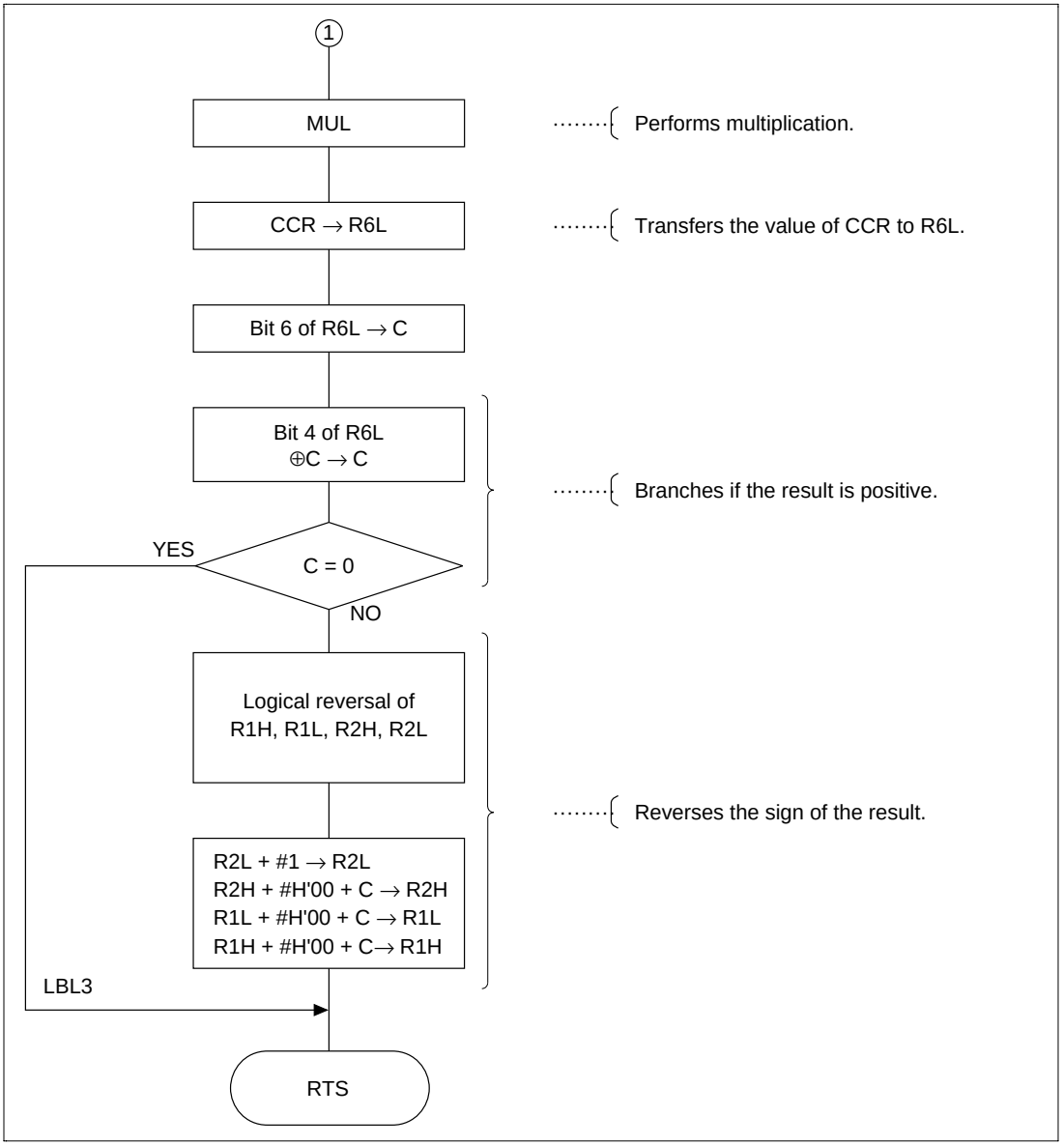
(v) The CCR is transferred to R6L.

(vi) The result is modified or unmodified depending on the signs of the multiplicand and multiplier, as follows:

(Multiplicand)	(Multiplier)	
(+)	(+)	→ The result is unmodified.
(+)	(–)	
(–)	(+)	→ The result has its sign inverted.
(–)	(–)	

7.14.7 Flowchart




1
MUL
Performs multiplication.
CCR → R6L
Transfers the value of CCR to R6L.
Bit 6 of R6L → C
Bit 4 of R6L
⊕C → C
Branches if the result is positive.
YES
C = 0
NO
Logical reversal of
R1H, R1L, R2H, R2L
Reverses the sign of the result.
R2L + #1 → R2L
R2H + #H'00 + C → R2H
R1L + #H'00 + C → R1L
R1H + #H'00 + C→ R1H
LBL3
RTS

7.14.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:16:51
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME       :SIGNED 16 BIT BINARY MULTIPLICATION (SMUL)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENRTRY          :R1     (MULTIPLICAND)
    8                                   ;*                    R0     (MULTIPLIER)
    9                                   ;*
   10                                   ;*   RETURNS         :R1     (UPPER WORD OF RESULT)
   11                                   ;*                    R2     (LOWER WORD OF RESULT)
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 SMUL_cod C 0000                        .SECTION        SMUL_code,CODE,ALIGN=2
   16                                        .EXPORT         SMUL
   17                                   ;
   18 SMUL_cod C      00000000          SMUL .EQU    $               ;Entry point
   19 SMUL_cod C 0000 06AD                   ANDC.B  #H'AD,CCR       ;Clear user bits
   20 SMUL_cod C 0002 7771                   BLD     #7,R1H          ;Load sign bit of multiplicand
   21 SMUL_cod C 0004 4408                   BCC     LBL1            ;Branch if C=0
   22 SMUL_cod C 0006 0440                   ORC.B   #H'40,CCR       ;Bit set user bit (bit 6 of CCR)
   23 SMUL_cod C 0008 1701                   NOT     R1H             ;2's complement multiplicand
   24 SMUL_cod C 000A 1709                   NOT     R1L
   25 SMUL_cod C 000C 0B01                   ADDS.W  #1,R1
   26 SMUL_cod C 000E                   LBL1
   27 SMUL_cod C 000E 7770                   BLD     #7,R0H          ;Load sign bit of multiplier
   28 SMUL_cod C 0010 4408                   BCC     LBL2            ;Branch if C=0
   29 SMUL_cod C 0012 0410                   ORC.B   #H'10,CCR       ;Bit set user bit (bit 4 of CCR)
   30 SMUL_cod C 0014 1700                   NOT     R0H             ;2's complement multiplier
   31 SMUL_cod C 0016 1708                   NOT     R0L
   32 SMUL_cod C 0018 0B00                   ADDS.W  #1,R0
   33 SMUL_cod C 001A                   LBL2
   34 SMUL_cod C 001A 0C9A                   MOV.B   R1L,R2L         ;
   35 SMUL_cod C 001C 0C1C                   MOV.B   R1H,R4L         ;
   36 SMUL_cod C 001E 0C9B                   MOV.B   R1L,R3L         ;
   37 SMUL_cod C 0020 0C19                   MOV.B   R1H,R1L         ;
   38 SMUL_cod C 0022 5082                   MULXU   R0L,R2          ;R0L * R2L -> R2
   39 SMUL_cod C 0024 5084                   MULXU   R0L,R4          ;R0L * R4L -> R4
   40 SMUL_cod C 0026 5003                   MULXU   R0H,R3          ;R0H * R3L -> R3
   41 SMUL_cod C 0028 5001                   MULXU   R0H,R1          ;R0H * R1L -> R1
   42 SMUL_cod C 002A 08C2                   ADD.B   R4L,R2H         ;R2H + R4L       -> R2H
   43 SMUL_cod C 002C 9400                   ADDX.B  #H'00,R4H       ;R4H + #H'00 + C -> R4H
   44 SMUL_cod C 002E 0839                   ADD.B   R3H,R1L         ;R1L + R3L       -> R1L
   45 SMUL_cod C 0030 9100                   ADDX.B  #H'00,R1H       ;R1H + #H'00 + C -> R1H
   46 SMUL_cod C 0032 08B2                   ADD.B   R3L,R2H         ;R2H + R3L       -> R2H
   47 SMUL_cod C 0034 0E49                   ADDX.B  R4H,R1L         ;R1L + R4H   + C -> R1L
   48 SMUL_cod C 0036 9100                   ADDX.B  #H'00,R1H       ;R1H + #H'00 + C -> R1H
   49                                   ;
   50 SMUL_cod C 0038 020E                   STC     CCR,R6L         ;CCR -> R6L
   51 SMUL_cod C 003A 776E                   BLD     #6,R6L          ;Load sign bit of multiplicand
   52 SMUL_cod C 003C 754E                   BXOR    #4,R6L          ;Bit exclusive OR sign bits
   53 SMUL_cod C 003E 4410                   BCC     LBL3            ;Branch if C=0
   54 SMUL_cod C 0040 1701                   NOT     R1H             ;2's complement sign bits
   55 SMUL_cod C 0042 1709                   NOT     R1L             ;
   56 SMUL_cod C 0044 1702                   NOT     R2H             ;
   57 SMUL_cod C 0046 170A                   NOT     R2L             ;
   58 SMUL_cod C 0048 8A01                   ADD.B   #1,R2L          ;
   59 SMUL_cod C 004A 9200                   ADDX.B  #H'00,R2H       ;
   60 SMUL_cod C 004C 9900                   ADDX.B  #H'00,R1L       ;
   61 SMUL_cod C 004E 9100                   ADDX.B  #H'00,R1H       ;
   62 SMUL_cod C 0050                   LBL3
   63 SMUL_cod C 0050 5470                   RTS
   64                                   ;
   65                                        .END
 *****TOTAL ERRORS       0
 *****TOTAL WARNINGS     0
```

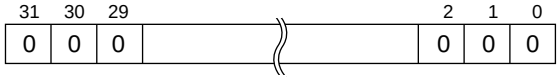
About Short Floating-Point Numbers

Formats of Short Floating-Point Numbers

1. Internal representation of short floating-point numbers

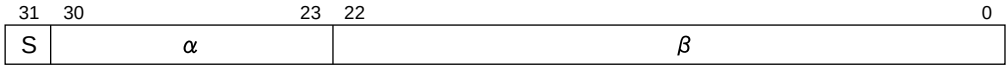
   For purposes of this Application Note, the following formats of representation apply to short floating-point numbers (R = real number):

   a. Internal representation for R = 0



   All of the 32 bits are 0's.

   b. Normalized format

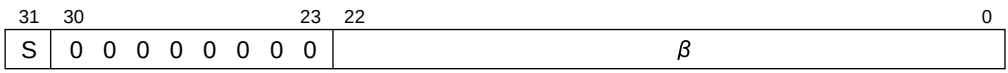


   α is an exponent whose field is 8 bits long. β is a mantissa whose field is 32 bits long. The value of R can be represented by the following equation (on conditions that 1≤α≤254:

$$R = 2^{S} \times 2^{\alpha-127} \times (1 + 2^{-1} \times \beta_{22} + 2^{-2} \times \beta_{21} + \cdots\cdots + 2^{-23} \times \beta_{0})$$

   where βi is the value of the i-th bit (0 ≤ i ≤ 22) and S is a sign bit.

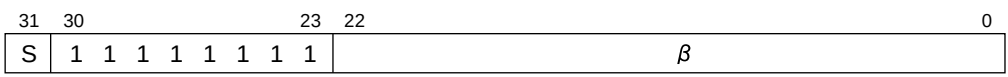
   c. Denormalized format



   where α is a mantissa whose field is 23 bits long. This format is used to represent a real number too small to be represented in the normal format. In this format, R can be represented by the following equation:

$$R = 2^{S} \times 2^{\alpha-126} \times (2^{-1} \times \beta_{22} + 2^{-2} \times \beta_{21} + \cdots\cdots + 2^{-23} \times \beta_{0})$$

   d. Infinity



   where β is a mantissa whose field is 32 bits long. In this Application Note, however, the following rules apply if all exponents are 1's;

   Positive infinity when S = 0

   $R = +\infty$

   Negative infinity when S = 1

   $R = -\infty$

2. Example of internal representation

If S = B'0 (binary)

α = B'10000011 (binary)

β = B'1011100……0 (binary)

Then the corresponding real number is as follow:

$$R = 2^{0} \times 2^{131-127} \times (1 + 2^{-1} + 2^{-2} + 2^{-3} + \cdots\cdots + 2^{-5})$$

$$= 16 + 8 + 2 + 1 + 0.5 = 27.5$$

a. Maximum and minimum values

Up to the following absolute maximum value (Rmax) and minimum value (Rmin) can be represented;

$$R_{MAX} = 2^{254-127} \times (1 + 2^{-1} + 2^{-3} + 2^{-4} \cdots\cdots + 2^{-5})$$

$$= 3.37 \times 1038$$

$$R_{MIN} = 2^{-128} \times 2^{-23} = 2^{-140} \fallingdotseq 1.40 \times 10^{-45}$$

7.15 Change of a Short Floating-Point Number to a Signed 32-Bit Binary Number

MCU: H8/300 Series
H8/300L Series

Label name: FKTR

7.15.1 Function

1. The software FKTR changes a short floating-point number (placed in a general-purpose register) to a signed 32-bit binary number.
2. "0"is output when the short floating-point number is "0".
3. When the short floating-point number is not less than $|2^{31}|$, a maximum value ($2^{31} - 1$ or -2^{31}) with the same sign as that number is output. When the short floating-point number is not more than |1|, "0" is output.

7.15.2 Arguments

Description		Memory area	Data length (bytes)
Input	Short floating-point number	R0, R1	4
Output	Signed 32-bit binary number	R2, R3	4

7.15.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	↕	↕	•	×	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.15.4 Specifications

Program memory (bytes)
100
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
108
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.15.5 Notes

The clock cycle count (108) in the specifications is for the example shown in figure 7.42

For the format of floating-point numbers, see "About Short Floating-point Numbers <Reference>."

7.15.6 Description

1. Details of functions
   a. The following arguments are used with the software FKTR:
      (i) Input arguments:
         R0: Contains the upper 2 bytes of a short floating-point number.
         R1: Contains the lower 2 bytes of the short floating-point number.
      (ii) Output arguments
         R2: Contains the upper 2 bytes of a signed 32-bit binary number.

R3: Contains the lower 2 bytes of the signed 32-bit binary number.

b. Figure 7.42 shows an example of the software FKTR being executed. When the input arguments are set as shown in (1), the result of change is placed in R2 and R3 as shown in (2).

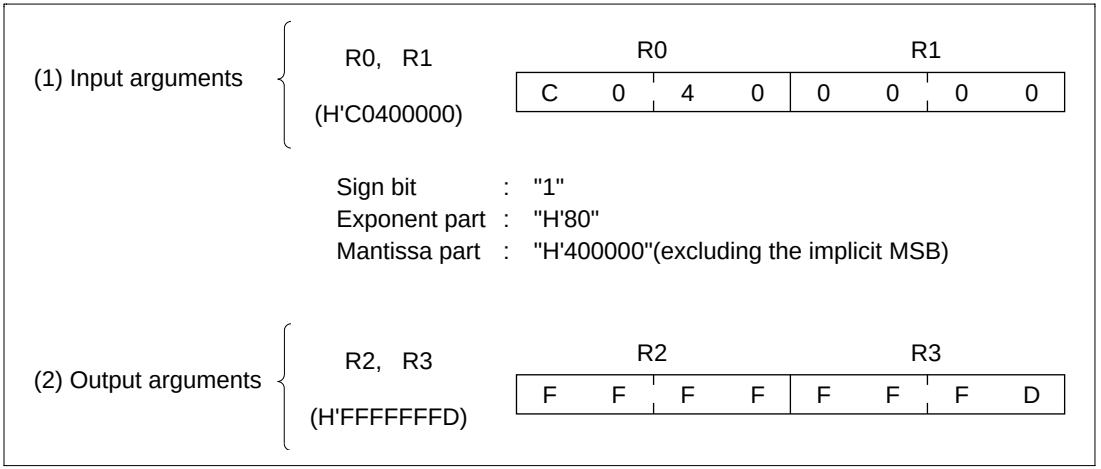


Figure 7.42 Example of Software FKTR Execution

2. Notes on usage
   a. When the short floating-point number is "0" or not more than |1|, "0" is output.
   b. When the short floating-point number is not less than $|2^{31}|$, a maximum value with the same sign (H'7FFFFFFF or H'80000000) is output.
   c. After execution of the software FKTR, the input arguments placed in R0 and R1 are destroyed. If the input arguments are necessary after software FKTR execution, save them on memory.
3. Data memory

   The software FKTR does not use the data memory.

4. Example of use

Set a short floating-point number in the general-purpose register and call the software FKTR as a subroutine.

```
WORK1     . DATA. W   2,  0          ..... Reserves a data memory area in which the
                                           user program places a short floating-point
                                           number.
WORK2     . DATA. W   2,  0          ..... Reserves a data memory area in which the
                                           user program places a signed 32-bit binary
                                           number.
              :
          MOV. B      @WORK1, R0     ..... Places in R0 and R1 the short floating-
          MOV. W      @WORK1+2, R1           point number set by the user program.

          JSR         @FKTR          ..... Calls the software FKTR as a subroutine.

          MOV. W      R2, @WORK2     ..... Places in R2 and R3 the signed 32-bit
          MOV. W      R3, @WORK2+2           binary number set in the output argument.
              :
```

5. Operation

a. The software FKTR takes the following steps to change the short floating-point number to a signed 32-bit binary number:

b. First, the input argument is checked.

(i) If the input argument is "0", then "0"is output.

(ii) If the exponent part is smaller than "H'7F", then "0"is output.

(iii) If the exponent part is not less than "H'9E", a maximum value with the same sign is output.

c. Next, if the input argument is not "0" and its absolute value is not less than "1" (the exponent part=H'7F) and smaller than 2^{31} (the exponent part = H'9E), the following operations are performed;

(i) The implicit MSB is set.

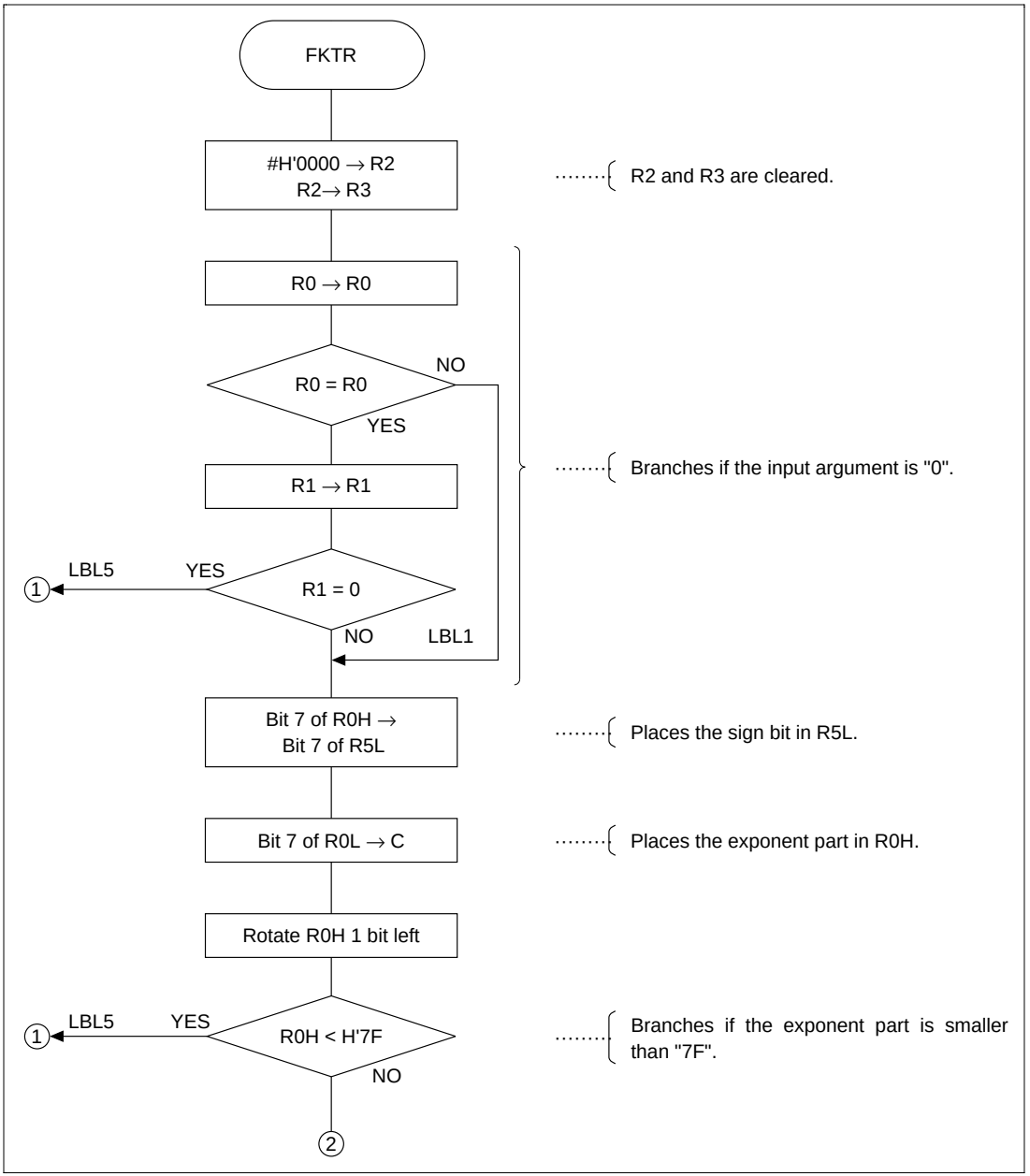
(ii) The mantissa part (24 bits long) in which the implicit MSB is contained is shifted 1 bit to the left.

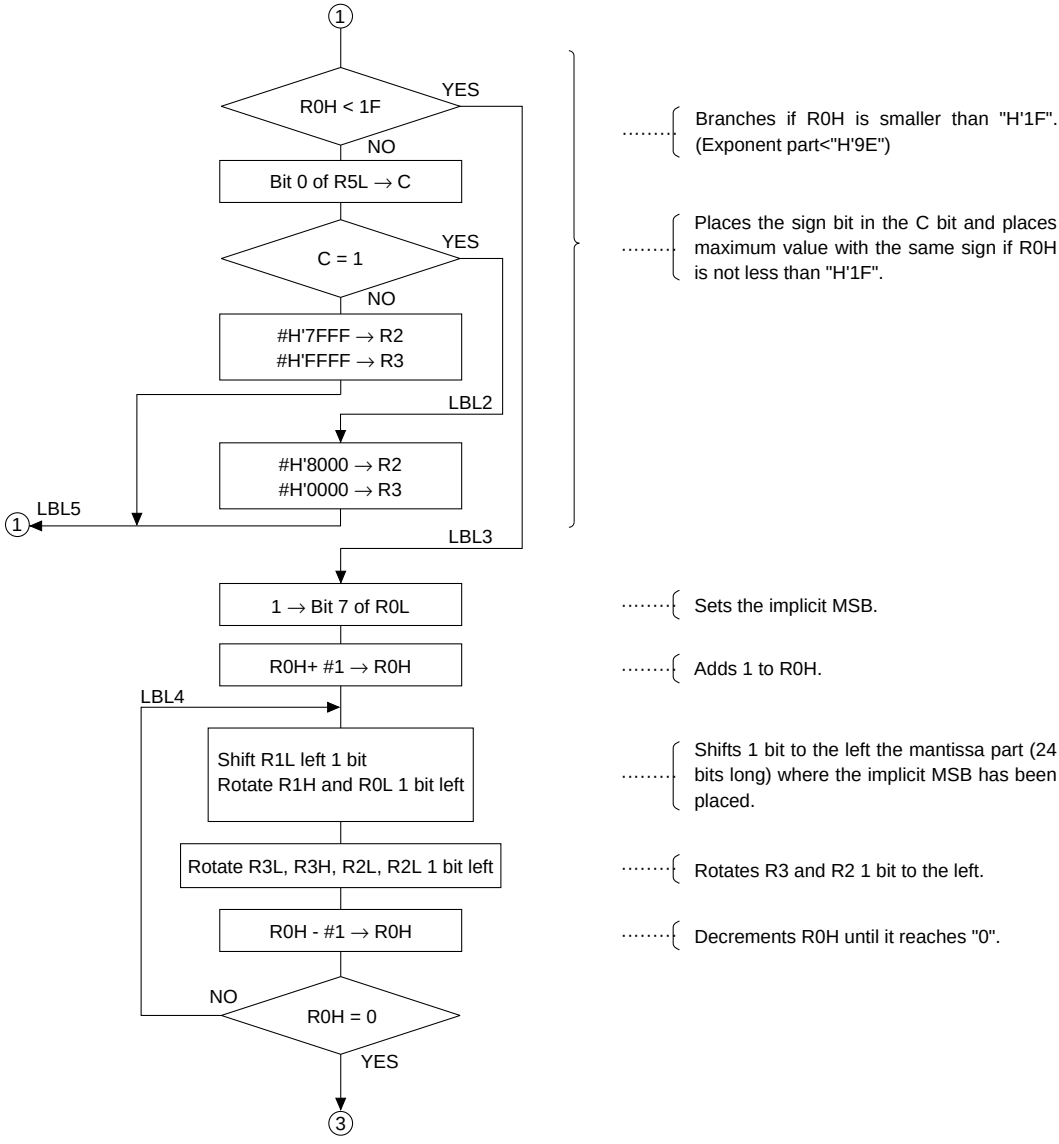
(iii) R3 and R2 are rotated 1 bit to the left.

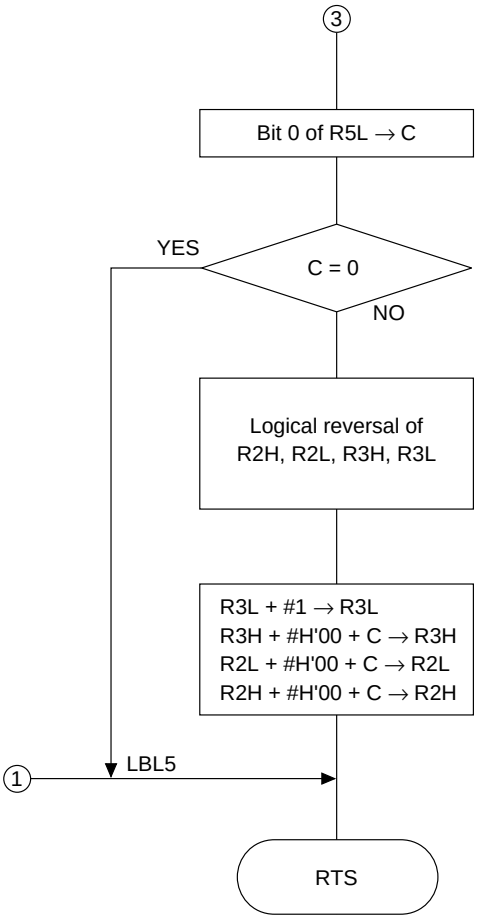
(iv) Steps (ii) and (iii) are repeated as many times as "R0H+1".

(v) A negative number is obtained by two's complement when the sign bit is negative.

7.15.7 Flowchart




1
R0H < 1F
YES
NO
Bit 0 of R5L → C
C = 1
YES
NO
#H'7FFF → R2
#H'FFFF → R3
LBL2
#H'8000 → R2
#H'0000 → R3
1
LBL5
LBL3
1 → Bit 7 of R0L
R0H+ #1 → R0H
LBL4
Shift R1L left 1 bit
Rotate R1H and R0L 1 bit left
Rotate R3L, R3H, R2L, R2L 1 bit left
R0H - #1 → R0H
NO
R0H = 0
YES
3
Branches if R0H is smaller than "H'1F". (Exponent part<"H'9E")
Places the sign bit in the C bit and places maximum value with the same sign if R0H is not less than "H'1F".
Sets the implicit MSB.
Adds 1 to R0H.
Shifts 1 bit to the left the mantissa part (24 bits long) where the implicit MSB has been placed.
Rotates R3 and R2 1 bit to the left.
Decrements R0H until it reaches "0".



Branches if the sign bit is "0" (a positive number).

Takes on two's complement of the 32-bit binary number (placed in R2 and R3) to obtain a negative number.

7.15.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:17:31
PROGRAM NAME =

    1                                ;*****************************************************************
    2                                ;*
    3                                ;*   00 - NAME       :CHANGE FLOATING POINT TO 32 BIT BINARY
    4                                ;*                    (FKTR)
    5                                ;*
    6                                ;*****************************************************************
    7                                ;*
    8                                ;*   ENTRY           :R0 (UPPER WORD OF FLOATING POINT)
    9                                ;*                    R1 (LOWER WORD OF FLOATING POINT)
   10                                ;*
   11                                ;*   RETURNS         :R2     (UPPER WORD OF 32 BIT BINARY)
   12                                ;*                    R3     (LOWER WORD OF 32 BIT BINARY)
   13                                ;*
   14                                ;*****************************************************************
   15                                ;
   16 FKTR_cod C 0000                     .SECTION        FKTR_code,CODE,ALIGN=2
   17                                     .EXPORT         FKTR
   18                                ;
   19 FKTR_cod C      00000000       FKTR .EQU    $               ;Entry point
   20 FKTR_cod C 0000 79020000            MOV.W   #H'0000,R2      ;Clear R2
   21 FKTR_cod C 0004 0D23                MOV.W   R2,R3           ;Clear R3
   22                                ;
   23 FKTR_cod C 0006 0D00                MOV.W   R0,R0
   24 FKTR_cod C 0008 4604                BNE     LBL1
   25 FKTR_cod C 000A 0D11                MOV.W   R1,R1
   26 FKTR_cod C 000C 4754                BEQ     LBL5            ;Branch if R0=R1=0
   27 FKTR_cod C 000E                LBL1
   28 FKTR_cod C 000E 7770                BLD     #7,R0H
   29 FKTR_cod C 0010 670D                BST     #0,R5L          ;Set sign bit to bit 0 of R5L
   30 FKTR_cod C 0012 7778                BLD     #7,R0L
   31 FKTR_cod C 0014 1200                ROTXL.B R0H             ;Set expornent
   32 FKTR_cod C 0016 F57F                MOV.B   #H'7F,R5H
   33 FKTR_cod C 0018 1850                SUB.B   R5H,R0H
   34 FKTR_cod C 001A 4546                BCS     LBL5            ;Branch if R0H<"H'7F"
   35 FKTR_cod C 001C A01F                CMP.B   #H'1F,R0H
   36 FKTR_cod C 001E 4518                BCS     LBL3            ;Branch if R0H<"H'1F"
   37 FKTR_cod C 0020 770D                BLD     #0,R5L
   38 FKTR_cod C 0022 450A                BCS     LBL2            ;Branch if sign bit = 1
   39 FKTR_cod C 0024 79027FFF            MOV.W   #H'7FFF,R2
   40 FKTR_cod C 0028 7903FFFF            MOV.W   #H'FFFF,R3      ;Set "H'7FFFFFFF"
   41 FKTR_cod C 002C 4034                BRA     LBL5            ;Branch always
   42 FKTR_cod C 002E                LBL2
   43 FKTR_cod C 002E 79028000            MOV.W   #H'8000,R2
   44 FKTR_cod C 0032 79030000            MOV.W   #H'0000,R3      ;Set "H'80000000"
   45 FKTR_cod C 0036 402A                BRA     LBL5
   46                                ;
   47 FKTR_cod C 0038                LBL3
   48 FKTR_cod C 0038 7078                BSET    #7,R0L          ;Set implicit MSB
   49 FKTR_cod C 003A 8001                ADD.B   #1,R0H          ;R0H + #1 -> R0H
   50 FKTR_cod C 003C                LBL4
   51 FKTR_cod C 003C 1009                SHLL.B  R1L             ;Shift mantissa 1 bit left
   52 FKTR_cod C 003E 1201                ROTXL.B R1H
   53 FKTR_cod C 0040 1208                ROTXL.B R0L
   54                                ;
   55 FKTR_cod C 0042 120B                ROTXL.B R3L             ;Rotate 32 bit binary 1 bit left
   56 FKTR_cod C 0044 1203                ROTXL.B R3H
   57 FKTR_cod C 0046 120A                ROTXL.B R2L
   58 FKTR_cod C 0048 1202                ROTXL.B R2H
   59 FKTR_cod C 004A 1A00                DEC.B   R0H             ;Decrement R0H
   60 FKTR_cod C 004C 46EE                BNE     LBL4            ;Branch if Z=0
   61                                ;
   62 FKTR_cod C 004E 770D                BLD     #0,R5L          ;Bit load sign bit to C flag
   63 FKTR_cod C 0050 4410                BCC     LBL5            ;Branch if C=0
   64 FKTR_cod C 0052 1702                NOT     R2H             ;2's complement 32 bit binary
```

```
 65 FKTR_cod C 0054 170A                    NOT     R2L
 66 FKTR_cod C 0056 1703                    NOT     R3H
 67 FKTR_cod C 0058 170B                    NOT     R3L
 68 FKTR_cod C 005A 8B01                    ADD.B   #H'01,R3L
 69 FKTR_cod C 005C 9300                    ADDX.B  #H'00,R3H
 70 FKTR_cod C 005E 9A00                    ADDX.B  #H'00,R2L
 71 FKTR_cod C 0060 9200                    ADDX.B  #H'00,R2H
 72 FKTR_cod C 0062                   LBL5
 73 FKTR_cod C 0062 5470                    RTS
 74                                   ;
 75                                         .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

7.16 Change of a Signed 32-Bit Binary Number to a Short Floating-Point Number

MCU: H8/300 Series
H8/300L Series

Label name: KFTR

7.16.1 Function

1. The software KFTR changes a signed 32-bit binary number (placed in a general-purpose register) to a short floating-point number.

7.16.2 Arguments

Description		Memory area	Data length (bytes)
Input	Signed 32-bit binary number	R0, R1	4
Output	Short floating-point y number	R0, R1	4

7.16.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	•	•	×	×	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.16.4 Specifications

Program memory (bytes)
98
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
346
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.16.5 Notes

The clock cycle count (346) in the specifications is for the example shown in figure 7.43.

For the format of floating-point numbers, see "About Short Floating-point Numbers <Reference>."

7.16.6 Description

1. Details of functions
   a. The following arguments are used with the software KFTR:
      (i) Input arguments:
      R0: Contains the upper 2 bytes of a signed 32-bit binary number.
      R1: Contains the lower 2 bytes of the signed 32-bit binary number.

(ii) Output arguments

R2: Contains the upper 2 bytes of a short floating-point number.

R3: Contains the lower 2 bytes of the short floating-point number.

b. Figure 7.43 shows an example of the software KFTR being executed. When the input arguments are set as shown in (1), the result of change is placed in R0 and R1 as shown in (2).

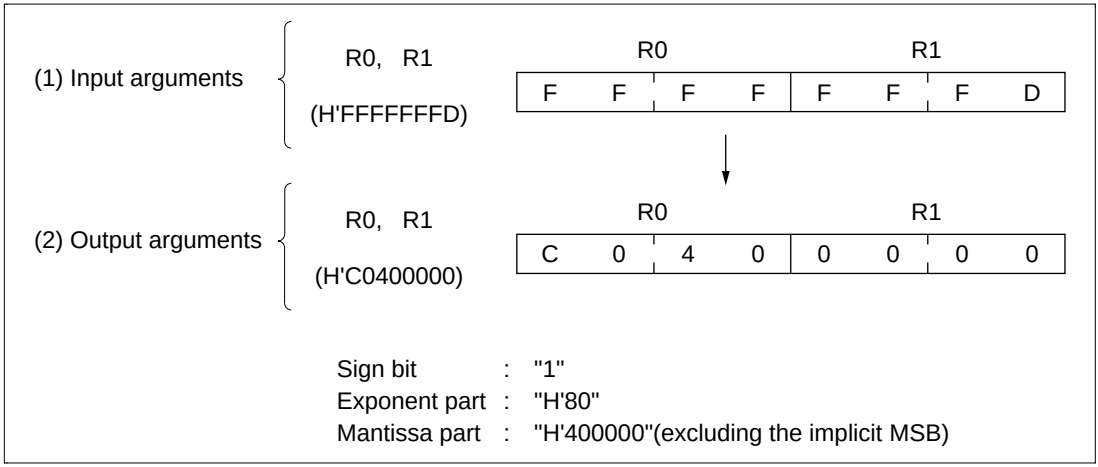


Figure 7.43 Example of Software KFTR Execution

2. Notes on usage

a. After execution of the software KFTR, the signed 32-bit binary number is destroyed because the result of change is placed in R0 and R1. If the signed 32-bit binary number is necessary after software KFTR execution, save it on memory.

3. Data memory

The software KFTR does not use the data memory.

4. Example of use

Set a signed 32-bit binary number in the general-purpose register and call the software KFTR as a subroutine.

