マイクロコントローラ技術情報

技術通知 **統合開発環境パッケージ** CubeSuite **使用制限事項の件**								発行番号	ＺＭＴ－Ｆ３５－１０－００１１号 1/3
								発行日	２０１１年 3月 ２３日
								発行部門	ルネサス エレクトロニクス株式会社 ＭＣＵ事業本部 ソフトウェア統括部 ＭＣＵツール技術部
文書分類	○	使用制限事項		バージョン・アップ		ドキュメント誤記訂正(正誤表)		その他	
関連資料	なし								

1. 対象製品

CubeSuite（**正式品名：**QS78K，QS850）

対象バージョンについては，各別紙を参照してください。

2. 新たな制限事項

今回新たに，以下の制限事項を追加しました。

対象製品については，各別紙を参照してください。

【CA78K0（別紙 3)】

- No.75　##演算子を使ったマクロ展開でエラーとなる制限
- No.76　アセンブラ・ソースに割り込み関数のシンボル情報が出ない制限

【CX（別紙 5)】

- No.8　関数呼び出し後に参照した変数値が不正になる制限
- No.9　間接代入後に参照した変数値が不正になる制限

【CA78K0R（別紙 9)】

- No.30　-qjオプション指定に関するC0101エラーとなる制限
- No.31　far領域のアドレスを long/unsigned long 型にキャストしたときの制限
- No.32　##演算子を使ったマクロ展開でエラーとなる制限
- No.33　アセンブラ・ソースに割り込み関数のシンボル情報が出ない制限
- No.34　ESレジスタの設定コードに関する制限

3. 回避策

今回追加した制限事項の回避策です。詳細は各別紙を参照してください。

【CA78K0（別紙 3)】

- No.75　以下のいずれかで回避してください。
  ・## 演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成しない時は、
  ## 演算子を使わないでください。
  ・## 演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成する時は、
  ## 演算子の直後に関数形式マクロのパラメータを置いてください。
- No.76　割り込み関数を定義する、あるいはオブジェクト・モジュール・ファイルを出力してください。

【CX（別紙 5)】

- No.8　以下のいずれかを適用して下さい。
  1．最適化オプションとして-O/-Osize/-Ospeed 以外を指定
  2．関数"F1"の任意の行に__asm("¥n");を挿入
  3．変数"V"を volatile 修飾
- No.9　以下のいずれかを適用して下さい。
  1. 最適化オプションとして-O/-Osize/-Ospeed 以外を指定
  2. 関数"F1"の任意の行に__asm("¥n");を挿入
  3. 変数"V"を volatile 修飾

【CA78K0R（別紙 9)】

- No.30　以下のいずれかで回避してください。
  ・ジャンプ最適化オプション(-qj)を無効にしてください。
  ・実行しない処理自体を削除する、あるいは#if 0 で囲んでコンパイル対象としないでください。
- No.31　アドレスが初期値の時は、long/unsigned long 型にキャストしないでください。
- No.32　以下のいずれかで回避してください。
  ・## 演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成しない時は、
  ## 演算子を使わないでください。
  ・## 演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成する時は、
  ## 演算子の直後に関数形式マクロのパラメータを置いてください。
- No.33　割り込み関数を定義する、あるいはオブジェクト・モジュール・ファイルを出力してください。
- No.34　間接参照を除く浮動小数点型変数のインクリメント／デクリメント演算直後にダミーのポインタ間接参照式を記述してください。

4. 改善計画

今回追加した新たな制限事項はそれぞれの別紙の改善策を参照ください。

5. 制限事項一覧

制限事項の履歴と，その詳細情報が含まれました制限事項一覧を，別紙 1～別紙 12 に記載します。

別紙 1：　デバッグ・ツール制限事項一覧
別紙 2：　78K0R/Kx3 用コード生成の制限事項一覧
別紙 3：　CA78K0 の制限事項一覧
別紙 4：　CA850 の制限事項一覧
別紙 5：　CX の制限事項一覧
別紙 6：　CX ユーティリティの制限事項一覧
別紙 7：　スタック見積もりツールの制限事項一覧
別紙 8：　78K0R/Kx3 用シミュレータの制限事項一覧
別紙 9：　CA78K0R の制限事項一覧
別紙 10：　リアルタイム OS 連係の制限事項一覧
別紙 11：　ビルド・ツールの制限事項一覧
別紙 12：　プログラミング・ツールの制限事項一覧

6. 発行文書履歴

統合開発環境 CubeSuite 使用制限事項 発行文書履歴

文書番号	発行日	記事
ZBG-CD-09-0007	2009. 1. 21	初版
ZBG-CD-09-0013	2009. 2. 18	デバッグ・ツールの新規制限事項追加（No.9）
ZBG-CD-09-0023	2009. 5. 11	デバッグ・ツールの制限事項追加（No.10） CA78K0R の制限事項追加（No.27,No.28） リアルタイム OS 連係の制限事項追加（No.1）
ZBG-CD-09-0029	2009. 6. 11	CA850 制限事項追加(No.104，No.105，No.106，No.107，No.108，No.109，No.110）
ZBG-CD-09-0037	2009. 7. 15	デバッグ・ツールの制限事項追加（No.11） ビルド・ツールの制限事項追加（No.1，No.2）
ZBG-CD-09-0052	2009. 9. 7	デバッグ・ツールの制限事項追加（No.12）
ZBG-CD-09-0059	2009. 11. 9	プログラミング・ツールの制限事項追加（No.1）
ZBG-CD-10-0011	2010.3.11	デバッグ・ツールの制限事項追加（No.13）
ZBG-CD-10-0013	2010.3.18	デバッグ・ツールの制限事項追加（No.14, No.15）
ZBG-CD-10-0020	2010.5.20	デバッグ・ツールの制限事項追加（No.19, No.20） CX の制限事項追加（No.6） CA78K0R の制限事項追加（No.29）
ZBG-CD-10-0025	2010.8.16	CX ユーティリティの制限事項追加（No.1） CA78K0 の制限事項追加（No.65-No.74）
ZMT-F35-10-0003	2010.11.11	CA78K0 の制限事項追加（No.1） CA850 の制限事項追加（No.111，No.112） CX の制限事項追加（No.7）
ZMT-F35-10-0011	2011.3.23	CA78K0 の制限事項追加（No.75，No.76） CX の制限事項追加（No.8，No.9） CA78K0R の制限事項追加（No.30，No.31，No.32，No.33，No.34）

以 上

デバッグ・ツールの制限事項一覧

1. 製品履歴

No.	対象デバッグ・ツール	対象デバイス	仕様変更・追加／制限事項	CubeSuite パッケージ									
				V1.00	V1.10		V1.12				V1.20		
				CubeSuite									
				V1.00	V1.10	V1.11	V1.12	V1.13	V1.14	V1.15	V1.20	V1.21	V1.30 / V1.31
1	MINICUBE2	V850	MINICUBE2 の通信方式設定に関する制限	×	×	×	×	×	×	×	○	○	○
2	全ツール	78K0R	ポインタ変数のウォッチ表示に関する制限	×	×	×	×	×	×	×	○	○	○
3	全ツール	78K0	スタック・トレース表示に関する制限	×	×	×	×	×	×	×	×	×	×
4	全ツール	78K0	メモリ・バンク内でステップ・インした際の制限	×	×	×	×	×	×	×	×	×	×
5	全ツール	78K0, 78K0R	ローカル変数の表示に関する制限	×	×	×	×	×	×	×	×	×	×
6	全ツール	78K0	逆アセンブル・ウインドウについての制限	×	×	×	×	×	×	×	×	×	×
7	全ツール	全デバイス	ブレークポイントの設定等が不正になる制限	×	×	×	×	×	×	×	×	×	×
8	IECUBE， MINICUBE(2)	V850	ブレークの競合に関する制限	×	×	×	×	×	×	×	×	×	×
9	全ツール	78K0, 78K0R	CPU レジスタの表示についての制限	×	○	○	○	○	○	○	○	○	○
10	全ツール	全デバイス	メイン・パネルに関する制限	×	×	○	○	○	○	○	○	○	○
11	シミュレータ	78K0R	ブレークポイントの設定箇所に関する制限	×	×	×	○	○	○	○	○	○	○
12	IECUBE	全デバイス	トレース･データの保存に関する制限	×	×	×	○	○	○	○	○	○	○
13	MINICUBE2	78K0R	ワイド・ボルテージ・モードに関する制限	×	×	×	×	×	×	○	○	○	○
14	MINICUBE2	78K0	DMM 機能に関する制限	×	×	×	×	×	×	×	○	○	○
15	IECUBE， シミュレータ	全デバイス	ポイント･トレースに関する制限	×	×	×	×	×	×	×	○	○	○
16	MINICUBE	V850E2M	RRM 機能，DMM 機能に関する制限	−	−	−	−	−	−	−	×	×	×
17	MINICUBE	V850E2M	フラッシュ・オプション設定に関する制限	−	−	−	−	−	−	−	×	×	×
18	MINICUBE	V850E2M	ステップ実行時に変数値に関する制限	−	−	−	−	−	−	−	×	×	×
19	MINICUBE2， シミュレータ	78K0R	オペランドが word[BC]等の命令(mov, movw)実行時の制限	×	×	×	×	×	×	×	×	○	○
20	IECUBE， MINICUBE2	78K0R	ブレーク時の周辺エミュレーション停止における制限	×	×	×	×	×	×	×	×	○	○

×：該当する
○：該当しない
−：対象外

2. 使用制限事項の詳細

No.1　MINICUBE2の通信方式設定に関する制限

【対　象】MINICUBE2，V850
【内　容】デバッグ・ツールのプロパティ[接続用設定]タブにおける[MINICUBE2 とターゲット・ボードとの接続]で選択する数値は，下表を参照して選択してください。

対象デバイス	設定値
V850ES/Kx1+，V850ES/Kx2	UART0：0，CSI0：1
V850ES/Jx2，V850ES/Jx3，V850ES/Jx3-L	UARTA0：0，CSIB0：1，CSIB3：2
V850ES/Hx2，V850ES/IE2，V850E/MA3 V850E/IA3，V850E/IA4，V850E/Ix3	UARTA0：0，CSIB0：1
V850ES/Jx3-H，V850ES/Jx3-U	UARTC0：0，CSIF0：1，CSIF3：2
V850ES/Fx3，V850ES/Fx3-L	UARTD0：0，CSIB0：1

【回避策】ありません。
【改善策】CubeSuite Ver.1.20 で修正しました。

No.2　ポインタ変数のウォッチ表示に関する制限

【対　象】全デバッグ・ツール，78K0R
【内　容】最適化をかけている場合，ポインタ変数のウォッチ表示が不正な表示になることがあります。
【回避策】デバッグを中心に行なう場合は，デバッグ優先オプション(-qg)をつけてください。
【改善策】CubeSuite Ver.1.20 で修正しました。

No.3　スタック・トレース表示に関する制限

【対　象】全デバッグ・ツール，78K0
【内　容】スタック・トレース表示機能は，スタックにフレーム・ポインタ（HL）を Push しない関数（noauto，norec 関数等）がある場合やメモリ・バンクを使っている場合には，main 関数まで正しく表示されないことがあります。
また，スタックにフレーム・ポインタ（HL）を Push しない関数（noauto，norec 関数等）や，メモリ・バンク関数からリターン実行した場合，フリーラン状態になることがあります。
【回避策】ありません。
【改善策】修正を計画中です。

No.4　メモリ・バンク内でステップ・インした際の制限

【対　象】全デバッグ・ツール，78K0
【内　容】メモリ・バンク内のユーザ定義ライブラリ関数またはメモリ・バンク内のデバッグ情報なし関数にソース・レベルでステップ・インした場合，バンク切り替えライブラリ内でブレークします。
【回避策】ありません。
【改善策】修正を計画中です。

No.5　ローカル変数の表示に関する制限

【対　象】全デバッグ・ツール，78K0，78K0R
【内　容】スタック・トレース・パネルで，カレント PC のスコープ外のローカル変数は，正しく表示できません。
【回避策】ありません。
【改善策】修正を計画中です。

No.6　逆アセンブル・ウインドウについての制限

【対　象】全デバッグ・ツール，78K0
【内　容】コモン領域内の命令を逆アセンブル・ウインドウで表示する際，表示される命令にメモリ・バンク領域内のシンボルが使用されていると，異なるバンクのシンボルを表示してしまう場合があります。
【回避策】ありません。
【改善策】修正を計画中です。

No.7　ブレークポイントの設定等が不正になる制限

【対　象】全デバッグ・ツール，全デバイス
【内　容】関数名や変数名を，先頭のアンダー・バーの有無などで使い分けている場合，デバッガが誤認識してしまい，シンボル変換や，ブレークポイントの設定が不正になる場合があります。
例えば_reset と__reset という 2 つの関数が存在していた場合などが該当します。
【回避策】類似する関数名や変数名は先頭のアンダー・バーだけで識別しないようにしてください。
【改善策】修正を計画中です。

No.8　ブレークの競合に関する制限

【対　象】IECUBE／MINICUBE(2)，V850
【内　容】ソフトウエア・ブレークと以下のハードウエア・ブレークとが競合した場合，PC の値を不正に補正する場合があります。

(1) トレース・フル・ブレーク
(2) ノンマップ・ブレーク
(3) ライト・プロテクト・ブレーク
(4) IOイリーガル・アクセス・ブレーク
(5) 停止ボタンによる強制ブレーク
(6) イベント・ブレーク（ハードウエア・ブレーク）
(7) タイムアウト・ブレーク

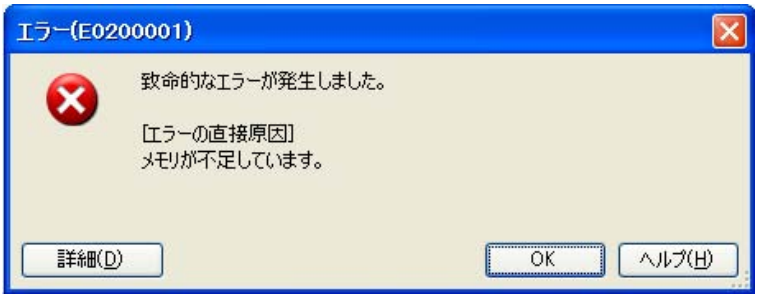
【回避策】ソフトウエア・ブレークではなく，ハードウエア・ブレークを使用してください。
【改善策】修正を計画中です。

No.9　CPUレジスタの表示についての制限

【対　象】全デバッグ・ツール，78K0，78K0R
【内　容】① 制御レジスタのレジスタ・バンク選択フラグ（RBS0，RBS1）を切り替えても，CPU レジスタパネルのカレント・レジスタ・バンクの表示が切り替え後のバンクの値を表示しません。常にバンク 0 を表示します。
ウォッチ パネルに汎用レジスタを登録した場合も同様に，常にバンク 0 を表示します。
② 制御レジスタのレジスタ・バンク選択フラグ（RBS0，RBS1）を切り替えた場合，切り替え後のバンクの汎用レジスタの HL 表示が，不正になります。
【回避策】それぞれ，次のように回避してください。
① 汎用レジスタを確認する場合は，CPU レジスタパネルの各汎用レジスタを参照してください。
② 回避策はありません。
【改善策】CubeSuite Ver.1.10 で修正しました。

No.10　メイン・パネルに関する制限

【対　象】全デバッグ・ツール，全デバイス
【内　容】メイン・パネルにソース・タブが存在する状態で以下に示す動作を行った場合，CubeSuite がバッファとして使用するホスト・マシンのメモリを開放しません。結果として，メモリが不足することにより下図に示すようなエラーが発生し，CubeSuite が正常に動作しなくなることがあります。
・プログラムの再ダウンロードをする。
・ソースの編集をする。
・ソースのスクロールをする。
・トレース・メモリ・サイズを変更する（シミュレータのみ）



