お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

===== 必ずお読みください =====

7700 ファミリ 79xx シリーズ用 C コンパイラパッケージ
V.4.10 Release 1C
リリースノート
(第 5 版)

株式会社ルネサス ソリューションズ

2006 年 2 月 16 日

概要

この度は 7700 ファミリ 79xx シリーズ用 C コンパイラパッケージ V.4.10 Release 1C をお買い上げいただきましてありがとうございます。

本資料は C コンパイラパッケージの電子マニュアルの補足等について説明します。電子マニュアルの該当項目をご覧になる場合は、併せてこのリリースノートをご覧いただきますようお願い申し上げます。

1. 注意事項

製品をご使用になる際は、以下の内容に従ってくださるようお願いいたします。

1.1. コンパイラに関する注意事項

1.1.1. スタートアッププログラムに関する注意事項

添付のスタートアッププログラム(ncrt0.a79、sect79.inc)は M37911 グループ用にメモリ配置しています。
このため、M37902 グループ等、M37911 グループとは異なるマイコンを使用される場合は、ご使用のマイコンに合わせてメモリ配置を変更してください。

- 記述例

(1) 割り込みベクタ

```
[ M37911/10 ]
                        .section    vector
                        .org        7fffb0H
        RESERVED15:     .word       OFFSET dummy_int
                                      :
                                      :
[ M37902/20 ]
                        .section    vector
                        .org        00ffc0H
        DMA3:           .word       OFFSET dummy_int
                                      :
                                      :
```

(2) 割り込み処理プログラムの再配置

```
[ M37911/10 ]
                        .section    interrupt
                        .org        7f0000H
[ M37902/20 ]
                        .section    interrupt
                        .org
```

1.1.2. 入出力関数の引数に関する注意事項

以下の入出力関数の書式指定において、フィールド幅を 0 フラグと'*'で指定した場合、誤った結果を出力します。

- fprintf
- printf
- sprintf
- vfprintf
- vprintf
- vsprintf

例えば、printf("%0*d",keta,val) の書式指定は、以下を意味します。

```
0 ---- 右寄せして、表示桁数に不足する分を文字'0'で埋める。
* ---- 表示する桁数を引数 keta から取り込む。
d ---- 引数 val を int 型変数として解釈し、10 進数で表示する。
```

しかし、'0'に後続する'*'が書式指定として正しく解釈されないため、"*d"の文字列が表示されてしまいます。

- 発生例

```
#include <stdio.h>

int             main( void )
{
                printf( "%0*d¥n", 4, 123 );
}
```

- 回避策

  フィールド幅を 10 進整数で指定してください。

```
#include <stdio.h>

int             main( void )
{
                printf( "%04d¥n", 123 );
}
```

1.1.3. 標準ライブラリ関数 atof および strtod に関する注意事項

標準ライブラリ関数 atof または strtod の引数が".12345"のようにピリオドで始まる文字列の場合、その関数の変換結果が"0"になります。

- 発生例

```
#include <stdlib.h>

double      d;

int             main( void )
{
                d = atof( ".12345" );      /* 引数の文字列がピリオドで始まっている */
}
```

- 発生条件

  引数の文字列から空白文字を除くと、先頭の文字が".(ピリオド)"になる場合に発生します。

- 回避策

  ピリオドの前に"0"を付加してください。

```
#include <stdlib.h>

double      d;

int             main( void )
{
                d = atof( "0.12345" );
}
```

1.1.4. 構造体型の配列定義に関する注意事項

構造体配列のメンバへアクセスするコードを記述した場合、正しいコードが生成されない場合があります。

- 発生条件

  以下の条件をすべて満たす場合に発生します。

  (1) 構造体型の配列定義がある
  (2) 構造体の定義がある
  (3) (1)の定義が(2)の定義よりも先に記述されている
  (4) (1)で定義したメンバへアクセスするコードを記述している

- 発生例

```
struct AAA a[10];                   /*  発生条件(1) &  発生条件(3) */

struct AAA {                        /*  発生条件(2) */
        int         a;
        int         b;
};

int         gi;

void        smp(int i)
{
        gi = a[i].b;                /*  発生条件(4) */
}
```

- 回避策

  発生条件(2)の定義を、発生条件(1)の定義よりも先に記述してください。

```
struct AAA {
        int         a;
        int         b;
};

struct AAA a[10];
```

1.1.5. 標準ライブラリ関数 sprintf に関する注意事項

標準ライブラリ関数 sprintf 関数の引数 % と f の間にスペースを挿入した場合、代入結果が 0.000000 となる場合があります。

- 発生条件

  浮動小数点の整数部が 0 で小数部が 999999 のように、小数点末尾に対する四捨五入の結果、整数部へ桁上がりとなる数字が引数として与えられた場合に発生します。

- 発生例

```
#include <stdio.h>

float     f;

int         main( void )
{
            char      buf[10];

            f = 0.999999;
            sprintf( buf,"% -8.2f",f );
}
```

- 回避策

  以下のいずれかの方法で回避してください。

  (1) ライブラリ関数のソースファイル print.c を修正し、再度ライブラリファイルを作成する。関数_f8prn の「四捨五入の結果、整数部へ桁上がりする場合」の処理を変更してください。

```
/* 四捨五入の結果、整数部へ桁上がりする場合 */
if ( CHK_KETA ) {
        if ( (*format == 'e' || *format == 'E') && inte[0]=='9' ) {
                /* %e 指定で整数部が 9 のとき */
                inte[0] = '1';          /* 整数部を１にする */
                if ( CHK_EFUGO ) {
                        /* 指数部の符号が負の場合 */
                        cnt--;
                        if ( !cnt )
                                /* 指数部が０なら指数部の符号を正に */
                                CLR_EFUGO;
                } else
                        cnt++;
        } else {
                for ( r=0; r<seisu; r++ ) {
                        if ( inte[r] == '9' )
                                inte[r] = '0';
                        else {
                                ++inte[r];
                                break;
                        }
                }
                if ( r==seisu && r!=0) {
                        inte[seisu] = '1';
                        seisu++;
                }
                else if( seisu == 0){           // 追加
                        inte[seisu] = '1';      // 追加
                        seisu++;                // 追加
                }                               // 追加
        }
}
```

(2) sprintf 関数の引数に浮動小数点の絶対値を渡す。

```
#include <stdio.h>
#include <math.h>                       /* math.h をインクルード */

float   f;

int     main( void )
{
        char    buf[10];

        f = 0.999999;

        /* 正ならばスペース、負ならば -(マイナス)記号をバッファの 0 番目に代入 */
        if( f >= 0.0F )
                buf[0] = ' ';
        else
                buf[0] = '-';

        /* 実数の絶対値をバッファの 1 番目以降に代入 */
        /* %の次には空白を挿入しない*/
        /* %以降の引数は 8.2 から 7.2 に変更 */
        sprintf( &buf[1],"%-7.2f",fabs(f) );
}
```

