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。

RJJ10J0593-0300

＝＝＝　必ずお読み下さい　＝＝＝

77xx シリーズ用 C コンパイラ

M3T–NC77WA V.5.20 Release 4B リリースノート（第 15 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 M3T–NC77WA V.5.20 Release 4B をお買い上げいただきまして誠に有難うございます。本資料は M3T–NC77WA(以下 NC77WA) 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。

★本製品は TM V.3.xx **用の情報を書き込みます。本製品使用後に** TM V.1.xx・V.2.xx **を使用される場合は、必ず本製品をアンインストールして下さい。**

★本製品は TM V.3.xx **とのみご使用頂けます。**TM V.1.xx・V.2.xx **は使用できません。**

目次

1　お問合せ先について

1.1　社名変更、お問合せ先変更のご案内

2003 年 4 月 1 日、三菱電機セミコンダクタ・アプリケーション・エンジニアリング株式会社は、株式会社ルネサス ソリューションズに社名変更いたしました。ドキュメントに記載されている旧メールアドレスおよび URL は以下のとおり最新のものに読み替えをお願いいたします。

- ユーザ登録窓口
  旧：　regist@tool.mesc.co.jp
  　　regist@tool.maec.co.jp
  新：　regist_tool@renesas.com
- ツール技術サポート窓口
  旧：　support@tool.mesc.co.jp
  　　support@tool.maec.co.jp
  新：　csc@renesas.com
- ツールホームページ
  旧：　http://www.tool-spt.mesc.co.jp/
  　　http://www.tool-spt.maec.co.jp/
  新：　http://japan.renesas.com/tools
- 社名
  旧：　三菱電機セミコンダクタソフトウエア (株)
  　　三菱電機セミコンダクタシステム (株)
  　　三菱電機セミコンダクタ・アプリケーション・エンジニアリング (株)
  新：　株式会社ルネサス ソリューションズ

1.2　最新情報＆ FAQ のご案内)

本製品の情報については以下を参照してくださるようお願いします。
http://japan.renesas.com/tools

2　製品内容

3　ソフトウェアのバージョン一覧

NC77WA　V.5.20 Release 4B に含まれているソフトウェアの各バージョンは以下の通りです。

- nc77 V.4.01.00
- cpp77 V.4.01.00
- ccom77 V.2.04.02(NC_CORE Version 2.02.04)
- loop77 V.1.20.00
- s2ie V.1.33.00
- stk77 V.1.21.00

- rasm77 V.5.10.00
- pre77 V.5.00.00
- br77 V.1.00.00
- link77 V.2.10.00
- lib77 V.5.00.00
- crf77 V.2.10.10
- hextos2 V.2.00.00
- lst77 V.1.00.10
- xref V.1.10.00

4 動作環境

4.1 動作確認環境

NC77, RASM77 の動作を確認しているホストマシン、および OS のバージョンを以下に示します。

ホスト名	OS のバージョン	CD–ROM のディレクトリ
IBM[*1] PC/AT 及び互換機	Microsoft[*5] Windows[*2] 98	W95J, W95E
	Microsoft Windows Me	W95J, W95E
	Microsoft WindowsNT Workstation 4.0	W95J, W95E
	Microsoft Windows 2000	W95J, W95E
	Microsoft Windows XP	W95J, W95E

上記以外のホストおよび OS 上での動作については、ホストマシンおよび OS の供給メーカーに依存しますので、上記条件で動作するソフトウェアがお客様のホストマシンおよび OS で動作するかどうかを供給メーカーにお問い合わせ願います。

4.2 関連製品の対応バージョン

4.2.1 RASM77 の対応バージョンについて

NC77WA V.5.20 Release 4B は、RASM77 V.5.10 以上のみと組み合わせてご使用頂けます。

4.2.2 M3T–MR7700 の対応バージョンについて

NC77WA V.5.20 Release 4B は、M3T–MR7700 V.3.10 Release 3 のみと組み合わせてご使用頂けます。

[*1] IBM 及び AT は、米国 International Business Machines Corporation の登録商標です。
[*2] Microsoft、MS、MS–DOS、Windows、WindowsNT 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。

5 製品のインストール

5.1 インストールを始める前に

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。
- M3T-NC77WA 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。
- RASM77 のライセンス ID では、NC77WA はインストールできません。

5.2 インストーラの格納場所

インストーラは次に示す環境 (対応ホスト、対応 OS 、言語) 毎に用意しています。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。

- 日本語環境用のインストーラ

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 Windows95 Microsoft Windows 98 Microsoft Windows Me Microsoft Windows NT Microsoft Windows 2000 Microsoft Windows XP	setup.exe	¥NC77WA ¥W95J
SPARC station	Solaris 2.x	setup	/nc77wa/solaris
HP9000/700	HP-UX 10.2x	setup	/nc77wa/hp700