```
WORK1     . DATA. W   2,  0        ..... Reserves a data memory area in which the
                                         user program places a signed 32-bit binary
                                         number.
WORK2     . DATA. W   2,  0        ..... Reserves a data memory area in which the
                                         user program places a short floating-point
                                         number.
               :
          MOV. W     @WORK1, R0    ..... Places in R0 and R1 the signed 32-bit
          MOV. W     @WORK1+2, R1        binary number set by the user program.

          JSR        @KFTR         ..... Calls the software KFTR as a subroutine.

          MOV. W     R0, @WORK2    ..... Places in R0 and R1 the short floating-
          MOV. W     R1, @WORK2+2        point number set in the output argument.
               :
```

5. Operation

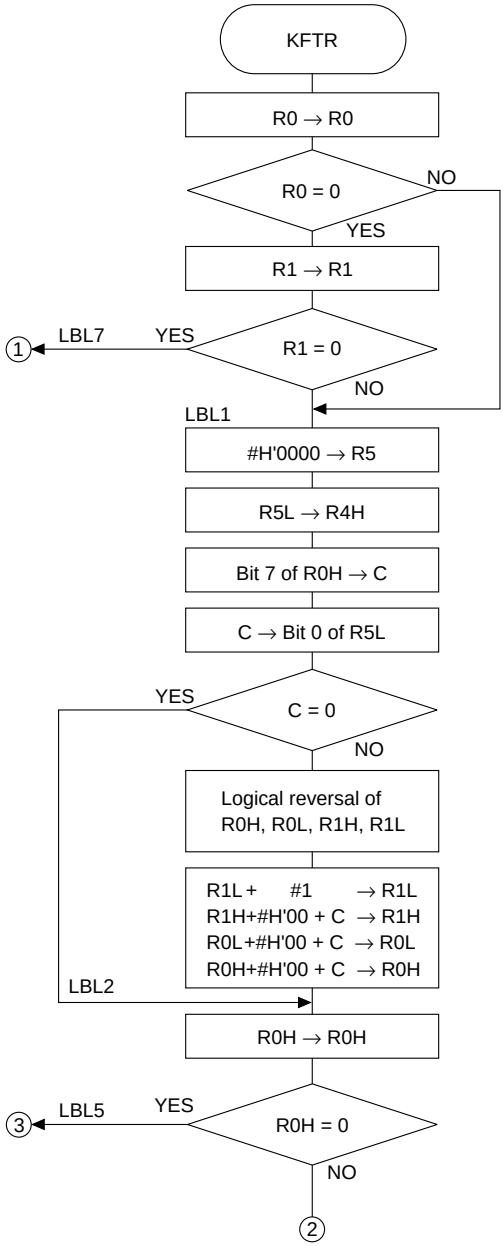
a. The software KFTR first checks whether the signed 32-bit binary number is positive or negative; if it is negative, the software takes two's complement of the number. Next, the software performs either of the following operations depending on whether the upper 8 bits are "H'00" or not;

(i) When the upper 8 bits are not "H'00", the exponent part is calculated and shifted to the right to obtain a 24-bit binary number.

(ii) When the upper 8 bits are "H'00', the exponent part is calculated and shifted to the left to place "1" inn the MSB of the lower 24 bits.

Finally, "H'7F" is added to the exponent part to obtain floating-point form.

7.16.7 Flowchart



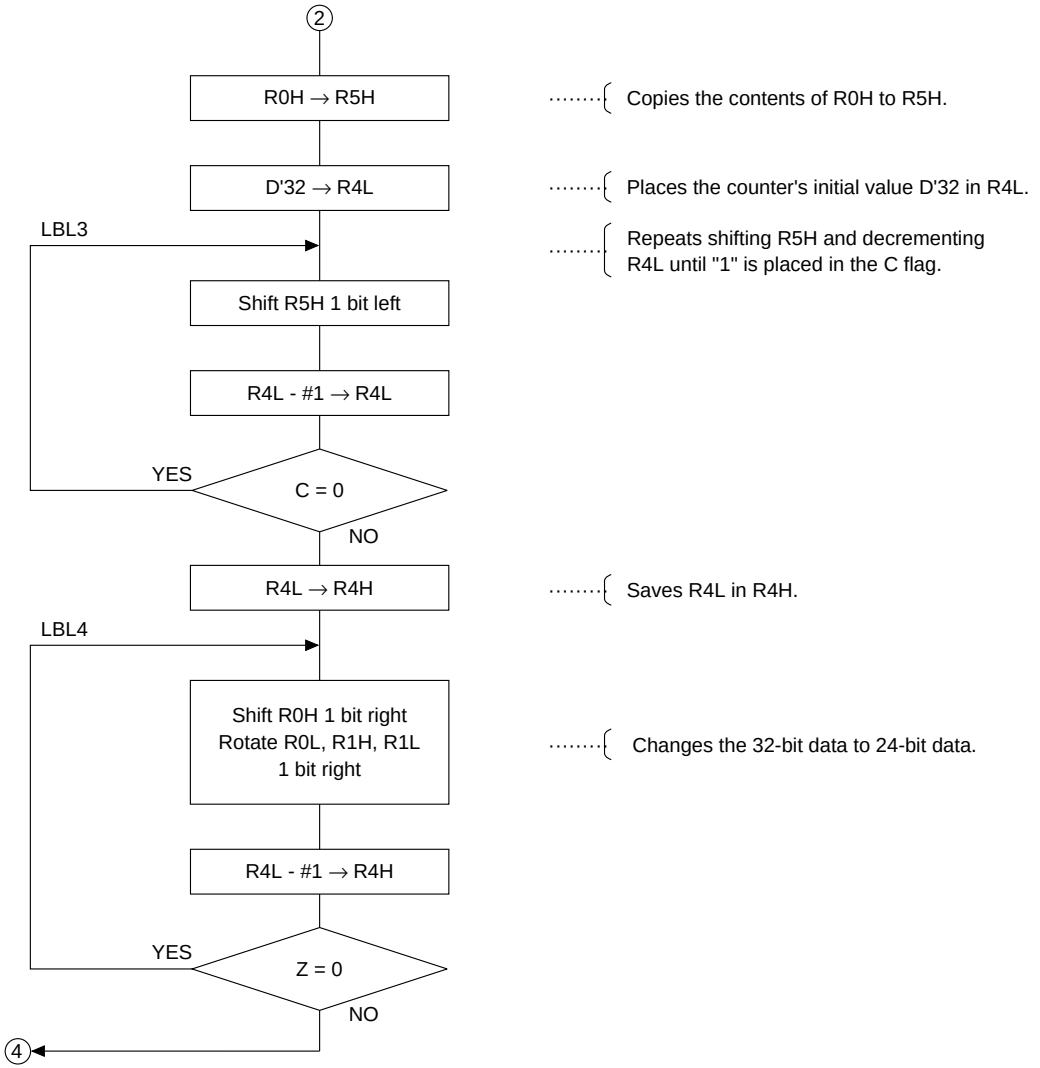
Branches if the signed 32-bit binary number is "0".

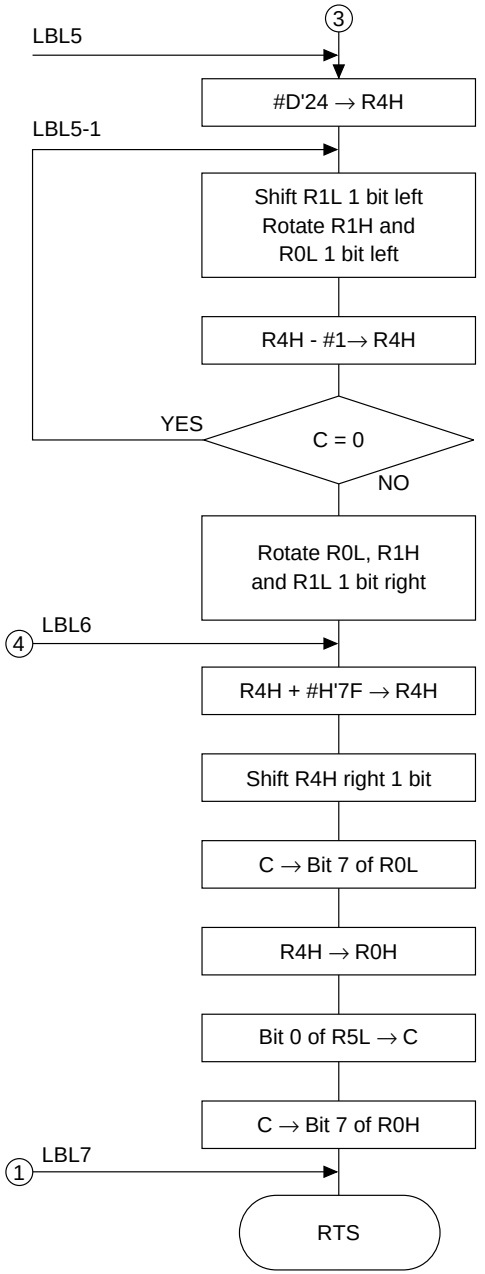
Clears the work registers (R5 and R4H).

Places the sign bit of the signed 32-bit binary number in the LSB of R5L.

Takes two's complement if the signed 32-bit binary number is negative; branches if it is positive.

Branches if R0H is "0".


2
R0H → R5H
D'32 → R4L
LBL3
Shift R5H 1 bit left
R4L - #1 → R4L
YES
C = 0
NO
R4L → R4H
LBL4
Shift R0H 1 bit right
Rotate R0L, R1H, R1L
1 bit right
R4L - #1 → R4H
YES
Z = 0
NO
4
Copies the contents of R0H to R5H.
Places the counter's initial value D'32 in R4L.
Repeats shifting R5H and decrementing
R4L until "1" is placed in the C flag.
Saves R4L in R4H.
Changes the 32-bit data to 24-bit data.



Repeats this operation until "1" is placed in the C flag. Also decrements R4H as many times as the shift count.

Adds "H'7F" to R4H (getabaki representation ??????). Changes the exponent part to short floating-point format.

Places the sign bit in the MSB of R0H.

7.16.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 09:39:38
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME       :CHANGE 32 BIT BINARY TO FLOATING POINT
    4                                   ;*                    (KFTR)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0     (UPPER WORD OF 32 BIT BINARY)
    9                                   ;*                    R1     (LOWER WORD OF 32 BIT BINARY)
   10                                   ;*
   11                                   ;*   RETURNS         :R0 (UPPER WORD OF FLOATING POINT)
   12                                   ;*                    R1 (LOWER WORD OF FLOATING POINT)
   13                                   ;*
   14                                   ;*********************************************************************
   15                                   ;
   16 KFTR_cod C 0000                        .SECTION        KFTR_code,CODE,ALIGN=2
   17                                        .EXPORT         KFTR
   18                                   ;
   19 KFTR_cod C      00000000           KFTR .EQU    $               ;Entry point
   20 KFTR_cod C 0000 0D00                   MOV.W   R0,R0
   21 KFTR_cod C 0002 4604                   BNE     LBL1
   22 KFTR_cod C 0004 0D11                   MOV.W   R1,R1
   23 KFTR_cod C 0006 4758                   BEQ     LBL7            ;Branch if R0=R1=0
   24 KFTR_cod C 0008                   LBL1
   25 KFTR_cod C 0008 79050000               MOV.W   #H'0000,R5      ;Clear R5
   26 KFTR_cod C 000C 0CD4                   MOV.B   R5L,R4H         ;Clear R4H
   27 KFTR_cod C 000E 7770                   BLD     #7,R0H
   28 KFTR_cod C 0010 670D                   BST     #0,R5L          ;Set sign bit to bit 0 of R5L
   29 KFTR_cod C 0012 4410                   BCC     LBL2            ;Branch if 32 bit binary is minus
   30 KFTR_cod C 0014 1700                   NOT     R0H             ;2's complement 32 bit binary
   31 KFTR_cod C 0016 1708                   NOT     R0L
   32 KFTR_cod C 0018 1701                   NOT     R1H
   33 KFTR_cod C 001A 1709                   NOT     R1L
   34 KFTR_cod C 001C 8901                   ADD.B   #H'01,R1L
   35 KFTR_cod C 001E 9100                   ADDX.B  #H'00,R1H
   36 KFTR_cod C 0020 9800                   ADDX.B  #H'00,R0L
   37 KFTR_cod C 0022 9000                   ADDX.B  #H'00,R0H
   38 KFTR_cod C 0024                   LBL2
   39 KFTR_cod C 0024 0C00                   MOV.B   R0H,R0H
   40 KFTR_cod C 0026 471A                   BEQ     LBL5            ;Branch if R0H=0
   41 KFTR_cod C 0028 0C05                   MOV.B   R0H,R5H
   42 KFTR_cod C 002A FC20                   MOV.B   #D'32,R4L       ;Set bit counter1
   43 KFTR_cod C 002C                   LBL3
   44 KFTR_cod C 002C 1005                   SHLL.B  R5H             ;Shift R5H 1 bit left
   45 KFTR_cod C 002E 1A0C                   DEC.B   R4L             ;Decrement R4L
   46 KFTR_cod C 0030 44FA                   BCC     LBL3            ;Branch if C=0
   47 KFTR_cod C 0032 0CC4                   MOV.B   R4L,R4H         ;Push R4L to R4H
   48 KFTR_cod C 0034                   LBL4
   49 KFTR_cod C 0034 1100                   SHLR.B  R0H             ;Change 32 bit binary to mantissa
   50 KFTR_cod C 0036 1308                   ROTXR.B R0L
   51 KFTR_cod C 0038 1301                   ROTXR.B R1H
   52 KFTR_cod C 003A 1309                   ROTXR.B R1L
   53 KFTR_cod C 003C 1A0C                   DEC.B   R4L             ;Decrement bit counter1
   54 KFTR_cod C 003E 46F4                   BNE     LBL4            ;Branch if Z=0
   55 KFTR_cod C 0040 4012                   BRA     LBL6            ;Branch always
   56                                   ;
   57 KFTR_cod C 0042                   LBL5
   58 KFTR_cod C 0042 F418                   MOV.B   #D'24,R4H       ;Set bit counter2
   59 KFTR_cod C 0044                   LBL5_1
   60 KFTR_cod C 0044 1009                   SHLL.B  R1L             ;Change 32 bit binary to mantissa
   61 KFTR_cod C 0046 1201                   ROTXL.B R1H
   62 KFTR_cod C 0048 1208                   ROTXL.B R0L
   63 KFTR_cod C 004A 1A04                   DEC.B   R4H             ;Decrement bit counter2
```

```
 64 KFTR_cod C 004C 44F6                    BCC     LBL5_1
 65 KFTR_cod C 004E 1308                    ROTXR.B R0L             ;Rotate mantissa 1 bit right
 66 KFTR_cod C 0050 1301                    ROTXR.B R1H
 67 KFTR_cod C 0052 1309                    ROTXR.B R1L
 68 KFTR_cod C 0054                 LBL6
 69 KFTR_cod C 0054 847F                    ADD.B   #H'7F,R4H       ;Biased exponent
 70 KFTR_cod C 0056 1104                    SHLR.B  R4H             ;Change floating point format
 71 KFTR_cod C 0058 6778                    BST     #7,R0L
 72 KFTR_cod C 005A 0C40                    MOV.B   R4H,R0H
 73 KFTR_cod C 005C 770D                    BLD     #0,R5L
 74 KFTR_cod C 005E 6770                    BST     #7,R0H
 75 KFTR_cod C 0060                 LBL7
 76 KFTR_cod C 0060 5470                    RTS
 77                                 ;
 78                                         .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

7.17 Addition of Short Floating-Point Numbers

MCU: H8/300 Series
H8/300L Series

Label name: FADD

7.17.1 Function

1. The software FADD adds short floating-point numbers placed in four general-purpose registers and places the result of addition in two of the four general-purpose registers.
2. All arguments used with the software FADD are represented in short floating-point form.

7.17.2 Arguments

Description		Memory area	Data length (bytes)
Input	Augend	R0, R1	4
	Addend	R2, R3	4
Output	Result of addition	R0, R1	4

7.17.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	×	×	×	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.17.4 Specifications

Program memory (bytes)
280
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
268
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.17.5 Notes

The clock cycle count (268) in the specifications is for the example shown in figure 7.44.

For the format of floating-point numbers, see "About Short Floating-point Numbers <Reference>."

7.17.6 Description

1. Details of functions
   a. The following arguments are used with the software FADD:
      (i) Input arguments:
         R0: Contains the upper 2 bytes of a short floating-point augend.
         R1: Contains the lower 2 bytes of the short floating-point augend.
         R2: Contains the upper 2 bytes of a short floating-point addend.
         R3: Contains the lower 2 bytes of the short floating-point addend.
      (ii) Output arguments
         R0: Contains the upper 2 bytes of the result.
         R1: Contains the lower 2 bytes of the result.
   b. Figure 7.44 shows an example of the software FADD being executed. When the input arguments are set as shown in (1), the result of addition is placed in R0 and R1 as shown in (2).

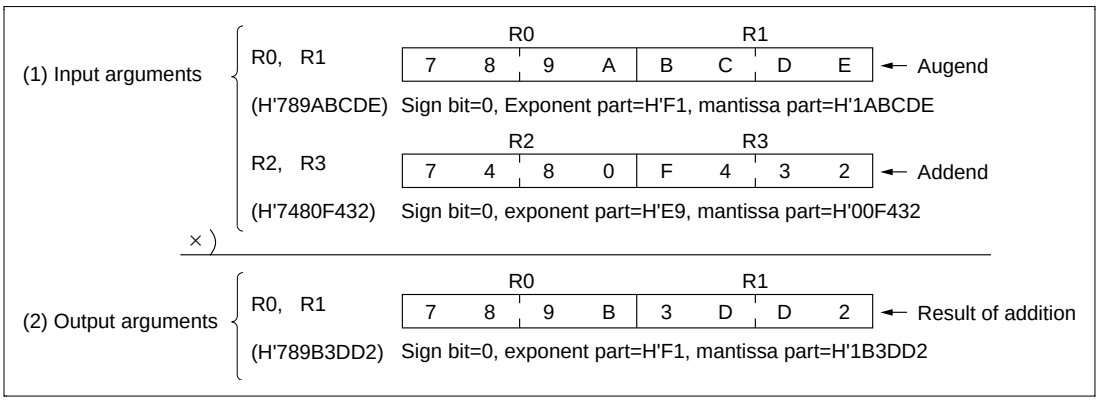


Figure 7.44 Example of Software FADD Execution

2. Notes on usage
   a. The maximum and minimum values that can be handled by the software FADD are as follows:

      Positive maximum H'7F800000
      Positive minimum H'00000001
      Negative maximum H'80000001
      Negative minimum H'FF800000

   b. All positive short floating-point numbers H'7F800000 to H'7FFFFFFF are treated as a maximum value (H'7F800000). All negative short floating-point numbers H'FF800000 to H'FFFFFFFF are treated as a minimum value (H'FF800000).

c. As a maximum value is treated as infinity (∞), the result of ∞ + 100 or ∞ - 100 becomes infinite. (See table 7.5.)

Table 7.5 Examples of Operation with Maximum Values Used as Arguments

Augend	Addend	Result
7F800000 to 7FFFFFFF	********	7F800000
7F800000 to FFFFFFFF	7F800000 to 7FFFFFFF	7F800000
FF800000 to FFFFFFFF	********	FF800000
7F800000 to 7FFFFFFF	FF800000 to FFFFFFFF	FF800000

Note: * represents a hexadecimal number.

d. H'80000000 is treated as H'00000000 (zero).

e. After execution of the software FADD, the augend and addend data are destroyed. If the input arguments are necessary after software FADD execution, save them on memory.

3. Data memory

   The software FADD does not use the data memory.

4. Example of use

   Set an augend and an addend in the general-purpose register and call the software FADD as a subroutine.

