To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

M16C/26

Optimizing Code with Assembly Language

1.0 Abstract

The following article discusses assembly functions that can be called from C programs for the M16C/26 MCU. Two sample programs are provided to run on MSV30262 board for testing and evaluation. Each program outlines a particular method of assembly function call and also provides a typical C function call for comparison by relevant CPU cycle times.

2.0 Introduction

The Renesas M30262 is a 16-bit MCU based on the M16C/60 series CPU core. The MCU features include up to 64KB of Flash ROM, 2KB of RAM, and 4KB of virtual EEPROM. The peripheral set includes 10-bit A/D, UARTs, Timers, DMA, and GPIO.

Assembler functions are called from C programs using the name of the assembler functions in the same way that functions written in C would be. The first label in an assembly function must be preceded by an underscore "_". However, when calling the assembly function from the C program, the underscore is omitted. The calling C program must include a prototype declaration for the assembly function. List 1 is an example of calling assembly function asm_func and

List 2 shows the resulting compiled code. The general method for writing an assembly function is described in detail in Appendix A.1.

```
extern void int asm_func( void ); /* assembler function prototype declaration */
void main()
{
        …
        (omitted)
        …
        asm_func();          /* calling assembly function */
        …
```

List 1 Example of calling assembly function without parameter

```
        .glb _main
_main:
        …
        (omitted)

        …
        jsr    _asm_func     ; Calls assembly function (preceded by '_')
        rts
```

List 2 Compiled code of assembly function in List 1

The two ways of passing arguments while calling an assembly function are shown below and described in the following sections.

1. By declaring *#pragma PARAMETER*, the C program and the arguments are passed via CPU registers R0 and R1.
2. By declaring global variables in the C program and also their corresponding global variables in assembly function, the arguments are passed via these global variables.

3.0 Passing Arguments by *#pragma PARAMETER*

The *#pragma PARAMETER* passes arguments to assembly functions via:

- 32-bit general-purpose registers (R2R0, R3R1, A1A0)
- 16-bit-bit general-purpose registers (R0, R1, R2, R3) or,
- 8-bit general purpose registers (R0L, R0H, R1L, R1H) and address registers (A0, A1)

The return value from an assembly function, if there is any, will be returned by register depending on the ***type*** of return value as discussed in Appendix A.2.

The following program segment, extracted from section 7.2 source code, shows the sequence of operations for calling an assembly function using *#pragma PARAMETER*.

1. Write a prototype declaration for the assembly function before the *#pragma PARAMETER* declaration. The parameter type(s) must also be declared.
2. Declare the name of the register using the *#pragma PARAMETER* in the assembly function's parameter list.

```
…
/* declare assembly function to be called from C */
extern unsigned int asm_add(unsigned int, unsigned int);

/* declare pragma to assign the variable transferring registers */
#pragma PARAMETER asm_add( R0, R1)
…
/* Calling assembly add function */
result = asm_add( addend, augend);
…
```

List 3 Prototype Declaration in C with #pragma PARAMETER

(code snippet from section 7.2.1)

```
.SECTION      program_a
.GLB   _asm_add      ;declaring global assembly function

_asm_add:
       ADD.W  R1,R0  ;math operation (addition) on variables passed
       RTS           ;R0 will be returned to the calling function
       .END
```

List 4 Assembly Function Declaration

(code snippet from section7.2.2)

4.0 Passing Arguments By Global Variables

In this method, a prototype for the assembly function must be declared but there is no need to mention the C variables in the prototype declaration. The C variables that need to be operated upon in the called assembly function should be declared as global variables in the C program as well as in the called assembly function. In order to refer to the C variables in the assembly function, each of them need to be preceded with an underscore "_". The return value from an assembly function, if there is any, will be returned by CPU register which depends on the ***type*** of the return value as described in Appendix A.2.