1.1.6. 同一変数を複数の if 文で連続して参照する場合の注意事項

複数の if 文条件式に同じ変数がある場合、コンパイル時に System Error が発生する場合があります。

- 発生条件

  以下の条件をすべて満たす場合に発生することがあります。

  (1) オプション -O, -O[1-5], -OR, -OS のいずれか 1 つ以上を選択している。
  (2) if-else の形式の if 文が、else 側に 2 段以上ネストして存在する。ただし、最も内側は else がない if 文であっても発生する。
  (3) (2)のネストしているすべての if 文の条件式に同じ変数がある。
  (4) (2)の if-else 文の後続処理に、(2)の if 文と同じ変数を使う if 文がある。
  (5) (4)の if 文の実行直前に必ずしも(2)の if-else 文を実行しない経路がある。
  (6) (4)における if 文の直前の位置に、無条件分岐または関数からの復帰がある。

- 発生例
  - 例 1

```
int     a, b, cond;

void    func(void)
{
        if (a == 1) {                           /*  発生条件(5) */
                if (cond > 10) {                /*  発生条件(2)、(3)1 段目のネスト  */
                        b += 1;
                } else if (cond > 5) {          /*  発生条件(2) 、(3)2 段目のネスト  */
                        b += 2;
                } else if (cond > 3) {          /*  発生条件(2) 、(3)3 段目のネスト  */
                        return;                 /*  発生条件(6) */
                }
        }
        if (cond == 1) {                        /*  発生条件(4) */
                b += 3;
        }
}
```

  - 例 2

```
int     a, b, cond;
void    func(void)
{
        if (a == 1) {                           /*  発生条件(5) */
                if (cond > 10) {                /*  発生条件(2)、(3) 1 段目のネスト  */
                        b += 1;
                } else
                        if (cond > 5) {         /*  発生条件(2)、(3) 2 段目のネスト  */
                                b += 2;
                        } else {
                                if (cond > 3)   /*  発生条件(2)、(3) 3 段目のネスト  */
                                        b += 3;
                        }
                }
        }
        else {
                if (a == 2)
                        return;                 /*  発生条件(6) */
        }
        if (cond == 0x0001)                     /*  発生条件(4) */
                b += 3;
}
```

- 回避策

  発生条件(4)の直前に、ダミーの asm 関数を挿入してください。

  - 例 1 の回避例

```
int         a, b, cond;

void        func(void)
{
            if (a == 1) {
                        if (cond > 10) {
                                    b += 1;
                        } else if (cond > 5) {
                                    b += 2;
                        } else if (cond > 3) {
                                    return;
                        }
            }
            asm();                                  /* ダミーの asm 関数を挿入 */
            if (cond == 1) {
                        b += 3;
            }
}
```

1.1.7. typedef により構造体または共用体の配列を型定義する場合の注意事項

構造体または共用体の配列を typedef によって型定義し、その定義した型の変数に対して near / far 修飾子を付加して宣言した場合、不正なコードを生成する場合があります。

- 発生条件

  以下の条件をすべて満たす場合に問題が発生する場合があります。

  (1) 構造体または共用体を定義している。
  (2) (1)の構造体または共用体の配列を typedef によって型定義している。
  (3) (2)で定義した型によって変数を宣言している。
  (4) (3)の宣言には near / far 修飾子が付加されている
  (5) (1)の構造体または共用体のメンバを参照している。

- 発生例

```
struct tag {                                        /* 発生条件(1) */
            long        l;
            char        c;
};

typedef   struct tag    ARR[3];                     /* 発生条件(2) */
far   const   ARR       dat                         /* 発生条件(3)、(4) */
                        = { 1, 2, 3, 4, 5, 6 };

void        func(int i)
{
            char c;

            c = dat[i].c + 1;                       /* 発生条件(5) */
}
```

- 回避策

  発生条件(4)で該当した修飾子を、発生条件(2)の型定義時に構造体または共用体の配列に付加してください。

```
struct tag {
        long        l;
        char        c;
};

typedef   struct tag   far          ARR[3];             /* far 修飾子を付加 */
far   const   ARR                   dat = { 1, 2, 3, 4, 5, 6 };

void        func(int i)
{
        char c;

        c = dat[i].c + 1;
}
```

1.1.8. 標準関数ライブラリ scanf, fscanf, sscanf に関する注意事項

scanf, fscanf, sscanf 関数を用いて、'0'を含んだ入力文字列を変換指定子 'x'を用いて変換すると、正しく変換されない場合があります。

- 発生条件

  以下 4 点の条件をすべて満たす場合に発生します。

  (1) 入力文字列中に '0' が存在する。
  (2) (1)の'0'を変換指定子'x'を用いて変換している。
  (3) (2)の'0'は、以下のいずれかに該当する。
  (4) 入力文字列の先頭に位置する。
  (5) '0'の直前の文字が '0'～'9'、'a'～'f'、'A'～'F'ではない。
  (6) (2)における'0'の直後の文字が '0'～'9'、'a'～'f'、'A'～'F'、'x'、'X'、'¥0'ではない。

- 発生例
  - プログラム例

```
#include <stdio.h>
void        main(void);
void        func(void);

void        main(void)
{
        func();
}

void        func(void)
{
        int         input = 1234;
        int         returnVal = 0;
        const char* pStr = "0";                     /*  発生条件(1)、(3)、(4) */

        returnVal = sscanf(pStr, "%x", &input);     /*  発生条件(2) */
}
```

  - 上記プログラム例の実行結果

```
returnVal = -1
input = 1234
        正しい変換結果は returnVal の値は 1、input の値は 0 です。
```

- 回避策

  以下に述べる方法でライブラリソースファイル scan.c を修正し、ライブラリを再作成してください。

ライブラリを再作成する方法については、各ユーザーズマニュアルの「標準入出力関数ライブラリのカスタマイズ方法」を参照してください。

(1) 該当個所

インストールディレクトリ下の src79¥lib ディレクトリの scan.c 318 行目以降

(2) 修正前

```
hex:
        data = 0L;
        if ( !width || !isxdigit( c ) )
                break;
        if(c=='0'){
                c=(*f_in)();
                if(c=='x'||c=='X'){
                        if(!isxdigit(c=(*f_in)()))
                                goto next;
                }
                else if(!isxdigit(c))
                        break;
        }
```

