To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

HITACHI SEMICONDUCTOR TECHNICAL UPDATE

Classification of Production	Development Environment			No	TN-CSX-035A/E	
THEME	SuperH RISC engine C/C++ compiler package Ver.7.0.02(PC version)/ Ver.7.0.03(UNIX version) 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	SH-1,SH-2,SH-2E, SH2-DSP,SH-3, SH3-DSP,SH-4	Lot No. All	Reference Documents	—	Rev. 1	Effective Date Eternity

SuperH RISC engine C/C++ compiler package is updated in Ver.7.0.02(PC version)/Ver.7.0.03(UNIX version).

Refer to the attached document, P0700CAS7-020312E, for details.

A user who has the following product should be notified.

SuperH RISC engine C/C++ compiler Package Ver.7.0B, Ver.7.0.01(PC version), or Ver.7.0.02(UNIX version)

Attached: P0700CAS7-030312E

SuperH RISC engine C/C++ compiler package
Ver. 7.0.02 (PC version)/Ver. 7.0.03 (UNIX version)
Updates

SuperH RISC engine C/C++ compiler Package
Ver. 7.0.02 (PC version)/Ver. 7.0.03 (UNIX version)
Updates

The contents of updates in this package are shown below.

The item 1 holds true only for PC version.

1. Hitachi Embedded Workshop

1.1 Illegal conversion of the HIM project

When the HIM project that does not include the C compiler phase is converted for use in HEW2.0, a problem was modified that the default option cannot be correctly set on the CPU tab in the C compiler.

1.2 Illegal setting of the Denormalize assembler option at build

When the workspace has both projects that assembler option Denormalize"=ON" and "=OFF", a problem was modified that Denormalize"=ON" cannot be set at build (assembling) of the Denormalize"=ON" project.

1.3 Modification of the generation data for project generator

The following CPU's I/O definition file (iodefine.h) was modified:

SH7708: st_sci definition

SH7709A: st_scif, st_irda definition

1.4 Illegal display of the stack size of Runtime Library

A problem was modified that the stack size of Runtime Library cannot be correctly displayed with the stack analysis tool.

2. Compiler (Ver.7.0.03 -> Ver.7.0.04)

2.1 Illegal elimination of an if statement which has side effect

The problem in which a conditional expression of if statement with side effect may be eliminatedis fixed.

[Example]

```
extern int sub();
main() {
    int i=0;
    if(sub()) { /* The call of the function sub() is deleted.          */
        i=10;   /* Because the local variable i is not used after      */
    }           /* this point, the optimizer deletes this statement.   */
}
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) Both of the then and else clauses of the if statement is empty.
(including the case when another optimization makes them empty)

2.2 Illegal stack access

The problem in which load/store against the stack may be invalid is fixed.

[Example]

```
        :
MOV         R15,R2
ADD         #72,R2
MOV         #104,R0    ; H'00000068
MOV.L       R2,@(R0,R15)      <- store data to (SP+72)
MOV         R15,R3
ADD         #92,R3            <- should be #72
MOV.L       @R3,R6            <- load data from (SP+92) incorrectly
MOV.L       L37+60,R4  ; L52
BSR         _func
MOV         #52,R5      ; H'00000034
      :
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) The stack is accessed.

(2) The optimizer divides the function into multiple optimization ranges.
(The condition of division of optimization ranges depends on the size, number of variables, number of function calls in the function, and so on)

2.3 Illegal save/restore in #pragma interrupt function

The following problem is fixed.

When a function specified with #pragma inline_asm is called from a function specified with #pragma interrupt, the registers used in the inline_asm function may be destroyed.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) #pragma inline_asm function is called from a #pragma interrupt function.

(2) There is no other function call in the #pragma interrupt function.

2.4 Illegal save/register MAC registers

The following problem is fixed.

Though macsave=1 is specified, the MACH/MACL registers are not saved in the #pragma regsave function.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) macsave=1 is specified.

(2) #pragma regsave is specified.