The following program segment, extracted from section 7.3, shows the sequence of operations for calling an assembly function while passing variables by global variable declaration.

```
…
/* declare assembly function to be called from C */
extern unsigned int asm_add(void);
…
/* declare global variables to be passed to called assembly function */
unsigned int addend = 2;
unsigned int augend = 6;
unsigned int result; /* this variable will hold the return value */
…
/* calling assembly add function */
result = asm_add();
…
```

List 5 Prototype Declaration in C using Global Variables

(code snippet from section 7.3.1)

```
.SECTION      program_a
.GLB          _asm_add             ;declaring assembly function as global
.GLB          _addend, _augend     ;referring to the global variables declared in C

_asm_add:
       MOV.W  _addend,R1    ;passing variables into registers
       MOV.W  _augend,R0

       ADD.W  R1,R0         ;math operation (addition) on variables
       RTS                  ;R0 will be returned to the calling function
       .END
```

List 6 Assembly Function Declaration

5.0 Optimization by Assembly Function Calls

An approximate idea about optimization by assembly function call may be obtained by calculating the CPU cycles involved in the assembly functions and also corresponding C function mentioned in sections 3.0 and 4.0. These functions are reproduced in sections are shown in List 7 to List 9 for convenience. The calculated number of CPU cycles for the assembly and C function calls in sections List 7 to List 9 are 8, 14 and 27 respectively (see [M16C/60 software manual]). Thus, there is an approximate saving of 70% computation time by the assembly function call by *#pragma PARAMETERS* when compared to corresponding C function call.

The approximate saving in computation time by assembly function call using global variables over C function call

is 50%. Moreover the C function call requires manipulation of stack frame buffer and that involves more RAM space. Thus assembly function calls realize optimization in computation time as well as in memory space when compared with equivalent function call in C.

```
_asm_add:
        ADD.W   R1,R0           ; add contents of R0 and R1 and put the result in R0
        RTS                     ; R0 will be returned to calling function
        .END
```

List 7 Addition function in assembly while passing arguments by registers R0 and R1

```
_asm_add:
        MOV.W   _addend,R1      ;argument transferred to register R1
        MOV.W   _augend,R0              ;argument transferred to register R2
        ADD.W   R1,R0           ;math operation (addition) on variables
        RTS                     ;R0 will be returned to the calling function
        .END
```

List 8 Addition function in assembly while passing arguments by global variable declaration

```
unsigned int add_C(unsigned int addend, unsigned int augend){
ENTER     #06H
MOV.W     R1,-2H[FB]
MOV.W     R2,-4H[FB]
        unsigned int k;
        k = addend + augend;
MOV.W     -2H[FB],-6H[FB]
ADD.W     -4H[FB],-6H[FB]
        return k;
MOV.W     -6H[FB],R0
EXITD
}
```

List 9 Addition function in C

6.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheets

- M16C/62 datasheets, 62aeds.pdf

User's Manual

- KNC30 Users Manual, KNC30UE.PDF
- M16C/60 and M16C/20 C Language Programming Manual, 6020EC.PDF
- Assembler for M16C/60 and M16C/20 User's Manual, AS30UE.PDF
- MSV30262-SKP or MSV-Mini26-SKP Quick start guide
- MSV30262-SKP or MSV-Mini26-SKP Users Manual
- MDECE30262 or MSV-Mini26-SKP Schematic

7.0 Sample Program

7.1 Application Code Outline

The sample programs, written in C and assembly, were compiled using the KNC30 compiler and will run on the MSV30262 SKP board. To run a program perform the following steps:

1. Load target program executable in the MCU Flash memory using KD30 (or FoUSB).
2. Reset the loaded program by pressing RESET and then run the program by pressing GO on the KD30 GUI. The following message appears on the LCD display:

S2 = ASM **S3 = C**

3. To select the mode for assembly function call, press button S2. The following message appears on the LCD display confirming the selection:

ASM mode **Press S4**

4. Press S4 for invoking the assembly function call and following message appears on LCD display confirming the invocation of an assembly add function:

calling **ASM add**

5. As soon as the assembly function call returns, the result of the function call is displayed on the LCD display as shown below:

3 + 6 **RES_a= 9**

6. To return to the initial menu, start over from step 2 mentioned above.
7. C function call may be invoked by following steps similar to 2-5 while choosing switch S3 instead of S2 in step 3.