(3) 修正後

```
hex:
        data = 0L;
        if ( !width || !isxdigit( c ) )
                break;
                if(c=='0'){
                        c=(*f_in)();
                        width--;                        /* width--; を追加します */
                        if(c=='x'||c=='X'){
                                if(!isxdigit(c=*f_in)()))
                                        goto next;
                        }
                        else if(!isxdigit(c)){          /* break; を { } で囲み、break; の前に
                                status = TRUE;             status = TRUE; を追加します。 */
                                break;
                        }
        }
```

1.1.9. switch 文の分岐先に関する注意事項

1 つの関数内に 2 つ以上の switch 文が含まれている C 言語ソースファイルをコンパイルすると、switch 文の分岐先を誤った不正なコードが生成されることがあります。

C コンパイラでは、分岐先の多い switch 文について、各分岐先のアドレスの一連のテーブル（以下、分岐先テーブルといいます）を生成して、実行命令の並びの中に挿入し、この分岐先テーブルを参照して間接分岐を行うコードを生成します。

当該現象が発生した場合、この分岐先テーブルが部分的に欠落して、正しいアドレスに分岐しなくなります。

- 発生条件

以下 6 点の条件をすべて満たす場合に発生することがあります。

(1) コンパイルオプション ”-OR” を使用している。

(2) コンパイルオプション ”-ONBSD (-Ono_break_source_debug) “は使用していない。

(3) コンパイルオプション ”-fST(-fswitch_table)”を使用している。

(4) 1 つの関数内で 2 つ以上の switch 文を記述している。

(5) 2 つの switch 文は、コンパイルの結果、どちらもそれぞれ分岐先アドレスのテーブルとそれを参照して間接分岐を行うコードに展開されている。

(6) 以下のいずれか 1 つ以上を満たす。

- 一方の switch 文における分岐先のいずれかに、他方の switch 文の分岐先のいずれかと共通な処理が

ある。

- 両方の switch 文ともに、何もせずに switch 文を抜ける場合がある。
- switch 文から抜けるための break 文のみを持つ case ラベルまたは default ラベルがある。
- default ラベルがない。

- 発生例

```
extern int   a, b, c, x;

void         func(void)
{
             if (a) {
                          switch (b) {                    /*  発生条件(4)  第一の switch 文  */
                          case 8:
                          case 9:
                          case 11:
                          case 12:
                          case 13:
                          case 14:
                          case 15:
                                       x++;               /*  発生条件(6) */
                                       break;
                          case 10:
                                       x = 2;
                                       break;
                          default:
                                       x = 0;              /*  発生条件(6) */
                                       break;
                          }
             } else {
                          switch (c) {                    /*  発生条件(4)  第二の switch 文  */
                          case 33:
                          case 34:
                          case 35:
                          case 36:
                          case 37:
                          case 38:
                          case 39:
                          case 40:
                                       x++;               /*  発生条件(6)と共通処理  */
                                       break;
                          default:
                                       x = 0;             /*  発生条件(6)と共通処理  */
                                       break;
                          }
             }
}
```

- 回避策

  共通処理がある全 case ラベルおよび default ラベルの直後にダミーの asm 関数を挿入してください。

```
extern int   a, b, c, x;

void         func( void )
{
             if (a) {
                          switch ( b ) {
                          case 8:
                                       /* 中略 */
                          case 15:
                                       asm();                  /* ダミーの asm 関数 */
                                       x++;
                                       break;
                          case 10:
                                       x = 2;
                                       break;
                          default:
                                       asm();                  /* ダミーの asm 関数 */
                                       x = 0;
                                       break;
                          }
             } else {
                          switch (c) {
                          case 33:
                                       /* 中略 */
                          case 40:
                                       asm();                  /* ダミーの asm 関数 */
                                       x++;
                                       break;
                          default:
                                       asm();                  /* ダミーの asm 関数 */
                                       x = 0;
                                       break;
                          }
             }
}
```

1.1.10. if 文中のビットフィールドの評価結果に関する注意事項

if 文の条件式に論理 AND 演算子 (&&) で連結したビットフィールドの評価式が複数記述されている場合、不正なコードを生成します。

- 発生条件

  以下 6 点の条件をすべて満たす場合に発生します。

  (1) コンパイルオプションに "-OR" を使用し、かつ "-O5" または "-O4" を併用している。
  (2) if 文の条件式にビットフィールド (1 ビット) の評価式が複数記述されている。
  (3) (2)の複数ある評価式が"==" のみの判定である。
  (4) (2)の複数ある評価式に即値 "1" と比較する評価式が 1 つ以上ある。
  (5) (2)の複数ある評価式が論理 AND 演算子 (&&) で連結されている。
  (6) 条件成立時の処理が何も行わないもしくは無条件分岐である。

- 発生例

```
sturct      bitf {
            int         b0:1;
            int         b1:1;
            int         b2:1;
}bit;
int         i;

void        func1(void)
{
            if((bit.b0 == 1)&&(bit.b2 == 1)){       /* 発生条件(2)、(3)、(4)、(5) */
                        ;                           /* 発生条件(6) */
            }else{
                        i = 1;
            }
}

void        func2(void)
{
            if((bit.b0 == 1)&&(bit.b2 == 1)){       /* 発生条件(2)、(3)、(4)、(5) */
                        goto L1;                    /* 発生条件(6) */
            }
            i = 1;
L1:;
}
```

- 回避策

  以下のいずれかの方法で回避してください。

  (1) コンパイルオプション "-Ono_logical_or_combine" (-ONLOC) を使用してコンパイルする。

  (2) if 文の条件式が成立した場合に実行される実行文の先頭に、ダミーの asm 関数を記述する。

```
void        func1(void)
{
            if((bit.b0 == 1)&&(bit.b2 == 1)){
                        asm();                      /* ダミーの asm 関数を記述 */
            }else{
                        i = 1;
            }
}
```

1.1.11. switch-case 文に関する注意事項

switch-case 文のコード生成において不正なコードが生成される場合があります（switch-case 文において、case 値に依存せず一定の case あるいは default ラベルにのみジャンプするコードが生成されます）。

- 発生条件

  コンパイルオプション-fswitch_table[-fST]を指定し、かつ下記に示す条件のうちいずれかを満たした場合に発生します。

  (1) switch 文の条件式中に unsigned char 型の変数を使用しており、下記 3 項目のいずれか 1 つを満たしている場合
  - case 値の最小値が"1"でかつ最大値が"255"、その case 値が"1"から"255"まで隙間なく連続している
  - case 値の最小値が"0"でかつ最大値が"254"、その case 値が"0"から"254"まで隙間なく連続している
  - case 値の最小値が"0"でかつ最大値が"255"、その case 文（default を含まない）の総数が 143 個以上存在する

