Ｉｎｌｉｎｅ関数の対応について（ルネサス純正Ｃ版）

Ｒｅｖ１．２０　２００９／０４／２４
ＤＥＦバージョン６．８０Ｃ仕様　より

【対象】

ルネサスＣ（ELF/Dwarf2）でコンパイル／リンクした全CPU品種が対象になります。

【概要】

ＤＥＦバージョン６．８０Ｂ仕様より、Inline関数記述した場合でも出来る限りデバッグし易くするため工夫しました。

その特徴および仕様に関する説明を記述します。

【１．宣言およびＣ記述例】　Inline関数をヘッダファイルにまとめて使用した場合

<inline.h>　　H8/300,H8S,H8SX の場合　[H8S,H8/300 Tool Chain Ver6.2.0.0]　　記述例１

```
#define Inline                  static __inline
//*************************************************
Inline char sil_and_mem(char *mem,char and)
{
        return (*((volatile char *) mem) &= and);
}
//*************************************************
Inline char sil_xor_mem(char *mem,char xor)
{
        return (*((volatile char *) mem) ^= xor);
}
```

[1-0-1]
Inline関数をまとめたヘッダーファイル例１
【対象】　H8/300x, H8S, H8SX の全品種

<inline.h>　　SH-2/E/A の場合　[SuperH RISC engine Tool Chain Ver9.2.0.0]　　記述例２

```
#define Release     // debug 中はコメントにする。
#ifdef      Release
        #pragma inline(sil_and_mem)
        #pragma inline(sil_xor_mem)
#endif
//*************************************************
static char sil_and_mem(char *mem,char and)
{
        return (*((volatile int *) mem) &= and);
}
//*************************************************
static char sil_xor_mem(char *mem,char xor)
{
        return (*((volatile char *) mem) ^= xor);
}
```

[1-0-2]
Inline関数をまとめたヘッダーファイル例２
【対象】　SH-2/SH-2E/SH-2A の全品種

<main.c>　　Inline 関数の使用例　　記述例３

```
#include "inline.h"         // inline 関数をここでインクルードする。
// 変数宣言
char Temp[3];
// メイン　Inline 関数の使用例
void main(void) // line(15)
{
        char        *mem;
        while(1) { // line(18)
                mem = (char *)0xffe000; // line(19)
                Temp[0] = sil_and_mem((char *)mem+0x100,0x10); //line(20)
                Temp[1] = sil_xor_mem((char *)mem+0x100,0x20); //line(21)
                Temp[0] = sil_and_mem((char *)mem+0x102,0x10); //line(22)
                Temp[1] = sil_xor_mem((char *)mem+0x102,0x20); //line(23)
                Temp[0] = sil_and_mem((char *)mem+0x100,0x40); //line(24)
        } // line(25)
} // line(26)
// line(27)
```

[1-0-3]
Inline 関数の使用例
【対象】　全品種共通

【説明】

ルネサス純正Ｃの場合、ＣＰＵ品種別に Inline 関数時のライン情報(ELF/Dwarf2)出力に相違がありましたので、代表品種別で説明します。

【ルネサス純正Ｃ－パターン１】

＜H8／３０３８Ｆ－ONEの例＞　対象：H8／３００，H8／３００H，H8／Ｔｉｎｙシリーズの場合

・H8S,H8/300 Tool Chain(Ver6.2.0.0)　・C/C++ Compiler(Ver6.02.00)　・Assmebler(Ver6.02.00)　・Optimizing Linkage Editor(Ver9.03.00)

[1-1-1]

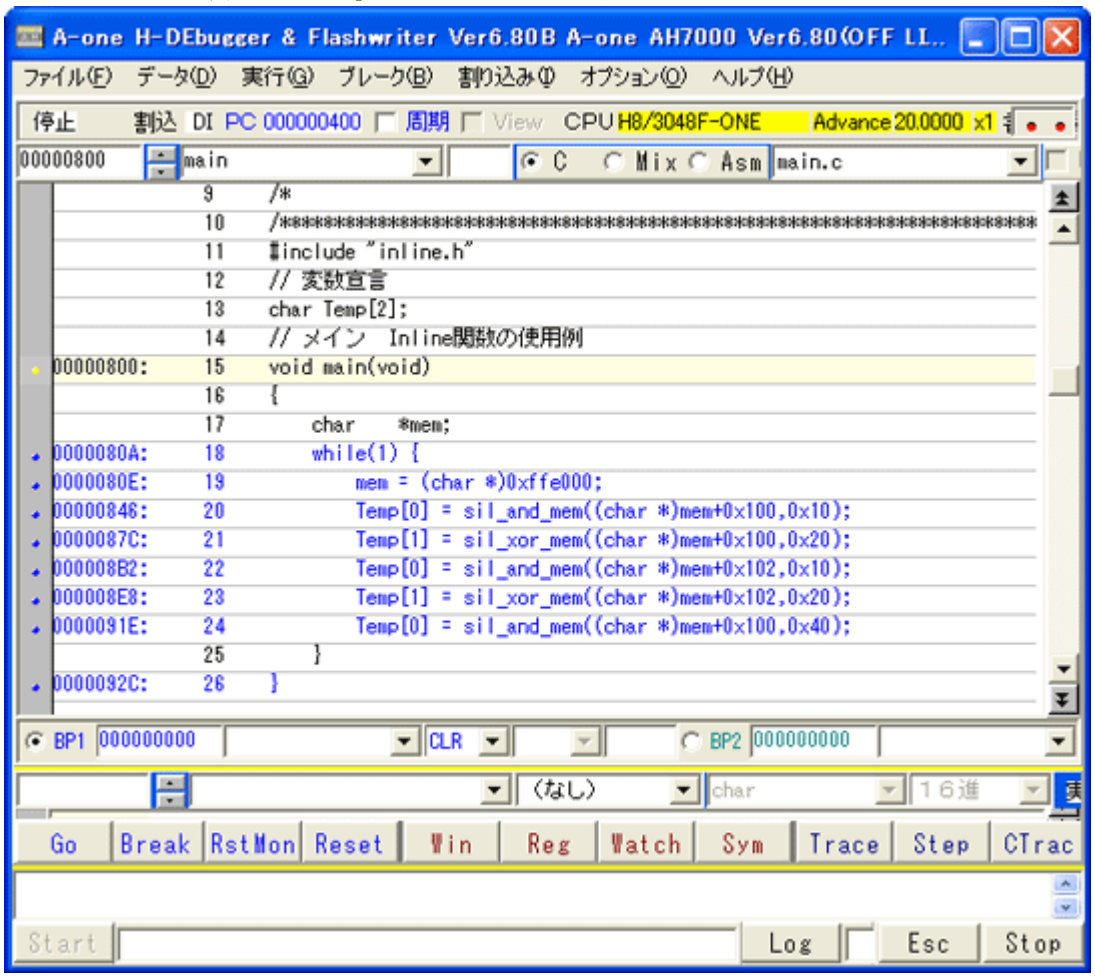
[ELF/Dwarf2 .debug_line section Code]	(address) (line)	(address) (line)	
/ DW_LNS_extended(0x00) DW_set_address(0x02) adr(0x800)	+**** main.c		
/ DW_advence_line(0x03) op(14) line(15)			
/ DW_copy(0x01) Lines(15) Address(0x800)	+0x00000800 15		
/ DW_special(0x36) dl(3) line(18) da(0xA) adr(0x80A)	+0x0000080A 18		
/ DW_special(0x1C) dl(1) line(19) da(0x4) adr(0x80E)	+0x0000080E 19		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-14) line(5)			
/ DW_advence_pc(0x02) pc(0x6) adr(0x81A)			
/ DW_copy(0x01) Lines(5) Address(0x81A)		(+0x0000081A 5)	*
/ DW_special(0x6C) dl(1) line(6) da(0x18) adr(0x832)		(+0x00000832 6)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(14) line(20)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x846)			
/ DW_copy(0x01) Lines(20) Address(0x846)	+0x00000846 20		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-10) line(10)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x850)			
/ DW_copy(0x01) Lines(10) Address(0x850)		(+0x00000850 10)	*
/ DW_special(0x6C) dl(1) line(11) da(0x18) adr(0x868)		(+0x00000868 11)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(10) line(21)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x87C)			
/ DW_copy(0x01) Lines(21) Address(0x87C)	+0x0000087C 21		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-16) line(5)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x886)			
/ DW_copy(0x01) Lines(5) Address(0x886)		(+0x00000886 5)	*
/ DW_special(0x6C) dl(1) line(6) da(0x18) adr(0x89E)		(+0x0000089E 6)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(16) line(22)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x8B2)			
/ DW_copy(0x01) Lines(22) Address(0x8B2)	+0x000008B2 22		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-12) line(10)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x8BC)			
/ DW_copy(0x01) Lines(10) Address(0x8BC)		(+0x000008BC 10)	*
/ DW_special(0x6C) dl(1) line(11) da(0x18) adr(0x8D4)		(+0x000008D4 11)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(12) line(23)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x8E8)			
/ DW_copy(0x01) Lines(23) Address(0x8E8)	+0x000008E8 23		
/ DW_advence_line(0x03) op(-5) line(18)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x8F2)			
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-18) line(5)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x8F2)			
/ DW_copy(0x01) Lines(5) Address(0x8F2)		(+0x000008F2 5)	*
/ DW_special(0x6C) dl(1) line(6) da(0x18) adr(0x90A)		(+0x0000090A 6)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(18) line(24)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x91E)			
/ DW_copy(0x01) Lines(24) Address(0x91E)	+0x0000091E 24		
/ DW_advence_line(0x03) op(-6) line(18)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x928)			
/ DW_copy(0x01) Lines(18) Address(0x928)	+0x00000928 18		
/ DW_advence_line(0x03) op(8) line(26)			
/ DW_advence_pc(0x02) pc(0x2) adr(0x92C)			
/ DW_copy(0x01) Lines(26) Address(0x92C)	+0x0000092C 26		
/ DW_advence_pc(0x02) pc(0x3) adr(0x932)	+0x00000932 27		

<main.c>
ELF/Dwarf2 のライン情報

Inline 関数の場合、
＊のように多重情報になる。
つまり、１行のソースラインに複数アドレスが存在します。

＊も同上

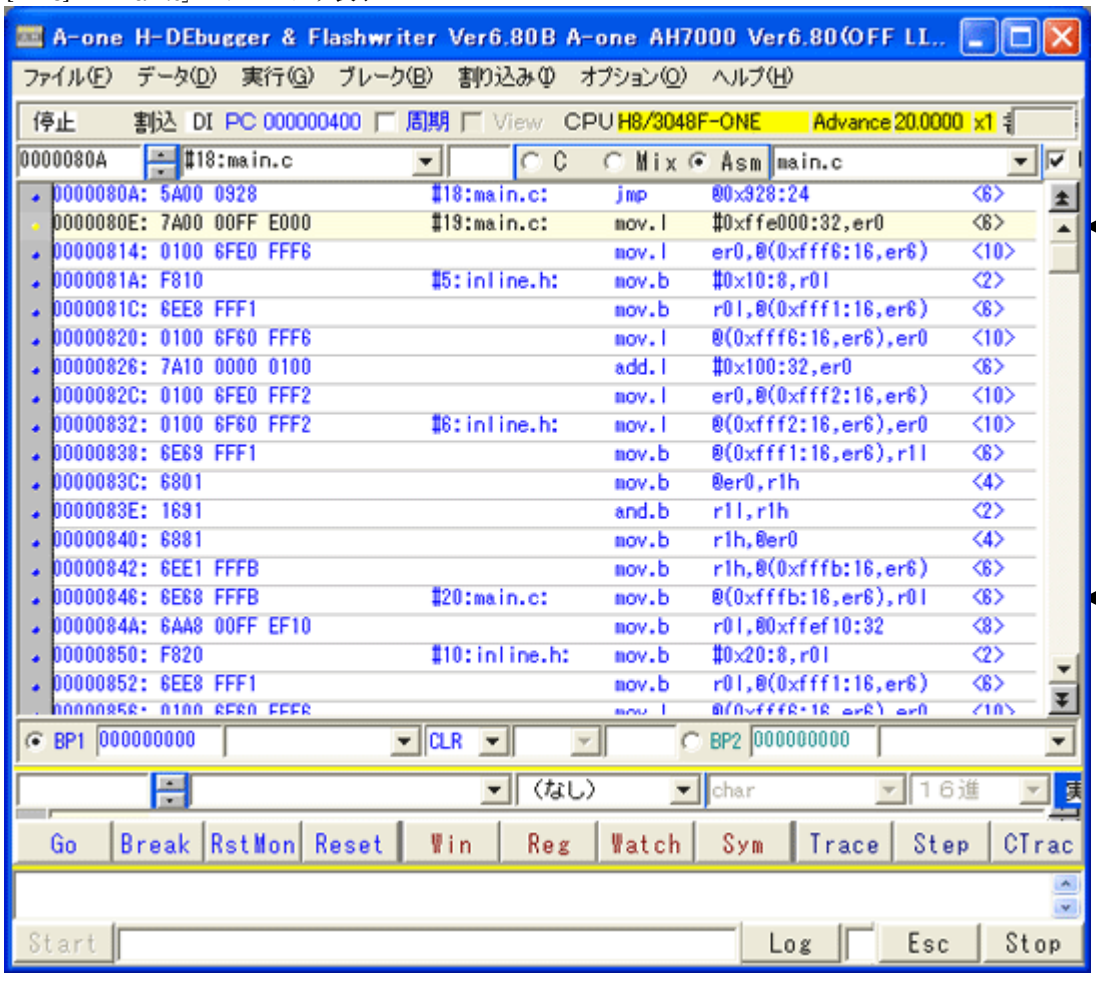
[1-1-2]　上記ライン情報で「main.c」をＣＶｉｅｗ表示した場合



⚠注意点（1）

上記ライン情報から判るように、「main.c」のライン（１９）とライン（２０）の間にインライン関数部分のコードが生成されています。
つまり、ライン（２０）の時点で、既にインライン関数の処理は終了していることになります。
若干イメージと違いますので注意して下さい。

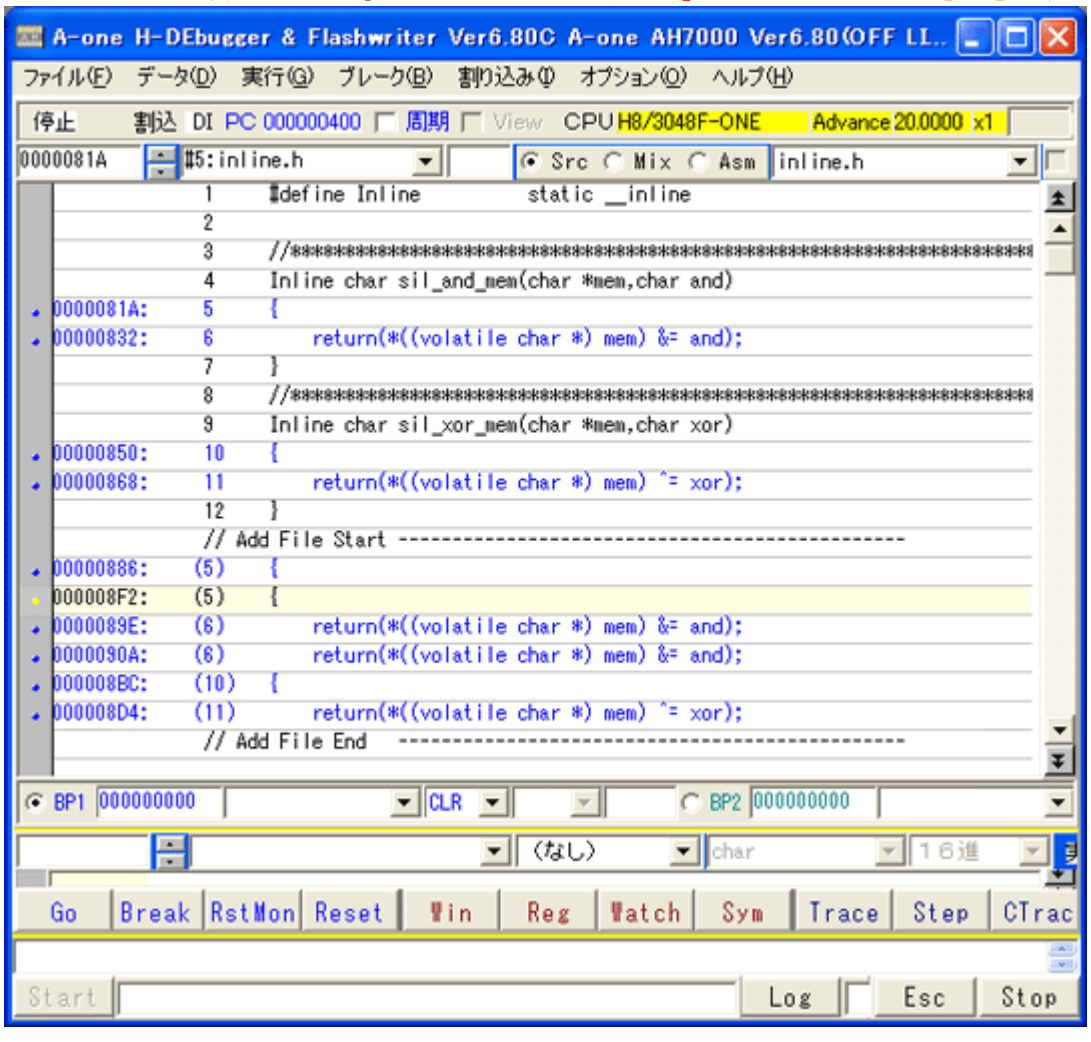
[1-1-3]　「main.c」のアセンブラ表示



⚠注意点（2）

「main.c」のライン（１９）と（２０）の間にインライン関数「inline.h」のライン（5）～(6)のコードが入っているのが確認できます。

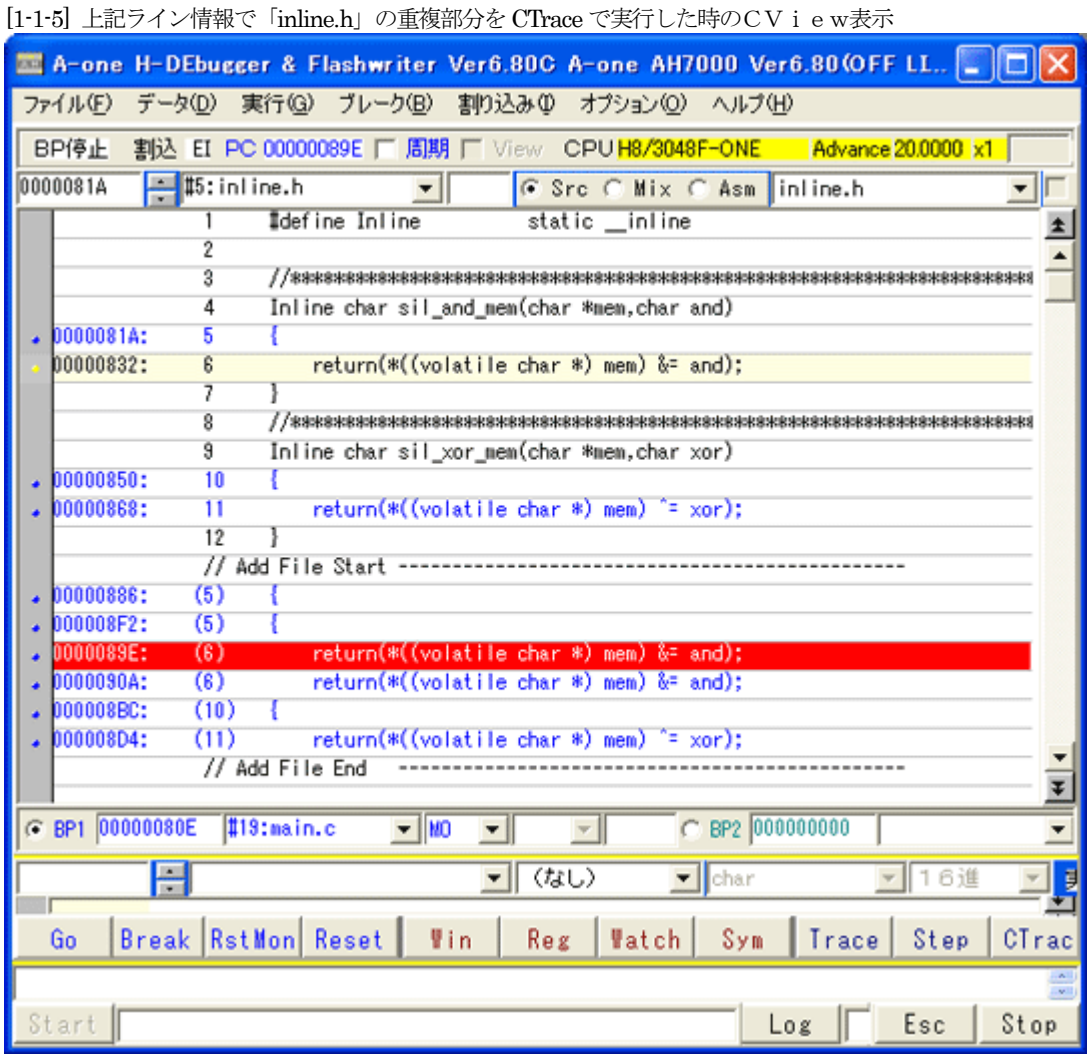
[1-1-4]　上記ライン情報で「inline.h」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［－m］の指定が無い場合】**