【回避策】回避策はありません。
【改善策】CubeSuite Ver.1.11 で修正しました。

No.11　ブレークポイントの設定箇所に関する制限

【対　象】シミュレータ，78K0R
【内　容】①条件付きスキップ命令(SKC，SKNC，SKZ，SKNZ，SKH，SKNZ)の次命令にブレークポイントを設定している
②①の条件付きスキップ命令にはブレークポイントが設定されていない
③ウォッチ･パネル，メモリ･パネル，逆アセンブル･パネル，ローカル変数パネル，コールスタック･パネルのいずれかを表示している
①～③の条件を全て満たした上でプログラムを実行すると，スキップ条件の成立/不成立に関わらず設定したブレークポイントでブレークしてしまいます。
【回避策】回避策はありません。
【改善策】CubeSuite Ver.1.12 で修正しました。

No.12　トレース･データの保存に関する制限

【対　象】IECUBE，全デバイス
【内　容】デバッグ・ツールに IECUBE を使用するプロジェクトで，トレース・データの保存を実行すると，PC が再起動される場合があります。また、Windows のエラー画面(ブルースクリーン)になる場合があります。
【回避策】回避策はありません。
【改善策】CubeSuite Ver.1.12 で修正しました。

No.13　ワイド・ボルテージ・モードに関する制限

【対　象】MINICUBE2，78K0R
【内　容】デバッグ・ツールのプロパティ[接続用設定]タブにおける[フラッシュ]の[ワイド・ボルテージ・モードを使用する]で「はい」を選択してもワイド・ボルテージ・モードが設定されません。フル・スピード・モードが設定されます。
【回避策】回避策はありません。
【改善策】CubeSuite Ver.1.15 で修正しました。

No.14　DMM機能に関する制限

【対　象】MINICUBE2，78K0
【内　容】デバッグ・ツールのプロパティ[デバッグ･ツール設定]タブにおける[実行中のメモリ･アクセス]の[実行を一瞬停止してアクセスする]と[リアルタイム表示更新を自動設定する]を共に「はい」を選択しても,プログラム実行中に Watch パネルや Memory パネルに値を書き込んでも変わりません。
【回避策】回避策はありません。
【改善策】CubeSuite Ver.1.20 で修正しました。

No.15　ポイント・トレースに関する制限

【対　象】IECUBE，シミュレータ，全デバイス
【内　容】変数のポイント･トレースを 3 個以上設定した場合，次回プロジェクト読み込み時，またはロード・モジュールのダウンロード時にエラーが発生し，ポイント･トレースの設定が削除されることがあります。
【回避策】回避策はありません。
【改善策】CubeSuite Ver.1.20 で修正しました。

No.16　RRM機能，DMM機能に関する制限

【対　象】MINICUBE，V850E2M
【内　容】RRM，DMM 機能は使用できません。使用した場合，パラメータ・エラーが発生します。
【回避策】回避策はありません。
【改善策】修正を計画中です。

No.17　フラッシュ・オプション設定に関する制限

【対　象】MINICUBE，V850E2M
【内　容】フラッシュ・オプション設定プロパティのセキュリティ設定とブート・ブロック・クラスタ設定には未対応です。どのような値を設定しても無視します。
【回避策】回避策はありません。
【改善策】修正を計画中です。

No.18　ステップ実行時に変数値に関する制限

【対　象】MINICUBE，V850E2M

【内　容】関数呼び出しを跨いで生存する変数がレジスタに割り付けられている場合，デバッガで関数呼び出しの直前から呼び出し後にその変数が参照されるまでの区間でその変数を参照すると，不正な値が表示されることがあります。

【回避策】回避策はありません。

【改善策】修正を計画中です。

No.19　オペランドがword[BC]等の命令(mov, movw)実行時の制限

【対　象】MINICUBE2，シミュレータ，78K0R

【内　容】アセンブラ命令で"word[bc]"など，汎用レジスタ+オフセットをアクセス先アドレス指定のオペランドとして持つ命令[注]で，かつアクセス先アドレス指定のオペランドが 10000H を超える場合，命令の実行結果が不正になります。

・MINICUBE2　：ステップ実行時のみ問題が発生します。

・シミュレータ　：プログラム実行時，ステップ実行時ともに問題が発生します。

注：命令のオペランドとして[HL+byte]，[DE+byte]，[SP+byte]，word[B]，word[C]，word[BC]，[HL+B]，[HL+C]を含む命令。ただし、ES:[HL+byte]，ES:[DE+byte]，ES:word[B]，ES:word[C]，ES:word[BC]，ES:[HL+B]，ES:[HL+C]を含む命令は除きます。

【回避策】回避策はありません。

【改善策】CubeSuite Ver.1.21 で修正しました。

No.20　ブレーク時の周辺エミュレーション停止における制限

【対　象】IECUBE，MINICUBE2，78K0R

【内　容】デバッグ・ツールのプロパティ[デバッグ・ツール設定]タブにおける[ブレーク]項目にある周辺エミュレーションを停止する設定が下記のように逆になります。

・[停止時にタイマ系周辺エミュレーションを停止する]で「はい」を選択した場合，[停止時にシリアル系周辺エミュレーションを停止する]が「はい」に設定されます。

・[停止時にシリアル系周辺エミュレーションを停止する]で「はい」を選択した場合，[停止時にタイマ系周辺エミュレーションを停止する]が「はい」に設定されます。

【回避策】回避策はありません。

【改善策】CubeSuite Ver.1.21 で修正しました。

78K0R/Kx3 用コード生成の制限事項一覧

1. 製品履歴

No.	仕様変更・追加／制限事項	CubeSuite パッケージ							
		V1.00	V1.10		V1.12		V1.20		
		78K0R/Kx3 用コード生成							
		V1.00	V1.10	V1.11	V1.21	V2.00			
1	シリアル・インタフェース IIC0 に関する制限	×		○					

×：該当する
○：該当しない
－：対象外

2. 使用制限事項の詳細

No. 1　　シリアル・インタフェース IIC0 に関する制限

【内　容】　シリアル・インタフェース IIC0 の初期化手順が，デバイスのユーザーズ・マニュアルに記載されている手順と異なっています。
シリアル・インタフェース IIC0 に関する出力コードにて，以下の関数が対象です。

- IIC0_Init( )　　---　　初期化
- IIC0_MasterSendStart( )　　---　　マスタの送信動作
- IIC0_MasterReceiveStart( )　　---　　マスタの受信動作
- IIC0_SlaveSendStart( )　　---　　スレーブの送信動作
- IIC0_SlaveReceiveStart( )　　---　　スレーブの受信動作

【回避策】　P60，P61の出力ラッチへ0を設定するタイミングと動作許可のタイミングを下記のように修正してください。
マスタの送信動作時の出力コードで説明します。

```
void IIC0_Init(void)
{
        :
    /* Set INTIIC0 low priority */
    IICPR10 = 1U;
    IICPR00 = 1U;
    /* Set SCL0 pin */
    P6 &= 0xFEU;
    /* Set SDA0 pin */
    P6 &= 0xFDU;
    IICCL0 = _00_IIC0_CLOCK0 | _00_IIC0_FILTER_OFF;
    IICX0 = _00_IIC0_EXPENSION0;
    SVA0 = _10_IIC0_MASTERADDRESS;
    STCEN = 1U;
    IICRSV = 1U;
    SPIE0 = 0U;
    WTIM0 = 1U;
    ACKE0 = 1U;
    IICMK0 = 0U;
    IICE0 = 1U;          /* Enable IIC0 operation */
    /* Set SCL0 pin */
    PM6 &= 0xFEU;
    ~~P6 &= 0xFEU;~~
    /* Set SDA0 pin */
    PM6 &= 0xFDU;
    ~~P6 &= 0xFDU;~~
}

MD_STATUS IIC0_MasterSendStart(引数)
{
    MD_STATUS status = MD_OK;
    ~~IICE0 = 1U;~~
    LREL0 = 1U;
    :
}
```

移動)
出力ラッチに 0 を設定

移動)
動作許可

削除

削除

削除

【改善策】　CubeSuite にパッケージされる 78K0R/Kx3 用コード生成 Ver.1.11 で修正しました。

CA78K0 の制限事項一覧

1. 製品履歴

（1）アセンブラ部の制限事項一覧

No.	仕様変更・追加／制限事項	CubeSuite パッケージ						
		V1.00	V1.10		V1.12		V1.20		V1.40
		CA78K0						
		V1.00					V1.10	V1.11
1	構造化アセンブリ言語記述で範囲が交差する制御文がエラーとなる制限	×					×	×
5	saddr.bit の値を持つビット・シンボルを EQU 定義する際，saddr の部分に最適化の影響を受けるレーベルが記述されるとエラー等が不正になる制限	×					×	×
9	マクロ疑似命令 IRP をネストさせるとコンカティネート(&)が結合されない制限	－					－	－

×：該当する
○：該当しない
－：対象外

注意）項目の No.については，別製品の RA78K0 と同じ番号にしてありますので連続ではありません。

（2）コンパイラ部の制限事項一覧

No.	仕様変更・追加／制限事項	CubeSuite パッケージ						
		V1.00	V1.10		V1.12		V1.20	V1.40
		CA78K0						
		V1.00					V1.10	V1.11
12	ブロック内で extern 宣言された変数と同名の変数との結合が不正となる場合がある制限	×					○	○
14	大きさが定義されていない多次元配列が不正動作となる場合がある制限	×					○	○
16	signed 型のビット・フィールドを符号無しのビット・フィールドとして処理する制限	×					×	×
24	自動変数の配列名を参照で W0503 を出力する制限	×					○	○
41	大きさが定義されていない配列の初期化で，初期化子の中括弧の囲みが不統一な場合，確保される領域のサイズが不正となる制限	×					○	○
43	入出力関数の標準ライブラリの出力変換で，動作が不正となる制限	×					×	×
44	int/short 型の最小値-32768 のサイズが 4 となる制限	×					×	×
45	条件演算の第 2,3 項に関数名または関数ポインタを記述して関数を呼び出すと，エラーとなる制限	×					○	○
47	関数定義の識別子の型に対して，仮引数型と関数定義の識別子の型とが合わずに，エラーとなる制限	×					×	×
48	関数定義の識別子並びで，宣言していない仮引数を int 型とせずエラーとなる制限	×					×	×
49	#演算子が正しく展開できない制限	×					×	×
65	後置++/--ポインタが指す 1 バイトデータ参照時にコード不正となる制限	×※					○	○
66	char/signed char/unsigned char 型配列の初期化子並びの最後が文字列で、文字列の前に 1 個以上の定数または文字定数が並ぶ時にエラーとならず、コード不正となる制限	×※					○	○
67	ポインタ同士の減算結果をオフセットとしたポインタ参照時に、コード不正となる制限	×※					○	○
68	-qc オプションが未指定（int 型拡張する）の時、コード不正となる制限	×※					○	○
69	BCD 演算関数”adbcdw”, ”sbbcdw”のコード不正となる制限	×					○	○
70	-ng オプションを指定し、ASM 文を含む関数において、分岐命令でエラーとなる制限	×					○	○
71	ネストした if 文を抜けた直後の文に対して、行番号情報が出ない制限	×					○	○
72	割り込み関数内の long 型変数への代入でコード不正となる場合がある制限	×※					○	○
73	バンク関数呼び出しコードが出ない場合がある制限	×※					○	○
74	norec 関数にて１バイトの引数または auto 変数使用時に、コード不正となる場合がある制限	×※					○	○
75	##演算子を使ったマクロ展開でエラーとなる制限	×					×	×
76	アセンブラ・ソースに割り込み関数のシンボル情報が出ない制限	×					×	×

×：該当する
○：該当しない
－：対象外
※：チェックツールあり

注意）項目の No.については，別製品の CC78K0 と同じ番号にしてありますので連続ではありません。

（3）パッケージングに関する制限事項一覧

No.	**仕様変更・追加／制限事項**	CubeSuite **パッケージ**						
		V1.00	V1.10		V1.12		V1.20		V1.40
		CA78K0						
		V1.00					V1.10	V1.11
1	**[変数配置オプション]タブに関するエラー制限**	ー					×	○

×：該当する
○：該当しない
ー：対象外

2. 使用制限事項の詳細

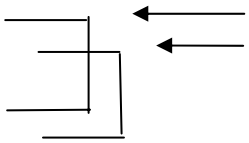
（1）アセンブラ部の制限事項詳細

No. 1　**構造化アセンブリ言語記述で範囲が交差する制御文がエラーとなる制限**

【内　容】　**制御文を**#ifdef～#endif **分割または交差するように囲んだ場合，**#ifdef **が真の時に制御文がエラーになります。**

(**例**)

```
   switch(mode)
#ifdef      stsw
 case 1:
             break
#endif
 default:
             break
ends
```



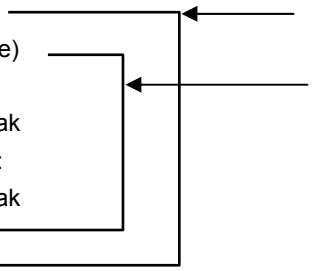
#ifdef～#else/#endif **の範囲**

case～**次の** case/default/ends **までの範囲**

【回避策】　**ネスト(入れ子)の場合には，正しく処理されます。制御文の範囲を交差させないようにソースを書き直してください。**

(**例**)

```
#ifdef stsw
 switch(mode)
      case 1:
          break
      default:
          break
 ends
#else
   switch(mode)
      default:
         break
 ends
#endif
```



#ifdef～#else/#endif **の範囲**

case～**次の** case/default/ends **までの範囲**

【改善策】　**制限事項とします。**

No. 5　saddr.bit の値を持つビット・シンボルを EQU 定義する際，saddr の部分に最適化の影響を受けるレーベルが記述されるとエラー等が不正になる制限

【内　容】 saddr.bit の値を持つビット・シンボルを EQU 定義する際に，saddr の部分に最適化の影響を受けるレーベルが記述されると，不正な処理を行います。
以下のケースで，不正な処理をします。

(1) saddr.bit が 0FD20H の時，レーベルがパス 1 で領域外，パス 2 で領域内となる場合，EQU 定義行に対し，パス１ではエラーを出力しますが，パス 2 では出力されず，不正なオブジェクトを生成します。

(2) saddr が 0FF1FH の時，レーベルがパス 1 で領域内，パス 2 で領域外となる場合，EQU 定義行に対し，パス 1 ではエラーになりませんが，パス 2 ではエラーを出力します。この EQU シンボルを参照した後に定義するレーベルに対し，以下のアセンブラエラーが出力されます。

「F410 Phase error」

また，このレーベルを参照するとオブジェクトが不正になります。

【回避策】 ありません。

【改善策】 制限事項とします。

No. 9　マクロ疑似命令 IRP をネストさせるとコンカティネート(&)が結合されない制限

【内　容】 文字列の結合記号 “ & ” を含む IRP をネストさせると，パラメータが置換されず，マクロの展開結果が不正になります。
(例)

```
IRP   ZZZ,<1,2,3>
      IRP   XXX,<4,5,6>
            LABEL&ZZZ&XXX: ;   展開結果が不正になります
      ENDM
ENDM
```

【回避策】 IRP をネストさせた場合は，“ & ” を使用しないでください。

（2）コンパイラ部の制限事項詳細

No. 12 **ブロック内で extern 宣言された変数と同名の変数との結合が不正となる場合がある制限**

【内　容】 **ブロック内で extern 宣言された変数と同名の変数との結合が，以下の 4 つの条件のうち，いずれかに該当する場合に不正となります。**

(1) **ブロック内で extern 宣言された変数と，以降のブロック外で static 宣言された変数が同名である場合。**
この場合，エラーとならず，結合もしないため，この変数を参照すると不正なコードを出力します。

(**例**)

```
void    f ( void ) {
            extern    int    i ;
            i = 1 ;                    /*  不正コード出力  */
}
static    int    i ;
```

(2) **ブロック内で extern 宣言された変数と，以降のブロック外で static 宣言されない変数が同名である場合。**
この場合，結合せずに，不正なコードを出力します。

(**例**)

```
void    f ( void ) {
            extern    int    i ;
            i = 1 ;                    /*  不正コード出力  */
}
int    i ;
```