```
WORK1   . RES. W   2          ……  Reserves a data memory area
WORK2   . RES. W   2          ……  in which the user program
WORK3   . RES. W   2          ……  places an augend.
                                   an addend.
                                   the result of addition.
        MOV. W   @WORK1,R0    }
        MOV. W   @WORK1+2,R1  }…… Places in R0 and R1 the augend set by the user program.
        MOV. W   @WORK2,R2    }
        MOV. W   @WORK2+2,R3  }…… Places in R2 and R3 the addend set by the user program.
        JSR      @FADD        …… Calls the software FADD as a subroutine.
        MOV. W   R0, @WORK3   }
        MOV. W   R1, @WORK3+2 }…… Places in R0 and R1 the result of addition set in the output argument.
```

5. Operation

Addition of short floating-point numbers is done in the following steps:

a. The software checks whether the augend and addend are +Åá or -Åá .

(i) When the exponent part of the augend is H'FFF, either of the following values is output depending on the state of the sign bit:

Sign bit	Output value
0 (positive)	H'7F800000 (+∞)
1 (negative)	H'FF800000 (–∞)

(ii) The table shown in (a)-(i) also applies when the augend is neither +∞ nor –∞ and the exponent part of the addend is H'FF.

b. The software checks whether the augend and addend are "0".

(i) If either the augend or addend is "0", the other number is output. (If both are "0", "H'00000000" is output.)

c. The software attempts to match the exponent part of the augend with that of the addend.

(i) The smaller number of the exponent part is incremented and, at the same time, the mantissa part (including the implicit MSB) is shifted digit by digit to the right until the exponent part of the augend matches that of the addend. (In the case of the denormalized format, 1 is added to the exponent part aso that the implicit MSB of the mantissa part is treated as "0".

d. The mantissa part of the augend is added to that of the addend.

e. The result of addition is represented in floating-point format.

(Example)

Augend

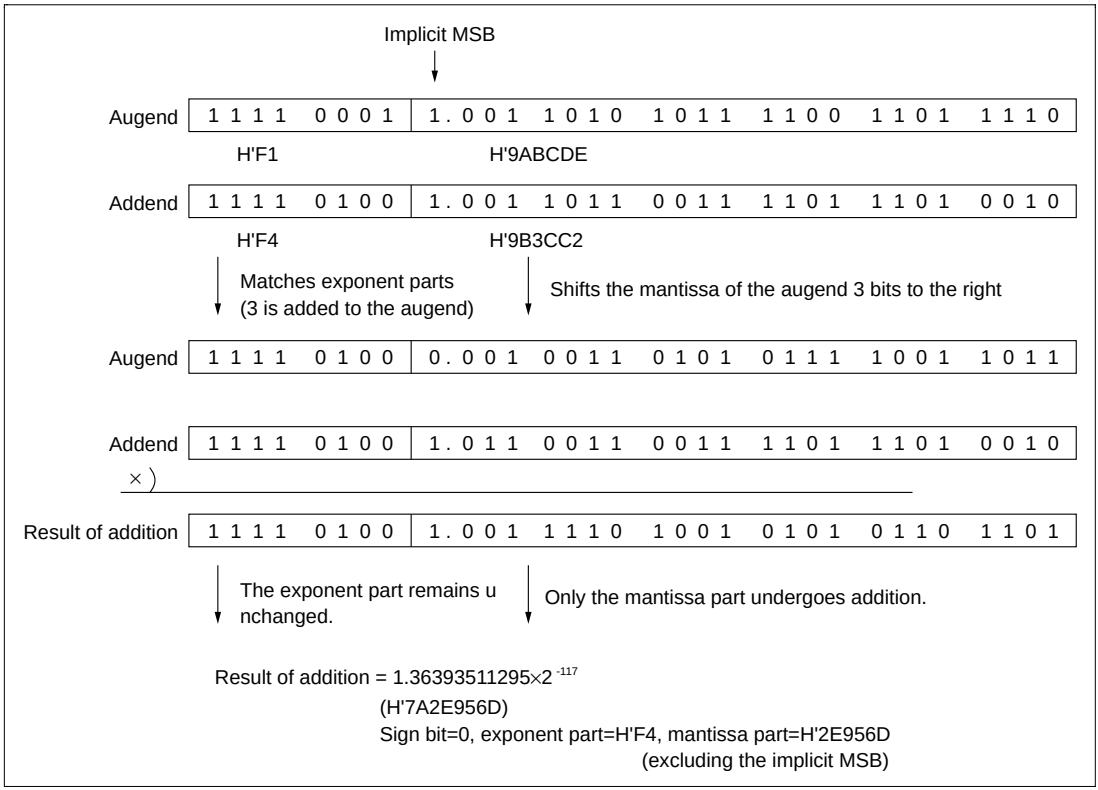
Sign bit=0, exponent part=H'F1, mantissa part=H'1ABCDE

(excluding the implicit MSB)

Addend

Sign bit=0, exponent part=H'F4, mantissa part=H'1B3DD2

(excluding the implicit MSB)



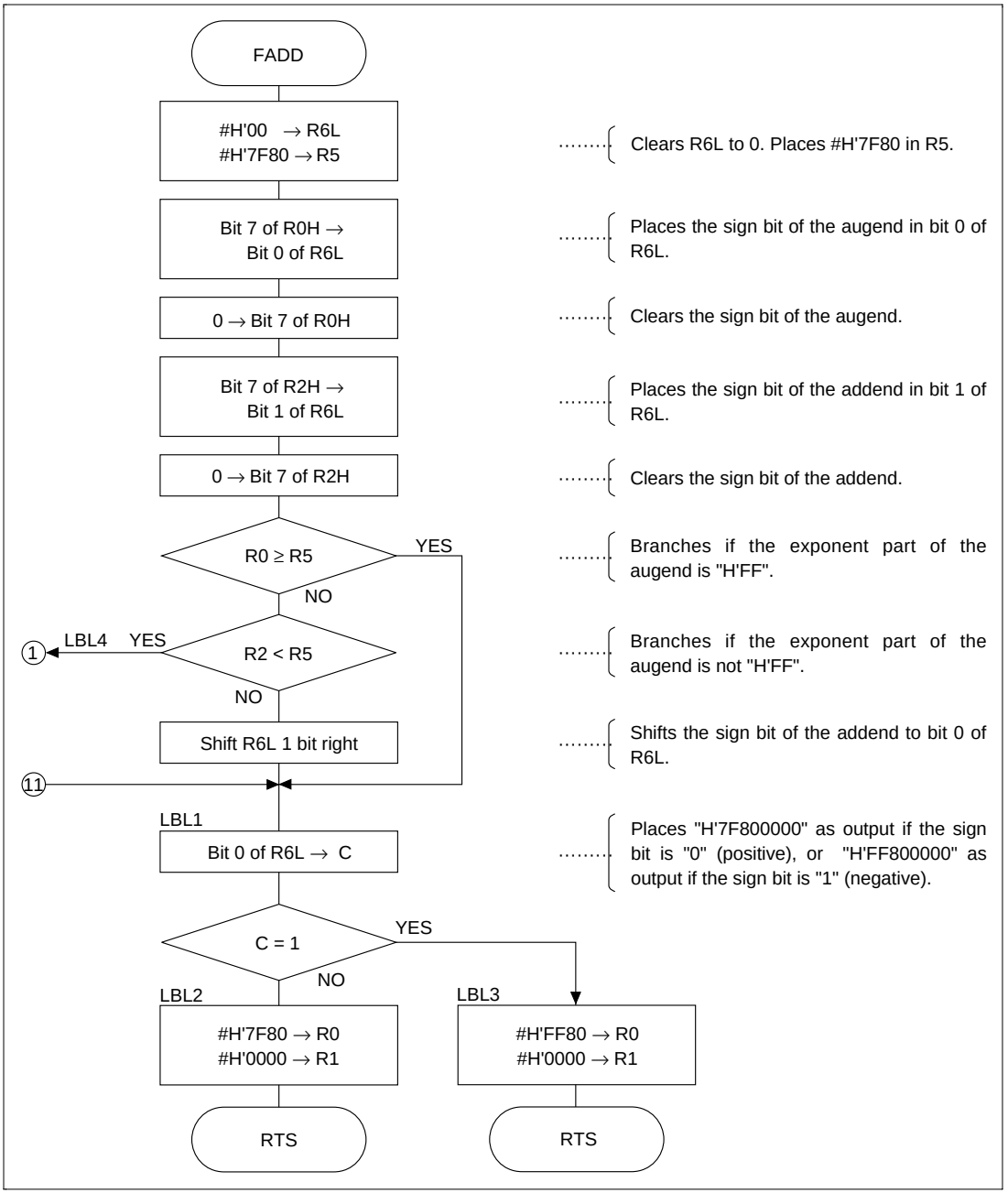
Result of addition = $1.36393511295 \times 2^{-117}$

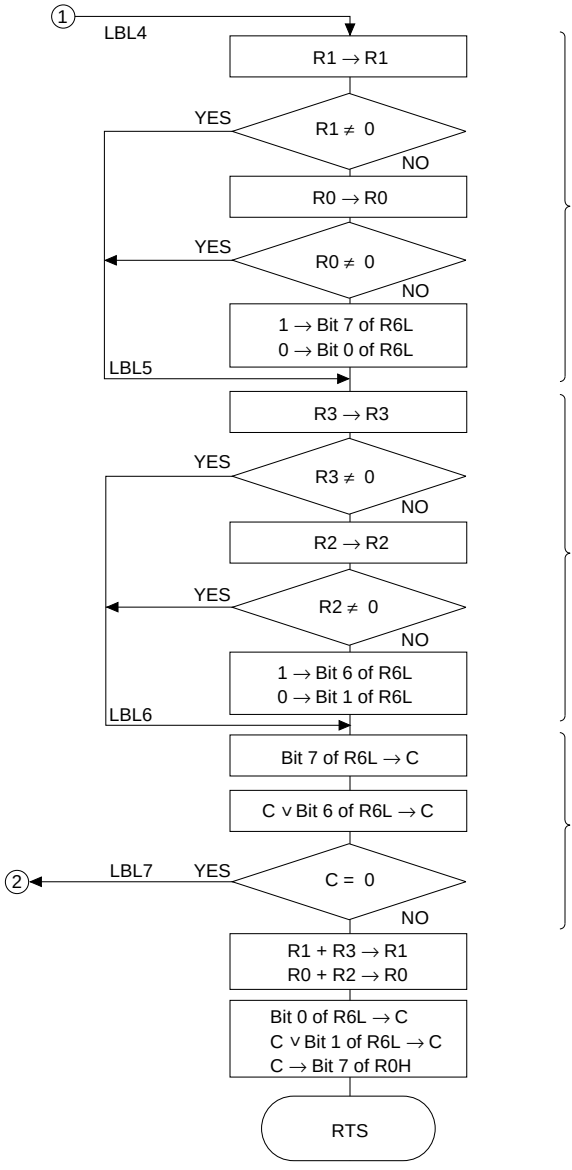
(H'7A2E956D)

Sign bit=0, exponent part=H'F4, mantissa part=H'2E956D

(excluding the implicit MSB)

7.17.7 Flowchart



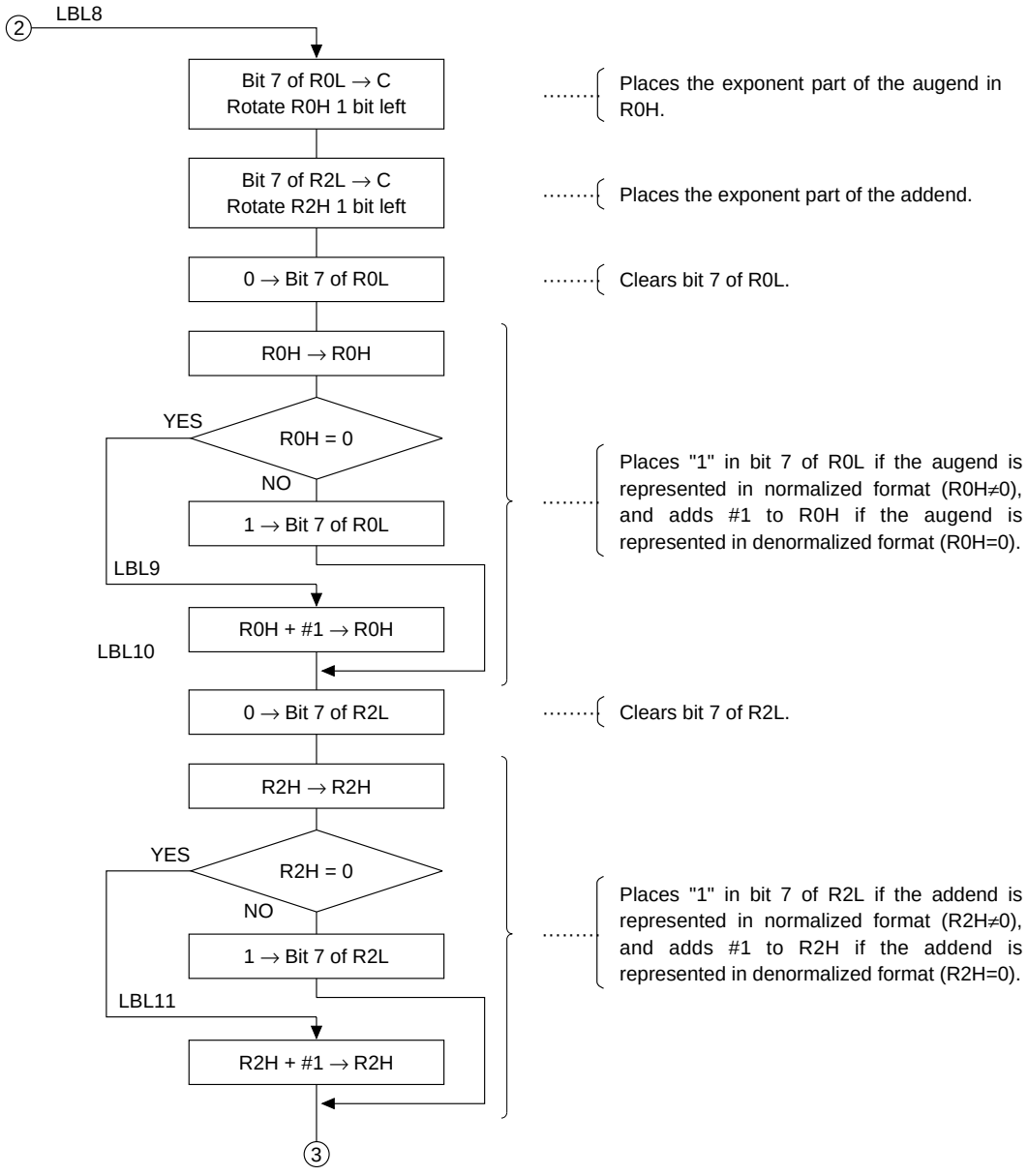


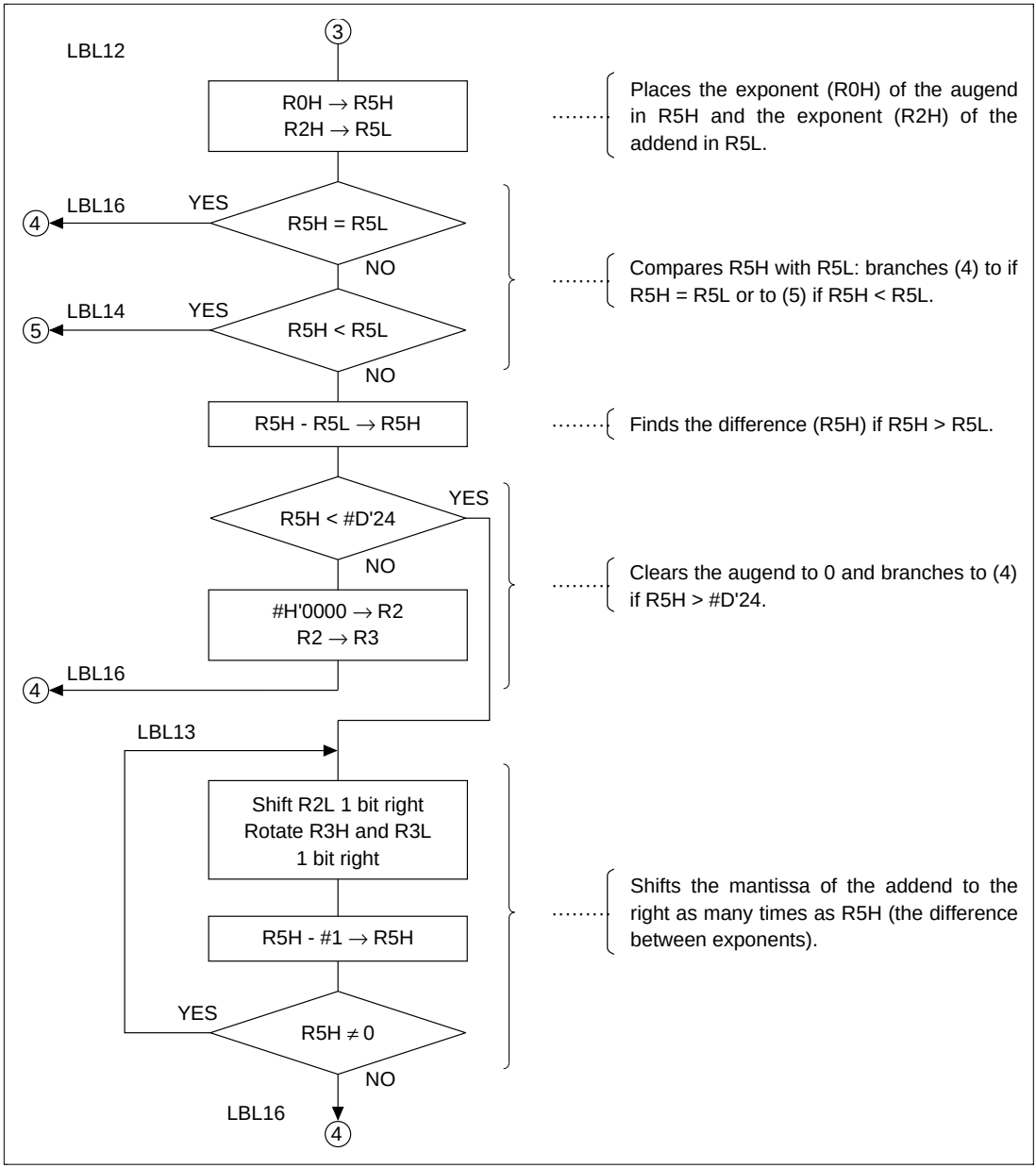
Places "1" in bit 7 of R6L if the augend is "0".

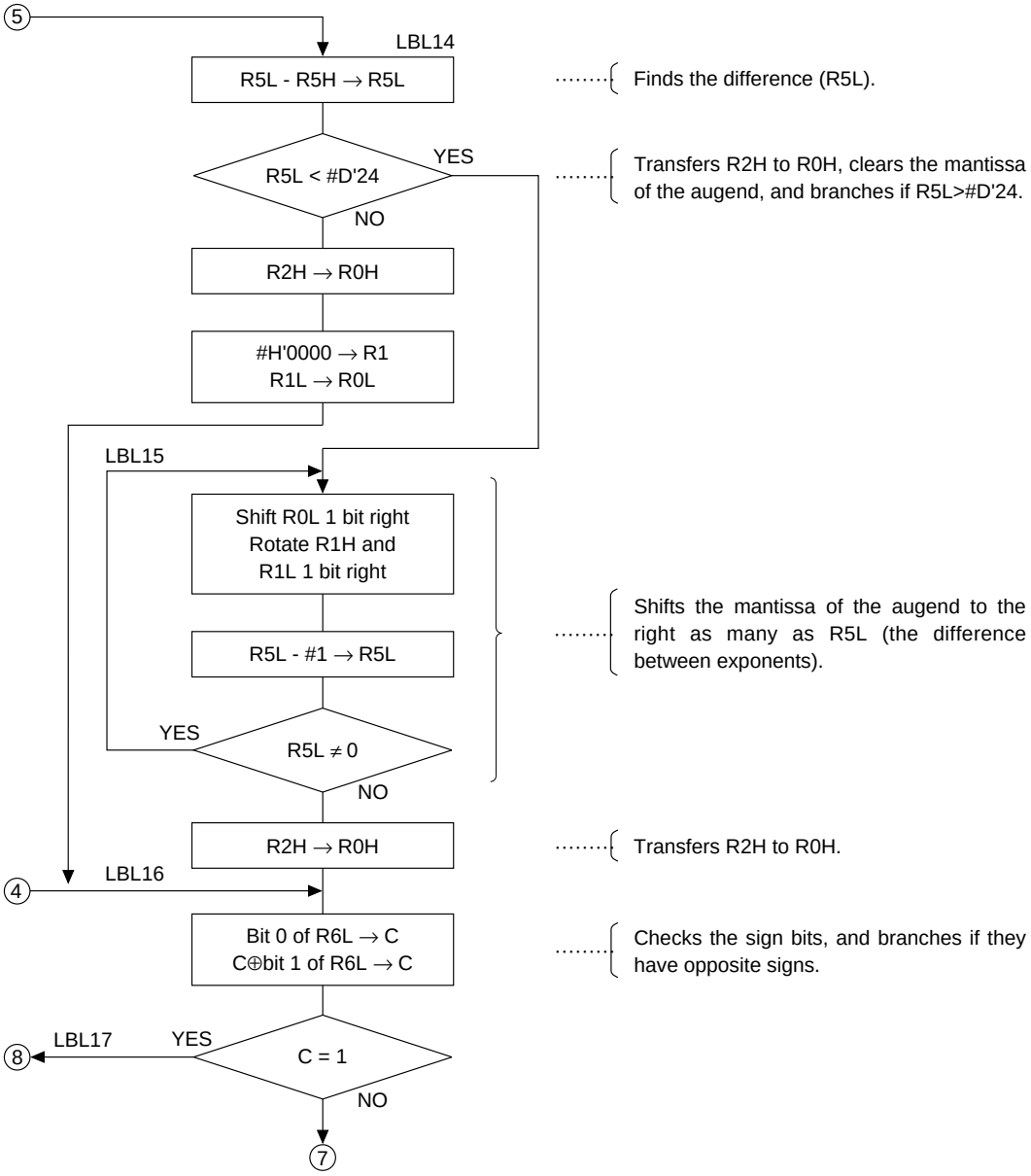
Places "1" in bit 6 of R6L if the addend is "0".

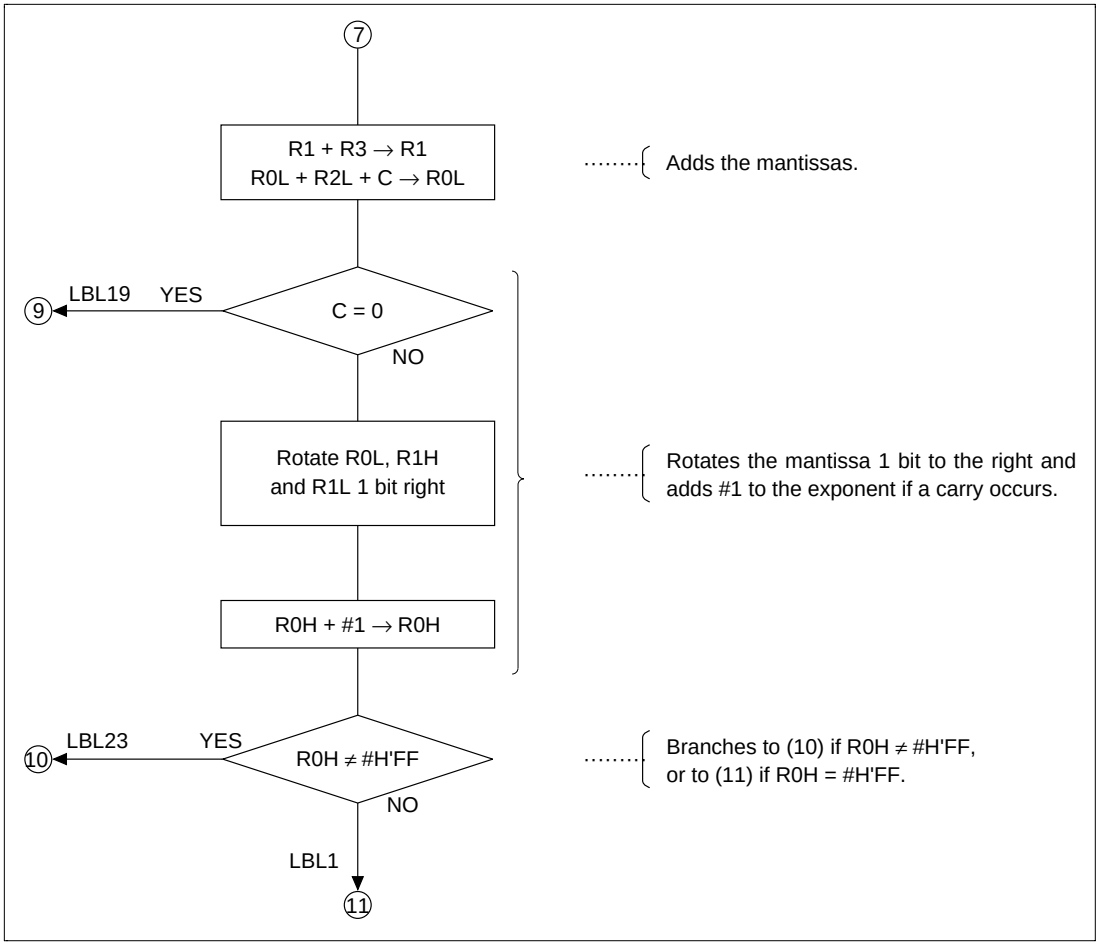
Bit 7 of R6L is ANDed with bit 6 of R6L. Branches if C= "0" (augend ≠ "0"addend ≠ "0").

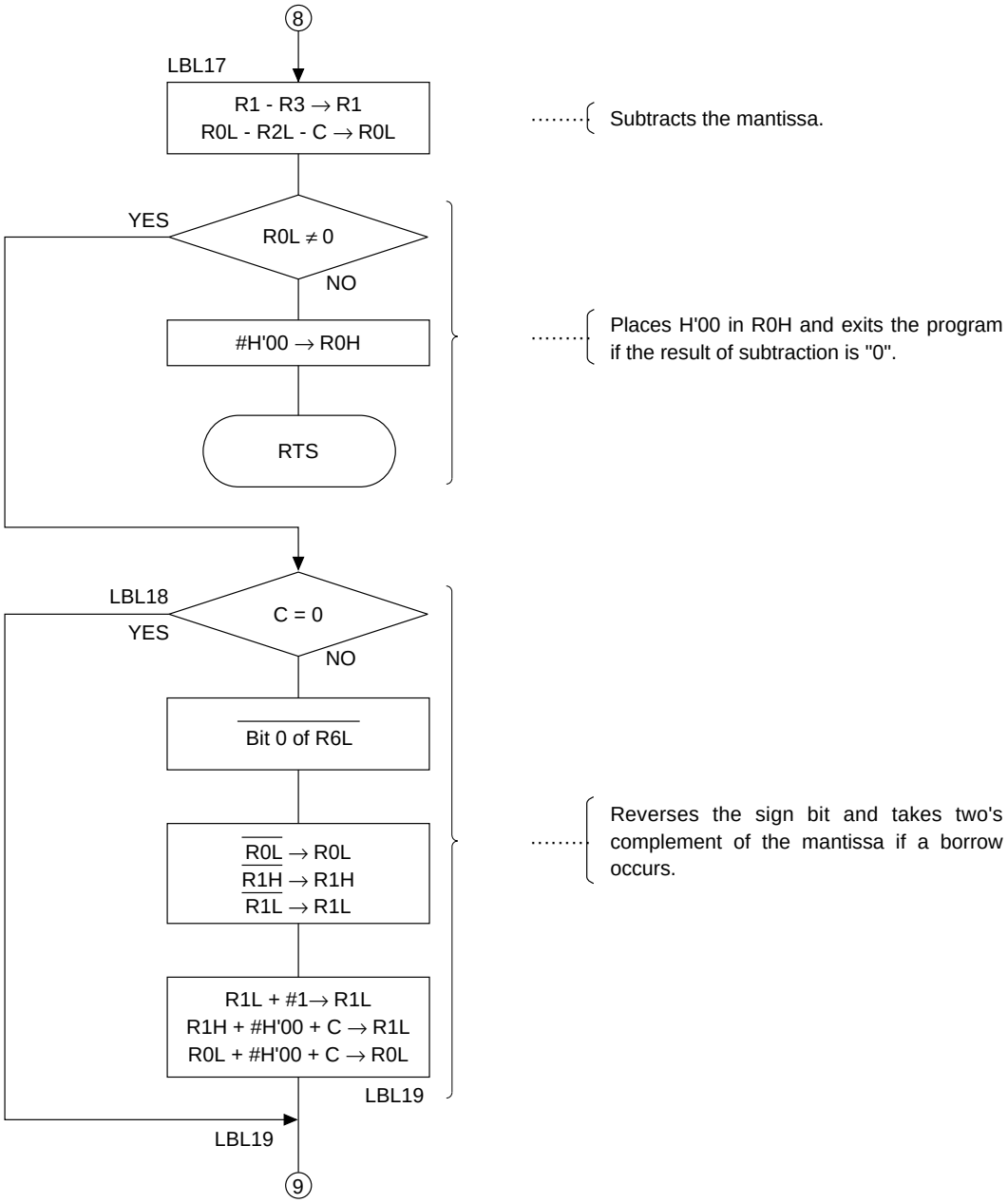
Places in R0 and R1 the augend or addend as an output value..

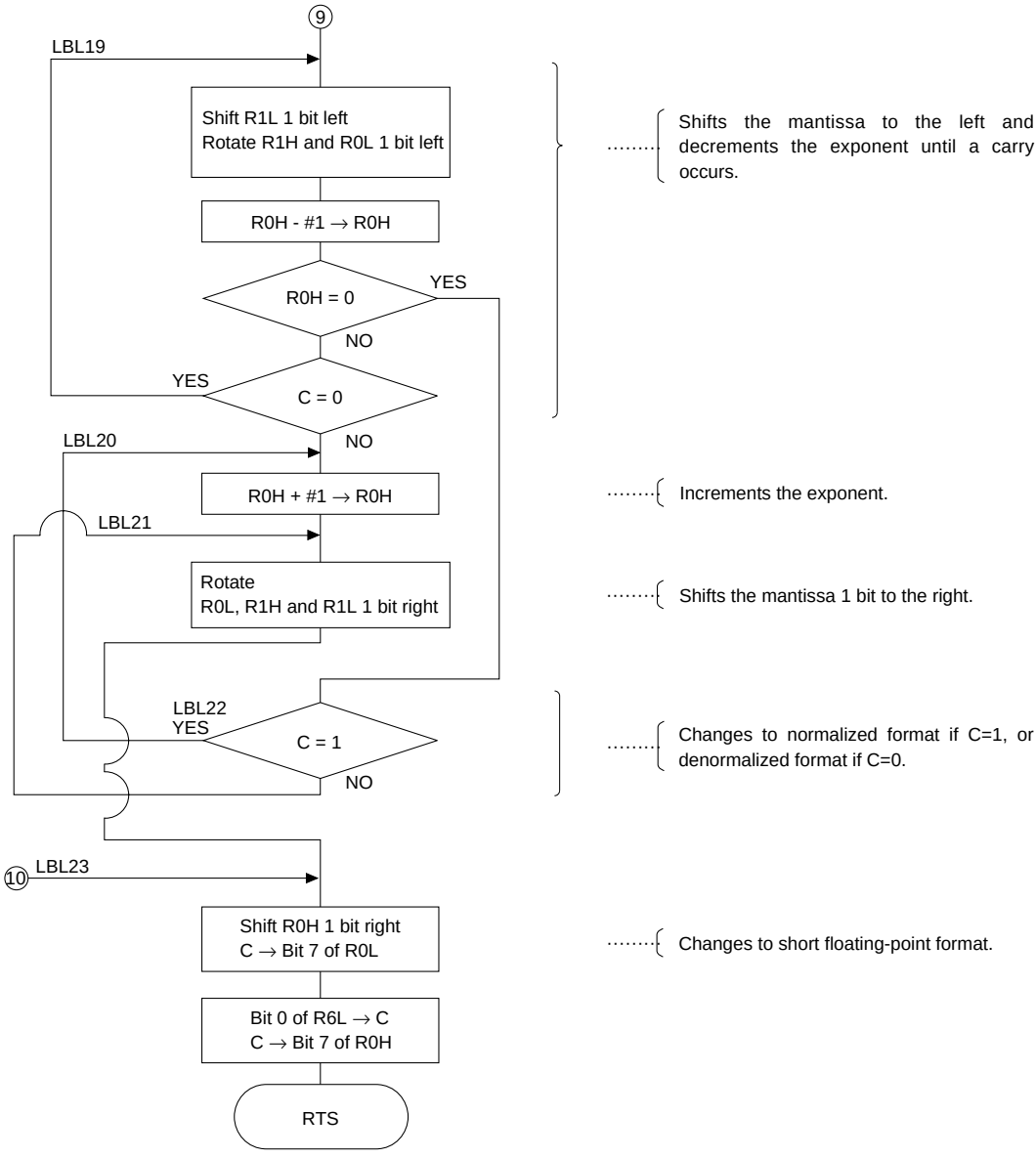

②
LBL8
Bit 7 of R0L → C
Rotate R0H 1 bit left
Bit 7 of R2L → C
Rotate R2H 1 bit left
0 → Bit 7 of R0L
R0H → R0H
YES
R0H = 0
NO
1 → Bit 7 of R0L
LBL9
R0H + #1 → R0H
LBL10
0 → Bit 7 of R2L
R2H → R2H
YES
R2H = 0
NO
1 → Bit 7 of R2L
LBL11
R2H + #1 → R2H
③
Places the exponent part of the augend in R0H.
Places the exponent part of the addend.
Clears bit 7 of R0L.
Places "1" in bit 7 of R0L if the augend is represented in normalized format (R0H≠0), and adds #1 to R0H if the augend is represented in denormalized format (R0H=0).
Clears bit 7 of R2L.
Places "1" in bit 7 of R2L if the addend is represented in normalized format (R2H≠0), and adds #1 to R2H if the addend is represented in denormalized format (R2H=0).


LBL12
3
R0H → R5H
R2H → R5L
Places the exponent (R0H) of the augend in R5H and the exponent (R2H) of the addend in R5L.
LBL16
YES
4
R5H = R5L
NO
LBL14
YES
5
R5H < R5L
NO
Compares R5H with R5L: branches (4) to if R5H = R5L or to (5) if R5H < R5L.
R5H - R5L → R5H
Finds the difference (R5H) if R5H > R5L.
YES
R5H < #D'24
NO
#H'0000 → R2
R2 → R3
LBL16
4
Clears the augend to 0 and branches to (4) if R5H > #D'24.
LBL13
Shift R2L 1 bit right
Rotate R3H and R3L
1 bit right
R5H - #1 → R5H
YES
R5H ≠ 0
NO
LBL16
4
Shifts the mantissa of the addend to the right as many times as R5H (the difference between exponents).


5
LBL14
R5L - R5H → R5L
Finds the difference (R5L).
R5L < #D'24
YES
NO
Transfers R2H to R0H, clears the mantissa of the augend, and branches if R5L>#D'24.
R2H → R0H
#H'0000 → R1
R1L → R0L
LBL15
Shift R0L 1 bit right
Rotate R1H and
R1L 1 bit right
R5L - #1 → R5L
R5L ≠ 0
YES
NO
Shifts the mantissa of the augend to the right as many as R5L (the difference between exponents).
R2H → R0H
Transfers R2H to R0H.
4
LBL16
Bit 0 of R6L → C
C⊕bit 1 of R6L → C
Checks the sign bits, and branches if they have opposite signs.
8
LBL17
YES
C = 1
NO
7


7
R1 + R3 → R1
R0L + R2L + C → R0L
Adds the mantissas.
9
LBL19
YES
C = 0
NO
Rotate R0L, R1H
and R1L 1 bit right
Rotates the mantissa 1 bit to the right and
adds #1 to the exponent if a carry occurs.
R0H + #1 → R0H
10
LBL23
YES
R0H ≠ #H'FF
Branches to (10) if R0H ≠ #H'FF,
or to (11) if R0H = #H'FF.
NO
LBL1
11


8
LBL17
R1 - R3 → R1
R0L - R2L - C → R0L
Subtracts the mantissa.
YES
R0L ≠ 0
NO
#H'00 → R0H
Places H'00 in R0H and exits the program if the result of subtraction is "0".
RTS
LBL18
C = 0
YES
NO
Bit 0 of R6L
R0L → R0L
R1H → R1H
R1L → R1L
Reverses the sign bit and takes two's complement of the mantissa if a borrow occurs.
R1L + #1→ R1L
R1H + #H'00 + C → R1L
R0L + #H'00 + C → R0L
LBL19
LBL19
9


9
LBL19
Shift R1L 1 bit left
Rotate R1H and R0L 1 bit left
R0H - #1 → R0H
R0H = 0
YES
NO
YES
C = 0
NO
LBL20
R0H + #1 → R0H
LBL21
Rotate
R0L, R1H and R1L 1 bit right
LBL22
YES
C = 1
NO
10
LBL23
Shift R0H 1 bit right
C → Bit 7 of R0L
Bit 0 of R6L → C
C → Bit 7 of R0H
RTS
Shifts the mantissa to the left and decrements the exponent until a carry occurs.
Increments the exponent.
Shifts the mantissa 1 bit to the right.
Changes to normalized format if C=1, or denormalized format if C=0.
Changes to short floating-point format.

7.17.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:20:43
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME       :FLOATING POINT ADDITION (FADD)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0     (UPPER WORD OF SUMMAND)
    8                                   ;*                    R1     (LOWER WORD OF SUMMAND)
    9                                   ;*                    R2     (UPPER WORD OF ADDEND)
   10                                   ;*                    R3     (LOWER WORD OF ADDEND)
   11                                   ;*
   12                                   ;*   RETURNS         :R0     (UPPER WORD OF RESULT)
   13                                   ;*                    R1     (LOWER WORD OF RESULT)
   14                                   ;*
   15                                   ;*********************************************************************
   16                                   ;
   17 FADD_cod C 0000                        .SECTION        FADD_code,CODE,ALIGN=2
   18                                        .EXPORT         FADD
   19                                   ;
   20 FADD_cod C      00000000       FADD .EQU    $               ;Entry point
   21 FADD_cod C 0000 FE00                   MOV.B   #H'00,R6L       ;Clear R6L
   22 FADD_cod C 0002 79057F80               MOV.W   #H'7F80,R5      ;Set "H'7F80"
   23                                   ;
   24 FADD_cod C 0006 7770                   BLD     #7,R0H
   25 FADD_cod C 0008 670E                   BST     #0,R6L          ;Set sign bit to bit 0 of R6L
   26 FADD_cod C 000A 7270                   BCLR    #7,R0H          ;Bit clear bit 7 of R0H
   27                                   ;
   28 FADD_cod C 000C 7772                   BLD     #7,R2H
   29 FADD_cod C 000E 671E                   BST     #1,R6L          ;Set sign bit to bit 1 of R6L
   30 FADD_cod C 0010 7272                   BCLR    #7,R2H          ;Bit clear bit 7 of R2H
   31                                   ;
   32 FADD_cod C 0012 1D05                   CMP.W   R0,R5
   33 FADD_cod C 0014 4306                   BLS     LBL1            ;Branch if "exponent of summand"="H'FF"
   34 FADD_cod C 0016 1D25                   CMP.W   R2,R5
   35 FADD_cod C 0018 421A                   BHI     LBL4            ;Branch if not "exponent of summand"="H'FF"
   36 FADD_cod C 001A 110E                   SHLR    R6L             ;Shift R6L 1 bit right
   37 FADD_cod C 001C                   LBL1
   38 FADD_cod C 001C 770E                   BLD     #0,R6L          ;Bit load sign bit
   39 FADD_cod C 001E 450A                   BCS     LBL3            ;Branch if sign bit=1
   40 FADD_cod C 0020                   LBL2
   41 FADD_cod C 0020 79007F80               MOV.W   #H'7F80,R0      ;Set plus maximum number
   42 FADD_cod C 0024 79010000               MOV.W   #H'0000,R1
   43 FADD_cod C 0028 5470                   RTS
   44 FADD_cod C 002A                   LBL3
   45 FADD_cod C 002A 7900FF80               MOV.W   #H'FF80,R0      ;Set minus minimum number
   46 FADD_cod C 002E 79010000               MOV.W   #H'0000,R1
   47 FADD_cod C 0032 5470                   RTS
   48                                   ;
   49 FADD_cod C 0034                   LBL4
   50 FADD_cod C 0034 0D11                   MOV.W   R1,R1           ;
   51 FADD_cod C 0036 4608                   BNE     LBL5            ;Branch if Z=0
   52 FADD_cod C 0038 0D00                   MOV.W   R0,R0
   53 FADD_cod C 003A 4604                   BNE     LBL5            ;Branch if Z=0
   54 FADD_cod C 003C 707E                   BSET    #7,R6L          ;Bit set bit 7 of R6L
   55 FADD_cod C 003E 720E                   BCLR    #0,R6L          ;Bit clear bit 0 of R6L
   56 FADD_cod C 0040                   LBL5
   57 FADD_cod C 0040 0D33                   MOV.W   R3,R3
   58 FADD_cod C 0042 4608                   BNE     LBL6            ;Branch if Z=0
   59 FADD_cod C 0044 0D22                   MOV.W   R2,R2
   60 FADD_cod C 0046 4604                   BNE     LBL6            ;Branch if Z=0
   61 FADD_cod C 0048 706E                   BSET    #6,R6L          ;Bit set bit 6 of R6L
   62 FADD_cod C 004A 721E                   BCLR    #1,R6L          ;Bit clear bit 1 of R6L
   63 FADD_cod C 004C                   LBL6
```

```
 64 FADD_cod C 004C 777E                    BLD     #7,R6L
 65 FADD_cod C 004E 746E                    BOR     #6,R6L
 66 FADD_cod C 0050 440C                    BCC     LBL8            ;Branch if not summand=addend=0
 67 FADD_cod C 0052 0931                    ADD.W   R3,R1           ;Set summand and addend to result
 68 FADD_cod C 0054 0920                    ADD.W   R2,R0
 69 FADD_cod C 0056 770E                    BLD     #0,R6L
 70 FADD_cod C 0058 741E                    BOR     #1,R6L
 71 FADD_cod C 005A 6770                    BST     #7,R0H          ;Set sign bit
 72 FADD_cod C 005C 5470                    RTS
 73                                 ;
 74 FADD_cod C 005E                 LBL8
 75 FADD_cod C 005E 7778                    BLD     #7,R0L
 76 FADD_cod C 0060 1200                    ROTXL   R0H             ;Set exponent of summand to R0H
 77                                 ;
 78 FADD_cod C 0062 777A                    BLD     #7,R2L
 79 FADD_cod C 0064 1202                    ROTXL   R2H             ;Set exponent of addend to R0L
 80                                 ;
 81 FADD_cod C 0066 7278                    BCLR    #7,R0L
 82 FADD_cod C 0068 0C00                    MOV.B   R0H,R0H
 83 FADD_cod C 006A 4704                    BEQ     LBL9            ;Branch if summand is normalized
 84 FADD_cod C 006C 7078                    BSET    #7,R0L          ;Set implicit MSB to summand
 85 FADD_cod C 006E 4002                    BRA     LBL10           ;Branch always
 86 FADD_cod C 0070                 LBL9
 87 FADD_cod C 0070 8001                    ADD.B   #H'01,R0H
 88 FADD_cod C 0072                 LBL10
 89 FADD_cod C 0072 727A                    BCLR    #7,R2L
 90 FADD_cod C 0074 0C22                    MOV.B   R2H,R2H
 91 FADD_cod C 0076 4704                    BEQ     LBL11           ;Branch if addend is normalized
 92 FADD_cod C 0078 707A                    BSET    #7,R2L          ;Set implicit MSB to addend
 93 FADD_cod C 007A 4002                    BRA     LBL12           ;Branch always
 94 FADD_cod C 007C                 LBL11
 95 FADD_cod C 007C 8201                    ADD.B   #H'01,R2H
 96                                 ;
 97 FADD_cod C 007E                 LBL12
 98 FADD_cod C 007E 0C05                    MOV.B   R0H,R5H
 99 FADD_cod C 0080 0C2D                    MOV.B   R2H,R5L
100 FADD_cod C 0082 1CD5                    CMP.B   R5L,R5H
101 FADD_cod C 0084 4738                    BEQ     LBL16           ;Branch if R5H=R5L
102 FADD_cod C 0086 451A                    BCS     LBL14           ;Branch if R5H<R5L
103                                ;
104 FADD_cod C 0088 18D5                    SUB.B   R5L,R5H
105 FADD_cod C 008A A518                    CMP.B   #D'24,R5H       ;Set bit counter
106 FADD_cod C 008C 4508                    BCS     LBL13           ;Branch if R5H<D'24
107 FADD_cod C 008E 79020000                MOV.W   #H'0000,R2      ;Clear addend
108 FADD_cod C 0092 0D23                    MOV.W   R2,R3
109 FADD_cod C 0094 4028                    BRA     LBL16           ;Branch always
110 FADD_cod C 0096                LBL13
111 FADD_cod C 0096 110A                    SHLR    R2L             ;Shift mantissa of addend 1 bit left
112 FADD_cod C 0098 1303                    ROTXR   R3H
113 FADD_cod C 009A 130B                    ROTXR   R3L
114 FADD_cod C 009C 1A05                    DEC.B   R5H             ;Decrement bit counter
115 FADD_cod C 009E 46F6                    BNE     LBL13           ;Branch Z=0
116 FADD_cod C 00A0 401C                    BRA     LBL16           ;Branch always
117                                ;
118 FADD_cod C 00A2                LBL14
119 FADD_cod C 00A2 185D                    SUB.B   R5H,R5L
120 FADD_cod C 00A4 AD18                    CMP.B   #D'24,R5L
121 FADD_cod C 00A6 450A                    BCS     LBL15           ;Branch if R5L<D'24
122 FADD_cod C 00A8 0C20                    MOV.B   R2H,R0H
123 FADD_cod C 00AA 79010000                MOV.W   #H'0000,R1      ;Clear summand
124 FADD_cod C 00AE 0C98                    MOV.B   R1L,R0L
125 FADD_cod C 00B0 400C                    BRA     LBL16           ;Branch always
126 FADD_cod C 00B2                LBL15
127 FADD_cod C 00B2 1108                    SHLR    R0L             ;Shift mantissa of summand 1 bit right
128 FADD_cod C 00B4 1301                    ROTXR   R1H
129 FADD_cod C 00B6 1309                    ROTXR   R1L
130 FADD_cod C 00B8 1A0D                    DEC.B   R5L             ;Decrement bit counter
131 FADD_cod C 00BA 46F6                    BNE     LBL15           ;Branch if Z=0
132 FADD_cod C 00BC 0C20                    MOV.B   R2H,R0H
133                                ;
```

```
134 FADD_cod C 00BE                    LBL16
135 FADD_cod C 00BE 770E                    BLD     #0,R6L
136 FADD_cod C 00C0 751E                    BXOR    #1,R6L
137 FADD_cod C 00C2 4516                    BCS     LBL17           ;Branch if different sign bit
138                                    ;
139 FADD_cod C 00C4 0931                    ADD.W   R3,R1           ;Addition mantissa
140 FADD_cod C 00C6 0EA8                    ADDX.B  R2L,R0L
141 FADD_cod C 00C8 442A                    BCC     LBL19           ;Branch if C=0
142 FADD_cod C 00CA 1308                    ROTXR   R0L             ;Rotate mantissa 1 bit right
143 FADD_cod C 00CC 1301                    ROTXR   R1H
144 FADD_cod C 00CE 1309                    ROTXR   R1L
145 FADD_cod C 00D0 8001                    ADD.B   #H'01,R0H       ;Increment exponent
146 FADD_cod C 00D2 A0FF                    CMP.B   #H'FF,R0H
147 FADD_cod C 00D4 4638                    BNE     LBL23           ;Branch if not exponent=H'FF
148 FADD_cod C 00D6 5A000000                JMP     @LBL1           ;Jump
149                                    ;
150 FADD_cod C 00DA                    LBL17
151 FADD_cod C 00DA 1931                    SUB.W   R3,R1           ;Substruct mantissa
152 FADD_cod C 00DC 1EA8                    SUBX.B  R2L,R0L
153 FADD_cod C 00DE 4604                    BNE     LBL18           ;Branch if Z=0
154 FADD_cod C 00E0 F000                    MOV.B   #H'00,R0H       ;Clear R0H
155 FADD_cod C 00E2 5470                    RTS
156 FADD_cod C 00E4                    LBL18
157 FADD_cod C 00E4 440E                    BCC     LBL19           ;Branch if C=0
158 FADD_cod C 00E6 710E                    BNOT    #0,R6L          ;Bit not sign bit
159 FADD_cod C 00E8 1708                    NOT     R0L             ;2's complement mantissa
160 FADD_cod C 00EA 1701                    NOT     R1H
161 FADD_cod C 00EC 1709                    NOT     R1L
162 FADD_cod C 00EE 8901                    ADD.B   #H'01,R1L
163 FADD_cod C 00F0 9100                    ADDX.B  #H'00,R1H
164 FADD_cod C 00F2 9800                    ADDX.B  #H'00,R0L
165                                    ;
166 FADD_cod C 00F4                    LBL19
167 FADD_cod C 00F4 1009                    SHLL    R1L             ;Shift mantissa 1 bit left
168 FADD_cod C 00F6 1201                    ROTXL   R1H
169 FADD_cod C 00F8 1208                    ROTXL   R0L
170 FADD_cod C 00FA 1A00                    DEC.B   R0H             ;Decrement exponemt
171 FADD_cod C 00FC 470C                    BEQ     LBL22           ;Branch if exponent=0
172 FADD_cod C 00FE 44F4                    BCC     LBL19           ;Branch if exponent>0
173 FADD_cod C 0100                    LBL20
174 FADD_cod C 0100 0A00                    INC.B   R0H             ;Increment exponent
175 FADD_cod C 0102                    LBL21
176 FADD_cod C 0102 1308                    ROTXR   R0L             ;Rotate mantissa 1 bit right
177 FADD_cod C 0104 1301                    ROTXR   R1H
178 FADD_cod C 0106 1309                    ROTXR   R1L
179 FADD_cod C 0108 4004                    BRA     LBL23           ;Branch always
180 FADD_cod C 010A                    LBL22
181 FADD_cod C 010A 45F4                    BCS     LBL20           ;Branch if C=1
182 FADD_cod C 010C 40F4                    BRA     LBL21           ;Branch always
183                                    ;
184 FADD_cod C 010E                    LBL23                                   ;Chage floating point format
185 FADD_cod C 010E 1100                    SHLR    R0H
186 FADD_cod C 0110 6778                    BST     #7,R0L
187 FADD_cod C 0112 770E                    BLD     #0,R6L
188 FADD_cod C 0114 6770                    BST     #7,R0H
189 FADD_cod C 0116 5470                    RTS
190                                    ;
191                                         .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

7.18 Multiplication of Short Floating-Point Numbers

MCU: H8/300 Series
H8/300L Series

Label name: FMUL

7.18 Function

1. The software FMUL performs multiplication of short floating-point numbers placed in four general-purpose registers and places the result of multiplication in two of the four general-purpose registers.
2. All arguments used with the software FMUL are represented in short floating-point form.

7.18.1 Arguments

Description		Memory area	Data length (bytes)
Input	Multiplicand	R0, R1	4
	Multiplier	R2, R3	4
		R0, R1	4
Output	Result of multiplication		

7.18.2 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
↕	↕	×	×	×	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.18.3 Specifications

Program memory (bytes)
348
Data memory (bytes)
0
Stack (bytes)
16
Clock cycle count
1078
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.18.4 Notes

The clock cycle count (16) in the specifications is for the example shown in figure 7.45.

For the format of floating-point numbers, see "About Short Floating-point Numbers <Reference>."

7.18.5 Description

1. Details of functions
   a. The following arguments are used with the software FMUL:
      (i) Input arguments:
      R0: Contains the upper 2 bytes of a short floating-point multiplicand.
      R1: Contains the lower 2 bytes of the short floating-point multiplicand.
      R2: Contains the upper 2 bytes of a short floating-point multiplier.
      R3: Contains the lower 2 bytes of the short floating-point multiplier.

(ii) Output arguments

R0: Contains the upper 2 bytes of the result.

R1: Contains the lower 2 bytes of the result.

b. Figure 7.45 shows an example of the software FMUL being executed. When the input arguments are set as shown in (a), the result of multiplication is placed in R0 and R1 as shown in (b).

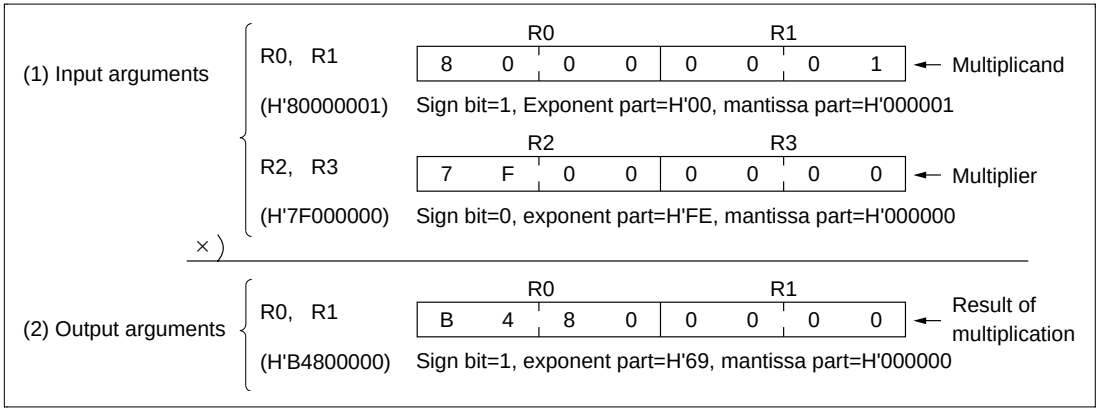


Figure 7.45 Example of Software FMUL Execution

2. Notes on usage

a. The maximum and minimum values that can be handled by the software FADD are as follows:

Positive maximum H'7F800000
Positive minimum H'00000001
Negative maximum H'80000001
Negative minimum H'FF800000

b. All positive short floating-point numbers H'7F800000 to H'7FFFFFFF are treated as a maximum value (H'7F800000). All negative short floating-point numbers H'FF800000 to H'FFFFFFFF are treated as a minimum value (H'FfF800000).

c. As a maximum value is treated as infinity (∞),∞ x 100 = ∞ or ∞ x (-100) = –∞. (See table 7.6)

Table 7.6 Examples of Operation with Maximum Values Used as Arguments

Multiplicand	Multiplier	Result
>H'7F800000	Positive number	H'7F800000 (+∞)
(+∞)	Negative number	H'FF800000 (–∞)
<H'FF800000	Positive number	H'FF800000 (–∞)
(–∞)	Negative number	H'7F800000 (+∞)
Positive number	>H'7F800000 (+∞)	H'7F800000 (+∞)
	<H'FF800000 (–∞)	H'FF800000 (–∞)
Negative number	>H'7F800000 (+∞)	H'FF800000 (–∞)
	<H'FF800000 (–∞)	H'7F800000 (+∞)

d. H'80000000 is treated as H'00000000 (zero).

e. After execution of the software FMUL, the multiplicand and multiplier data are destroyed. If the input arguments are necessary after software FMUL execution, save them on memory.

3. Data memory

The software FMUL does not use the data memory.

4. Example of use

Set a multiplicand and a multiplier in the general-purpose registers and call the software FMUL as a subroutine.