ポイント（新機能）

重複している「inline.h」のライン５～６とライン１０～１１を左図のように「inline.h」に追加(Add File)して表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

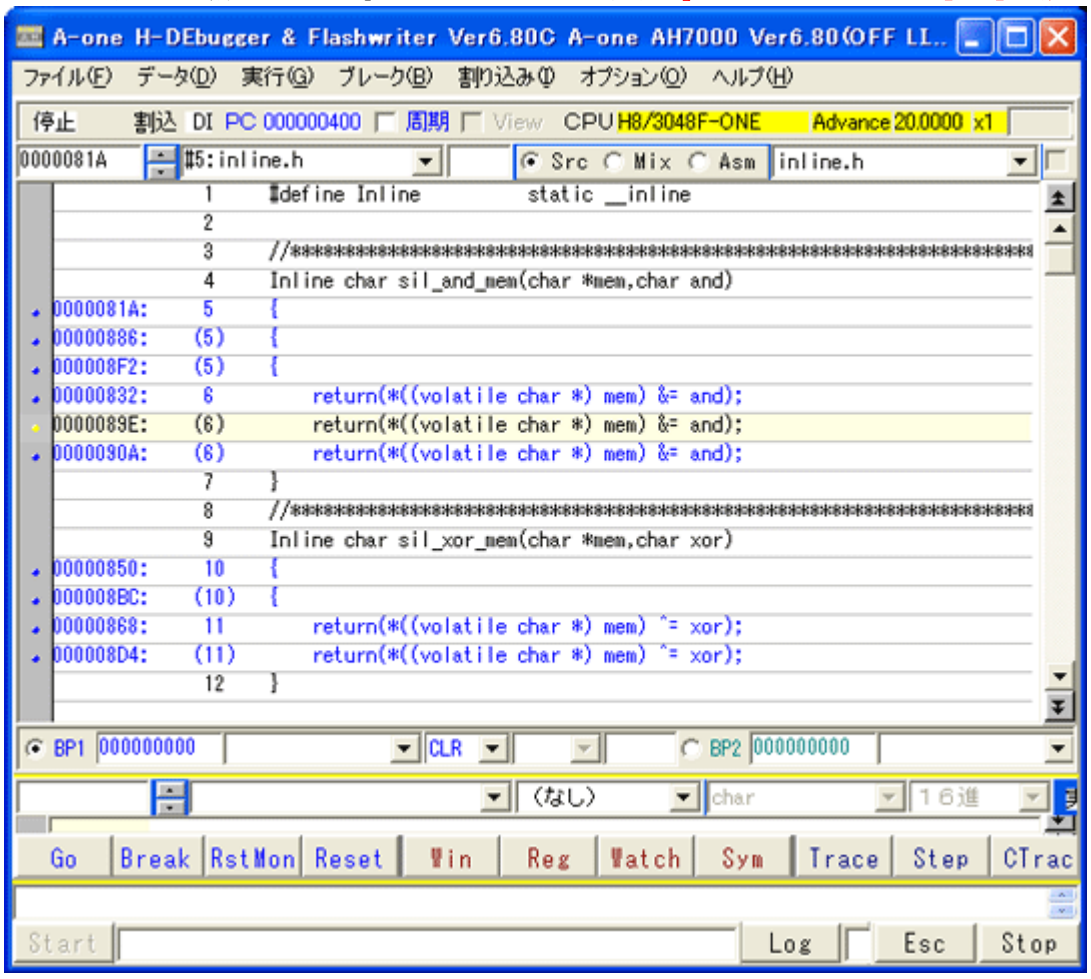
[1-1-5] 上記ライン情報で「inline.h」の重複部分を CTrace で実行した時のＣＶｉｅｗ表示



ポイント（新機能）

「main.c」ライン（２１）から CTrace した場合、
この位置で「inline.h」ソースライン（6）の PC 値赤帯表示をします。

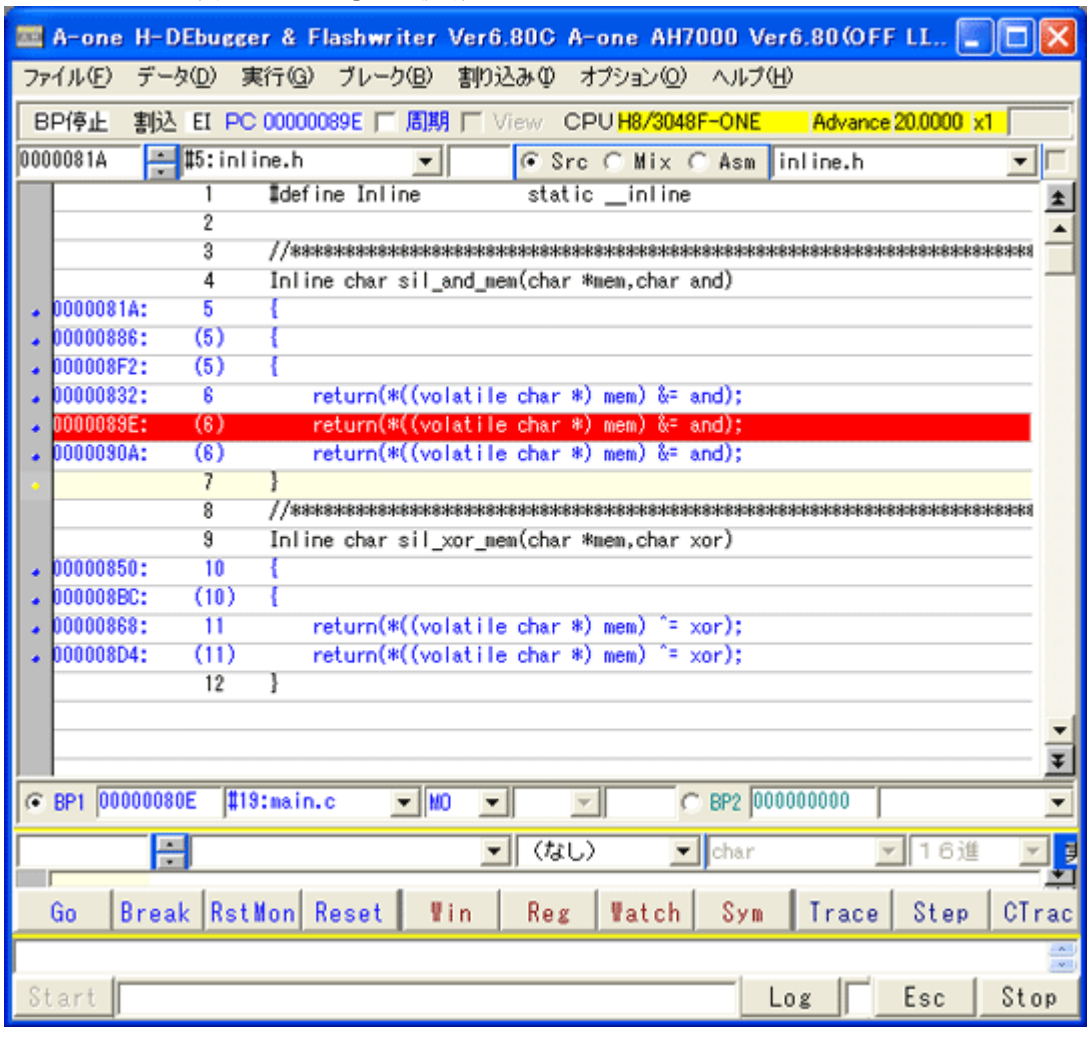
[1-1-6]　上記ライン情報で「inline.h」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［－m］を指定した場合】**



ポイント（新機能）

重複している「inline.h」のライン５～６とライン１０～１１を左図のように「inline.h」にマージして表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

[1-1-7] 上記ライン情報で「inline.h」の重複部分を CTrace で実行した時のＣＶｉｅｗ表示



ポイント（新機能）

「main.c」ライン（２１）から CTrace した場合、
この位置で「inline.h」ソースライン（６）の PC 値赤帯表示をします。

【ルネサス純正Cーパターン２】

＜Ｈ８ＳＸ／１５４４Ｆの例＞　対象：Ｈ８Ｓ，Ｈ８ＳＸシリーズの場合

・H8S,H8/300 Tool Chain(Ver6.2.0.0)　・C/C++ Compiler(Ver6.02.00)　・Assmebler(Ver6.02.00)　・Optimizing Linkage Editor(Ver9.03.00)

[1-2-1]

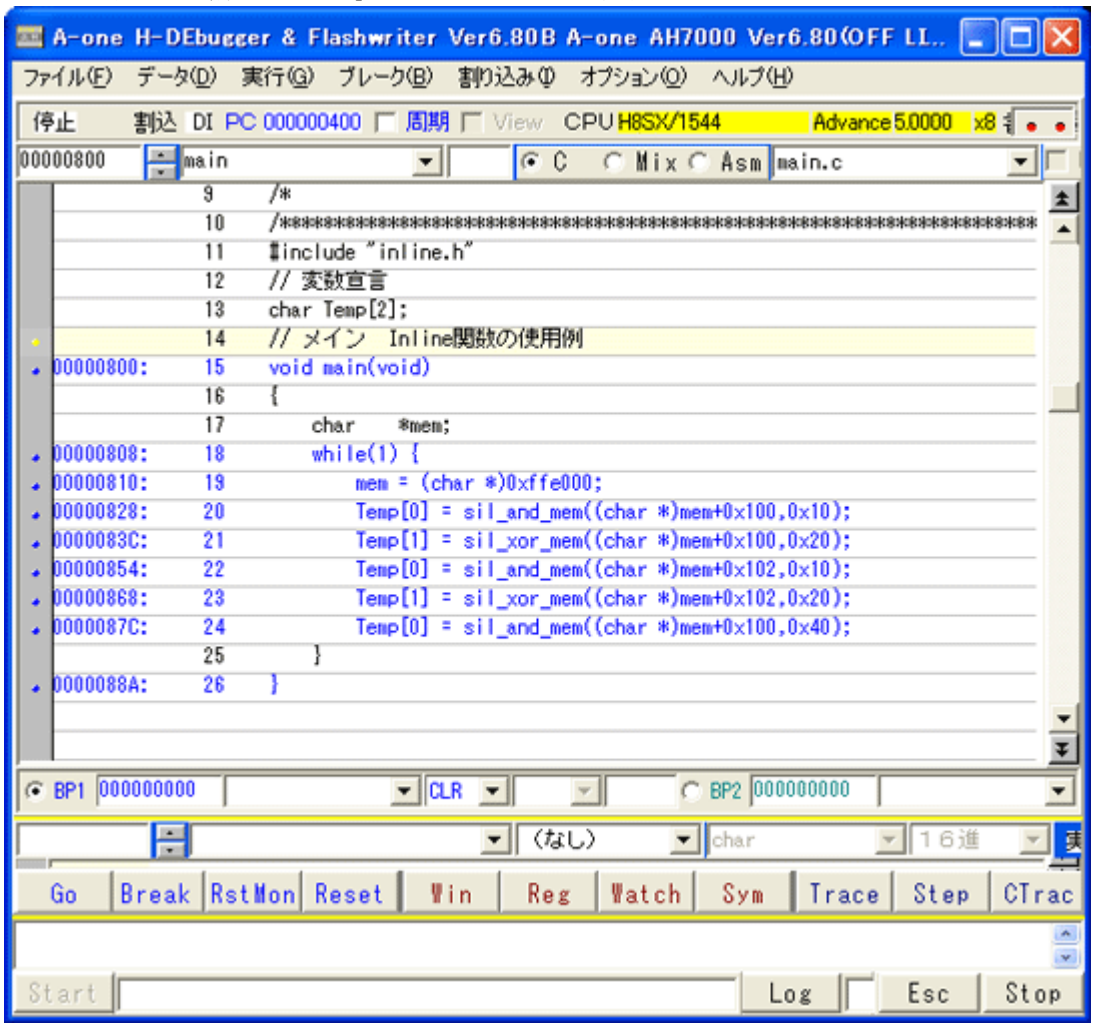
[ELF/Dwarf2 .debug_line section Code]	(address) (line)	(address) (line)	
/ DW_LNS_extended(0x00) set_address(0x02) adr(0x800)	+**** main.c		
/ DW_advence_line(0x03) op(14) line(15)			
/ DW_copy(0x01) Lines(15) Address(0x800)	+0x00000800 15		
/ DW_special(0x2E) dl(3) line(18) da(0x8) adr(0x808)	+0x00000808 18		
/ DW_special(0x2C) dl(1) line(19) da(0x8) adr(0x810)	+0x00000810 19		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-13) line(6)			
/ DW_advence_pc(0x02) pc(0x8) adr(0x820)			
/ DW_copy(0x01) Lines(6) Address(0x820)		(+0x00000820 6)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(14) line(20)			
/ DW_advence_pc(0x02) pc(0x4) adr(0x828)			
/ DW_copy(0x01) Lines(20) Address(0x828)	+0x00000828 20		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-9) line(11)			
/ DW_advence_pc(0x02) pc(0x6) adr(0x834)			
/ DW_copy(0x01) Lines(11) Address(0x834)		(+0x00000834 11)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(10) line(21)			
/ DW_advence_pc(0x02) pc(0x4) adr(0x83C)			
/ DW_copy(0x01) Lines(21) Address(0x83C)	+0x0000083C 21		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-15) line(6)			
/ DW_advence_pc(0x02) pc(0x8) adr(0x84C)			
/ DW_copy(0x01) Lines(6) Address(0x84C)		(+0x0000084C 6)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(16) line(22)			
/ DW_advence_pc(0x02) pc(0x4) adr(0x854)			
/ DW_copy(0x01) Lines(22) Address(0x854)	+0x00000854 22		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-11) line(11)			
/ DW_advence_pc(0x02) pc(0x6) adr(0x860)			
/ DW_copy(0x01) Lines(11) Address(0x860)		(+0x00000860 11)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(12) line(23)			
/ DW_advence_pc(0x02) pc(0x4) adr(0x868)			
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-17) line(6)			
/ DW_advence_pc(0x02) pc(0x6) adr(0x874)			
/ DW_copy(0x01) Lines(6) Address(0x874)		(+0x00000874 6)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(18) line(24)			
/ DW_advence_pc(0x02) pc(0x4) adr(0x87C)			
/ DW_copy(0x01) Lines(24) Address(0x87C)	+0x0000087C 24		
/ DW_advence_line(0x03) op(-6) line(18)			
/ DW_advence_pc(0x02) pc(0x3) adr(0x882)			
/ DW_copy(0x01) Lines(18) Address(0x882)	+0x00000882 18		
/ DW_advence_line(0x03) op(8) line(26)			
/ DW_advence_pc(0x02) pc(0x4) adr(0x88A)			
/ DW_copy(0x01) Lines(26) Address(0x88A)	+0x0000088A 26		
/ DW_advence_pc(0x02) pc(0x3) adr(0x890)	+0x00000890 27		

<main.c>
ELF/Dwarf2 のライン情報

Inline 関数の場合、
＊のように多重情報になる。
つまり、１行のソースライン
に複数アドレスが存在し
ます。

＊も同上

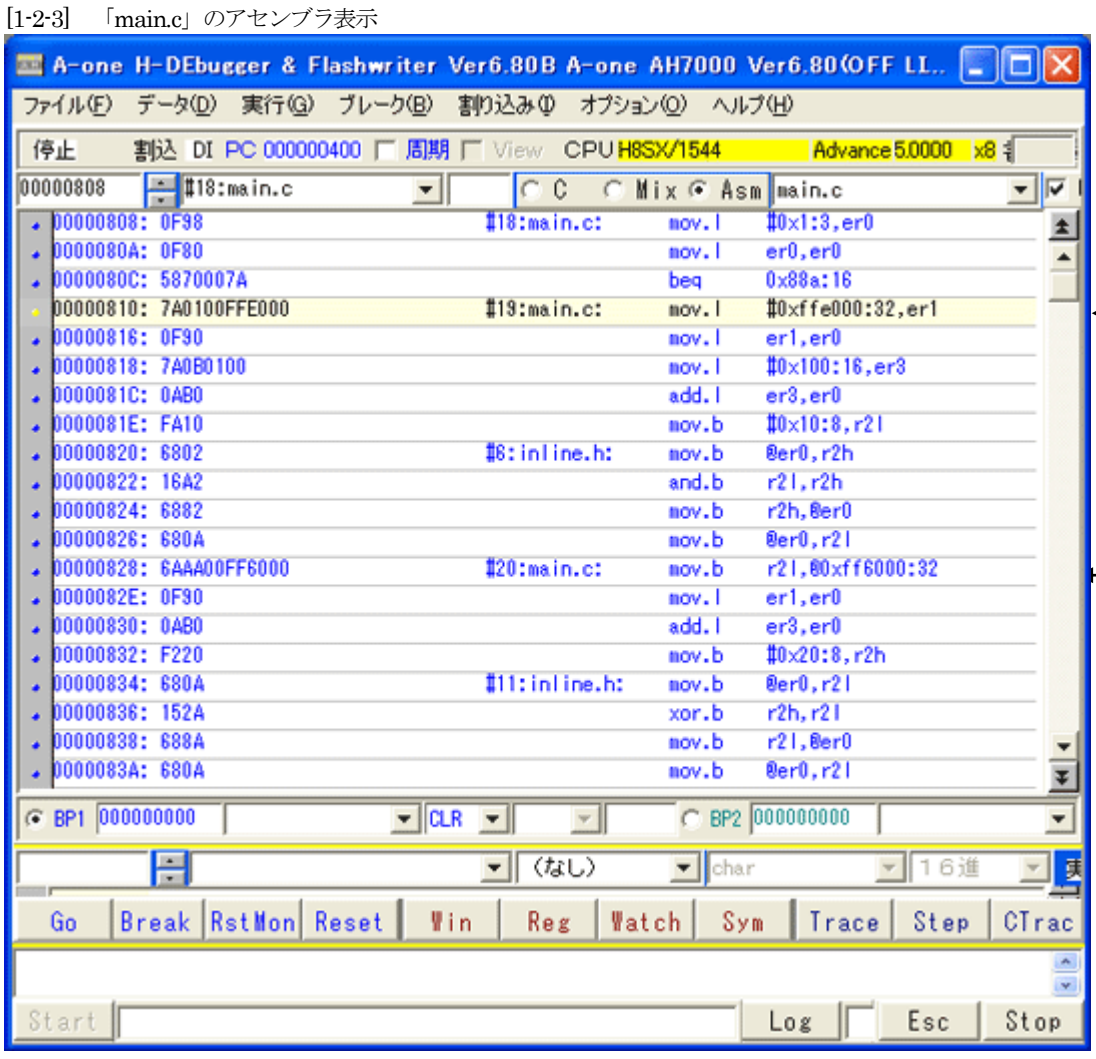
[1-2-2]　上記ライン情報で「main.c」をＣＶｉｅｗ表示した場合



注意点（1）

上記ライン情報から判るように、「main.c」のライン（１９）とライン（２０）の間にインライン関数部分のコードが生成されています。
つまり、ライン（２０）の時点で、既にインライン関数の処理は終了していることになります。
若干イメージと違いますので注意して下さい。