  (2) switch 文の条件式中に signed char 型の変数を使用しており、下記 3 項目のいずれか 1 つを満たしている場合
  - case 値の最小値が"-127"でかつ最大値が"127"、その case 値が"-127"から"127"まで隙間なく連続している
  - case 値の最小値が"-128"でかつ最大値が"126"、その case 値が"-128"から"126"まで隙間なく連続している

- case 値の最小値が"-128"でかつ最大値が"127"、その case 文（default を含まない）の総数が 143 個以上存在する

- 発生例

```
char        c;

/* case 値の 1 から 255 までが隙間なく連続している場合 */
switch( c ){
case        0:
case        1:
case        2:
                    func( 3 );
                    break;
                      :
                      :
case        255:
                    func( 255 );
                    break;
default:
                    break;
}
```

- 回避策

  対象となる switch 文の条件式中の変数が unsigned char 型の場合は unsigned int 型にキャストを、signed char 型の場合は signed int 型にキャストしてください。

```
char        c;

switch( (unsigned int )c ){          /* 変数 c を unsigned int 型にキャスト */
case   0:
                    :
                    :
case   255:
                    :
                    :
}
```

1.1.12. 標準ライブラリ"pow"関数に関する注意事項

標準ライブラリの"pow"関数を呼び出した場合、定義域内の値を引数に渡しているにも関わらず、定義域エラーが発生することがあります。

- 発生条件

  以下 2 点のいずれか 1 つを満たす場合に発生することがあります。

  (1) 第一引数がゼロかつ第二引数が正である

  (2) 第一引数が非ゼロかつ第二引数が負である

- 発生例

```
#include <math.h>
#include <error.h>

double      ans1;

int         func( void )
{
            ans1 = pow( 0.0 , 5.0 );            /* 発生条件(1)*/
            if( errorno == EDOM )
                    return   -1;                /* 定義域エラー発生のチェック */
            return      0;
}
```

- 回避策

  製品に添付されているライブラリソースファイルを修正し、ライブラリファイルを再作成してユーザプログラムをリンクしなおしてください。
  なお、ライブラリファイルの再作成方法の詳細については、ユーザーズマニュアルの「E.3.2　入出力関数の変更手順を変更したソースプログラムの組み込み」を参照してください。

  - 正

```
if(( x == 0    && y <= 0 ) | | ( x < 0 && y != (int)y )){
```

  - 誤

```
if( x == 0    | | y <= 0 ) | | ( x < 0 && y != (int)y )){
```

1.1.13. "-OS"オプション指定時におけるループ文に関する注意事項

C コンパイラは、コンパイルオプション"-OS"指定時に「ループ内にある不変式をループ前に移動させる最適化」を実施しています。本最適化は、ループ内の条件分岐（if 文、switch 文等）により実行されないことが有り得る経路上の式については適用しません。しかし、switch 文の分岐先での式に対して誤ってループ前に移動してしまうことがあります。この時生成されたコードは実行しないかもしれない計算を必ず実行してしまいます。

- 発生条件

  以下 6 点をすべて満たす場合に発生することがあります。

  (1) コンパイルオプション"-OS"を指定している
  (2) コンパイルオプション"-fswitch_table[-fST]"を指定している
  (3) for または while 文中に switch 文が存在する
  (4) 自動引数、レジスタ変数、または引数のいずれかが case ラベル、または default ラベルに分岐した先で更新されている
  (5) 自動引数、レジスタ変数、または引数のいずれかが case ラベル、または default ラベルに分岐した先で更新されている変数に格納される値がループの反復実行によって変化しない
  (6) コンパイルにより、for または while 文中に存在する switch 文が分岐先テーブル　による間接分岐の命令列を生成する

- 発生例

```
int         func( void )
{
            int         j, a, flag = 0;                 /*  発生条件(4) */

            for( j = 0; j < 1; j++){
                        switch( a ){                    /*  発生条件(3) */
                        case    8:
                                    break;
                        case    9:
                        case    10:
                        case    11:
                        case    12:
                        case    13:
                        case    14:
                        case    15:
                        default:
                                    flg = 1;            /*  発生条件(4)、(5)　この式が不正に移動 */
                                    break;
                        }
            }
            return    flg;
}
```

- 回避策

  本現象が発生する反復処理の先頭にダミーの asm 関数を挿入してください。

```
for( j = 0; j < 1; j++){
        asm();                  /* ダミーの asm(); 関数を挿入 */
        switch( a ){
        case   8:
                break;
```

1.1.14. ループ内でポインタ変数を更新する場合の注意事項

ポインタ変数によって間接参照した値にそのポインタ自体を更新する処理を記述するとコンパイルが強制終了することがあります。この場合、EWS 版ではコアダンプが発生し、PC 版では「不正な処理を行いました」等のメッセージが表示されます。

- 発生条件

  以下４点をすべて満たす場合に発生することがあります。条件をすべて満たす場合でも現象が発生しないことがありますが、この場合生成されたコードに問題はありません。

  (1) ポインタ変数で間接参照した値をそのポインタ自身に書き込んでいる
  (2) (1)のポインタ変数がレジスタに割り当てられている
  (3) (1)のポインタを使用する for 文、または while 文がある
  (4) (3)と同じ for 文、または while 文の中に間接参照がないと仮定すれば、ループ外に移動できそうな不変式が存在する

- 発生例

```
struct   l{
        unsigned int            no;
        struct   l    *next;
} *pp;

int         func( void )
{
        register struct   l     *p = pp;                    /*  発生条件(2) */
        int                     j;
        int                     n = 1;
        int                     x = 0;

        for(   j = 0; j < n-1; j++ ){                       /*  発生条件(4)   “n-1”は不変式 */
                if( p->no > (p->next)->no )         x++;    /*  発生条件(3) */
                p = p->next;                                /*  発生条件(1) */
        }
}
```

- 回避策

  本現象が発生する反復処理の先頭にダミーの asm 関数を挿入してください。

```
for(   j = 0; j < n-1; j++ ){
        asm();                                              /* ダミーの asm(); 関数を挿入 */
        if( p->no > (p->next)->no )         x++;
}
```

1.1.15. 整数型配列に関する注意事項

整数型配列の初期値が正しくコード生成されない場合があります。

- 発生条件

  以下 4 点をすべて満たす場合に発生する場合があります。なお、条件をすべて満たす場合でも C ソースの構文によっては問題が発生しない場合があります。

  (1) 「外部変数」または「関数内 static 変数」として配列を定義する時に配列初期値を設定している
  (2) (1)の配列初期値の第 1 要素に演算子を含んだ式が存在しない
  (3) (1)の配列初期値の第 2 要素以降に演算子を含んだ式を記述している