2.5 Illegal object of expression in a nested loop

The following problem is fixed.

When the speed option or the loop option is specified for a nested loop, the output code may be invalid.

[Example]

```
          :
    for(...)
        for(...)
           x = x + i;    /* The value of x may be invalid. */
          :
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) The speed option or loop option is specified.

(3) There is a nested loop, and inside the loop, the same variable is used both in the left-hand-side and in the right-hand-side of the assignment like x=x+a.

2.6 Illegal elimination of sign/zero extension

The problem in which the optimizer might eliminate an indispensable EXTS/EXTU instruction in the load/store code is fixed.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) The assignment target size (or the load size of the second or later load instruction) < The assignment source size <= 4 bytes.

2.7 Illegal expansion of switch statement

The following problem is fixed.

When the switch statement is expanded into a jump table, a literal is placed in the delay slot of the jump, and generates illegal code.

[Example]

```
            SHLL        R6
            MOVA        L204,R0
            MOV.W       @(R0,R6),R0
            BRAF        R0
            NOP
    L203:
            .RES.W      1
            .DATA.L     _f1
            .DATA.L     _f2
            .DATA.L     _f3
            .DATA.L     _f4
            .DATA.L     _f5
            .DATA.L     _f6
    L204:
            .DATA.W     L143-L203
            .DATA.W     L143-L203
            BRA         L180
            .DATA.W     L143-L203     ; <- allocated in the delay slot
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) A switch statement is specified, and the jump table is generated.

2.8 Illegal constant expression

The problem in which the constant expression value may not be correct is fixed.

[Example]

```
    void f() {
        sl = 0x80000001l;
                :
        if (*scp16==(char)(0x80000001L))/* Should compare with 1,    */
                :                       /* which is converted to    */
                :                       /* to char from 0x80000001, */
                :                       /* but the actual code      */
    }                                   /* compares with 0x80000001.*/
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) In the same function there are constant expressions with the different sizes and partially have same value (the value which matches in the lower bytes).

2.9 Illegal alias access

The problem in which the assignment to the pointer target may not be done correctly is fixed.

[Example]

```
    int *gp;
    void g(int *p) {
        gp = p;
    }
    int f(int *q) {
        static int i, *p = &i;
        g(p);      /* gp is initialized to point to i. */
            :
        i = 1;
        *gp = 2;   /* The compiler cannot recognize that *gp and I  */
             :     /* are the same location.                        */
        return i;  /* The value i=1 is assumed and 1 is returned    */
    }              /* instead of 2.                                 */
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) An address value is used as the initializer of a static or a global variable, or there is an address reference to an array without using the & operator.

2.10 Illegal save/restore of FPSCR in #pragma interrupt function

The problem in which FPSCR is not saved nor restored in the #pragma interrupt function is fixed.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) cpu=sh2e/sh4 is specified.
(2) There is no function call in the #pragma interrupt function.
(3) There is no instruction with an operand FPSCR in the function.
(4) There is an FPU operation(FABS,FSQRT,FNEG,FCNVDS,FCNVSD,FLOAT,FTRC,FADD, FSUB,FMUL,FDIV,FCMP/EQ,FCMP/GT, or FMAC) in the #pragma interrupt function.

2.11 Illegal scheduling of intrinsic function

The problem in which an instruction may be moved across a call of the following intrinsic functions is fixed.
set_cr, get_cr, set_imask, get_imask, set_vbr, get_vbr, trapa, trapa_svc, prefetch

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.
(2) One of the following intrinsic functions is used.
set_cr, get_cr, set_imask, get_imask, set_vbr, get_vbr, trapa, trapa_svc, prefetch

2.12 Illegal elimination of sign/zero extension in a multiplication result

The following problem is fixed.
When a multiplication result is stored into a 1 or a 2 byte area, a sign/zero extension instruction may be illegally eliminated.