```
WORK1    . RES. B    2          ·····  Reserves a data memory area
WORK2    . RES. B    2          ·····  in which the user program
WORK3    . RES. B    2          ·····  places { a multiplicand.
                                              { a multiplier.
                                              { the result of multiplication.
           :
         MOV. W  @WORK1,R0      ·····  Places in R0 and R1 the multiplicand set
         MOV. W  @WORK1+2,R1    ·····  by the user program.
         MOV. W  @WORK2,R2      ·····  Places in R2 and R3 the multiplier set by
         MOV. W  @WORK2+2,R3    ·····  the user program.
         JSR     @FMUL          ·····  Calls the software FMUL as a subroutine.
         MOV. W  R0, @WORK3     ·····  Places in R0 and R1 the result of
         MOV. W  R1, @WORK3+2   ·····  multiplication set in the output argument.
           :
```

5. Operation

Multiplication of short floating-point numbers is done in the following steps:

a. The software checks whether the multiplicand and multiplier are "0".

(i) If either the multiplicand or multiplier is "0", H'00000000 is output.

b. The software checks whether the multiplicand and multiplier are infinite.

If they are infinite, the values listed in table 7.6 are output.

c. Assume that the multiplicand is R_1 (sign bit=S_1, exponent part=α_1, mantissa part=β_1) and the multiplier is R_2 (sign bit=S_2, exponent part= α_2, mantissa part= β_2). Then R1 and R2 are given by

$$R1 = (-1)^{S1} \times 2^{\alpha 1-127} \times \beta_1$$

$$R2 = (-1)^{S2} \times 2^{\alpha 2-127} \times \beta_2$$

Multiplication of these two numbers is given by

$$R1 \times R2 = (-1)^{S1+S2} \times 2^{\alpha 1+ \alpha 2-127-127} \times \beta_1 \times \beta_2$$

In the case of the floating-point format, the multiplication equation changes as follows, because H'7F (D'127) is added to the result of multiplication of the exponent parts:

$$R_1 \times R_2 = (-1)^{S1+S2} \times 2^{\alpha 1+ \alpha 2-127} \times \beta_1 \times \beta_2$$

Thus, the multiplication is performed in the steps below:

(i) The software checks the sign bits of R1 × R2.

(ii) Addition is done on the exponent parts.

Both $\alpha_1$and α_2 involve addition of H'7F (D'127) according to the floating-point format. As H'7F (D'127) is also added to the result of multiplication, the operation goes as follows:

$$(\alpha_1 - H'7F) + (\alpha_2 - H'7F) + H'7F = \alpha_1 + \alpha_2 - H'7F$$

(In the case of the denormalized format, 1 is added to the exponent part before multiplication is done.)

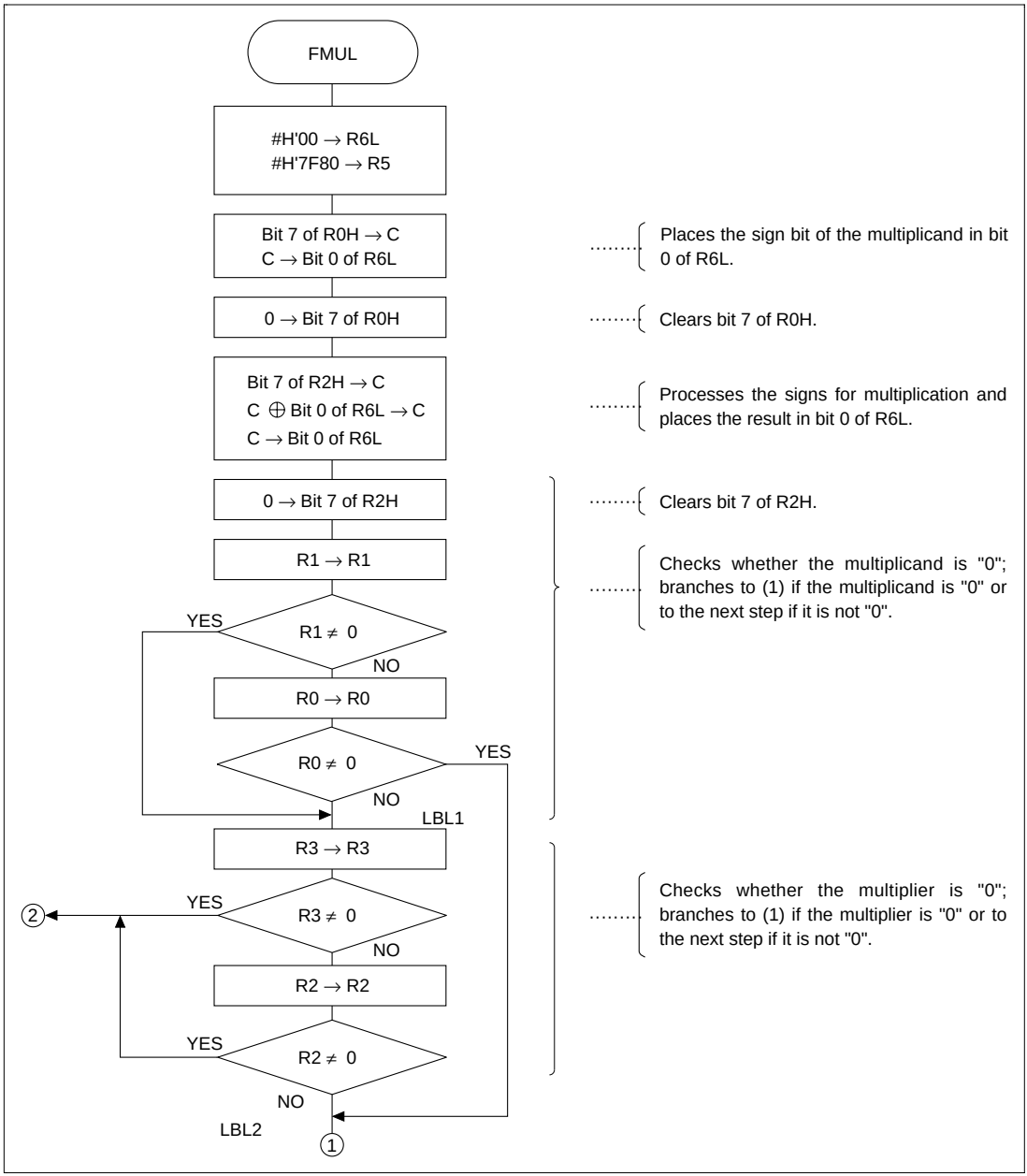
(iii) Multiplication is done on the mantissa parts.

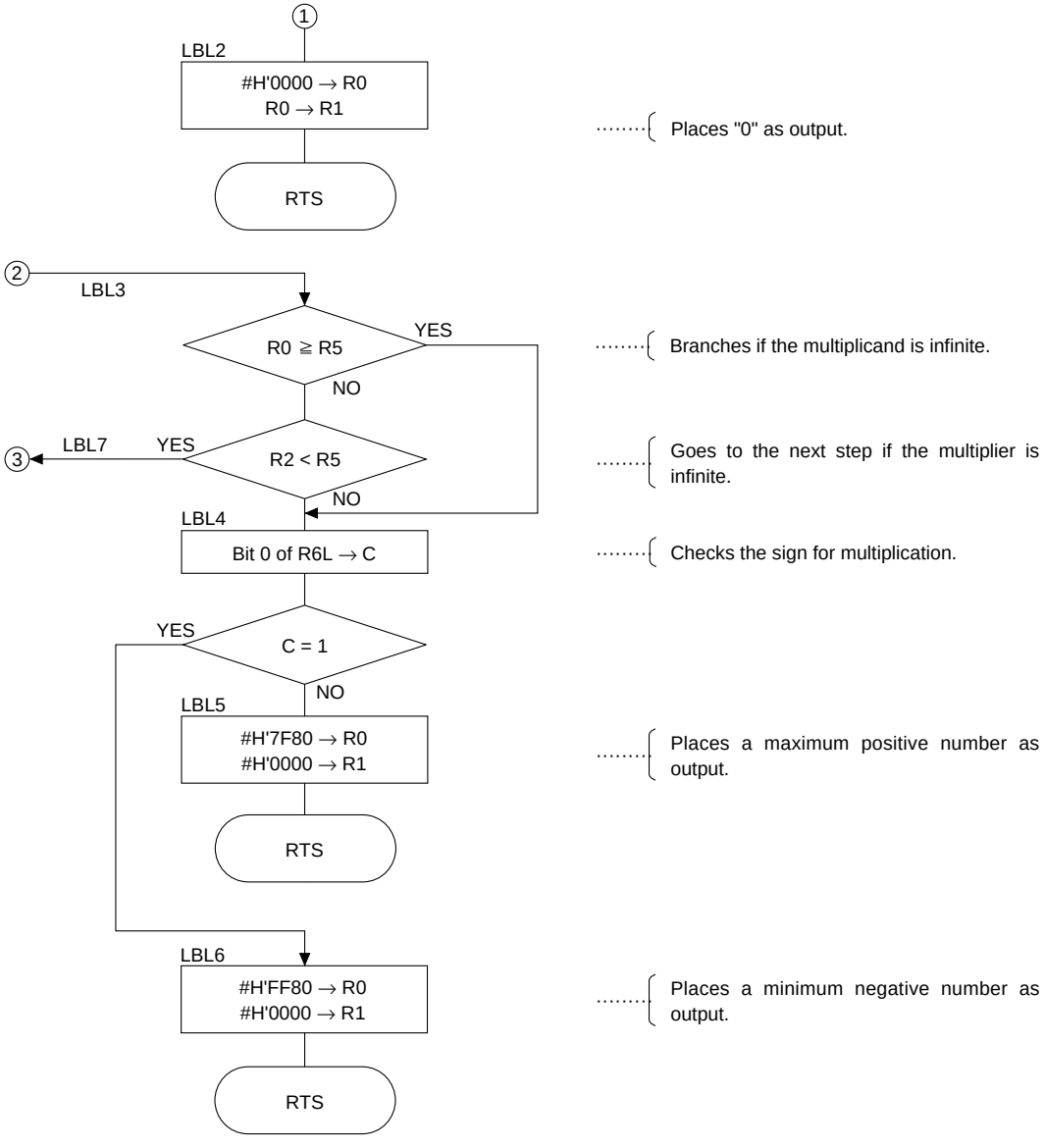
This operation includes the value of the implicit MSB.

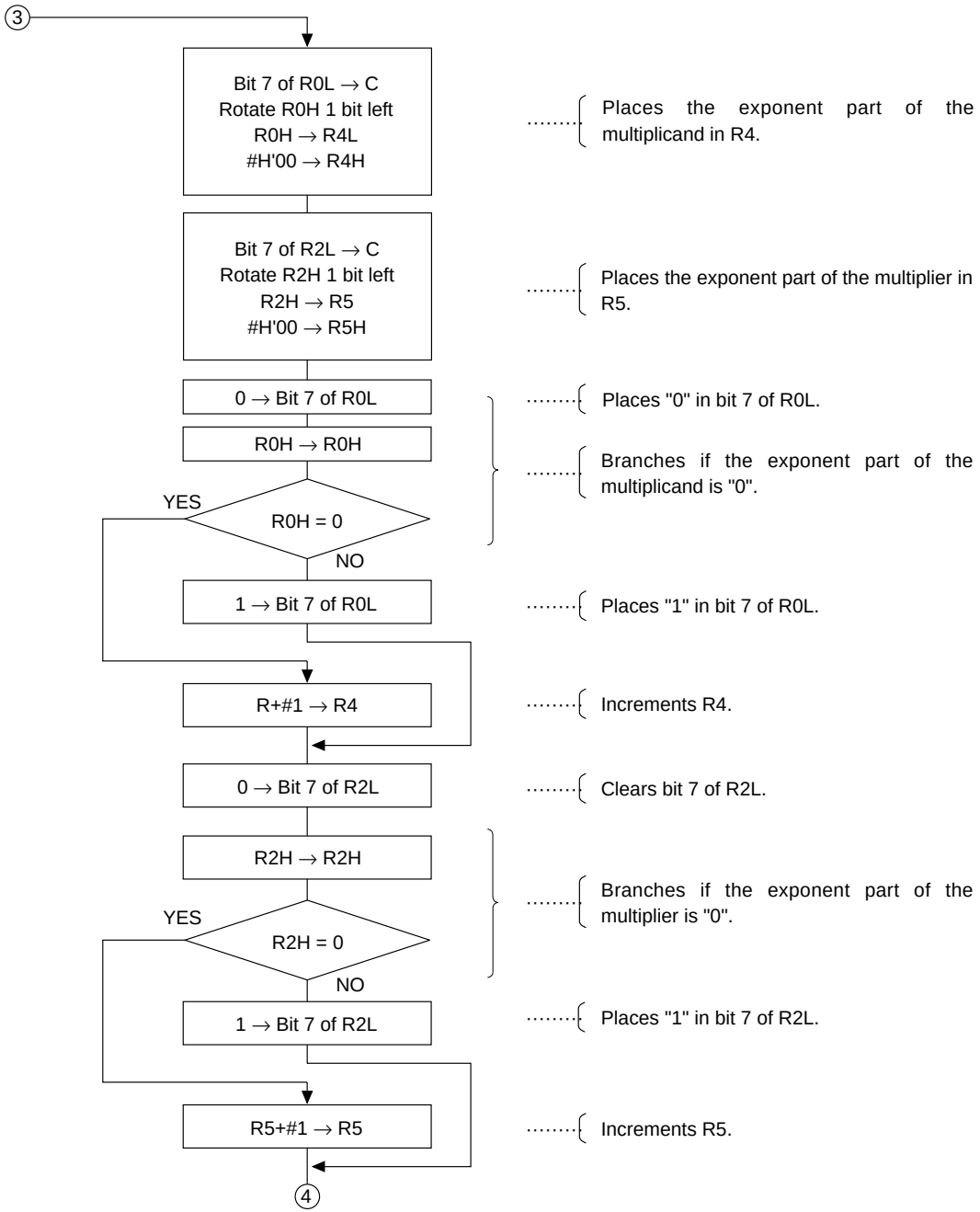
(In the case of the denormalized format, the implicit MSB of the mantissa part is treated as "0".)

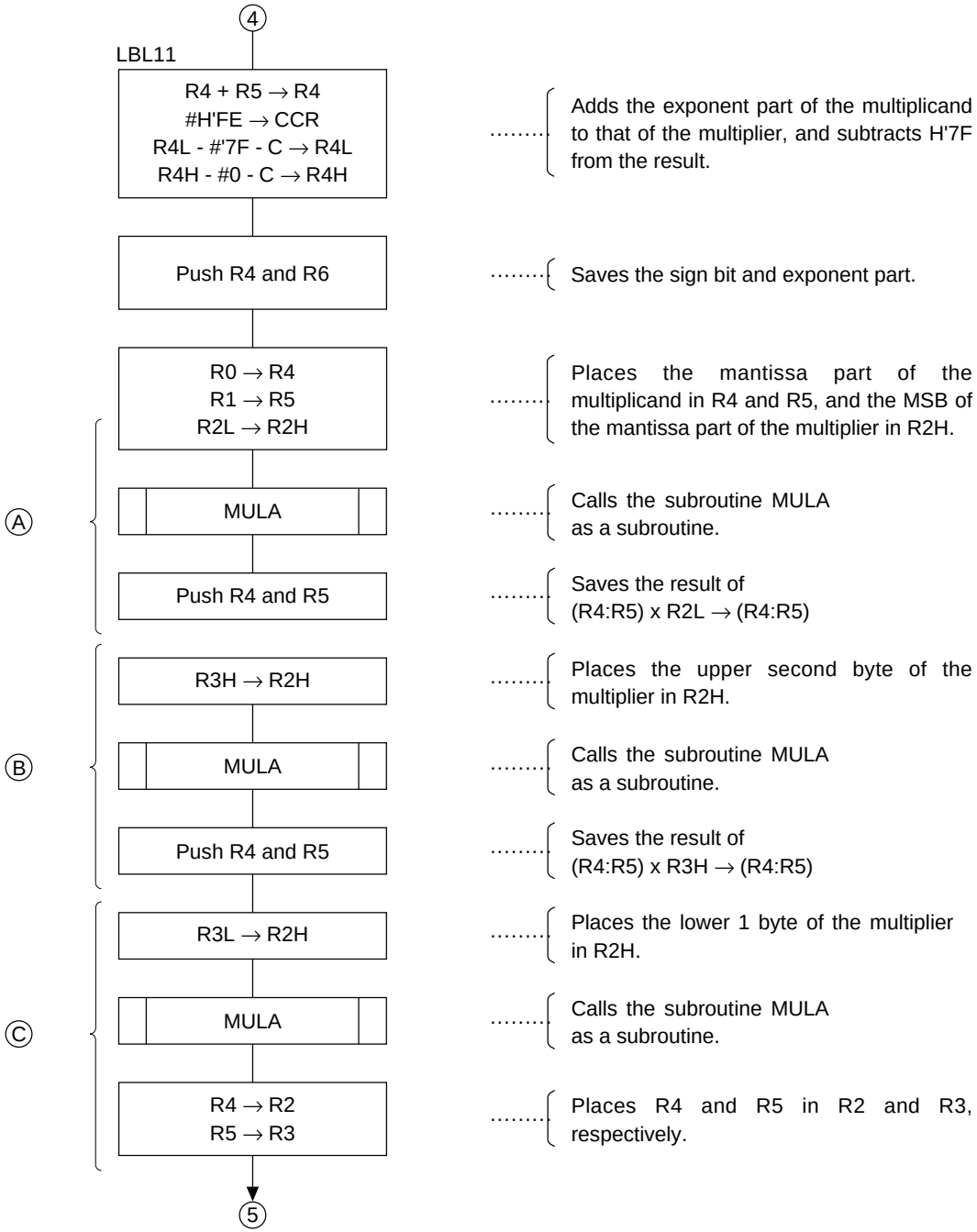
(iv) The result of multiplication is represented in floating-point format.

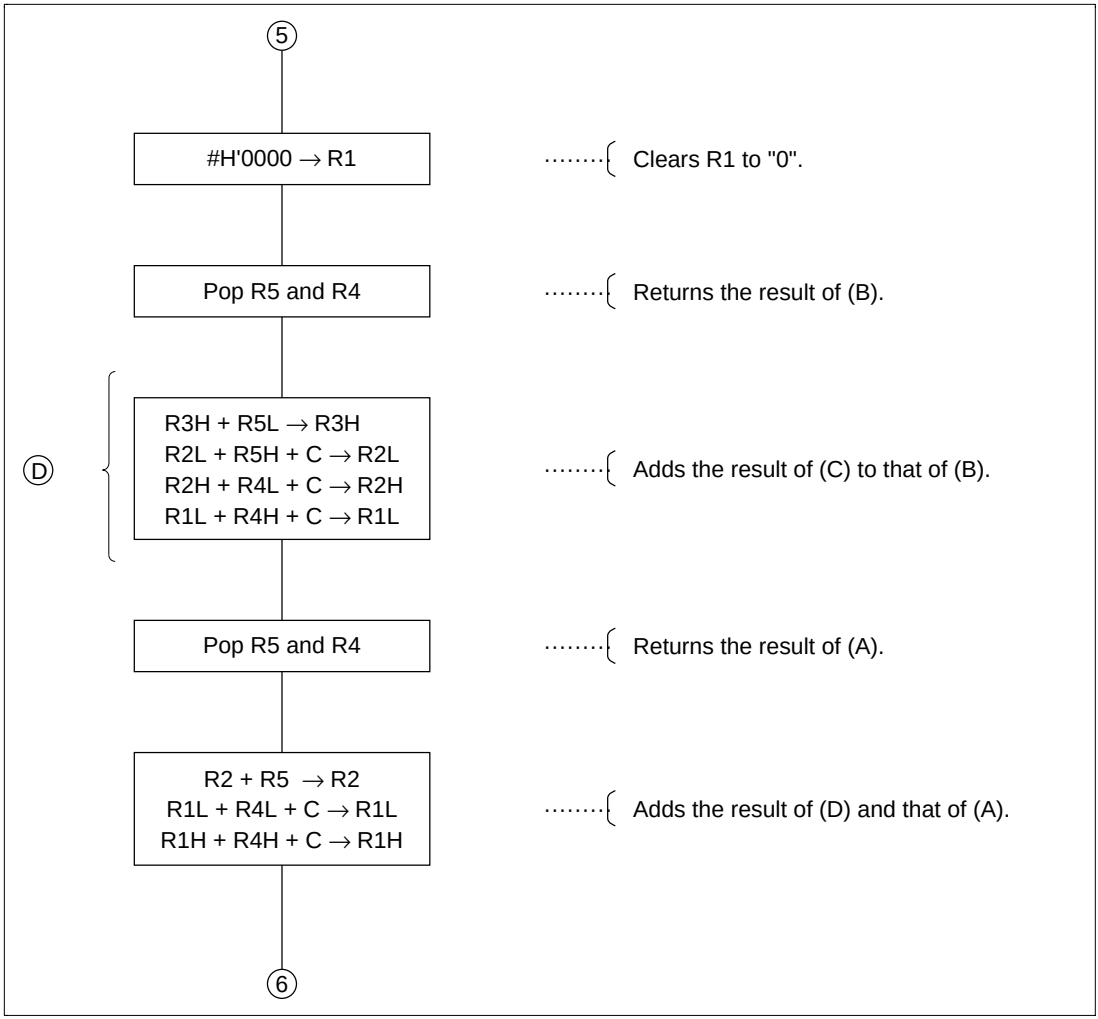
7.18.6 Flowchart

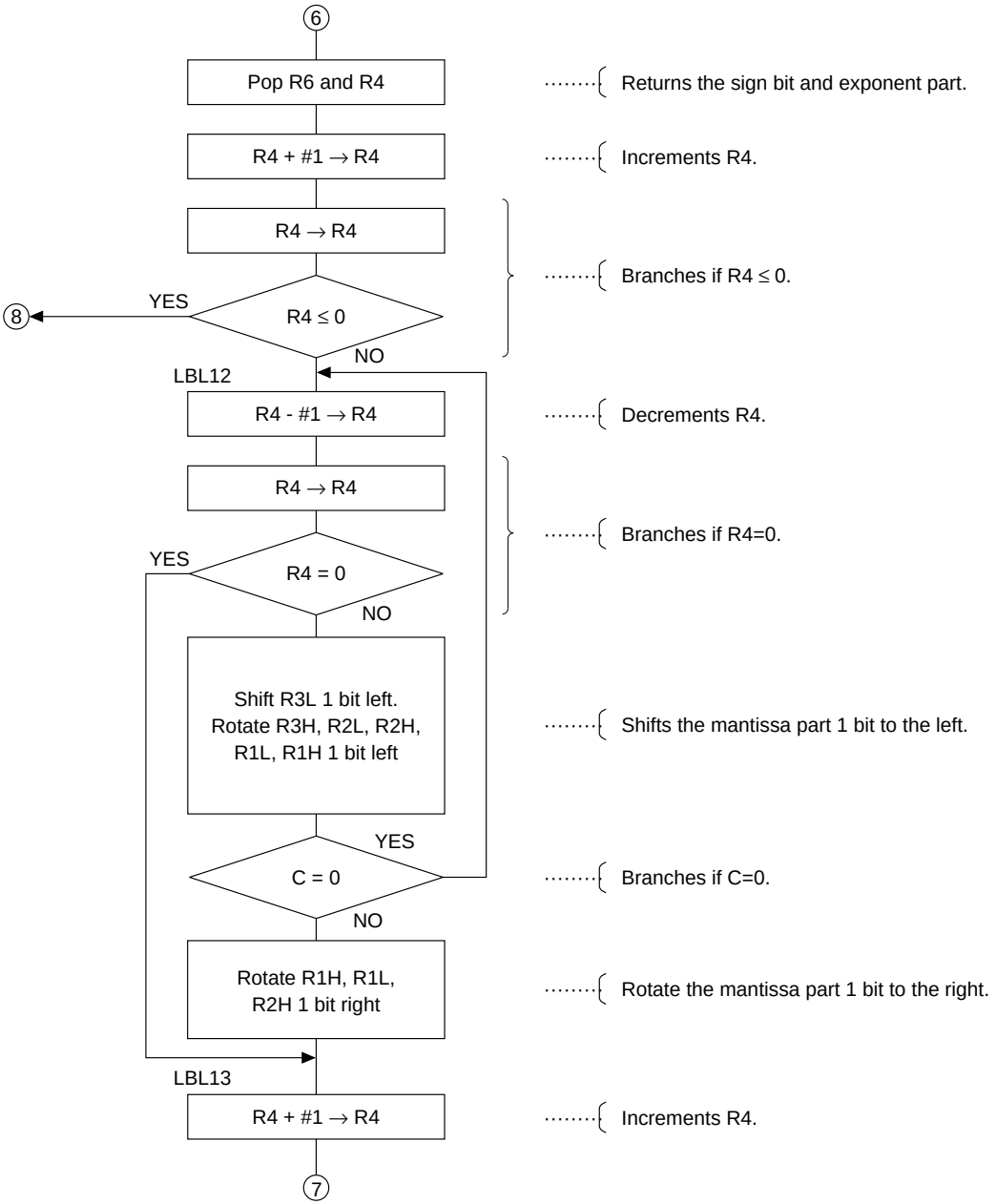


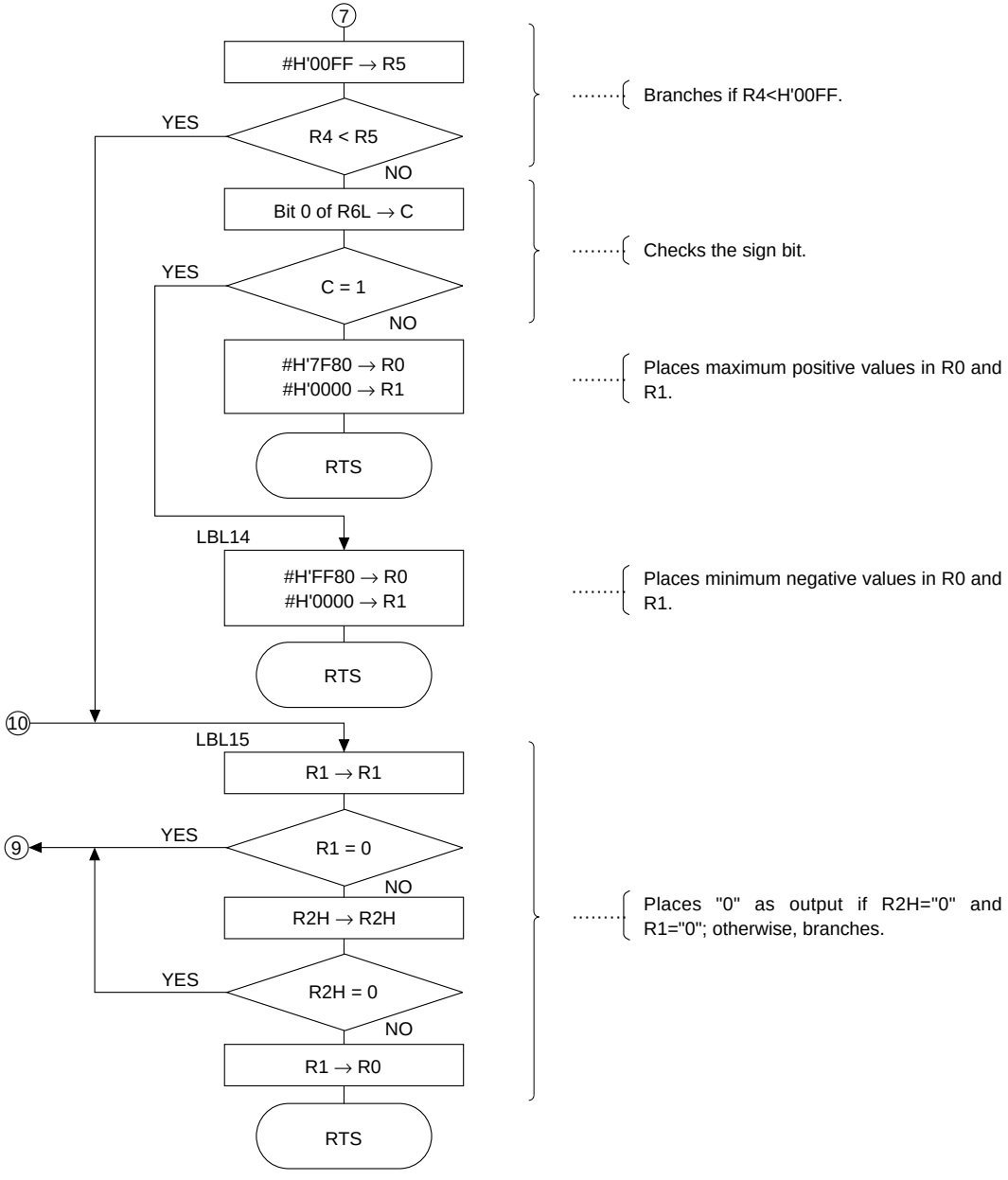

1
LBL2
#H'0000 → R0
R0 → R1
RTS
Places "0" as output.
2
LBL3
R0 ≧ R5
YES
NO
Branches if the multiplicand is infinite.
3
LBL7
YES
R2 < R5
NO
Goes to the next step if the multiplier is infinite.
LBL4
Bit 0 of R6L → C
Checks the sign for multiplication.
YES
C = 1
NO
LBL5
#H'7F80 → R0
#H'0000 → R1
RTS
Places a maximum positive number as output.
LBL6
#H'FF80 → R0
#H'0000 → R1
RTS
Places a minimum negative number as output.

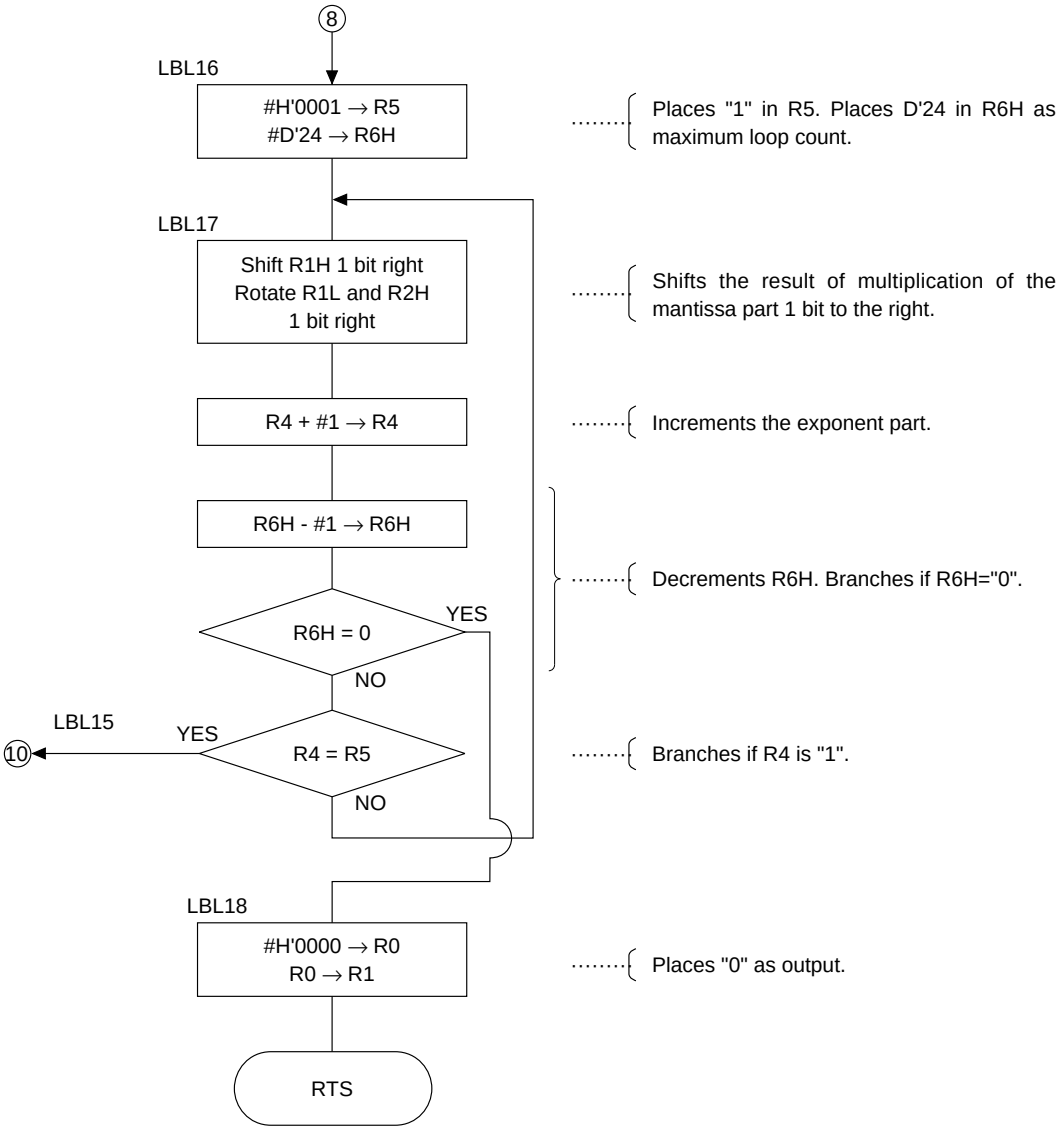

3
Bit 7 of R0L → C
Rotate R0H 1 bit left
R0H → R4L
#H'00 → R4H
Places the exponent part of the multiplicand in R4.
Bit 7 of R2L → C
Rotate R2H 1 bit left
R2H → R5
#H'00 → R5H
Places the exponent part of the multiplier in R5.
0 → Bit 7 of R0L
Places "0" in bit 7 of R0L.
R0H → R0H
Branches if the exponent part of the multiplicand is "0".
YES
R0H = 0
NO
1 → Bit 7 of R0L
Places "1" in bit 7 of R0L.
R+#1 → R4
Increments R4.
0 → Bit 7 of R2L
Clears bit 7 of R2L.
R2H → R2H
Branches if the exponent part of the multiplier is "0".
YES
R2H = 0
NO
1 → Bit 7 of R2L
Places "1" in bit 7 of R2L.
R5+#1 → R5
Increments R5.
4

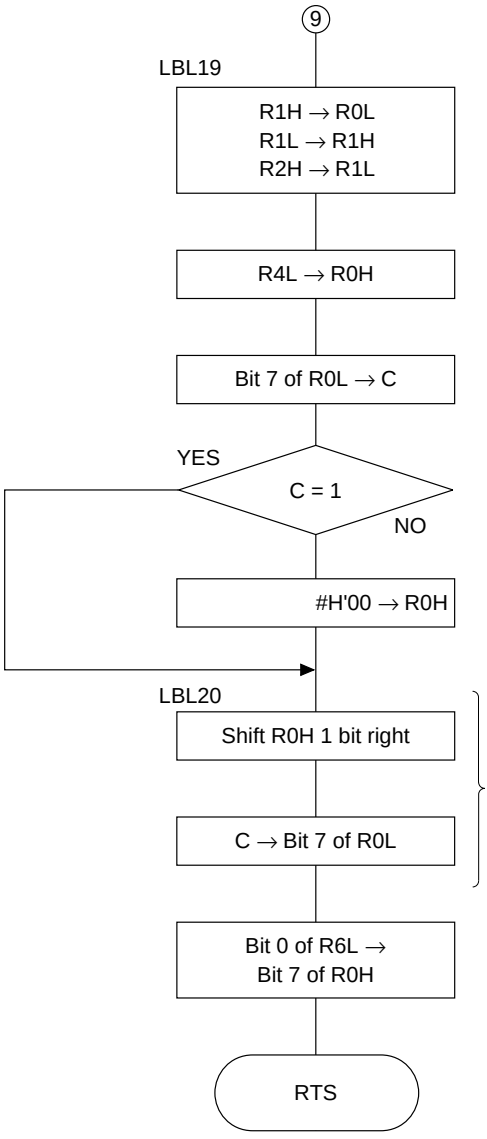

4
LBL11
R4 + R5 → R4
#H'FE → CCR
R4L - #'7F - C → R4L
R4H - #0 - C → R4H
Adds the exponent part of the multiplicand to that of the multiplier, and subtracts H'7F from the result.
Push R4 and R6
Saves the sign bit and exponent part.
R0 → R4
R1 → R5
R2L → R2H
Places the mantissa part of the multiplicand in R4 and R5, and the MSB of the mantissa part of the multiplier in R2H.
A
MULA
Calls the subroutine MULA as a subroutine.
Push R4 and R5
Saves the result of (R4:R5) x R2L → (R4:R5)
R3H → R2H
Places the upper second byte of the multiplier in R2H.
B
MULA
Calls the subroutine MULA as a subroutine.
Push R4 and R5
Saves the result of (R4:R5) x R3H → (R4:R5)
R3L → R2H
Places the lower 1 byte of the multiplier in R2H.
C
MULA
Calls the subroutine MULA as a subroutine.
R4 → R2
R5 → R3
Places R4 and R5 in R2 and R3, respectively.
5


5
#H'0000 → R1
Clears R1 to "0".
Pop R5 and R4
Returns the result of (B).
D
R3H + R5L → R3H
R2L + R5H + C → R2L
R2H + R4L + C → R2H
R1L + R4H + C → R1L
Adds the result of (C) to that of (B).
Pop R5 and R4
Returns the result of (A).
R2 + R5 → R2
R1L + R4L + C → R1L
R1H + R4H + C → R1H
Adds the result of (D) and that of (A).
6


6
Pop R6 and R4
Returns the sign bit and exponent part.
R4 + #1 → R4
Increments R4.
R4 → R4
Branches if R4 ≤ 0.
YES
8
R4 ≤ 0
NO
LBL12
R4 - #1 → R4
Decrements R4.
R4 → R4
Branches if R4=0.
YES
R4 = 0
NO
Shift R3L 1 bit left.
Rotate R3H, R2L, R2H,
R1L, R1H 1 bit left
Shifts the mantissa part 1 bit to the left.
YES
C = 0
Branches if C=0.
NO
Rotate R1H, R1L,
R2H 1 bit right
Rotate the mantissa part 1 bit to the right.
LBL13
R4 + #1 → R4
Increments R4.
7


7
#H'00FF → R5
R4 < R5
YES
NO
Branches if R4<H'00FF.
Bit 0 of R6L → C
C = 1
YES
NO
Checks the sign bit.
#H'7F80 → R0
#H'0000 → R1
RTS
Places maximum positive values in R0 and R1.
LBL14
#H'FF80 → R0
#H'0000 → R1
RTS
Places minimum negative values in R0 and R1.
10
LBL15
R1 → R1
R1 = 0
YES
NO
9
R2H → R2H
R2H = 0
YES
NO
R1 → R0
RTS
Places "0" as output if R2H="0" and R1="0"; otherwise, branches.


8
LBL16
#H'0001 → R5
#D'24 → R6H
Places "1" in R5. Places D'24 in R6H as maximum loop count.
LBL17
Shift R1H 1 bit right
Rotate R1L and R2H
1 bit right
Shifts the result of multiplication of the mantissa part 1 bit to the right.
R4 + #1 → R4
Increments the exponent part.
R6H - #1 → R6H
Decrements R6H. Branches if R6H="0".
R6H = 0
YES
NO
LBL15
YES
10
R4 = R5
Branches if R4 is "1".
NO
LBL18
#H'0000 → R0
R0 → R1
Places "0" as output.
RTS



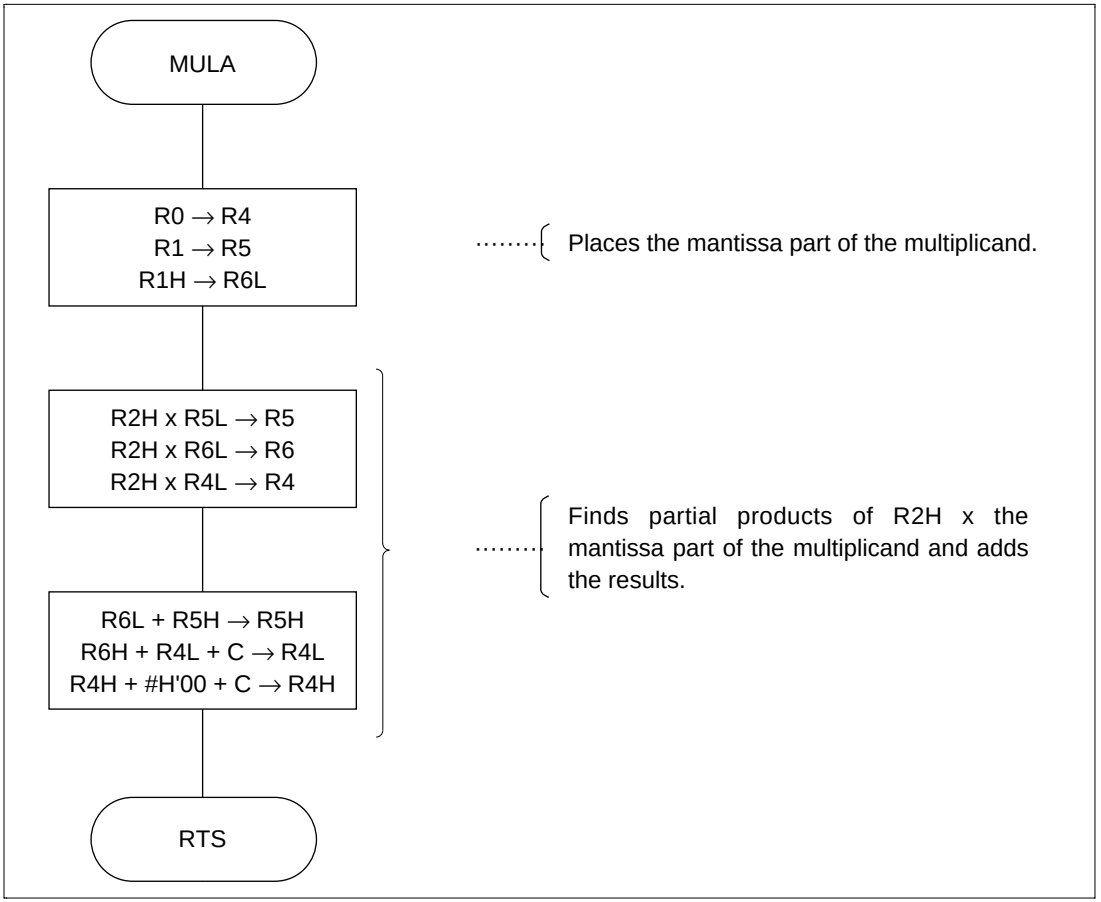
Places the mantissa parts in respective registers.

Places the exponent part in R0H and changes it to floating-point format.

Sets the exponent part to "0" if the MSB of the mantissa part is "0". (Denormalized format)

Changes the exponent part to floating-point format.

Sets the sign bit.


MULA
R0 → R4
R1 → R5
R1H → R6L
Places the mantissa part of the multiplicand.
R2H x R5L → R5
R2H x R6L → R6
R2H x R4L → R4
R6L + R5H → R5H
R6H + R4L + C → R4L
R4H + #H'00 + C → R4H
Finds partial products of R2H x the mantissa part of the multiplicand and adds the results.
RTS

7.18.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:22:23
PROGRAM NAME =

    1                                ;*********************************************************************
    2                                ;*
    3                                ;*   00 - NAME       :FLOATING POINT MULTIPLICATION (FMUL)
    4                                ;*
    5                                ;*********************************************************************
    6                                ;*
    7                                ;*   ENTRY           :R0     (UPPER WORD OF MULTI PLICAND)
    8                                ;*                    R1     (LOWER WORD OF MULTI PLICAND)
    9                                ;*                    R2     (UPPER WORD OF MULTIPLIER)
   10                                ;*                    R3     (LOWER WORD OF MULTIPLIER)
   11                                ;*
   12                                ;*   RETURNS         :R0     (UPPER WORD OF RESULT)
   13                                ;*                    R1     (LOWER WORD OF RESULT)
   14                                ;*
   15                                ;*********************************************************************
   16                                ;
   17 FMUL_cod C 0000                     .SECTION        FMUL_code,CODE,ALIGN=2
   18                                     .EXPORT         FMUL
   19                                ;
   20 FMUL_cod C      00000000       FMUL .EQU    $               ;Entry point
   21 FMUL_cod C 0000 FE00                MOV.B   #H'00,R6L       ;Clear R6L
   22 FMUL_cod C 0002 79057F80            MOV.W   #H'7F80,R5      ;Set "H'7F80"
   23                                ;
   24 FMUL_cod C 0006 7770                BLD     #7,R0H          ;Set sign bit of multiplicand
   25 FMUL_cod C 0008 670E                BST     #0,R6L          ; to bit 0 of R6L
   26 FMUL_cod C 000A 7270                BCLR    #7,R0H          ;Bit clear bit 7 of R0H
   27                                ;
   28 FMUL_cod C 000C 7772                BLD     #7,R2H          ;
   29 FMUL_cod C 000E 750E                BXOR    #0,R6L          ;Set sign bit of result
   30 FMUL_cod C 0010 670E                BST     #0,R6L          ; to bit 0 of R6L
   31 FMUL_cod C 0012 7272                BCLR    #7,R2H          ;Bit clear bit 7 of R2H
   32                                ;
   33 FMUL_cod C 0014 0D11                MOV.W   R1,R1
   34 FMUL_cod C 0016 4604                BNE     LBL1
   35 FMUL_cod C 0018 0D00                MOV.W   R0,R0
   36 FMUL_cod C 001A 4708                BEQ     LBL2            ;Branch if R1=R0=0
   37 FMUL_cod C 001C                LBL1
   38 FMUL_cod C 001C 0D33                MOV.W   R3,R3
   39 FMUL_cod C 001E 460C                BNE     LBL3            ;Branch if not R3=0
   40 FMUL_cod C 0020 0D22                MOV.W   R2,R2
   41 FMUL_cod C 0022 4608                BNE     LBL3            ;Branch if not R2=0
   42                                ;
   43 FMUL_cod C 0024                LBL2
   44 FMUL_cod C 0024 79000000            MOV.W   #H'0000,R0      ;Set 0 to result
   45 FMUL_cod C 0028 0D01                MOV.W   R0,R1
   46 FMUL_cod C 002A 5470                RTS
   47                                ;
   48 FMUL_cod C 002C                LBL3
   49 FMUL_cod C 002C 1D05                CMP.W   R0,R5
   50 FMUL_cod C 002E 4304                BLS     LBL4            ;Branch if R0>=R5
   51 FMUL_cod C 0030 1D25                CMP.W   R2,R5
   52 FMUL_cod C 0032 4218                BHI     LBL7            ;Branch if R2>=R5
   53 FMUL_cod C 0034                LBL4
   54 FMUL_cod C 0034 770E                BLD     #0,R6L          ;Load sign bit
   55 FMUL_cod C 0036 450A                BCS     LBL6            ;Branch if C=1
   56 FMUL_cod C 0038                LBL5
   57 FMUL_cod C 0038 79007F80            MOV.W   #H'7F80,R0      ;Set #H'7F800000 to result
   58 FMUL_cod C 003C 79010000            MOV.W   #H'0000,R1
   59 FMUL_cod C 0040 5470                RTS
   60 FMUL_cod C 0042                LBL6
   61 FMUL_cod C 0042 7900FF80            MOV.W   #H'FF80,R0      ;Set #H'FF800000 to result
   62 FMUL_cod C 0046 79010000            MOV.W   #H'0000,R1
   63 FMUL_cod C 004A 5470                RTS
```

```
 64                                     ;
 65 FMUL_cod C 004C                     LBL7
 66 FMUL_cod C 004C 7778                     BLD     #7,R0L          ;
 67 FMUL_cod C 004E 1200                     ROTXL   R0H             ;
 68 FMUL_cod C 0050 0C0C                     MOV.B   R0H,R4L         ;Set exponent of multiplicand to R4
 69 FMUL_cod C 0052 F400                     MOV.B   #H'00,R4H
 70                                     ;
 71 FMUL_cod C 0054 777A                     BLD     #7,R2L
 72 FMUL_cod C 0056 1202                     ROTXL   R2H
 73 FMUL_cod C 0058 0C2D                     MOV.B   R2H,R5L         ;Set exponent of multiplier to R5
 74 FMUL_cod C 005A F500                     MOV.B   #H'00,R5H
 75                                     ;
 76 FMUL_cod C 005C 7278                     BCLR    #7,R0L          ;Clear bit 7 of R0L
 77 FMUL_cod C 005E 0C00                     MOV.B   R0H,R0H
 78 FMUL_cod C 0060 4704                     BEQ     LBL8            ;Branch if multiplicand is denormalized
 79 FMUL_cod C 0062 7078                     BSET    #7,R0L          ;Set implicit MSB
 80 FMUL_cod C 0064 4002                     BRA     LBL9            ;Branch always
 81 FMUL_cod C 0066                     LBL8
 82 FMUL_cod C 0066 0B04                     ADDS.W  #1,R4
 83                                     ;
 84 FMUL_cod C 0068                     LBL9
 85 FMUL_cod C 0068 727A                     BCLR    #7,R2L          ;Clear bit 7 of R2L
 86 FMUL_cod C 006A 0C22                     MOV.B   R2H,R2H
 87 FMUL_cod C 006C 4704                     BEQ     LBL10           ;Branch if multiplier is denormalized
 88 FMUL_cod C 006E 707A                     BSET    #7,R2L          ;Set implicit MSB
 89 FMUL_cod C 0070 4002                     BRA     LBL11           ;Branch always
 90 FMUL_cod C 0072                     LBL10
 91 FMUL_cod C 0072 0B05                     ADDS.W  #1,R5
 92                                     ;
 93 FMUL_cod C 0074                     LBL11
 94 FMUL_cod C 0074 0954                     ADD.W   R5,R4           ;addition exponents
 95 FMUL_cod C 0076 06FE                     ANDC    #H'FE,CCR       ;Clear C flag of CCR
 96 FMUL_cod C 0078 BC7F                     SUBX.B  #H'7F,R4L       ;R4L - #H'7F - C -> R4L
 97 FMUL_cod C 007A B400                     SUBX.B  #H'00,R4H
 98                                     ;
 99 FMUL_cod C 007C 6DF4                     PUSH    R4              ;Push R4
100 FMUL_cod C 007E 6DF6                     PUSH    R6              ;Push R6
101                                     ;
102 FMUL_cod C 0080 0D04                     MOV.W   R0,R4           ;
103 FMUL_cod C 0082 0D15                     MOV.W   R1,R5
104                                     ;
105 FMUL_cod C 0084 0CA2                     MOV.B   R2L,R2H
106 FMUL_cod C 0086 5E000000                JSR     @MULA           ;R2L * (R0L:R1) -> (R4:R5)
107 FMUL_cod C 008A 6DF4                     PUSH    R4              ;Push    R4
108 FMUL_cod C 008C 6DF5                     PUSH    R5              ;Push    R5
109                                     ;
110 FMUL_cod C 008E 0C32                     MOV.B   R3H,R2H
111 FMUL_cod C 0090 5E000000                JSR     @MULA           ;R3L * (R0L:R1) -> (R4:R5)
112 FMUL_cod C 0094 6DF4                     PUSH    R4              ;Push    R4
113 FMUL_cod C 0096 6DF5                     PUSH    R5              ;Push    R5
114                                     ;
115 FMUL_cod C 0098 0CB2                     MOV.B   R3L,R2H         ;
116 FMUL_cod C 009A 5E000000                JSR     @MULA           ;R3L * (R0L:R1) -> (R4:R5)
117 FMUL_cod C 009E 0D42                     MOV.W   R4,R2           ;Push    R4
118 FMUL_cod C 00A0 0D53                     MOV.W   R5,R3           ;Push    R5
119                                     ;
120 FMUL_cod C 00A2 79010000                 MOV.W   #H'0000,R1      ;Clear R1
121 FMUL_cod C 00A6 6D75                     POP     R5              ;Pop     R5
122 FMUL_cod C 00A8 6D74                     POP     R4              ;Pop     R4
123                                     ;
124 FMUL_cod C 00AA 08D3                     ADD.B   R5L,R3H         ;R3H + R5L     -> R3H
125 FMUL_cod C 00AC 0E5A                     ADDX.B  R5H,R2L         ;R2L + R5H + C -> R2L
126 FMUL_cod C 00AE 0EC2                     ADDX.B  R4L,R2H         ;R2H + R4L + C -> R2H
127 FMUL_cod C 00B0 0E49                     ADDX.B  R4H,R1L         ;R1L + R4H + C -> R1L
128                                     ;
129 FMUL_cod C 00B2 6D75                     POP     R5              ;Pop     R5
130 FMUL_cod C 00B4 6D74                     POP     R4              ;Pop     R4
131 FMUL_cod C 00B6 0952                     ADD.W   R5,R2           ;R2  + R5      -> R2
132 FMUL_cod C 00B8 0EC9                     ADDX.B  R4L,R1L         ;R1L + R4L + C -> R1L
133 FMUL_cod C 00BA 0E41                     ADDX.B  R4H,R1H         ;R1H + R4H + C -> R1H
```

```
                                        ;
FMUL_cod C 00BC 6D76                        POP     R6              ;Pop     R6
FMUL_cod C 00BE 6D74                        POP     R4              ;Pop     R4
FMUL_cod C 00C0 0B04                        ADDS.W  #1,R4
FMUL_cod C 00C2 0D44                        MOV.W   R4,R4
                                        ;
FMUL_cod C 00C4 474A                        BEQ     LBL16           ;Branch if R4=0
FMUL_cod C 00C6 4B48                        BMI     LBL16           ;Branch if R4<0
FMUL_cod C 00C8                         LBL12
FMUL_cod C 00C8 1B04                        SUBS.W  #1,R4
FMUL_cod C 00CA 0D44                        MOV.W   R4,R4
FMUL_cod C 00CC 4714                        BEQ     LBL13           ;Branch if R4=0
FMUL_cod C 00CE 100B                        SHLL    R3L             ;Shift mantissa 1 bit left
FMUL_cod C 00D0 1203                        ROTXL   R3H
FMUL_cod C 00D2 120A                        ROTXL   R2L
FMUL_cod C 00D4 1202                        ROTXL   R2H
FMUL_cod C 00D6 1209                        ROTXL   R1L
FMUL_cod C 00D8 1201                        ROTXL   R1H
FMUL_cod C 00DA 44EC                        BCC     LBL12           ;Branch if C=0
FMUL_cod C 00DC 1301                        ROTXR   R1H             ;Rotate mantissa 1 bit right
FMUL_cod C 00DE 1309                        ROTXR   R1L
FMUL_cod C 00E0 1302                        ROTXR   R2H
FMUL_cod C 00E2                         LBL13
FMUL_cod C 00E2 0B04                        ADDS.W  #1,R4
                                        ;
FMUL_cod C 00E4 790500FF                    MOV.W   #H'00FF,R5      ;
FMUL_cod C 00E8 1D45                        CMP.W   R4,R5
FMUL_cod C 00EA 4418                        BCC     LBL15           ;Branch if R5>R4
FMUL_cod C 00EC 770E                        BLD     #0,R6L          ;Load sign bit
FMUL_cod C 00EE 450A                        BCS     LBL14           ;Branch if C=1
FMUL_cod C 00F0 79007F80                    MOV.W   #H'7F80,R0      ;Set H'7F800000 to result
FMUL_cod C 00F4 79010000                    MOV.W   #H'0000,R1
FMUL_cod C 00F8 5470                        RTS
                                        ;
FMUL_cod C 00FA                         LBL14
FMUL_cod C 00FA 7900FF80                    MOV.W   #H'FF80,R0      ;Set H'FF800000 to product
FMUL_cod C 00FE 79010000                    MOV.W   #H'0000,R1
FMUL_cod C 0102 5470                        RTS
                                        ;
FMUL_cod C 0104                         LBL15
FMUL_cod C 0104 0D11                        MOV.W   R1,R1
FMUL_cod C 0106 4628                        BNE     LBL19           ;Branch if not R1=0
FMUL_cod C 0108 0C22                        MOV.B   R2H,R2H
FMUL_cod C 010A 4624                        BNE     LBL19           ;Branch if not R2H=0
FMUL_cod C 010C 0D10                        MOV.W   R1,R0
FMUL_cod C 010E 5470                        RTS
                                        ;
FMUL_cod C 0110                         LBL16
FMUL_cod C 0110 79050001                    MOV.W   #H'0001,R5      ;Set #H'0001 to R5
FMUL_cod C 0114 F618                        MOV.B   #D'24,R6H       ;Se bit counter
FMUL_cod C 0116                         LBL17
FMUL_cod C 0116 1101                        SHLR    R1H             ;Shift mantissa 1 bit right
FMUL_cod C 0118 1309                        ROTXR   R1L
FMUL_cod C 011A 1302                        ROTXR   R2H
FMUL_cod C 011C 0B04                        ADDS.W  #1,R4           ;Increment exponent
FMUL_cod C 011E 1A06                        DEC.B   R6H             ;Decrement bit counter
FMUL_cod C 0120 4706                        BEQ     LBL18           ;Branch if Z=1
FMUL_cod C 0122 1D54                        CMP.W   R5,R4
FMUL_cod C 0124 47DE                        BEQ     LBL15           ;Branch if R5=R4
FMUL_cod C 0126 40EE                        BRA     LBL17           ;Branch always
FMUL_cod C 0128                         LBL18
FMUL_cod C 0128 79000000                    MOV.W   #H'0000,R0      ;Clear result
FMUL_cod C 012C 0D01                        MOV.W   R0,R1
FMUL_cod C 012E 5470                        RTS
                                        ;
FMUL_cod C 0130                         LBL19
FMUL_cod C 0130 0C18                        MOV.B   R1H,R0L
FMUL_cod C 0132 0C91                        MOV.B   R1L,R1H
FMUL_cod C 0134 0C29                        MOV.B   R2H,R1L
                                        ;
```

```
  204 FMUL_cod C 0136 0CC0                    MOV.B   R4L,R0H
  205 FMUL_cod C 0138 7778                    BLD     #7,R0L
  206 FMUL_cod C 013A 4502                    BCS     LBL20           ;Branch if C=1
  207 FMUL_cod C 013C F000                    MOV.B   #H'00,R0H
  208 FMUL_cod C 013E                   LBL20                             ;Change floating point format
  209 FMUL_cod C 013E 1100                    SHLR    R0H
  210 FMUL_cod C 0140 6778                    BST     #7,R0L
  211 FMUL_cod C 0142 770E                    BLD     #0,R6L
  212 FMUL_cod C 0144 6770                    BST     #7,R0H
  213 FMUL_cod C 0146 5470                    RTS
  214                                   ;
  215                                   ;---------------------------------------------
  216                                   ;
  217 FMUL_cod C 0148                   MULA                        ;R2H * (R0L:R1) -> (R4:R5)
  218 FMUL_cod C 0148 0D04                    MOV.W   R0,R4           ;R0  -> R4
  219 FMUL_cod C 014A 0D15                    MOV.W   R1,R5           ;R1  -> R5
  220 FMUL_cod C 014C 0C1E                    MOV.B   R1H,R6L         ;R1H -> R6L
  221                                   ;
  222 FMUL_cod C 014E 5025                    MULXU   R2H,R5          ;R2H * R5L -> R5
  223 FMUL_cod C 0150 5026                    MULXU   R2H,R6          ;R2H * R6L -> R6
  224 FMUL_cod C 0152 5024                    MULXU   R2H,R4          ;R2H * R4L -> R4
  225                                   ;
  226 FMUL_cod C 0154 08E5                    ADD.B   R6L,R5H         ;R5H + R6L       -> R5H
  227 FMUL_cod C 0156 0E6C                    ADDX.B  R6H,R4L         ;R4L + R6H   + C -> R4L
  228 FMUL_cod C 0158 9400                    ADDX.B  #H'00,R4H       ;R4H + #H'00 + C -> R4H

*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:22:23                                        PAGE
5
PROGRAM NAME =

  229 FMUL_cod C 015A 5470                    RTS
  230                                   ;
  231                                         .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

7.19 Square Root of a 32-Bit Binary Number

MCU: H8/300 Series
H8/300L Series

Label name: SQRT

7.19.1 Function

1. The software SQRT finds the square root of a 32-bit binary number and outputs the result in 16-bit binary format.
2. All arguments used with the software SQRT are represented in unsigned integers.
3. All data is manipulated on general-purpose registers.

7.19.2 Arguments

Description		Memory area	Data length (bytes)
Input	32-bit binary number	R4, R5	4
Output	Square root	R3	2

7.19.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	×	↕	×	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

7.19.4 Specifications

Program memory (bytes)
94
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
1340
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

7.19.5 Notes

The clock cycle count (1340) in the specifications is for the example shown in figure 7.46.

7.19.6 Description

1. Details of functions
   a. The following arguments are used with the software SQRT:
      R4: Contains, as an input argument, the upper word of a 32-bit binary number whose square root is to be found.
      R5: Contains, as an input argument, the lower word of the 32-bit binary number whose square root is to be found.
      R3: Contains, as an output argument, the square root of the 32-bit binary number.
   b. Figure 7.46 shows an example of the software SQRT being executed. When the input arguments are set as shown in (1), the square root is placed in R3 shown in (2).

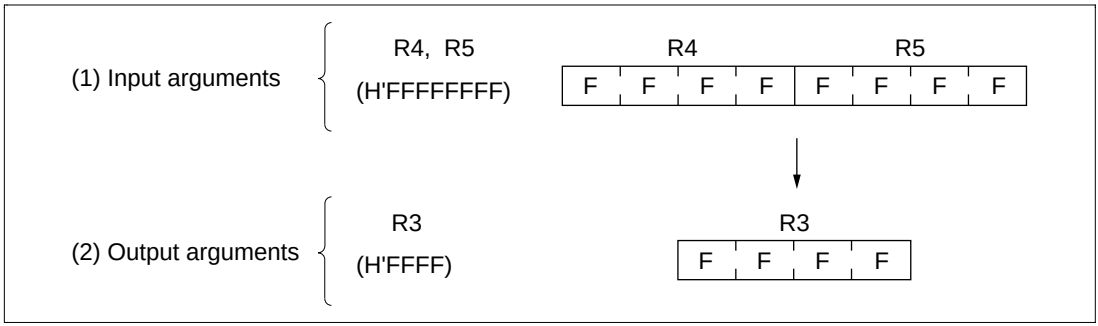


Figure 7.46 Example of Software SQRT Execution

2. Notes on usage
   a. When upper bits are not used (see figure 7.47), set 0's in them; otherwise, no correct result can be obtained because the square root is found on numbers including indeterminate data placed in the upper bits.

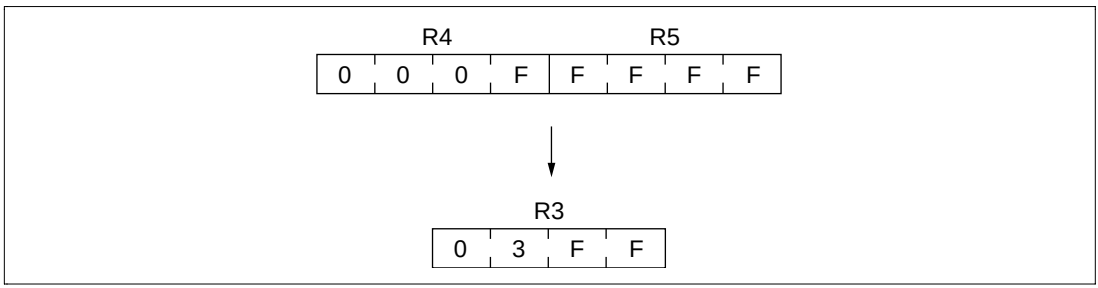


Figure 7.47 Examples of Operation with Upper Bits Unused

   b. Figures below the decimal point are discarded.
3. Data memory
   The software SQRT does not use the data memory.

4. Example of use

Set a 32-bit decimal number whose square root is to be found and call the software SQRT as a subroutine.

