date: 2002/09/09

HITACHI SEMICONDUCTOR TECHNICAL UPDATE

Classification of Production	Development Environment		No	TN-CSX-043A/E	Rev	1
THEME	SuperH RISC engine C/C++ compiler package Ver.7.1.00 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	P0700CAS7-MWR P0700CAS7-SLR P0700CAS7-H7R	Lot No.: All	Reference Documents	SuperH RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor User's Manual ADE-702-246A Rev.2.0	Effective Date	Eternity

SuperH RISC engine C/C++ compiler package is updated in Ver.7.1.00.

Refer to the attached document, P0700CAS7-020826E, for details.

Inform the customers who have the package version in the table below.

	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
P0700CAS7-SLR	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06
P0700CAS7-H7R	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06

Attached : P0700CAS7-020826E
SuperH RISC engine C/C++ Compiler
Ver.7.1.00 Updates

SuperH RISC engine C/C++ Compiler Package Ver. 7.1.00 Updates

The contents of updates in this package are shown below.
The item 1 and 2 hold true only for PC version. The item 7 and 8 hold true only for UNIX version.

1. Hitachi Embedded Workshop (PC version)

1.1 Loading and Unloading the Project

When two or more projects have been used in the workspace, only the current project is loaded and referred while the workspace is opened. Other projects can be referred when the target project is selected on the Workspace window and [Load Project] is specified from the popup menu by clicking the right-hand mouse button. The project that needs not be referred is temporarily released from the management on HEW when [Unload Project] is similarly specified from the popup menu. (This operation saves the memory.)

1.2 Setting the Navigation Tab in the Workspace Window

When [Configure view...] is selected from the popup menu (by clicking the right-hand mouse button) on the Navigation tab in the Workspace window, the information to be displayed on the Navigation tab can be set. By default, 'ANSI C Functions' and 'C Defines' are displayed.

1.3 Setting Header and Footer in Printing

When [Page Setup...] is selected from the [File] menu, the header and footer in printing can be set.

1.4 Setting the Custom Placeholder

The user-specific placeholder can be set on the Placeholder tab in [Customize] from the [Tools] menu. Set 'Application wide custom placeholders:' for using the placeholder in the whole HEW. Set 'Workspace wide custom placeholders:' for using the placeholder in each workspace.

1.5 Automatic Build Corresponding to MAP Optimization

In HEW2.0, a custom phase has been provided to execute optimization using the external symbol allocation information that has been output by the optimizing linkage editor in the C compiler.

In HEW2.1, the build for optimization can be executed by setting options only. When 'Include map file' is specified as Optimize on the C/C++ tab in the Toolchain Option dialog box, a build is automatically executed again from the C compiler after the optimizing linkage editor has been executed.

1.6 Adding and Modifying the Data Generated by the Project Generator

Project generation of the following CPU has been newly added:

SH7290

The I/O definition file (iodefine.h) of the following CPU has been modified:

SH7727

1.7 To support data merge feature on the Stack Analysis tool

One of the utility tools, Stack Analysis tool supports DataMerge feature. You can see stack data which has already generated and stack information file which is generated optimize linkage editor as merged data on it.

1.8 Modification of the generation data for project generator

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

SH7622: st_scif0 definition

2. SuperH RISC engine simulator/debugger (PC version)

2.1 Supporting Sessions (Ver. 8.0.01 -> Ver. 8.1.00)
The target-specific information is saved in the session. Two or more sessions can be contained in one configuration.

2.2 Enabling Specification of the Execution Order of Command Batch File (Ver. 8.0.01 -> Ver. 8.1.00)
The order for automatically executing the command batch file can be specified. The execution timing can be selected either when the target is connected or before or after loading the user program.

2.3 Enforcing the Source Window (Ver. 8.0.01 -> Ver. 8.1.00)
The corresponding address and coverage information are displayed on the Source window. It is possible to select whether the address and coverage information is displayed or not.
The coverage information can be displayed in the C-source level.