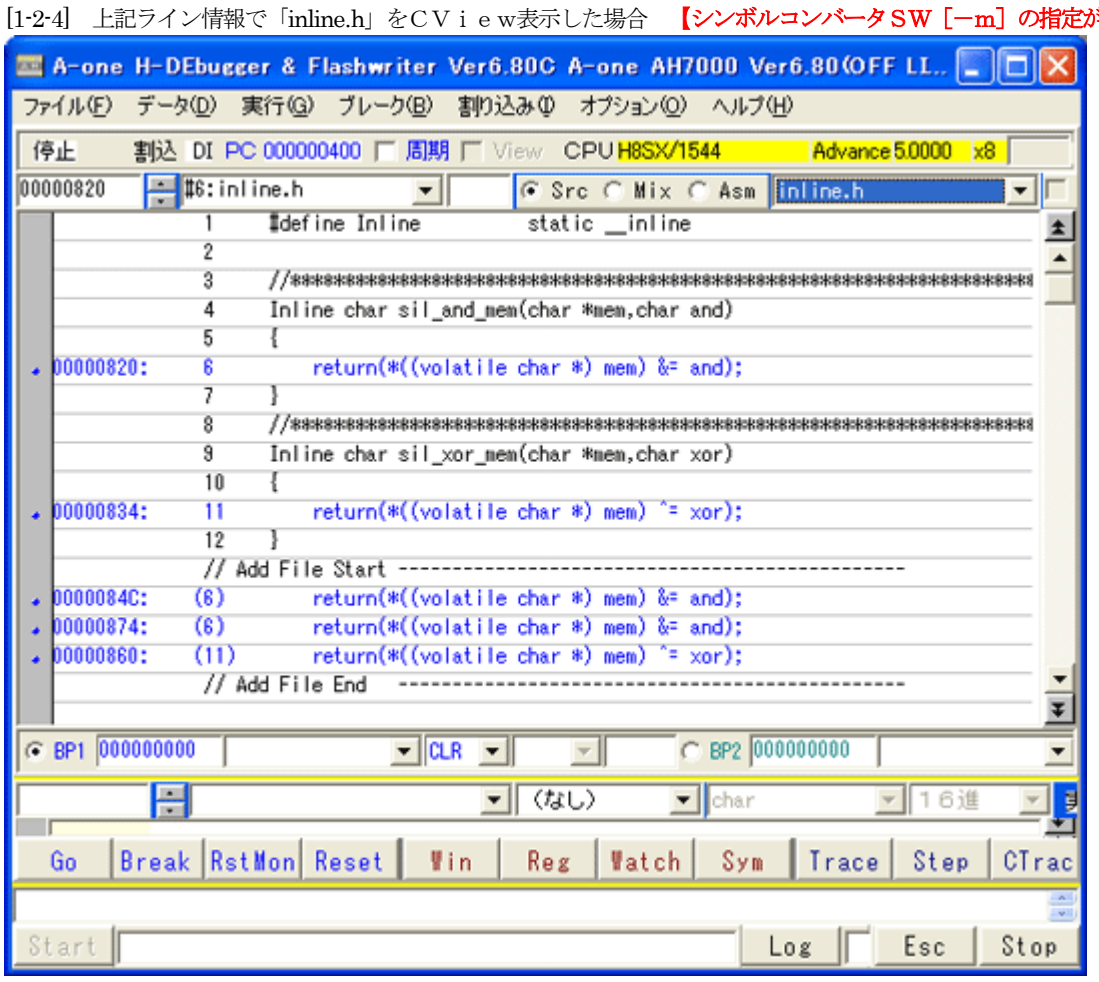
[1-2-3]　「main.c」のアセンブラ表示



注意点（2）

「main.c」のライン（１９）と（２０）の間にインライン関数「inline.h」のライン（6）のコードが入っているのが確認できます

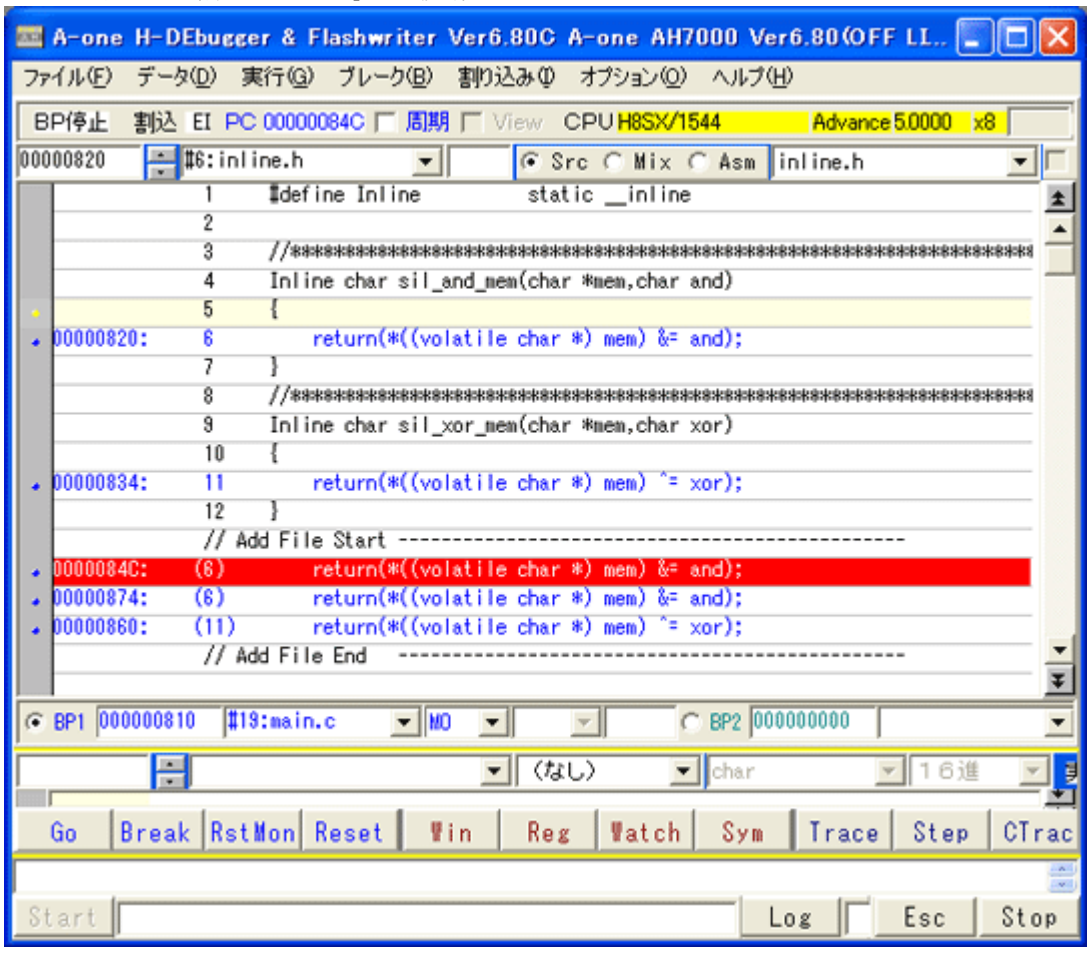
[1-2-4]　上記ライン情報で「inline.h」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［－m］の指定が無い場合】**



⚠ポイント（新機能）

重複している「inline.h」のライン（６）とライン（１１）を左図のように「inline.h」に追加(Add File)して表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

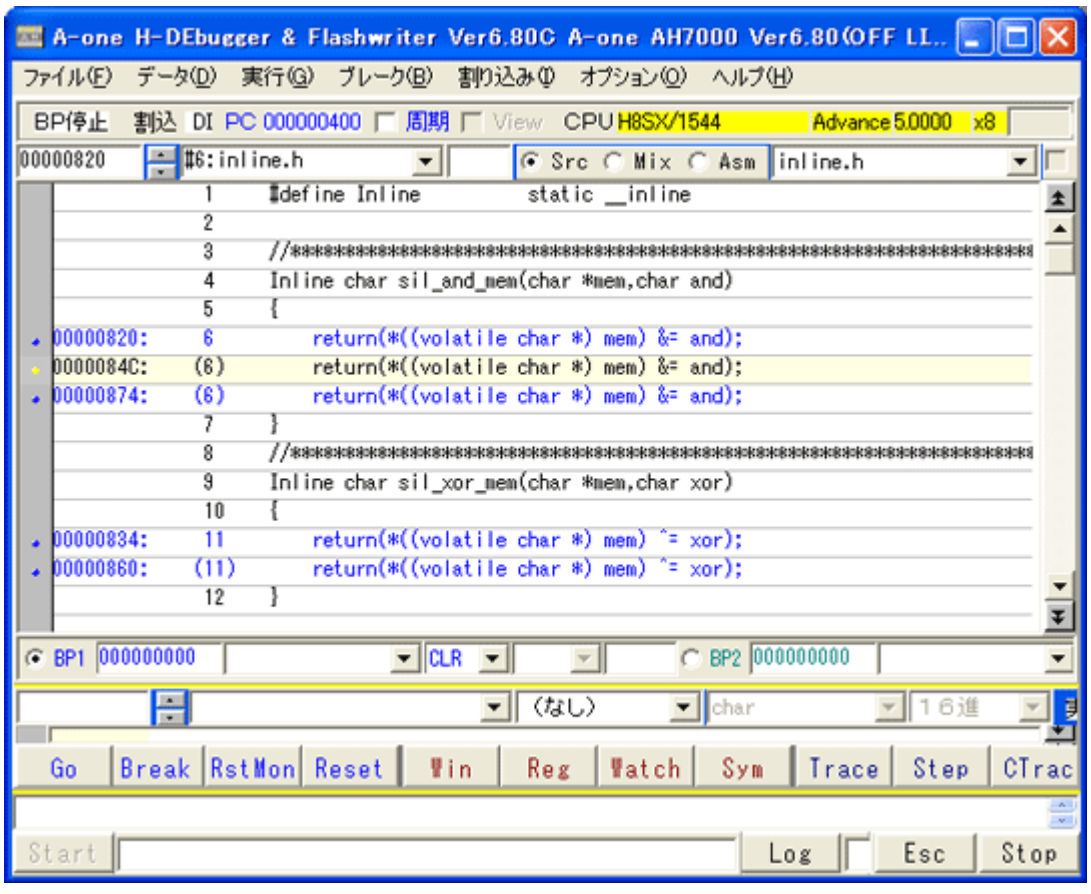
[1-2-5] 上記ライン情報で「inline.h」の重複部分を CTrace で実行した時のＣＶｉｅｗ表示



⚠ポイント（新機能）

「main.c」ライン（２１）から CTrace した場合、
この位置で「inline.h」ソースライン（６）の PC 値赤帯表示をします。

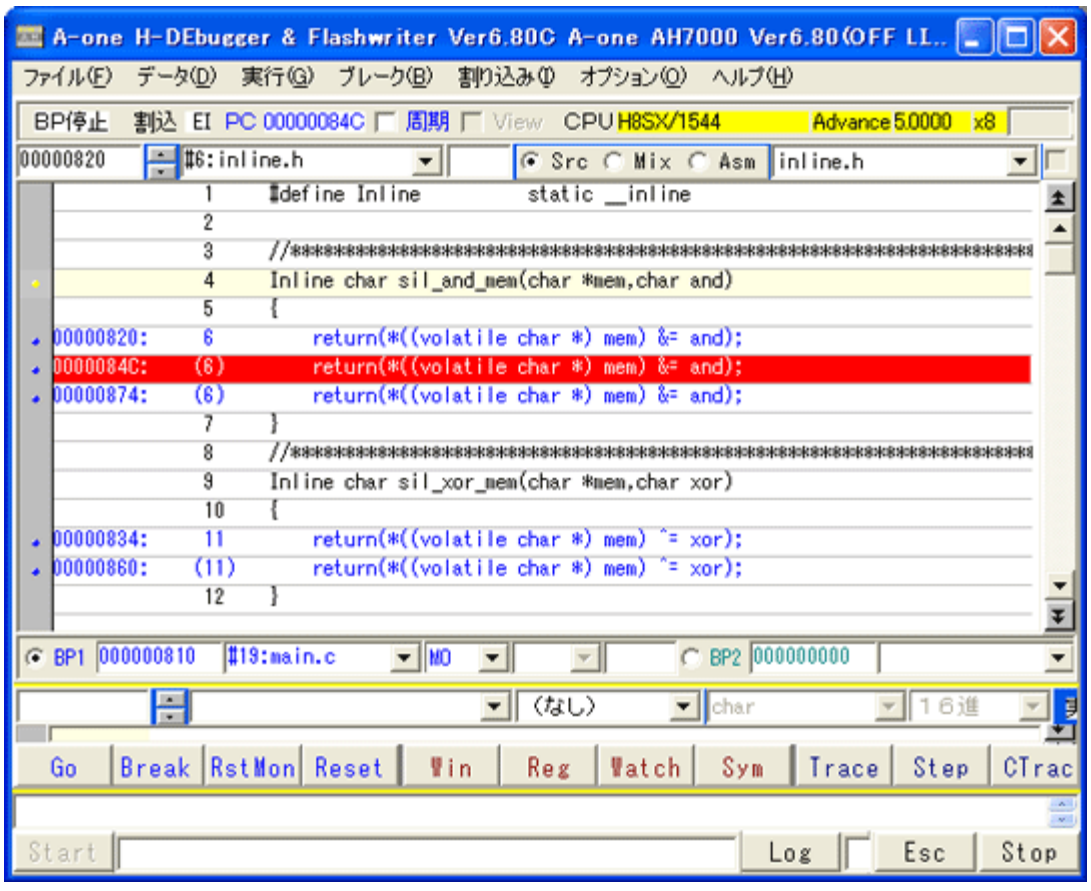
[1-2-6]　上記ライン情報で「inline.h」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［－m］を指定した場合】**



⚠ポイント（新機能）

重複している「inline.h」のライン６とライン１１を左図のように「inline.h」にマージして表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

[1-2-7] 上記ライン情報で「inline.h」の重複部分を CTrace で実行した時のＣＶｉｅｗ表示



⚠ポイント（新機能）

「main.c」ライン（２１）から CTrace した場合、この位置で「inline.h」ソースライン（6）の PC 値赤帯表示をします。

【ルネサス純正Ｃ－パターン３】

＜ＳＨ７２１１Ｆの例＞　対象：ＳＨ－２／Ｅ／Ａシリーズの場合

・SuperH RISC engine Tool Chain(Ver9.2.0.0)　・C/C++ Compiler(Ver9.02.00)　・Assmebler(Ver7.01.02)　・Optimizing Linkage Editor(Ver9.04.00)

[1-3-1]

[ELF/Dwarf2 .debug_line section Code]	(address) (line)	(address) (line)	
/ DW_LNS_extended(0x00) set_address(0x02) adr(0x1000)	+**** main.c		
/ DW_advence_line(0x03) op(14) line(15)			
/ DW_copy(0x01) Lines(15) Address(0x1000)	+0x00001000 15		
/ DW_special(0x26) dl(3) line(18) da(0x6) adr(0x1006)	+0x00001006 18		
/ DW_special(0x2C) dl(1) line(19) da(0x8) adr(0x100E)	+0x0000100E 19		
/ DW_special(0x1C) dl(1) line(20) da(0x4) adr(0x1012)	+0x00001012 20		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-11) line(9)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x101C)			
/ DW_copy(0x01) Lines(9) Address(0x101C)		(+0x0000101C 9)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(12) line(21)			
/ DW_advence_pc(0x02) pc(0x9) adr(0x102E)			
/ DW_copy(0x01) Lines(21) Address(0x102E)	+0x0000102E 21		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-7) line(14)			
/ DW_advence_pc(0x02) pc(0x5) adr(0x1038)			
/ DW_copy(0x01) Lines(14) Address(0x1038)		(+0x00001038 14)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(8) line(22)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x104C)			
/ DW_copy(0x01) Lines(22) Address(0x104C)	+0x0000104C 22		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-8) line(14)			
/ DW_advence_pc(0x02) pc(0x3) adr(0x1052)			
/ DW_copy(0x01) Lines(14) Address(0x1052)		(+0x00001052 14)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(9) line(23)			
/ DW_advence_pc(0x02) pc(0x9) adr(0x1064)			
/ DW_copy(0x01) Lines(23) Address(0x1064)	+0x00001064 23		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-9) line(14)			
/ DW_advence_pc(0x02) pc(0x3) adr(0x106A)			
/ DW_copy(0x01) Lines(14) Address(0x106A)		(+0x0000106A 14)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(10) line(24)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x107E)			
/ DW_copy(0x01) Lines(24) Address(0x107E)	+0x0000107E 24		
/ DW_set_file(0x04) op(0x2)		(+**** inline.h)	
/ DW_advence_line(0x03) op(-15) line(9)			
/ DW_advence_pc(0x02) pc(0x3) adr(0x1084)			
/ DW_copy(0x01) Lines(9) Address(0x1084)		(+0x00001084 9)	*
/ DW_set_file(0x04) op(0x1)	+**** main.c		
/ DW_advence_line(0x03) op(9) line(18)			
/ DW_advence_pc(0x02) pc(0x9) adr(0x1096)			
/ DW_copy(0x01) Lines(18) Address(0x1096)	+0x00001096 18		
/ DW_advence_line(0x03) op(8) line(26)			
/ DW_advence_pc(0x02) pc(0x7) adr(0x10A4)			
/ DW_copy(0x01) Lines(26) Address(0x10A4)	+0x000010A4 26		
/ DW_advence_pc(0x02) pc(0x4) adr(0x10AC)	+0x000010AC 27		

<main.c>
ELF/Dwarf2 のライン情報

⚠ ここのライン情報は、本来（9）であるべきです。

【注記】

・上記のように本来のライン情報でない為、**現ツールチェインバージョンではＳＨ－２シリーズの「Ｉｎｌｉｎｅ関数」対応は出来ない**と判断します。

【対応策】[1-3-2]

<inline.h>　SH-2/E/A の場合　[SuperH RISC engine Tool Chain Ver9.2.0.0]　記述例２

```
// #define Release      // debug 中はコメントにする。
#ifdef      Release
            #pragma inline(sil_and_mem)
            #pragma inline(sil_xor_mem)
#endif
//************************************************
static char sil_and_mem(char *mem,char and)
{
            return (*((volatile int *) mem) &= and);
}
//************************************************
static char sil_xor_mem(char *mem,char xor)
{
            return (*((volatile char *) mem) ^= xor);
}
```

← 上記の理由により、
デバッグ中は、Inline 関数定義をコメントにし「static 関数」としてコンパイルします。

[1-3-3]　「static 関数」にした場合のライン情報

[ELF/Dwarf2 .debug_line section Code]		(address) (line)	(address) (line)	
/ DW_LNS_extended(0x00)	set_address(0x02) adr(0x1000)	+**** main.c		
/ DW_set_file(0x04)	op(0x2)		(+**** inline.h)	*
/ DW_special(0x11)	dl(6) line(7) da(0x0) adr(0x1000)		(+0x00001000 7)	*
/ DW_special(0x0D)	dl(2) line(9) da(0x0) adr(0x1000)		(+0x00001000 9)	*
/ DW_special(0x4C)	dl(1) line(10) da(0x10) adr(0x1010)		(+0x00001010 10)	*
/ DW_LNS_extended(0x00)	set_address(0x02) adr(0x1012)			
/ DW_special(0x0D)	dl(2) line(12) da(0x0) adr(0x1012)		(+0x00001012 12)	*
/ DW_special(0x0D)	dl(2) line(14) da(0x0) adr(0x1012)		(+0x00001012 14)	*
/ DW_special(0x4C)	dl(1) line(15) da(0x10) adr(0x1022)		(+0x00001022 15)	*
/ DW_LNS_extended(0x00)	set_address(0x02) adr(0x1024)			
/ DW_set_file(0x04)	op(0x1)	+**** main.c		
/ DW_special(0x1E)	dl(3) line(18) da(0x4) adr(0x1028)	+0x00001028 18		*
/ DW_special(0x2C)	dl(1) line(19) da(0x8) adr(0x1030)	+0x00001030 19		
/ DW_special(0x24)	dl(1) line(20) da(0x6) adr(0x1036)	+0x00001036 20		
/ DW_special(0x6C)	dl(1) line(21) da(0x18) adr(0x104E)	+0x0000104E 21		
/ DW_special(0x74)	dl(1) line(22) da(0x1A) adr(0x1068)	+0x00001068 22		
/ DW_special(0x5C)	dl(1) line(23) da(0x14) adr(0x107C)	+0x0000107C 23		
/ DW_special(0x64)	dl(1) line(24) da(0x16) adr(0x1092)	+0x00001092 24		
/ DW_advence_line(0x03)	op(-6) line(18)			
/ DW_advence_pc(0x02)	pc(0xa) adr(0x10A6)			
/ DW_copy(0x01)	Lines(18) Address(0x10A6)	<+0x000010A6 18>		*
/ DW_advence_line(0x03)	op(8) line(26)			
/ DW_advence_pc(0x02)	pc(0x6) adr(0x10B2)			
/ DW_copy(0x01)	Lines(26) Address(0x10B2)	+0x000010B2 26		
/ DW_advence_pc(0x02)	pc(0x3) adr(0x10B8)	+0x000010B8 27		