- 英語環境のインストーラ

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 Windows95 Microsoft Windows 98 Microsoft Windows Me Microsoft Windows NT Microsoft Windows 2000 Microsoft Windows XP	setup.exe	¥NC77WA ¥W95J

5.3 インストール手順

5.3.1 PC 版のインストール

注意事項：

- 空白を含む名前は、インストール先ディレクトリとして指定できません。
- インストール中の「コンポーネントの選択」画面で「インストール先ディレクトリ」項目を「参照 (R)」を使用して変更する場合、「ディレクトリの選択」画面の「ドライブ (V)」で表示されるドライブの幾つかが選択できなくなることがあります。この時は「コンポーネントの選択」画面に戻り、その中の「ディスク容量 (S)」でドライブを指定してください。それでも選択できない場合はインストールを中止・OS を再起動してから、再度インストーラを実行してください。

インストール手順：

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作業を行ってください。

インストールの途中で入力するデータは、ユーザー登録用ファイルを作成するためのみに使用します。

5.4 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。

5.4.1 PC 版の環境設定

表中の「自動」と書かれた箇所は、インストール時に「autoexec.bat を書き換える」を選択すると、その内容をインストーラが autoexec.bat に書き込みます。したがってインストール時に「autoexec.bat を書き換える」を選択した場合（インストーラのデフォルト動作）は、改めて設定する必要はありません。

環境変数	設定例
BIN77	自動 (SET BIN77=C:¥MTOOL¥BIN)
INC77	自動 (SET INC77=C:¥MTOOL¥INC77)
LIB77	自動 (SET LIB77=C:¥MTOOL¥LIB77)
TMP77	自動 (SET TMP77=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)

5.4.2 漢字コード設定

環境変数 NCKIN と NCKOUT には、NC77WA を実行する場合の漢字コードを設定します。

■NCKIN　入力コード系（C プログラム中に記述されている文字コード）に対する漢字コードを指定します。次のどちらかを指定します。

- EUC

入力文字コードを EUC と解釈します。ただし、1 文字が 3 バイトで構成される文字（外字など）は使用できません。

- SJIS
  入力文字コードをシフト JIS と解釈します。

環境変数 NCKIN を設定しないでコンパイルした場合のデフォルト値は次の通りです。

ホスト	ディフォルト値
PC 版	SJIS

■NCKOUT　出力コード系（アセンブリファイルに出力される文字コード）に対する漢字コードを指定します。次のどちらかを指定します。意味は NCKIN と同じです。

- EUC
- SJIS

環境変数 NCOUT を設定しないでコンパイルした場合のデフォルト値は次の通りです。

ホスト	ディフォルト値
PC 版	SJIS

日本語および英語以外（たとえばヨーロッパ語）でコンパイラを使用される場合は、必ず *EUC* を指定してください。

5.5　freeware に格納されているプログラムについて

製品 CD-ROM の freeware ディレクトリには、製品の機能を補足する機能を持つプログラムが格納されています。これらのプログラムはインストーラではインストールしません。各プログラムごとに CD-ROM から直接インストールしてください。インストール方法および機能の詳細については、各プログラムごとのドキュメントファイルを参照してください。

chk77　複数ファイル間での関数の引数、戻り値および外部変数の型整合性を検査します。
dbsym77　オプション “-g” を使用したときに生成されるシンボルファイルから、コンパイラが生成するローカルシンボルを削除します。これによりシンボルファイルのサイズを削減することができます。
dump695　IEEE-695 フォーマットのオブジェクトモジュール、およびロードモジュール内の情報を、テキストファイルとして出力します。
sfr77　特殊機能レジスタ (SFR) に対して不正な命令でアクセスしている場合に警告メッセージを出力します。
lst7　アセンブラが生成するリストファイルのアドレスデータを、リンク後の絶対番地に書き換えます。
xref77　ローカルおよびグローバルシンボルの相互参照リストを生成します。

freeware は動作保障、保守は原則として行いません。

6　ユーザー登録

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、登録はご購入後 *30* **日以内**に登録してくださるようお願いいたします。

6.1 PC 版のユーザー登録方法

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 ¥nc308wa ¥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 の内容を全てそのまま、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

7 バージョンアップレポート

7.1 NC77 V.5.10 Release1 からの変更点

7.1.1 最適化オプションの追加

以下のオプションを新たに追加しました。

■–Ono_logical_or_combine(–ONLOC)

- 概要
  「論理 OR で連結した比較条件をまとめる最適化」を抑止します。
- オプション名
  –Ono_logical_or_combine(–ONLOC)