2.4 Enforcing the Coverage Function (Ver. 8.0.01 -> Ver. 8.1.00)
Saving or loading the coverage information is supported. Merging with the previous coverage method is possible.

2.5 Enforcing the Watch Function (Ver. 8.0.01 -> Ver. 8.1.00)
The realtime display of the watch infromation is supported. The information is updated in realtime during program execution.
In addition, saving the watch information in the file is also supported.

2.6 Enforcing the Trace Function (Ver. 8.0.01 -> Ver. 8.1.00)
The statistic analysis function of the trace acquisition information is supported. When the item to be analyzed is specified, it is possible to display the kind of information and the acquisition times.

2.7 Supporting the Trigger Window (Ver. 8.0.01 -> Ver. 8.1.00)
The Trigger window is supported. A pseudo-interrupt can be generated at any timing.

2.8 Supporting the SH3-DSP (core) Simulator (Ver. 8.0.01 -> Ver. 8.1.00)
The SH3-DSP (core) simulator is supported. Debugging the SH3-DSP for ASIC will be enabled.

3. Compiler
3.1 Added options (Ver.7.0.04 -> Ver.7.0.06)
The following shows the options added to Ver.7.0.06.

(1) global_volatile={0 | 1}
Treatment of global variables
(2) opt_range={all | noloop | noblock}
Optimizing range of global variables
(3) del_vacant_loop={0 | 1}
Deletion of vacant loop
(4) max_unroll=<numeric number>
Specification of maximum unroll factor
(5) infinite_loop={0 | 1}
Deletion of assignments before an infinite loop
(6) global_alloc={0 | 1}
Allocation of global variable
(7) struct_alloc={0 | 1}
Allocation of struct/union member
(8) const_var_propagate={0 | 1}
Propagation of const-qualified variable
(9) const_load={inline | literal}
Inline expansion of constant load
(10) schedule={0 | 1}
Scheduling of instructions

3.2 Illegal destruction of the R0 register (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

When a parameter is passed via the stack, the R0 register may be illegally overwritten.

[Example]

```
short func1(short a0, int *a1, int a2, short a3, short a4, short a5, short a6, int a7, int a8, int a9);
void func0(short a0, int *a1, int a2, short a3, short a4, short a5, short a6) {
                              :
    r1=func1(0,a1,0,0,0,0,0,0,0,0);
    if((r1>0)&&(r1!=1)) {
        func1(a0,a1,0,a3,a4,a5,a6,0,0,0);  /* R0 is destroyed illegally. */
    }
                              :
}
```

```
                              :
        MOV.L                 R0,@(32,R15)       ; Stores R0 at @ (32, R15).
        MOV                   R8,R5
        MOV                   #66,R0             ; Destroys R0.
        MOV.W                 @(R0,R15),R3
        MOV                   R9,R6
        MOV                   #70,R0             ; Destroys R0.
        MOV.W                 @(R0,R15),R1
        MOV                   R0,R4              ; @(32,R15) has been illegally replaced with R0.
        MOV.L                 R3,@(4,R15)
        MOV.L                 R1,@(8,R15)
        MOV.L                 R9,@(12,R15)
        MOV.L                 R9,@(16,R15)
        BSR                   _func1
        MOV.L                 R9,@(20,R15)
                              :
```

[Condition]

This problem may occur when both of the following conditions are satisfied

(1) The optimize=1 option is specified.
(2) A function receives a formal parameter passed via the stack.

3.3 Illegal branch target of the BRA instruction (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

In a program having an unconditional branch, the forward branch target of the BRA instruction may be illegal if the distance from the instruction to the target is 4094 bytes.

[Condition]

This problem may occur when both of the following conditions are satisfied

(1) Either the code=machinecode option is specified, or the code option is not specified.
(2) The distance from the BRA instruction to the forward branch target is 4094 bytes.

