CUSTOMER NOTIFICATION

SUD-DT-04-0058

January 30, 2004

Koji Nishibayashi, Senior System Integrator
Microcomputer Group
2nd Solutions Division
Solutions Operations Unit
NEC Electronics Corporation

CP(K), O

# V850 Series C Compiler Package

# CA850 V2.61

# Operating Precautions

Be sure to read this document before using the product.

Thank you for purchasing V850 Series C compiler package CA850 Ver. 2.61.
This document explains the modifications from V850 Series C compiler package CA850 Ver. 2.60 to Ver. 2.61, cautions, and usage restrictions. Be sure to read this document before using CA850 Ver. 2.60.

## 1. USER'S MANUAL

The following user's manuals are available for this version as a PDF file only.

| Document Name | Document Number |
|---|---|
| CA850 C Compiler Package - C Language | U16054E |
| CA850 C Compiler Package - Assembly Language | U16042E |
| CA850 C Compiler Package - Operation | U16053E |
| CA850 C Compiler Package - Coding Technique | U16076E |
| PM plus **Note** | U16569E |

**Note** Only the Windows version PM plus user's manual is supplied.

The above manuals are user's manuals (including online help) for CA850 Ver. 2.50. Refer to the documents listed below for the functions added in Ver. 2.60.

| Document Name | Document Number |
|---|---|
| V850 Series C Compiler Package CA850 V2.60 Operating Precautions | SUD-DT-03-0257-E |
| V850 Series C Compiler Package CA850 Document Modifications | SBG-DT-03-0219-E |

Refer to this document (SUD-DT-04-0058) for the functions added in Ver. 2.61.

## 2. SUPPORTED TOOLS

Use the following version of PM plus when using CA850 Ver. 2.61.

- PM plus Ver. 5.10

Note that the following version of the project manager is not supported.

- PM plus Ver. 5.10 or earlier
- All versions of PM

Use the following versions of the integrated debugger or system simulator when using in combination with CA850 Ver. 2.61 and PM plus.

| Debugger/Simulator Name | Version |
|---|---|
| Integrated debugger ID850 | Ver. 2.50 or later |
| Integrated debugger ID850NW | Ver. 2.50 or later |
| Integrated debugger ID850NWC | Ver. 2.50 or later |
| System simulator SM850 | Ver. 2.50 or later |

## 3. CHANGES FROM Ver. 2.60 TO Ver. 2.61

### 3.1 Replacement of Assembler

The assembler as850 (Ver. 3.20), which is included with CA850 Ver. 2.61, has been replaced by one that can avoid the restriction on conflict between the **sld** instruction and interrupt in the 32-bit microcontrollers V850E/xxx and V850ES/xxx. See the documents of each device for details of this restriction.

### 3.2 Addition of Restrictions on CA850

Restrictions No. 88 and No.89 have been added.

No.88 Restriction on generating codes for complex expression

[Description]

An illegal code may be generated or a compile error may be output (E3228: illegal operand (must be register)) when executing a complex expression that includes many operations (at least four operators are included) and the expression satisfies the conditions shown below.

This bug may occur even if the number of operators is less than 4 when expressions are combined internally by optimization processing and satisfy the conditions shown below.
This bug more likely occurs when the number of temporary registers is insufficient upon code generation and the stack must be used instead. However, this bug does not always occur even if the conditions shown below are satisfied because it depends on the status of use of the registers and stack upon code generation.

The conditions under which this bug may occur are shown below. The conditions vary according to the CPU used (V850 core or V850E/V850ES core) and the version of the CA850.

Conditions:

(1) When using V850E/V850ES core and CA850 Ver.2.40/2.41, or when using V850 core and CA850 Ver. 2.40/2.41/2.50/2.60

(a) The operation includes a reference to a member that uses a “structure packing function”, and the member is

(a-1) a “2-byte type structure member allocated to an odd address”: See Example 1.

(a-2) a “4-byte type structure member allocated to an address that cannot be divided by 4”: See Example 2.

(b) The operation includes a reference from a structure pointer variable that uses a “structure packing function”, the packing value of the structure is 1, and the reference is to a 2-byte or 4-byte type member: See Example 1.

This bug may occur if any of (a-1), (a-2), or (b) is satisfied.

(2) When using V850E/V850ES core and CA850 Ver. 2.50/2.60

(a) The operation includes a reference to a member that uses a “structure packing function”, and:

(a-1) the member is a “2-byte or 4-byte type structure member allocated to an odd address”: See Example 1.

(a-2) the structure is allocated to the “tidata section”, and the member is a “2-byte type structure member allocated to an odd address” or “4-byte type structure member allocated to an address that cannot be divided by 4”: See Examples 1 and 2.

(b) The operation includes a reference from a structure pointer variable that uses a “structure packing function”, the packing value of the structure is 1, and the reference is to a 2-byte or 4-byte type member: See Example 3.

(c) The operation includes the include function __sasf() (the operation result of the __sasf function is used).

This bug may occur if any of (a-1), (a-2), (b), or (c) is satisfied.

**Caution** A “structure” includes the “bit field”. The “bit field members” are affected by “type”, not by “bit width”. For example, “int a:8;” is a 4-byte type member, and “short b:8;” is a 2-byte type member.