■–Ocompare_byte_to_word(–OCBTW)

- 概要
  連続したバイト領域の比較を、ワードで比較します。
- オプション名
  –Ocompare_byte_to_word(–OCBTW)

7.1.2 警告オプションの追加

以下のオプションを新たに追加しました。

■–Wlarge_to_small(–WLTS)

- 概要
  暗黙の型変換に対するオプション–Wlarge_to_small(–WLTS) を追加しました。
- オプション名
  –Wlarge_to_small(–WLTS)
- 機能
  以下の場合にワーニングを発生します。

– 「int = long;」のように型が小さくなる方向への代入
– 関数引数が小さい型である場合、関数呼び出し時に大きな型を入れる

- 注意事項
  オプション–Wall を指定しても、本オプションは有効になりません。これは本オプションを使用する事により、極めて多量のワーニングが発生する可能性があるためです。
  NC77 が持つ全ての警告機能を有効にするためには「–Wall, –WLTS」を同時に指定してください。

■–Wuninitialize_variable (–WUV)

- 概要
  「初期化していない auto 変数」を参照した場合に警告を出します。
- オプション名
  –Wuninitialize_variable (–WUV)
- 使用例
  以下のプログラムをコンパイルすると、警告を出します。

```
1: int x;
2: func()
3: {
4:          int a;
5:          x = a;
6: }

[Warning(ccom):sample.c,line 6] uninitialized variable 'a'
```

- 注意事項
  ワーニングメッセージの行番号は、関数の最後の行番号 (通常は} の行) を表示します。
  また if、for 等による条件分岐中で変数を初期化する場合、初期化されていないとみなして警告を出力します。

```
例)
main()
{
        int i;
        int val;
        for(i = 0; i < 2; i++){
                f();
                val = 1; // 論理上、ここで必ず初期化される
        }
        ff( val );
}
```

7.1.3 #pragma MX1FUNCTION の追加

1. 概要

   #pragma M1FUNCTION 同様の関数操作である、#pragma MX1FUNCTION を追加しました。指定された関数は M,X フラグが共に 1 として関数呼び出しを行います。

2. 詳細

   - プラグマ名

     #pragma MX1FUNCTION

   - 機能

     関数呼び出し、及び定義時のデフォルトを「M フラグ=1」かつ「X フラグ=1」にします。

7.1.4 asm() 関数の変数指定に＄@を追加

1. 概要

   asm() 関数で引数を解釈するフォーマットに「$@」を追加しました。

2. 詳細

   - フォーマット

     ```
     $@
     ```

   - 機能

     引数が auto 変数、near(far) 型外部変数のいずれかを自動的に判断し、DP:, DT:, LG:を付けた形でオフセット (シンボル名) を出力します。

   - 記述例

     ```
     asm("    lda.W  A,$@", mem_i); // DP, DT, LG 不要
     ```

7.1.5 改修内容

ツールニュースでお知らせした以下の注意事項を改善しました。

TOOL NEWS	項目
1999 年 5 月 16 日 (MESCT–NC77–990516D)	初期値を持つ far 型配列に関して
	sizeof 演算子に関して
1999 年 6 月 16 日 (MESCT–NC77WA–990616D)	関数引数へのアクセスについて
	s2ie が正常終了しない場合があります
1999 年 8 月 1 日 (MESCT–NC79WA_2–990801D)	標準ライブラリ関数 longjmp 実行中に割り込み処理が発生する場合
1999 年 10 月 16 日 (MESCT–NC79WA–991016D)	整定数をポインタにキャストして演算を行った場合

7.2 NC77 V.5.20 Release1 からの変更点

7.2.1 改修内容

ツールニュースでお知らせした以下の注意事項を改善しました。

TOOL NEWS	項目
2000 年 1 月 16 日 (MESCT–NC77WA–000116D)	float 型または double 型の外部変数を初期化する際に、 正しくデータが設定されない場合があります。
2000 年 2 月 1 日 (MESCT–NC77WA_1–000201D)	const 修飾子で指定された変数を記述したプログラムを コンパイルすると System Error が発生する場合があります。

7.3 NC77 V.5.20 Release2 からの変更点

7.3.1 改修内容

ツールニュースでお知らせした以下の注意事項を改善しました。

TOOL NEWS	項目
2000 年 4 月 1 日 (MESCT–CC32R–000401D)	root 権限でインストールした場合、一般ユーザーで 製品にアクセスできない可能性があります。
2000 年 4 月 1 日 (MESCT–NC77WA–000401D)	同一ファイル内に同一セクションを連続して定義した場合、 その後に定義したセクションが LST77 でエラーとなることがあります。
2000 年 5 月 1 日 (MESCT–NC77WA–000501D)	最適化オプションを使用した場合に、Windows エラー メッセージ「不正な処理を行ないました」が出力されることがあります。
2000 年 6 月 16 日 (MESCT–RASM77–000616D)	コマンド入力時に、変換対象ファイル名を指定せずに オプションだけを指定した場合、LST74 が強制終了します。
2000 年 7 月 16 日 (MESCT–NC77_1–000716D)	浮動小数点定数の畳み込みで、減算が不正な値になることがあります。
2000 年 7 月 16 日 (MESCT–NC77_1–000716D)	unsigned long 型定数の畳み込みで、除算・剰余算が不正な値に なることがあります。