3.4 Illegal deletion of the save/restoration of PR register (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

When following program is compiled with the speed option, the saving and restoring code of PR register may be illegally deleted.

```
[Example]
int x;
extern void f1();
extern void f2();
void f() {
   if (x == 2){
      f1();                 // The then-clause ends with a function call
   }
   f2();                    // The function end with a function call
   return;
}

_f:
       MOV.L             L14,R6             ; _x
       MOV.L             @R6,R0
       CMP/EQ            #2,R0
       BT                L11
L12:
       MOV.L             L14+4,R2           ; _f2
       JMP               @R2                ; The function ends with a function call,
       NOP                                  ; so JSR changed into JMP and RTS is deleted
L11:
       MOV.L             L14+8,R2           ; _f1
       JSR               @R2                ; Saving and restoring code of PR register
       NOP                                  ; is deleted though a function call exists
       BRA               L12
       NOP
```

[Condition]

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

(1) The speed option is specified.
(2) The function ends with a function call.
(3) The if-then clause exists before the function call of (2), and the last statement of then-clause is function call.
(4) The function call is not in-line expanded.

3.5 Illegal reference of T-bit in SR register (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

In the following case, conditional branch may be illegally done.

(1) When following program is compiled:

```
    [Example]
#include <machine.h>
extern void f();
int a;
void func() {
   int b;
   b = (a == 0);            // Sets the result of comparison to variable b
   f();                     // Exists function call or set_cr()
   if (b) {                 // Compares variable b with 0 or 1
      a=1;
   }
}
```

```
_func:
        STS.L               PR,@-R15
        MOV.L               L13,R6          ; _a
        MOV.L    @R6,R2
        TST                 R2,R2           ; Sets the result of comparison to T-bit
        MOV.L               L13+4,R2        ; _f
        JSR                 @R2             ; T-bit may be changed in the callee
        NOP                                 ; function
        BF                  L12             ; Refers to T-bit and branch
        MOV.L               L13,R6          ; _a
        MOV                 #1,R2
        MOV.L               R2,@R6
L12:
        LDS.L               @R15+,PR
        RTS
        NOP
```

(2) When the class with destructor call which is declared in a local block in a function is written in the C++ program

[Condition]

This problem may occur when either (1) to (3) or (4) to (5) conditions are satisfied.

<for C program>

(1) The optimize=1 option is specified.
(2) A result of a comparison is set to a variable.
(3) The variable of (2) is compared with 0 or 1 after a function call or set_cr().

or

<for C++ program>

(4) The optimize=1 option is specified.
(5) The class with destructor call which is declared in a local block in a function is written in the C++ program

3.6 Illegal unification of constant values (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

When following program is compiled with the optimize=1 option, the constant values may be illegally unified.

[Example]

```
#define a (*(volatile unsigned short *)0x400)
#define b (*(volatile unsigned short *)0x4000)
#define c (*(volatile unsigned short *)0x402)
int d;
void func() {
    a = 0x8000;              /* (A)    */
    b = 0x8000;              /* (A')   */
    d = c + 0x8000;          /* (B)    */
}
```

```
_func:
        MOV.W          L15,R6          ; H'8000  set R6 to 0xFFFF8000
        MOV            #4,R5
        MOV            #64,R2
        SHLL8          R5
        SHLL8          R2
        MOV.W          R6,@R5          ; (A) set variable a to 0x8000
        MOV.W          R6,@R2          ; (A') set variable b to 0x8000
        MOV.W          @(2,R5),R0
        EXTU.W         R0,R2
        ADD            R6,R2           ; set R2 to (c+0xFFFF8000)
        MOV            R2,R6
        MOV.L          L15+4,R2        ; _d
        RTS
        MOV.L          R6,@R2          ; (B)  set variable d to (c+0xFFFF8000)
                                       ;      (c+0x00008000) is correct
```

[Condition]

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

(1) The optimize=1 option is specified.
(2) The same value from 128 to 255 or from 32768 to 65535 is used more than once in the function.
(3) The value of (2) is used with different sizes.
For above example, (A) and (A') are used as a 2-byte value and (B) is used as a 4-byte value.

