P0700CAS7-RM020725E

****

SuperH RISC engine C/C++ Compiler Package CD-ROM

README.TXT
July, 2002
Hitachi ULSI Systems Co., Ltd.

www.hmse.com

****

This "README.TXT" includes installation procedure of SuperH RISC engine C/C++ Compiler Package and additional notes on the user's manual. Please read before using CD-ROM.

Directory Structure

This CD-ROM has the following directory structure.

```
-+--Readme_e.txt
 |  [This file]
 +--Readme_j.txt
 |  [Readme(Japanese)]
 +--Program.tar
 |  [Tar file of the compiler, the assembler and the installer for simulator]
 +--Simulator.tar
 |  [Tar file of the simulator]
 +--Pdf_Read (Windows(R) version)
 |
 +--Tools---+--Japanese
 |          |  [Stack analysis tool (Japanese)]
 |          +--English
 |             [Stack analysis tool (English)]
 +--Manuals-+--Jshu0207.pdf
            |  [Main file of the manual (Japanese)]
            +--Eshu0207.pdf
            |  [Main file of the manual (English)]
            +--Pdfs+--Jshcum19.pdf
                   |  [User's manual of the compiler, the assembler and
                   |   the optimizing linkage editor (Japanese)]
                   +--Jshdum0b.pdf
                   |  [User's manual of the simulator debugger (Japanese)]
                   +--Jshcap27.pdf
                   |  [Supplement for the compiler (Japanese)]
                   +--Jshcrn21.pdf
                   |  [Notes on the compiler (Japanese)]
                   +--Eshcum19.pdf
                   |  [User's manual of the compiler, the assembler and
                   |   the optimizing linkage editor (English)]
                   +--Eshdum0b.pdf
                   |  [User's manual of the simulator debugger(English)]
                   +--Eshcap27.pdf
                   |  [Supplement for the compiler (English)]
                   +--Eshcrn21.pdf
                     [Notes on the compiler (English)]
```

Execution Environment

C/C++ Compiler/Simulator
 - OS: Solaris 2.5 or later version, HP-UX 10.2 or later version
 - Disk Space: about 70MB

Stack Analysis Tool
 - Host Computer: IBM PC Compatible Machine, PC-9800 Series
   (CPU□FThose CPUs on which Windows(R)98, Windows(R)Me, Windows NT(R)4.0,
    Windows(R) 2000 or Windows(R)XP is running)
 - Microsoft(R) Windows(R)98, Windows(R)Me, Windows NT(R)4.0, Windows(R)2000
   or Windows(R)XP.
 - CD-ROM drive with more than double speed
 - Disk Space: about 15.0MB
 * Please refer to "Compiler, Assembler, Optimizing Linkage Editor User's Manual"
   for the usage.

User's Manual:
 - Host Computer: IBM PC Compatible Machine, PC-9800 Series
   (CPU□FThose CPUs on which Windows(R)98, Windows(R)Me, Windows NT(R)4.0,
    Windows(R)2000 or Windows(R)XP is running)
 - Microsoft(R) Windows(R)98, Windows(R)Me, Windows NT(R)4.0, Windows(R)2000
   or Windows(R)XP.
 - CD-ROM drive with more than double speed
 - Disk Space: about 15.0MB
 * User's manuals can be opend on Windows(R)98, Windows(R)Me, Windows NT(R) 4.0,
   Windows(R)2000 or Windows(R)XP.

[Caution]
Never play this CD-ROM on an audio CD player.

  Installation Procedure of the Compiler/Simulator

This section explains the installation procedure of the compiler and the assembler system.
For the installation procedure of the simulator, please refer to "SuperH RISC engine
Simulator/Debugger Install Guide".

1. Mount the CD-ROM (If the CD-ROM are automatically mounted, this command is not
necessary).
  <Example>
  % mount -r -F hsfs /dev/dsk/c0t6d0s2 /cdrom (Solaris)
  % mount /dev/dsk/c0t2d0 /SD_CDROM   (HP-UX)
   Specify the device name of the CD-ROM drive instead of "c0t6d0s2" and "c0t2d0".