```
WORK1    . RES. W    2          ..... Reserves a data memory area in which the
                                       user program places a 32-bit binary
                                       number whose square root is to be found.
WORK2    . RES. W    2          ..... Reserves a data memory area in which the
                                       user program places the square root (16-
                                       bit binary) of the 32-bit binary number.
         MOV. W      @WORK1, R4 ..... Places in the input argument the 32-bit
         MOV. W      @WORK1+2, R5       binary number set by the user program.
         JSR         @SQRT      ..... Calls the software SQRT as a subroutine.
         MOV. W R3,  @WORK2     ..... Places the 16-bit binary square root (set in
                                       the output argument) in the data memory
                                       area of the user program.
```

5. Operation

a. Figure 7.48 shows the method of finding the square root H'05 (binary) of H'22 (a 16-bit binary).

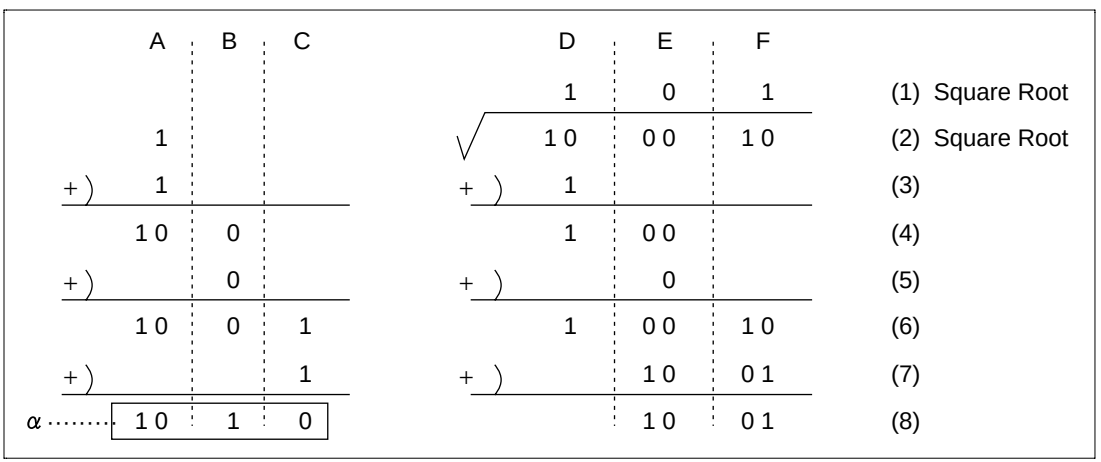


Figure 7.48 Computation to Find Square Root

(i) As shown in figure 7.48, the square root of a binary number can be found by processing the number by 2 bits in bit-descending order.

(ii) The square root ((1)) is equal to Éø (found through processes A, B and C) divided by 2. The software SQRT computes Éø to find the square root.

b. The program is executed in the following steps:

(i) The number of steps (D'16) in which the 32-bit binary number is processed by 2 bits is placed in R6L.

(ii) The square root areas R2 and R3 and the work areas R0 and R1 are cleared.

(iii) R4, R5 and R0, R1 are rotated 2 bits to the left to place the upper 2 bits of the input square root in R0 and R1.

(iv) "1" is set in R2 and R3. (2)

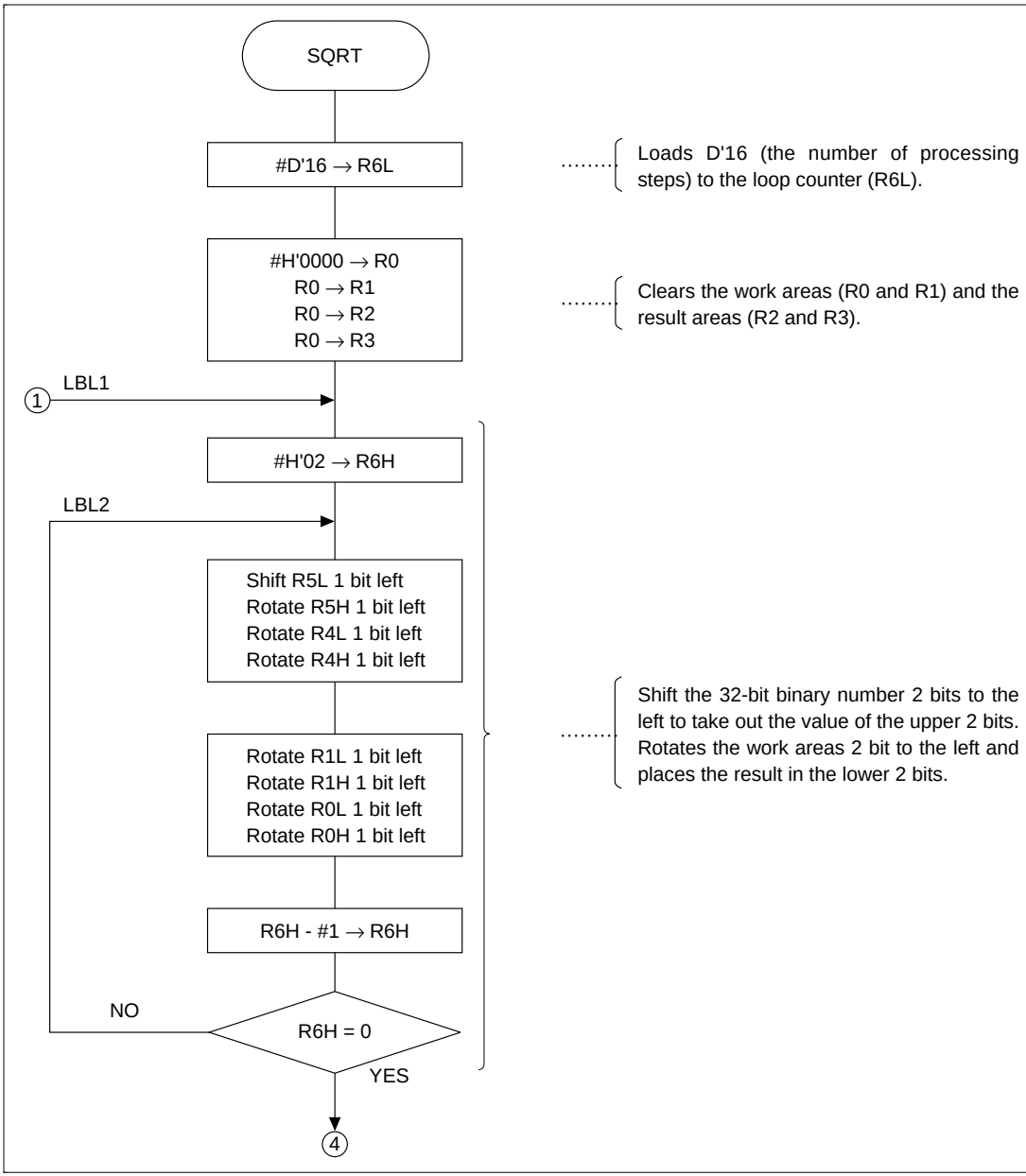
(v) R2 and R3 are subtracted from R0 and R1 to find the difference. (D, (2), (3), (4)) The difference is placed in R0 and R1.

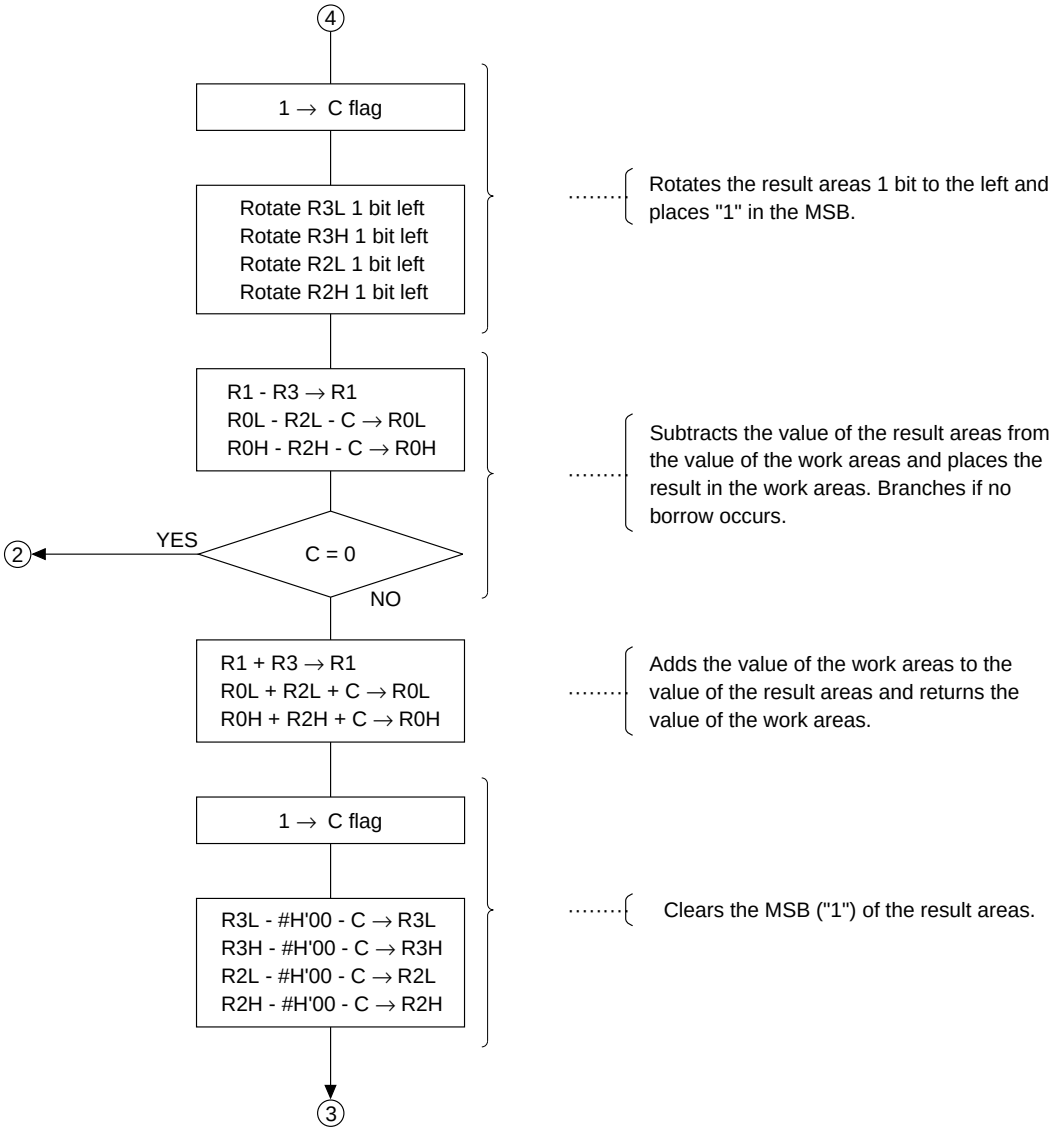
(vi) If the result is positive, R2 and R3 are incremented. (A, -(4))

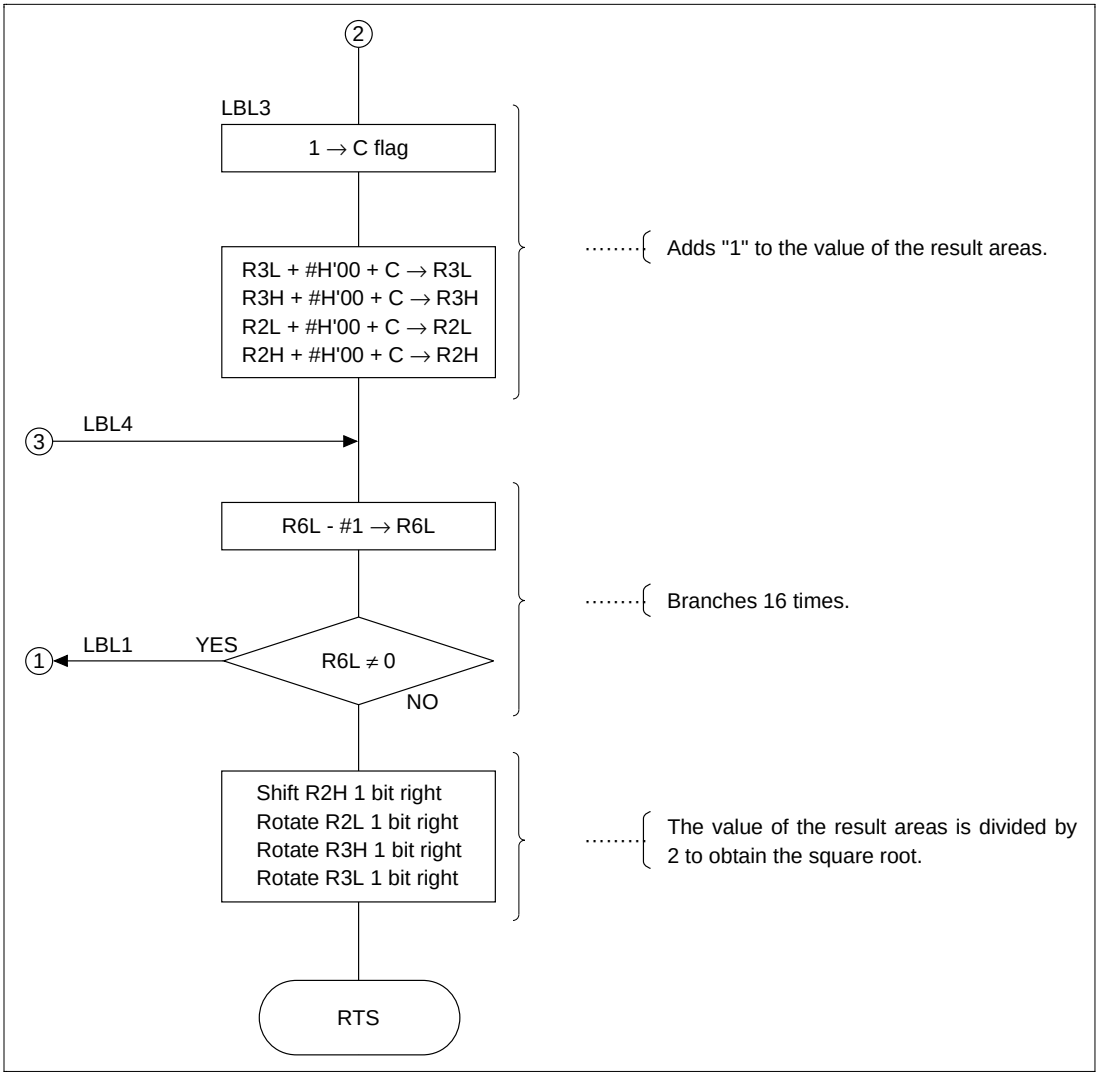
If the result is negative, R2 and R3 are decremented, and R2 and R3 are added to R0 and R1. (D, E, (6))

c. In the software SQRT, R6 is decremented each time the process (iii) through (vi) of (b) is done. This processing continued until R6 reaches "0".

7.19.7 Flowchart




4
1 → C flag
Rotate R3L 1 bit left
Rotate R3H 1 bit left
Rotate R2L 1 bit left
Rotate R2H 1 bit left
Rotates the result areas 1 bit to the left and places "1" in the MSB.
R1 - R3 → R1
R0L - R2L - C → R0L
R0H - R2H - C → R0H
Subtracts the value of the result areas from the value of the work areas and places the result in the work areas. Branches if no borrow occurs.
YES
2
C = 0
NO
R1 + R3 → R1
R0L + R2L + C → R0L
R0H + R2H + C → R0H
Adds the value of the work areas to the value of the result areas and returns the value of the work areas.
1 → C flag
R3L - #H'00 - C → R3L
R3H - #H'00 - C → R3H
R2L - #H'00 - C → R2L
R2H - #H'00 - C → R2H
Clears the MSB ("1") of the result areas.
3


2
LBL3
1 → C flag
R3L + #H'00 + C → R3L
R3H + #H'00 + C → R3H
R2L + #H'00 + C → R2L
R2H + #H'00 + C → R2H
Adds "1" to the value of the result areas.
3
LBL4
R6L - #1 → R6L
Branches 16 times.
1
LBL1
YES
R6L ≠ 0
NO
Shift R2H 1 bit right
Rotate R2L 1 bit right
Rotate R3H 1 bit right
Rotate R3L 1 bit right
The value of the result areas is divided by 2 to obtain the square root.
RTS

7.19.8 Program List