3.7 Illegal generation of a literal pool (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

When a program is compiled with the align16 option, the reference to a literal pool may be illegal.
When both code=machinecode and goptimize are specified, the error may occur at linkage.
When code=machinecode is specified, an illegal object code may be created at compilation.
When code=asmcode is specified, the error may occur in assembling.

[Example]

```
              :
        MOV.L          L154+2,R2     ; (1) L158      refers to literal (A)
        MOV.L          R2,@R15
        MOV.L          L154+6,R2     ; (2) _printf   refers to literal (B)
        JSR            @R2
        NOP
              :
        .ALIGN         16
L86:
        ADD            #1,R2
        BRA            L153          ; unconditional branch is created and
        MOV.L          R2,@R4        ; a literal pool is generated
L154:
        .RES.W         1
        .DATA.L        L158          ; (A) literal which cannot be reached
                                     ;    from (1)
        .DATA.L        _printf       ; (B) literal which cannot be reached
                                     ;    from (2)
        .ALIGN         16
L153:
              :
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The align16 option is specified.
(2) An unconditional branch is created and a literal pool is generated.

3.8 Illegal code motion to a delay slot (Ver.7.0.04 -> Ver.7.0.06)
The following problem is fixed.
When a program is compiled with the optimize=1 option, an instruction may be illegally moved to a delay slot.
[Example]
<before>

```
              :
    SHLL      R2
    MOV       R2,R0
    MOVA      L88,R0
    BRA       L144
    NOP
              :
```

<after>

```
              :
    SHLL      R2
                        ; instruction is moved to a delay slot
    MOVA      L88,R0    ; set R0
    BRA       L144
    MOV       R2,R0     ; destroy R0
              :
```

[Condition]
This problem may occur when the following condition is satisfied.
(1) The same register is updated consecutively by more than one instruction.

3.9 Illegal offset of a GBR-relative logical operation (Ver.7.0.04 -> Ver.7.0.06)
The following problem is fixed.
When a program is compiled with the optimize=1 option and the compiler creates a GBR-relative logical operation to a 1-byte struct member, the offset may be illegal.
[Example]

```
struct {
   int a;
   unsigned char b;
} ST;
char c;
void f() {
   ST.b |= 1;
   c &= 1;
}

_f:
    STC       GBR,@-R15
    MOV       #0,R0          ; H'00000000
    LDC       R0,GBR
    MOV.L     L11+2,R0       ; H'00000008+_ST  <- (ST+4) is correct
    OR.B      #1,@(R0,GBR)
    MOV.L     L11+6,R0       ; _c
    AND.B     #1,@(R0,GBR)
    RTS
    LDC       @R15+,GBR
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) Either the gbr=user option is specified and #pragma gbr_base/gbr_base1 is used, or the gbr=auto option is specified and the map option is not specified.
(3) A global struct with a 1-byte member exists in a program.
(4) This 1-byte member is not located at the top of the struct.
(5) This member is used for logical operation in a function.
(6) This member is not used except (5).
(7) A global variable except this member exists in a function.