2. Create the installation directory for the software (in the following explanation,
the installation directory "/usr/cross_soft" is assumed).

  % mkdir /usr/cross_soft

3. Move to the installation directory, and unpack the software.
  <Example>
  % cd /usr/cross_soft
  % tar -xvf /cdrom/sh_sparc/Program.tar (Solaris)
  % tar -xvf /SD_CDROM/"PROGRAM.TAR;1"   (HP-UX)

4. Set up the environment for the software execution (C Shell is assumed).

  % set path=(/usr/cross_soft $path)
  % setenv SHC_LIB /usr/cross_soft
  % setenv SHC_INC /usr/cross_soft
  % setenv SHC_TMP /usr/tmp

% setenv HLNK_TMP /usr/tmp
* Specifies the location for the intermediate files of the optimizing linkage editor.
Without this specification, intermediate files are created in the current directory.

% setenv HLNK_LIBRARY1 /usr/cross_soft/********.lib
% setenv HLNK_LIBRARY2 /usr/cross_soft/********.lib
* You can implicitly input these files without LIBRARY option or LIBRARY suboption to the linkage editor.
For details, please refer to "Compiler, Assembler, Optimizing Linkage Editor User's Manual".

% setenv HLNK_DIR /usr/cross_soft
* Specifies the location of the input file of the optimizing linkage editor.
For details, please refer to "Compiler, Assembler, Optimizing Linkage Editor User's Manual".

5. Unmount CD-ROM. (If the CD-ROM is automatically mounted, this command is not necessary.)

% umount /cdrom (Solaris)
% umount /SD_CDROM (HP-UX)

Usage of Stack Analysis Tool

Stack Analysis Tool can be used on Windows(R)98, Windows(R)Me, Windows NT(R)4.0, Windows(R)2000 or Windows(R)XP. Please copy the files under "Tools/Japanese" (Japanese)
or "Tools/English" (English) and use them.

Details of the usage are described in "Compiler, Assembler, Optimizing Linkage Editor User's Manual".

User's Manual

With this CD-ROM, you can read User's Manuals using Windows(R) and Acrobat(R) Reader. The following describes the usage.

1. Invoke Acrobat(R) Reader.
2. From Acrobat(R) Reader, open "Jshu0207.pdf" (Japanese) or "Eshu0207.pdf" (English).
To display the user's manual from the index, click on the name of the manual.

Caution

Please prepare Acrobat(R) Reader before using this CD-ROM.

Installation of Acrobat(R) Reader:

(1) Insert the CD-ROM into the CD-ROM drive. (Hereafter, the D drive is assumed.)
(2) Click [Run...] in the Windows(R) [Start] menu.
(3) Using the [Run] dialog box, select Ar505enu.exe (English edition) in the [PDF_Read\English]
directory or Ar505jpn.exe (Japanese edition) in the [PDF_Read\Japanese] in the CD-ROM (e.g.,
D:\PDF_Read\English\Ar505enu.exe), and click [OK].
(4) Follow the instruction of the installer displayed on the screen.

The programs may not work correctly when the required memory for each program cannot be allocated,
or when the system is not stable. Please check before using the CD-ROM.

Updates

---

1. Compiler (Ver.6.0A -> Ver.7.1.00)

1.1 Added and modified options (Ver.6.0A -> Ver.7.0B)

The following options were added.

(1) map=<file name>
Optimized for accesses to external variables.

(2) gbr={auto|user}
Automatically generates GBR-relative access and logical operation codes.

(3) shift={inline|runtime}
Generates a shift operation as inline code or call of runtime routine.

(4) blockcopy={inline|runtime}
Generates a block transfer code as inline code or call of runtime routine.

(5) latin1
Supports ISO-Latin1 code.

The following options were modified.

(6) division=cpu={inline|runtime}
Generates a division as inline code or call of runtime routine.