(3) **ブロック内で extern 宣言された変数と，以前のブロック外で extern 宣言されない変数が同名であり，さらに extern 宣言された変数があるブロックを囲んでいるブロック中で宣言されている自動変数が同名である場合。**
この場合，ブロック外の変数とブロック内で extern 宣言された変数は結合せず，不正なコードを出力します。

(**例**)

```
int    i    =    1 ;
void    f ( void ) {
            int    i ;
              {
                        extern    int    i ;
                        i = 1 ;       /*  不正コード出力  */
            }
}
```

(4)**ブロック内で extern 宣言された変数と，他のブロック内で extern 宣言された変数が同名である場合**
この場合，結合せず，不正なコードを出力します。

(**例**)

```
void    f1( void ) {
            extern    int    i ;
            i = 2 ;
}
void    f2( void ) {
            extern    int    i ;
            i = 3 ;
}
```

【回避策】 **ありません**

【改善策】 CA78K0 Ver.1.10 **で修正しました。**

No. 14 **大きさが定義されていない多次元配列が不正動作となる場合がある制限**

【内　容】 **大きさが定義されていない多次元配列が，不正動作となる場合があります。**

(**例１**)
```
char   c [ ] [3] = { { 1 }, 2, 3, 4, 5 } ;     /* 不正コード */
```
(**例２**)
```
char   c [ ] [2] [3] = { "ab", "cd", "ef" } ; /* エラー(E0756) */
```

【回避策】 **多次元配列の大きさを定義してください。**

【改善策】 CA78K0 Ver.1.10 **で修正しました。**

No. 16 signed **型のビット・フィールドを符号無しのビット・フィールドとして処理する制限**

【内　容】 signed **型のビット・フィールドを，符号無しのビット・フィールドとして処理します。**

【回避策】 **ありません。**

【改善策】 **制限事項とします。**

No. 24 **自動変数の配列名を参照で** W0503 **を出力する制限**

【内　容】 **初期化なしの自動変数の配列に対して，式中で配列名を参照すると，W0503 を出力してしまいます。**

```
W0503  Possible use of ‘ 変数名 ’ before definition
```

注意：
「初期化」とは，int a[2]={0,0}; のような　宣言時の初期値のことです。a[0] = 0; a[1] = 0; のような代入式は含みません。
「配列名」とは，int a[2] の'a'自体のことです。a[0]，a[1]，&a[0]などは含みません。
(例)

```
void func(void)
{
   int a[2];
   int *b;

   a[0] = 0;
   a[1] = 0;
   b = a;            /* この行で W0503 が発生 */
}
```

【回避策】 W0503 **が出力された場合には，該当箇所を確認していただき，文中で初期化していた場合には，無視してください。**

【改善策】 CA78K0 Ver.1.10 **で修正しました。**

No. 41 **大きさが定義されていない配列の初期化で，初期化子の中括弧の囲みが不統一な場合，確保される領域のサイズが不正となる制限**

【内　容】 **大きさが定義されていない配列の初期化で，初期化子の中括弧の囲みが不統一な場合，確保される領域のサイズが不正となります。**

（例）

```
struct   t {
        int a;
        int b;
} x[ ] = {1, 2, {3, 4}};
```

【回避策】 **次のいずれかの方法で回避してください**

(1) **中括弧の囲み方を統一させる。**

```
struct   t {
        int a;
        int b;
} x[ ] = {{1, 2}, {3, 4}};
```

(2) **配列の大きさを定義する。**

```
struct   t {
              int a;
        int b;
} x[2] = {1, 2, {3, 4}};
```

【改善策】 CA78K0 Ver.1.10 **で修正しました。**

No. 43 **入出力関数の標準ライブラリの出力変換で，動作が不正となる制限**

【内　容】 **関数** printf,sprintf,vprintf,vsprintf **の出力変換で，以下のような場合に動作が不正となります。**
変換指定子"d, i, o, u, x **および** X"**に対して，**".2"**のように精度を指定した場合に，**0 **フラグを無視しません。**

（例）

```
    #include <stdio.h>
void func()
{
    printf("%04.2d¥n", 77);
}
```

※(**不正な動作**)　“0077”**となります**
(**正しい動作**)　“　77”**となります**

変換指定子"g,G"**に対して，指定した精度**+1 **を精度とします。**

（例）

```
#include <stdio.h>
void func()
{
    printf("%.2g", 12.3456789);
}
```

※（**不正な動作**）　“12.3”**となります**
（**正しい動作**）　“12”**となります**

【回避策】 **ありません。**

【改善策】 **制限事項とします。**

No. 44　int/short 型の最小値-32768 のサイズが 4 となる制限

【内　容】　int/short 型の最小値-32768 のサイズが 4 となります。

(例)
```
int x;
void func()
 {
        x = sizeof(-32768);
  }
```

※(不正な動作) x の値が 4 となります
(正しい動作) x の値が 2 となります

【回避策】　（-32767-1）と記述してください。

【改善策】　制限事項とします。

No. 45　条件演算の第 2,3 項に関数名または関数ポインタを記述して関数を呼び出すと,エラーとなる制限

【内　容】　条件演算の第 2,3 項に関数名または関数ポインタを記述して関数を呼び出すと，E0307 エラーとなります。

(例)
```
void f1(), f2();
 int x;
void func()
{
        (x ? f1 : f2)();
}
```

【回避策】　条件演算子ではなく if 文にしてください。

```
 (x ? f1 : f2)();
          ↓
if (x) {
        f1();
  }
else {
        f2();
    }
```

【改善策】　CA78K0 Ver.1.10 で修正しました。

No. 47　関数定義の識別子の型に対して，仮引数型と関数定義の識別子の型と合わずに，エラーとなる制限

【内　容】　関数定義の識別子の型に対して実引数拡張をしていないため，仮引数型と関数定義の識別子の型と適合せずに，E0747 エラーとなります。

（例）

```
int    fn_char(int);
int    fn_char(c)
char c;
{
        return 98;
}
```

【回避策】　仮引数型と関数定義の識別子の型を合わせてください。

【改善策】　制限事項とします。

No. 48　関数定義の識別子並びで，宣言していない仮引数を int 型とせずエラーとなる制限

【内　容】　関数定義の識別子並びで，宣言していない仮引数を int 型とせず E0706 エラーとなります。

（例）

```
void func(x1, x2, f, x3, lp, fp)
int (*fp)( );
long *lp;
float f;
{
                    :
}
```

【回避策】　関数定義の仮引数はすべて宣言してください。

【改善策】　制限事項とします。

No. 49　**#演算子が正しく展開できない制限**

【内　容】　**以下のいずれかの条件で正しく展開できません。**

（条件１）#演算子で「'"'」を正しく展開できずに，コンパイル時にエラーとなります。
(条件１の例)

```
#include <string.h>
#define str( a) (# a)
int x;
void func()
{
	if (strcmp(str('"'), "'¥"'") == 0) x++;
}
```

※(不正な動作) コンパイル時にエラーとなります
(正しい動作)　if (strcmp(("'\"'") , "'\"'") == 0) x++; となります

（条件２）#演算子とネストを含むマクロを正しく展開できません。

(条件２の例)

```
#define str(a) #a
#define xstr(a) str(a)
#define EXP 1
char *p;
void func()
{
	p = xstr(12EEXP);
}
```

※(**不正な動作**)　“p = ("12E1");”**となります。**
(**正しい動作**)　“p =　("12EEXP") ;”**となります。**

【回避策】　**ありません。**
【改善策】　**制限事項とします。**

No.65 **後置++/--ポインタが指す 1 バイトデータ参照時にコード不正となる制限**

【内　容】 **ポインタが指す 1 バイトの参照直後に、後置インクリメント/デクリメントする同ポインタが指す先を参照すると、コード不正となる場合があります。**

【例】

```
void func()
{
        unsigned char tmp, *src, dst;
            *src = tmp + 0x80;
             dst += *src++;
}
```

【回避策】 **以下のいずれかの方法で回避してください。**

(1) **代入＆後置インクリメントを、別々の式に分ける。**

```
dst += *src;
src++;
```

(2) **テンポラリ変数を用意して式を分割する。**

```
tmp2 = tmp + 0x80;
*src = tmp2;
```

【改善策】 CA78K0 Ver.1.10 **で修正しました。**
また、本制限に該当するか否かをチェックするツールを用意しております。
ツールに関しましては、販売店または特約店にお問い合わせください。

No.66 char/signed char/unsigned char 型配列の初期化子並びの最後が文字列で、文字列の前に 1 個以上の定数または文字定数が並ぶ時にエラーとならず、コード不正となる制限

【内　容】 char/signed char/unsigned char 型配列の初期化子並びの最後が文字列で、文字列の前に 1 個以上の定数または文字定数が並ぶ時にエラーとならず、コード不正となる場合があります。
char/signed char/unsigned char 型は、文字列リテラルもしくは定数値による初期化しか許されません。
【例】

```
 [ .c ]
const char a1[ ] = {0x01, “abc”};
char *const TBL[3] = { a1 };
char *ptr1;
void func()
 {
                ptr1 = TBL[0];
}
[.asm]
@@CNST   CSEG   UNITP
_a1:            DB       01H          ; 1
                DB       'ab'
_TBL:           DW       _a1          ; _TBL が奇数アドレス
                DB       (4)

@@DATA   DSEG   UNITP
_ptr1:       DS       (2)

; line       1 : const char a1[] = { 0x01, "abc" };
; line       2 : char *const TBL[3] = { a1 };
; line       3 : char *ptr1;
; line       4 : void func()
; line       5 : {

@@CODE     CSEG
_func:
; line       6 :   ptr1 = TBL[0];
                movw     ax,!_TBL              ; 奇数アドレスを参照
                movw     !_ptr1,ax
```

【回避策】 初期値を正しく記述してください。

```
const char a1[] = { 0x01, 'a', 'b', 'c', '¥0' };
char *const TBL[3] = { a1 };
char *ptr1;
void func()
{
            ptr1 = TBL[0];
}
```

【改善策】 CA78K0 Ver.1.10 で修正しました。
また、本制限に該当するか否かをチェックするツールを用意しております。
ツールに関しましては、販売店または特約店にお問い合わせください。

No.67 ポインタ同士の減算結果をオフセットとしたポインタ参照時に、コード不正となる制限

【内　容】 以下の条件をすべて満たす時にコード不正となります。
(1) 「ポインタ＋オフセット」の参照
(2) (1)のオフセットが、ポインタ同士の減算
(3) (2)のポインタ同士の減算において、ポインタがオフセット付き

【例】
```
[ .c ]
void main(void)
 {
      char   *p1;
      char   *p2;
      char   *p3;
          *p1 = *(p2 + (p1 − (p3 + 2)));
}
```

【回避策】 式を分割してください。
```
[ .c ]
void main(void)
{
    char   *p1;
    char   *p2;
    char   *p3;
    int   tmp;                    /*  テンポラリ変数を用意する  */
    tmp = (p1 - (p3 + 2));        /*  テンポラリ変数に代入する  */
    p1 = *(p2 + tmp);
}
```

【改善策】 CA78K0 Ver.1.10 で修正しました。
また、本制限に該当するか否かをチェックするツールを用意しております。
ツールに関しましては、販売店または特約店にお問い合わせください。

No.68 -qc オプションが未指定（int 型拡張する）の時、コード不正となる制限

【内　容】 以下の条件をすべて満たす時にコード不正となります。
(1) -qc 未指定(int 型拡張する)
(2) 次のいずれかの組合わせの乗算（演算子の左右を入れ換えても発生する）
- 「0～255 の定数を代入した unsigned char 型 sreg 変数」と「0～255 の定数」
- 「0～255 の定数を代入した unsigned char 型 sreg 変数」同士
- 「0～255 の定数を代入した unsigned char 型 sreg 変数」と「0～127 の定数を代入した char/signed char 型 sreg 変数」

(3) 乗算結果が 256 以上(unsigned char 型で表現できない値)
(4) 演算結果を int 型としてあつかう

以下の例では、正しくは Temp1 が 0x1FE になるはずが、0xFE になる。
【例】
```
[ .c ]
      unsigned int     Temp1;
          __sreg unsigned char    Byte1;
          Temp1 = (Byte1 = 255) * 2;
```

【回避策】 変数を int/unsigned int 型にキャストしてください。
```
[ .c ]
unsigned int     Temp1;
unsigned char    Byte1;

Temp1 = (unsigned int) (Byte1 = 255) * 2;
```

【改善策】 CA78K0 Ver.1.10 で修正しました。
また、本制限に該当するか否かをチェックするツールを用意しております。
ツールに関しましては、販売店または特約店にお問い合わせください。

No.69 BCD 演算関数"adbcdw", "sbbcdw"のコード不正となる制限

【内　容】 BCD 演算関数"adbcdw"および"sbbcdw"を使用すると、コード不正となる場合があります。
以下のいずれかの条件でコード不正になります。
(1) 代入先が配列またはポインタで、アドレス計算コードが発生する。
(2) テンポラリ変数を代入する前に、他の演算でレジスタを使用したコードが発生する。
(3) 代入したテンポラリ変数を、そのまま条件式など他の演算に使用する。

【例】

```
[ .c ]
 void func()
{
   unsigned int     tmp1[3];
   unsigned int     tmp2, tmp3;
      unsigned int      a = 10, i = 0, *p;

      tmp1[i] = adbcdw(80, 50);                          /* (1) */
      tmp2 = adbcdw(80, 50) + (a + 1);          /* (2) */
      if ((tmp3 = adbcdw(80, 50) == *p )        /* (3) */
                        :
}
```

【回避策】 関数"adbcdw", "sbbcdw"を呼び出すだけの関数を用意し、その関数を呼び出してください。
引数、返り値は関数 adbcdw,sbbcdw と同じにしてください。

【例】

```
 [ .c ]

unsigned int adbcdw_new(unsigned int a, unsigned int b)
{
          return adbcdw(a, b);
}
void func()
{
unsigned int     tmp1[3];
unsigned int     tmp2, tmp3;
unsigned int     a = 10, i = 0, *p;

tmp1[i] = adbcdw_new(80, 50);                    /* (1) */
tmp2 = adbcdw_new(80, 50) + (a + 1);         /* (2) */
if ((tmp3 = adbcdw_new(80, 50) == *p ) …     /* (3) */
                        :
}
```

【改善策】 CA78K0 Ver.1.10 で修正しました。

No.70 -ng オプションを指定し、ASM 文を含む関数において、分岐命令でエラーとなる制限

【内　容】 -ng オプションを指定し、ASM 文を含む関数において、ASM 文より後ろにある分岐命令でエラーとなる場合があります。
以下のすべての条件を満たすときエラーとなる場合があります。
(1) 関数内に ASM 文がある。
(2) 同関数内に分岐命令を出力する文(if 文, for 文, while 文など)がある。

ただし、この時、アセンブル時にエラーとなりますので、オブジェクト・モジュール・ファイルは生成されません。　エラーとならなければ、本制限に該当しません。

【例】

```
[ .c ]
unsigned int i;
void func()
{
        do {
                __asm(“¥t DB    (1000)”);
                i++;
        } while ( i < 10) ;
}
```

【回避策】 -g オプションに変更してください。

【改善策】 CA78K0 Ver.1.10 で修正しました。

No.71　　ネストした if 文を抜けた直後の文に対して、行番号情報が出ない制限

【内　容】　以下のすべての条件を満たすときに、ネストした if 文を抜けた直後の文に対して、行番号情報が出ない場合があります。　ただし、出力コードは正しいです。
行番号情報が出なかった行にはブレーク・ポイントの設定が出来ません。
(1)　2 つ以上の if 文が入れ子になっているネストレベル 3 以上の if 文がある。
(2)　上のネストレベルにある else が if 文の行番号より大きい。
(3)　直後に文が続く、上のネストレベルにある if 文が少なくとも１つある。

【例】

```
[ .c ]
int   f0, f1, f2, f3;
int   g0, g1, g2;
void func(void)
{
                if (f0) {
                        if (f1) {                   /* ネストレベル 1 の if 文 */
                            g2 = 5;
                        }
                        else if (f2) {              /* ネストレベル 2 の if 文 */
                            g2 = 4;
                        }
                        else if (f3) {              /* ネストレベル 3 の if 文(条件(1))  */
                            g2 = 3;
                        }
                        else {
                            g2 = 2;
                        }
                        g0 = 0x1234;                /* ネストレベル 1 の if 文の直後に文あり(条件(3))  */
                        g1 = 0x5678;
                    }
                    else {                          /* ここの else がネストレベル 3 の if 文の */
                        g2 = 1;                     /* 行番号より大きい(条件(2))               */
                    }
}
```