<main.c>
ELF/Dwarf2 のライン情報

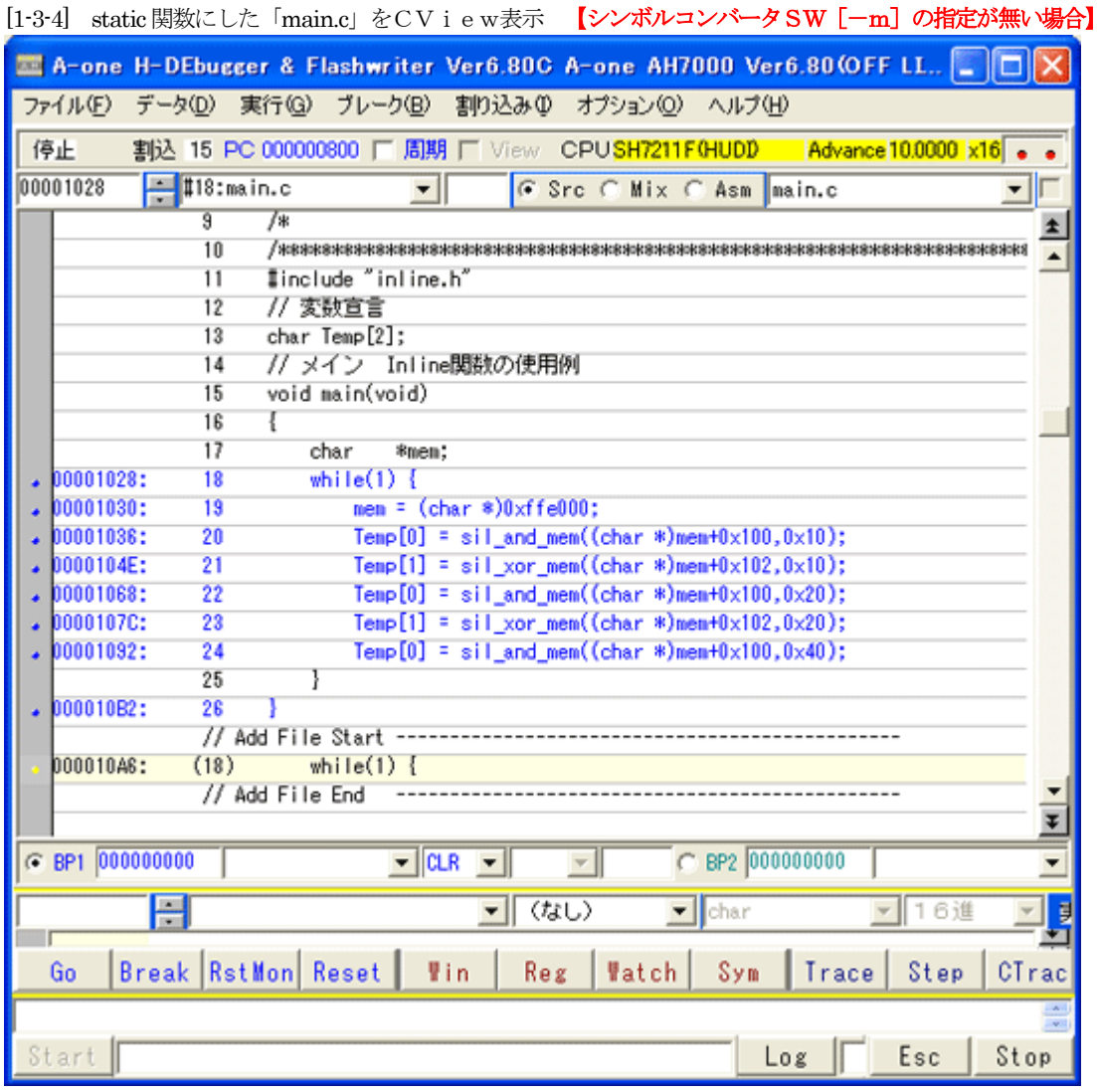
⚠新機能

DEF バージョン６．８０Ｂより、モジュール内にインクルードされたファイルにアドレスが存在した場合は、追加モジュールとして追加します。

⚠ポイント（新機能）

同じライン情報が出現した場合は、Inline 関数時と同じように追加(Add File)します。

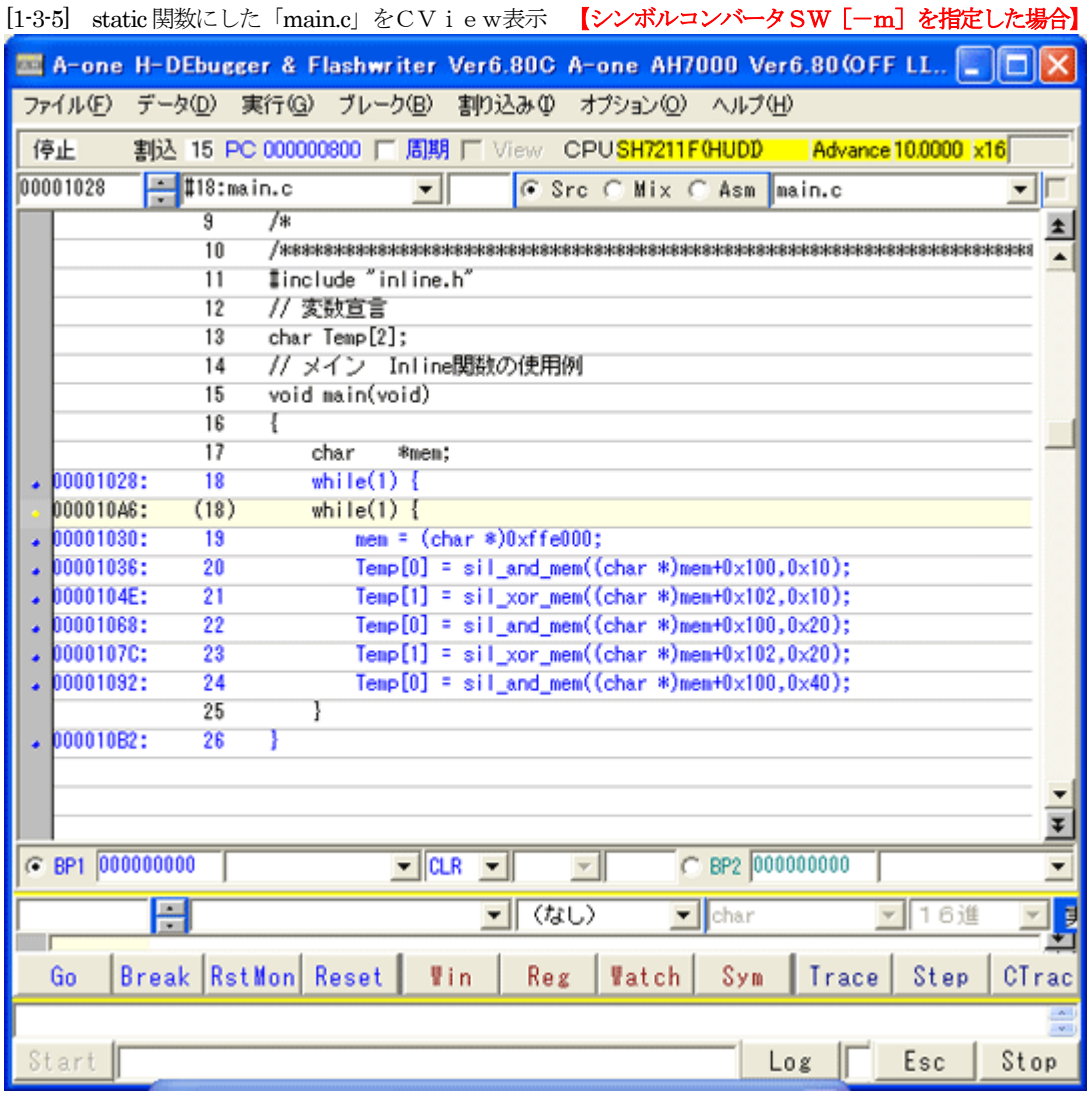
[1-3-4]　static 関数にした「main.c」をＣＶｉｅｗ表示　**【シンボルコンバータＳＷ［－m］の指定が無い場合】**



⚠ポイント（新機能）

重複しているリストライン１８を左図のように追加(Add File)して表示します。
（ｎｎ）内の数字は、同ファイルのソースラインを示します。

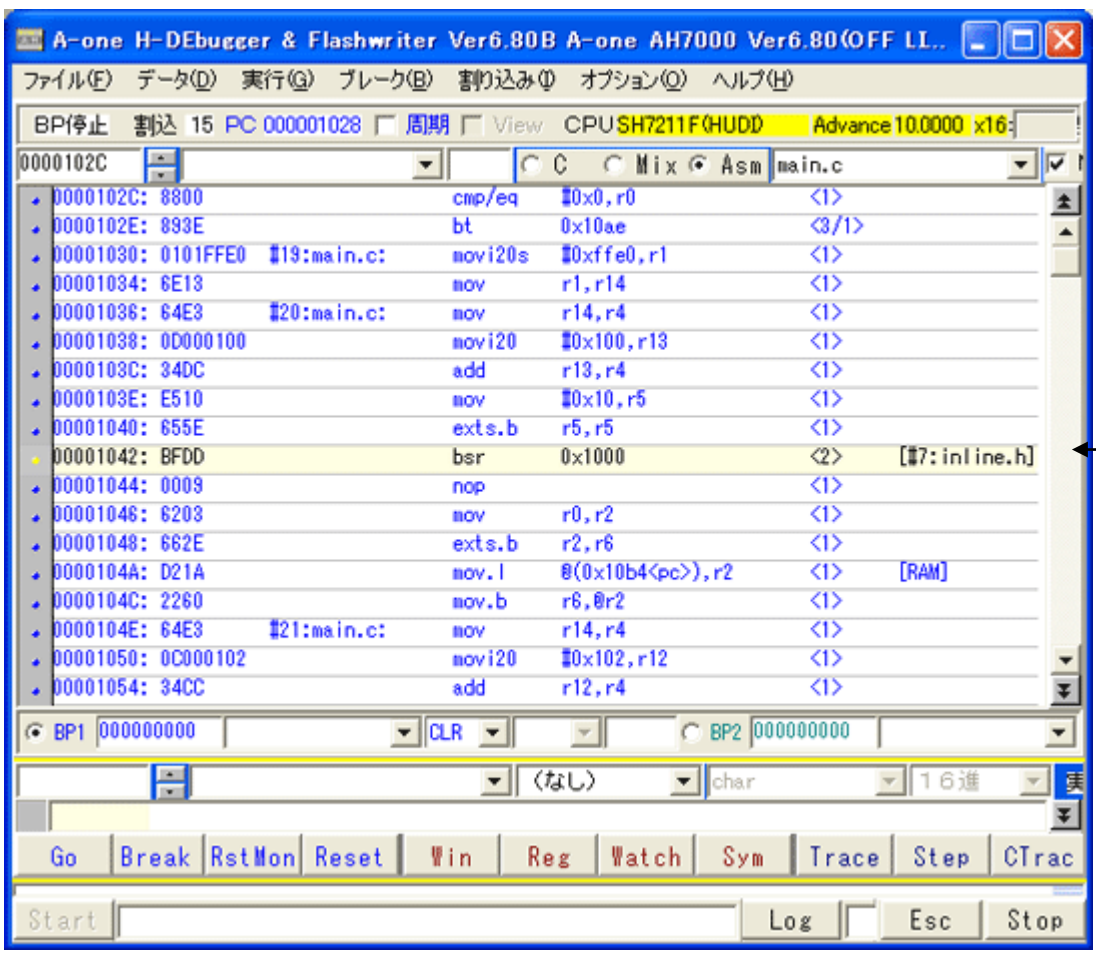
[1-3-5]　static 関数にした「main.c」をＣＶｉｅｗ表示　**【シンボルコンバータＳＷ［－m］を指定した場合】**



⚠ポイント（新機能）

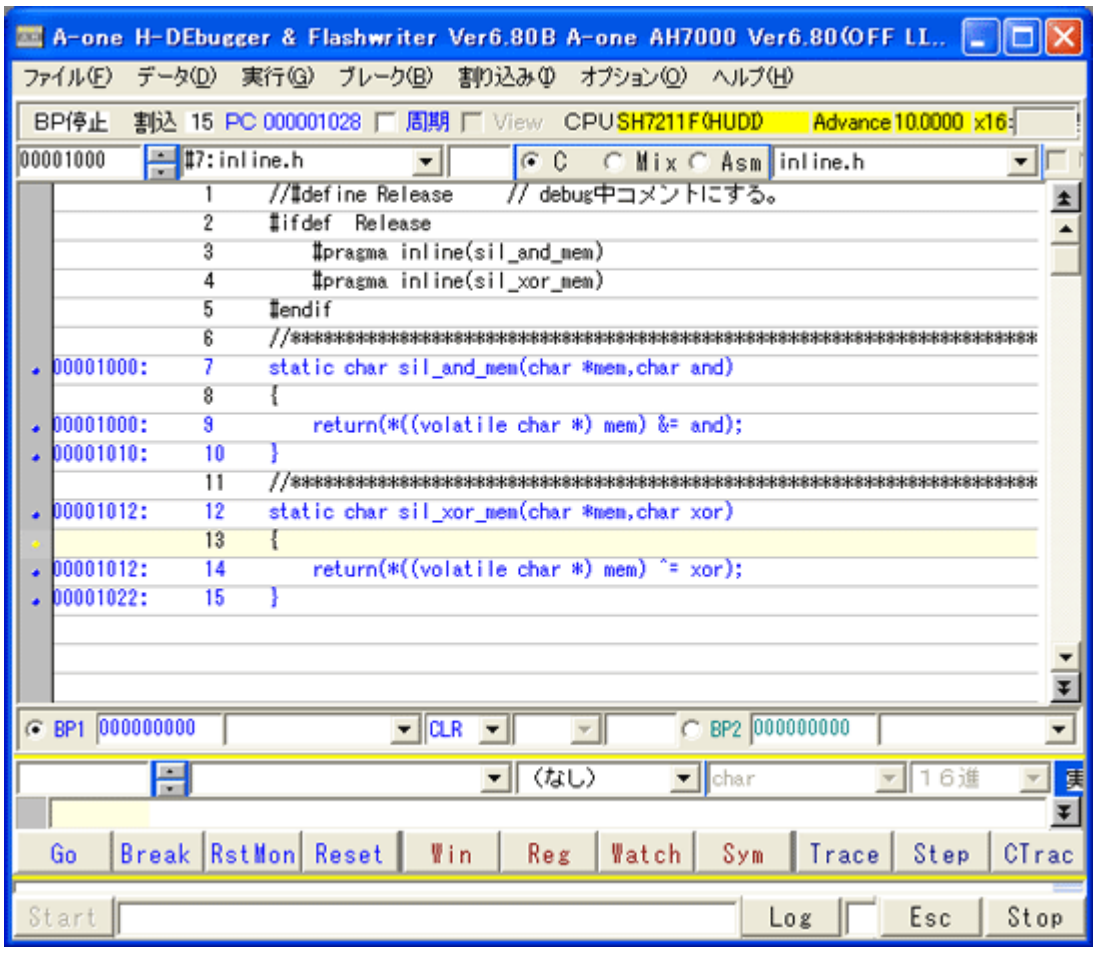
重複しているリストライン１８を左図のようにマージして表示します。
（ｎｎ）内の数字は、同ファイルのソースラインを示します。

[1-3-6]　static 関数にした「main.c」のアセンブラ表示

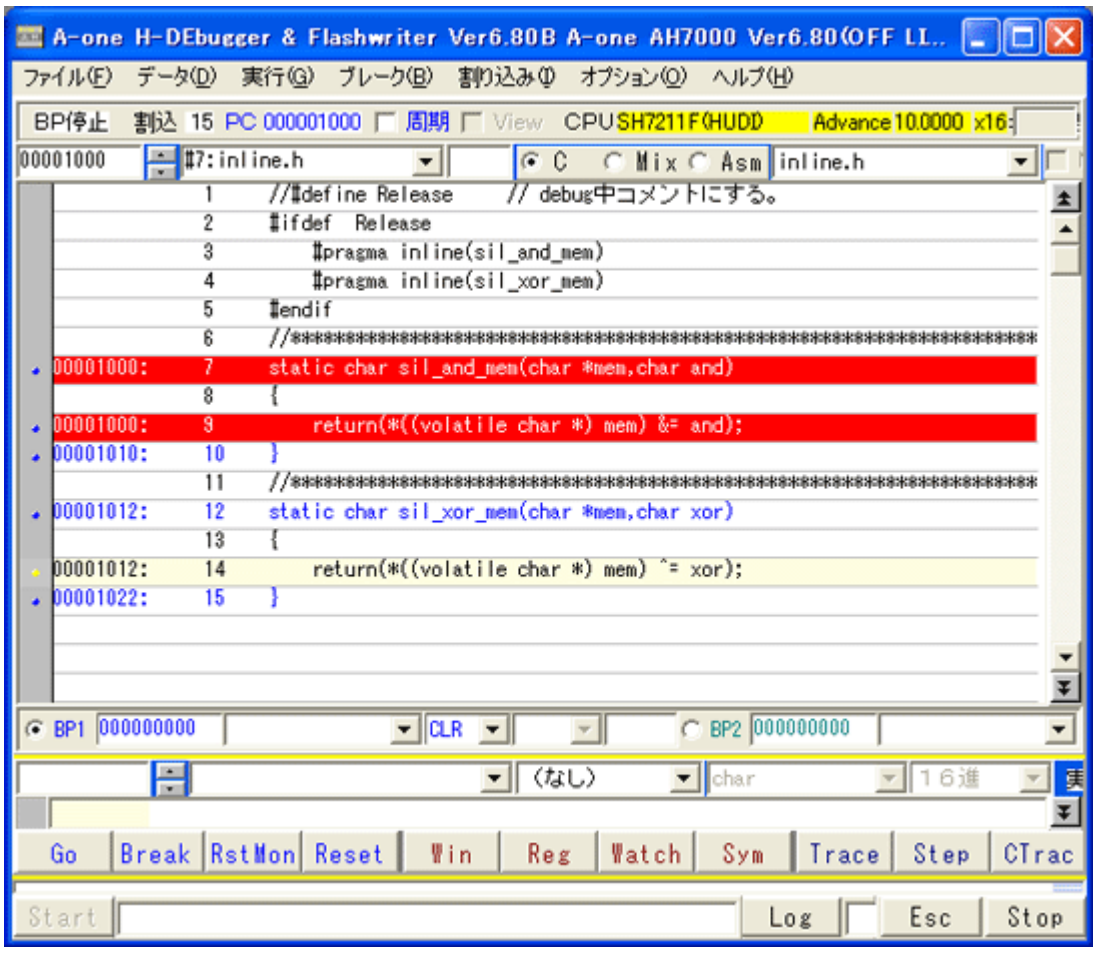


「static 関数」のコードになっていることが確認できます。

[1-3-7]　static 関数にしたことにより「inline.h」をモジュール追加したＣＶｉｅｗ表示 **【シンボルコンバータＳＷ［－m］の指定有無共通】**



[1-3-8]　「main.c」より CTrace 実行した時のＣＶｉｅｗ表示



「main.c」ライン（２０）から CTrace した場合、
「inline.h」にモジュールチェンジされます。

【２．宣言およびC記述例】Inline 関数を同じモジュール内で宣言して使用した場合

<main.c>　　H8/300,H8S,H8SX の場合　[H8S,H8/300 Tool Chain Ver6.2.0.0]　記述例１

```
#define Inline              static __inline

//*********************************************************************/
Inline char sil_and_mem(char *mem,char and)
{
        return(*((volatile char *) mem) &= and);   // line(16)
}
//*********************************************************************/
Inline char sil_xor_mem(char *mem,char xor)
{
        return(*((volatile char *) mem) ^= xor);   // line(21)
}
//*********************************************************************/
// 変数宣言
char Temp[2];
// メイン　Inline 関数の使用例
void main(void)
{
        char        *mem;
        while(1) {  // line(30)
                mem = (char *)0xffe000;   // line(31)
                Temp[0] = sil_and_mem((char *)mem+0x100,0x10); // line(32)
                Temp[1] = sil_xor_mem((char *)mem+0x100,0x20); // line(33)
                Temp[0] = sil_and_mem((char *)mem+0x102,0x10);// line(34)
                Temp[1] = sil_xor_mem((char *)mem+0x102,0x20);// line(35)
                Temp[0] = sil_and_mem((char *)mem+0x100,0x40);// line(36)
        }
}
```