(7) inline=<number>
Specifies a percentage of increase in the code size due to the use of inline expansion.

(8) show=tab={4|8}
Specifies the size of a tab code in a listing file.

(9) nomessage[=<number>[,...]]
Inhibites an output of the specified information-level messages.

The nestinline option was abolished. The compiler automatically performs nested inline expansion.

1.2 Added and modified #pragma extentions (Ver.6.0A -> Ver.7.0B)

The following #pragma extentions were added.

(1) #pragma entry
Specifies the entry function.

(2) #pragma stacksize
Specifies the stacksize.

The following #pragma extentions were modified.

(1) #pragma interrupt
Supports sp=symbol+<number>.

(2) #pragma gbr_base/#pragma gbr_base1
Invalidates these #pragma extentions when gbr=auto is specified.

1.3 Supported section address operators (Ver.6.0A -> Ver.7.0B)

The following section address operators were supported.

(1) _ _sectop("<section name>")
Refers to the top address of <section name>.

(2) _ _sectop("<section name>")
Refers to the end address of <section name> + 1.

1.4 Added intrinsic functions (Ver.6.0A -> Ver.7.0B)

The following intrinsic functions were added.

(1) 64-bit multiplication

dmuls_h, dmuls_l, dmulu_h, dmulu_l
(2) SWAP instructions and endian convertion
swapb, swapw, end_cnvl
(3) LDTLB instruction and NOP instruction
ldtlb, nop

1.5 Relaxation of error detection against an address cast (Ver.6.0A -> Ver.7.0B)
Relaxed error detection of an cast expression against an address initialization when external variables are initialized.

1.6 Supported the variable allocated size of malloc and the variable number of I/O files (Ver.6.0A -> Ver.7.0B)
Specify the size to be allocated by malloc using the _sbrk_size variable.
Specify the number of I/O files using the _nfiles variable.

1.7 Incorrect sign/zero extension (Ver.7.0B -> Ver.7.0.03)
The problem in which the compiler might incorrectly sign/zero-extend the lower 1 or 2 byte in the result of a common subexpression is fixed.

```
short sub(long);
short a,b,c,d;

void f()
{
    if (d) {
        b = sub((long)a * (long)b * (long)c / 0x4000);
            /* ^^^^^^^^^^^^^^^^^ */
    } else {
        b = sub((long)a * (long)b * / 0x80);
            /* ^^^^^^^^^^^^^^^^^ */
    }
}
```

[Conditions]
The problem occurs if both of the following conditions are satisfied.
(1) A function has a common subexpression (marked with ^^^^ above) which produces the same result.
(2) It is guaranteed that the ranges of the operands in the common subexpression are no larger than 1 or 2 byte integer.
(In the above example, it is obvious that the ranges of the operands in the common subexpression are no larger than 2 byte integer because the types of the variables before type-conversion are short.)

1.8 Illegal destruction of register (Ver.7.0B -> Ver.7.0.03)
The problem in which the live contents of a register might be illegally destroyed with optimize=1 option is fixed.

[Conditions]
The problem occurs if all of the following conditions are satisfied.
(1) optimize=1 option is specified.
(2) Two or more basic blocks in a function have the same branch target including the loop entrance and return statements, where a basic block means a sequence of instrcutions without branches and labels inside.
(3) Each of the basic blocks mentioned in (2) above has assignments to the same variable.

1.9 Illegal assignment of 1-bit bitfield member (Ver.7.0B -> Ver.7.0.03)
The following problem is fixed.
An internal error may occur or incorrect object code may be generated if a structure has one or more 1-bit bitfields, if they are assigned constant

values sequentially and if more than one assignment to the same bitfield appears in sequence.