【回避策】　ネストした if 文を抜けた直後の文の前に、空行を数個挿入してください。
上記例では「g0 = 0x1234;」の前に、空行を数個挿入してください。

【改善策】　CA78K0 Ver.1.10 で修正しました。

No.72 **割り込み関数内の long 型変数への代入でコード不正となる場合がある制限**

【内　容】 **割り込み関数内で long 型変数への代入を記述し、-ql3 オプション以上を指定し、ランタイム・ライブラリ関数@@dels03 または@@hlls03 を呼び出すコードをコンパイラが生成した際に、割り込み関数内の他の処理で BC レジスタを使わず、割り込み関数内から関数を呼ばない時に、BC レジスタの中身を破壊します。**

【例】

```
[ .c ]
unsigned long l;
unsigned int i;
__interrupt void func()
{
                        l = (unsigned long)i;
}
```

【回避策】 **以下のいずれかで回避してください。**

(1) **関数末尾の** return **文(なければ追加)の後に、**long **型変数をインクリメントする処理を追加してください。**

```
[ .c ]
unsigned long l;
unsigned int i;
__interrupt void func()
{
        l = (unsigned long)i;
        return;
        l++;          /*  実行されない  */
}
```

(2) **空の処理のダミー関数を用意し、割り込み関数から呼び出してください。**

```
[ .c ]
unsigned long l;
unsigned int i;
void dummy { }
__interrupt void func()
{
        l = (unsigned long)i;
        dummy();
}
```

(3) **最適化オプション**-ql **のレベルを、**-ql1 **または**-ql2 **にしてください。**

【改善策】 CA78K0 Ver.1.10 **で修正しました。**
また、本制限に該当するか否かをチェックするツールを用意しております。
ツールに関しましては、販売店または特約店にお問い合わせください。

No.73 バンク関数呼び出しコードが出ない場合がある制限

【内　容】 以下の条件のいずれかを満たす時に、関数情報ファイルでバンク領域に配置するように指定した関数に対して、バンク関数呼び出しコードが出ない場合があります。
また、(3)に対しては「C0101：Internal error」が出る場合があります。
出なければ問題はなく、プログラム・コードには影響はありません。

(1) 関数ポインタにキャストして関数を呼び出す。
(2) typedef 名を使って宣言した関数を呼び出す。
(3) -mf オプションを指定し、関数ポインタを使って関数を呼び出す。

【例】

```
[ .c ]
typedef void F(void);
typedef void (*FP)(void);
void func1(void);
F func2;
void func()
{
        ((FP)func1)();
        func2();
}
```

【回避策】 関数ポインタにキャストしてバンク関数を呼び出さないようにしてください。
typedef 名を使ってバンク関数を宣言しないようにしてください。

```
[ .c ]
void func1(void);
void func2(void);
void func()
{
    func1();
    func2();
}
```

【改善策】 CA78K0 Ver.1.10 で修正しました。
また、本制限に該当するか否かをチェックするツールを用意しております。
ツールに関しましては、販売店または特約店にお問い合わせください。
ただし、条件(3)に関してはチェックを行いません。

No.74 norec 関数にて１バイトの引数または auto 変数使用時に、コード不正となる場合がある制限

【内　容】 norec 関数にて１バイトの引数または auto 変数の使用時に、norec 関数内で long 型変数に対して間接参照すると、コード不正となる場合があります。

【例】

```
[ .c ]
long buf[8];
norec void func(void)
{
            unsigned char c = 7;
        long *s = &buf[c-1], *d = &buf[c];
        *d = *s;
}
```

【回避策】 norec 関数でなく通常の関数にするか、１バイト変数を２バイト変数に変えてください。

【改善策】 CA78K0 Ver.1.10 で修正しました。
また、本制限に該当するか否かをチェックするツールを用意しております。
ツールに関しましては、販売店または特約店にお問い合わせください。

No.75 **##演算子を使ったマクロ展開でエラーとなる制限**

【内　容】 **## 演算子の直後が、関数形式マクロのパラメータでなく、大小英字でも下線 _ でもない時、**E0803 **エラーが出る場合があります。**
演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成する時は、E0711,E0301 **などの** E0803 **以外のエラーが出る場合があります。**

【例１】

```
[*.c]
#define m1(x) (x ## .c1 + 23)
#define m2(x) (x ## .c1 + 122)
struct t1 {
    unsigned char c1;
} st1;
unsigned char x1, x2;
void func1()
{
        x1 = m1(st1) + 100;     /* E0803 エラー (NG) */
        x2 = m2(st1) + 1;       /* ノーエラー   (OK) */
}
```

【例２】

```
[*.c]
#define m3(x) (x ## 1)
unsigned char x3, uc1;
void func2()
{
        x3 = m3(uc);            /* E0711, E0301 エラー (NG) */
}
```

【回避策】 **## 演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成しない時は、**
演算子を使わないでください。

```
#define m1(x)      (x ## .c1 + 23)
#define m2(x)      (x ## .c1 + 122)
          ↓
#define m1(x)      ((x).c1 + 23)
#define m2(x)      ((x).c1 + 122)
```

演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成する時は、
演算子の直後に関数形式マクロのパラメータを置いてください。

```
#define m3(x)      (x ## 1)
unsigned char x3, uc1;
void func2()
{
        x3 = m3(uc);
}
          ↓
#define m3(x, y)   (x ## y)
unsigned char x3, uc1;
void func2()
{
        x3 = m3(uc, 1);
}
```

【改善策】 CA78K0 Ver.1.20 **で修正いたします。**

No.76 **アセンブラ・ソースに割り込み関数のシンボル情報が出ない制限**

【内　容】 **以下の条件をすべて満たすときに、リンク時に** E3405 **エラーになります。**
(1) #pragma interrupt **による割り込み関数のベクタテーブルの生成指定がある。**
(2) **同一ソース内に割り込み関数の定義がない。**
(3) -no, **アセンブラ・ソース・モジュール・ファイル出力**(-a or -sa)**、デバッグ情報出力**(-g) **オプションを有効にする**

```
[*.c]
#pragam interrupt INTP0 inter
/*                              割り込み関数の定義がコンパイル対象ではない
__interrupt void inter()
{
            :
}
*/
```

【回避策】 **割り込み関数を定義する、あるいはオブジェクト・モジュール・ファイルを出力してください。**

【改善策】 CA78K0 Ver.1.20 **で修正いたします。**

（3）パッケージングに関する制限事項詳細

No. 1 **[変数配置オプション]タブに関するエラーの制限**

【内　容】 **ビルド・ツールのプロパティ・パネルの[変数配置オプション]タブの[変数情報ファイルを出力する]を”はい”、あるいは**[ROM/RAM **使用量を表示する]を”はい”にしてビルドした場合、以下のエラーが表示されます。**

(**プログラム**（C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥CA78K0¥V1.10¥Bin¥vf78k0.exe)**の起動に失敗しました。**(E0200002))

【回避策】 **ありません。**
変数情報ファイルの出力、または ROM/RAM **使用量表示は使用しないでください。**

【改善策】 CA78K0 V1.11 **で修正しました。**

CA850 の制限事項一覧

1. 製品履歴

No.	仕様変更・追加／制限事項	CubeSuite パッケージ				
		V1.00	V1.10	V1.12	V1.20	V1.40
		CA850				
		V3.31	V3.41	V3.43	V3.45	V3.47
4	浮動小数点定数演算の演算時精度の制限	×	×	×	×	×
6	関数の引数で構造体型の条件演算子の制限	×	×	×	×	×
7	関数の間接呼び出しの制限	×	×	×	×	×
8	意味不明な関数定義がエラーにならない制限	×	×	×	×	×
12	関数宣言で ( ) が余計に付いている場合エラーになる制限	×	×	×	×	×
21	セクション配置の制限	×	×	×	×	×
22	変数の実体がアセンブリ・ソースにあるセクション・ファイルの制限	×	×	×	×	×
23	同名の外部変数の仮定義が複数ファイルにあるセクション・ファイルの制限	×	×	×	×	×
29	最適化オプション指定時のエラーの制限	×	×	×	×	×
30	最適化時のオブジェクト・サイズの制限	×	×	×	×	×
31	最適化時のデバッグの制限	×	×	×	×	×
33	構造体メンバのアドレスの制限	×	×	×	×	×
34	ビット・フィールドの制限	×	×	×	×	×
40	C ソース内における，固有シンボル，予約シンボルの参照の制限	×	×	×	×	×
95	浮動小数点定数と汎整数型の制限	×	×	×	×	×
96	入出力関数の標準ライブラリの入力変換の制限	×	×	×	×	×
104	ループの実行回数が不正になる制限	×※	○	○	○	○
105	文字列定数の内容が不正になる制限	×※	○	○	○	○
106	sscanf，fscanf，scanf 関数の型指定に関する制限	×	○	○	○	○
107	atoi，atol，strtol，strtoul 関数の引数文字列に関する制限	×	○	○	○	○
108	ビットフィールドをメンバに持つ構造体型変数の初期化の制限	×※	○	○	○	○
109	条件アセンブル擬似命令のネストの制限	×	○	○	○	○
110	switch，if 文内での代入の制限	×※	○	○	○	○
111	キャストを伴う比較演算が不正に最適化される制限	×※	×※	×※	×※	○
112	代入文に対する不正移動の制限	×※	×※	×※	×※	○

×：該当する
○：該当しない
－：対象外
※：チェックツールあり

※項目の No.については，別製品の CA850（CA703000）と同じ番号にしてありますので連続ではありません。

2. 使用制限事項の詳細

No. 4　浮動小数点定数演算の演算時精度の制限

【内　容】　整数へのキャストをともなうコンパイルにおいて，演算ができてしまうような浮動小数点演算を記述した場合，極微少の精度ずれが発生し，整数へのキャストにより不正な値になることがあります。なお，浮動小数点のまま扱う場合，問題は発生しません。

（例）
```
(long)(1.12 * 100);
```

【回避策】　下記のいずれかのように変更してください。

```
float  f=1.12;
(long)(f * 100);
```

または，

```
float  f;
(long)(f=1.12 * 100);
```

【改善策】　制限事項とします。

No. 6　関数の引数で構造体型の条件演算子の制限

【内　容】　引数に構造体型の条件演算子が存在する場合に正しい分岐コードが生成されません。

（例）
```
typedef struct {int i;}S;
S ss1, ss2;
int  j;
void func() {
     func_call((j>10)?ss1:ss2);
}
```

【回避策】　下記のように if 文に変更してください。

```
if (j > 10){
     func_call(ss1);
 } else {
     func_call(ss2);
}
```

【改善策】　制限事項とします。

No. 7　関数の間接呼び出しの制限

【内　容】 関数の間接呼び出し式が，オフセットを必要とする呼び出し方が，以下のいずれかのコンパイラ内部エラーとなります。

（メッセージ）

C2000：internal：gen_binary()：OP_CALL：left-child's operator is wrong

C5211: syntax error at line <num> in intermediate file

（例）
```
struct S{;
    int dummy;
    int func_body[0x100];
}sobj;
void f() {
    ((void(*)())sobj.func_body)();
    fp();
}
```

【回避策】 下記のようにオフセット計算式と呼び出し式を分けてください。

```
void f() {
    void(*fp)()=(void(*)())&sobj.func_body;
    fp();
}
```

【改善策】 制限事項とします。

No. 8　意味不明な関数定義がエラーにならない制限

【内　容】 次のような意味不明な関数定義がエラーになりません。

（例）
```
typedef int INTFN();
INTFN  f{return(0);}
```

【回避策】 該当するようなコーディングはしないでください。

【改善策】 制限事項とします。

No. 12　関数宣言で（）が余計に付いている場合エラーになる制限

【内　容】 関数宣言で()が余計に付いている場合，エラーになります。

（例）
```
typedef int Int;
void f1((Int));
```

【回避策】 記述を以下のように変更してください。

```
typedef int Int;
void f1(Int);
```

【改善策】 制限事項とします。

No. 21 セクション配置の制限

【内　容】 #pragma section 指令による tidata セクション配置指定，または sf850 を用いて tidata セクションに配置指定した「構造体の char 型配列」や「char 型メンバへのアクセス」において，その配列，またはメンバのアクセスに使用する sst 命令／sld 命令のディスプレースメント値を越えた場合，リンカでエラーを発生します。

【回避策】 以下のいずれかの方法で回避してください。

1. リンカのエラーとなる構造体の char 型メンバや char 型配列を使用しないでください。
2. リンカのエラーとなる構造体の char 型メンバや char 型配列を tidata セクションに割り当てないでください。

【改善策】 制限事項とします。

No. 22 変数の実体がアセンブリ・ソースにあるセクション・ファイルの制限

【内　容】 変数の実体がアセンブリ・ソースにあり，その変数を C ソース上から参照しているアプリケーションにおいて，セクション・ファイルを sf850 で生成していた場合，リンク時にエラーが発生します。

【回避策】 アセンブリ・ソース中にある変数を，セクション・ファイルから削除して下さい。

【改善策】 制限事項とします。

No. 23 同名の外部変数の仮定義が複数ファイルにあるセクション・ファイルの制限

【内　容】 同名の外部変数の仮定義が複数ファイルにあるとき，セクション・ファイルを sf850 で生成していた場合，リンク時にシンボルの二重定義になる場合がある。

（メッセージ）

ld850：fatal error: symbol "_xxxx" multiply defined.