(4) (3)の演算子を含んだ式の直後の値が 10 進数で 256 以上 342 以下の整定数である

- 発生例
  - 例 1

```
int         array1[] = { 1,-5,302 };            /* NG */
int         array2[] = { -1,-5,302 };           /* OK */
```

  - 例 2

```
int         j;
unsigned    long        addr1[] = { 0, (unsigned    long)& j; 208 };                    /* NG */
unsgined    long        addr2[] = { (unsigned long)0 ; (unsigned    long)& j; 308 };  /* OK */
```

- 回避策

  本現象が発生した場合には配列初期値の第 1 要素にキャスト演算子を追加してください。

1.1.16. enum 宣言に関する注意事項

タグ名のない enum 型を宣言した場合、以下のエラーが発生する場合があります。

```
Windows 版：「不正な処理を行いました」
EWS 版：「core dump」
```

- 発生条件

  以下 2 点の条件をすべて満たした場合に発生します（ポインタではなく、enum 値そのものを利用する場合は問題ありません）。

  (1) タグ名のない enum 型へのポインタ、あるいはタグ名のない enum 型の配列を宣言している

  (2) (1)のポインタ、または配列を使って次のいずれかの処理を行っている
    - 代入演算、または比較演算を行っている
    - プロトタイプ宣言を行っている

- 発生例

```
enum { A_1, A_2 }       *ap;                    /*  発生条件(1) */
enum { B_1A_2 }         *bp;                    /*  発生条件(1) */

ap = bp;                                        /*  代入演算：発生条件(2) */
ap < bp;                                        /*  比較演算：発生条件(2) */
ap != bp;                                       /*  比較演算：発生条件(2) */

void        f( enum { C_1, C_2 } *);/*  プロトタイプ宣言：発生条件(2) */

void        func( void )
{
            f( ap );
}
```

- 回避策

  enum 型宣言時にタグ名をつけてください。

```
enum    e_a   { A_1, A_2 }          *ap;        /*  タグ名 e_a を記述する  */
enum    e_b   { B_1A_2 }            *bp;        /*  タグ名 e_b を記述する  */
```

1.1.17. #prgama STRUCT に関する注意事項

「#pragma STRUCT タグ名 unpack」を構造体の配列に対して指定した場合、unpack 指定されたデータを正しくアクセスすることができません。

- 発生条件
  以下 2 点の条件をすべて満たした場合に発生します。
  (1) char 型のメンバを含む構造体の配列を宣言している
  (2) (1)の構造体に対して「#pragma STRUCT タグ名 unpack」を指定している
- 発生例

```
#pragma   STRUCT   S          unpack;          /* 発生条件(2) */

struct   S{
                   char       c;               /* 発生条件(1) */
                   int        j;
};
sturct   S   s[ 3 ] = { { 1,2 },{ 3,4 },{ 5,6 }};
```

- 回避策
  char 型のメンバを含む構造体の配列に対しては「#pragma STRUCT タグ名 unpack」を指定しないでください。

1.1.18. #pragma ADDRESS に関する注意事項

C コンパイラパッケージ V.3.20 Release 1 より#pragma ADDRESS で指定された変数のアドレス値は絶対番地として扱うようになりました。このため、変数のアドレス値をバンク値からの相対番地で記述している場合にはこれを絶対番地に修正してください。

1.1.19. inline 関数に関する注意事項

inline 関数中で分岐を伴う命令を使用した場合 System Error の発生する可能性があります。本現象が発生した場合は、分岐を伴う命令を inline から取り除く、または inline 関数ではなく通常の関数としてご使用ください。

1.1.20. インクルードファイルの検索に関する注意事項

#include の記述において、ドライブ名付きで記述しコンパイル対象となるファイルが存在するディレクトリとは異なったディレクトリからコンパイルした場合、インクルードファイルを検索できない場合があります。

1.1.21. インラインアセンブル機能(#pragma ASM～#pragma ENDASM、asm 関数) に関する注意事項

- #pragam ASM～#pragma ENDASM 内の記述に対して、アセンブル及びリンク時のエラーメッセージの行数、デバッグ情報の行情報等が正常に出力されない場合があります。
- コンパイラは、レジスタや変数の有効範囲について、プログラムフローを解析して処理を行っているため、インラインアセンブル機能(#pragma ASM～#pragma ENDASM または asm 関数)でフローに影響を与えるようなブランチ（条件ブランチ含む）を記述しないようにしてください。
- インラインアセンブル機能を使用してレジスタの値を変更する記述をする場合、有効範囲中でレジスタの値を変更した情報を得ることができません。必ずレジスタを退避・復帰してください。

1.1.22. 前処理命令#define に関する注意事項

マクロ ULONG_MAX と同一値になるマクロを定義する場合は、必ず接尾語 UL を付けてください。

1.2. アセンブラに関する注意事項

1.2.1. セクションが存在しないリロケータブルモジュールファイルのリンクに関する注意事項

セクションが存在しないリロケータブルモジュールファイル（以降リロケータブルファイルと称す）を連続してリンクした場合、"value is undefined" または "Illegal format" エラーが発生し、正しくリンクできないことがあります。

- 発生条件
  以下の条件をすべて満たす場合に発生します。
  (1) リロケータブルファイルが 65 個以上存在する。
  (2) リンク順の 33 番目以降にセクションが存在しないリロケータブルファイル（外部変数や外部関数の宣言のみのファイル等）を連続して 32 個以上リンクしている。

(3) (2)の後にセクションが存在するリロケータブルファイルをリンクしている。

- 発生例

```
>ln79 @cmdfile

<リンク順> <.r79 file>
------------------------
   1        file1.r79
                :
  32        file32.r79
------------------------
  33        file33.r79          |
             :                  |<-- セクションが存在しない
  64        file64.r79          |       32 個のリロケータブルファイル
------------------------
  65        file65.r79
------------------------
```

- 回避策

以下のいずれかの方法で回避してください。

(1) セクションが存在しないリロケータブルファイルが 32 個以上連続しないようにリンク順序を変更する。

(2) セクションが存在しないリロケータブルファイルが 32 個以上連続しないように容量ゼロのセクションを持つリロケータブルファイルを生成し、追加する。

- 容量ゼロのセクションを持つソースファイル例

```
C プログラムの場合   empty.c
------------------------------
#pragma ASM
                    .section empty
#pragma ENDASM
------------------------------
アセンブラプログラムの場合   empty.a79
------------------------------
                    .section empty
                    .end
------------------------------
```