Example 1:

```
/* Example of member allocated to odd address */
#pragma pack(1)
struct {
    char pad1;
        ...
      /*
        Up to here, the total size of members from the start is an odd
number
      */
    short mem; /* odd address */
};
```

Example 2:

```
/* Example of member allocated to address that cannot be divided by 4 (1)
*/
#pragma pack(1)
struct {
   char pad1;
       ...
     /*
       Up to here, the total size of members from the start is "multiple
of 4 + 1"
     */
   int mem; /* odd address */
};

/* Example of member allocated to address that cannot be divided by 4 (2)
*/
#pragma pack(1)
struct {
```

Example 3:

```
/* Example when packing value is 1 and "2-byte or 4-byte type member is
referenced by structure pointer variable" */

#pragma pack(1)
struct {
   char c;
   int  i;
} *pst1;

int i1, i2, i3, i4;

int func() {
   return((~i1) << ((~i2)<<(((i3)+(i4)) << (pst1->i))));
                                              /* pst1->i is applicable */
}
```

[Workaround]

Temporarily store the structure member and the operation result of the include function __sasf() included in the expression to which the bug applies in a local variable that has been declared as volatile, and use the local variable for the expression.

Example:

```
/* When bug applies to "st1.i" in argument of return */

#pragma pack(1)
struct {
    char c;
    int  i;
} st1;

int i1, i2, i3, i4, i5;

int func() {
    return((~i1) << ((~i2)<<(((i3)+(i4)) << (i5 + st1.i))));
}
```

Modified

```
/* Modification to prevent bug */

#pragma pack(1)
struct {
    char c;
    int  i;
} st1;

int i1, i2, i3, i4, i5;

int func() {
    volatile int  tmp = st1.i;
    return((~i1) << ((~i2)<<(((i3)+(i4)) << i5 + tmp)));
}
```

Temporarily store the value of **st1.i** in a variable that has been declared as **volatile**, and use the variable for the location to which the bug applies.

[Correction]

This bug will be corrected in Ver. 2.70.

A tool used to check whether this restriction applies or not is available.

Contact an NEC Electronics sales representative or distributor for details.

No.89 Restriction on argument of inline function

[Description]

An illegal code is generated if a function on which inline expansion is performed (hereafter referred to as an inline function) satisfies the both following conditions.

- The address of an argument of the inline function is acquired, and the address is cast to a pointer whose type is different from the argument, and the contents of the argument are referenced.
- The argument above is not used by any other location.

The address resulting from cast becomes illegal where the inline function is expanded.
If there is a function for which #pragma inline is specified in the program, this bug may apply to the function. In addition, a function is expanded inline and this bug may also occur depending on the optimization options “-Os” and “-Ot”, and the combination of options related to inline expansion.

Example of codes that cause bug:

```
#pragma inline subfunc

unsigned int s;

static void
subfunc(int i) {        /* this function is expanded inline */
    s = *((unsigned int*)&i); /* cast */
}

void
func(int ii) {
    subfunc(ii);
}
```

[Workaround]
Implement any of the following workarounds
(1) Suppress inline expansion.
(a) When using CA850 Ver. 2.40 or Ver. 2.41, implement (a-1) or (a-2).
(a-1) Unselect #pragma inline and specify an optimization option **other than** “-Os” and “-Ot”.
(a-2) Specify the options shown below if the #pragma inline specification remains as is (the source not modified) and the optimization options are not modified. The option to be specified varies depending on “#pragma inline specified or not” and “combination of optimization options”.

<table>
<tr><th></th><th>#pragma inline Specified</th><th>#pragma inline Not Specified</th></tr>
<tr><td>When optimization option other than -Os and -Ot is specified</td><td rowspan="3">Specify all the following options.<br>-Wp,-N0<br>-Wp,-G0<br>-Wp,-Sn</td><td>This bug does not apply.<br>((a-1))</td></tr>
<tr><td>When optimization option -Ot is specified</td><td>Specify all the following options.<br>-Wp,-N0<br>-Wp,-G0<br>-Wp,-Sn</td></tr>
<tr><td>When optimization option -Os is specified</td><td>-Wp,-Sn</td></tr>
</table>

[How to specify options in PM and PM plus]
Input 0 in the Code Threshold field for specifying the -Wp,-N0 option, and 0 in the Stack Threshold field for specifying the -Wp,-G0 option on the Optimization2 tab in the Compiler Options dialog box. Input “-Wp,-Sn” in the Any Option field on the Others tab in the Compiler Options dialog box for specifying the -Wp,-Sn option.

(b) When using CA850 Ver. 2.50 or Ver. 2.60, implement (b-1) or (b-2).

(b-1) Unselect #pragma inline and specify an optimization option **other than** "-Ot"

(b-2) Specify the options shown below if the #pragma inline specification remains as is (the source not modified) and the optimization options are not modified. The option to be specified varies depending on "#pragma inline specified or not" and "combination of optimization options".

| | #pragma inline Specified | #pragma inline Not Specified |
|---|---|---|
| When optimization option other than -Os and -Ot is specified | -Wp,-no_inline | This bug does not apply. ((b-1)) |
| When optimization option -Ot is specified | | **-Wp,-no_inline** |
| When optimization option -Os is specified | | This bug does not apply. ((b-1)) |

[How to specify options in PM and PM plus]

Specify "No Expantion [-Wp,-no_inline]" in the Control of Inline Expansion drop-down menu on the Detail of Optimization tab in the Compiler Options dialog box.

(2) Assign the argument of the inline function to another variable before using the argument.

(3) Specify the "-Wp,-a-" option, which is used to suppress the function to delete unreferenced arguments.

When setting in PM or PM plus, describe "-Wp,-a-" in the Any Option field on the Others tab in the Compiler Options dialog box.

[Correction]

This bug will be corrected in Ver. 2.70.

A tool used to check whether this restriction applies or not is available.

Contact an NEC Electronics sales representative or distributor for details.