Example:

```
struct {
    unsigned char b1:1;
    unsigned char b2:1;
} ST;

/* Internal error */
void f() {
    ST.b1=1;
    ST.b2=0;
    ST.b2=0;
}

/* Illegal object */
void g() {
    ST.b1=1;
    ST.b2=1;
    ST.b2=0;
}
```

[Conditions]
The problem occurs if both of the following conditions are satisfied.
(1) A structure has one or more 1-bit bitfields and a sequence of simple assignments (=) of constant values to several 1-bit bitfields is written.
(2) More than one assignment to the same 1-bit bitfield in the sequence assign the same constant value (then an internal error occurs), or an assignment of a value other than 0 to a 1-bit bitfield is followed by an assignment of 0 to the same 1-bit bitfield in the sequence (then incorrect object code is generated).

1.10 Saveing/restoring registers illegally (Ver.7.0B -> Ver.7.0.03)
The following problem is fixed.
An interrupt function might not save or restore R0, R1 or R3 even though they are used in the function.

[Conditions]
The problem occurs if all of the following conditions are satisfied.
(1) The function is an interrupt function.
(2) The function does not have any function calls including runtime library calls.
(3) The function has one of the following.
  (a) a switch statement
  (b) a call of a function specified by #pragma inline_asm
    (the size of this function is no less than 255 or unspecified.)
  (c) an intra-function branch that does not reach using the 8 bit displacement

1.11 Illegal destruction of the R0 register (Ver.7.0B -> Ver.7.0.03)
The problem in which the live contents of the R0 register might be illegally destroyed with the optimize=1 option is fixed.

[Conditions]
The problem occurs if both of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) The function has one or more formal parameters allocated to the stack area.

1.12 Illegal reference of a struct member in a switch condition (Ver.7.0B -> Ver.7.0.03)
The following problem is fixed.
The offset of a struct member might be incorrect when a nested struct member exists in the condition of a switch statement.

[Conditions]
The problem occurs if all of the following conditions are satisfied.
(1) A nested struct member exists.
(2) A reference of this member is written with "(&member)->" instead of ".".
(3) This reference exists in the condition of a switch statement.

1.13 Illegal debugging information (Ver.7.0B -> Ver.7.0.03)
The following problem is fixed.
The debugging information is incorrect when a pointer to a function exists in a struct member and if the function is declared with a function prototype.

[Conditions]
The problem occurs if all of the following conditions are satisfied.
(1) The program is written in the C language.
(2) A pointer to a function exists in a struct or union member.

1.14 Internal error (Ver.7.0B -> Ver.7.0.03)
The bug in which an internal error occurs with one of the following conditions is fixed.
(1) A file path contains a blank.
(2) A Chinense/Japanese character is specified in the file path of the map option.

1.15 Illegal elimination of an if statement which has side effect (Ver.7.0.03 -> Ver.7.0.04)
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)

1.16 Illegal stack access (Ver.7.0.03 -> Ver.7.0.04)
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)

1.17 Illegal save/restore in #pragma interrupt function (Ver.7.0.03 -> Ver.7.0.04)

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.

1.18 Illegal save/register MAC registers (Ver.7.0.03 -> Ver.7.0.04)

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.

1.19 Illegal object of expression in a nested loop (Ver.7.0.03 -> Ver.7.0.04)

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.

1.20 Illegal elimination of sign/zero extension (Ver.7.0.03 -> Ver.7.0.04)

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.

1.21 Illegal expansion of switch statement (Ver.7.0.03 -> Ver.7.0.04)

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     _fl
        .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.

1.22 Illegal constant expression (Ver.7.0.03 -> Ver.7.0.04)
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 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).

1.23 Illegal alias access (Ver.7.0.03 -> Ver.7.0.04)
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.

1.24 Illegal save/restore of FPSCR in #pragma interrupt function (Ver.7.0.03 -> Ver.7.0.04)
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.

1.25 Illegal scheduling of intrinsic function (Ver.7.0.03 -> Ver.7.0.04)
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

1.26 Illegal elimination of sign/zero extension in a multiplication result (Ver.7.0.03 -> Ver.7.0.04)
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.

1.27 Illegal cast to the pointer to volatile type (Ver.7.0.03 -> Ver.7.0.04)
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.