[2-0-1]
同モジュール内に Inline 関数を宣言したソース例１
【対象】　H8/300x, H8S, H8SX の全品種

<main.c>　　SH-2/E/A の場合　[SuperH RISC engine Tool Chain Ver9.2.0.0]　記述例２

```
#define Release       // debug 中コメントにする。
#ifdef      Release
        #pragma inline(sil_and_mem)
        #pragma inline(sil_xor_mem)
#endif
//*********************************************************************/
static char sil_and_mem(char *mem,char and)
{
        return(*((volatile char *) mem) &= and);// line(13)
}
//*********************************************************************/
static char sil_xor_mem(char *mem,char xor)
{
        return(*((volatile char *) mem) ^= xor);// line(24)
}
// 変数宣言
char Temp[2];
// メイン　Inline 関数の使用例
void main(void)
{
        char        *mem;
        while(1) {// line(32)
                mem = (char *)0xffe000; // line(33)
                Temp[0] = sil_and_mem((char *)mem+0x100,0x10);// line(34)
                Temp[1] = sil_xor_mem((char *)mem+0x100,0x20);// line(35)
                Temp[0] = sil_and_mem((char *)mem+0x102,0x10);// line(36)
                Temp[1] = sil_xor_mem((char *)mem+0x102,0x20);// line(37)
                Temp[0] = sil_and_mem((char *)mem+0x100,0x40);//line(38)
        }
}
```

[2-0-2]
同モジュール内に Inline 関数を宣言したソース例２
【対象】　SH-2/SH-2E/SH-2A の全品種

【説明】

ルネサス純正Ｃの場合、ＣＰＵ品種別に Inline 関数時のライン情報(ELF/Dwarf2)出力に相違がありましたので、代表品種別で説明します。

【ルネサス純正Ｃ－パターン１】

＜Ｈ８／３６６４Ｆの例＞　対象：Ｈ８／３００，Ｈ８／３００Ｈ，Ｈ８／Ｔｉｎｙシリーズの場合

・H8S,H8/300 Tool Chain(Ver6.2.0.0)　・C/C++ Compiler(Ver6.02.00)　・Assmebler(Ver6.02.00)　・Optimizing Linkage Editor(Ver9.03.00)

[2-1-1]

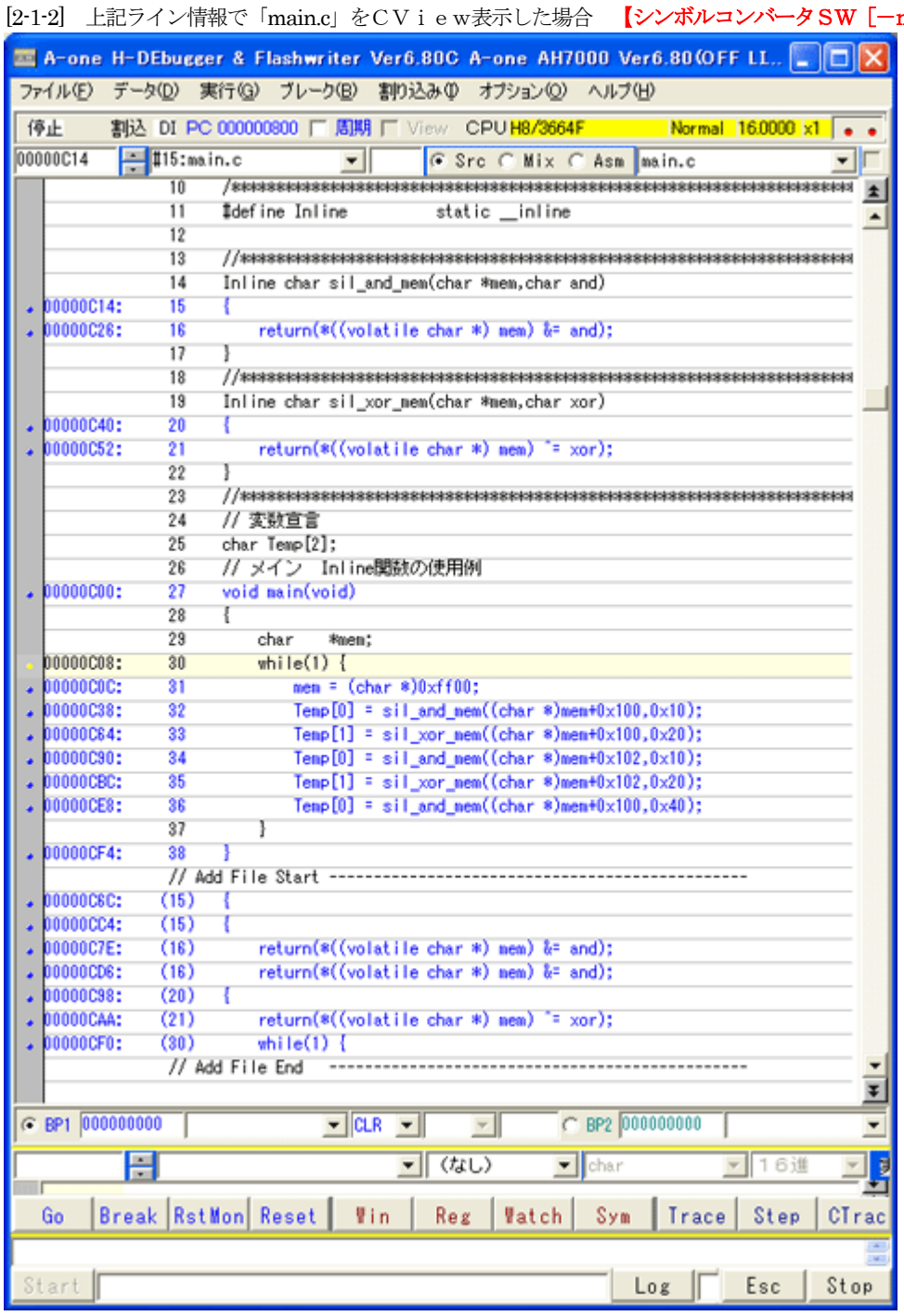
＜main.c＞
ELF/Dwarf2 のライン情報

[ELF/Dwarf2 .debug_line section Code]		(address)	(line)	(address)	(line)	
/ DW_LNS_extended(0x00)	set_address(0x02) adr(0xC00)	+**** main.c				
/ DW_advence_line(0x03)	op(26) line(27)					
/ DW_copy(0x01)	Lines(27) Address(0xC00)	+0x00000C00	27			
/ DW_special(0x2E)	dl(3) line(30) da(0x8) adr(0xC08)	+0x00000C08	30			
/ DW_special(0x1C)	dl(1) line(31) da(0x4) adr(0xC0C)	+0x00000C0C	31			
/ DW_advence_line(0x03)	op(-16) line(15)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0xC14)					
/ DW_copy(0x01)	Lines(15) Address(0xC14)	+0x00000C14	15			*
/ DW_special(0x54)	dl(1) line(16) da(0x12) adr(0xC26)	+0x00000C26	16			*
/ DW_advence_line(0x03)	op(16) line(32)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0xC38)					
/ DW_copy(0x01)	Lines(32) Address(0xC38)	+0x00000C38	32			
/ DW_advence_line(0x03)	op(-12) line(20)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0xC40)					
/ DW_copy(0x01)	Lines(20) Address(0xC40)	+0x00000C40	20			*
/ DW_special(0x54)	dl(1) line(21) da(0x12) adr(0xC52)	+0x00000C52	21			*
/ DW_advence_line(0x03)	op(12) line(33)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0xC64)					
/ DW_copy(0x01)	Lines(33) Address(0xC64)	+0x00000C64	33			
/ DW_advence_line(0x03)	op(-18) line(15)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0xC6C)					
/ DW_copy(0x01)	Lines(15) Address(0xC6C)	+0x00000C6C	15			*
/ DW_special(0x54)	dl(1) line(16) da(0x12) adr(0xC7E)	+0x00000C7E	16			*
/ DW_advence_line(0x03)	op(18) line(34)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0xC90)					
/ DW_copy(0x01)	Lines(34) Address(0xC90)	+0x00000C90	34			
/ DW_advence_line(0x03)	op(-14) line(20)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0xC98)					
/ DW_copy(0x01)	Lines(20) Address(0xC98)	+0x00000C98	20			*
/ DW_special(0x54)	dl(1) line(21) da(0x12) adr(0xCAA)	+0x00000CAA	21			*
/ DW_advence_line(0x03)	op(14) line(35)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0xCBC)					
/ DW_copy(0x01)	Lines(35) Address(0xCBC)	+0x00000CBC	35			
/ DW_advence_line(0x03)	op(-20) line(15)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0xCC4)					
/ DW_copy(0x01)	Lines(15) Address(0xCC4)	+0x00000CC4	15			*
/ DW_special(0x54)	dl(1) line(16) da(0x12) adr(0xCD6)	+0x00000CD6	16			*
/ DW_advence_line(0x03)	op(20) line(36)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0xCE8)					
/ DW_copy(0x01)	Lines(36) Address(0xCE8)	+0x00000CE8	36			
/ DW_advence_line(0x03)	op(-6) line(30)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0xCF0)					
/ DW_copy(0x01)	Lines(30) Address(0xCF0)	+0x00000CF0	30			
/ DW_advence_line(0x03)	op(8) line(38)					
/ DW_advence_pc(0x02)	pc(0x2) adr(0xCF4)					
/ DW_copy(0x01)	Lines(38) Address(0xCF4)	+0x00000CF4	38			
/ DW_advence_pc(0x02)	pc(0x2) adr(0xCF8)	+0x00000CF8	39			

Inline 関数の場合、
＊のように多重情報になる。
つまり、１行のソースラインに複数アドレスが存在します。

＊も同　上

[2-1-2]　上記ライン情報で「main.c」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［－m］の指定が無い場合】**



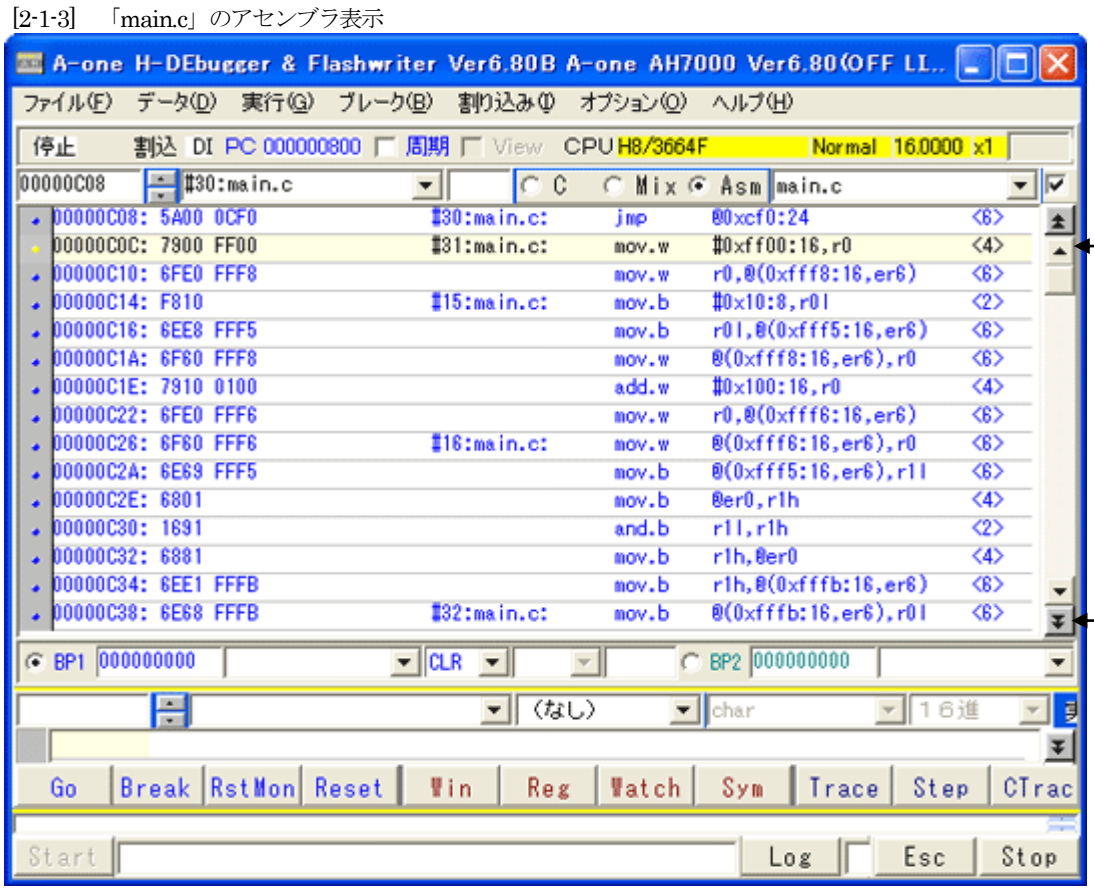
⚠注意点（1）

上記ライン情報から判るように、「main.c」のライン（３１）とライン（３２）の間にインライン関数のライン（１５）～（１６）のコードが生成されています。
つまり、ライン（３２）の時点で、既にインライン関数の処理は終了していることになります。
若干イメージと違いますので注意して下さい。

⚠ポイント（新機能）

重複しているのライン１５～１６とライン２０～２１を左図のように追加(Add File)して表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

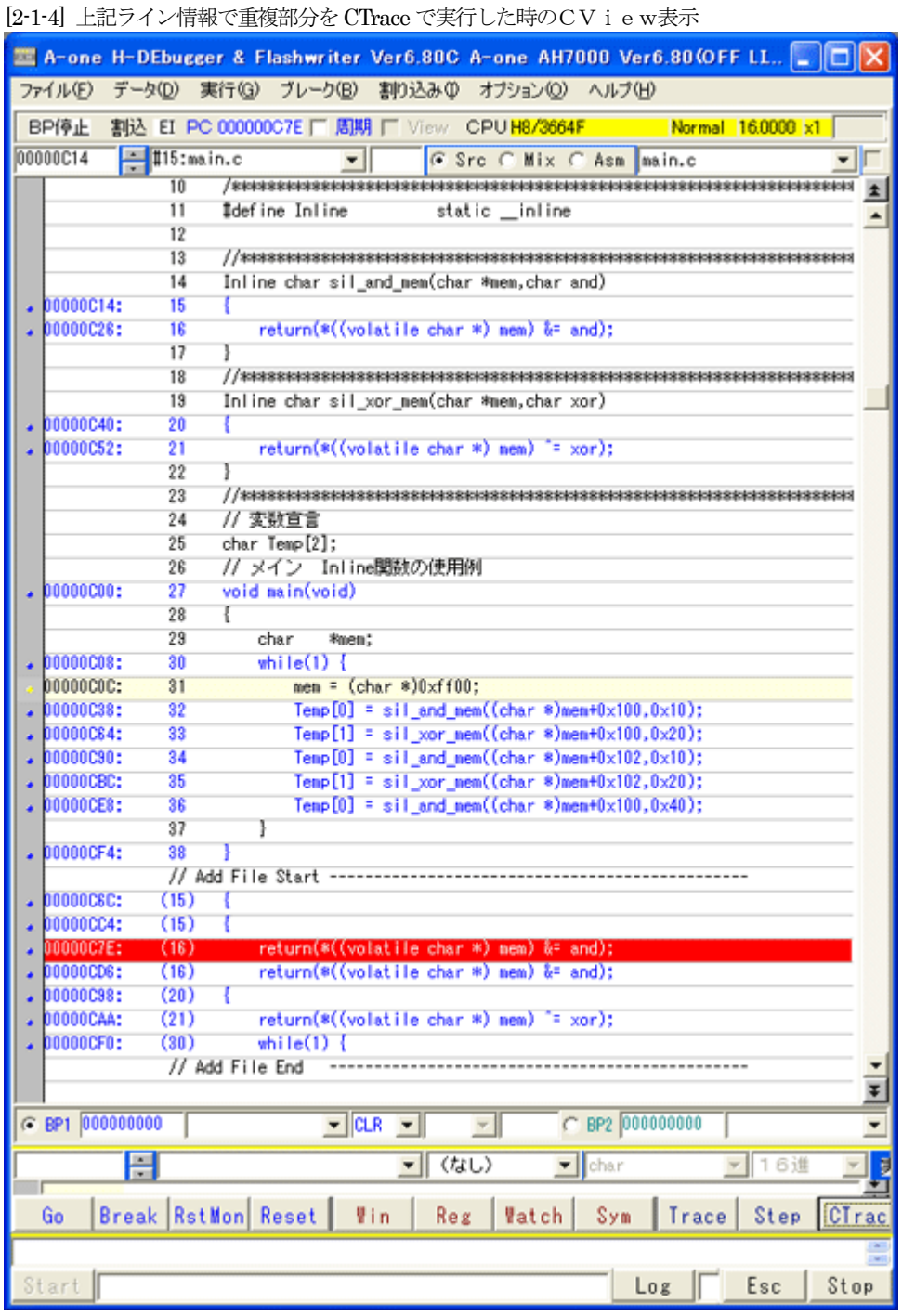
[2-1-3]　「main.c」のアセンブラ表示



⚠注意点（2）

「main.c」のライン（３１）と（３２）の間にインライン関数のライン（１５）～（１６）のコードが入っているのが確認できます。

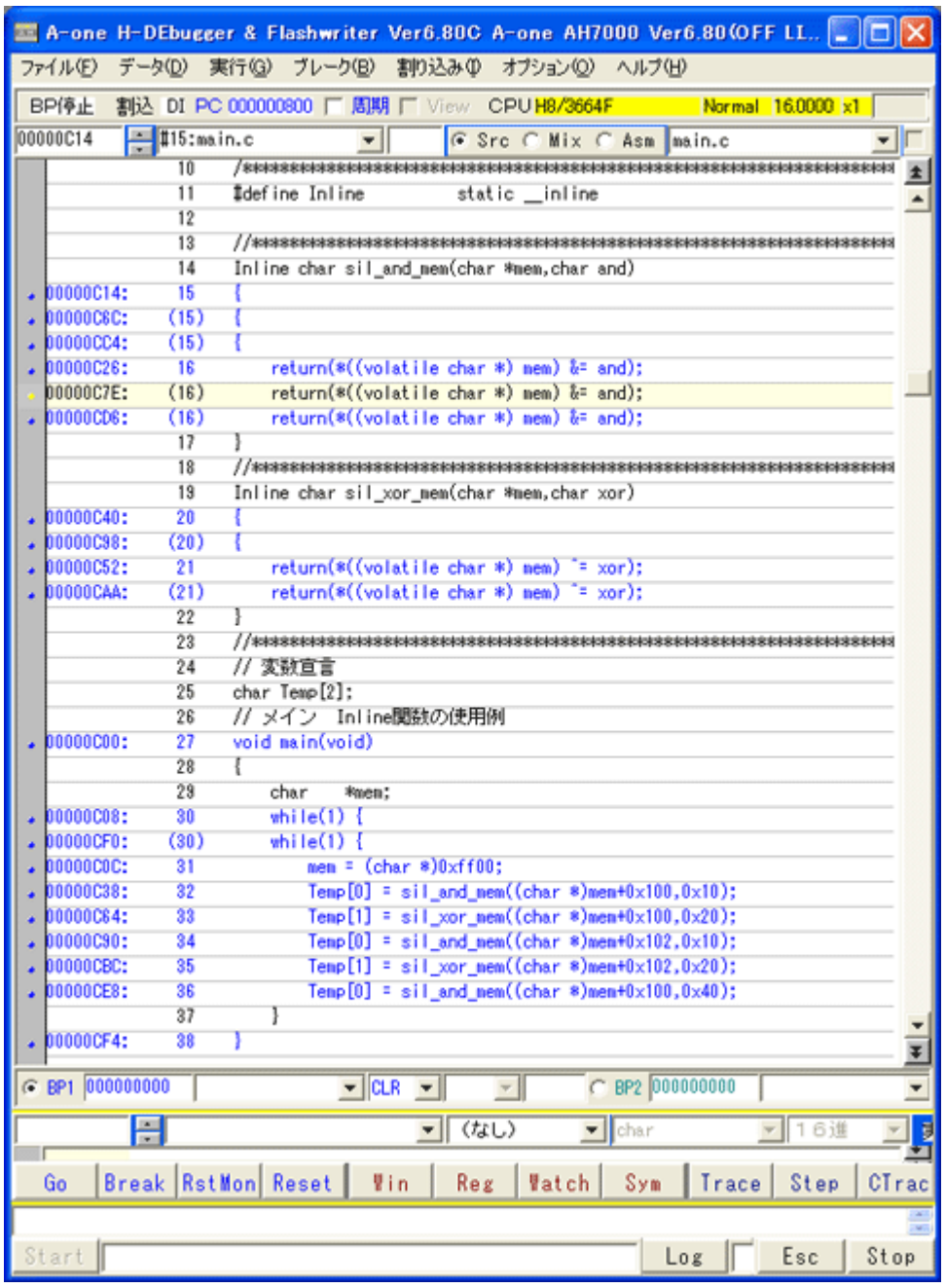
[2-1-4] 上記ライン情報で重複部分を CTrace で実行した時のＣＶｉｅｗ表示