7.2 Software Source Code for Passing Arguments by *#pragma PARAMETER*

7.2.1 Source Code in C

```
/***********************************************************************
*
*       File Name: main.c
*
*       Content:  This program demonstrates the technique of calling assembly
*                 function from C program. The calling function arguments are
*                 passed from C to the assembly function by declaring them as
*                 #pragma PARAMETERs. A return value is also generated from the
*                 assembly function and returned to the calling function in C.
*                 The program runs on the M16C/26 (firefly MCU) SKP board
*                 MSV30262. In order to invoke an addition function call in
*                 in assembly, press switch S2 and then S4. Whereas to invoke
*                 an addition function in C, press S2 and then S4.
*                 Press the reset switch (S1) to get the initial menu on the
*                 LCD display and proceed as mentioned above.
*
*       Date:  2-20-2003
*       This program was written to run on the MDECE30262 Board for MSV30262-SKP.
*
*       Copyright 2003 Renesas Technology America, Inc.
*       All rights reserved
*=======================================================================
*       $Log:$
*=======================================================================*/

#include "sfr262.h"
#include "skp26.h"
#include <stdio.h>
/* NOTE: include skp26.c file into project for LCD function calls */
/* skp26.c is used for the LCD on M30262 (firefly) SKP board */

/* declare assembly function to be called from C */
extern unsigned int asm_add(unsigned int, unsigned int);
/* declare pragma to assign the variable transferring registers */
#pragma PARAMETER asm_add( R0, R1)

/* define switches on skp board */
/* SWITCHES */
#define sw2             p10_5
#define sw3             p10_6
#define sw4             p10_7

/* define LEDs on skp board*/
#define red_led         p7_0
#define yellow_led      p7_1
#define green_led       p7_2

char buf[9]; /* buffer for displaying on 8 character line LCD */
```

```
/* declare variables */
unsigned int addend = 3;
unsigned int augend = 6;
unsigned int result;    /* result will hold addend+augend */
int loop = 1;
int asm_mode = 0;       /* assembly mode */
int C_mode   = 0;       /* C mode */

/* declare routine for displaying addend, augend and result on LCD */
void display_all(int addend, int augend, int result);
void init_LED( void ); /* routine that initializes LEDs */
unsigned int  add_C( unsigned int, unsigned int ); /* add function in C */

/*************************************************************************
Name:   main
Parameters: None
Returns: None
Description:  main program loop and initialization
*************************************************************************/
main(){
        init_LED();     /* initialize LEDs */
        init_disp();    /* initialize display */
        while(loop == 1){
                /* print initial message on LCD */
                        sprintf( buf, "S2 = ASM" );
                        display(0,buf);
                        sprintf(buf, "S3 = C  ");
                        display(1, buf);
                        loop = 2;       /* exit to next while loop */
        }

        /* select asm or C mode */
        while (loop==2){
                if( sw2 == 0 ){         /* sw2 pressed */
                        asm_mode = 1;   /* select asm mode */
                        C_mode = 0;
                        red_led = 0;    /* turn ON red LED */
                        sprintf( buf, "ASM mode" );
                        display(0,buf);
                        sprintf(buf, "Press S4");
                        display(1, buf);
                }
                if(sw3 == 0 ){          /* sw3 pressed */
                        asm_mode = 0;
                        C_mode  = 1;    /* select C mode */
                        yellow_led = 0;  /* turn ON yellow LED */
                        sprintf( buf, "C mode  " );
                        display(0,buf);
                        sprintf(buf, "Press S4");
                        display(1, buf);
                }
```

```
                if( sw4 == 0 ){         /* sw4 pressed */
                        loop = 0;       /* exit from while loop */
                        green_led = 0;  /* turn ON green LED */
                        sprintf( buf, "calling " );
                        display(0,buf);
                        if( asm_mode == 1){
                                sprintf(buf, "ASM add ");
                                display(1, buf);
                        }
                        if( C_mode == 1){
                                sprintf( buf, "C add   ");
                                display(1, buf);
                        }
                }
        }
        if( asm_mode == 1 ){
                result = asm_add( addend, augend);     /* Calling assembly add function */
                display_all( addend, augend, result);  /* Calling display routine */
        }
        if( C_mode == 1 ){
                result = add_C( addend, augend );      /* Calling C add function */
                display_all( addend, augend, result);  /* Calling display routine */
        }
}

/*************************************************************************
Name:  display_all
Parameters: int, int, int
Returns: int
Description: displays addend, augend and result on LCD display
*************************************************************************/
void display_all( int addend, int augend, int result){

        init_disp(); /* initialize and clear display */

        sprintf( buf, "%2i +%2i  ", addend, augend);
        display( 0, buf);               /* display in first line of LCD */

        if( asm_mode == 1){
                sprintf( buf, "RES_a=%2i", result); /* copy into buf */
                display( 1, buf); /* display result in second line of LCD */
        }
        else if( C_mode == 1){
                sprintf( buf, "RES_c=%2i", result); /* copy into buf */
                display( 1, buf); /* display result in second line of LCD */
        }
        return;
}
```

```
/***********************************************************************
Name:  init_LED
Parameters: None
Returns: None
Description: Initializes LEDs (D3, D4 and D5)
***********************************************************************/
void init_LED ( void ){
        /* set LED ports to outputs (connected to LEDs) */
        pd7_0 = 1;
        pd7_1 = 1;
        pd7_2 = 1;

        /* turn OFF all LEDs */
        red_led = 1;
        yellow_led = 1;
        green_led = 1;
}

/***********************************************************************
Name:  add_C
Parameters: int, int, int
Returns: int
Description: adds two integers and returns the result as an integer
***********************************************************************/
unsigned int add_C(unsigned int addend, unsigned int augend){
        unsigned int k;
        k = addend + augend;
        return k;
}
```