1.28 Illegal object of GBR related logical operation (Ver.7.0.03 -> Ver.7.0.04)
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) ok(1);   /* The variable guc1 is not reloaded.          */
     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).

1.29 Illegal elimination of sign/zero extension using #pragma global_register (Ver.7.0.03 -> Ver.7.0.04)
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.

1.30 Illegal object using struct or union with struct or union array member (Ver.7.0.03 -> Ver.7.0.04)
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];       /* copy ans1 before set the value to st.u1[1].u12[1] (A) */
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).

1.31 Illegal optimization at linkage (Ver.7.0.03 -> Ver.7.0.04)
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.
(2) optimize=register is specified at linkage.
(3) Register save/restore code exists in delay slot.
(for example MOV.L @R15+,Rn/MOV.L Rn,@-R15)

1.32 Suppress movement of intrinsic function nop() to a delay slot (Ver.7.0.03 -> Ver.7.0.04)
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.

1.33 Illegal cast of signed char/short type constant (Ver.7.0.03 -> Ver.7.0.04)
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.

1.34 Illegal expression for a parameter with side effect (Ver.7.0.03 -> Ver.7.0.04)
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).

1.35 Illegal object of sign/zero extension of return value by rtnext (Ver.7.0.03 -> Ver.7.0.04)
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.

1.36 Illegal data of struct with anonymous bitfield member (Ver.7.0.03 -> Ver.7.0.04)
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.

1.37 Suppress elimination of variable reference whose variable is volatile (Ver.7.0.03 -> Ver.7.0.04)
When the following expression is described at a variable with a volatile declaration, a reference to the variable is guaranteed.

```
a &= 0;
```

1.38 Illegal debugging information (Ver.7.0.03 -> Ver.7.0.04)
The problem in which a debugging information of local array, struct, or union which is allocated on stack is not output is fixed.

1.39 Internal error (Ver.7.0.03 -> Ver.7.0.04)
The bug in which an internal error occurs with one of the following conditions is fixed.

1.40 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
For details, refer to "Supplement for SuperH RISC engine C/C++ Compiler Ver.7.0.06".

1.41 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.

1.42 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.

1.43 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.

1.44 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

1.45 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.

1.46 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.

1.47 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 consequtively by more than one instruction.

1.48 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.

1.49 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.

1.50 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) 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.

1.51 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.

1.52 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.

1.53 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                 ;                V
  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.

1.54 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++)

1.55 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.

1.56 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;
```

1.57 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 !=).

1.58 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.

1.59 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]

```
            F
    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
          F
```

[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
       F
    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.

2. Standard library Generator (Ver.1.0A -> Ver.2.0)

2.1 Supported reentrant library

Generates the reentrant library when the reent option is specified.

2.2 Supported simple I/O functions

Generates the simple and small I/O functions without floating-point convertion when the nofloat option is specified.

2.3 realloc space (from this package)

The following problem is fixed.

After the call of realloc(), malloc() may fail to allocate space even though there should be enough space remaining.

3. Assembler (Ver.5.0B -> Ver.5.1)

3.1 Removal of the restrictions on use of local labels or .ASSIGN definition symbols

Fixed the problem in which the PC-relative value in the instruction becomes invalid when a local label or a .ASSIGN definition symbol is used.

