Date: Jun.14.2004

# RENESAS TECHNICAL UPDATE

Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan
RenesasTechnology Corp.

| Product Category | User Development Environment | | Document No. | TN-CSX-071A/EA | Rev. | 1.0 |
|---|---|---|---|---|---|---|
| Title | SuperH RISC engine C/C++ Compiler ver.7<br>Known Bugs Report(11) | | Information Category | Usage Limitation | | |
| Applicable Product | P0700CAS7-MWR<br>P0700CAS7-SLR<br>P0700CAS7-H7R | Lot No.<br>Ver.7.x | Reference Document | SuperH RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor User's Manual<br>REJ10B0047-0100H<br>Rev.1.00 | | |

Attached is the description of the known bugs in Ver. 7 series of the SuperH RISC engine C/C++ compiler.

The bugs will affect this package version.

| | Package Version | Compiler Version |
|---|---|---|
| P0700CAS7-MWR | 7.0B | 7.0B |
| | 7.0.01 | 7.0.03 |
| | 7.0.02 | 7.0.04 |
| | 7.0.03 | 7.0.06 |
| | 7.1.00 | 7.1.00 |
| | 7.1.01<br>7.1.02 | 7.1.01 |
| | 7.1.03 | 7.1.02 |
| | 7.1.04 | 7.1.03 |
| P0700CAS7-SLR | 7.0B | 7.0B |
| | 7.0.02 | 7.0.03 |
| | 7.0.03 | 7.0.04 |
| | 7.0.04 | 7.0.06 |
| | 7.1.00 | 7.1.00 |
| | 7.1.01<br>7.1.02 | 7.1.01 |
| | 7.1.03 | 7.1.02 |
| | 7.1.04 | 7.1.03 |
| P0700CAS7-H7R | 7.0B | 7.0B |
| | 7.0.02 | 7.0.03 |
| | 7.0.03 | 7.0.04 |
| | 7.0.04 | 7.0.06 |
| | 7.1.00 | 7.1.00 |
| | 7.1.01<br>7.1.02 | 7.1.01 |
| | 7.1.03 | 7.1.02 |
| | 7.1.04 | 7.1.03 |

The check tool can be downloaded from the following URL.

http://www.renesas.com/eng/products/mpumcu/tool/index.html

Attached: P0700CAS7-040610E

SuperH RISC engine C/C++ Compiler Ver. 7 Known Bugs Report (11)

# SuperH RISC engine C/C++ Compiler ver.7
# Known Bugs Report(11)

Problems with the ver. 7 series of the SuperH RISC engine C/C++ compiler are listed below.
The check tool can be downloaded from the following URL:
http://www.renesas.com/eng/products/mpumcu/tool/index.html

1. Illegal Copy Propagation

[Description]
When a copy instruction existed in a block with multiple branch sources, the copy instruction might be illegally eliminated.

[Example]

```
     int func(int *x) {
        int ret=0;
        while(*x++){
           if(*x==1){
              ret+=2;
           }
        }
        return (ret+2);
     }

     _func:
          MOV           #0,R5     ; Illegally eliminated the copy instruction and converted R7 to R5
     L11:
          MOV.L         @R4,R2
          ADD           #4,R4
                                  ; *1 Illegally eliminated MOV R7,R5
          TST           R2,R2
          ADD           #2,R5
          BT            L13
          MOV.L         @R4,R0
          CMP/EQ        #1,R0
          BT            L11       ; *2 By *3, BF L11 was converted
          BRA           L11
          NOP                     ; *3 Illegally eliminated MOV R5,R7
     L13:
          RTS
          MOV           R5,R0
```

[Conditions]
This problem might occur when all of the following conditions were fulfilled.

(1) The optimize=1 option was specified.
(2) A conditional statement was described.
(3) A copy instruction existed in a block with multiple branch sources (*1 in the above example).
(4) The block of the branch sources in (3) had a path with no definition of the copy source register (R7 in the above example) for the copy instruction (in the example, the path branching from *2 to L11).