【回避策】 同名の外部変数の仮定義が複数ある場合は，外部変数を参照するファイルでは，必ず extern 宣言してください。

【改善策】 制限事項とします。

No. 29 最適化オプション指定時のエラーの制限

【内　容】 コンパイル時に指定する最適化オプションの最適化レベルを高くすることにより，コンパイル中に実行されるフェーズ（最適化機能やコンパイル機能など）が多くなります。-Ot オプションを指定した場合，このフェーズ間で生成される中間ファイルが巨大になり，fatal error を起こす場合があります。

【回避策】 最適化オプションのレベルを下げて（-Os など）コンパイルして下さい。

【改善策】 制限事項とします。

No. 30　最適化時のオブジェクト・サイズの制限

【内　容】　最適化を指定し，デバッグ情報を含むオブジェクト・ファイルのサイズは，非常に大きくなる場合があります。

【回避策】　最適化レベルを下げるか，デバッグしたいファイルだけにデバッグ情報を出力する“ -g オプション ”をつけてコンパイルするようにしてください。

【改善策】　制限事項とします。

No. 31　最適化時のデバッグの制限

【内　容】　最適化を指定して，ソース・デバッグを行う際は次の制限事項があります。

1. 変数の値を参照した際，正しい値でなく，計算途中の一時的な値が得られることがあります。
2. 配列の一部，構造体の要素，ユーザー定義型のポインタ変数がレジスタに割りつけられた場合，デバッガの変数ウインドウ等にて変数の表示／変更が不正になることがあります。
3. 自動変数の配列の一部，構造体の要素が使用されない場合，領域が削除されることがあります。この場合，デバッガの変数ウインドウ等にて変数の表示／変更が不正になることがあります。変更の場合，スタックを破壊する可能性があります。

【回避策】　ありません。

【改善策】　制限事項とします。

No. 33　構造体メンバのアドレスの制限

【内　容】　構造パッキングを行って，以下のいずれかの条件に該当する場合，

1. ミス・アライン・アクセスに対応していないデバイスを使用している
2. ミス･アライン・アクセスに対応したデバイスで，ミス・アライン・アクセスを禁止している

データのアクセスは，デバイスのデータ・アライメントに従い，アドレスをマスクしてアクセスされるため，構造体メンバのアドレスでのアクセスで，データの消失や切り捨てが生じます。

（例）
```
struct test {
    char c;    /* offset 0 */
    int  i;    /* offset 1-4 */
} test;
int *ip, i;
void func(){
    i = *ip;   /* マスクされたアドレスでアクセスされる */
}
void func2(){
    ip = &(test.i);
}
```

【回避策】　ありません。

【改善策】　制限事項とします。

No. 34　ビット・フィールドの制限

【内　容】 構造パッキングにおけるビット・フィールドへのアクセスにおいて，ビット・フィールドへのアクセスではビット・フィールドの幅がメンバの型以下の場合，メンバの型で読み込むため，オブジェクトの外部（データのない領域）もアクセスします。通常，正常に実行されますが，I/O がマッピングされていると，不正アクセスとなる場合があります。

（例）
```
struct S {
    int  x:21;
} sobj;       /* 3 バイト */
sobj.x = 1;
```

【回避策】 ありません。

【改善策】 制限事項とします。

No. 40　C ソース内における，固有シンボル，予約シンボルの参照の制限

【内　容】 C ソース内において，__gp_DATA などのターゲット固有シンボル，__stext などの予約シンボルを参照することができません。

【回避策】 C ソース内では，ターゲット固有シンボルや予約シンボルを使用しないで下さい。

【改善策】 制限事項とします。

No. 95　浮動小数点定数と汎整数型の制限

【内　容】 浮動小数点型の値を汎整数型に型変換する場合，整数部の値で表現できる範囲を signed int/signed long 型の値域として，それを越えて unisigned int/unsigned long 型に変換する場合には，コンパイル・エラーとなることがあります。

```
E2519: invalid has occurred at compile time.
```

【 例 】

```
unsigned int ui = 2147483647.0;         /* OK */
unsigned int ui = 2147483648.0;         /* エラー */
```

【回避策】 整数型にしてください。

【 例 】

```
unsigned int ui = 2147483648;
```

【改善策】 制限事項とします。

No. 96　入出力関数の標準ライブラリの入力変換の制限

【内　容】 入出力関数の標準ライブラリの関数 printf, sprintf, vprintf, vsprintf の入力変換で，変換指定子 "g,G"に対して，指定した精度を+1 します。

【 例 】

```
printf("%.2g", 12.3456789);
/* 12 となるべきですが，12.3 となってしまいます。*/
```

【回避策】 ありません。

【改善策】 制限事項とします。

No. 104　ループの実行回数が不正になる制限

【内　容】　下記の 1.～5. の条件を全て満たす場合に，ループの実行回数が不正になる場合があります。

【 条 件 】

1. 最適化オプションとして"-Og" "-O" "-Os" "-Ot"のいずれかを指定している
2. 帰納変数が volatile 未指定である
3. ループの終了条件が帰納変数と定数との比較である
4. ループ内で帰納変数に対して定数の加減算を行っている
5. ループ内で帰納変数を(ア)または(イ)のように使用している
   (ア)　a.から e.の全てを満たす【例 1】
      a. 帰納変数を配列のインデックスとして使用
      b. a.の配列の要素が構造体・共用体・配列のいずれか
      c. a.の配列の要素サイズが 2 のべき乗以外
      d. a.の配列の要素サイズが 65534 バイトの範囲内
      e. ループの終了条件として指定している 3. の定数と，
         a.の配列の要素サイズの積が 32bit 整数を超過
   (イ)　f.から i.の全てを満たす【例 2】
      f. 帰納変数を定数との乗算の一方のみに使用
      g. f.の定数が 2 のべき乗以外
      h. f.の定数が-65534～65534 の範囲内
      i. ループの終了条件として指定している 3. の定数と，
         f.の定数との積が 32bit 整数を超過

(注意) 帰納変数とは，ループの終了条件を制御する変数です。

【 例 1 】

```
struct {
    int s1;
    int s2;
    int s3;          //配列 ary の要素は構造体
} ary[100];          // サイズ(12 バイト)は 2 のべき乗以外で 65534 バイトの範囲内

int i;                                    //帰納変数 i は volatile 未指定
for ( i = 0; i < INT_MAX; i ++ ){         //ループの終了条件が帰納変数と定数との比較
                                          //帰納変数 i に対して定数の加算

    ary[i].s1 = 1;                        //帰納変数 i を配列のインデックスに使用
     …
}
```

INT_MAX*12(=0x5FFFFFFF4)が 32bit 整数を超過しているため，本制限に該当する可能性があります。

【 例 2 】

```
volatile int vi;

int i = 0;                                //帰納変数 i  は volatile 未指定
while ( i < INT_MAX ){                    //終了条件が帰納変数と定数との比較
  vi = i * 100;                           //帰納変数 i を定数との乗算のみに使用
  …                                       //定数は-65534～65534 の範囲内

  i++ ;                                   //帰納変数 i に対して定数の加算
}
```

INT_MAX*100(=0x31FFFFFF9C)が 32bit 整数を超過しているため，本制限に該当する可能性があります。

【回避策】 下記の 1.〜3. のいずれかを適用してください。

1. ループ終了条件の定数を下記のいずれかのように変更する
   - ループ終了条件の定数と，帰納変数をインデックスとする配列の要素サイズとの積が 32bit 整数を超過しない
   - ループ終了条件の定数と，帰納変数と乗算を行っている定数との積が 32bit 整数を超過しない

2. 帰納変数に対して volatile 指定する

【 変更前 】

```
int i;
for ( i = 0; i < INT_MAX; i ++ ){

    ary[i].s1 = 1;
    …
}
```

【 変更後 】

```
volatile int i;
for ( i = 0; i < INT_MAX; i ++ ){

    ary[i].s1 = 1;
    …
}
```

3. 最適化オプションとして"-Od" "-Ob" のいずれかを指定する

【改善策】 CA850 Ver.3.41 で修正しました。
また，本制限に該当するかをチェックするツールを用意しています。
チェックツールに関しましては，弊社営業または特約店にお問い合わせください。

No. 105 文字列定数の内容が不正になる制限

【内　容】 下記の 1. 〜 2. の条件を全て満たす場合，文字列定数の内容が不正になります。

【 条 件 】

1. 文字列定数中に ASCII コードの 0x00 を使用している
2. 1.に対して ASCII コードの 0x30〜0x37 のいずれかが連続する

【 例 】ASCII コード 0x00 の直後の ASCII コードが 0x37 である場合

```
char string1[] = "¥x00¥x37";
char string2[] = "¥000¥067";     // (37)16 = (67)8
char string3[] = "¥x00" "7";     // (37)16 = '7 '
```

いずれも 0x00, 0x37, 0x00 が正常な出力ですが 0x07, 0x00 と不正な出力となります。

【回避策】 下記の 1.〜2. のいずれかを適用してください。

1. 文字列定数を使用しないで初期化する

```
char string4[] = {'¥x00', '¥x37', '¥0'};
```

2. ASCII コードの 0x30〜0x37 以外で初期化した後，動的に書き換える

```
char string5[] = "¥x00*";
string5[1] = '¥x37';
```

【改善策】 CA850 Ver.3.41 で修正しました。
また，本制限に該当するかをチェックするツールを用意しています。
チェックツールに関しましては，弊社営業または特約店にお問い合わせください。

No. 106　sscanf，fscanf，scanf 関数の型指定に関する制限

【内　容】　下記の 1.～3.の条件を全て満たす場合，3.の型指定に対応する引数の内容が書き換わります。

【 条 件 】

1. sscanf，fscanf，scanf 関数のいずれかを使用している
2. 型指定の数より入力フィールドが少ない
3. 余った最初の型指定文字が s，e，f，g，E，F，G，[ ] のいずれかである【例 1】【例 2】

また下記の 4.～5.の条件を全て満たす場合，5.の型指定に対応する引数の内容が書き換わります。

【 条 件 】

4. sscanf, fscanf, scanf 関数のいずれかを使用している
5. 型指定文字として[ ] を使用し，[ ]で囲まれた文字パターンが入力フィールドにない【例 3】

【 例 1 】余った最初の型指定文字が「f」の場合

```
char ary1[5];
float  f1 = 2.0, f2 = 3.0;

sscanf ("aaaa", "%s %f %f", ary1, &f1, &f2);
                            ↖ 余った最初の型指定
```

入力フィールド"aaaa" は「ary1」に文字列として格納されます。しかし「%s」以降の型指定「%f %f」に対応する入力フィールドがありません。このとき，余った最初の型指定に対応する引数「f1」の値が書き換わります。

[期待値]	[出力結果]
ary1 = "aaaa", f1 = 2.0, f2 = 3.0	ary1 = "aaaa", f1 = 0.0, f2 = 3.0

【 例 2 】余った最初の型指定文字が「s」の場合

```
int  data1, data2;
char ary2[5]="test";

sscanf ("1 2", "%d %d %s", &data1, &data2, ary2);
                           ↖ 余った最初の型指定
```

入力フィールド"1" は「data1」に 10 進整数として格納されます。入力フィールド"2" は「data2」に 10 進整数として格納されます。しかし「%d %d」以降の型指定「%s」に対応する入力フィールドがありません。このとき，余った最初の型指定に対応する引数「ary2」の内容が書き換わります。

[期待値]	[出力結果]
data1 = 1, data2 = 2, ary2 = "test"	data1 = 1, data2 = 2, ary2 = "¥0"

【 例 3 】[ ]で囲まれた文字パターンが入力フィールドにない場合

```
char ary3[5] = "test";
char ary4[5] = "test";
char ary5[5] = "test";

sscanf ("aaaa bbbb cccc", "%s %[a] %s", ary3, ary4, ary5);
                          ↖ 合致しない型指定
```

入力フィールド"aaaa" は「ary3」に文字列として格納されます。続いて入力フィールド"bbbb" から「a」に合致する文字だけを文字列として「ary4」に格納しようとしますが，合致するパターンがありません。このとき「ary4」の内容が書き換わります。

[期待値]	[出力結果]
ary3 = "aaaa", ary4 = "test", ary5 = "test"	ary3 = "aaaa", ary4 = "¥0", ary5 = "test"

【回避策】　ありません。

【改善策】　CA850 Ver.3.41 で修正しました。

No. 107　atoi，atol，strtol，strtoul 関数の引数文字列に関する制限

【内　容】　下記の 1.～5.の条件を全て満たす場合，戻り値が不正になります。また strtol，strtoul 関数の場合は，グローバル変数 errno にマクロ ERANGE が設定されません。

【 条 件 】

1. atoi，atol，strtol，strtoul 関数のいずれかを使用している
2. atoi, atol 関数の場合は，引数文字列を 10 進数値として表現すると，32 ビットを超過する
   strtol, strtoul 関数の場合は，第一引数の文字列を，第三引数により指定した基数の値で表現すると，32 ビットを超過する
3. 2.の引数文字列の先頭からある文字までを変換した数値と，先頭からある文字の次の文字までを変換した数値の絶対値の下位 32 ビット同士を比較して，後者の値が前者の値以上である
4. 3.の比較が文字列の先頭から末尾まで成立する
5. atoi，atol，strtol 関数の場合は，2.の変換数値の下位 32 ビットに符号を付加した値が LONG_MIN～LONG_MAX の範囲内である

【 例 1 】strtoul 関数を使用している場合

```
char *p;
unsigned long ul;

ul = strtoul("123456789", &p, 16);
```

16 進数の 0x123456789 は 32 ビットを超過しています。また，条件 3.の比較が文字列の先頭から末尾まで成立します。
(例) 文字列"12345678"を 16 進数変換した 0x12345678 と,その次の文字を含む"123456789"を 16 進数変換した 0x123456789 の下位 32 ビットを比較して,後者が前者より大きい。
0x12345678 < 0x23456789

[期待値]	[出力結果]
ul = ULONG_MAX　errno = ERANGE	ul = 0x23456789

【 例 2 】strtol 関数を使用している場合

```
char *p;
signed long l;

l = strtol("-123456789", &p, 16);
```

16 進数の 0x-123456789 は 32 ビットを超過しています。また，条件 3.の比較が文字列の先頭から末尾まで成立します。

[期待値]	[出力結果]
l = LONG_MIN　errno = ERANGE	l = DCBA9877(-0x23456789)

【 例 3 】atoi 関数を使用している場合

```
signed int i;

i = atoi("5368709120");
```

10 進数の 5368709120(=0x140000000)は 32 ビットを超過しています。また，条件 3.の比較が文字列の先頭から末尾まで成立します。
(例) 文字列" 536870912"を 10 進数変換した 536870912(=0x20000000)と，その次の文字を含む" 5368709120" を 10 進数変換した 5368709120(=0x140000000) の下位 32 ビットを比較して，後者が前者より大きい。
0x20000000 < 0x40000000

[期待値]	[出力結果]
i = LONG_MAX	i = 1073741824(0x40000000)

【回避策】　ありません。

【改善策】　CA850 Ver.3.41 で修正しました。

No. 108　**ビットフィールドをメンバに持つ構造体型変数の初期化の制限**

【内　容】　**下記の** 1.～2.**の条件を全て満たす場合，ビットフィールドが正しく初期化されません。【例** 1**】**

【 条 件 】

1. **ビットフィールド，構造体（または共用体）の順番で連続しているメンバを持つ構造体型変数を使用している**
2. 1.**の構造体型変数のメンバを両方とも初期値を利用して初期化している**

また下記の 3.～5.**の条件を全て満たす場合もビットフィールドが正しく初期化されない場合があります。【例** 2**】**

【 条 件 】

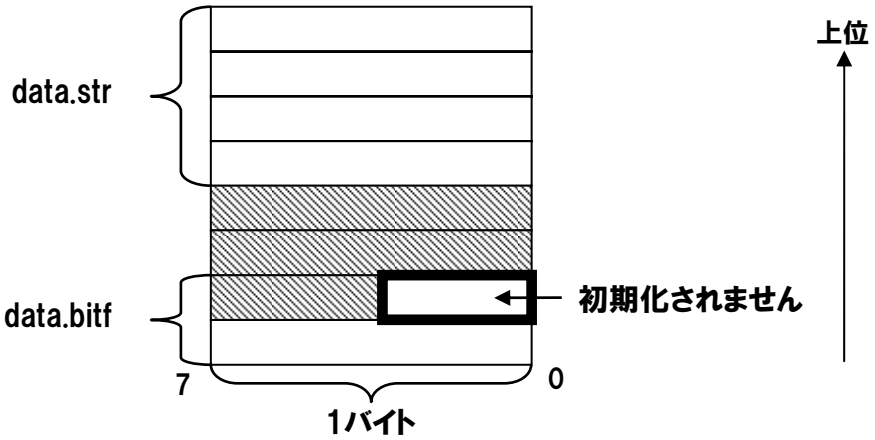
3. **構造体型自動変数の配列を使用している**
4. 3.**の構造体のメンバに，ビットフィールドと** 125 **バイト以上の要素を含む**
5. 125 **バイト以上の要素に対する初期化子を省略し，暗黙の** 0 **初期化をしている**