```
  .IMPORT isym
  MOV.L #(isym - ?lab),R0  ; ?lab is invalid.
       :
?lab:                      ; the same occurs when $ is assigned by .Assign.
       :
```

[Condition]

This problem occured when one of the following conditions was satisfied.

(1) The JMP/JSR branch to the .ASSIGN label or a local label is written.

(2) The expression including either .ASSIGN label or local label and external symbol (export/inport) is written.

4. Format converter (Ver.1.0B -> Ver.1.0.04)

4.1 Illegal line information after changing an assembly section (Ver.1.0B -> Ver.1.0C)

Fixed the problem in which the line information is illegal after section change when a section of an object file is changed in an assembly file.

4.2 Illegal align of the section (Ver.1.0B -> Ver.1.0C)

Fixed the problem in which the 2003 error illegally occurs when the ELF/DWARF2 file is changed for ELF/DWARF1 or SYSROF.

4.3 Illegal symbol-allocation address in an object without debugging information (Ver.1.0B -> Ver.1.0C)

Fixed the problem in which a symbol address is illegal in an object without debugging information when there are objects with/without debugging information in an absolute file.

4.4 Illegal conversion of an absolute file with cache optimization (Ver.1.0B -> Ver.1.0C)
Fixed the problem in which an application error occurs in a file with cache optimization of optlnk.

4.5 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.

5. Optimizing Linkage Editor (Ver.7.0A -> Ver.7.1.06)

5.1 Added options (Ver.7.0A -> Ver.7.1)
The following options were added.
(1) map[=<file name>]
Outputs a file for the compiler optimization for accesses to external variables.
(2) compress/nocompress
Specifies the compression of the debugging information.

5.2 Fixed internal errors (Ver.7.0A -> Ver.7.1)
Fixed the following internal errors.
(1703) "File:oc_updat.cpp Line:5496 /This IDisp is Limit Over" /"oc_update_jsrdsp()"
(1704) "File:oc_updat.cpp Line:6129 /This IDisp is Limit Over" /"oc_update_movdsp()"
(3061) "File:dw_arng.cpp Line:209 /Cannot find relocation table" /"DWArangeList::createArange()"
(3081) "File:dw_loc.cpp Line:191 /Illegal address size" /"createLocation()"
(8090) "File:in_devtb.cpp Line:9177 /Unexpected table end(add,sub,mov...etc..)" /"in_lastcode_chk_SH"
(8833) "File : ca_updat.cpp Line : 3163 /The contents of the OFFINF_SH->LITRINF table are not literal data." /"ca_update_Add_Literal _Operation"
(8870) "File:oc_updat.cpp Line:10251 /Position is NULL" /"OBF_AT"
(8874) "File:oc_serch.cpp Line:2583 /Position is NULL" /"SUB_AT"

5.3 Error at creating a library (Ver.7.0A -> Ver.7.1)
Fixed the following problem.
The error "L3310 (F) Cannot open temporary file" occurs and no library is generated, when the standard library generator for UNIX is used and when HLNK_DIR is specified at generating a standard library.

5.4 Illegal linking of absolute address sections (Ver.7.0A -> Ver.7.1)
Fixed the problem in which an object is illegal when an absolute address section of size 0 is linked with the same name after the absolute address section.

5.5 Illegal execution with goptimize option (Ver.7.0A -> Ver.7.1)
Fixed the problem in which the execution is illegal when there is a relationship that calls a function with the goptimize option from functions without (including an assembly function) and with the goptimize option.

5.6 Illegal branch width for a symbol (Ver.7.0A -> Ver.7.1)
Fixed the problem in which the branch width for a symbol created by optimizing the common code is illegal when the following conditions are satisfied at the same time.

[Conditions]
(1) The goptimize option is specified in an input object.
(2) A code section is specified as the operand of rom option.
(3) Optimize is specified at linking.

5.7 Deleting an illegal symbol (Ver.7.0A -> Ver.7.1)
Fixed the problem in which a symbol is illegally deleted when the following conditions are

satisfied at the same time.

[Conditions]
(1) A code exists to refer to an object without goptimize option from an object with gopttimize option.
(2) ENTRY is specified.
(3) Deleting unreferred variables and functions is optimized (at specifying optimize=symbol_delete).

5.8 Illegal save/retrieve register (Ver.7.0A -> Ver.7.1)
Fixed the problem in which the register save/restore code is illegally optimized when a function
that is defined next from a function in the same C source.

5.9 Illegal literal pool value (Ver.7.0A -> Ver.7.1)
Fixed the bug in which a literal pool value becomes invalid when the following conditions are satisfied at the same time with the optimization of register save/restore code.

[Conditions]
(1) The register save/restore code of the corresponding function is optimized.