7.3.2 TM との連携機能

TM V.2.xx と NC77WA の組合わせで、構造化記述アセンブリファイル（拡張子.p77 のファイル）からリロケータブルオブジェクトファイルを生成できるようになりました。

7.4 NC77 V.5.20 Release3 からの変更点

7.4.1 改修内容

ツールニュースでお知らせした以下の注意事項を改善しました。

TOOL NEWS	項目
2000 年 11 月 16 日 (MESCT–NC77WA–001116D)	”if - else if”選択文の”if”および”else if”条件式にビットフィールドを使用した場合のコード生成に関する注意事項

7.4.2 TM との連携機能

TM 最新版との対応状況は、TM のリリースノートをご確認下さい。

7.5 RASM77 V.5.00 からの変更点

7.5.1 RASM77 の変更点

- 条件アセンブル偽部分に記述されたマクロ引数を展開してしまう問題点を修正しました。

7.5.2 LINK77 の変更点

- –BRAL コマンドオプションを追加しました。
  飛び先番地が xxFFFFh 番地となる BRAL 命令の命令コードを検出した際には、ワーニングメッセージを表示します。なお、BRAL 命令の検出は機械語コードパターンにより検出します。したがって、該当するデータが実際に BRAL 命令記述によるものであるかどうかはソースプログラムで確認して下さい。
- ワーニングメッセージを追加しました。

  ```
  WARNING No.1: BRAL specified address is xxFFFFh(description address xxxxh)
  ```

  xxxxh 番地に、飛び先が xxFFFFh となる BRAL 命令が配置されている可能性があります。

7.5.3 BR77 の追加

- 分岐最適化ツール br77 を bin ディレクトリに追加しました。
  br77 は、分岐命令の分岐先アドレスが範囲外のためアセンブラでエラーとなる分岐命令を分岐可能な命令に変換するツールです。

7.5.4 LST77 の変更点

- 同一ファイル内に同一セクション名が連続して記述されていると、その後に記述されたセクションが LST77 でエラーとなる場合がある問題点を修正しました。
- 変換対象ファイル名を指定せずオプションを指定すると LST77 が強制終了する問題点を修正しました。

8 注意事項

8.1 使用上のご注意

8.1.1 X フラグの整合性に関する注意事項

1. 概要

   for 文等の、後方分岐を伴う処理に対して X フラグの整合性がとれない場合があります。

2. 発生条件

   以下の条件を全て満たす場合に発生する可能性があります。

   （a）for・while・do-while・goto（後方分岐のみ）の範囲内（以下、後方岐のブロック）に、char 型データとそれ以外の型のデータが混在している。

   （b）(1) の後方分岐のブロックに入った直後に、X(Y) レジスタと定数値を比較するコードがコンパイラから生成される。（発生例では、s.ch1 のオフセット値に X レジスタを使用）

   （c）(1) の後方分岐のブロック末尾で、X(Y) レジスタと定数ゼロとを比較するコードがコンパイラから生成される。（発生例では、uc[i] のオフセット値に X レジスタを使用）

   （d）(3) の条件式の結果により、X フラグが 0・1 のどちらになるかが異なる状態で、後方分岐のブロック先頭にジャンプする。

3. 発生例

```
/* この例ではコンパイルオプション -OS 指定時に発生します */
struct S{
    unsigned char   ch1;
    unsigned char   ch2;
} s;

unsigned char   uc[5];

void func(void)
{
    int i, j;

    for( i=0; i<2; i++ ){
        if( s.ch1 == 1 ){                   /* 発生条件 (2) */
            uc[i] = 1;
        }

        if( (uc[i] == 5) || (uc[i] == 0) ){ /* 発生条件 (3) */
            uc[i] = 0;
        }
        /* 発生条件 (4)
         * if が成立すれば「X フラグ = 1」、不成立ならば
         * 「X フラグ = 0」の状態で分岐する。
         */
    }
}
```