【 例 1 **】**

```
struct {
        int bitf : 12 ;          // ビットフィールド
        struct {                 // ビットフィールドに続いて構造体
                int s ;
        } str ;
} data = { 0xFFF, { 2 } } ;  // ビットフィールド・構造体のメンバを共に初期化
```

data.bitf **の上位** 4 **ビット分が初期化されません。**

[期待値]	**[出力結果]**
data.bitf　= 0xFFF	data.bitf　= 0xFF



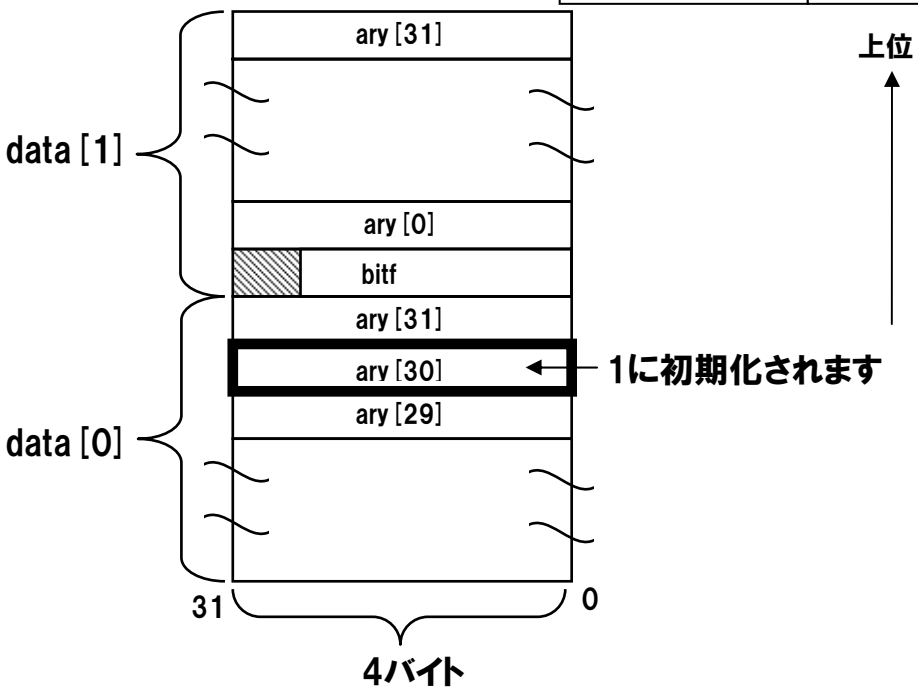
【 例 2 **】**

```
void func(void)
{
  struct {
      int bitf : 25;                 // ビットフィールド
      int ary[32];                   // 125 バイト以上の要素
  } data[2] = { { 1 }, { 1 } };      // 構造体型自動変数の配列 data を使用
  …                                  // ary を暗黙の 0 初期化
}
```

data[1].bitf **を** 1 **に設定する初期化コードのオフセットが下位方向にずれて，**data[0].ary[30] **を** 1 **に初期化してしまいます。**data[1].bitf **は不定値となります。**

[期待値]	**[出力結果]**
data[0].ary[30] = 0 data[1].bitf = 1	**data[0].ary[30] = 1** data[1].bitf **は不定値**

【内　容】
（続き）



【回避策】　下記の 1.～2. のいずれかを適用してください。

1. 初期化を代入に変更する

【 例 1 の場合 】

```
struct {
    int bitf : 12 ;
    struct {
            int s ;
    } str ;
} data ;                    // 初期化を使用しない
…
data.bitf = 0xFFF ;         //代入に変更する
data.str.s = 2 ;
```

2. ビットフィールドのメンバを構造体内の構造体にする

【 例 1 の場合 】

```
struct {
    struct {
            int bitf : 12 ;
    } str2 ;                // 構造体内の構造体にする
    struct {
            int s ;
    } str ;
} data = { { 0xFFF }, { 2 } } ;
```

【改善策】　CA850 Ver.3.41 で修正しました。
また，本制限に該当するかをチェックするツールを用意しています。
チェックツールに関しましては，弊社営業または特約店にお問い合わせください。

No. 109　条件アセンブル擬似命令のネストの制限

【内　容】　下記の 1.～4.の条件を全て満たす場合，エラーメッセージ（F3510）を出力します。

【 条 件 】

1. .elseif，または.elseifn 擬似命令を使用している
2. 1.の擬似命令で指定した式が真である
3. 1.の擬似命令に対応する擬似命令が.elseif，または.else 擬似命令である
4. 1.の擬似命令に対応するブロック間に条件アセンブル疑似命令をネストしている

【 例 】

```
.set  FLAG1, 0
.set  FLAG2, 1
.set  FLAG3, 1
.set  FLAG4, 0

.if FLAG1 == 1
  .set   TEMP, 1
.elseif FLAG2 == 1  --条件 1./条件 2.に合致
   .if FLAG3 == 1    --条件 4.に合致
     .set   TEMP, 2                    } ネスト部
   .endif
.elseif FLAG4 == 1  --条件 3.に合致
  .set   TEMP, 3
.endif
```

【回避策】　条件アセンブル擬似命令をネストする際には，下記の 1.～2. のいずれかのブロックで行ってください。

1. .if 擬似命令のブロック

```
.if FLAG1 == 1
  .set   TEMP, 1
.else
  .if FLAG2 == 1
    .if FLAG3 == 1
      .set   TEMP, 2    } ネスト部
    .endif
  .elseif FLAG4 == 1
    .set   TEMP, 3
  .endif
.endif
```

2. .elseif～.endif 擬似命令のブロック

```
.if FLAG1 == 1
  .set   TEMP, 1
.elseif FLAG4 == 1
  .set   TEMP, 3
.elseif FLAG2 == 1
   .if FLAG3 == 1
     .set   TEMP, 2    } ネスト部
   .endif
.endif
```

【改善策】　CA850 Ver.3.41 で修正しました。

No. 110　　switch，if 文内での代入の制限

【内　容】　下記の 1.～2.の条件を全て満たす場合，その後の ca850 の最適化により，switch 文あるいは if 文内の処理が不正となる場合があります。

【 条 件 】

1. C ソース上において(ア)～(ウ)の条件を全て満たす【例 1】
   (ア)　Ver.2.50 未満の場合，コンパイラの最適化オプションとして"-Os" "-Ot" のいずれかと，"-Ol"を指定しており，"-Wi,-O4" は未指定である
   Ver.2.50 以上の場合，コンパイラの最適化オプションとして"-Os" "-Ot" のいずれかを指定しており，"-Wi,-O4" は未指定である
   (イ)　switch 文，あるいは if 文内で同じ変数に値を代入する処理が並列している
   (ウ)　(イ)の代入後，同じ位置に分岐する

【 例 1 】

```
int func(int val) {
  int res;
  switch (val) {
  case 0xA: res = 0x1; break;
  case 0xB: res = 0x2; break;
  case 0xC: res = 0x3; break;
  case 0xD: res = 0x3; break;
  case 0xE: res = 0x3; break;
  default:  res = 0x3; break;
  }
  return res;
}
```

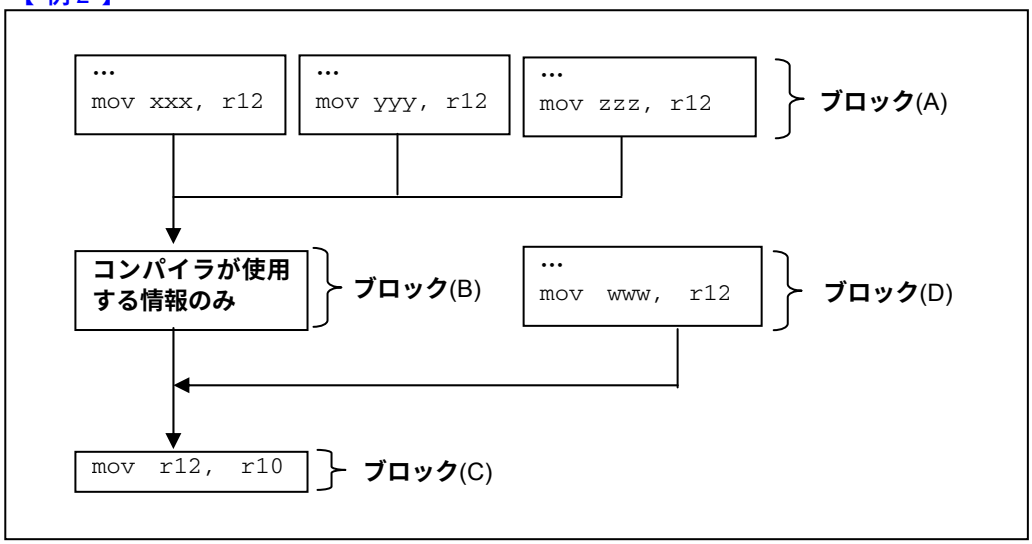
同じ変数"res"に値を代入する処理が並列

変数"res"に値を代入後，同じ位置に分岐

2.　1.の C ソース記述により出力されたアセンブラが(エ)～(ク)の条件を全て満たす【例 2】
   (エ)　2 つ以上の基本ブロック(A)の末尾で，同じレジスタへ mov 命令または ld 命令で転送する
   (オ)　(A)のブロックが全て１つのブロック(B)へ合流する
   (カ)　(B)のブロックに命令がなく，出力コードとはならないコンパイラが使用する情報が１つ以上含まれる
   (キ)　(B)の次のブロック(C)で，(A)で転送したレジスタを他のレジスタへ転送する
   (ク)　ブロック(C)に直接合流するブロックの内，(A)と同様のレジスタへ転送を行っているブロック(D)が存在する

(注意)　「基本ブロック」とは，必ず先頭の命令から入り，分岐命令までの命令並びです。

【 例 2 】



【回避策】　下記の 1.〜3. のいずれかを適用してください。

1. __asm("¥n"); を挿入する（挿入が必要な位置候補をチェックツールが出力します）

```
int func(int val) {
  int res;
  switch (val) {
  case 0xA: res = 0x1; break;
  case 0xB: res = 0x2; break;
  case 0xC: res = 0x3; break;
  case 0xD: res = 0x3; break;
  case 0xE: res = 0x3; break;
  default:  res = 0x3; break;
  }
  __asm("¥n"); ←――――――――__asm("¥n"); の挿入
  return res;
}
```

2. 最適化オプションとして”-O” 以下を指定する
3. -Wi,-O4 オプションを指定する

【改善策】　CA850 Ver.3.41 で修正しました。
また，本制限に該当するかをチェックするツールを用意しています。
チェックツールに関しましては，弊社営業または特約店にお問い合わせください。

No. 111 **キャストを伴う比較演算が不正に最適化される制限**

【内　容】 **下記の** 1.～5. **の条件を全て満たす場合に，**1.**の比較演算を不正に最適化します。**

【 条 件 】

1. **整数型同士の比較演算**(<, <=, >, >=, ==, !=)**を行っている**
2. **比較の一方が下記のいずれかのキャストを行っている**
   - signed char -> unsigned char
   - signed char -> unsigned short
   - unsigned char -> signed char
   - signed short -> unsigned short
   - unsigned short -> signed short
   - unsigend short -> signed char
3. **キャスト後の型が，比較のもう一方の型に等しい**
4. **キャストを行っていないもう一方が，最適化の影響で定数として扱われる**
5. 4.**の定数と** 2.**のキャストを無効とした場合の** 1.**の比較が，常に真または偽となる**

【 例 】

```
short s; unsigned short us;
s = -1;
…
if (s == (short)us)
```

4 行目の比較演算において，左辺の"s"**が** CA850 **の最適化の影響で定数**"-1"**として扱われた場合に，右辺の**"us"**の** short **型へのキャストを無効とした場合の比較結果は常に偽となります。そのため，**1.～5. **の全ての条件に該当します。この比較演算は不正に最適化されて，削除されます。**

【回避策】 **制限に該当する場合，不正に最適化される比較演算の行番号をチェックツールが出力します。**
不正に最適化される比較演算が記載されている関数がインライン展開される場合，その関数をコールしている行番号を出力します。
不正最適化される比較演算の直前に「__asm("¥n");**」を挿入してください。**

【 例 】

```
short s; unsigned short us;
s = -1;
…
__asm("¥n");
if (s == (short)us)
```

【改善策】 Ver.3.47 **で修正しました。**
また，本制限に該当するかをチェックするツールを用意しています。
チェックツールに関しましては，弊社営業または特約店にお問い合わせください。

No. 112　**代入文に対する不正移動の制限**

【内　容】　**下記の 1.〜2. の条件を全て満たす場合に，基本ブロック 1 内の変数 X への代入文を，基本ブロック 2 内の変数 X への代入文の後に不正に移動します。**

【 条 件 】

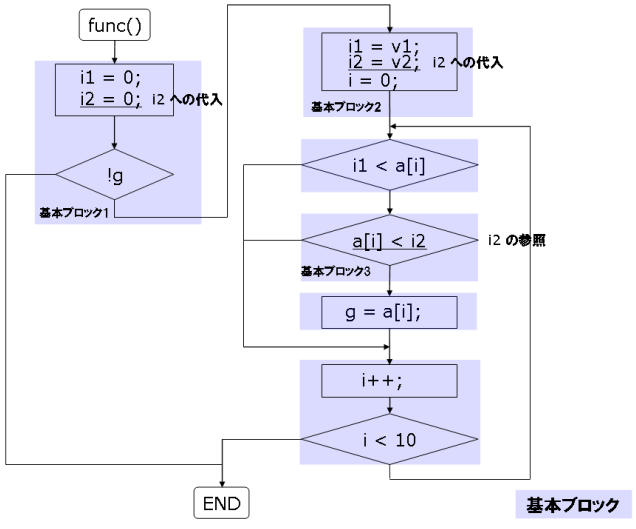
1. **最適化レベルとして**"-Og" "-O" "-Os" "-Ot"**のいずれかを指定している**
2. **関数の制御フローが**(a)〜(e)**のような基本ブロックから構成される**
   (a) volatile **指定されていない自動変数，あるいは** volatile **指定されていない引数 X に対して，基本ブロック** 1 **と基本ブロック** 2 **で値を代入し，基本ブロック** 3 **で参照する**
   (b) **基本ブロック** 2 **の X への代入値は，**volatile **指定された変数式，あるいは周辺** I/O **レジスタ名を使用した式である**
   (c) **基本ブロック** 2 **に合流するのは，基本ブロック** 1 **に限定される**
   (d) **基本ブロック** 2 **の後に，基本ブロック** 3 **に合流するパスと合流しないパスが存在する**
   (e) **基本ブロック** 1 **から基本ブロック** 3 **の間に X への間接アクセスが無い**

(注意)　**「基本ブロック」とは，一つの入り口（すなわち，内部のコードが他のコードの分岐先になっていない)と一つの出口を持ち，内部に分岐を含まないコードを指します。**

【 例 】

```
int g, a[10];
volatile int v1, v2;
void func(void) {
  int i1 = 0, i2 = 0;
  if(!g){
    int i;
    i1 = v1;
    i2 = v2;
    for(i = 0; i < 10; i++)
      if(i1 < a[i] && a[i] < i2)
        g = a[i];
  }
}
```

上記の例は以下のような基本ブロックから構成される制御フローとなり，(a)〜(e)**の条件に該当します。**



【回避策】 **制限に該当する場合，チェックツールが行番号情報を出力します。**
その行番号の直前に「__asm(“¥n”);」を挿入してください。

【 例 】

```
if(!g){
  int i;
  i1 = v1;
  i2 = v2;
  __asm("¥n");
  for(i = 0; i < 10; i++)
    if(i1 < a[i] && a[i] < i2)
      g = a[i];
}
```

【改善策】 Ver.3.47 **で修正しました。**
また，本制限に該当するかをチェックするツールを用意しています。
チェックツールに関しましては，弊社営業または特約店にお問い合わせください。

CX の制限事項一覧

1. 製品履歴

No.	仕様変更・追加／制限事項	CubeSuite パッケージ		
		V1.20		V1.40
		CX		
		V1.00	V1.01	V1.10
1	-Xno_romize オプションと-Xtwo_pass_link オプションを同時指定するとエラーになる制限	×	×	×
2	非 ASCII 文字を含む入力ファイル名の制限	×	×	×
3	別ドライブに置かれたファイルを相対パスで指定するとエラーになる制限	×	×	○
4	-Xr オプションの制限	×	×	○
5	static 修飾した割り込み関数およびスマート・コレクション修正後関数の制限	×	○	○
6	data / bss 属性セクションに配置された配列の制限	×	○	○
7	data / bss 属性セクションに配置指定した変数の制限	×	×	○
8	関数呼び出し後に参照した変数値が不正になる制限	○	○	×
9	間接代入後に参照した変数値が不正になる制限	○	○	×

×：該当する
○：該当しない
－：対象外
※：チェックツールあり

2. 使用制限事項の詳細

No. 1　-Xno_romize オプションと-Xtwo_pass_link オプションを同時指定するとエラーになる制限

【内　容】　-Xno_romize オプションと-Xtwo_pass_link オプションを同時に指定した場合に，出力ロード・モジュール・ファイルと同名のファイルが既に存在すると，F0562003 エラーとなります。