7.2.2 Source Code of Assembly Function

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Name: asm_add
;Parameters: addend, augend (passed through registers R0 and R1)
;Returns: int (passed through register R0)
;Description: Assembly addition function called from C program
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.SECTION       program_a
.GLB    _asm_add               ;declaring global assembly function

_asm_add:
        ADD.W   R1,R0   ;math operation (addition) on variables passed
        RTS             ;R0 will be returned to the calling function
        .END
```

7.3 Software Source Code for Passing Arguments by Global Variables

7.3.1 Source Code in C

```
/***********************************************************************
*
*       File Name: main.c
*
*       Content:   This program demonstrates the technique of calling assembly
*                  function from C program. The calling function arguments are
*                  passed from C to the assembly function by declaring them as
*                  #pragma PARAMETERs. A return value is also generated from the
*                  assembly function and returned to the calling function in C.
*                  The program runs on the M16C/26 (firefly MCU) SKP board
*                   MSV30262. In order to invoke an addition function call in
*                  in assembly, press switch S2 and then S4. Whereas to invoke
*                  an addition function in C, press S2 and then S4.
*                  Press the reset switch (S1) to get the initial menu on the
*                  LCD display and proceed as mentioned above.
*
*       Date:  2-20-2003
*       This program was written to run on the MDECE30262 Board for MSV30262-SKP.
*
*       Copyright 2003 Renesas Technology America, Inc.
*       All rights reserved
*
*=======================================================================
*       $Log:$
*=======================================================================*/
#include "sfr262.h"
#include "skp26.h"
#include <stdio.h>