```
*** H8/300 ASSEMBLER              VER 1.0B **   08/18/92 10:23:40
PROGRAM NAME =

    1                                ;*********************************************************************
    2                                ;*
    3                                ;*   00 - NAME               :32 BIT SQUARE ROOT (SQRT)
    4                                ;*
    5                                ;*********************************************************************
    6                                ;*
    7                                ;*   ENTRY           :R4,R5  (32 BIT BINARY)
    8                                ;*
    9                                ;*   RETURN          :R3     (SQUARE ROOT)
   10                                ;*
   11                                ;*********************************************************************
   12                                ;
   13 SQRT_cod C 0000                     .SECTION        SQRT_code,CODE,ALIGN=2
   14                                     .EXPORT         SQRT
   15                                ;
   16 SQRT_cod C      00000000       SQRT .EQU    $               ;Entry point
   17 SQRT_cod C 0000 FE10                MOV.B   #D'16,R6L       ;Set shift counter
   18 SQRT_cod C 0002 79000000            MOV.W   #H'0000,R0      ;Clear R0
   19 SQRT_cod C 0006 0D01                MOV.W   R0,R1           ;Clear R1
   20 SQRT_cod C 0008 0D02                MOV.W   R0,R2           ;Clear R2
   21 SQRT_cod C 000A 0D03                MOV.W   R0,R3           ;Clear R3
   22 SQRT_cod C 000C                LBL1
   23 SQRT_cod C 000C F602                MOV.B   #H'02,R6H
   24 SQRT_cod C 000E                LBL2
   25 SQRT_cod C 000E 100D                SHLL.B  R5L             ;Shift 32 bit binary 1 bit left
   26 SQRT_cod C 0010 1205                ROTXL.B R5H
   27 SQRT_cod C 0012 120C                ROTXL.B R4L
   28 SQRT_cod C 0014 1204                ROTXL.B R4H
   29 SQRT_cod C 0016 1209                ROTXL.B R1L
   30 SQRT_cod C 0018 1201                ROTXL.B R1H
   31 SQRT_cod C 001A 1208                ROTXL.B R0L
   32 SQRT_cod C 001C 1200                ROTXL.B R0H
   33 SQRT_cod C 001E 1A06                DEC.B   R6H             ;Decrement R6H
   34 SQRT_cod C 0020 46EC                BNE     LBL2            ;Branch if Z=0
   35 SQRT_cod C 0022 0401                ORC.B   #H'01,CCR       ;Set C flag of CCR
   36 SQRT_cod C 0024 120B                ROTXL.B R3L             ;Rotate square root
   37 SQRT_cod C 0026 1203                ROTXL.B R3H
   38 SQRT_cod C 0028 120A                ROTXL.B R2L
   39 SQRT_cod C 002A 1202                ROTXL.B R2H
   40 SQRT_cod C 002C 1931                SUB.W   R3,R1           ;R1  - R3      -> R1
   41 SQRT_cod C 002E 1EA8                SUBX.B  R2L,R0L         ;R0L - R2L - C -> R0L
   42 SQRT_cod C 0030 1E20                SUBX.B  R2H,R0H         ;R0H - R2H - C -> R0H
   43 SQRT_cod C 0032 4412                BCC     LBL3            ;Branch if C=0
   44 SQRT_cod C 0034 0931                ADD.W   R3,R1           ;R1  + R3      -> R1
   45 SQRT_cod C 0036 0EA8                ADDX.B  R2L,R0L         ;R0L + R2L + C -> R0L
   46 SQRT_cod C 0038 0E20                ADDX.B  R2H,R0H         ;R0H + R2H + C -> R0H
   47 SQRT_cod C 003A 0401                ORC.B   #H'01,CCR       ;Bit set C flag of CCR
   48 SQRT_cod C 003C BB00                SUBX.B  #H'00,R3L       ;R3L - #H'00 - C -> R3L
   49 SQRT_cod C 003E B300                SUBX.B  #H'00,R3H       ;R3H - #H'00 - C -> R3H
   50 SQRT_cod C 0040 BA00                SUBX.B  #H'00,R2L       ;R2L - #H'00 - C -> R2L
   51 SQRT_cod C 0042 B200                SUBX.B  #H'00,R2H       ;R2H - #H'00 - C -> R2H
   52 SQRT_cod C 0044 400A                BRA     LBL4            ;Branch always
   53 SQRT_cod C 0046                LBL3
   54 SQRT_cod C 0046 0401                ORC.B   #H'01,CCR       ;Bit set C flag of CCR
   55 SQRT_cod C 0048 9B00                ADDX.B  #H'00,R3L       ;R3L + #H'00 + C -> R3L
   56 SQRT_cod C 004A 9300                ADDX.B  #H'00,R3H       ;R3H + #H'00 + C -> R3H
   57 SQRT_cod C 004C 9A00                ADDX.B  #H'00,R2L       ;R2L + #H'00 + C -> R2L
   58 SQRT_cod C 004E 9200                ADDX.B  #H'00,R2H       ;R2H + #H'00 + C -> R2H
   59 SQRT_cod C 0050                LBL4
   60 SQRT_cod C 0050 1A0E                DEC.B   R6L             ;Decrement shift counter
   61 SQRT_cod C 0052 46B8                BNE     LBL1            ;Branch if Z=0
   62 SQRT_cod C 0054 1102                SHLR.B  R2H
   63 SQRT_cod C 0056 130A                ROTXR.B R2L
```

```
 64 SQRT_cod C 0058 1303                    ROTXR.B R3H             ;Rotate square root
 65 SQRT_cod C 005A 130B                    ROTXR.B R3L
 66 SQRT_cod C 005C 5470                    RTS
 67                                    ;
 68                                         .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

Section 8 DECIMAL ↔ HEXADECIMAL CHANGE

8.1 Change a 2-Byte Hexadecimal Number to a 5-Character BCD Number

MCU: H8/300 Series
H8/300L Series

Label name: HEX

8.1.1 Function

1. The software HEX changes a 2-byte hexadecimal number (placed in a general-purpose register) to a 5-character BCD (binary-coded decimal) number and places the result of change in general-purpose registers.
2. All arguments used with the software HEX are represented in unsigned integers.
3. All data is manipulated on general-purpose registers.