F0562003:"*file*" は、ELF 形式のオブジェクト・ファイルではありません。

【回避策】　実行前に出力ロード・モジュール・ファイルと同名のファイルを削除してください。

【改善策】　制限事項とします。

No. 2　非 ASCII 文字を含む入力ファイル名の制限

【内　容】　パスを含む入力ファイル名の中に非 ASCII 文字[注] が含まれている場合に，-Xpass_source を指定すると，下記の 1.～2. のいずれかになります。

1. アセンブラ・ソース・ファイル中にコメントとして出力される C ソース行が不正となる
2. E0592018 エラーとなる

E0592018:リスト・ファイル "*file*" のオープンに失敗しました。

（注）日本語版 OS では，非 ASCII 文字として 2 バイト文字および半角カタカナが該当します。

【回避策】　パスを含むファイル名に，非 ASCII 文字を含まないように変更してください。

【改善策】　制限事項とします。

No. 3　別ドライブに置かれたファイルを相対パスで指定するとエラーになる制限

【内　容】　入出力ファイル，またはオプションのパラメータとして，別ドライブに置かれたファイルを相対パスで指定した場合に， E0511102 エラーとなります。

E0511102：*"文字列"* オプションで指定されたファイル"*file*" が見つかりません。

【 例 】-Xstartup オプションのパラメータに指定した場合

```
>cx.exe -Cf3507 -Xstartup=c:..¥cstart.obj main.c
```

以下の E0511102 エラーとなります。

E0511102:"-Xstartup"オプションで指定されたファイル"c:..¥cstart.obj" が見つかりません。

【回避策】　別ドライブに置かれたファイルは絶対パスで指定してください。

【改善策】　CX V1.10 で修正しました。

No. 4 -Xr オプションの制限

【内　容】 複数の-Xr オプションを使用して，異なるレジスタに同じ外部変数を指定した場合にエラーとなりません。このような場合，-Xr オプションにより先に指定したレジスタに割り付けます。

【 例 】r15，r16，r17 レジスタに外部変数 A を割り付けた場合

```
>cx.exe -Cf3507 -Xreg_mode=22 -Xr15=A -Xr16=A -Xr17=A main.c
```

この場合，外部変数 A は r15 レジスタに割り付けます。

【回避策】 -Xr オプション使用時は，異なるレジスタに同じ外部変数を指定しないでください。

【改善策】 CX V1.10 で修正しました。

No. 5 static 修飾した割り込み関数およびスマート・コレクション修正後関数の制限

【内　容】 割り込み関数を static 修飾している場合，下記の 1.～2. のいずれかになります。

1. -Odelete_static_func=off オプションを指定した場合，CX が異常終了する
2. -Odelete_static_func=off オプションを指定しない場合，static 修飾した割り込み関数が削除される

また，スマート・コレクション機能による修正後関数に static 修飾している場合， F0550508 エラーとなります。

```
F0550508:定義されていない識別子 xxx が参照されています。
```

【回避策】 割り込み関数，およびスマート・コレクション修正後関数を static 修飾しないでください。

【改善策】 CX V1.01 で修正しました。

No. 6　　data / bss 属性セクションに配置された配列の制限

【内　容】　下記の 1.～4.の条件を全て満たす場合に，3.のアクセスあるいはアドレス取得が不正になる場合があります。

1. 最適化オプションとして"-O" "-Osize" "-Ospeed" のいずれかを指定している
2. data 属性セクションあるいは bss 属性セクションに配列を配置している
3. 2.の配列に対して変数を用いた添え字により，先頭以外の要素へのアクセスあるいは先頭以外の要素のアドレス取得を行っている
4. 3.の添え字が最適化の影響で定数として扱われる

【 例 】

```
#pragma section data  /* 配列 ary を.bss セクションに配置 */
int ary[7];

void f(void)
{
    int i;
    i = 1;
    ary[i] = 0;         /* 変数"i"が最適化の影響で 1 となる */
}                       /* 先頭以外の要素"ary[1]"へのアクセス */
```

【回避策】　配列の添え字として使用している変数を volatile 指定してください。

```
#pragma section data
int ary[7];

void f(void)
{
    volatile int i;
    i = 1;
    ary[i] = 0;
}
```

【改善策】　CX V1.01 で修正しました。

No. 7　data / bss 属性セクションに配置指定した変数の制限

【内　容】　別ファイルに定義されている下記の 1.～2.の条件を全て満たす変数を，C ソース中においてアクセスした場合，F0560419 エラーとなります。

1. data / bss 属性セクションに配置指定
2. GP シンボルから±32K バイトを超えた範囲に配置

【回避策】　制限に該当して F0560419 エラーとなっている全ての変数に対して，変数を使用している関数の先頭に以下のインラインアセンブラを挿入してください。

```
__asm("$data _変数名");
```

【 例 】変数 “ val ” が制限に該当している場合

```
__asm("$data _val");
```

※挿入に関しては以下の注意事項がございます。

- そのファイル内で最初に参照する関数の先頭に挿入する
  （同じファイル内の別の関数で参照していても，再挿入はしない）
- その変数を定義しているファイルには挿入しない

【改善策】　CX V1.10 で修正しました。

No. 8　**関数呼び出し後に参照した変数値が不正になる制限**

【内　容】　下記 1.～6.の条件を全て満たす場合に、関数“F2”の呼び出し後の変数"V"の値が不正になります。

【 条 件 】

1. 最適化オプションとして-O/-Osize/-Ospeed のいずれかを指定している。
2. volatile 指定されていないグローバル変数、または volatile 指定されていないファイル内 static 変数“V”が存在し、関数“F1”の中でアドレスが取られない。
3. 関数“F1”の中に、変数”V”の値を書き換える関数“F2”の呼び出しが存在し、その呼び出し前後に変数"V"の読み出しが存在する。
4. 関数“F2”の呼び出し後に変数“V”を読み出す基本ブロック“B2”から、関数"F2“の呼び出し前に変数"V"を読み出す基本ブロック"B1"にさかのぼる実行パスが 1 つに限定される。
5. 基本ブロック“B2”以降の実行パスに、関数呼び出しが存在しない。または関数呼び出しが存在するが、その関数呼び出しからさかのぼると必ず基本ブロック“B2”に到達する。
6. 関数“F1”の中に、組み込み関数 asm/nop/halt/set_il/ldsr/stsr/ldgr/stgr のいずれも含まない。

（注意）「基本ブロック」とは、必ず先頭の命令から入り、最後の命令以外からは分岐しない最大の命令の並びです。

【 例 】

```
1: int F2(void);        // この関数内で変数 V の書き換え
2: int V;
3:
4:  void F1(void)
5:  {
6:    if (V){           // V の読み出し
7:      if (F2()){      // F2 により変数 V の書き換え
8:      }
9:      else {
10:       V++;          // V の読み出し
11:     }
12:   }
13: }
```

【回避策】　以下のいずれかを適用して下さい。

1. 最適化オプションとして-O/-Osize/-Ospeed 以外を指定
2. 関数"F1"の任意の行に__asm("¥n");を挿入
3. 変数"V"を volatile 修飾

【改善策】　CX V1.11 で修正する予定です。

No. 9　　**間接代入後に参照した変数値が不正になる制限**

【内　容】　　**下記** 1.～5.**の条件を全て満たす場合に、間接代入“**A**”の後の変数"V"の値が不正になります。**

【 条 件 】

1. **最適化オプションとして**-O/-Osize/-Ospeed **のいずれかを指定している。**
2. volatile **指定されていない変数“**V**”のアドレスが、ポインタの初期値として使用される。あるいは別ファイルで変数“**V**”のアドレスが取られる。**
3. **関数“**F1**”の中で、ポインタを用いた間接代入“**A**”により変数”**V**”の値を書き換えており、その間接代入前後に変数“**V**”の読み出しが存在する。**
4. **間接代入“**A**”の後に変数“**V**”を読み出す基本ブロック“**B2**”から、間接代入“**A**”の前に変数“**A**”を読み出す基本ブロック“**B1**”にさかのぼる実行パスが** 1 **つに限定される。**
5. **関数“**F1**”の中に、組み込み関数** asm/nop/halt/set_il/ldsr/stsr/ldgr/stgr **のいずれも含まない。**

（注意）　**「基本ブロック」とは、必ず先頭の命令から入り、最後の命令以外からは分岐しない最大の命令の並びです。**

【 例 】

```
1: int  V;
2: int  j;
3: int* P = &V; // 変数"V"のアドレスをポインタの初期値として使用
4:
5: void F1(void)
6: {
7:   if (V) {   // V の読み出し
8:     *P = 0;  // 間接代入により変数 V の書き換え
9:     if (V)   // V の読み出し
10:      j = 0;
11:  }
12: }
```

【回避策】　　**以下のいずれかを適用して下さい。**

1. **最適化オプションとして**-O/-Osize/-Ospeed **以外を指定**
2. **関数**"F1"**の任意の行に**__asm("¥n");**を挿入**
3. **変数**"V"**を** volatile **修飾**

【改善策】　　CX V1.11 **で修正する予定です。**

CX ユーティリティの制限事項一覧

1. 製品履歴

No.	仕様変更・追加／制限事項	CubeSuite パッケージ		
		V1.20		V1.40
		CX		
		V1.00	V1.01	V1.10
1	構造体および共用体定義時の CubeSuite 向け情報ファイルの制限	×	○	○

×：該当する
○：該当しない
－：対象外

No. 1　**構造体および共用体定義時の** CubeSuite **向け情報ファイルの制限**

【内　容】　**下記の** 1.〜3.**の条件を全て満たす場合に，**CX **が出力する** CubeSuite **向け情報ファイルが不正になります。ただしコンパイル処理自体は正常に行われます。**

【 条 件 】

1. -Xcube_suite_info **オプションを指定している**
2. **構造体あるいは共用体（ビットフィールドを含む）を定義している**
3. 2.**に**“typedef”**を用いた型定義をしている，または** 2.**の変数に対して型修飾子**(const **または** volatile**，またはその両方**)**を付加している**

CubeSuite **向け情報ファイルが不正となることにより，**CubeSuite **の以下の機能が使用できなくなります。**

- **関数ジャンプ機能**
- **プログラム解析機能**

【回避策】　**ありません。**

【改善策】　CubeSuite V.1.30 **で修正しました。**

スタック見積もりツールの制限事項一覧

1. 製品履歴

No.	仕様変更・追加／制限事項	CubeSuite パッケージ					
		V1.00	V1.10		V1.12		V1.20	
		スタック見積もりツール					
		V4.00					V4.01
1	アセンブリ・ファイルの出力フォルダの制限	×					○

×：該当する
○：該当しない
－：対象外

2. 使用制限事項の詳細

No. 1　　**アセンブリ・ファイルの出力フォルダの制限**

【内　容】　**下記の 1.～2. の条件を全て満たす場合に，個別コンパイル・オプションを設定していないファイルに対して，スタック見積もりツールを正しく起動することができません。**

【 条 件 】

1. プロジェクトの種類として"CA850 用プロジェクト"，あるいは"CA850 用ライブラリ・プロジェクト"を選択している

2. コンパイル・オプションの"アセンブリ・ファイルの出力フォルダ"をデフォルトの %BuildModeName% から変更している

【回避策】　**下記の** 1.～2. **のいずれかを適用してください。**

1. **コンパイル・オプションの"アセンブリ・ファイルの出力フォルダ"を**
   デフォルトの %BuildModeName% **に戻す**
2. **個別コンパイル・オプションで"アセンブリ・ファイルの出力フォルダ"を設定する**

【改善策】　**スタック見積もりツール** Ver.4.01 **で修正しました。**

78K0R/Kx3 用シミュレータの制限事項一覧

1. 製品履歴

No.	仕様変更・追加／制限事項	CubeSuite パッケージ					
		V1.00	V1.10		V1.12		V1.20
		78K0R/Kx3 用シミュレータ					
		V2.42	V2.43				
1	デバッグ･ツール切断に関する制限	×	○				

×：該当する
○：該当しない
－：対象外

2. 使用制限事項の詳細

No. 1　　デバッグ･ツール切断に関する制限

【内　容】 Windows に制限ユーザ(Administrator では無いユーザ)でログインした状態で，78K0R/Kx3 シミュレータを使用する場合，デバッグ･ツール切断時に以下のエラー･メッセージが出る場合があります。
「デバッガの終了に失敗しました」(エラー番号：E0210001)
一度このエラーが発生すると，何度デバッグ･ツールを切断しようとしても同じエラー･メッセージが出るため，CubeSuite を終了できなくなります。

【回避策】 ありません。
なお，問題が発生した場合は CubeSuite を強制終了してください。

【改善策】 78K0R/Kx3 用シミュレータ Ver.2.43（CubeSuite Ver.1.10）で修正しました。

CA78K0R の制限事項一覧

1. 製品履歴

（1）コンパイラ部の制限事項一覧

No.	仕様変更・追加／制限事項	CubeSuite パッケージ						
		V1.00	V1.10		V1.12		V1.20		V1.40
		CA78K0R						
		V1.00		V1.01			V1.02	
27	-ng指定でASM文を含む関数において，分岐命令でエラーとなる制限	×		○			○	
28	ネストしたif文を抜けた直後の文に対して，行番号情報が出ない制限	×		○			○	
29	オペランドがword[BC]の命令(mov, movw)におけるアクセスの制限	×※		×※			○	
30	-qjオプション指定に関するC0101エラーとなる制限	×		×			×	
31	far領域のアドレスを long/unsigned long 型にキャストしたときの制限	×※		×※			×※	
32	##演算子を使ったマクロ展開でエラーとなる制限	×		×			×	
33	アセンブラ・ソースに割り込み関数のシンボル情報が出ない制限	×		×			×	
34	ESレジスタの設定コードに関する制限	×※		×※			×※	

×：該当する
○：該当しない
－：対象外
※：チェックツールあり

※項目の No.については，別製品の CC78K0R と同じ番号にしてありますので連続ではありません。

2. 使用制限事項の詳細

No. 27 -ng 指定で ASM 文を含む関数において，分岐命令でエラーとなる制限

【内　容】 -ng オプション指定で ASM 文を含む関数において，ASM 文より後ろにある分岐命令でエラーとなる場合があります。
以下のすべての条件を満たすときエラーとなる場合があります。

(1) 関数内に ASM 文がある。
(2) 同関数内に分岐命令を出力する文(if 文, for 文, while 文など)がある。

ただし，この時，アセンブル時にエラーとなりますので，オブジェクト・モジュール・ファイルは生成されません。　エラーとならなければ，本制限に該当しません。

【例】

```
[ .c ]
unsigned int i;
void func()
{
      do {
            __asm(“¥t DB    (1000)”);
            i++;
      } while ( i < 10) ;
}
```

【回避策】 -g オプションに変更してください。

【改善策】 CA78K0R Ver.1.01 で修正しました。

No. 28 **ネストしたif文を抜けた直後の文に対して，行番号情報が出ない制限**

【内　容】 **以下のすべての条件を満たすときに，ネストした if 文を抜けた直後の文に対して，行番号情報が出ない場合があります。　出力コードは正しいです。**
行番号情報が出なかった行にはブレークポイントの設定が出来ません。

(1)　2 つ以上の if 文が入れ子になっているネストレベル 3 以上の if 文がある。
(2)　上のネストレベルにある else が if 文の行番号より大きい。
(3)　直後に文が続く，上のネストレベルにある if 文が少なくとも１つある。

【例】

```
[ .c ]
int  f0, f1, f2, f3;
int  g0, g1, g2;
void func(void)
{
      if (f0) {
            if (f1) {          /* ネストレベル 1 の if 文 */
                  g2 = 5;
             }
             else if (f2) {    /* ネストレベル 2 の if 文 */
                  g2 = 4;
             }
             else if (f3) {    /* ネストレベル 3 の if 文(条件(1))  */
                  g2 = 3;
             }
             else {
                  g2 = 2;
             }
             g0 = 0x1234;      /* ネストレベル 1 の if 文の直後に文あり(条件(3))  */
             g1 = 0x5678;
        }
        else {                 /* ここの else がネストレベル 3 の if 文の */
             g2 = 1;           /* 行番号より大きい(条件(2))              */
        }
}
```