3.10 Illegal EXTU after SWAP instruction (Ver.7.0.04 -> Ver.7.0.06)
The following problem is fixed.
When a program is compiled with the optimize=1 option and a pointer is used to store the return value of swapb, swapw, or end_cnvl intrinsic function, EXTU may be illegally created after the SWAP instruction.
[Example]

```
#include <machine.h>
unsigned short *a,*b;
void func() {
  *b=swapb(*a);
}

_func:
    MOV.L       L13+2,R2    ; _a
    MOV.L       L13+6,R5    ; _b
    MOV.L       @R2,R6
    MOV.W       @R6,R2
    SWAP.B      R2,R6
    MOV.L       @R5,R2
    EXTU.B      R6,R6       ; The result of SWAP instruction is illegally expanded
    RTS
    MOV.W       R6,@R2
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) A swapb, swapw, or end_cnvl intrinsic function is used.
(3) A pointer is used to store the return value of this intrinsic function.

3.11 Illegal bitfield data (Ver.7.0.04 -> Ver.7.0.06)
The following problem is fixed.
When an initial value is set to a struct with an anonymous bitfield, the initial value may be illegal.
[Example]

```
struct st {
  short a:4;
  short b;
  short :12;          // anonymous bitfield
  short c:4;
} ST={1,1,3};

_ST:
     .DATA.W       H'1000
     .DATA.W       H'0001
     .DATAB.B      1,0          ; ".DATA.W H'0003"
     .DATA.W       H'0300       ; is correct.
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) (1) A struct has a bitfield, an anonymous bitfield, and a member which is not a bitfield, and they are defined in the order shown below.

```
struct A {
            :
      bitfield
            :
      member which is not bitfield
      anonymous bitfield        // (A)
      bitfield                  // (B)
            :
}
```

(2) The size of the underlying type of (A) and (B) is 2-byte or 4-byte.
(3) The bitwidth of (B) is 8-bit or more.
(4) The total bitwidth size of (A) and (B) is below.
   (a) When the sizes of the underlying type of (A) and (B) is 2-byte : 16-bit or less
   (b) When the sizes of the underlying type of (A) and (B) is 4-byte : 32-bit or less
(5) The struct declared with an initial value.

3.12 Illegal loop expansion (Ver.7.0.04 -> Ver.7.0.06)
The following problem is fixed.
When the speed option or the loop option is specified, a loop conditional expression may be illegally replaced.
[Example]

```
int a[100];
void main(int n) {
  int i;
  for (i=0; i<n; i++) {
    a[i] = 0;
  }
}

_main:
      MOV         R4,R7
      ADD         #-1,R4      ; When a value of R4 is 0x80000000, underflow occurs
                              ; and R4 will have 0x7FFFFFFF.
      MOV         R4,R6
      CMP/PL      R4          ; The result of the comparison is incorrect.
      MOV         #0,R4
      BF          L12
      ADD         #-1,R6
          :
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The speed option or the loop option is specified.
(2) A loop statement is used in a function.
(3) The loop upper bound is in the range shown below.
   (a) When the loop control variable, say i, is incremented like i += step, the value is in the range from 0x80000000 to 0x80000000+step-1
   (b) When the loop control variable, say i, is decremented like i -= step, the value is in the range from 0x7FFFFFFF to 0x7FFFFFFF-step+1

When loop upper bound is in a variable and its value is in the range above, the behavior may be incorrect.

3.13 Illegal reference to a struct or an array parameter (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

When a struct, a union or an array is used in a parameter of a function and this parameter is referred to in the function, the address to refer to this parameter may be illegal.

[Example]

```
typedef  struct{
   int A[10];
   double B;
   char F[20];
} ST;
extern ST f(ST a,ST b);
ST S;
extern int X;
void func(ST a,ST b) {
   ST t;
   if (a.B!=f(S,t).B){
      X++;
   }
   if (a.B!=b.B){
      X++;
   }
}

                              :
L12:
       MOV          R15,R2
       MOV.W        L15+2,R0       ; H'014C
       ADD          R0,R2
       MOV          R15,R0
       MOV.W        L15+4,R0       ; H'0190  R0 is destroyed illegally.
       ADD          R0,R0
       MOV.L        @R2,R4
       MOV.L        @(4,R2),R7
       MOV          R0,R2
       MOV.L        @R2,R6         ; An address which b.B is not located is accessed.
                              :
```

[Condition]

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

(1) A function has a struct, a union or an array parameter.
(2) This parameter is referred to in the function.

3.14 Illegal deletion of a JMP instruction (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

When the speed option is specified, a JMP instruction may be deleted.

[Example]