4. 回避策

本現象が発生した場合には、後方分岐のブロック末尾に asm 関数を入れることで、X フラグを強制的に変更してください。

```
struct S{
    unsigned char   ch1;
    unsigned char   ch2;
} s;

unsigned char   uc[5];

void func(void)
{
    int i, j;

    for( i=0; i<2; i++ ){
        if( s.ch1 == 1 ){
            uc[i] = 1;
        }

        if( (uc[i] == 5) || (uc[i] == 0) ){
            uc[i] = 0;
        }
        asm(0,0);  /* 追加 */
    }
}
```

8.1.2 unsigned long 型の除算・剰余算に関する注意事項

1. 概要

unsigned long 型変数で除算、または剰余算を行なう場合に、X フラグをクリアしないで除算（剰余算）のサブルーチンを呼び出す場合があります。

2. 発生条件

以下 5 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。

（a）被除数は変数である。

（b）除数は定数である。

（c）被除数、除数のどちらか一方、または両方が unsigned long 型である。

（d）(1) の直前で「引数が char 型は配列」を使用する。

（e）「(4) の配列」の添字を、繰り返し文の条件判定に使用する。

※条件をすべて満たす場合でも、C ソースの構文次第では問題が発生しない場合があります。発生しない場合はまったく問題ありません。

3. 発生例

```
void func(unsigned char ch, unsigned long *array)
{
    unsigned long tmp;

    while( ch-- != 0 ){          /* 発生条件 (4), (5) */
        array[ch] = tmp / 10;    /* 発生条件 (1), (2), (3) */
    }
}
```

4. 回避策

本現象が発生した場合には、配列の添字を char(signed char,unsigned char) 型以外の型で記述してください。

```
void func(unsigned char ch, unsigned long *array) /* 変更 */
{
    unsigned long tmp;

    while( ch-- != 0 ){
        array[ch] = tmp / 10;
    }
}
```

8.1.3 整数型配列の初期設定に関する注意事項

1. 概要

整数型配列の初期設定値が、正しくコード生成されない場合があります。

2. 発生条件

以下 4 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。

条件をすべて満たす場合でも、C ソースの構文次第では問題が発生しない場合があります。発生しない場合は全く問題ありません。

(a)「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。

(b) (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。

(c) (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。

(d)「(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。

※マイナス記号やキャスト等も演算子の一種です。

3. 発生例

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

[例 2]
  int i;
  unsigned long addr1[] = {0,(unsigned long)&i,77};/* NG */
  unsigned long addr2[] = {(unsigned long)0,
                           (unsigned long)&i,77}; /* OK */
```

4. 回避策

本現象が発生した場合には、配列初期値の第一要素にキャスト演算子を追加してください。

```
[発生例 1 の回避策]
  int array1[] = {(int)1,-5,302};    /* キャスト演算子"int"を追加 */

[発生例 2 の回避策]
  int i;
  unsigned long addr1[] = {(unsigned long)0,
                           (unsigned long)&i,77};
                          /* キャスト演算子"unsigned long"を追加 */
```

8.1.4 標準ライブラリ pow 関数の不正な定義域エラーに関する注意事項

1. 概要

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。

2. 発生条件

以下 2 点の条件のいずれか 1 つを満たす場合に発生します。

(a) 第一引数がゼロかつ第二引数が正である。

(b) 第一引数が非ゼロかつ第二引数が負である。

3. 発生例

```
include <math.h>
include <errno.h>

ouble  ans1;

nt func(void)

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

4. 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、ライブラリを再作成して、ユーザプログラ

ムをリンクし直してください。なお、ライブラリ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 c. 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。
修正個所は、以下の通りです。ライブラリソースファイル”pow.c” の ”pow”関数の最初の if 文（先頭から 5 行目前後）
[正] `if ((x == 0 && y <= 0) || (x < 0 && y != (int)y)) {`
[誤] `if (x == 0 || y <= 0 || (x < 0 && y != (int)y)) {`

8.1.5 switch 文を含むループに関する不正な最適化に関する注意事項

1. 概要
   switch 文を含むループを記述すると、コンパイル時に不正な最適化が行われ、移動してはいけない命令を、ループの前に移動することがあります。
2. 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。
   条件をすべて満たす場合でも、C ソースの構文次第では問題が発生しない場合があります。発生しない場合は全く問題ありません。
   （a）コンパイル時に”-OS”オプションを指定している。
   （b）”-fST”オプションを指定している。
   （c）for または while 文の中に switch 文が存在する。
   （d）自動変数、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。
   （e）(4) で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。
   （f）コンパイルによって (3) の 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。
   補足:
   分岐先テーブルとは、命令中に埋め込まれた分岐先アドレスの並びです。コンパイラは、分岐先テーブルの参照による間接分岐のほうが効率がよいと判断した場合、テーブルから分岐先アドレスを取得し間接分岐するコードを生成します。
   この判断は、switch 文の case ラベルのラベル数、値の範囲、連続性などに基づいて行われます。
3. 発生例