⚠ポイント（新機能）

「main.c」ライン（３３）から CTrace した場合、
この位置でソースライン（１６）の PC 値赤帯表示をします。

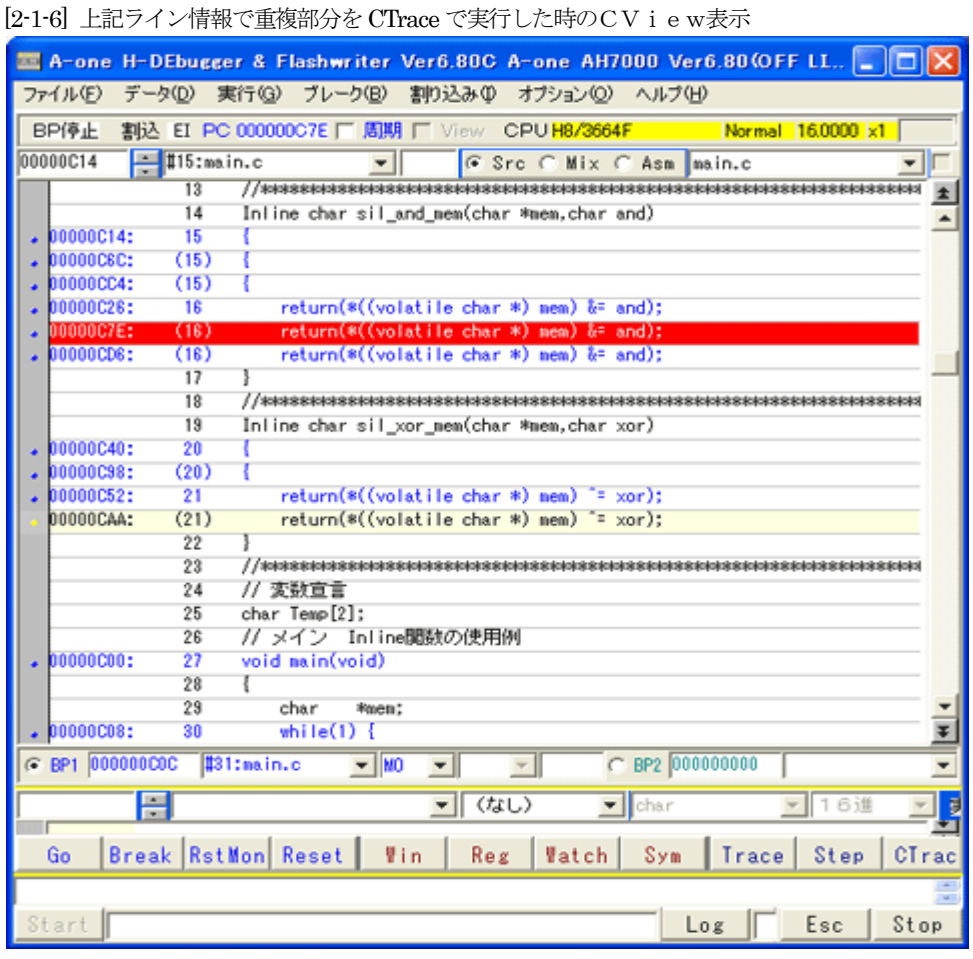
[2-1-5]　上記ライン情報で「main.c」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［－m］を指定した場合】**



⚠ポイント（新機能）

重複しているのライン１５～１６とライン２０～２１を左図のようにマージして表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

[2-1-6]　上記ライン情報で重複部分を CTrace で実行した時のＣＶｉｅｗ表示



⚠ポイント（新機能）

「main.c」ライン（３３）から CTrace した場合、
この位置でソースライン（１６）の PC 値赤帯表示をします。

【ルネサス純正Cーパターン２】

＜Ｈ８Ｓ／２６３８Ｆの例＞　対象：Ｈ８Ｓ，Ｈ８ＳＸシリーズの場合

・H8S,H8/300 Tool Chain(Ver6.2.0.0)　・C/C++ Compiler(Ver6.02.00)　・Assmebler(Ver6.02.00)　・Optimizing Linkage Editor(Ver9.03.00)

[2-2-1]

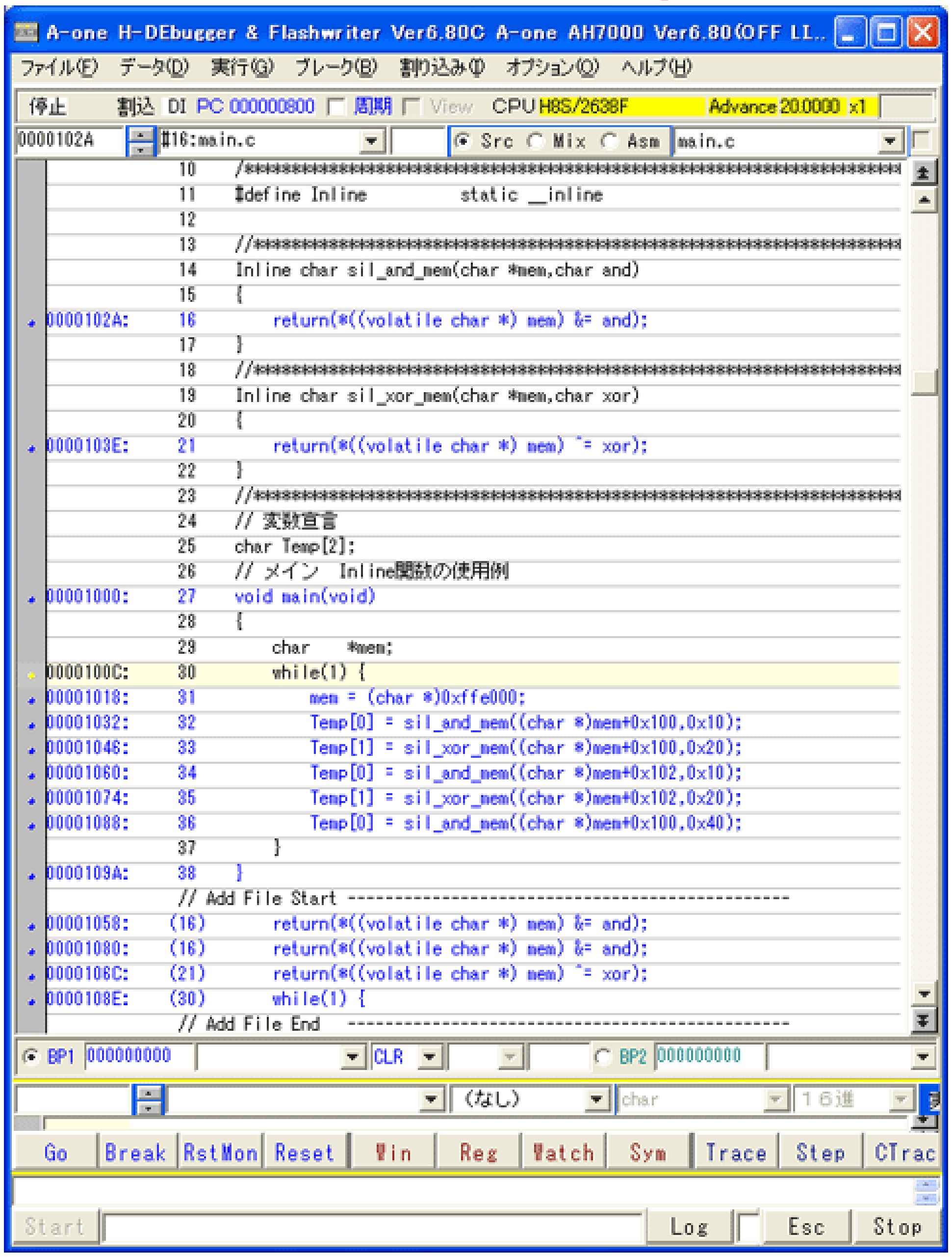
[ELF/Dwarf2 .debug_line section Code]	(address)	(line)	(address)	(line)	
/ DW_LNS_extended(0x00) set_address(0x02) adr(0x1000)	+**** main.c				
/ DW_advence_line(0x03) op(26) line(27)					
/ DW_copy(0x01) Lines(27) Address(0x1000)	+0x00001000	27			
/ DW_special(0x3E) dl(3) line(30) da(0xC) adr(0x100C)	+0x0000100C	30			
/ DW_special(0x3C) dl(1) line(31) da(0xC) adr(0x1018)	+0x00001018	31			
/ DW_advence_line(0x03) op(-15) line(16)					
/ DW_advence_pc(0x02) pc(0x9) adr(0x102A)					
/ DW_copy(0x01) Lines(16) Address(0x102A)	+0x0000102A	16			*
/ DW_advence_line(0x03) op(16) line(32)					
/ DW_advence_pc(0x02) pc(0x4) adr(0x1032)					
/ DW_copy(0x01) Lines(32) Address(0x1032)	+0x00001032	32			
/ DW_advence_line(0x03) op(-11) line(21)					
/ DW_advence_pc(0x02) pc(0x6) adr(0x103E)					
/ DW_copy(0x01) Lines(21) Address(0x103E)	+0x0000103E	21			*
/ DW_advence_line(0x03) op(12) line(33)					
/ DW_advence_pc(0x02) pc(0x4) adr(0x1046)					
/ DW_copy(0x01) Lines(33) Address(0x1046)	+0x00001046	33			
/ DW_advence_line(0x03) op(-17) line(16)					
/ DW_advence_pc(0x02) pc(0x9) adr(0x1058)					
/ DW_copy(0x01) Lines(16) Address(0x1058)	+0x00001058	16			*
/ DW_advence_line(0x03) op(18) line(34)					
/ DW_advence_pc(0x02) pc(0x4) adr(0x1060)					
/ DW_copy(0x01) Lines(34) Address(0x1060)	+0x00001060	34			
/ DW_advence_line(0x03) op(-13) line(21)					
/ DW_advence_pc(0x02) pc(0x6) adr(0x106C)					
/ DW_copy(0x01) Lines(21) Address(0x106C)	+0x0000106C	21			*
/ DW_advence_line(0x03) op(14) line(35)					
/ DW_advence_pc(0x02) pc(0x4) adr(0x1074)					
/ DW_copy(0x01) Lines(35) Address(0x1074)	+0x00001074	35			
/ DW_advence_line(0x03) op(-19) line(16)					
/ DW_advence_pc(0x02) pc(0x6) adr(0x1080)					
/ DW_copy(0x01) Lines(16) Address(0x1080)	+0x00001080	16			*
/ DW_advence_line(0x03) op(20) line(36)					
/ DW_advence_pc(0x02) pc(0x4) adr(0x1088)					
/ DW_copy(0x01) Lines(36) Address(0x1088)	+0x00001088	36			
/ DW_advence_line(0x03) op(-6) line(30)					
/ DW_advence_pc(0x02) pc(0x3) adr(0x108E)					
/ DW_copy(0x01) Lines(30) Address(0x108E)	+0x0000108E	30			
/ DW_advence_line(0x03) op(8) line(38)					
/ DW_advence_pc(0x02) pc(0x6) adr(0x109A)					
/ DW_copy(0x01) Lines(38) Address(0x109A)	+0x0000109A	38			
/ DW_advence_pc(0x02) pc(0x7) adr(0x10A8)	+0x000010A8	39			

<main.c>
ELF/Dwarf2 のライン情報

Inline 関数の場合、
＊のように多重情報になる。
つまり、１行のソースライン
に複数アドレスが存在し
ます。

＊も同上

[2-2-2]　上記ライン情報で「main.c」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［−m］の指定が無い場合】**



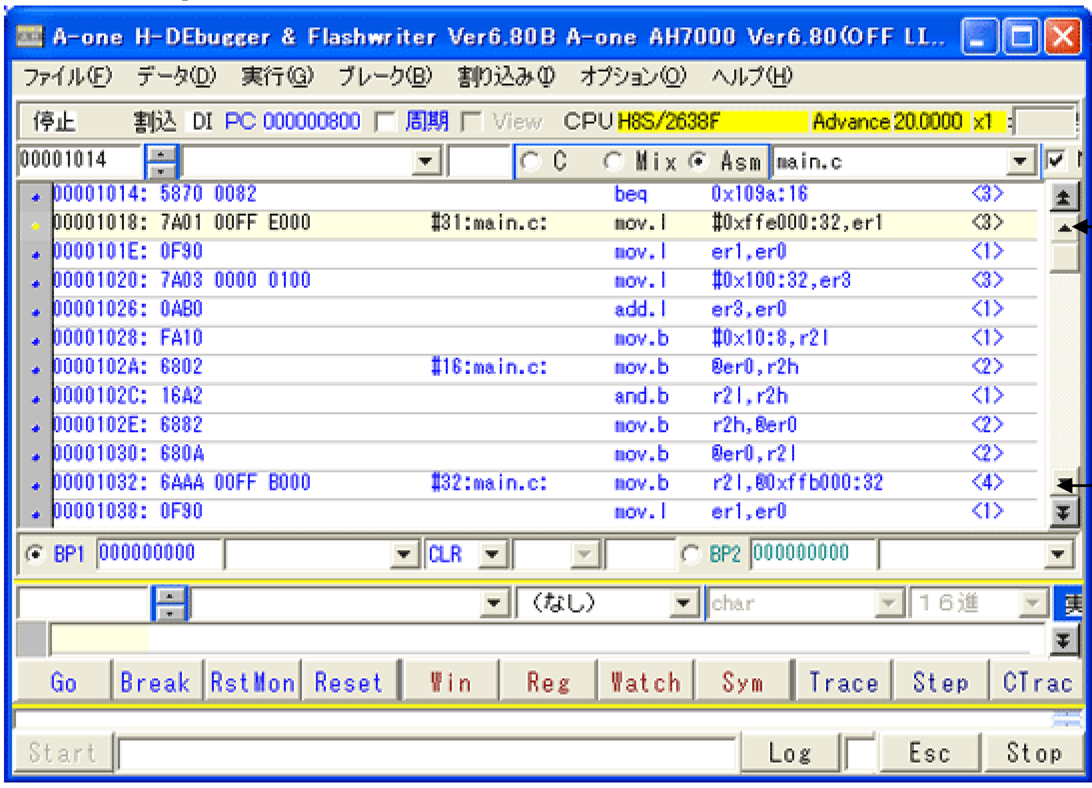
注意点（１）

上記ライン情報から判るように、「main.c」のライン（３１）とライン（３２）の間にインライン関数のライン（１６）のコードが生成されています。
つまり、ライン（３２）の時点で、既にインライン関数の処理は終了していることになります。
若干イメージと違いますので注意して下さい。

ポイント（新機能）

重複しているのライン１６とライン２１を左図のように追加(Add File)して表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