```
void f(){
   int i,j=0;
   for (i=0; i<10; i++)
      if (j%2) j++;
   sub();
}

void sub() {
      :
}
```

```
_f:
                          :
L20:
        ADD             #-1,R5
        TST             R5,R5
        BF              L11
        MOV.L           L23,R2          ; _sub   These instructions are deleted.
        JMP             @R2             ;                        ↓
        NOP                             ;
L18:
        MOV             R6,R0
        AND             #1,R0
        BRA             L16
        MOV             R0,R2
L13:
        MOV             R6,R0
        AND             #1,R0
        BRA             L14
        MOV             R0,R2
_sub:
                          :
```

[Condition]

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

(1) The speed option is specified.
(2) The function ends with a function call.
(3) This function call is not in-line expanded.
(4) The definition of the callee function follows that of the caller function.
(5) A loop statement or a conditional statement is used in the caller function.

3.15 Illegal loop expansion (Ver.7.0.04 -> Ver.7.0.06)

The following problem is fixed.

When the following program is compiled with the optimize=1 option, a loop conditional expression may be illegally replaced.

[Example]

```
void f1() {
    int i;
    for (i=-1; i<INT_MAX; i++) {
        a[i] = 0;
    }
}

_f1:
        MOV.L           L13,R2          ; _a
        MOV             #-4,R6          ; H'FFFFFFFC
        MOV             R6,R5
        MOV             #0,R4           ; H'00000000
        ADD             #-4,R2
L11:
        ADD             #4,R6
        MOV.L           R4,@R2
        CMP/GE          R5,R6           ; compare with H'FFFFFFFC
        ADD             #4,R2
        BF              L11
        RTS
        NOP
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) A loop statement is used in the function.
(3) The result of (loop upper bound - loop lower bound) overflows.
For example : for(i=-1; i<0x7FFFFFFF; i++)

3.16 Illegal stack access (Ver.7.0.06 -> Ver.7.1.00)

The following problem is fixed.
When an address of a parameter passed on a register is referred to, a data on the stack may be superseded illegally.

[Example]
```
extern void er();
extern char f2(char *a);
void f(char a) {
  char c;
  a++;                        // updates the variable which is a parameter on a register
  do {
    c=f2(&a);                 // refers to an address of a parameter on a register
    if (c) er();
    else return;
  } while(c);
}

_f:
      MOV.L          R13,@-R15
      MOV.L          R14,@-R15
      STS.L          PR,@-R15
      ADD            #-4,R15
      MOV            R4,R0
      ADD            #1,R0
      MOV.B          R0,@(3,R15)        ; updates the variable "a"
      MOV.L          R4,@R15            ; the variable "a" is superseded illegally
      MOV.L          L14+2,R13          ; _f2
                 :
```

[Condition]
The problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) A parameter on a register exists.
(3) An address of this parameter is referred to in a function.
(4) An assignment to this parameter exists before reference to the address.
(5) Copying a parameter on a register into the stack is moved after assignment to the parameter to the parameter by the optimization of instruction scheduling.

3.17 Deletion of an assignment to the register (Ver.7.0.06 -> Ver.7.1.00)

The following problem is fixed.

An assignment to the register for which #pragma global_register has been specified may be illegally deleted.

[Condition]

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

(1) #pragma global_register is specified.
(2) The optimize=1 option is specified.
(3) The variable specified in (1) is defined or used in a single expression such as a compound assignment expression.

```
<Example>
#pragma global_register(a=R14)
:
a += b;    // or a=a+b;
```

3.18 Illegal comparison of a 32-bit bitfield (Ver.7.0.06 -> Ver.7.1.00)

The following problem is fixed.