/* NOTE: include skp26.c file into project for LCD function calls */
/* skp26.c is provided for the LCD on M16C/26 (firefly) SKP board*/
/* declare assembly function to be called from C */
extern unsigned int asm_add(void);
/* define switches on skp board */
/* SWITCHES */
#define sw2             p10_5
#define sw3             p10_6
#define sw4             p10_7

/* define LEDs on skp board*/
#define red_led         p7_0
#define yellow_led      p7_1
#define green_led       p7_2

char buf[9]; /* buffer required for displaying on 8 character line LCD */
```

```
unsigned int addend = 2;/* declare global variables.. */
unsigned int augend = 6;/* to be transferred to the assembly function*/
unsigned int result;    /* result variable will hold addend+augend */

int loop = 1;
int asm_mode = 0;       /* assembly mode */
int C_mode   = 0;       /* C mode */

/* declare routine for displaying addend, augend and result on LCD display */
void display_all(int addend, int augend, int result);
void init_LED( void ); /* routine that initializes LEDs */
unsigned int  add_C( unsigned int, unsigned int ); /* add function in C */

/************************************************************************
Name:   main
Parameters: None
Returns: None
Description:  main program loop and initialization
************************************************************************/
main(){
        init_LED();     /* initialize LEDs */
        init_disp();    /* initialize display */
        /* select asm or C mode */
        while(loop == 1){
                /* print initial message on LCD */
                        sprintf( buf, "S2 = ASM" );
                        display(0,buf);
                        sprintf(buf, "S3 = C  ");
                        display(1, buf);
                        loop = 2;       /* exit to next while loop */
        }

        while (loop == 2){
                if( sw2 == 0 ){         /* sw2 pressed */
                        asm_mode = 1;   /* select asm mode */
                        C_mode = 0;
                        red_led = 0;    /* turn ON red LED */
                        sprintf( buf, "ASM mode" );
                        display(0,buf);
                        sprintf(buf, "Press S4");
                        display(1, buf);
                }
                if(sw3 == 0 ){          /* sw3 pressed */
                        asm_mode = 0;
                        C_mode  = 1;    /* select C mode */
                        yellow_led = 0; /* turn ON yellow LED */
                        sprintf( buf, "C mode  " );
                        display(0,buf);
                        sprintf(buf, "Press S4");
                        display(1, buf);

                }
```

```
                if( sw4 == 0 ){         /* sw4 pressed */
                        loop = 0;       /* exit from while loop */
                        green_led = 0;  /* turn ON green LED */
                        sprintf( buf, "calling " );
                        display(0,buf);
                        if( asm_mode == 1){
                                sprintf(buf, "ASM add ");
                                display(1, buf);
                        }
                        if( C_mode == 1){
                                sprintf( buf, "C add   ");
                                display(1, buf);
                        }
                }
        }
        if( asm_mode == 1 ){
        result = asm_add();                     /* calling assembly add function */
        display_all( addend, augend, result);  /* calling display routine*/
        }
        if( C_mode == 1 ){
                result = add_C( addend, augend );    /* Calling C add function */
                display_all( addend, augend, result);/* Calling display routine */
        }
}
/*************************************************************************
Name:  display_all
Parameters: int, int, int
Returns: int
Description: displays addend, augend and result on LCD display
*************************************************************************/
void display_all( int addend, int augend, int result){

        init_disp(); /* initialize and clear display */
        sprintf( buf, "%2i +%2i  ", addend, augend);
        display( 0, buf);       /* display in first line of LCD */

        if( asm_mode == 1){
                sprintf( buf, "RES_a=%2i", result); /* copy into buf */
                display( 1, buf); /* display result in second line of LCD */
        }
        else if( C_mode == 1){
                sprintf( buf, "RES_c=%2i", result); /* copy into buf */
                display( 1, buf); /* display result in second line of LCD */
        }
        return;
}
```

