date: 2003/01/21

HITACHI SEMICONDUCTOR TECHNICAL UPDATE

Classification of Production	Development Environment				No	TN-CSX-047A/E
THEME	H8S,H8/300 Series C/C++ compiler Package Updates	Classification of Information	[✓ 1. Spec change] 2. Supplement of Documents 3. Limitation of Use	4. Change of Mask 5. Change of Production Line		
PRODUCT NAME	PS008CAS5-MWR PS008CAS4-MWR PS008CAS4-SLR PS008CAS4-H7R	Lot No.	Reference Documents	H8S, H8/300 Series C/C++ Compiler Assembler Optimizing Linkage Editor ADE-702-247 Rev. 1.0	Rev.	Effective Date
		All			1	Eternity

H8S,H8/300 series C/C++ compiler Package is updated to Ver.5.0.03 for Windows® and Ver.4.0.07 for UNIX.

Refer to the attached document, PS008CAS5-030115E, for details.

A user who has the following product should be informed.

For Windows® : H8S,H8/300 series C/C++ compiler PackageVer.5.0, Ver. 5.0.01 or Ver.5.0.02 of PS008CAS5-MWR.
H8S,H8/300 series C/C++ compiler Package Ver.4.0, Ver.4.0r1, Ver.4.0A, Ver.4.0Ar1 or Ver.4.0Ar2 of PS008CAS4-MWR.

For SPARC® : H8S,H8/300 series C/C++ compiler Package Ver.4.0, Ver.4.0A, Ver.4.0Ar1, Ver.4.0B, Ver.4.0.05 or Ver.4.0.06 of PS008CAS4-SLR.

For HP9000 : H8S,H8/300 series C/C++ compiler Package Ver4.0, Ver.4.0A, Ver.4.0Ar1, Ver.4.0B, Ver.4.0Br1, Ver.4.0.05 or Ver.4.0.06 of PS008CAS4-H7R.

Attached:
"Updates in H8S,H8/300 Series C/C++ compiler Package Ver.5.0.03 for Windows® and Ver.4.0.07 for UNIX"
(PS008CAS5-030115E) , 4 pages

PS008CAS5-030115E

Updates in H8S,H8/300 Series C/C++ compiler Package Ver. 5.0.03 for Windows® and 4.0.07 for UNIX

1. Hitachi Embedded Workshop 2 (updates only in Windows® Ver. 5.0.03 package)

1.1 Supporting Drag&Drop operation to add the variable to Watch Window

When you add a variable to Watch window, you drag the variable on Editor and drop it on Watch window.

1.2 Supporting an Out-of-process Server for HEW

Hew server is supported. It is based on COM technology and Out-of-process server. See the additional document about it.

1.3 Adding and Modifying the Data Generated by the Project Generator

Project generation of the following CPU has been newly added:
H8S/2168F, H8S/2367F, H8S/2375R, H8S/2377R, H8S/2628F
The I/O definition file (iodefine.h) of the following CPU has been modified:
H8S/2148, H8S/2612, H8S/2678

2. Compiler (Ver. 4.0.03 -> Ver. 4.0.04)

2.1 Correction of incorrect code related to a unary minus operator

Fixed is the following problem. The evaluation of a unary minus operator might be wrong if the unary minus operator is applied after the sign extension by the type conversion from a char, short or int type variable, including an implicit type conversion.

[The example of the bug #1]

[The source program]

```
char c:
short s;
int i;
long l;
void sub()
{
    s = –c;                     /* when c=–128, s=–128 (s=128 is correct) */
    i = –c;                     /* when c=–128, i=–128 (i=128 is correct) */
    l = – (long)c;              /* when c=–128, s=–128 (s=128 is correct) */
    l = – (long)s;              /* when s=–32768, l=–32768 (l=32768is correct) */
    l = –(long)i;               /* when i=–32768, l=–32768 (l=32768is correct) */
}
```

[The incorrect code] s=–c in the above source program is incorrectly compiled as follows.

```
MOV.B    @_c,R0L
NEG.B    R0L
EXTS.W   R0
MOV.W    R0,@_s
```

[The correct code] s=–c in the above source program should be compiled as follows

```
MOV.B    @_c,R0L
EXTS.W   R0
NEG.W    R0
MOV.W    R0,@_s
```

[The example of the bug #2]

```
char  c;
int  i1,i2;

i1 = –c + i2;                   /* when c=–128, -c would be –128 (-c is actually 128) */
```

[The example of the bug #3]

```
char c;
int  func();
int func()
{
      return( –c );                  /* when c=–128, -c would be –128 (-c is actually 128) */
}
```

2.2. Correction of incorrect generation of an AND instruction

Fixed is the following problem. An AND instruction might be incorrectly generated depending on the environment of the compiler (see the [Remark] below) if a bit-wise AND operator is applied after a shift operator or if the same compound assignment operators of &=, |= or ^= is applied to the same variable in succession.

[The example of the bug #1]

[The source program]

```
unsigned int X, Y;
void sub( void )
{
      X = (Y >> 14) & 0x2 ;
}
```

[The incorrect code]

```
MOV.W    @_Y,R0
ROTL.W   #H'2,R0
AND.L    #H'20002,ER0   ; Incorrect code
MOV.W    R0,@_X
```

[The correct code]

```
MOV.W    @_Y,R0
ROTL.W   #H'2,R0
AND.W    #H'2,ER0        ; Correct code
MOV.W    R0,@_X
```

[The example of the bug #2]

[The source program]

```
sub( int Y )
{
      Y &= 0x3;
      Y &= 0x2;
      return Y ;
}
```

[The incorrect code]

```
AND.L    #H'20002,ER0   ; Incorrect code
```

[The correct code] s=-c in the above source program should be compiled as follows.

```
AND.W    #H'2,R0         ; Correct code
```

[Remark]

The occurrence of the bug depends on the contents of the host computer’s memory area freed by the compiler. Even with the same source program and the same options are specified, the bug may or may not occur depending on the environment (or host operating system) of the compiler.