[C 言語ソースファイル]

```
int func(void)
{
    int i, a, flg = 0;          /* 発生条件 (4) */
         for (i = 0; i < 1; i++) {
        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される */
            break;
        }
    }
    return flg;
}
```

[コンパイル結果の例 (抜粋)]

```
;## # C_SRC :            for (i = 0; i < 1; i++) {
        ldm.W   #0000H,DP:3     ;  i      <== for 文 の i = 0
        ldx     DP:1    ;  a
        ldm.W   #0001H,DP:1     ;  flg    <== 不正に移動されている
L0:                                       <== 反復処理の先頭
        .cline  6
;## # C_SRC :                   switch (a) {
        txa
        sec
        sbc.W   A, #0008H               ; minimum case
        cmp.W   A, #0009H               ; case width check
        bcc             M4
        lda.W   A,#0008H
M4:
        asl             A
        tax
        jmp             (L16,X)
L16:                                      <== 分岐先テーブルの先頭
        .word   offset  L7
        .word   offset  L7
        .word   offset  L7
```

4. 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。これにより部分的に最適化が抑止されます。

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

8.1.6 ポインタ変数を更新する記述のコンパイル強制終了に関する注意事項

1. 概要

   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」などのメッセージが表示されます。

2. 発生条件

   以下 4 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。

   条件をすべて満たす場合でも、C ソースの構文次第では問題が発生しない場合があります。発生しない場合は全く問題ありません。

   （a）ポインタ変数で間接参照した値を、そのポインタ自体に書き込んでいる。

   （b）(1) のポインタ変数がレジスタに割り当てられている。

   （c）(1) のポインタを使用する for 文または while 文がある。

   （d）(3) と同じ for 文または while 文の中に、間接参照がないと仮定した状態で、ループ外に移動できそうな不変式が存在する。

3. 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 i;
    int n = 1;
    int x = 0;

    for (i = 0; i < n-1; i++) {         /* 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) */
    }
}
```

4. 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。これにより部分的に最適化が抑止されます。

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

8.1.7 switch-case 文に関する注意事項

1. 概要

   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。)

2. 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。

   - switch 文の条件式中に unsigned char 型の変数を使用している場合

     下記の 3 項目のうちどれか 1 つを満たしている

     （a）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。

     （b）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。

     （c）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。

   - switch 文の条件式中に signed char 型の変数を使用している場合

     下記の 3 項目のうちどれか 1 つを満たしている

     （a）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。

     （b）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。

     （c）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。

3. 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 1:
case 2:
        func(3);
        break;
          :
          :
case 255:
        func(255);
        break;
default:
        break;
}
```

4. 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 unisigned 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 1:
case 2:
        func(3);
        break;
          :
          :
case 255:
        func(255);
        break;
default:
        break;
}
```

8.1.8 関数の戻り値とゼロを比較する場合に M(X) フラグが正しく設定されない場合があることに関する注意事項

1. 概要

戻り型が (signed/unsigned)char である関数の戻り値と、ゼロを直接比較する場合に M(X) フラグの変更命令が正しく生成されないため、誤った比較を行う場合があります。

2. 発生条件

以下 3 点の条件をすべて満たす場合に発生します。

（a）関数の戻り値と、定数ゼロを直接比較する。

- “if(func())” 形式の記述も定数ゼロとの比較に含めます。
- 条件比較には if, do, while, for を含みます。

（b）その関数の戻り型は (signed/unsigned)char 型である。

（c）その関数呼び出し時に、引数がスタックに積まれる。

- 全ての引数がレジスタ経由で関数に渡される場合は、本問題は発生しません。

3. 発生例

以下の C ソースをコンパイルすると、比較命令直前に出力されるはずの sem 命令が、出力されません。

```
unsigned char func(unsigned char, unsigned char);
volatile int i;

void xxx( unsigned char c )
{
    if( func( c, 1 ) == 0 )
        i = 0;
    else
        i = 1;
}
```

4. 回避策

本現象が発生した場合には、以下どちらかの形式で記述して下さい。

（a）関数の戻り型を (signed/unsigned)char 型以外にする。

```
unsigned int func(unsigned char, unsigned char);
volatile int i;

void xxx( unsigned char c )
{
    if( func( c, 1 ) == 0 )
        i = 0;
    else
        i = 1;
}
```

（b）関数の戻り値を (signed/unsigned)char 型以外の型の変数に格納して、その変数をゼロと比較する。