[Solution]
If a relevant failure exists, prevent the problem by the following method.

(1) Specify optimize=0.

2. Illegal Elimination of Unnecessary Expressions

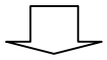
[Description]

If a then or else clause of a conditional statement had an assignment expression and another assignment expression, of which the both sides had the same variable, follows the said expression, the conditional statement might be illegally eliminated.

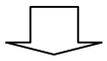
[Example]

```
int x;

void f(int y){
   if (y>=256){        /* Illegal elimination */
      x=0;             /* *1                  */
   }
   x=x;                /* *2  Eliminated the assignment expression that had the same variable in both sides */
   x++;
}
```



```
void f(int y){
   x=0;
   x++;                /* Propagated x=0 */
}
```



```
void f(int y){
   x=1;
}
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) The optimize=1 option was specified.
(2) A conditional statement was described.
(3) A then or else clause of the conditional statement of (2) had an assignment expression (*1 in the above example).
(4) An assignment expression, in which the both sides had the same variable as the variable assigned to in (3), followed the conditional statement of (2) (*2 in the above example).

[Solution]

If a relevant failure exists, prevent the problem by one of the following methods.

(1) Specify optimize=0.
(2) Specify opt_range=noblock.

3. Incorrect GBR Relative Logic Operation

[Description]

If a logic operation with a 1-byte array or a bit-field member for which #pragma gbr_base/gbr_base1 was specified was performed, the result of the operation might be written to an incorrect area.

[Example]

```
#pragma gbr_base a,b
char a[2],b[2];
void f() {
   a[0] = b[0] & 1;
}
```

```
   MOV         #_b-(STARTOF $G0),R0
   RTS
   AND.B       #1,@(R0,GBR)            ; Wrote the result of the operation to b[0]
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) The gbr=user option was specified.

(2) #pragma gbr_base/gbr_base1 was specified for any of the following variables:

- An (unsigned) char-type array
- A structure array that has an (unsigned) char-type member
- A structure that has an (unsigned) char-type array member
- A structure that has a bit-field member of 8 bits or less

(3) A logic operation of a constant (&, |, ^) with the variable of (2) (b[0] in the above example) was performed.

(4) The variable assigned to by the operation of (3) (a[0] in the above example) fulfilled the condition of (2).

(5) Variables of (3) and (4) were different variables, different elements of the same array, or different members of the same structure.

[Solution]

If a relevant failure exists, prevent the problem by one of the following methods.

(1) Cancel specification of #pragma gbr_base/gbr_base1.

(2) Specify gbr=auto (outputs a warning and invalidates #pragma gbr_base/gbr_base1).

(3) Assign the result of the operation to a temporary variable for which volatile has been specified.

Example:

```
void f() {
   volatile char temp;
   temp = b[0] & 1;
   a[0] = temp;
}
```

4. Illegal Elimination of Sign Extension

[Description]

If the address of a variable/constant or the index of an array was cast to 1 or 2 bytes and this value was used for accessing memory, an incorrect memory area might be accessed by eliminating the cast.

[Example]

```
   unsigned short x;
   char a[1000];

   void f() {
      a[(char)x] = 0;
   }
```

```
      MOV.L       L11+2,R2       ; _x
      MOV.L       L11+6,R6       ; _a
      MOV.W       @R2,R5
      EXTU.B      R5,R0
                                 ; Eliminated EXTS.B R0,R0
      MOV         #0,R5          ; H'00000000
      RTS
      MOV.B       R5,@(R0,R6)    ; When x was not within the range of 0 to 127,
                                 ; an incorrect address might be referred to.
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) The optimize=1 option was specified.

(2) The address of a variable/constant or the index of an array was explicitly cast to 1 or 2 bytes, or this function had a char/short type parameter and the parameter was used only in the index of an array..

(3) The value of (2) was used for accessing memory.

[Solution]

If a relevant failure exists, prevent the problem by the following method.

(1) Specify optimize=0.