```
/***********************************************************************
Name:  init_LED
Parameters: None
Returns: None
Description: Initializes LEDs (D3, D4 and D5)
***********************************************************************/
void init_LED ( void ){
        /* set LED ports to outputs (connected to LEDs) */
        pd7_0 = 1;
        pd7_1 = 1;
        pd7_2 = 1;

        /* turn OFF all LEDs */
        red_led = 1;
        yellow_led = 1;
        green_led = 1;
}

/***********************************************************************
Name:  add_C
Parameters: int, int, int
Returns: int
Description: adds two integers and returns the result as an integer
***********************************************************************/
unsigned int add_C(unsigned int addend, unsigned int augend){
        unsigned int k;
        k = addend + augend;
        return k;
}
```

7.3.2 Source Code of Assembly Function

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Name: asm_add
;Parameters: addend, augend (passed through global variable declaration)
;Returns: int (passed through register R0)
;Description: Assembly addition function called from C program
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.SECTION        program_a
.GLB            _asm_add                ;declaring assembly function as global
.GLB            _addend, _augend        ;referring to the global variables declared in C

_asm_add:
        MOV.W   _addend,R1      ;passing variables into registers
        MOV.W   _augend,R0

        ADD.W   R1,R0           ;math operation (addition) on variables
        RTS                     ;R0 will be returned to the calling function
        .END
```

Appendix A. Writing Assembly Functions

A.1 Method for Writing Called Assembly Function

The following shows a procedure for writing the entry processing of assembly functions.

1. Specify section names using the assembler pseudo command **.SECTION**. Sections can be assigned any desired name.
2. Global specify function name labels using the assembler pseudo command **.GLB**.
3. Add the underscore(_) to the function name to write it as label.
4. When modifying the B and U flags within the function, save the flag register to the stack beforehand.

The following shows a procedure for writing the exit processing of assembly functions.

1. If you modified the B and U flags within the function, restore the flag register from the stack.
2. Write the RTS instruction.

Do not change the contents of the SB and FB registers in the assembler function. If the contents of the SB and FB registers are changed, save them to the stack at the entry to the function, then restore their values from the stack at the exit of the function. List 10 is an example of how to code an assembler function. In this example, the section name is ‘program’, which is the same as the section name output by NC30.

```
        .SECTION      program                 <= (1)
        .GLB          _asm_func       <= (2)
_asm_func:                            <= (3)
        PUSHC         FLG             <= (4)
        PUSHM         R3, R1          <= (5)
        MOV.W         SYM1, R3
        MOV.W         SYM1+2, R1
                      (omitted)
                      … …
        POPM          R3, R1          <= (6)
        POPC          FLG             <= (7)
        RTS                           <= (8)
        .END
```

List 10 Coding example of an assembly function

A.2 Returning Return Values From Assembly Function

When returning values from an assembler function to a C language program, registers can be used through which to return the values of the integer, pointer and floating-point types. Table 1 lists the rules on calls regarding return values.

Table 1 Calling rules for return values

Return value types	Rules
boolean char	R0L register
Int near pointer	R0 register
float long far pointer	The 16-low order bits are stored in the R0 register and the 16 high order bits are stored in the R2 register as the value is returned.
double long double	The value is stored in 16 bits each beginning with the MSB in order of registers R3, R2, R1 and R0 as it is returned.
long long	The value is stored in 16 bits each beginning with the MSB in order of registers R3, R2, R1 and R0 as it is returned.
compound	Immediately before calling the function, the far address indicating the area for storing the return value is pushed to the stack. Before the return to the calling program, the called function writes the return value to the area indicated by the far address pushed to the stack.

Keep safety first in your circuit designs!

- 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

- 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.
- 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.
- 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).
- 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.
- 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.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
  Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.