```
unsigned char func(unsigned char, unsigned char);
volatile int i;

void xxx( unsigned char c )
{
    int tmp;

    tmp = func( c, 1 );
    if( tmp == 0 )
        i = 0;
    else
        i = 1;
}
```

8.1.9 switch 文の分岐先に関する注意事項

1. 概要

   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。

2. 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。

   （a）オプション –OR を使用している。

   （b）オプション –ONBSD (–Ono_break_source_debug) は使用していない。

   （c）オプション –fST(–fswitch_table) を使用している。

   （d）1 つの関数内で、2 つ以上の switch 文を記述している。

   （e）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。

   （f）下のいずれか 1 つ以上を満たす。

      i. 一方の switch 文の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。

      ii. 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。

3. 発生例

   [C 言語ソースファイル例]

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

void func(void)
{
    if (a) {
        switch (b) {    /* 発生条件(d) 第一の switch 文 */
        case 8:
        case 9:
        case 11:
        case 12:
        case 13:
        case 14:
        case 15:
            x++;        /* 発生条件(f)i  ※1 */
            break;
        case 10:
            x = 2;
            break;
        default:
            x = 0;      /* 発生条件(f)i  ※2 */
            break;
        }
    } else {
        switch (c) {    /* 発生条件(d) 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(f)i  ※1 と共通処理 */
            break;
        default:
            x = 0;      /* 発生条件(f)i  ※2 と共通処理 */
            break;
        }
    }
}
```

4. 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。

   [C 言語ソースファイル修正例]

```
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;
        }
    }
}
```

8.1.10 同一変数を複数の if 文で連続して参照する場合に関する注意事項

1. 概要

   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。

2. 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。

   （a）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。

   （b）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。

（c）(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。

（d）(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。

（e）(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。

（f）(4) の 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。

3. 発生例

[C 言語ソースファイル例]

```
[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;
    }
}
```

4. 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;
    }
}
```

8.1.11 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場合の注意事項

1. 概要

   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を near、far または const 修飾子を付加して宣言した場合、不正なコードが生成される場合があります。

2. 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。

   （a）構造体または共用体を定義している。

   （b）(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。

   （c）(2) で定義した型によって変数を宣言してる。

   （d）(1) の構造体または共用体のメンバを参照している。

3. 発生例

   [C 言語ソースファイル例]

```
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) */
}
```

4. 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;
}
```

8.1.12 標準ライブラリ関数 sprintf に関する注意事項

1. 概要

   標準ライブラリ関数 sprintf 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 等の値 (sprintf 関数のフォーマット指定により小数点以下は異なります) になる場合があります。

2. 発生条件

   浮動小数点の整数部が 0 で、小数部が 9999999 のように、小数点末尾を四捨五入した結果、整数部へ桁上がりになる数字が引数として与えられた場合に発生します。

3. 発生例

   [C 言語ソースファイル例]

```
#include <stdio.h>
float   f;
int     main( void )
{
    char   buf[10];
    f = 0.9999999;
    sprintf( buf,"% f",f );     /* % と f の間にスペースがある */
                                 /* buf に 0.000000 が代入されます。 */
}
```

4. 回避策

   以下のいずれかの方法で回避してください。

   - 標準ライブラリ関数 sprintf のソースファイルである print.c を修正し、ライブラリアンを使用して、再度標準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」に以下の例の「追加」部分の処理を追加してください。

     M3T-NC77WA V.5.20 Release 4 の場合は、871 行目以降です。この行はコンパイラ製品およびバージョンにより異なります。

```
/* 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){                  /* 追加する処理 */
            inte[seisu] = '1';                /* 追加する処理 */
            seisu++;                          /* 追加する処理 */
        }                                       /* 追加する処理 */
    }
}
```

標準ライブラリファイルは以下の手順で作成してください。

（a）製品ディレクトリ下の SRC77
lib ディレクトリにある print.c を修正する。

（b）上記ディレクトリにある makefile(EWS 版)、または makefile.dos(PC 版) を使用して標準ライブラリファイルを作成する。

（c）作成した標準ライブラリファイルを環境変数 LIB77 は製品の数字) が指すディレクトリにコピーする。

- 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 = 0.9999999;
        /* 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
         */
        buf[0] = ' ';                   /* buf[0] にスペースを挿入 */
        sprintf( &buf[1],"%f",f );      /* buf[1] 以降に f の値を代入 */
}
```

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

1. 概要

   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。

2. 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。

3. 発生例

   [C 言語ソースファイル例]

```
#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 */
}
```

4. 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" );
}
```

8.2 スタートアッププログラムのサンプルに関する注意事項

使用されるマイコンの機種・システムにより、添付しているスタートアッププログラムのサンプルは変更していただく必要があります。機種により変更を要する内容は対応機種のデータブック等を参照下さい。