[Example]

```
unsigned short eus=32768,eus2=32769;
void func () {
    unsigned short aus;
    aus= (eus--) * (eus2--);/* Zero extension is eliminated   */
    if (aus!=(unsigned short)((1+eus)*(1+eus2))){
            :               /* Zero extension is done to the  */
}                           /* result of (1+eus)*(1+eus2)     */
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.
(2) The result of multiplication is assigned to a 1 or 2 byte area.

2.13 Illegal cast to the pointer to volatile type

The following problem is fixed.

When a cast to the pointer to volatile type is specified, the volatile specification of the expression in a loop may not be effective.

[Example]

```
#define ADDRESS1 0xf0000000
#define ADDRESS2 0xf0000004
#define BIT0     0x0001
void shc_test(void) {
    unsigned short dataW,read_val;
    read_val = *((unsigned short*)ADDRESS1);
    while(1) {
    /* Volatile specification is not effective and the statement is */
    /* moved out of the loop.                                       */
        dataW = *((volatile unsigned short*)ADDRESS2);

        if(dataW & BIT0) {
            /* Volatile specification is not effective and the     */
            /* statement is moved out of the loop.                */
            *((volatile unsigned short*)ADDRESS2) &= ~BIT0;
            break;
        }
    }
}
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) There is a cast to pointer type which points to volatile type.

(3) The indirect reference with the volatile pointer is used in a loop.

2.14 Illegal object of GBR related logical operation

The problem in which logical compound assignment operator to an external variable may generate invalid code is fixed.

[Example]

```
unsigned char guc1=255;
void func001() {
    if (guc1 & 254){ /* The variable guc1 is loaded.            */
        guc1 &= 253; /* Change the value guc1 on the memory     */
    }               /* using AND.B #253,@(R0,GBR).             */
    if (guc1==253) { /* The variable guc1 is not reloaded.      */
        ok(1);
    } else {
        ng(1);
    }
}
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.
(2) gbr=auto is specified or gbr=user is specified and #pragma gbr_base or #pragma gbr_base1 is used.
(3) An external variable is used in a logical compound assignment operator(&=, |=, ^=).
(4) The logical compound assignment operator is translated into op.B #xx,@(R0,GBR) (op:AND/OR/XOR).

2.15 Illegal elimination of sign/zero extension using #pragma global_register

The following problem is fixed.

When a variable whose size is 1 or 2 byte size has #pragma global_register specification, an EXTS/EXTU instruction may be illegally eliminated.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.
(2) An 1 or 2 byte variable is specified as #pragma global_register.

2.16 Illegal object using struct or union with struct or union array member

The following problem is fixed.

When a struct or union has a member of struct/union array, illegal code may be generated.

[Example]

```
    void func() {
        struct tag {
            union {
                short  u11;
                char   u12[2];
            } u1[2];
        } st= {{{0x1234},{0x5678}}};
        char ans1;
        ans1=st.u1[1].u12[1]; /* (A) copy ans1 before set the   */
                              /*      value to st.u1[1].u12[1]  */
        printf("%x\n",ans1);  /* use ans1 copied at (A)        */
        if(ans1 == 0x78)      /* use ans1 copied at (A)        */
                :
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.

(2) A struct or a union has an array member of struct/union.

(3) The area of struct/union is accessed both by the member reference and by other direct reference (the expression which specifies the whole struct or another member of the union).

2.17 Illegal optimization at linkage

The following problem is fixed.

When goptimize is specified and register save/restore code moves to a delay slot, optlnk may eliminate register save/restore code illegally.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified at compilation.

(2) goptimize is specified at compilation.

(3) optimize=register is specified at linkage.

(4) Register save/restore code exists in delay slot.
(for example MOV.L @R15+,Rn/MOV.L Rn,@-R15)

2.18 Suppress movement of intrinsic function nop() to a delay slot

The following problem is fixed.

When intrinsic function nop() is used, NOP instruction is moved to the delay slot and cannot set to the specified position.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.
(2) Intrinsic function nop() is used.

2.19 Illegal cast of signed char/short type constant

The following problem is fixed.