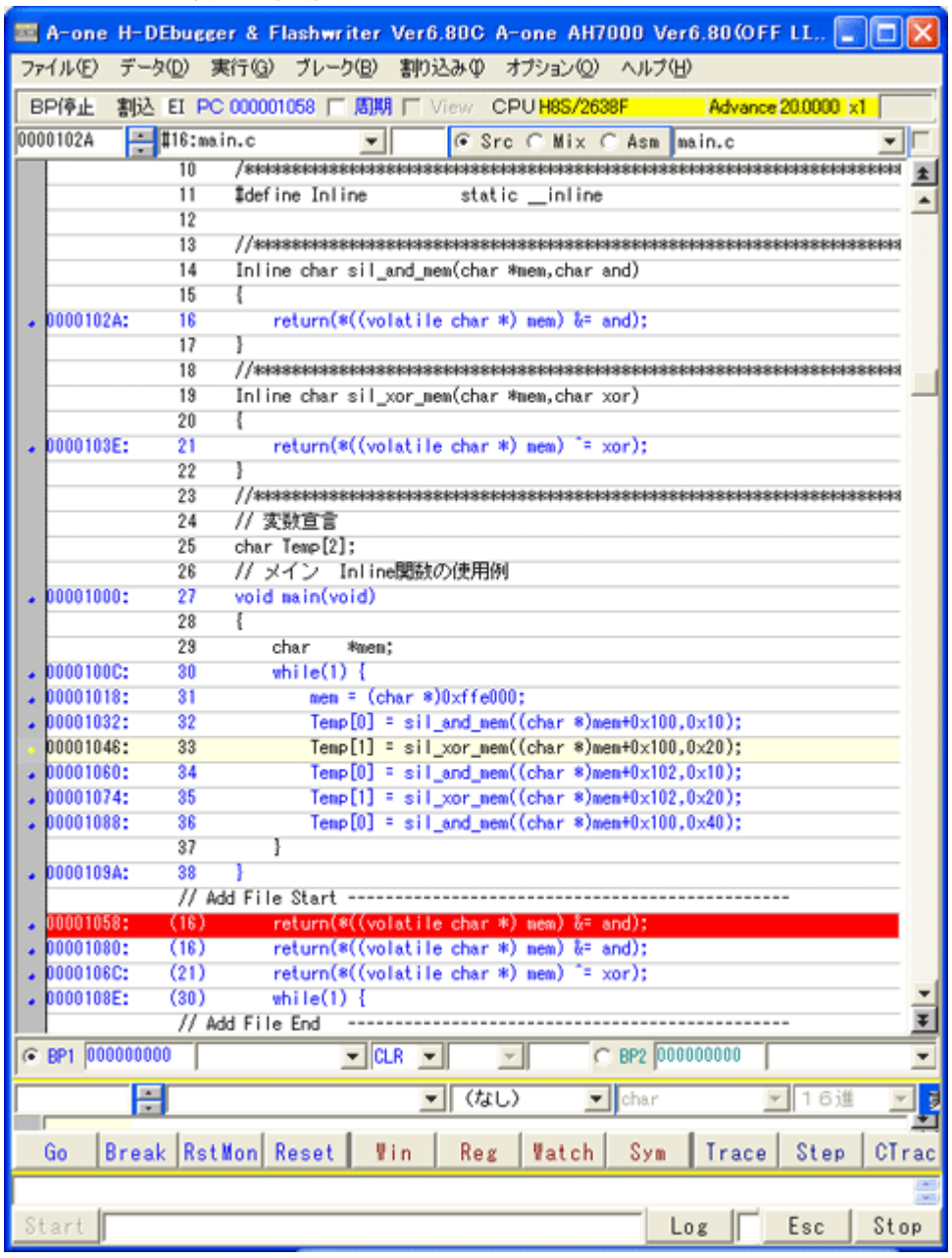
[2-2-3]　「main.c」のアセンブラ表示



注意点（２）

「main.c」のライン（３１）と（３２）の間にインライン関数のライン（１６）のコードが入っているのが確認できます。

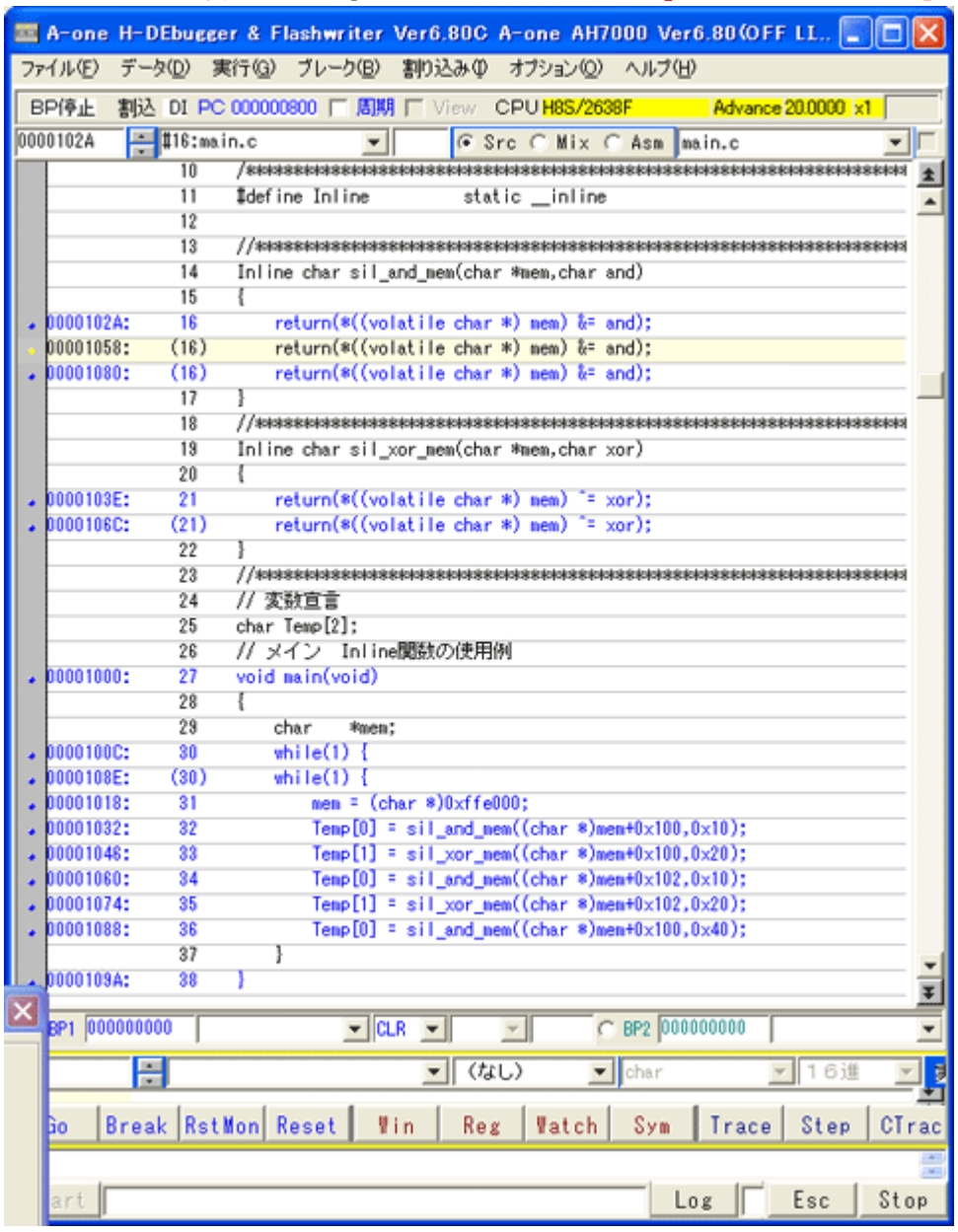
[2-2-4] 上記ライン情報で重複部分を CTrace で実行した時のＣＶｉｅｗ表示



⚠ポイント（新機能）

「main.c」ライン（３３）から CTrace した場合、
この位置でソースライン（１６）の PC 値赤帯表示をします。

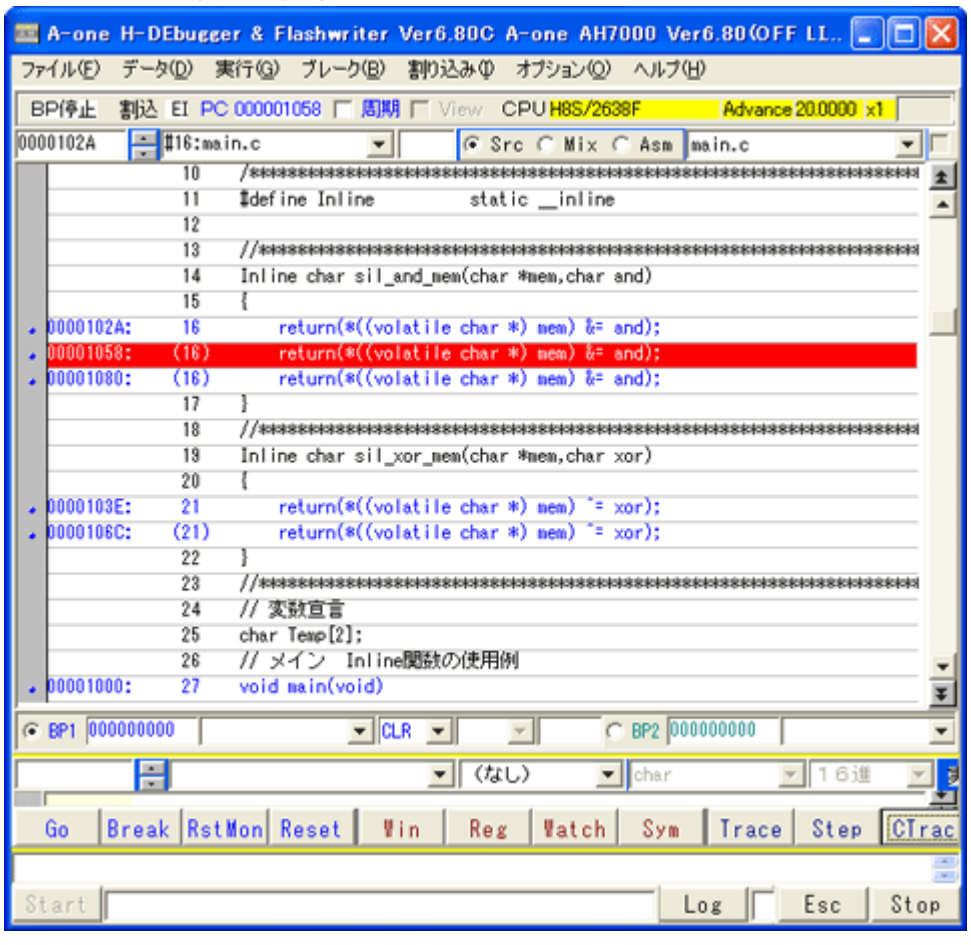
[2-2-5]　上記ライン情報で「main.c」をＣＶｉｅｗ表示した場合　**【シンボルコンバータＳＷ［－m］を指定した場合】**



⚠ポイント（新機能）

重複しているのライン１６とライン２１を左図のようにマージして表示します。
（ｎｎ）内の数字は、元ファイルのソースラインを示します。

[2-2-6] 上記ライン情報で重複部分を CTrace で実行した時のＣＶｉｅｗ表示



⚠ポイント（新機能）

「main.c」ライン（３３）から CTrace した場合、
この位置でソースライン（１６）の PC 値赤帯表示をします。

【ルネサス純正Ｃ－パターン３】

＜ＳＨ７０５１Ｆの例＞　対象：ＳＨ－２／Ｅ／Ａシリーズの場合

・SuperH RISC engine Tool Chain(Ver9.2.0.0)　・C/C++ Compiler(Ver9.02.00)　・Assmebler(Ver7.01.02)　・Optimizing Linkage Editor(Ver9.04.00)

[2-3-1]

[ELF/Dwarf2 .debug_line section Code]		(address)	(line)	(address)	(line)	
/ DW_LNS_extended(0x00)	set_address(0x02) adr(0x1800)	+**** main.c				
/ DW_advence_line(0x03)	op(28) line(29)					
/ DW_copy(0x01)	Lines(29) Address(0x1800)	+0x00001800	29			
/ DW_special(0x26)	dl(3) line(32) da(0x6) adr(0x1806)	+0x00001806	32			
/ DW_special(0x2C)	dl(1) line(33) da(0x8) adr(0x180E)	+0x0000180E	33			
/ DW_special(0x14)	dl(1) line(34) da(0x2) adr(0x1810)	+0x00001810	34			
/ DW_advence_line(0x03)	op(-15) line(19)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0x1818)					
/ DW_copy(0x01)	Lines(19) Address(0x1818)	+0x00001818	19			*
/ DW_advence_line(0x03)	op(16) line(35)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0x182A)					
/ DW_copy(0x01)	Lines(35) Address(0x182A)	+0x0000182A	35			
/ DW_advence_line(0x03)	op(-11) line(24)					
/ DW_advence_pc(0x02)	pc(0x3) adr(0x1830)					
/ DW_copy(0x01)	Lines(24) Address(0x1830)	+0x00001830	24			*
/ DW_advence_line(0x03)	op(12) line(36)					
/ DW_advence_pc(0x02)	pc(0xa) adr(0x1844)					
/ DW_copy(0x01)	Lines(36) Address(0x1844)	+0x00001844	36			
/ DW_advence_line(0x03)	op(-12) line(24)					
/ DW_advence_pc(0x02)	pc(0x4) adr(0x184C)					
/ DW_copy(0x01)	Lines(24) Address(0x184C)	+0x0000184C	24			*
/ DW_advence_line(0x03)	op(13) line(37)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0x185E)					
/ DW_copy(0x01)	Lines(37) Address(0x185E)	+0x0000185E	37			
/ DW_advence_line(0x03)	op(-13) line(24)					
/ DW_advence_pc(0x02)	pc(0x3) adr(0x1864)					
/ DW_copy(0x01)	Lines(24) Address(0x1864)	+0x00001864	24			*
/ DW_advence_line(0x03)	op(14) line(38)					
/ DW_advence_pc(0x02)	pc(0xa) adr(0x1878)					
/ DW_copy(0x01)	Lines(38) Address(0x1878)	+0x00001878	38			
/ DW_advence_line(0x03)	op(-19) line(19)					
/ DW_advence_pc(0x02)	pc(0x3) adr(0x187E)					
/ DW_copy(0x01)	Lines(19) Address(0x187E)	+0x0000187E	19			*
/ DW_advence_line(0x03)	op(13) line(32)					
/ DW_advence_pc(0x02)	pc(0x9) adr(0x1890)					
/ DW_copy(0x01)	Lines(32) Address(0x1890)	+0x00001890	32			
/ DW_advence_line(0x03)	op(8) line(40)					
/ DW_advence_pc(0x02)	pc(0x7) adr(0x189E)					
/ DW_copy(0x01)	Lines(40) Address(0x189E)	+0x0000189E	40			
/ DW_advence_pc(0x02)	pc(0x9) adr(0x18B0)	+0x000018B0	41			

<main.c>
ELF/Dwarf2 のライン情報

⚠
ここのライン情報は、本来（１９）であるべきです。

【注記】

・上記のように本来のライン情報でない為、**現ツールチェインバージョンではＳＨ－２シリーズの「Ｉｎｌｉｎｅ関数」対応は出来ない**と判断します。

【対応策】[2-3-2]

<main.c>　SH-2/E/A の場合　[SuperH RISC engine Tool Chain Ver9.2.0.0]　記述例 2

```
// #define Release        // debug 中コメントにする。
#ifdef      Release
            #pragma inline(sil_and_mem)
            #pragma inline(sil_xor_mem)
#endif
//*********************************************************************/
static char sil_and_mem(char *mem,char and)
{
            return(*((volatile char *) mem) &= and);
}
//*********************************************************************/
static char sil_xor_mem(char *mem,char xor)
{
            return(*((volatile char *) mem) ^= xor);
}
// 変数宣言
char Temp[2];
// メイン　Inline 関数の使用例
void main(void)
{
            char        *mem;
            while(1) {
                        mem = (char *)0xffe000;
                        Temp[0] = sil_and_mem((char *)mem+0x100,0x10);
                        Temp[1] = sil_xor_mem((char *)mem+0x100,0x20);
                        Temp[0] = sil_and_mem((char *)mem+0x102,0x10);
                        Temp[1] = sil_xor_mem((char *)mem+0x102,0x20);
                        Temp[0] = sil_and_mem((char *)mem+0x100,0x40);
            }
}
```

← 上記の理由により、
デバッグ中は、Inline 関数定義をコメントにし「static 関数」としてコンパイルします。

[2-3-3]　「static 関数」にした場合のライン情報

[ELF/Dwarf2 .debug_line section Code]	(address) (line)	(address) (line)	
/ DW_LNS_extended(0x00) set_address(0x02) adr(0x1800)	+**** main.c		
/ DW_advence_line(0x03) op(16) line(17)			
/ DW_copy(0x01) Lines(17) Address(0x1800)	+0x00001800 17		*
/ DW_special(0x0D) dl(2) line(19) da(0x0) adr(0x1800)	+0x00001800 19		*
/ DW_special(0x54) dl(1) line(20) da(0x12) adr(0x1812)	+0x00001812 20		*
/ DW_LNS_extended(0x00) set_address(0x02) adr(0x1816)			
/ DW_special(0x0D) dl(2) line(22) da(0x0) adr(0x1816)	+0x00001816 22		*
/ DW_special(0x0D) dl(2) line(24) da(0x0) adr(0x1816)	+0x00001816 24		*
/ DW_special(0x54) dl(1) line(25) da(0x12) adr(0x1828)	+0x00001828 25		*
/ DW_LNS_extended(0x00) set_address(0x02) adr(0x182C)			
/ DW_special(0x0F) dl(4) line(29) da(0x0) adr(0x182C)	+0x0000182C 29		
/ DW_special(0x36) dl(3) line(32) da(0xA) adr(0x1836)	+0x00001836 32		*
/ DW_special(0x2C) dl(1) line(33) da(0x8) adr(0x183E)	+0x0000183E 33		
/ DW_special(0x1C) dl(1) line(34) da(0x4) adr(0x1842)	+0x00001842 34		
/ DW_special(0x64) dl(1) line(35) da(0x16) adr(0x1858)	+0x00001858 35		
/ DW_special(0x64) dl(1) line(36) da(0x16) adr(0x186E)	+0x0000186E 36		
/ DW_special(0x64) dl(1) line(37) da(0x16) adr(0x1884)	+0x00001884 37		
/ DW_special(0x64) dl(1) line(38) da(0x16) adr(0x189A)	+0x0000189A 38		
/ DW_advence_line(0x03) op(-6) line(32)			
/ DW_advence_pc(0x02) pc(0xa) adr(0x18AE)			
/ DW_copy(0x01) Lines(32) Address(0x18AE)	+0x000018AE 32		*
/ DW_advence_line(0x03) op(8) line(40)			
/ DW_advence_pc(0x02) pc(0x9) adr(0x18C0)			
/ DW_copy(0x01) Lines(40) Address(0x18C0)	+0x000018C0 40		
/ DW_advence_pc(0x02) pc(0x8) adr(0x18D0)	+0x000018D0 41		

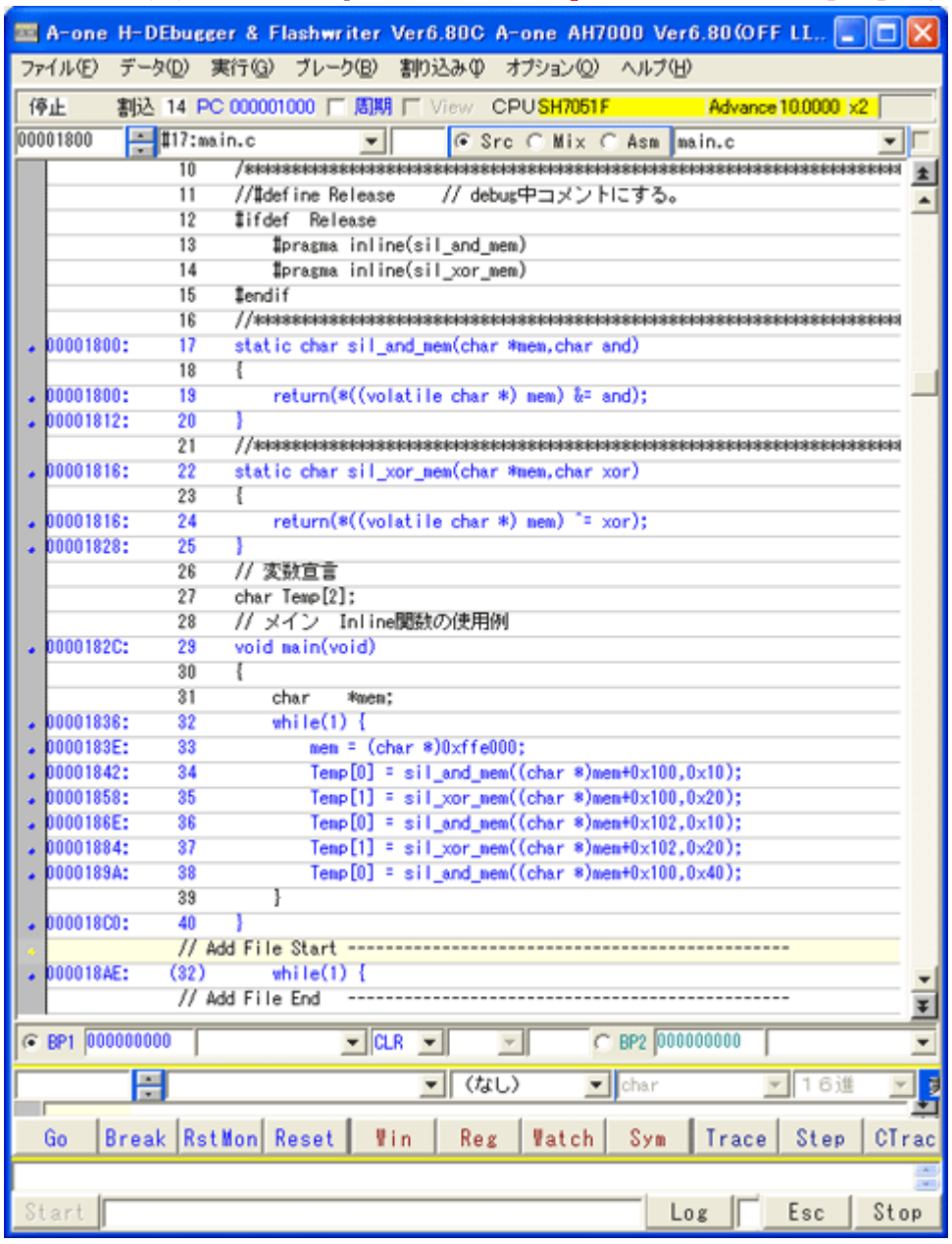
<main.c>
ELF/Dwarf2 のライン情報

static 関数としての情報です。

⚠ポイント（新機能）

同じライン情報が出現した場合は、
Inline 関数時と同じように追加
(Add File)します。

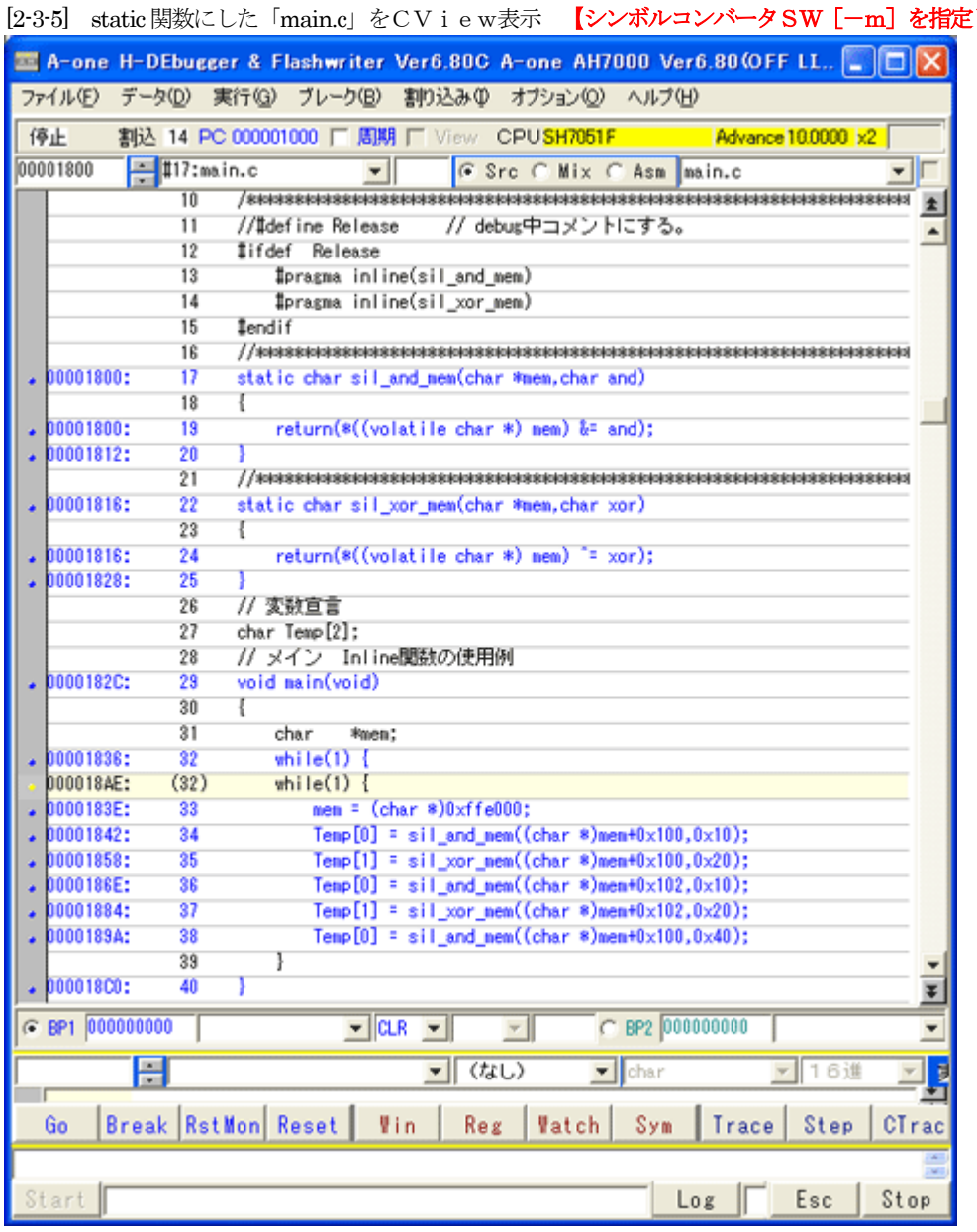
[2-3-4]　static 関数にした「main.c」をＣＶｉｅｗ表示　**【シンボルコンバータＳＷ［－m］の指定が無い場合】**