Before comparing a 32-bit bitfield to 0, an AND operation with 0 may be illegally generated. The result will always be true (or false).

[Example]

```
struct ST {
   unsigned int b: 32;
};

void f(struct ST *x) {
   if (x->b) {
         :
   }
}
              :
   MOV.L      @R4,R0
   AND        #0,R0         ; ANDed with 0.
   TST        R0,R0         ; Always true.
   BF         L12           ; A branch to L12 will not occur.
              :
```

[Condition]

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

(1) There is a 32-bit field member in a structure.
(2) The relevant member is compared to 0 (== or !=).

3.19 Illegal instruction to move stacks while the trapa_svc function is in use (Ver.7.0.06 -> Ver.7.1.00)

The following problem is fixed.

If pic=1 is specified for compilation, an illegal instruction to move stacks may be generated when loading the address of the function that uses the intrinsic function trapa_svc.

[Example]

```
#include <machine.h>
extern char *b(void (*yyy)(char));

void y(char c) {
   trapa_svc(160, 10, c);
}

char *a(void) {
   return b(y);
}
```

```
_a:
    MOV.L           L14,R4          ; _y-L12
    MOVA            L12,R0
    ADD             R0,R4
L12:
    MOV.L           @R15+,R0        ; <- Unnecessary instruction to move stacks
    MOV.L           L14+4,R2        ; b-L13
    MOVA            L13,R0
    ADD             R0,R2
L13:
    JMP             @R2
    MOV.L           @R15+,R0        ; <- Unnecessary instruction to move stacks
```

[Condition]

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

(1) An option other than cpu=sh1 is specified.
(2) The pic=1 option is specified.
(3) The intrinsic function trapa_svc is in use.
(4) The location of loading the address that uses the trapa_svc function comes later than the location where the trapa_svc function is called.

3.20 Illegal movement of a copy instruction for R0-R7 (Ver.7.0.06 -> Ver.7.1.00)

The following problem is fixed.

Due to an optimization, a copy instruction or an extended instruction for R0-R7 may be illegally moved beyond the range of calling functions. The result of CMP/EQ (TST) may be incorrect.

[Example]

```
                :
MOV.L           @R4,R2
MOV             R5,R14
MOV             #1,R5           ; H'00000001
ADD             #24,R2
MOV.L           @R2,R6
EXTU.W          R14,R1          ; The definition of R1 moves beyond JSR
MOV.L           @(8,R2),R7
JSR             @R7             ; R1 may be damaged at the callee
ADD             R6,R4
MOV             #4,R2           ; H'00000004
CMP/EQ          R2,R1           ; Compares by using the value of damaged R1
EXTU.W          R0,R0
BF              L16
                :
```

[Condition]
The problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) The program includes conditional branches and calling of functions.
(3) An optimization has been performed as follows;
<Before an optimization>

```
    MOV     R0, Rn          ; or EXTU  R0,Rn
                  :
    MOV     Rx, R0          ; or EXTU  Rx,R0
    TST     #imm, R0        ; or CMP/EQ  #imm,R0
    MOV     Rn, R0          ; or EXTU  Rn,R0
```

<After an optimization>

```
    MOV     Rx, Rm          ; or EXTU  Rx,Rm
    MOV     #imm, Ry
    TST     Ry, Rm          ; or CMP/EQ  Ry,Rm
```

(4) In the optimization mentioned in (3), the Rm register is R0-R7. No other instruction in this function uses this register.
(5) Due to another optimization, the MOV Rx,Rm (or EXTU) function, which has been generated by the optimization mentioned in (3), is moved beyond the range of calling functions.

3.21 Countermeasure against the chip defect regarding SH-2E FDIV (Ver.7.0.06 -> Ver.7.1.00)
The chip defect regarding the FDIV instruction of SH7055RF FPU has been taken into account. Specify both cpu=sh2e and patch=7055 options to avoid the defect.