When an explicit cast to a char/short is done at a constant value, illegal code may be generated.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.
(2) A cast to a signed char/short type is done at a constant value.
(3) This value is negative after cast.

2.20 Illegal expression for a parameter with side effect

The following problem is fixed.

When there is a post-increment or post-decrement expression against a variable with volatile declaration in a function parameter, a value of this variable is illegal in a callee function.

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) optimize=1 is specified.
(2) A variable with volatile declaration is specified in a function parameter.
(3) This parameter has side effect(post-increment or post-decrement).

2.21 Illegal object of sign/zero extension of return value by rtnext

The following problem is fixed.

When rtnext is specified, a return value whose size is 1 or 2 bytes is illegal.

[Example]

```
int a;
unsigned char func(){
    return a;  /* Upper 3byte of variable a is returned as undefined */
}
```

[Condition]

The problem may occur when all of the following conditions are satisfied.

(1) rtnext is specified.
(2) A return value has size of 1 or 2 bytes.
(3) A function have no prototype declaration.

2.22 Illegal data of struct with anonymous bitfield member

The following problem is fixed.

When endian=little is specified and there is an anomymous bitfield member in the struct, data allocations may be incorrect.

[Condition]

The problem may occur when all of the following conditions (1)-(4) are satisfied,

(1) endian=little is specified.
(2) There is an anomymous bitfield member whose width is more than 8 bits in a struct.
(3) This member is not allocated at the head of the struct.
(4) This struct has an initial value.

or when all of the following conditions (5)-(8) are satisfied.

(5) endian=little is specified.
(6) There is an array member in a struct, and the next member is a bitfield.
(7) A padding is inserted between the array member and the bitfield member.
(8) This struct has an initial value.

2.23 Suppress elimination of variable reference whose variable is volatile

When the following expression is described at a variable with a volatile declaration, a reference to the variable is guaranteed.

a &= 0;

2.24 Illegal debugging information

The problem in which a debugging information of local array, struct, or union which is allocated on stack is not output is fixed.

2.25 Internal error

The bug in which an internal error occurs is fixed.

3. Optimizing Linkage Editor (Ver.7.1.02 -> Ver.7.1.04)

3.1 Incorrect debug information caused by the compress option

Fixed is the problem in which compressed debug infomation specified with the compress option is incorrect.

3.2 Address check regarding external symbol allocation optimization

Symbol addresses compiled with the map option (optimization of external variable accesses) was checked only when the map option is specified to the linker so far. But the linker is modified so that the checking may be always done regardless of the map option with the linker.

3.3 Incorrect section attribute when a binary file is input

Fixed is the problem in which section attribute is illegal when the following conditions are satisfied at the same time.

[Condition]

(1) Input both an object file and a binary file.
(2) A section whose size is zero is defined in the input object file.
(3) The section whose size is zero is specified in the binary option.
(4) The object file of (2) is input earlier than the binary file of (3).

3.4 Internal error caused by the form=relocate option

Fixed is the problem that occurs when all of the condition (1) to (4) are satisfied.

[Conditions]

(1) The first input object file is compiled with the goptimize option.
The second or later input file is either compiled without the goptimize option or just assembled.
(2) The form=relocate option is specified.
(3) The profile option is specified.
(4) The optimize option is specified.

3.5 Internal error(1703) caused by the symbol_delete and register optimizations

Fixed is the problem that causes an internal error when the following conditions are satisfied at the same time.

[Conditions]

(1) The goptimize option has been specified in an input object file.
(2) There is a BSR or BRA instraction whose displacement is a boundary value (-4096 or 4094) in the object file of (1).
(3) The symbol_delete and register optimizations are specified.

3.6 Incorrect literal referring

Fixed is the problem in which a referred literal value is incorrect when the following conditions are satisfied at the same time.

[Conditions]

(1) The goptimize option has been specified in an input object file.
(2) More functions than one refer to the same literal in the object file.
(3) The optimize=register option is specified.