8.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	2-byte hexadecimal number	R0	2
Output	5-character BCD number (upper 1 character)	R2L	1
	5-character BCD number (lower 4 characters)	R3	2

8.1.3 Internal Register and Flag Changes

R0	R1	R2H	R2L	R3	R4	R5	R6	R7
×	•	×	↕	↕	•	•	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

8.1.4 Specifications

Program memory (bytes)
30
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
368
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

8.1.5 Description

1. Details of functions
   a. The following arguments are used with the software HEX:
      R0: Contains a 2-byte hexadecimal number as an input argument.
      R2L: Contains the upper 1 character (1 byte) of a 5-character BCD number as an output argument.
      R3: Contains the lower 4 characters (2 bytes) of the 5-character BCD number as an output argument.
      Figure 8.1 shows the formats of the input and output arguments.

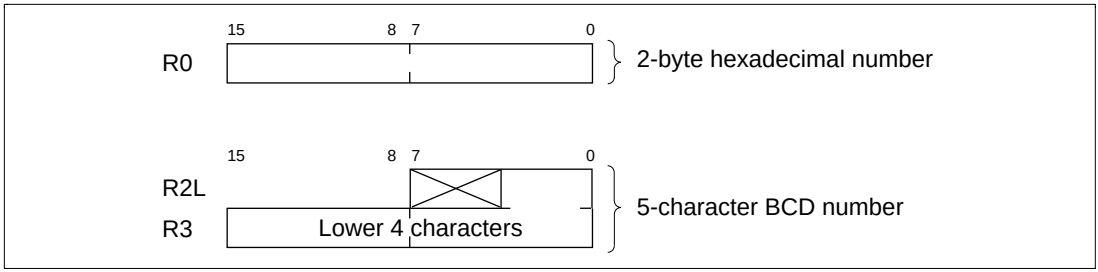


Figure 8.1 Example of Software FILL Execution

b. Figure 8.2 shows an example of the software HEX being executed. When the input argument is set as shown in (1), the 5-character BCD number is placed in R2L and R3 as shown in (2).

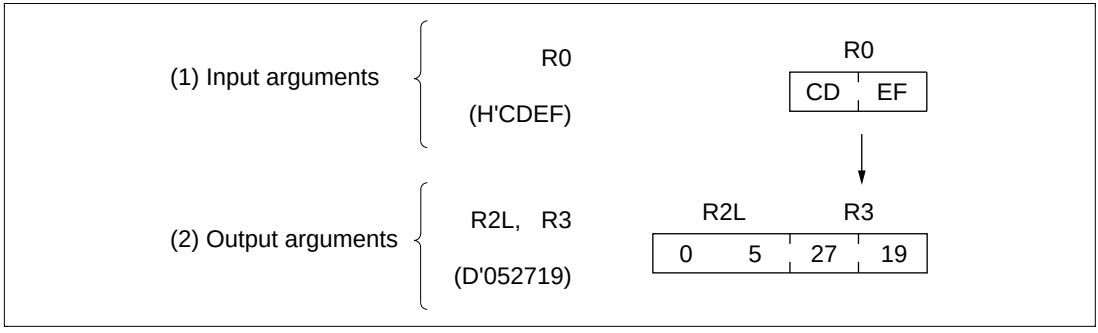


Figure 8.2 Example of Software HEX Execution

2. Notes on usage

When upper bits are not used (see figure 8.3), set 0's in them; otherwise, no correct result can be obtained because computation is made on numbers including indeterminate data placed in the upper bits.

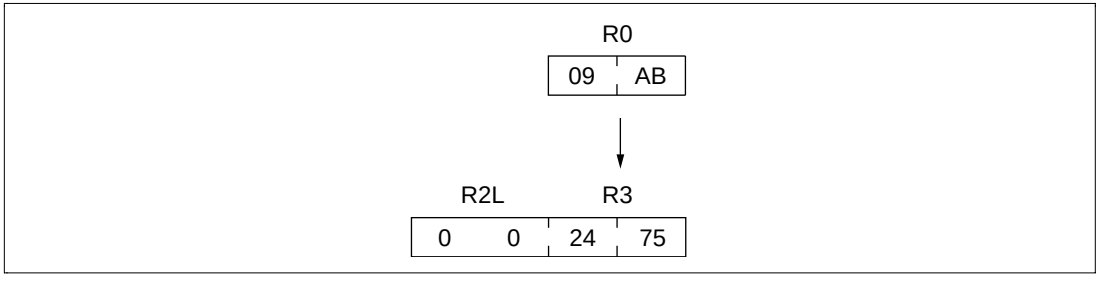


Figure 8.3 Examples of Operation with Upper Bits Unused

3. Data memory

The software HEX does not use the data memory.

4. Example of use

Set a 2-byte hexadecimal number in R0 and call the software HEX as a subroutine.

```
WORK1     . RES. W   1          ·····  Reserves a data memory area in which the
                                       user program places a 2-byte hexadecimal
                                       number.
WORK2     . RES. B   3          ·····  Reserves a data memory area in which the
                                       user program places a 5-character BCD
                                       number (3 bytes).
            :
          MOV. W   @WORK1, R0   ·····  Places in the input argument the 2-byte
                                       hexadecimal number set by the user
                                       program.
          JSR      @HEX         ·····  Calls the software HEX as a subroutine.
          MOV. B   R2L,@WORK2
          MOV. B   R3H,@WORK2+1 ·····  Places the 5-character BCD number (set in
          MOV. B   R3L,@WORK2+2        the output argument) in the data memory
                                       area of the user program.
            :
```

5. Operation

a. A 4-bit binary number "B3B2B1B0" is represented by equations 1 and 2 below:

$$B_3\ B_2\ B_1\ B_0 = B_3 \times 2^3 + B_2 \times 2^2 + B_1 \times 2^1 + B_0 \times 2^0 \quad \cdots\cdots\text{(equation 1)}$$

$$= ((B_3 \times 2 + B_2) \times 2 + B_1) \times 2 + B_0 \quad \cdots\cdots\text{(equation 2)}$$

α, β, γ

Figure 8.4 4-bit Binary Number "$B_3B_2B_1B_0$"

b. First, equation 2 is used to compute $\alpha = B_3 \times 2 + B_2$ (see figure 8.4) by executing an add instruction (the ADD.B instruction) and decimal correction (the DAA instruction). Next, a series of arithmetic operations such as $\beta = \alpha \times 2 + B_1$ and $\gamma = \beta \times 2 + B_0$ are performed to find a 5-character BCD number as the result.

c. The software HEX uses R0 (input) and R2L and R3 (outputs) to compute $\alpha = B_3 \times 2 + B_2$.

(i) R2H is used as the counter that shifts R0 (containing the input argument) bit by bit. D'16 is set in R2H for a total of 16 shifts.

(ii) R0 (containing the 2-byte hexadecimal number) is shifted 1 bit to the left, and the most significant bit is placed in the C flag.

(iii) R2L and R3 (containing the 5-character BCD number) are processed in ascending order, as follows:

$R3L + R3L + C \rightarrow R3L$ Decimal correction of R3L

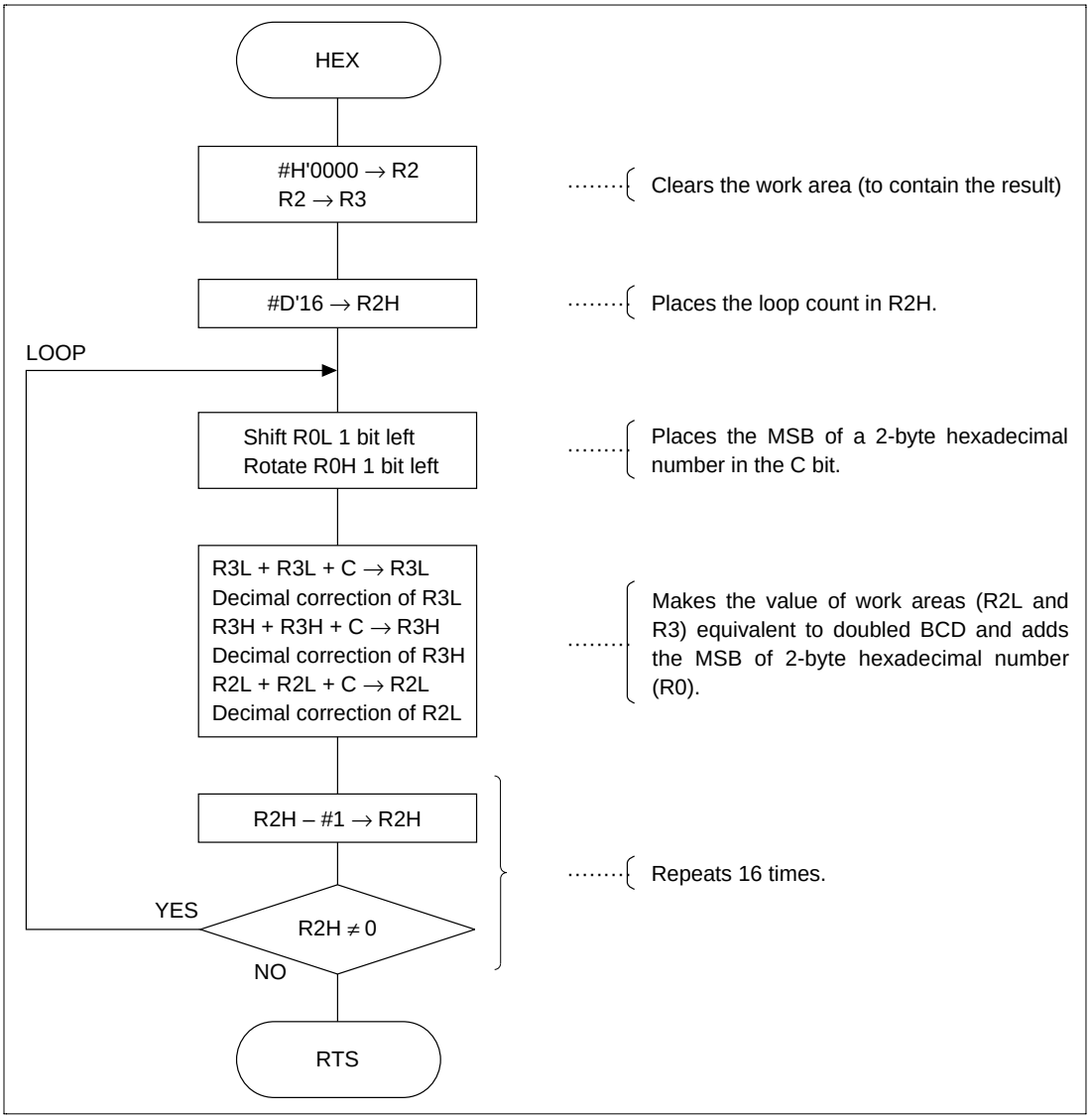
$R3H + R3H + C \rightarrow R3H$ Decimal correction of R3H

$R2L + R2L + C \rightarrow R2L$ Decimal correction of R2L

Thus, $\alpha = B_3 \times 2 + B_2$ has been computed.

(iv) In the software HEX, R2H is decremented each time the process (ii) to (iii) is performed. This processing continues until R2H reaches "0".

8.1.6 Flowchart



8.1.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:24:23
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :CHANGE 2 BYTE HEXADECIMAL
    4                                   ;*                            TO BCD   (HEX)
    5                                   ;*
    6                                   ;*********************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0     (HEXADECIMAL)
    9                                   ;*
   10                                   ;*   RETURNS         :R2L    (UPPER 1 CHARACTER (BY BCD))
   11                                   ;*                     R3     (LOWER 4 CHARACTER (BY BCD))
   12                                   ;*
   13                                   ;*********************************************************************
   14                                   ;
   15 HEX_code C 0000                        .SECTION        HEX_code,CODE,ALIGN=2
   16                                        .EXPORT         HEX
   17                                   ;
   18 HEX_code C      00000000          HEX  .EQU    $               ;Entry point
   19 HEX_code C 0000 79020000               MOV.W   #H'0000,R2      ;Clear R2
   20 HEX_code C 0004 0D23                   MOV.W   R2,R3           ;Clear R3
   21 HEX_code C 0006 F210                   MOV.B   #D'16,R2H       ;Set bit counter
   22 HEX_code C 0008                   LOOP
   23 HEX_code C 0008 1008                   SHLL.B  R0L             ;Shift hexadecimal 1 bit left
   24 HEX_code C 000A 1200                   ROTXL.B R0H
   25                                   ;
   26 HEX_code C 000C 0EBB                   ADDX.B  R3L,R3L         ;R3L + R3L     -> R3L
   27 HEX_code C 000E 0F0B                   DAA     R3L             ;Decimal adjust R3L
   28 HEX_code C 0010 0E33                   ADDX.B  R3H,R3H         ;R3H + R3H + C -> R3H
   29 HEX_code C 0012 0F03                   DAA     R3H             ;Decimal adjust R3H
   30 HEX_code C 0014 0EAA                   ADDX.B  R2L,R2L         ;R2L + R2L + C -> R2L
   31 HEX_code C 0016 0F0A                   DAA     R2L             ;Decimal adjust R2L
   32                                   ;
   33 HEX_code C 0018 1A02                   DEC.B   R2H             ;Decrement R2H
   34 HEX_code C 001A 46EC                   BNE     LOOP            ;Branch Z=0
   35 HEX_code C 001C 5470                   RTS
   36                                   ;
   37                                        .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

8.2 Change a 5-Character BCD Number to a 2-Byte Hexadecimal Number

MCU: H8/300 Series
H8/300L Series

Label name: BCD

8.2.1 Function

1. The software BCD changes a 5-character BCD (binary-coded decimal) Number (3 bytes, placed in a general-purpose registers) to a 2-byte hexadecimal number and places the result of change in a general-purpose register.
2. All data is manipulated on general-purpose registers.
3. The 5-character BCD number can be up to H'65535.

8.2.2 Arguments

Description		Memory area	Data length (bytes)
Input	5-character BCD number (upper 1 character)	R0L	1
	5-character BCD number (lower 4 characters)	R1	2
Output	2-byte hexadecimal number	R2	2

8.2.3 Internal Register and Flag Changes

R0H	R0L	R1	R2	R3	R4	R5H	R5L	R6	R7
×	•	•	↕	×	•	•	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

8.2.4 Specifications

Program memory (bytes)
64
Data memory (bytes)
0
Stack (bytes)
2
Clock cycle count
210
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

8.2.5 Description

1. Details of functions
   a. The following arguments are used with the software BCD:
      R0L: Contains the upper 1 character (1 byte) of a 5-character BCD number as an input argument.
      R1: Contains the lower 4 characters (2 bytes) of the 5-character BCD number as an input argument.
      R2: Contains a 2-byte hexadecimal number as an output argument.
      Figure 8.5 shows the formats of the input and output arguments.

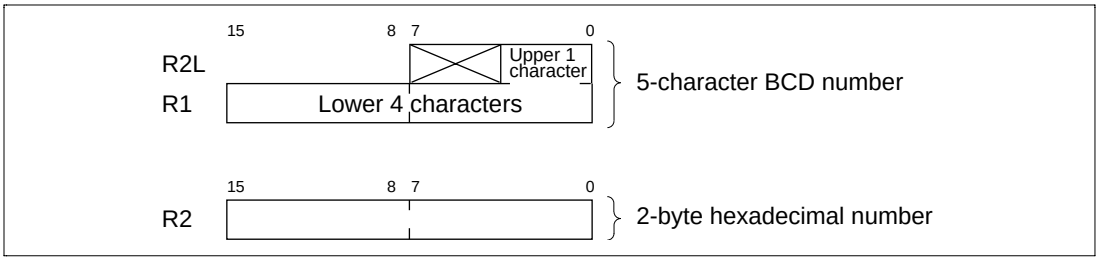


Figure 8.5 Example of Software MOVE1 Execution

b. Figure 8.6 shows an example of the software BCD being executed. When the input argument is set as shown in (1), the 2-byte hexadecimal number is placed in R2 as shown in (2).

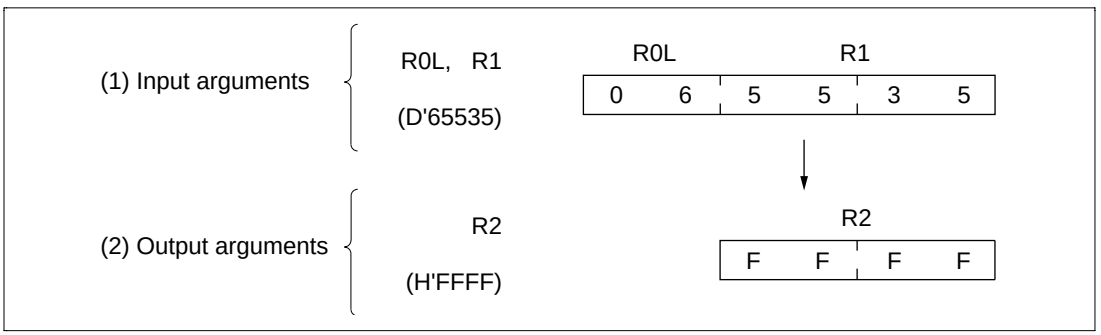


Figure 8.6 Example of Software BCD Execution

2. Notes on usage
   a. The values of bits 4 through 7 of R0L (containing the upper 1 character of the 5-character BCD number) remain unchanged. They are cleared to "0" after execution of the software BCD.
   b. The 5-character BCD number can be up to H'65535.
   c. When upper bits are not used, set 0's in them; otherwise, no correct result can be obtained because computation is made on numbers including indeterminate data placed in the upper bits.
3. Data memory

   The software BCD does not use the data memory.
4. Example of use

   Set a 5-character BCD number in the input arguments and call the software BCD as a subroutine.

```
WORK1       . RES. B    3          ·····  Reserves a data memory area in which the
                                          user program places a 5-character BCD
                                          number (3 bytes).

WORK2       . RES. B    2          ·····  Reserves a data memory area in which the
                                          user program places a 2-byte hexadecimal
                                          number.
            :
            MOV. B  @WORK1,   R0L
            MOV. B  @WORK1+1, R1H  ·····  Places in the input argument the 5-
            MOV. B  @WORK1+2, R1L         character BCD number set by the user
                                          program.
          | JSR       @BCD    |    ·····  Calls the software BCD as a subroutine.
            MOV. B  @WORK2,   R2H
            MOV. B  @WORK2+1, R2L  ·····  Places the 2-byte hexadecimal number
                                          (set in the output argument) in the data
                                          memory area of the user program.
            :
```

5. Operation
   a. The software BCD consists of two processes:
      (i) Popping up the 5-character BCD number character by character
      (ii) Changing the popped-up data to a hexadecimal number on a 4-bit basis.
   b. Figure 8.7 shows the method of computing a 1-character (4-bit) number.

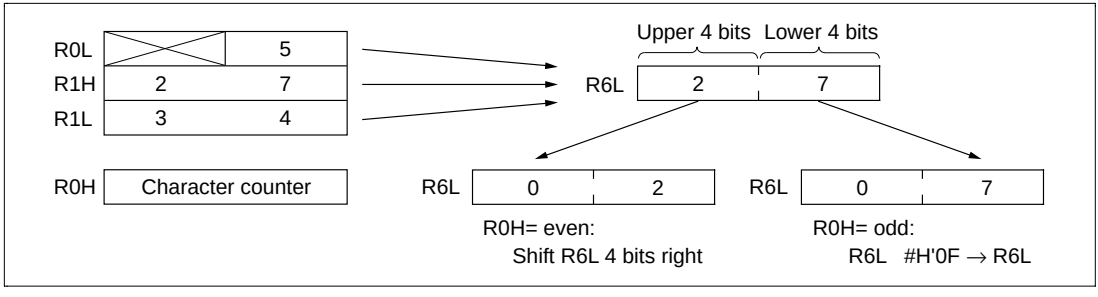


Figure 8.7 Method of Dividing 1-Byte Register Data by 2

(i) H'04 is placed for computation of the 5 characters.

(ii) The 5-character BCD number (R0L, R1H, R1L) is transferred to R6L starting with the most significant byte. Then the upper or lower 4 bits are selected.

(iii) R0H is decremented each time the process (ii) is performed.

(iv) When the process (ii) is performed, the software checks whether the counter (R0H) is even or odd.

- When R0H is odd, R6L is ANDed with H'0F to pop up the lower 4 bits.

- When R0H is even, R6L is shifted 4 bits to the right to pop up the upper 4 bits.

c. The BCD number is changed to a hexadecimal number in the following steps:

(i) A 4-character BCD "$D_3D_2D_1D_0$" is represented by equations 1 and 2 below:

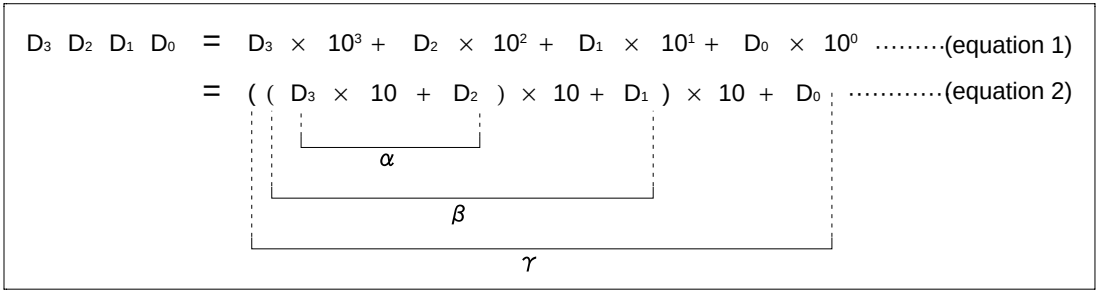


Figure 8.8 4-character BCD Number "$D_3D_2D_1D_0$"

(ii) First, equation 2 is used to compute $\alpha = D_3 \times 10 + D_2$ (see figure 8.8). Next, a series of arithmetic operations such as $\beta = \alpha \times 10 + D_1$ and $\gamma = \beta \times 10 + D_0$ are performed to find a hexadecimal number as the result.

(iii) Equations 3 and 4 are used to compute $D_3 \times 10$:

$$D_3 \times 10 = D_3 \times (2 + 8) \text{..... (equation 3)}$$

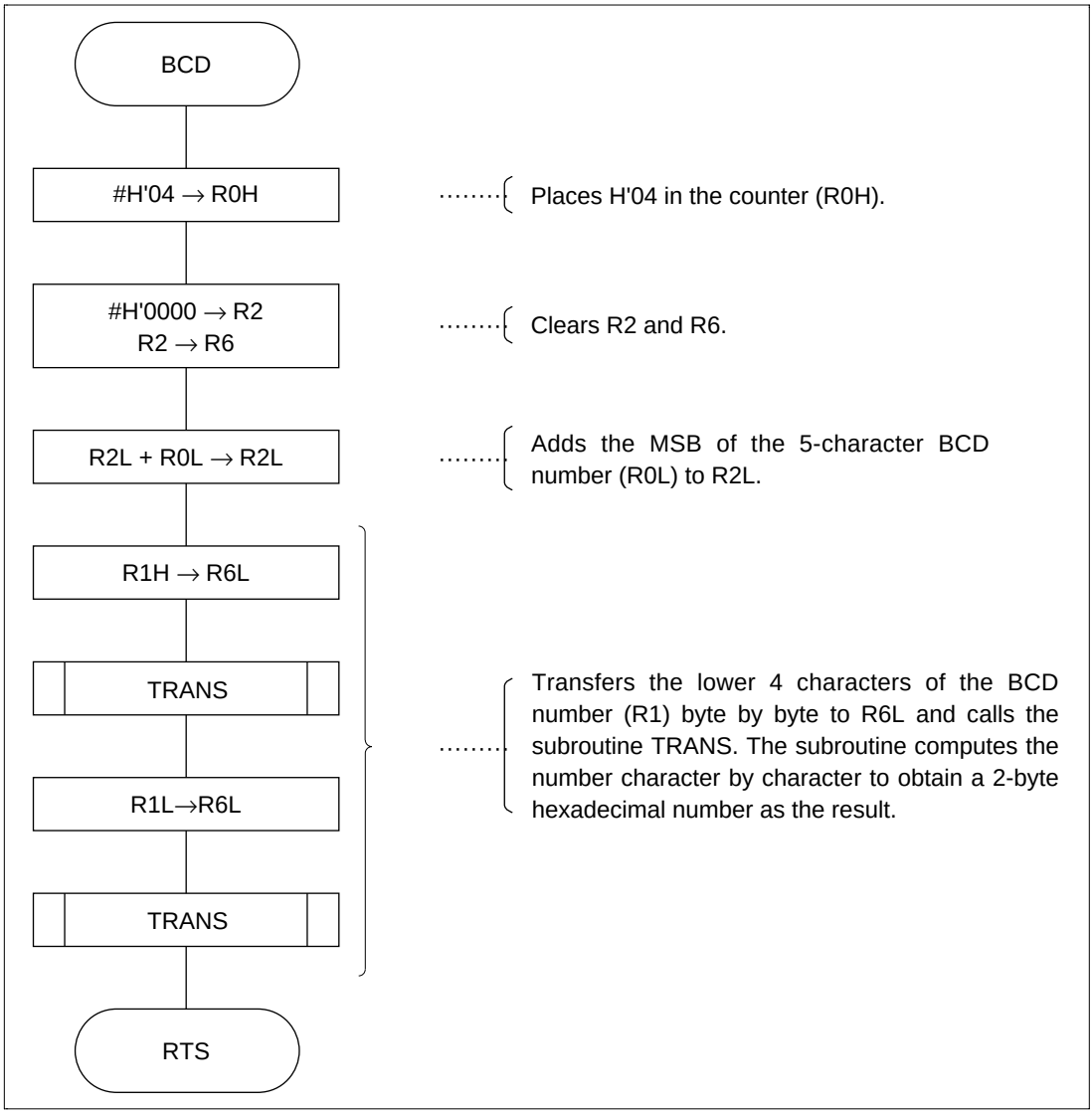
$$= D_3 \times 2 \times (1 + 2^2) \text{..... (equation 4)}$$

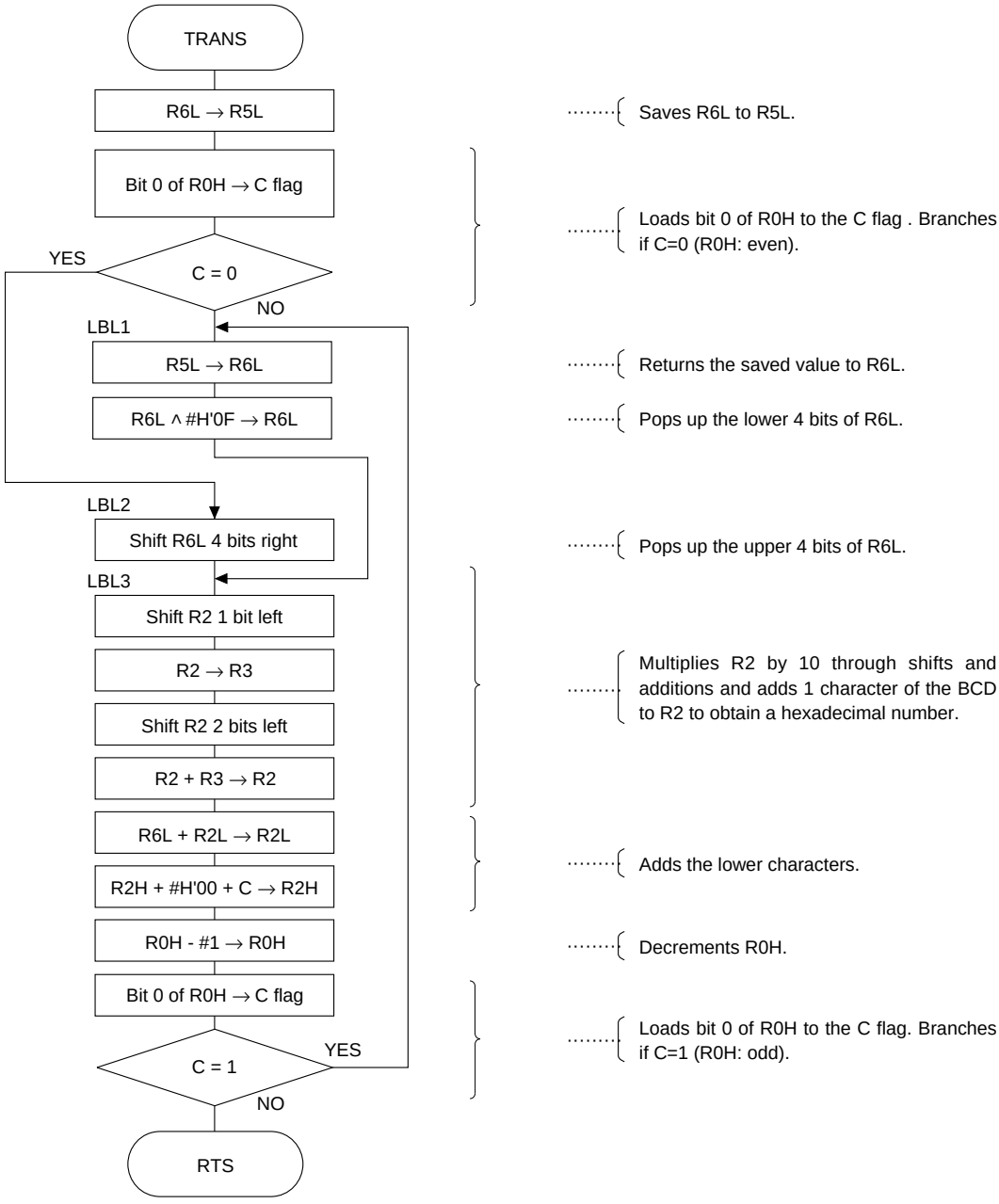
(iv) The software HEX uses R2 and R3 to compute equation 4 by taking the following steps:

1. Places D_3 in R2 and shifts it 1 bit to the left.
2. Transfers R2 to R3 and shifts it 1 bit to the left.
3. Adds R3 to R2.

d. The hexadecimal form of the 2-byte BCD number can be obtained by repeating the process b. to c. five times.

8.2.6 Flowchart




TRANS
R6L → R5L
Bit 0 of R0H → C flag
C = 0
YES
NO
LBL1
R5L → R6L
R6L ∧ #H'0F → R6L
LBL2
Shift R6L 4 bits right
LBL3
Shift R2 1 bit left
R2 → R3
Shift R2 2 bits left
R2 + R3 → R2
R6L + R2L → R2L
R2H + #H'00 + C → R2H
R0H - #1 → R0H
Bit 0 of R0H → C flag
C = 1
YES
NO
RTS
Saves R6L to R5L.
Loads bit 0 of R0H to the C flag . Branches if C=0 (R0H: even).
Returns the saved value to R6L.
Pops up the lower 4 bits of R6L.
Pops up the upper 4 bits of R6L.
Multiplies R2 by 10 through shifts and additions and adds 1 character of the BCD to R2 to obtain a hexadecimal number.
Adds the lower characters.
Decrements R0H.
Loads bit 0 of R0H to the C flag. Branches if C=1 (R0H: odd).