4. Standard library Generator
4.1 realloc space (Ver.2.0 -> Ver.2.0(01))
The following problem is fixed.
After the call of realloc(), malloc() may fail to allocate space even though there should be enough space remaining.

5. Format converter
5.1 Unrecognized input files (Ver.1.0C -> Ver.1.0.04)
Fixed is the problem in which the format converter can not recognize input files when input files exist in the folder with compress attribute.

6. Optimizing Linkage Editor
6.1 Internal error(7041) with the output option specified (Ver.7.1.04 -> Ver7.1.05)
Fixed is the problem in which an internal error(7041) occurs when the following conditions are satisfied at the same time.
[Conditions]
(1) The input file is compiled with the goptimize option.
(2) The output option is specified with range specification.
(3) The nooptimize option is not specified.

6.2 Incorrect optimization with partial suppression of the optimization
Fixed is the problem in which the partial suppression of the optimization does not work. The problem occurs when the following conditions are satisfied at the same time.
[Condition]
(1) The input file is compiled with the goptimize option.
(2) The nooptimize option is not specified.
(3) The absolute_forbid option is specified.
(4) More than one address ranges are specified to the absolute_forbid option.
(All but the first becomes ineffective)

6.3 Incorrect object code in generating a relocatable file (Ver.7.1.04 -> Ver7.1.05)

Fixed is the problem in which an incorrect object code is generated when the following conditions are satisfied at the same time.

[Condition]

(1) The input file is relocatable file.
(2) The form=relocate option is specified.
(3) The delete or rename option is specified.

6.4 Incorrect error(2410) with optimization of external variable accesses (Ver.7.1.04 -> Ver7.1.05)

Fixed is the problem in which an incorrect error occurs when the following conditions are satisfied at the same time.

[Condition]

(1) The input file is compiled with the map option.
(2) Sections are overlaid in the start option.

6.5 Internal error(1703) regarding the register save/restore optimization (Ver.7.1.04 -> Ver7.1.05)

Fixed is the problem in which an internal error(1703) may occur when the optimization regarding register save/restore is specified.

6.6 Internal error when optimization is specified (Ver.7.1.05 -> Ver.7.1.06)

Fixed the problem in which an internal error (1703 or 1704) occurs when optimization is specified.

6.7 Internal error when the output option is specified (Ver.7.1.05 -> Ver.7.1.06)

Fixed the problem in which an internal error (7707) occurs when the output range is specified with an address in the output option.

6.8 Internal error with the HEX, BIN, or Stype output (Ver.7.1.05 -> Ver.7.1.06)

Fixed the problem in which an internal error (3304) occurs when a source program is written in C++ and the output format is HEX, BIN, or Stype.

6.9 Internal error when the map option is specified (Ver.7.1.05 -> Ver.7.1.06)

Fixed problem in which an internal error (8093) occurs when the map option is specified.

6.10 Illegal object when save/restore registers optimization is specified (Ver.7.1.05 -> Ver.7.1.06)

Fixed the problem in which an illegal object code is output when optimization of codes for saving or restoring registers is specified and either of the following conditions is satisfied:

[Condition]

(1) No literal exists in the final function in the file.
(2) There is a branch destination immediately after the instruction for restoring registers.
(3) The SUBC instruction is included in the function to be optimized.

7. SuperH RISC engine simulator/debugger (UNIX version)

7.1 Supporting the SH3-DSP ASIC core Simulator (Ver.4.11 -> Ver.4.2.00)

This simulator/debugger supports SH3-DSP ASIC core.

The simulator supports co-simulation.

8. Stack Analysis tool

8.1 To support data merge feature on the Stack Analysis tool (Ver.1.2.00 -> Ver.1.3.00)

One of the utility tools, Stack Analysis tool supports DataMerge feature.

You can see stack data which has already generated and stack information file which is generated optimize linkage editor as merged data on it.