(2) The corresponding function has a stack access code across the register save/restore area.
(3) The corresponding function recieves a parameter passed via the stack and its offset from
SP is 2-byte.

5.10 Incorrect debug information caused by the rename option (Ver.7.1 -> Ver.7.1.02)
The problem in which the debug information of the symbols in the renamed section is deleted with the -form=relocate option is fixed.

[Conditions]
If both of the following, (1) and (2), are satisfied, the problem occurs.
(1) the -form=relocate option is specified.
(2) the -rename option is specified.

5.11 Invalid optimization of constant or literal data (Ver.7.1 -> Ver.7.1.02)
The problem in which symbols are incorrectly unified with the -optimize=string_unify is fixed.

[Conditions]
If both of the following, (1) and (2), are satisfied, the problem occurs.
(1) A C source file is compiled with the -goptimize option.
(2) The -optimize=string_unify option is specified to the optimizing linkage editor.

5.12 Internal error caused by cache optimization (Ver.7.1 -> Ver.7.1.02)
The problem in which an internal error occurs when both -optimize and -cache options are specified is fixed.

5.13 Incorrect debug information caused by the compress option (Ver.7.1.02 -> Ver.7.1.04)
Fixed is the problem in which compressed debug infomation specified with the compress option is incorrect.

5.14 Address check regarding external symbol allocation optimization (Ver.7.1.02 -> Ver.7.1.04)
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.

5.15 Incorrect section attribute when a binary file is input (Ver.7.1.02 -> Ver.7.1.04)
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).

5.16 Internal error caused by the form=relocate option (Ver.7.1.02 -> Ver.7.1.04)
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.

5.17 Internal error(1703) caused by the symbol_delete and register optimizations (Ver.7.1.02 -> Ver.7.1.04)
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.

5.18 Incorrect literal referring (Ver.7.1.02 -> Ver.7.1.04)
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.

5.19 Internal error(7041) with the output option specified (Ver.7.1.04 -> Ver.7.1.05)
Fixed is the problem in which an internal error(7041) occurs when the following conditions are satisfied at the same time.

[Conditions]
This problem may occur when all of the following conditions are satisfied.
(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.

5.20 Incorrect optimization with partial suppression of the optimization (Ver.7.1.04 -> Ver.7.1.05)
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.

5.21 Incorrect object code in generating a relocatable file (Ver.7.1.04 -> Ver.7.1.05)
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=rel option is specified.
(3) The delete or rename option is specified.

5.22 Incorrect error with optimization of external variable accesses (Ver.7.1.04 -> Ver.7.1.05)
Fixed is the problem in which an incorrect error occurs when the following conditions are satisfied at the same time.

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The input file is compiled with the map option.
(2) Sections are overlaid in the start option.

5.23 Internal error(1703) regarding the register save/restore optimization (Ver.7.1.04 -> Ver.7.1.05)
Fixed is the problem in which an internal error(1703) may occur when the optimization regarding register save/restore is specified.

5.24 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.

5.25 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.

5.26 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.

5.27 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.

5.28 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:
(1) No literal exists in the final function in the file.
(2) There is a branch destination immediately after the instuction for restoring registers.
(3) The SUBC instruction is included in the function to be optimized.

6. SuperH RISC engine simulator/debugger (Ver.4.00 -> Ver.4.2.00)

6.1 Supporting the SH-4R Simulator (Ver.4.00 -> Ver.4.11)
This simulator/debugger supports SH-4R(SH7750R).
The simulator supports 16KB/2WAY cache.

6.2 Supporting the Cache with the SH-4BSC Simulator (Ver.4.00 -> Ver.4.11)
The following failure was modified:

If the P1 area (H'80000000 to H'9FFFFFFF) is used in the copy-back mode (write-through mode by default) with the SH-4BSC simulator, the cache operation will be illegal.
When the P1 area is used in the copy-back mode, use the SH-4 simulator.

6.3 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.

7. Stack Analysis tool (Ver.1.2.00 -> Ver.1.3.00)

7.1 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.