- 発生例の回避例

```
<リンク順> <.r79 file>
------------------------
   1        file1.r79
             :
  32        file32.r79
------------------------
  33        file33.r79
  34        empty.r79          <-- 上記例で生成された、容量ゼロのセクションを持つ
:                                   リロケータブルファイル(empty.r79)を 34 番目に追加
  64        file63.r79
------------------------
  65        file64.r79
  66        file65.r79
------------------------
```

1.2.2. 拡張機能指示命令 "@(文字列の連結)"に関する注意事項

指示命令 "@(文字列の連結)" (以下、"@"と略す)"@" を固定データとして含む文字列を、".BYTE"命令のオペランドに記述すると、"@" が文字列の連結子として扱われる場合があります。

- 発生条件

以下の条件をすべて満たす場合に発生します。

(1) ".BYTE" 命令のオペランドに 2 つ以上の文字列が記述されている。

(2) (1)の文字列で、"@" を含む文字列より前に""(ダブルクォーテーション)"を固定データとして含む文字列が記述されている。

- 発生例

```
.byte   "", "A@B"                   ; "@"が文字列の連結子として扱われ、"", "AB" として処理される
```

以下は発生条件(2)に該当しないので、正常に処理される例です。

```
.byte   "A@B", ""                   ; "@"が固定データとして扱われ、"A@B", "" として処理される
```

- 回避策

"@" が含まれる文字列よりも前に""(ダブルクォーテーション)"を含む文字列を記述する場合、その"@"を含む文字列を分割して次の行に記述してください。

```
. .byte    ""
.byte    "A@B"                          ; "A@B" として処理される。
```

1.2.3. タグジャンプに関する注意事項

統合化開発環境 TM 使用時、アセンブラにて出力されたエラーまたはワーニングに対して、エディタのタグジャンプが機能しない場合があります。

- 発生条件

エラーまたはワーニングが、マクロ処理進行状況"---*---"メッセージと同一行に出力されている場合、問題が発生します。

- 発生例

```
7900 Series Assembler system Version 4.10 Release1
Copyright 2000, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

( test.a79 )
macro processing now
----*----*test.a79 26 Warning (mac79): Actual macro parameters are not enough
test.a79 29 Warning (mac79): Actual macro parameters are not enough
                :
                :
```

上記の場合、"test.a79 26 Warning (mac79): Actual . . ."に対してタグジャンプが機能しません。

- 回避策

以下のいずれかの方法で回避してください。

(1) アセンブラオプション"-." を指定して、画面への "---*---"メッセージ出力を抑止する。 ("-."を指定しても、エラーおよびワーニングメッセージは出力されます)

(2) アセンブラオプション"-T"を指定して、アセンブラエラータグファイルを作成し、そのファイルからタグジャンプする。

1.2.4. 指示命令 ".ORG"に関する注意事項

1 つの CODE(プログラム)セクションに複数の指示命令".ORG"を記述した場合、同一アドレスに複数のプログラムが配置され、不正な動作をする場合があります。

```
                .section    prg, code
                .org        4000H
                bbs         #1122h, extsym1, lab
                movm        extsym2, #3000H
                            :
                            :
                movm        extsym2, #4000H
lab:
                .org        4100H
                lda.W       A, #5000H
                            :
```

- 発生条件

  以下 3 点の条件をすべて満たしている場合に発生します。

  (1) 1つの CODE(プログラム)セクションに複数の指示命令".ORG"を記述している。

  (2) 指示命令".ORG"記述以降に分岐命令またはサブルーチン呼び出し命令が存在している。

  (3) アセンブルにより生成されたオブジェクトコードのロケーションアドレス(LOC.)が、2つ目以降に記述した指示命令".ORG"の指定アドレスと重なっている。

- 発生例

```
SEQ.   LOC.    OBJ.                0XMSJA   ....*....SOURCE STATEMENT....

   3                                           .section    prg, code
   4   004000                                  .org        4000H              –> 発生条件 1
   5   004000   414E0000r221102          J     bbs         #1122H, extsym1, lab –> 発生条件 2
                2003A7F600              J
   6   00400C   9600300000r                    movm        extsym2, #3000H
:
:
  36
  37   0040FD   9600400000r                    movm        extsym2, #4000H      –> 発生条件 3
  38   004102                          lab:
  39
  40   004100                                  .org        4100H              –> 発生条件 1
  41   004100   160050                        lda.W       A, #5000H:
                                  :
```

  上記例では、004100H 番地および 004101H 番地に対してプログラムが重なって配置されます(37 行目と 41 行目に記述された命令)。

- 回避策

  1 つの CODE(プログラム)セクションに複数の指示命令".ORG"を記述する場合、指示命令".SECTION"にて異なるセクション名を定義してから指示命令".ORG"を記述してください。

```
                .section    prg, code
                .org        4000H
                bbs         #1122h, extsym1, lab
movm            extsym2, #3000H
                            :
                            :
                movm        extsym2, #4000H
lab:
.               section     prg_1,CODE          ; 指示命令".SECTION" 定義の追加
.               org         4100H
l               da.W        A, #5000H
                            :
```

1.2.5. マクロ指示命令 ".LOCAL" に関する注意事項

マクロ定義内に、マクロ指示命令 ".LOCAL"と文字列データが記述されている場合、文字列データが正しく展開されな

い場合があります。

- 発生条件

  以下 3 点の条件をすべて満たしている場合に発生します。

  (1) マクロ定義内にマクロ指示命令 ".LOCAL"にてマクロローカルラベルが宣言されている。

  (2) (1)のマクロ定義内に文字列データの記述がされている。

  (3) (2)で記述した文字列データの末尾が ".(ピリオド)" である。

- 発生例

[アセンブリ言語ソースファイル]

```
mac     .macro
        .local      btop, bend          ; 発生条件(1)
btop:
        .byte       bend - btop
        .byte       "string."           ; 発生条件(2)、(3)
bend:
        .endm

        .section    prg,code
        mac
        .end
```

[発生例のマクロ展開結果]

```
        .local      btop, bend
..ml0001:
        .byte       ..ml0002 - ..ml0001
        .byte       "stringstring"      <-- 誤った展開を実施
..ml0002:
        .endm
```

- 回避策

  特種文字 "." とそれより前の文字列データを分割し、"."を即値にて定義してください。

```
mac     .macro
        .local      btop, bend
btop:
        .byte       bend - btop
        .byte       "string", 2EH       ; "." を即値で定義
bend:
        .endm
```

1.2.6. 行連結機能に関する注意事項

マクロ呼び出しの引数に行連結機能を使用している場合、その行連結以降の行情報がソース行番号と一致しなくなります。

(1) エラーが発生した場合、エラーが発生したソース行番号とは異なる行番号がエラーメッセージに出力されます。このため、エディタ等のタグジャンプ機能を使用した場合にエラーが発生したソース行に移動できません。