2.3. Activation of the loop invariant code motion

Fixed is the following problem. The optimization of the loop invariant code motion may not work depending on the environment of the compiler (see the [Remark] below).

[The example of the bug #1]

[The source program]

```
for (  ...  ;  ...  ;  ...  ){
    for (  ...  ;  ...  ;  ...  )
      X = 0;
}
```

A loop like this is compiled as follows.

[The code that the optimization does not work]

```
…
BRA      L1              ; (A)
```

```
L2:
   …
   BRA            L3
L4:
   …
   SUB.W          E0,E0                 ; (X)
   MOV.W          E0,@_X
L3:
   …
        Bcc       L4
L1:
        …
        Bcc       L2                    ; (B)
```

If the register E0 at (X) above is not used at any other point between (A) and (B), the code of (X) can be moved to the loop entrance as shown below.

[The code the optimization has worked]

```
        …
        SUB.W     E0,E0                 ;<----------+ code motion
        BRA       L1                    ; (A)       |
L2:                                                 |
   …                                    |
   BRA            L3                                |
L4:                                                 |
   …                                    |
                                        ; (X)-------+
   MOV.W          E0,@_X
L3:
   …
        Bcc       L4
L1:
        …
        Bcc       L2                    ; (B)
```

This optimization, or code motion, makes the loop execution faster. But the bug may prevent the code motion from working even though the conditions for the code motion to work is satisfied.

[Remark]

The occurrence of the bug depends on the contents of the host computer’s memory area freed by the compiler. Even with the same source program and the same options are specified, the bug may or may not occur depending on the environment (or host operating system) of the compiler.

2.4. Correction of incorrect branch code

Fixed is the following problem. A conditional branch after a comparison may be incorrectly taken or incorrectly not taken if one operand of the comparison has caused an overflow in its operation (Case #1). Furthermore, the same symptom occurs if that overflow is the result of an optimization to perform an operation in the smaller size (e.g. int type operation is performed in char type operation) triggered by a type-conversion (Case #2).

[The example of the bug – Case #1]

[The source program]

```
 int i1=1;
 int i2=–32767;
if ( (i1 – i2) < 0)
      printf("OK\n");
else
      printf("NG\n");
```

The result of (i1 – i2) should be –32768 according to the implementation-defined behavior of the compiler and “OK” should be printed. But the bug prints “NG”.

[The incorrect code]

```
MOV.W    @_i1,R0
MOV.W    @_i2,R1
SUB.W    R1,R0
```

```
        BGE      Ln
```
[The correct code]
```
        MOV.W   @_i1,R0
        MOV.W   @_i2,R1
        SUB.W   R1,R0
        MOV.W   R0,R0              ; This instruction is mistakenly deleted in the incorrect code above
        BGE     Ln
```
The V (overflow) flag of the CCR affects the BGE instruction. The BGE instruction above is used assuming that MOV.W R0,R0 above always clears the V flag. But SUB.W R1,R0 above can change the V flag. Deleting MOV.W R0,R0 does not guarantee that the V flag is always cleared when the BGE instruction is executed.

[The example of the bug – Case #2]

[The source program]
```
        int c1=1;
        int c2=–127;
        if ( (char) (c1 – c2) < 0)
              printf("OK\n");
        else
              printf("NG\n");
```
The result of (c1 – c2) should be –128 and “OK” should be printed. But the bug prints “NG”. According to the ANSI standard, (c1 – c2) should be an int-type operation and then no overflow occurs. But the optimization to perform the operation in the smaller size (i.e., char) can cause an overflow.

3. Optimizing linkage editor (Ver.7.1.06 -> Ver.7.1.07)

3.1 Internal error fixed

Fixed is the following internal error:

Internal error(8705) that occurs when optimization of a branch instruction has been specified against an assembler object file

3.2 Illegal operation with form={binary | stype | hexadecimal} option specified

Fixed is the following problem. An error message does not appear even though no output file is created in generating the binary/stype/hexadecimal format file if the directory specified in the output option does not exist.

3.3 Incorrect object code by optimization of deleting unused symbols

Fixed is the following problem. Elements in the two arrays may be incorrect due to the optimization when all of the following conditions satisfied:

(1) An array(say A_arr[]) to be accessed exists.
(2) An array(say B_arr[]) or variable to be deleted by optimization exists.
(3) A_arr and B_arr are in different sections. The sections for the two arrays are assumed to be A and B, respectively.
(4) The start option allocates the sections A and B in a way that the addresses of these sections overlap .
(5) The optimization of deleting unused symbols is enabled.

3.4 Incorrect error at linkage of C++ object codes

Fixed is the problem that an error P3300 (F) may occur incorrectly at linkage of C++ object codes created using templates.

3.5 Illegal lack of some data by conversion of object formats into sysrof

Fixed is the problem that some data incorretly disapper when all of the following conditions are satisfied:

(1) The source program is written in C++.
(2) The optimization of deleting unused symbols is enabled.
(3) The object file of the absolute file format is coverted as ELF->sysrof by the converter(helfcnv).

The end.