8.3 PC 版に関する注意事項

PC 版は Windows95、WindowsNT 4.0 以降の環境で動作します。Windows3.1 および WindowsNT 3.5x 以前のバージョンでは動作しません。

8.3.1 ファイル名についての注意事項

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、以下の注意事項に従って下さい。

- 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下さい。
- ショートカットは使用できません。
- 空白を含むディレクトリ名、ファイル名は指定できません。
- "..." 表記を用いて 2 つより上のディレクトリを指定する事は出来ません。
- 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。

8.3.2 MS–DOS プロンプトについての注意事項

- 日本語 WindowsNT 環境で DOS 窓のサイズが 80x25 以外に設定されている場合、NC77、RASM77 等を起動すると窓のサイズが頻繁に切り替わる事があります。窓のサイズを 80x25 にするか、TM を使用される事を推奨します。
- PC–9801 シリーズの Windows95 環境で、DOS 全画面表示状態で NC77、RASM77 等を起動すると DOS 窓表示に切り替わります。

8.4 デバッグ情報について

オプション –gie で出力されるデバッグ情報はレジスタ変数の情報を持ちません。そのため以下の制限が生じます。

- レジスタ変数を参照できない。
  C ソースファイル中で register 記憶クラスを指定し、かつオプション–Oenable_register を指定した場合のみ変数はレジスタに割り当てられます。NC77 のデバッグ情報はレジスタ変数の情報を持たないため**レジスタ変数は参照できません**。

なおレジスタ 引数 は、関数に入った後スタックに積まれるため従来通り参照可能です。

8.5 メモリ管理関数について

heap 領域には、ユーザーが使用する領域に加えて「メモリ管理関数が使用する管理領域」が格納されます。したがってユーザーが使用する領域のみのサイズで heap 領域を確保した場合は、プログラム実行中に heap 領域不足になる可能性があります。ご注意ください。

8.6 #pragma ASM について

8.6.1 #pragma ASM とレジスタ

コンパイラはレジスタを介して渡される引数、およびレジスタ変数に対して、これらの生存区間を解析しコードを生成します。しかしながら、インラインアセンブル機能 (#pragma ASM～#pragma ENDASM、asm 関数) を使用してレジスタ値を操作する記述を 行った場合、コンパイラはインラインアセンブル機能で記述された範囲における生存 区間の情報を保持することができません。

インラインアセンブル機能 (#pragma ASM～#pragma ENDASM、asm 関数) を使用してレジスタを操作する記述を行う場合は、必ずレジスタの退避・復帰を行ってください。

```
int   k;
int   func(int i)
{
      int j;

      asm( "pha" );          /* レジスタの退避 */
      asm( "lda.W       A,#000AH" );
      asm( "sta.W       A,DT:_k");
      asm( "pla" );          /* レジスタの復帰 */

      j = i + 100;
      return  j;
}
```

8.6.2 #pragma ASM, asm 関数と分岐命令

NC77 は#pragma ASM 及び asm 関数中に記述された内容をそのままアセンブリファイル中に出力します。下例のように#pragma ASM で C 言語ソースを含めた複雑な分岐を作成した場合は、C 言語変数の生存区間等を正しく解析できないため、正しいコードを生成できません。

```
int far flag;

int func(void)
{
        int     i;
        int     j;

        i = 1;
#pragma ASM
        lda     A, LG:_flag
        beq     LABEL2
LABEL1:
#pragma ENDASM
        return i;

#pragma ASM
LABEL2:
#pragma ENDASM
        j = 2;
#pragma ASM
        lda     A, LG:_flag
        cmp.W   A, #0001
        beq     LABEL1
#pragma ENDASM
        return j;
}
```

8.6.3 #pragma ASM を関数外に記述した場合

#pragma ASM を関数外に記述した場合は、該当箇所の C ソース行情報をアセンブリファイルに出力しません。

このため#pragma ASM〜#pragma ENDASM 内の記述に対して、アセンブル時及びリンク時のエラーメッセージの行番号情報、デバッグ時の行情報等が正常に出力されない事があります。

8.7 オプション–Ostack_align について

オプション–Ostack_align(–OSA) は、最適化オプション–O[1–5] を指定した場合のみ有効になります。–OSA を使用される場合は必ず–O[1–5] を指定して下さい。

8.8 inline 関数について

inline 関数中で分岐を伴う命令を使用した場合、System Error が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 関数中から取り除くか、inline 関数ではなく通常の関数をご使用下さい。

8.9 s2ie が認識可能なシンボルの長さについて

s2ie が認識可能なシンボル名の長さは 127 文字までです。128 文字超のシンボル名を使用された場合、s2ie は正しい IEEE–695 ファイルを作成する事が出来ません。127 文字以下のシンボル名をご使用下さい。