8.2.7 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/22/92 11:09:49
PROGRAM NAME =

    1                                   ;*****************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :CHANGE 5 CHARACTER
    4                                   ;*                            TO 2 BYTE HEXADECIMAL  (BCD)
    5                                   ;*
    6                                   ;*****************************************************************
    7                                   ;*
    8                                   ;*   ENTRY           :R0L    (UPPER 1 CHAR (BY BCD))
    9                                   ;*                    R1     (LOWER 4 CHAR (BY BCD))
   10                                   ;*
   11                                   ;*   RETURN          :R2     (2 BYTE HEXADECIMAL)
   12                                   ;*
   13                                   ;*****************************************************************
   14                                   ;
   15 BCD_code C 0000                        .SECTION        BCD_code,CODE,ALIGN=2
   16                                        .EXPORT         BCD
   17                                   ;
   18 BCD_code C      00000000          BCD  .EQU    $               ;Entry point
   19 BCD_code C 0000 F004                   MOV.B   #H'04,R0H       ;Set bit counter
   20 BCD_code C 0002 79020000               MOV.W   #H'0000,R2      ;Clear R2
   21 BCD_code C 0006 0D26                   MOV.W   R2,R6           ;Clear R6
   22                                   ;
   23 BCD_code C 0008 088A                   ADD.B   R0L,R2L         ;R2L + R0L -> R2L
   24 BCD_code C 000A 0C1E                   MOV.B   R1H,R6L         ;R1H -> R6L
   25 BCD_code C 000C 5506                   BSR     TRANS
   26 BCD_code C 000E 0C9E                   MOV.B   R1L,R6L         ;R1L -> R6L
   27 BCD_code C 0010 5502                   BSR     TRANS
   28 BCD_code C 0012 5470                   RTS
   29                                   ;
   30                                   ;---------------------------------------------------------------
   31                                   ;
   32 BCD_code C 0014                   TRANS                               ;Change BCD to hexadecimal
   33 BCD_code C 0014 0CED                   MOV.B   R6L,R5L         ;R6L -> R5L
   34 BCD_code C 0016 7700                   BLD     #0,R0H          ;load bit 0 of R0H
   35 BCD_code C 0018 4406                   BCC     LBL2            ;Branch if C=0
   36 BCD_code C 001A                   LBL1
   37 BCD_code C 001A 0CDE                   MOV.B   R5L,R6L         ;R5l -> R6L
   38 BCD_code C 001C EE0F                   AND.B   #H'0F,R6L       ;Clear bit 7-4 of R6L
   39 BCD_code C 001E 4008                   BRA     LBL3            ;Branch always
   40 BCD_code C 0020                   LBL2
   41 BCD_code C 0020 110E                   SHLR.B  R6L             ;Shift R6L 4 bit left
   42 BCD_code C 0022 110E                   SHLR.B  R6L
   43 BCD_code C 0024 110E                   SHLR.B  R6L
   44 BCD_code C 0026 110E                   SHLR.B  R6L
   45 BCD_code C 0028                   LBL3
   46 BCD_code C 0028 100A                   SHLL.B  R2L             ;Shift Hexadecimal 1 bit left
   47 BCD_code C 002A 1202                   ROTXL.B R2H
   48 BCD_code C 002C 0D23                   MOV.W   R2,R3           ;R2 -> R3
   49 BCD_code C 002E 100A                   SHLL.B  R2L             ;Shift Hexadecimal 2 bit left
   50 BCD_code C 0030 1202                   ROTXL.B R2H
   51 BCD_code C 0032 100A                   SHLL.B  R2L
   52 BCD_code C 0034 1202                   ROTXL.B R2H
   53 BCD_code C 0036 0932                   ADD.W   R3,R2           ;R3 + R2 -> R2
   54 BCD_code C 0038 08EA                   ADD.B   R6L,R2L
   55 BCD_code C 003A 9200                      ADDX.B  #0,R2H
   56 BCD_code C 003C 1A00                      DEC.B        R0H             ;Decrement bit counter
   57 BCD_code C 003E 7700                   BLD     #0,R0H          ;load bit 0 of R0H
   58 BCD_code C 0040 45D8                   BCS     LBL1            ;Branch if C=!
   59 BCD_code C 0042 5470                   RTS
   60                                   ;
   61                                        .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

Section 9 Sorting

9.1 Set Constants

MCU: H8/300 Series
H8/300L Series

Label name: SORT

9.1.1 Function

1. The software SORT sorts the data placed on the data memory, byte by byte, in descending order.
2. The number of bytes to be sorted can be up to 255.
3. Data to be sorted is represented as unsigned integers.

9.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	Number of bytes of data to be sorted	R0L	1
	Start address of the data to be sorted	R4	2
Output	—	—	—

9.1.3 Internal Register and Flag Changes

R0	R1	R2	R3	R4	R5	R6	R7
×	×	•	•	×	×	•	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

× : Unchanged
• : Indeterminate
↕ : Result

9.1.4 Specifications

Program memory (bytes)
34
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
789482
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

9.1.5 Note

The clock cycle count (789482) in the specifications is for sorting 255-byte data in descending order.

9.1.6 Description

1. Details of functions
   a. The following arguments are used with the software SORT:
      R0L: Contains the number of bytes of data to be sorted – 1 as an input argument.
      R4: Contains the start address of the data to be sorted (stored on RAM).

b. Figure 9.1 shows an example of the software SORT being executed. When the input argument is set as shown in (1), the data is sorted in descending order as shown in (2).

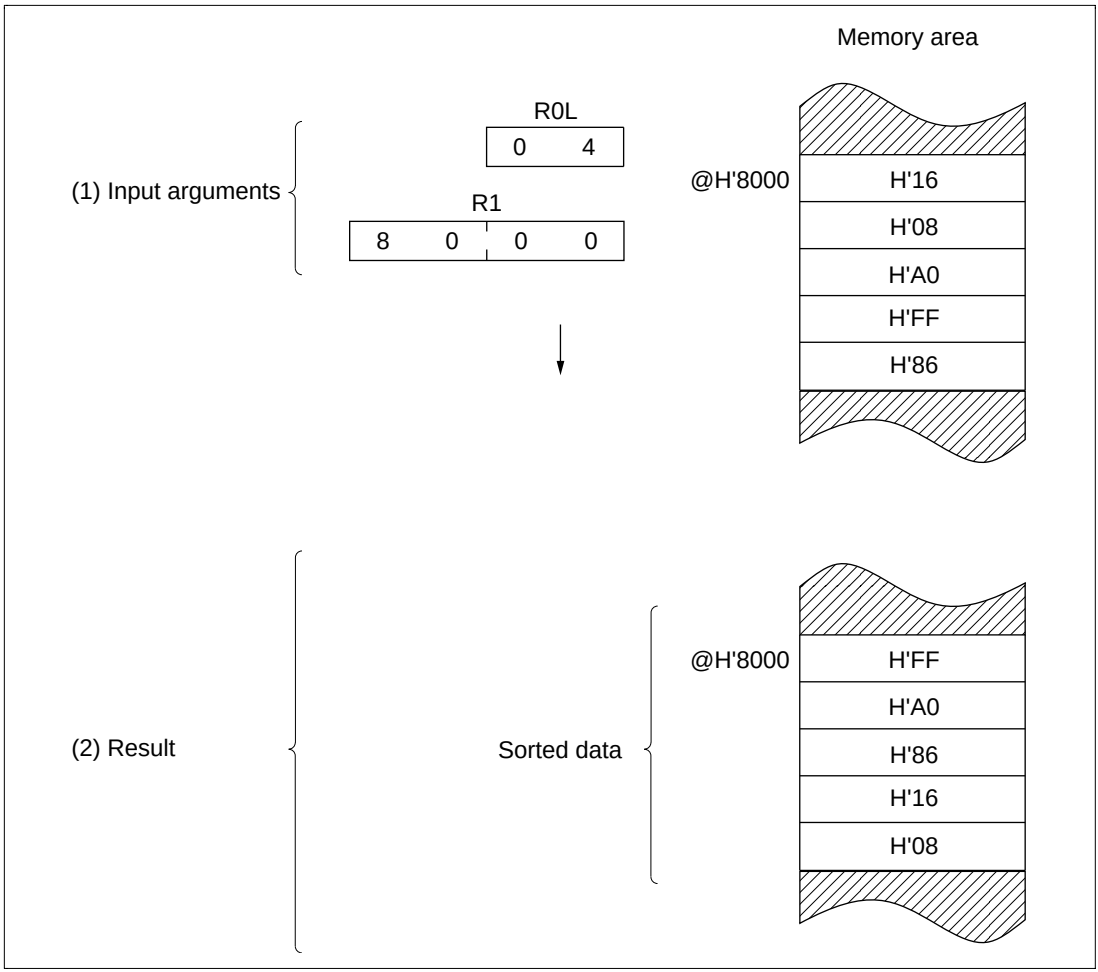


Figure 9.1 Example of Software SORT Execution

2. Notes on usage
   a. Do not set "0" in R0L; otherwise, the software SORT will not operate normally.
   b. R0L must contain the number of bytes of data to be sorted - 1.
3. Data memory
   The software SORT does not use the data memory.

4. Example of use

Set the input arguments in registers and call the software SORT as a subroutine.

```
WORK1    . RES. B    1          ·····  Reserves a data memory area in which the
                                        user program places the number of bytes
                                        of data to be sorted.
WORK2    . RES. B    255        ·····  Reserves a data memory area in which the
                                        user program places the data to be sorted.
           MOV. B    @WORK1, R0L ····  Places in the input argument the number of
                                        bytes of data to be sorted set by the user
           MOV. W    #WORK2 R4  ·····  Places in R4 the start address of the data
                                        to be sorted set by the user program.
           JSR       @SORT      ·····  Calls the software SORT as a subroutine.
```

5. Operation

a. Figure 9.2 shows an example of sorting where three pieces of data are sorted in descending order.

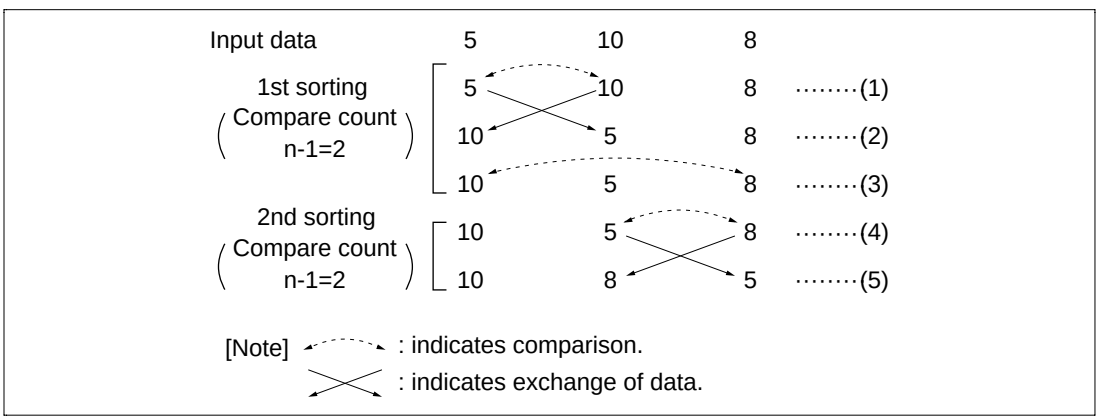


Figure 9.2 Example of Sorting

(i) The software searches the three pieces of data for the biggest number and sorts it at the extreme left. (See (1), (2) and (3) of figure 9.2.)

(ii) Next, the software identifies the greater of the second and last numbers as counted from the left and places it at the second place from the left. (See (4) and (5) of figure 9.2.)

b. Processing by programs

(i) R4 is used as the pointer for placing the biggest number. R5 is used as the pointer that indicates the address of the memory area containing the source number.

(ii) The comparand is placed in R1L.

(iii) The source number is placed in R1H.

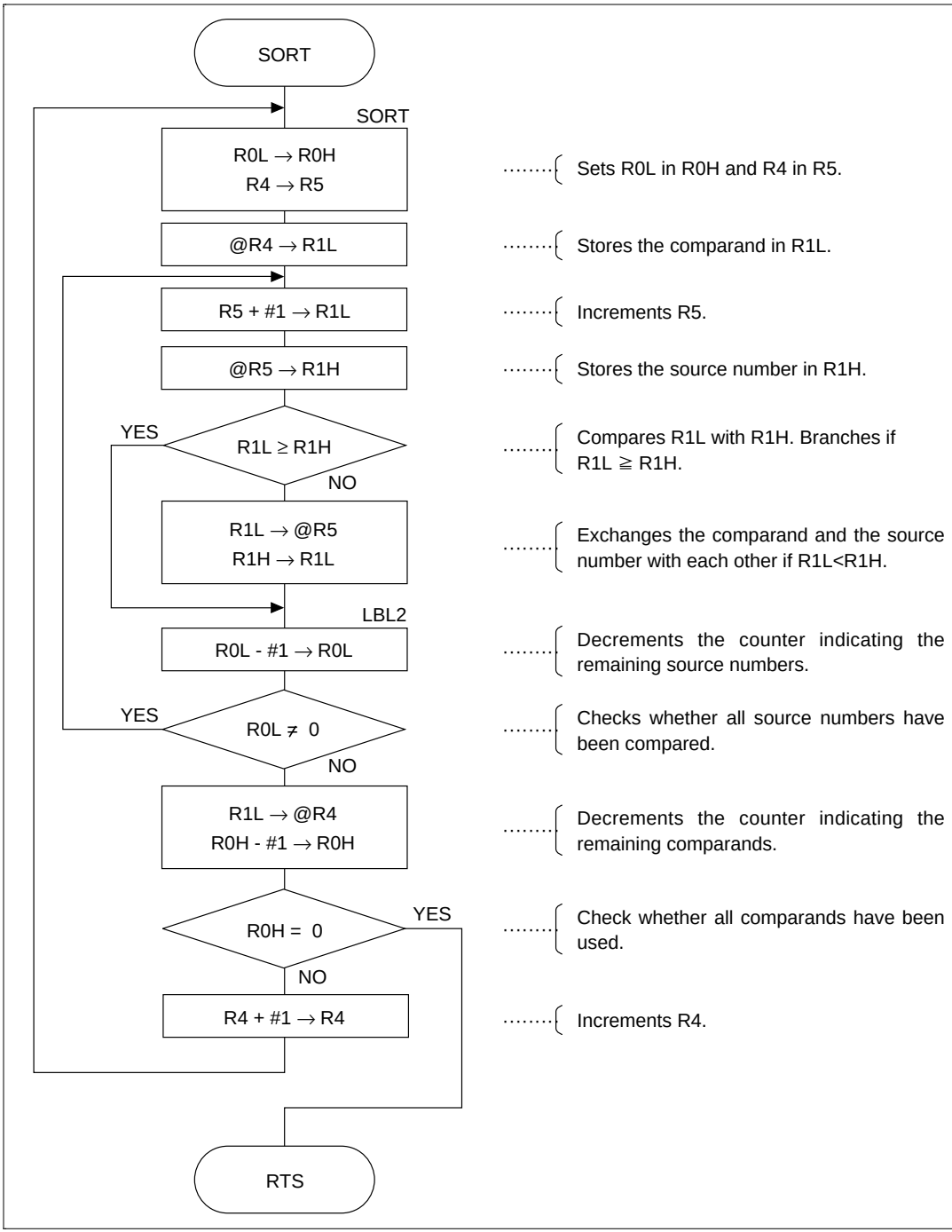
(iv) R1L and R1H are compared with each other. If the source number is greater than the comparand (R1H > R1L), the two numbers are exchanged.

(v) The process (iii) to (iv) is repeated until the counter R0L indicating the remaining source numbers reaches "0".

(vi) When R0L reaches "0", the data stored in @R4 is assumed the biggest of the data compared.

(vii)The R0H indicating the number of remaining comparands is decremented.

9.1.7 Flowchart



9.1.8 Program List

```
*** H8/300 ASSEMBLER               VER 1.0B **   08/18/92 10:26:21
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME               :SORTING (SORT)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0L    (BYTE NUMBER)
    8                                   ;*                    R4     (START ADDRESS OF DATA)
    9                                   ;*
   10                                   ;*   RETURN          :NOTHING
   11                                   ;*
   12                                   ;*********************************************************************
   13                                   ;
   14 SORT_cod C 0000                        .SECTION        SORT_code,CODE,ALIGN=2
   15                                        .EXPORT         SORT
   16                                   ;
   17 SORT_cod C      00000000           SORT .EQU    $               ;Entry point
   18 SORT_cod C 0000 0C80                   MOV.B   R0L,R0H         ;Set data counter
   19 SORT_cod C 0002 0D45                   MOV.W   R4,R5           ;R4 -> R5
   20 SORT_cod C 0004 6849                   MOV.B   @R4,R1L         ;@R4 -> data1
   21 SORT_cod C 0006                   LBL1
   22 SORT_cod C 0006 0B05                   ADDS.W  #1,R5           ;Increment address pointer1 (R5++)
   23 SORT_cod C 0008 6851                   MOV.B   @R5,R1H         ;@R5 -> data2
   24 SORT_cod C 000A 1C19                   CMP.B   R1H,R1L
   25 SORT_cod C 000C 4404                   BHS     LBL2            ;Branch if C=0
   26 SORT_cod C 000E 68D9                   MOV.B   R1L,@R5         ;Store data1 @R5
   27 SORT_cod C 0010 0C19                   MOV.B   R1H,R1L         ;data2 -> data1
   28 SORT_cod C 0012                   LBL2
   29 SORT_cod C 0012 1A00                   DEC.B   R0H             ;Decrement data counter
   30 SORT_cod C 0014 46F0                   BNE     LBL1            ;Branch if Z=0
   31 SORT_cod C 0016 68C9                   MOV.B   R1L,@R4         ;data1 -> @R4
   32 SORT_cod C 0018 1A08                   DEC.B   R0L             ;Decrement byte number
   33 SORT_cod C 001A 4704                   BEQ     EXIT            ;Branch Z=0
   34 SORT_cod C 001C 0B04                   ADDS.W  #1,R4           ;Increment address pointer2 (R4++)
   35 SORT_cod C 001E 40E0                   BRA     SORT            ;Branch always
   36 SORT_cod C 0020                   EXIT
   37 SORT_cod C 0020 5470                   RTS
   38                                   ;
   39                                        .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```

Section 10 ARRAY

10.1 2-Dimensional Array (ARRAY)

MCU H8/300 Series
H8/300L Series

Label name: ARRAY

10.1.1 Function

1. The software ARRAY retrieves data from a 2-dimensional array (hereafter called an "array") and sets its address and elements (x, y) of the array when the data matches.
2. Data to be processed by the software ARRAY are 1-byte unsigned integers.
3. The elements of an array are 1-byte unsigned integers.
4. An array can be set up in the range 255 bytes × 255 bytes.

10.1.2 Arguments

Description		Memory area	Data length (bytes)
Input	Data to be retrieved	R0L	1
	Start address of the array	R4	2
	Number of rows of the array	R2L	1
	Number of columns of the array	R3L	1
Output	Address of matched data	R4	2
	Array element x of matched data	R5H	1
	Array element y of matched data	R5L	1
	Presence of matched data	C flag (CCR)	

10.1.3 Internal Register and Flag Changes

R0H	R0L	R1	R2H	R2L	R3H	R3L	R4	R5H	R5L	R6	R7
×	•	•	×	×	×	×	↕	↕	↕	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	↕

× : Unchanged
• : Indeterminate
↕ : Result

10.1.4 Specifications

Program memory (bytes)
46
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
1986
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

10.1.5 Note

The clock cycle count (1986) in the specifications is for the example shown in figure 10.1.
If either element x or y is "0", the software terminates immediately and clears the C flag.

10.1.6 Description

1. Details of functions
   a. The following arguments are used with the software ARRAY:
      (i) Input arguments

R0L:	Data to be retrieved
R4:	Start address of the array
R2L:	Number of rows of the array (x)
R3L:	Number of columns of the array (y)

      (ii) Output arguments

R4:	Address of the matched data
R5H:	Array element x of the matched data
R5L:	Array element y of the matched data
C flag (CCR):	Indicates the state at the end of the software ARRAY.
C flag = 1:	Matched data is found on the array.
C flag = 0:	Matched data is not found on the array.

   b. Figure 10.1 shows an example of the software ARRAY being executed. When the input arguments are set as shown in (1), the software ARRAY references the array (16×16) in figure 10.2 and places the address of the matched data in R4, the array element x in R5H, and the array element y in R5L as shown in (2).

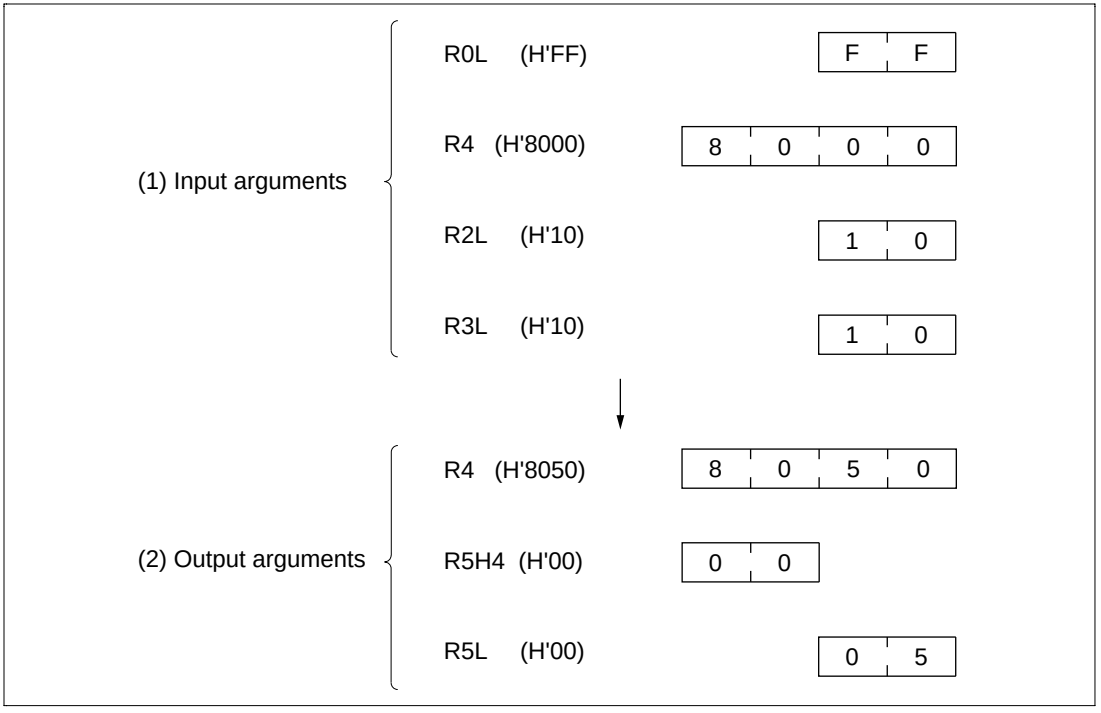


Figure 10.1 Example of Software ARRAY Execution

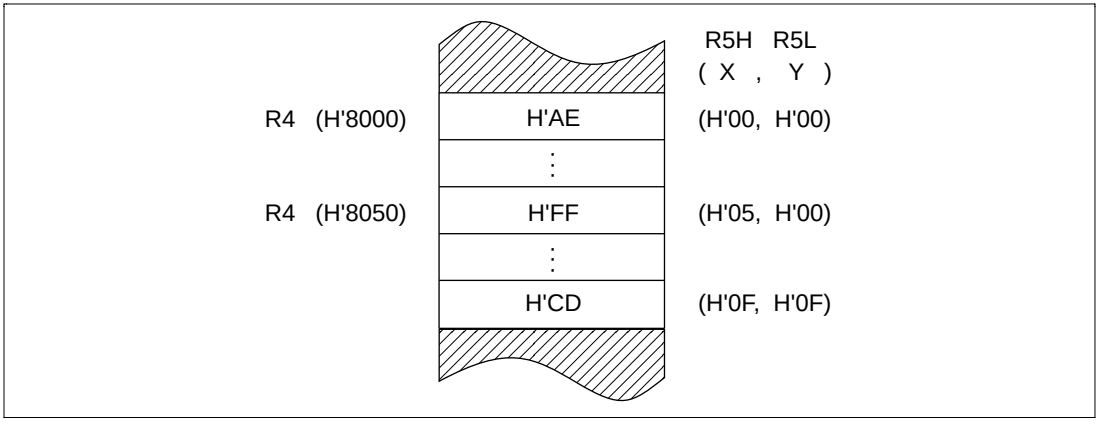


Figure 10.2 Array Space

c. Execution of the software ARRAY requires an array as shown in figure 10.3.

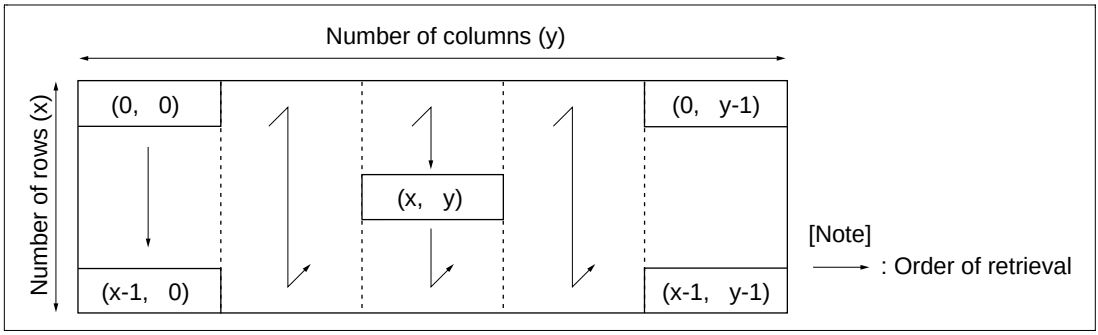


Figure 10.3 2-Dimensional Array

(i) The size of an array is identified by the number of rows (x) and the number of columns (y).

(ii) The elements of an array are represented by x (row) and y (column), which range from (0, 0) to (x – 1, y – 1).

(iii) The start address of an array is (0, 0), at which retrieval starts in the order as shown in figure 10.3.

2. Notes on usage

   Do not set "0" as x and y; otherwise, the software ARRAY do not start retrieval of data, clears the C flag, and terminates.

3. Data Memory

   The software ARRAY does not use the data memory.

4. Example of Use

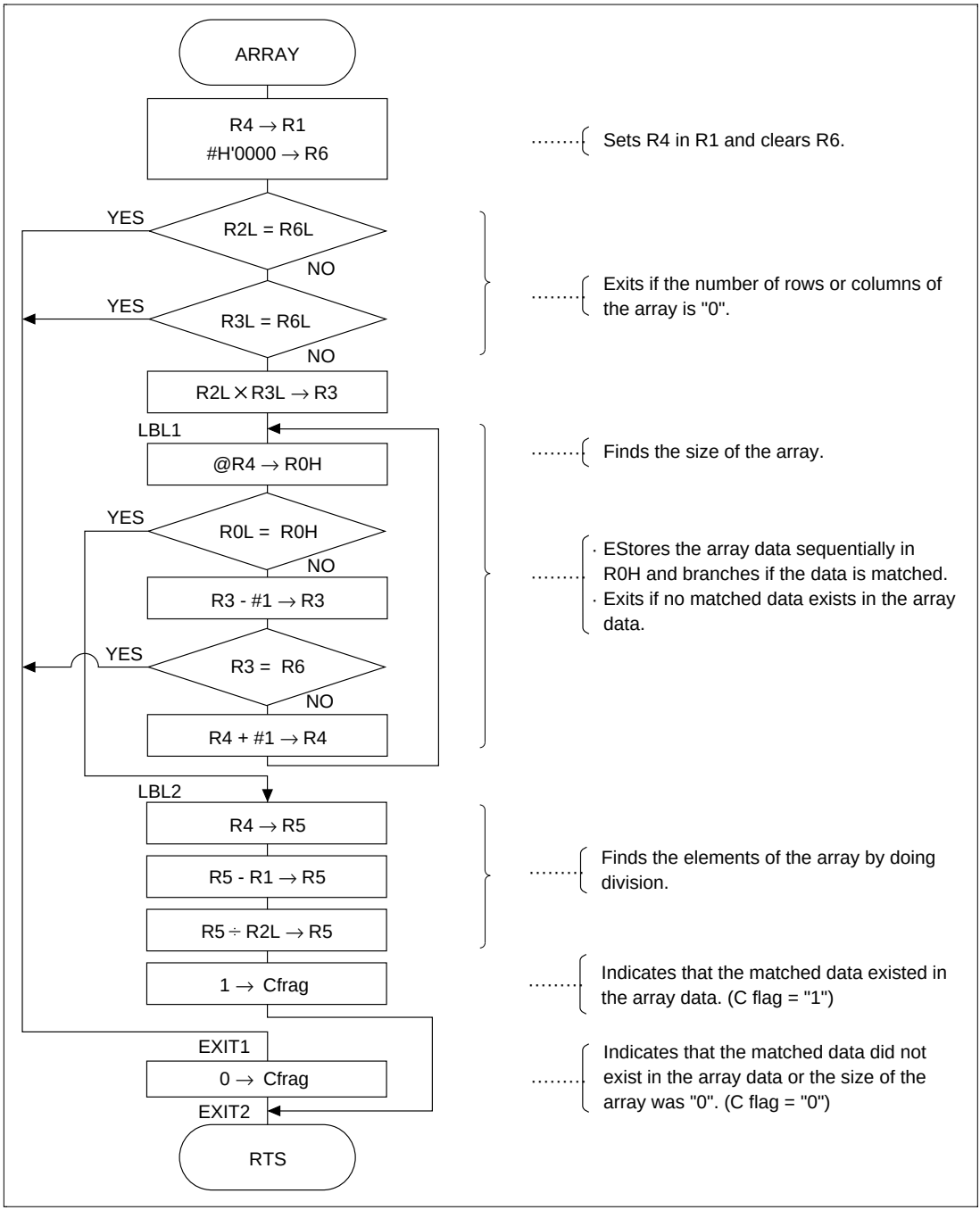
   Set the data to be retrieved and the start address, the number of rows and the number of columns of an array to be searched, and call the software ARRAY as a subroutine.

```
I-WORK1     .RES.W  1       Reserves a data memory area for the start address of the array.
I-WORK2     .RES.B  1       Reserves a data memory area for the number of rows of the array (x).
I-WORK3     .RES.B  1       Reserves a data memory area for the number of columns of the array (y).
I-WORK4     .RES.B  1       Reserves a data memory area for the data to be retrieved.
              :

O-WORK1     .RES.W  1       Reserves a data memory area for the address of the matched data.
O-WORK2     .RES.B  1       Reserves a data memory area for the element (x) of the array when the data is matched.
O-WORK3     .RES.B  1       Reserves a data memory area for the element (y) of the array when the data is matched.
              :

            MOV. B  @I_WORK4, R0L ..... Places the data to be retrieved.
            MOV. W  @I_WORK1, R4  ..... Places the start address of the array.
            MOV. B  @I_WORK2, R2H ..... Places the number of rows of the array (x).
            MOV. B  @I_WORK3, R2L ..... Places the number of columns of the array (y).
            JSR     @ARRAY        ..... Calls the software ARRAY as a subroutine.
            MOV. W  R4, @O_WORK1  ..... Stores the address of the matched data.
            MOV. B  R2H, @O_WORK2 ..... Stores the element of the array (x) when the data is matched.
            MOV. B  R2L, @O_WORK3 ..... Stores the element of the array (y) when the data is matched.
              :
```

10.1.7 Flowchart



10.1.8 Program List

```
*** H8/300 ASSEMBLER              VER 1.0B **   08/18/92 10:26:53
PROGRAM NAME =

    1                                   ;*********************************************************************
    2                                   ;*
    3                                   ;*   00 - NAME              :2-DIMENSIONAL ARRAY (ARRAY)
    4                                   ;*
    5                                   ;*********************************************************************
    6                                   ;*
    7                                   ;*   ENTRY           :R0L    (REFERENCE DATA)
    8                                   ;*                    R2L    (NUMBER OF COLUM [X])
    9                                   ;*                    R3L    (NUMBER OF ROW        [Y])
   10                                   ;*                    R4     (ARRAY START ADDR)
   11                                   ;*
   12                                   ;*   RETURNS         :R5H    (ARRAY ELEMENT OF COLUM [x])
   13                                   ;*                    R5L    (ARRAY ELEMENT OF LOW   [y])
   14                                   ;*                    R4     (MATCH DATA ADDR)
   15                                   ;*                    C flag OF CCR (C=1;TRUE , C=0;FALSE)
   16                                   ;*
   17                                   ;*********************************************************************
   18                                   ;
   19 ARRAY_co C 0000                        .SECTION        ARRAY_code,CODE,ALIGN=2
   20                                        .EXPORT         ARRAY
   21                                   ;
   22 ARRAY_co C      00000000          ARRAY        .EQU    $               ;Entry point
   23 ARRAY_co C 0000 0D41                   MOV.W   R4,R1
   24 ARRAY_co C 0002 79060000               MOV.W   #H'0000,R6      ;Clear R6
   25 ARRAY_co C 0006 1CAE                   CMP.B   R2L,R6L
   26 ARRAY_co C 0008 4720                   BEQ     EXIT1           ;Branch if Z=1 then exit
   27 ARRAY_co C 000A 1CBE                   CMP.B   R3L,R6L
   28 ARRAY_co C 000C 471C                   BEQ     EXIT1           ;Branch if Z=1 then exit
   29 ARRAY_co C 000E 50A3                   MULXU   R2L,R3          ;Get total number of array(R3)
   30 ARRAY_co C 0010                   LBL1
   31 ARRAY_co C 0010 6840                   MOV.B   @R4,R0H         ;Load array data
   32 ARRAY_co C 0012 1C80                   CMP.B   R0L,R0H
   33 ARRAY_co C 0014 470A                   BEQ     LBL2            ;Branch if data find
   34 ARRAY_co C 0016 1B03                   SUBS.W  #1,R3           ;Decrement R3
   35 ARRAY_co C 0018 1D36                   CMP.W   R3,R6
   36 ARRAY_co C 001A 4710                   BEQ     EXIT2           ;Branch if false
   37 ARRAY_co C 001C 0B04                   ADDS.W  #1,R4           ;Increment data pointer
   38 ARRAY_co C 001E 40F0                   BRA     LBL1            ;Branch always
   39 ARRAY_co C 0020                   LBL2
   40 ARRAY_co C 0020 0D45                   MOV.W   R4,R5
   41 ARRAY_co C 0022 1915                   SUB.W   R1,R5           ;Get count number of find data
   42 ARRAY_co C 0024 51A5                   DIVXU   R2L,R5          ;Get array element [x,y]
   43 ARRAY_co C 0026 0401                   ORC.B   #H'01,CCR       ;Set C flag of CCR
   44 ARRAY_co C 0028 4002                   BRA     EXIT2           ;Branch always
   45 ARRAY_co C 002A                   EXIT1
   46 ARRAY_co C 002A 06FE                   ANDC.B  #H'FE,CCR       ;Clear C flag of CCR
   47 ARRAY_co C 002C                   EXIT2
   48 ARRAY_co C 002C 5470                   RTS
   49                                   ;
   50                                        .END
  *****TOTAL ERRORS       0
  *****TOTAL WARNINGS     0
```