(2) シミュレータデバッガおよびデバッガにおいて、正しくソースレベルデバッグを行うことができません。

- 記述例

```
[ プログラム：test.a79 ]
[行番号]    [ソース]
 2          macTEST  .macro      _a, _b, _c
 3                   .byte       _a
 4                   .byte       _b
 5                   .byte       _c
 6                   .endm
 7
 8          macTEST  1, ¥¥                ; マクロ呼び出しの引数で行連結機能を使用
 9                   2, ¥¥
10                   3
11Ayte      err      ; エラーが発生する行

[ エラー発生例 ]
test.a79   9   Error(asp79): Undefined symbol exist ‘err’
```

- 回避策
  マクロ呼び出しの引数には行連結を使用しないようにしてください。

1.2.7. 指示命令".INCLUDE"に関する注意事項

以下の発生条件に該当するソースファイルをアセンブルした場合、"Can't open include file."エラーが発生する場合があります。

(1) 指示命令".INCLUDE"のオペランドに相対パス付きのインクルードファイル名が指定されている
(2) ソースファイルが、起動ディレクトリおよびインクルードファイルの存在するディレクトリと異なるディレクトリに存在する

- 発生例

```
[ ソースファイル例 ]
include     inc¥b.inc                   ; 発生条件(1)

[ ファイル構成例 ]
¥project ----   ¥work
          |
          ---- ¥src ---- a.a79          ; ソースファイル：発生条件(2)
                    |
                    ------ ¥inc ---- ¥b.inc     ; インクルードファイル：発生条件(2)

[ アセンブラ実行例（起動ディレクトリが¥project¥work の場合）]
C:¥> as79 ..¥src¥a.79
                    :
     Can’t open include file            エラー発生
```

- 回避策
  指示命令".INCLUDE"のオペランドの先頭に".¥"を付加し".¥inc¥b.inc"のように記述してください。

1.2.8. ロケーションシンボル"$"の記述に関する注意事項

無条件分岐命令(BRA)のオペランドにロケーションシンボル"$"を記述したソースファイルをアセンブルした場合、アセンブラが強制終了します。

- 発生例

```
            .section    program
            bra         $
            .end
```

- 回避策
  オペランドにロケーションシンボル"$"を記述する場合の無条件分岐命令は BRAL、JMP または JMPL を使用

してください。

1.2.9. ロードモジュールコンバータでコマンドオプション"-O"の使用に関する注意事項

ロードモジュールコンバータの「出力ファイル名を指定する」オプション"-O"を使用する場合、指定するファイル名に".(ピリオド)"が含まれるとファイル拡張子のないファイルを生成する場合があります。

- 発生例

```
> lmc79 -o ..¥output¥sample sample.x79
```

上記例では、ファイル名"sample.mot" が "sample"として生成されます。

- 回避策

".(ピリオド)"を含む出力ファイル名を指定する場合は、出力ファイル名に拡張子".mot"または".hex"を付加してください。

```
> lmc79 -o ..¥output¥sample.mot sample.x79              ; .mot を付加
```

1.3. 機種依存に関する注意事項

1.3.1. 7900 シリーズの機種依存部に関する注意事項

(1) JSR、JMP、RTS 命令をバンクの最上位番地、またはバンク境界にまたがって配置しないように注意してください。このような配置の可能性がある場合は、リンク時にリンケージエディタのオプション"-C"を指定してください。このオプションを指定すると JSR、JMP、RTS 命令がバンク境界に配置される場合にワーニングメッセージを出力します。

(2) SFR 領域のレジスタに書き込む、または呼び出す場合には特定の命令を使用しなければならないことがあります。この特定命令は機種毎に異なりますので、詳しくは各機種のユーザーズマニュアルを参照してください。この注意事項に関わる命令は asm 関数を使用してプログラム中に直接命令を記述してください。

1.3.2. SFR 領域のアクセスに関する注意事項

SFR 領域のレジスタをアクセスする場合には特定の命令を使用しなければならないことがあります。
この特定の命令は機種毎に異なりますので、詳しくは各機種のユーザーズマニュアルなどを参照してください。この注意事項に関わる命令は、asm 関数等のインラインアセンブル機能を使用して、プログラム中に命令を直接記述してください。

1.4. TM に関する注意事項

- 統合化開発環境 TM を使用する場合、V.3.00 またはそれ以降のバージョンをご使用ください。本バージョンの C コンパイラパッケージは、V.2.01 以前の TM ではご使用になれませんので、ご注意ください。
- TM V.3.00 以降の使用において、TM V.2.01 以前のバージョンで作成したプロジェクトを流用する場合に、コンパイルオプション、アセンブルオプションの '-finfo' が有効になりません。個別に '-finfo' オプションを追加してください。詳しくは、TM のリリースノートを参照してください。

1.5. MS-Windows に関する注意事項

1.5.1. 動作環境に関する注意事項

(1) C コンパイラパッケージは、Windows 95、Windows NT 4.0 以降の環境で動作します。Windows 3.1 および Windows NT 3.5x 以前のバージョンでは動作しません。

(2) 日本語 Windows NT 環境でコマンドプロンプトのサイズが「80 x 25」以外に設定されている場合、製品を起動するとコマンドプロンプトのサイズが頻繁に切り替わります。コマンドプロンプトのサイズは「80 x 25」に設定してください。

1.5.2. ファイル名に関する注意事項

ソースプログラムファイルの名前や作業を行うディレクトリ名は、次の注意事項に従ってください。

- 漢字を含むディレクトリ名、ファイル名は使用できません。
- ファイル名に使用するピリオド（.）は一つのみ使用可能です。
- ネットワークパス名は使用できません。ドライブ名に割り当ててご使用ください。
- 「ショートカット」は使用できません。
- 空白文字を含むディレクトリ名、ファイル名は使用できません。
- "..."表記を用いて 2 つ以上のディレクトリを指定することはできません。
- パス指定を含めたファイル名の長さが 128 文字以上になるものは使用できません。

1.5.3. ウィルスチェックプログラムに関する注意事項

ウィルスチェックプログラムが常駐した状態で C コンパイラパッケージを起動すると正常に起動しない場合があります。その場合は、ウィルスチェックプログラムの常駐を解除してから C コンパイラパッケージを起動しなおしてください。

1.5.4. バージョンアップするときの注意事項

C コンパイラパッケージをバージョンアップする場合は、あらかじめ、インストールされている C コンパイラパッケージをアンインストールしてから、新しいバージョンをインストールしてください。