【回避策】 **ネストした** if **文を抜けた直後の文の前に，空行を数個挿入してください。**
上記例では「g0 = 0x1234;**」の前に，空行を数個挿入してください。**

【改善策】 CA78K0R Ver.1.01 **で修正しました。**

No. 29 **オペランドがword[BC]の命令(mov, movw)におけるアクセスの制限**

【内　容】 **以下の条件(1)あるいは(2)のいずれかの場合、出力コードが不正となります。**
オペランド"word[BC]"で word+BC のアドレスが 10000H を超えると意図しないアドレスをアクセスします。

(1) **「ポインタ-定数」の形で減算を使用して間接参照する場合**
ただし、以下の条件は除きます。

(a) **ラージ･モデルを使用し、ポインタの指す先を near 領域指定していない場合**
(b) **ポインタの指す先を far 領域指定して使用している場合**
(c) **ポインタの指すアクセス先サイズが 1 バイトで、減算する定数が 1 の場合**
(d) **ポインタの指すアクセス先サイズが 1 バイトで、減算する定数が 2、最適化条件としてスピード優先最適化[注]を行っていない場合**
(e) **ポインタの指すアクセス先サイズが 2 バイトで、減算する定数が 1、最適化条件としてスピード優先最適化[注]を行っていない場合**

注：-qx1 を指定している場合、または-q オプションで"l"を指定していない場合

(2) **配列の要素を参照する際に、添字の式中の変数が負の値となる場合、もしくは「集成体*1 を示すアドレス+オフセット(負の値となる変数を含む式)」の形で間接参照する場合**

***1 構造体、配列です。**

【条件(1)の例】

```
------*.c-------------
unsigned char x[5], *ucp1;
unsigned short y[5], *usp1;
signed short si1;
void func1()
{
/*********ポインタのアクセス先のサイズが 1 バイトの場合**********/
  x[0] = *(ucp1 - 1);       /* 問題なし */
  x[1] = *(ucp1 - 2);       /* スピード優先最適化指定時は問題発生、未指定時は問題なし */
  x[2] = *(ucp1 - 3);       /* 問題発生 */
  x[3] = ucp1[-4];          /* 問題発生 *(ucp1 -4)と同じ意味となる記述 */
  x[4] = *(ucp1 + si1 - 5); /* 問題発生 */
/*********ポインタのアクセス先のサイズが 2 バイトの場合**********/
  y[0] = *(usp1 - 1);       /* スピード優先最適化指定時は問題発生、未指定時は問題なし */
  y[1] = *(usp1 - 2);       /* 問題発生 */
  y[2] = *(usp1 - 3);       /* 問題発生 */
  y[3] = usp1[-4];          /* 問題発生 *(usp1 -4)と同じ意味となる記述 */
  y[4] = *(usp1 + si1 - 5); /* 問題発生 */
 }

--------*.asm---------(上記のアセンブル・ソースの抜粋)
; line    25 :   x[2] = *(ucp1 - 3);    /* 問題発生 */
$DGL   0,9
       movw  bc,!_ucp1      ;[INF] 3, 1
       mov   a,65533[bc]    ;[INF] 3, 1 ← 不正なアドレスをアクセスする可能性がある
       mov   !_x+2,a        ;[INF] 3, 1
---
```

【条件(2)の例】

```
unsigned char x[4], *ucp1, uca1[10];
signed short sidx;
signed char cidx;
void func2()
{
   x[0] = uca1[cidx];         /* cidx が負の値で問題発生、正の値(0 含む)では問題なし */
   x[1] = *(&uca1[3] + cidx); /* cidx が負の値で問題発生、正の値(0 含む)では問題なし */
   x[2] = uca1[sidx];         /* sidx が負の値で問題発生、正の値(0 含む)では問題なし */
   x[3] = *(&uca1[4] + sidx); /* sidx が負の値で問題発生、正の値(0 含む)では問題なし */
}

--------*.asm---------(上記のアセンブル・ソースの抜粋)
; line    36 : x[0] = uca1[cidx]; /* cidx が負の値で問題発生、正の値(0 含む)では問題なし*/
$DGL  0,20
       mov     a,!_cidx       ;[INF] 3, 1
       sarw    ax,8           ;[INF] 2, 1
       movw    bc,ax          ;[INF] 1, 1
       mov     a,_uca1[bc]    ;[INF] 3, 1  ← 不正なアドレスをアクセスする可能性がある
       mov     !_x,a          ;[INF] 3, 1
--------------------
```

【回避策】 ・条件(1)の回避策

「ポインタ-定数」の形で減算を使用して間接参照する場合は「ポインタ-定数」の結果を一度別のポインタに代入してからアクセスしてください。

例 1：

<回避前>	<回避後>

unsigned char x, *ucp1;
void func1()
{
x = *(ucp1 - 3);
 	
}
	unsigned char x, *ucp1, *ucp2;
	void func1()
	{
	ucp2 = ucp1 - 3;
	x = *ucp2;
	}

例 2：

<回避前>	<回避後>

unsigned char x, *ucp1;
void func2()
{
x = ucp1[-4];
 	
}
	unsigned char x, *ucp1, *ucp2;
	void func2()
	{
	ucp2 = ucp1 - 4;
	x = *ucp2;
	}

・条件(2)の回避策
配列の添字が符号付きの式で配列の要素を参照する際に添字の式中の変数が負の値となる場合、もしくは「集成体を示すアドレス+オフセット(負の値となる変数を含む式)」の形で間接参照する場合、上記の式の演算結果をポインタに代入してからアクセスしてください。

例 1：

<回避前>	<回避後>

unsigned char x, uca1[10];
signed char cidx;
void func1()
{
　　x = uca1[cidx];
 	
}
	unsigned char x, uca1[10], *ucp1;
	signed char cidx;
	void func1()
	{
	　　ucp1 = uca1 + cidx;
	　　x = *ucp1;
	}

例 2：

<回避前>	<回避後>

unsigned char x, uca1[10];
signed char cidx;
void func2()
{
　　x = *(&uca1[3] + cidx);
 	
}
	unsigned char x, uca1[10], *ucp1;
	signed char cidx;
	void func2()
	{
	　　ucp1 = &uca1[3] + cidx;
	　　x = *ucp1;
	}

【改善策】 CA78K0R Ver.1.02 で修正しました。
また、本制限に該当するか否かをチェックするツールを用意しております。
チェックツールに関しましては、販売店または特約店にお問い合わせください。

No. 30 -qj オプション指定に関する C0101 エラーとなる制限

【内　容】 以下の条件をすべて満たすときに、内部エラー（C0101）になります。
(1) ジャンプ最適化オプション（-qj）を有効にする。
(2) if(0)、あるいは if(1)の else 節など、実行しない処理中で以下のいずれかを記述する。
- 配列、構造体、共用体の初期化処理
- テーブルジャンプを伴う switch 文

```
[*.c]
struct t1 {
       unsigned char uc1;
};
char c;
void func1()
{
    struct t1 st0 = {0x07};
    if(0) {
            struct t1 st0 = {0x08};     /* 実行しない処理 */
            c++;
    }
}
```

【回避策】 以下のいずれかで回避してください。
・ ジャンプ最適化オプション(-qj)を無効にしてください。
・ 実行しない処理自体を削除する、あるいは#if 0 で囲んでコンパイル対象としないでください。

【改善策】 CA78K0R Ver.1.10 で修正いたします。

No. 31 far 領域のアドレスを long/unsigned long 型にキャストしたときの制限

【内　容】 far 領域のアドレスを long/unsigned long 型にキャストした値を初期値とすると、アドレスの最上位 1 バイトの値が 0FH 固定となります。

```
[*.c]
__far int const fi1 = 5;
__far unsigned long const ula1[] = { &fi1 };
__far unsigned long const ula2[] = { (unsigned long)&fi1 };

[*.asm]
@@CNSTL  CSEG  PAGE64KP
_fi1:    DW    05H    ; 5
_ula1:   DG    _fi1
_ula2:   DW    loww (_fi1)
         DW    0FH    ; 15
```

【回避策】 アドレスが初期値の時は、long/unsigned long 型にキャストしないでください。

【改善策】 CA78K0R Ver.1.10 で修正いたします。
また、本制限に該当するか否かをチェックするツールを用意しております。
チェックツールに関しましては、販売店または特約店にお問い合わせください。

No. 32 ##演算子を使ったマクロ展開でエラーとなる制限

【内　容】 ## 演算子の直後が、関数形式マクロのパラメータでなく、大小英字でも下線 _ でもない時、E0803 エラーが出る場合があります。
演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成する時は、E0711,E0301 などの E0803 以外のエラーが出る場合があります。

【例１】
```
[*.c]
#define m1(x) (x ## .c1 + 23)
#define m2(x) (x ## .c1 + 122)
struct t1 {
    unsigned char c1;
} st1;
unsigned char x1, x2;
void func1()
{
        x1 = m1(st1) + 100;     /* E0803 エラー (NG) */
        x2 = m2(st1) + 1;       /* ノーエラー    (OK) */
}
```

【例２】
```
[*.c]
#define m3(x) (x ## 1)
unsigned char x3, uc1;
void func2()
{
        x3 = m3(uc);            /* E0711, E0301 エラー (NG) */
}
```

【回避策】 ## 演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成しない時は、
演算子を使わないでください。

```
#define m1(x)      (x ## .c1 + 23)
#define m2(x)      (x ## .c1 + 122)
          ↓
#define m1(x)      ((x).c1 + 23)
#define m2(x)      ((x).c1 + 122)
```

演算子の直前のトークンと直後のトークンを連結して、別のトークンを生成する時は、
演算子の直後に関数形式マクロのパラメータを置いてください。

```
#define m3(x)      (x ## 1)
unsigned char x3, uc1;
void func2()
{
          x3 = m3(uc);
}
          ↓
#define m3(x, y)   (x ## y)
unsigned char x3, uc1;
void func2()
{
          x3 = m3(uc, 1);
}
```

【改善策】 CA78K0R Ver.1.10 で修正いたします。

No. 33 **アセンブラ・ソースに割り込み関数のシンボル情報が出ない制限**

【内　容】 以下の条件をすべて満たすときに、リンク時に E3405 エラーになります。
(1) #pragma interrupt による割り込み関数のベクタテーブルの生成指定がある。
(2) 同一ソース内に割り込み関数の定義がない。
(3) -no, アセンブラ・ソース・モジュール・ファイル出力(-a or -sa)、デバッグ情報出力(-g) オプションを有効にする

```
[*.c]
#pragam interrupt INTP0 inter
/*                              割り込み関数の定義がコンパイル対象ではない
__interrupt void inter()
{
            :
}
*/
```

【回避策】 割り込み関数を定義する、あるいはオブジェクト・モジュール・ファイルを出力してください。

【改善策】 CA78K0R Ver.1.10 で修正いたします。

No. 34 ES レジスタの設定コードに関する制限

【内　容】 間接参照を除く浮動小数点型変数のインクリメント／デクリメント演算後に far 変数を参照すると、ES レジスタの設定コードが出ない場合があります。

[*.c]

```
__far char c1;
float f1;
void func()
{
        ++f1;
        c1 = 5;
}
```

【回避策】 間接参照を除く浮動小数点型変数のインクリメント／デクリメント演算直後にダミーのポインタ間接参照式を記述してください。

[*.c]

```
__far char c1;
float f1;
void func()
{
        char *cp1;              /* ダミー変数*/
        ++f1;
        *cp1;                   /* ダミーの間接参照式 */
        c1 = 5;
}
```

【改善策】 CA78K0R Ver.1.20 で修正いたします。
また、本制限に該当するか否かをチェックするツールを用意しております。
チェックツールに関しましては、販売店または特約店にお問い合わせください。

リアルタイム OS 連係の制限事項一覧

1. 製品履歴

No.	対象リアルタイム OS	仕様変更・追加／制限事項	CubeSuite パッケージ										
			V1.00	V1.10		V1.12				V1.20			
			CubeSuite										
			V1.00	V1.10	V1.11	V1.12	V1.13	V1.14	V1.15	V1.20	V1.21	V1.30	V1.31
1	RX78K0R，RX850V4，RX850 Pro	システム・パフォーマンス・アナライザのオブジェクト ID 末尾に不正な文字列がつく制限	－	×	○	○	○	○	○	○	○	○	○

×：該当する
○：該当しない
－：対象外

2. 使用制限事項の詳細

No. 1　システム・パフォーマンス・アナライザのオブジェクト ID 末尾に不正な文字列がつく制限

【対　象】　RX78K0R，RX850V4，RX850 Pro

【内　容】　システム・パフォーマンス・アナライザで， 表示されるオブジェクト ID（タスクやセマフォなどの ID）の末尾に不正な文字列が付加されます。

例：実行遷移ウインドウ上のタスク ID 表示

【回避策】　ありません。

【改善策】　CubeSuite Ver.1.11 で修正しました。

ビルド・ツールの制限事項一覧

1. 製品履歴

No.	対象ビルド・ツール	仕様変更・追加／制限事項	CubeSuite パッケージ								
			V1.00	V1.10		V1.12		V1.20			
			CubeSuite								
			V1.00	V1.10	V1.11	V1.12	V1.13	V1.14	V1.15	V1.20	V1.21	V1.30/V1.31
1	CA78K0	-qc オプションがプロジェクト・ファイルに保存されない制限	×			○						
2	全ツール	個別オプションを指定したファイルのビルドの制限	×			○						

×：該当する
○：該当しない
－：対象外

2. 使用制限事項の詳細

No. 1　-qc オプションがプロジェクト・ファイルに保存されない制限

【内　容】 ビルド・ツール CA78K0 の[最適化] → <最適化を行う> → はい(詳細設定)を選択し，「[最適化](詳細) → <char 型演算を符号拡張しない> → いいえ」に変更をして，再度プロジェクト・ファイルを読み込むと，「<char 型演算を符号拡張しない> → はい」に戻ります。

【回避策】 「<最適化を行う> → いいえ」に設定して，<その他の追加オプション>に必要な最適化オプションを設定してください。

【改善策】 CubeSuite Ver.1.12 で修正しました。

No. 2　個別オプションを指定したファイルのビルドの制限

【内　容】 ビルド・ツールの[個別コンパイル・オプション] ，または[個別アセンブル・オプション] → <ビルド・ツールに指定した全体インクルード・パスも使用する> → はい　を選択した場合，その個別オプションを指定したソースファイルはビルド処理にて必ずコンパイルされます。

【回避策】 「<ビルド・ツールに指定した全体インクルード・パスも使用する> → いいえ」に設定して、全体オプションに指定しているインクルード・パスを個別オプションにも同様に設定してください。

【改善策】 CubeSuite Ver.1.12 で修正しました。

プログラミング・ツールの制限事項一覧

1. 製品履歴

No.	対象プログラミング・ツール	仕様変更・追加／制限事項	CubeSuite パッケージ							
			V1.00	V1.10		V1.12		V1.20		V1.40
			CubeSuite							
			V1.00	V1.10	V1.11	V1.12	V1.13/V1.14/V1.15	V1.20	V1.21/V130/V131	V1.40
1	QB-Programmer	特定のヘキサ・ファイルを読み込むと CubeSuite が強制終了してしまう制限	×				○			

×：該当する
○：該当しない
－：対象外

2. 使用制限事項の詳細

No. 1　**特定のヘキサ・ファイルを読み込むと CubeSuite が強制終了してしまう制限**

【内　容】 **下記条件を満たすヘキサ・ファイルを，**QB-Programmer **のプロパティで読み込むと，**CubeSuite **が強制終了してしまいます。**

【条件】

ヘキサ・ファイルのファイル・サイズに 32 **バイトを加えた値が** 4096 **バイトの倍数になるヘキサ・ファイル**

【補足】

QB-Programmer **のプロパティで選択できるヘキサ・ファイルが１つのみの場合，**QB-Programmer **のプロパティを開くタイミングで読み込みます。ヘキサ・ファイルが複数選択できる場合，**QB-Programmer **のプロパティを開き，ファイル名を選択するタイミングで読み込みます。**

【回避策】 **ありません。**

【改善策】 CubeSuite Ver.1.13 **で修正しました。**