⚠ポイント（新機能）

重複しているリストライン３２を左図のように追加(Add File)して表示します。
（ｎｎ）内の数字は、同ファイルのソースラインを示します。

[2-3-5]　static 関数にした「main.c」をＣＶｉｅｗ表示　**【シンボルコンバータＳＷ［－m］を指定した場合】**

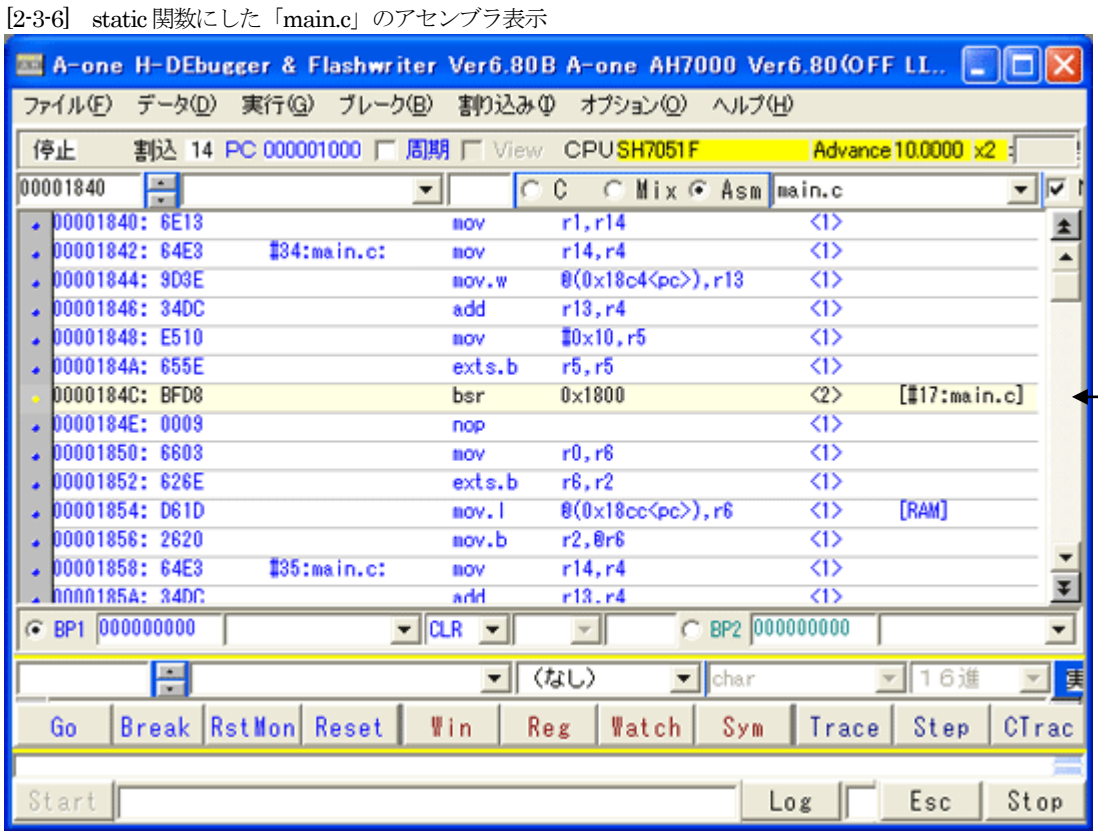


ポイント（新機能）

重複しているリストライン３２を左図のようにマージして表示します。

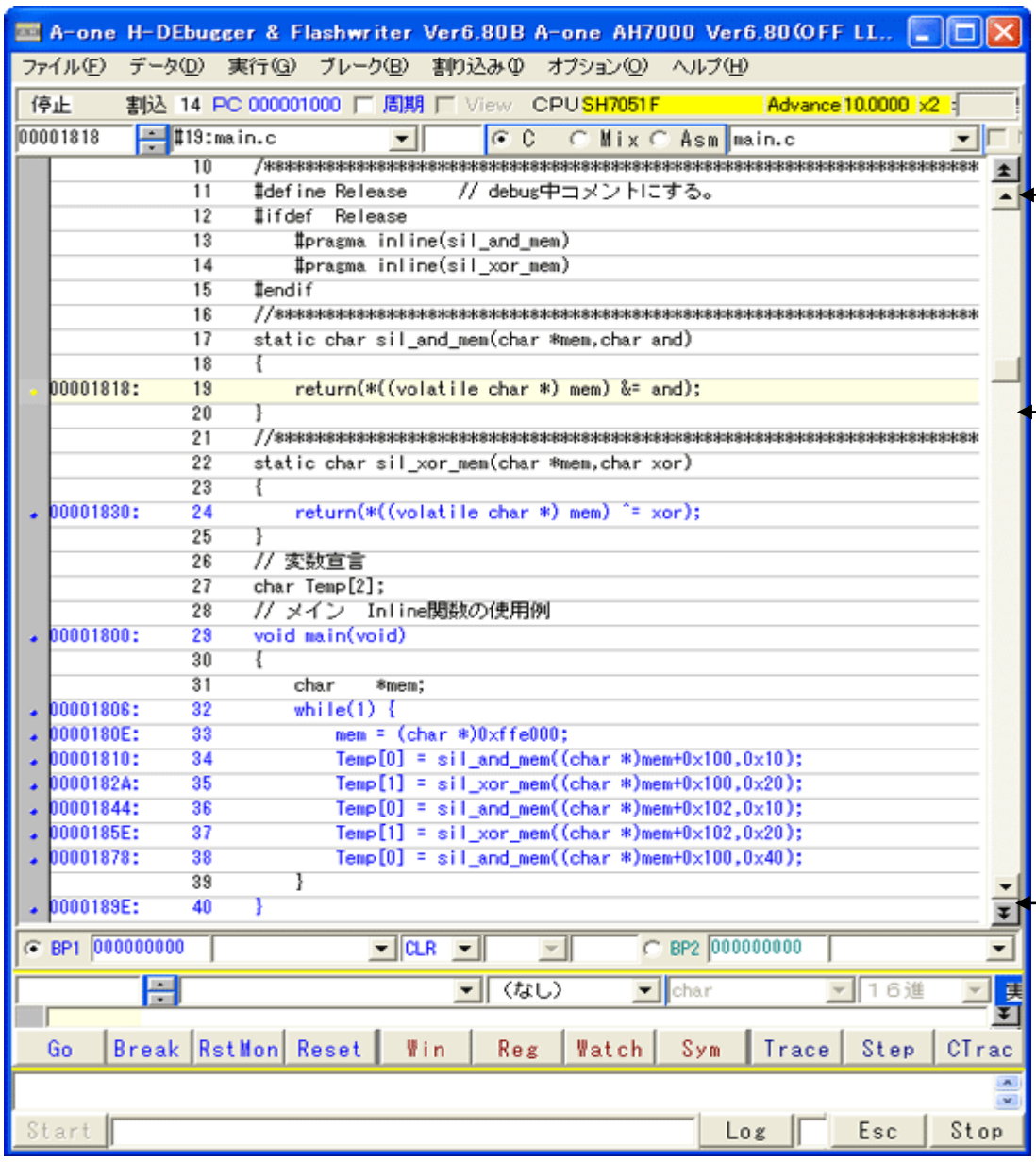
（ｎｎ）内の数字は、同ファイルのソースラインを示します。

[2-3-6]　static 関数にした「main.c」のアセンブラ表示



「static 関数」のコードになっていることが確認できます。

[2-3-7]　Inline 関数のデバッグ終了後、正式に Inline 定義を有効にさせた場合の CView 画面



ポイント

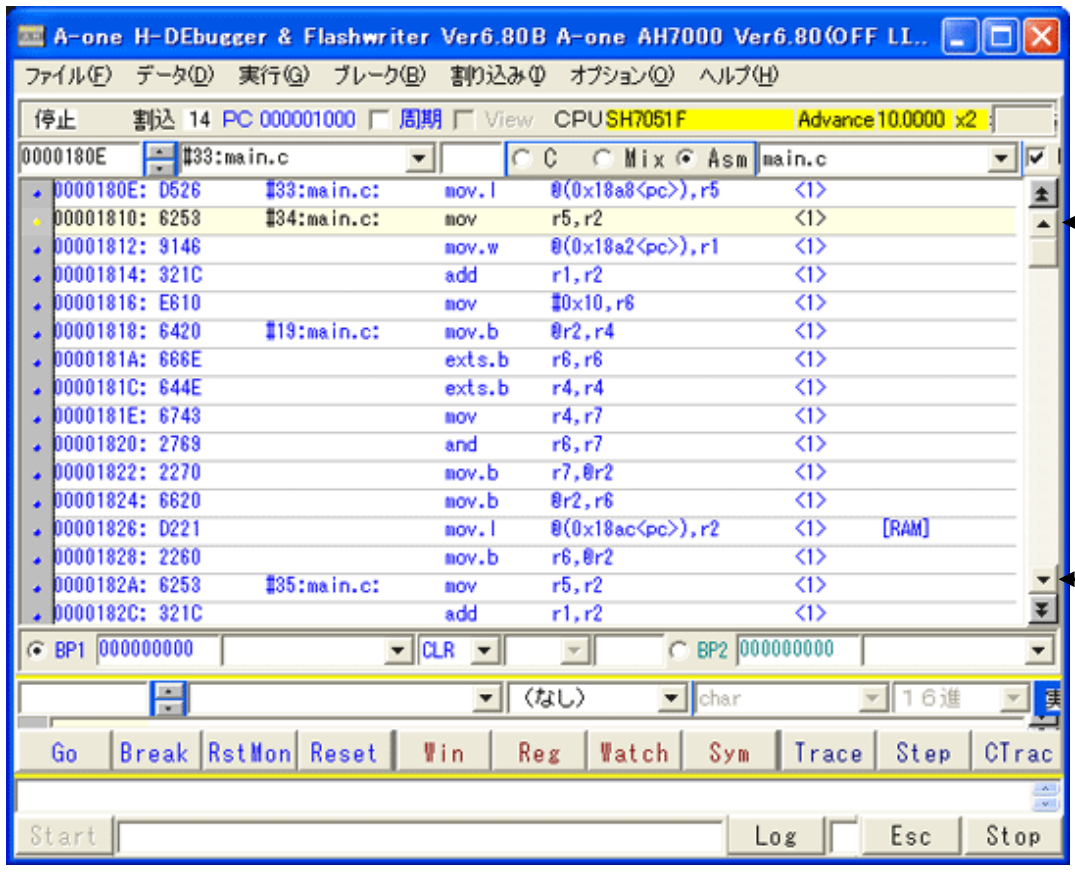
1．シンボルコンバータのオプション SW［－ｉ］を追加します。

2．左図のように、コメントを外し「inline」宣言を有効にします。

3．最初に発生した「inline」のアドレス情報のみ表示します。リスト 番号３４～３５の処理になりＨ８系と違いアドレス関係が正しい位置になります。

4．[2-3-4]　図の「Add File」部分が削除されます。

[2-3-8]　Inline 関数にした「main.c」のアセンブラ表示



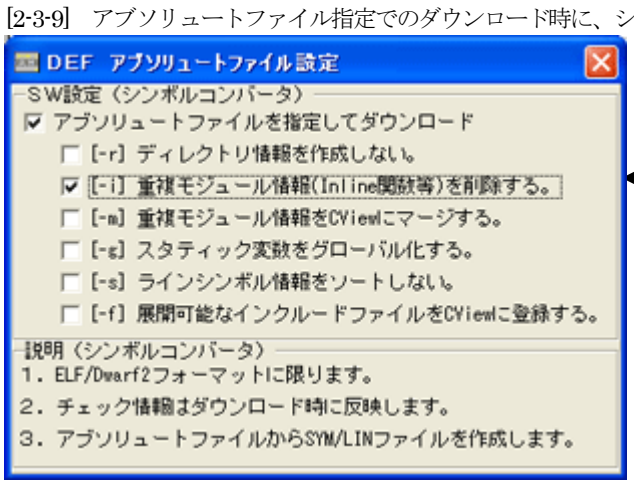
「main.c」のライン（３４）と（３５）の間にインライン関数のライン（１９）のコードが入っているのが確認できます。

この点は、Ｈ８系と違いイメージのあった展開になっています。

【ルネサス純正Ｃ－小括】　対象：全ＣＰＵ品種

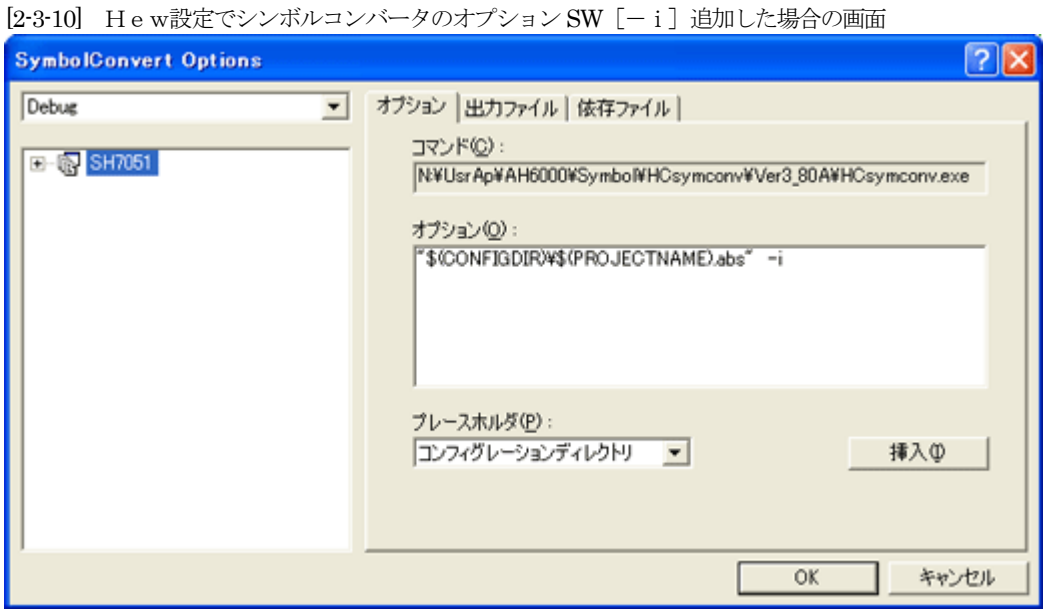
・デバッグが終了して「inline 関数」の情報が不要になった場合は、上記の方法でシンボルコンバータオプション SW［－ｉ］を追加することにより削除されます。

[2-3-9]　アブソリュートファイル指定でのダウンロード時に、シンボルコンバータのオプション SW［－ｉ］を追加した場合の画面

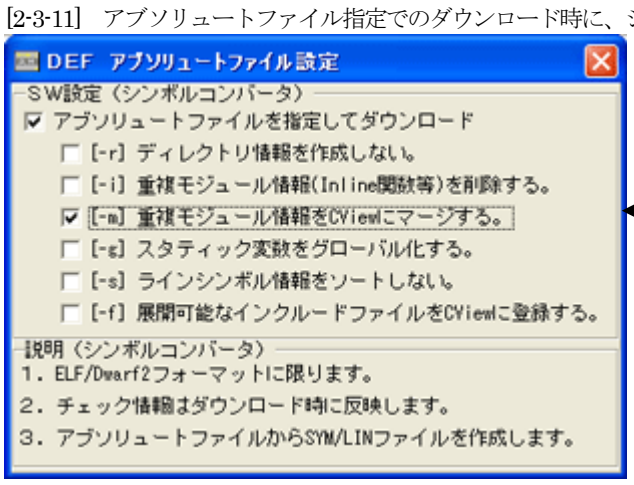


左図のように「チェック　レ」をします。

[2-3-10]　Ｈｅｗ設定でシンボルコンバータのオプション SW［－ｉ］追加した場合の画面

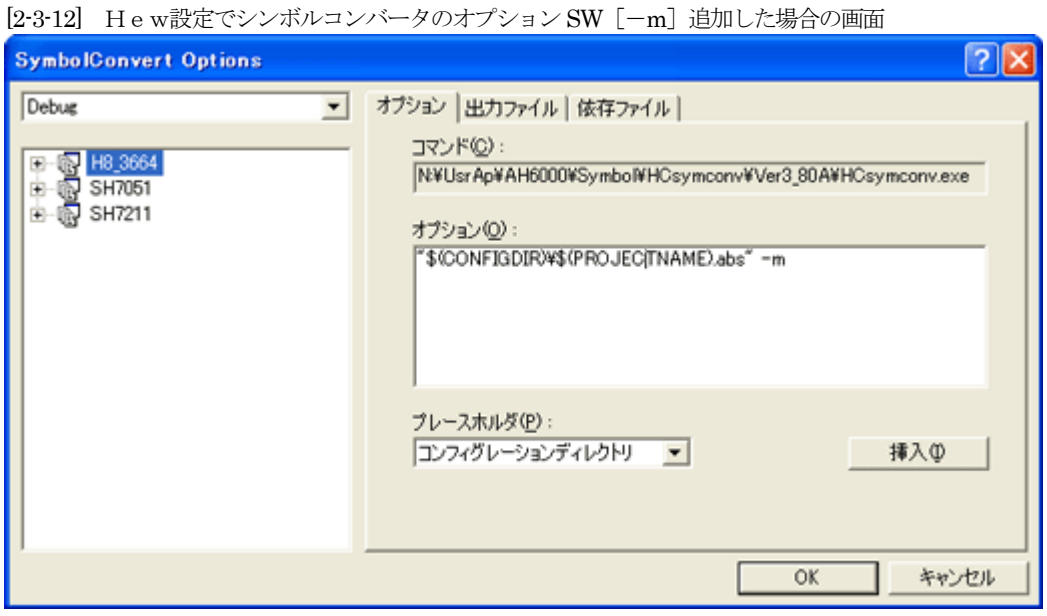


[2-3-11]　アブソリュートファイル指定でのダウンロード時に、シンボルコンバータのオプション SW［－m］を追加した場合の画面



左図のように「チェック　レ」をします。

[2-3-12]　Ｈｅｗ設定でシンボルコンバータのオプション SW［－m］追加した場合の画面



以上