- C コンパイラパッケージのアンインストール手順

  C コンパイラパッケージをアンインストールするには、「コントロールパネル」- 「アプリケーションの追加と削除」を選択しアンインストールを実行してください。

2. C コンパイラパッケージのインストール

2.1. インストールを始める前に

C コンパイラパッケージのインストールを始める前に次の内容をご確認ください。

- 本製品の「使用権許諾契約書」、「リリースノート」などをよくお読みください。製品をインストールした場合は、契約書の記載内容に同意されたものとみなします。
- C コンパイラパッケージを快適に使用するには、32M バイト以上のメモリと 20M バイト以上の空きハードディスク領域が必要です。
- 製品のインストールは専用のインストーラを使用してください。
- インストールの途中でライセンス ID を入力する必要があります。インストールを始める前にライセンス ID を確認してください。

2.2. インストーラ

インストーラは次に示す環境毎に用意されています。ご購入になった製品を確認の上、該当するインストーラを使用してください。

- 日本語環境

対応ホスト	対応 OS	インストーラ名	CD-ROM 上のディレクトリ
IBM[1] PC/AT互換機	Microsoft Windows[2] 98 Microsoft Windows Me Microsoft Windows NT Microsoft Windows 2000 Microsoft Windows XP	SETUP.EXE	¥NC79WA¥W95J

- 英語環境

対応ホスト	対応 OS	インストーラ名	CD-ROM 上のディレクトリ
IBM PC/AT 互換機	Microsoft Windows 98 Microsoft Windows Me Microsoft Windows NT Microsoft Windows 2000 Microsoft Windows XP	SETUP.EXE	¥NC79WA¥W95E

2.3. インストール手順

次の手順でインストールしてください。

(1) CD-ROM 上の対象製品のインストーラが配置されているディレクトリに移動します。
(2) インストーラを起動して表示されるメッセージにしたがってインストールを完了してください。

2.4. インストール後の環境設定

インストールが完了した後、次の環境変数を設定してください。
表中の「自動」は、インストーラが AUTOEXEC.BAT を書きかえます。したがって、デフォルトでインストールを実行した場合は、AUTOEXEC.BAT を書きかえる必要はありません。

環境変数	設定例
BIN79	自動 (SET BIN79=C:¥MTOOL¥BIN)
INC79	自動 (SET INC79=C:¥MTOOL¥INC79)
LIB79	自動 (SET LIB79=C:¥MTOOL¥LIB79)

[1] IBM およびAT は、米国International Business Machines Corporation の登録商標です。
[2] Microsoft、WindowsおよびWindows NTは、米国Microsoft Corporationの米国およびその他の国における登録商標です。

TMP79	自動 (SET TMP79=C:¥MTOOL¥TMP)
NCKIN	SET NCKIN=SJIS
NCKOUT	SET NCKOUT=SJIS
コマンドパス	自動 (C:¥MTOOL¥BIN を追加)

3. ユーザ登録

バージョンアップ情報や技術サポートなどのサービスを受けるためにユーザー登録を行ってください。ユーザー登録をされていない場合は、これらのサービスを受けることができません。
また、ユーザー登録は**ご購入後 30 日以内**に行ってくださるようお願い申し上げます。

3.1. ユーザー登録方法

C コンパイラパッケージをインストールすると以下のファイルが生成されます。

¥ mtool ¥ support ¥ nc79wa ¥ regist.txt

¥ mtool はデフォルトでインストールした場合のディレクトリです。
regist.txt のファイル内容をすべてカット&ペーストして以下の電子メールアドレス宛に送付してください。

regist_tool@renesas.com

4. バージョンアップ内容

4.1. コンパイラ機能追加

4.1.1. 拡張機能 ”#pragma TBLJMPOFF” の追加

本機能で指定された関数に対してはコンパイルオプション"-fswitch_table[-fST]"指定時でもテーブルジャンプコードを生成しません。

4.1.2. コンパイルオプション “-dsource_in_list[-dSL]” の追加

C ソースをコメントとして付加したリストファイル(.lst)を生成します。

4.1.3. コンパイルオプション ”-ferase_static_function=static 関数名[-fESF=static 関数名]”の追加

指定された static 関数に対するコード生成を行いません。

- 備考
  本オプションは、コンパイルオプション ”-Wno_used_static_function[-WNUSF]”で検出した static 関数に対して指定してください。

4.1.4. コマンドオプション “-Wno_used_argument[-WNUA]” の強化

関数定義中で未使用の仮引数がある場合に警告を出力します。

4.2. アセンブラ・リンケージエディタ機能追加

4.2.1. リンケージエディタの機能追加

コンパイルオプション"-fswitch_table[-fST]"指定時に生成されるテーブルジャンプコードがバンク境界をまたがっている場合、エラーメッセージ"SWITCH statement is crossing bank."を出力するようにしました。

4.3. コンパイラ仕様変更

コンパイルオプション"-fswitch_table[-fST]"指定時に生成されるテーブルジャンプコードがバンク境界をまたがっている場合、リンク時にエラーにするための機能を追加しました。

4.4. コンパイラ問題点修正

ツールニュースでお知らせの以下の問題点を修正しました。

- 関数へのポインタに関する注意事項
- 浮動小数点定数の最適化に関する注意事項
- unsigned long 型定数の最適化に関する注意事項
- 浮動小数点定数を unsigned long 型に型変換する場合の注意事項
- 条件分岐の条件式として、複数のビットフィールドの評価式を論理 AND 演算子(&&)で連結した式を記述している場合の注意事項
- DPnDATA 宣言ユーティリティに関する注意事項

4.5. アセンブラ・リンケージエディタ問題点修正

ツールニュースでお知らせの以下の問題点を修正しました。

- リンケージエディタが表示するトータル ROM サイズに関する注意事項

5. ソフトウェアのバージョン一覧

C コンパイラパッケージ V.4.10 Release 1C に含まれているソフトウェアの各バージョンは以下のとおりです。

- nc79 V.1.41.00
- cpp79 V.4.30.00
- ccom79 V.4.10.00
- as79 V.4.10.01
- mac79　V.3.20.01
- pre79 V.3.00.01
- asp79 V. 4.10.00
- ln79 V. 4.10.00
- lb79 V. 1.00.02
- lmc79 V. 3.20.00
- xrf79 V. 1.00.10
- abs79 V. 3.00.05
- stk79 V.1.00.00
- utl79 V.1.00.05
- sc79 V.1.00.00
- Map Viewer V.2.00.01
- Stk Viewer V.1.00.01

6. 7700 ファミリ 79xx シリーズ用リアルタイム OS の対応バージョン

本製品は、7700 ファミリ 79xx シリーズ用リアルタイム OS V.2.20 Release 1 に対応しています。本製品と組合せてリアルタイム OS をご使用になる場合には、上記バージョンをご使用ください。