Ｉｎｌｉｎｅ関数の対応について（ＫＰＩＴ－ＧＮＵ版）

Ｒｅｖ１．００　２００８／０４／１０
ＤＥＦバージョン６．８０Ｂ仕様　より

【対象】

ＫＰＩＴ－ＧＮＵ（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>　　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
【対象】　SH-2/SH-2E/SH-2A の全品種

<main.c>　　Inline関数の使用例　　記述例２

```
#include "inline.h"          // inline関数をここでインクルードする。
// 変数宣言
char Temp[2];
// 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)
```

[1-0-2]
Inline関数の使用例
【対象】　全品種共通

【説明】

ＫＰＩＴ－ＧＮＵの場合、全ＣＰＵ品種においてInline関数のライン情報(ELF/Dwarf2)出力仕様が全て同じでしたので代表品種のみで説明します。

【ＫＰＩＴ－GNU　パターン１】

＜Ｈ８ＳＸ／１５４４Ｆの例＞

・KPIT GNUH8[ELF] Toolchain(v0801)　・Compiler(v0801)　・Assmebler(v0801)　・Linker(v0801)

[1-1-1]

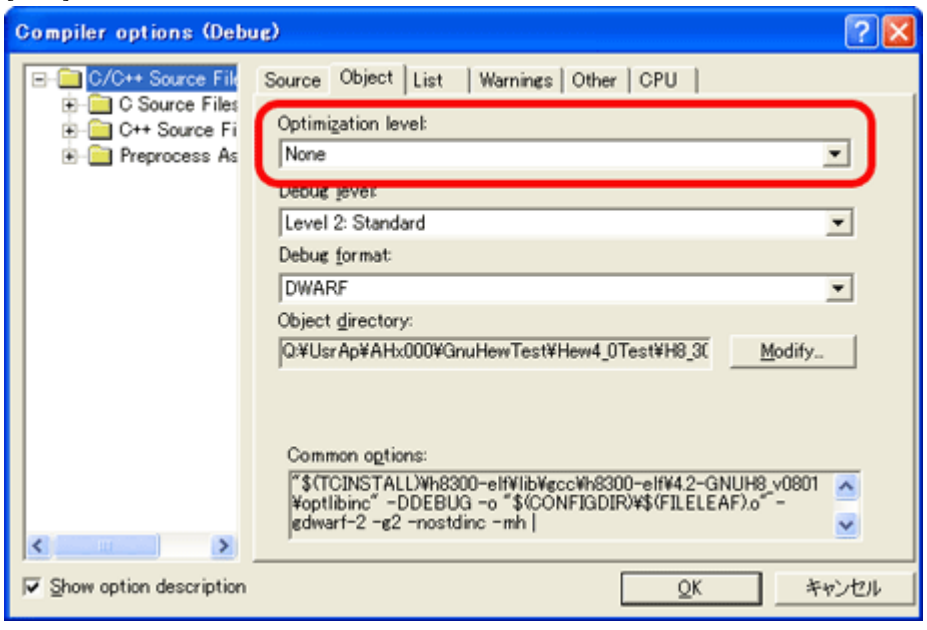
[ELF/Dwarf2 .debug_line section Code]	(address)	(line)	(address)	(line)
/ DW_LNS_extended(0x00)　set_address(0x02)　adr(0x0BA0)	+****　main.c			
/ DW_advence_line(0x03)　op(15)　line(16)				
/ DW_copy(0x01)　Lines(16)　Address(0xBA0)	+0x00000BA0	16		
/ DW_special(0x4D)　dl(3)　line(19)　da(0x8)　adr(0xBA8)	+0x00000BA8	19		
/ DW_special(0x59)　dl(1)　line(20)　da(0xA)　adr(0xBB2)	+0x00000BB2	20		
/ DW_special(0xC9)　dl(1)　line(21)　da(0x1A)　adr(0xBCC)	+0x00000BCC	21		
/ DW_special(0xC9)　dl(1)　line(22)　da(0x1A)　adr(0xBE6)	+0x00000BE6	22		
/ DW_special(0xC9)　dl(1)　line(23)　da(0x1A)　adr(0xC00)	+0x00000C00	23		
/ DW_special(0xC9)　dl(1)　line(24)　da(0x1A)　adr(0xC1A)	+0x00000C1A	24		
/ DW_special(0xC9)　dl(1)　line(25)　da(0x1A)　adr(0xC34)	+0x00000C34	25		
/ DW_set_file(0x04)　op(0x2)			+****　inline.h	
/ DW_advence_line(0x03)　op(-15)　line(10)				
/ DW_special(0x2E)　dl(0)　line(10)　da(0x4)　adr(0xC38)			+0x00000C38	10
/ DW_special(0x9F)　dl(1)　line(11)　da(0x14)　adr(0xC4C)			+0x00000C4C	11
/ DW_special(0xC9)　dl(1)　line(12)　da(0x1A)　adr(0xC66)			+0x00000C66	12
/ DW_advence_line(0x03)　op(-7)　line(5)				
/ DW_special(0x4A)　dl(0)　line(5)　da(0x8)　adr(0xC6E)			+0x00000C6E	5
/ DW_special(0x9F)　dl(1)　line(6)　da(0x14)　adr(0xC82)			+0x00000C82	6
/ DW_special(0xC9)　dl(1)　line(7)　da(0x1A)　adr(0xC9C)			+0x00000C9C	7
/ DW_advence_pc(0x02)　pc(0x4)　adr(0xCA4)			+0x00000CA4	13

<main.c>
ELF/Dwarf2 のライン情報

⚠ポイント

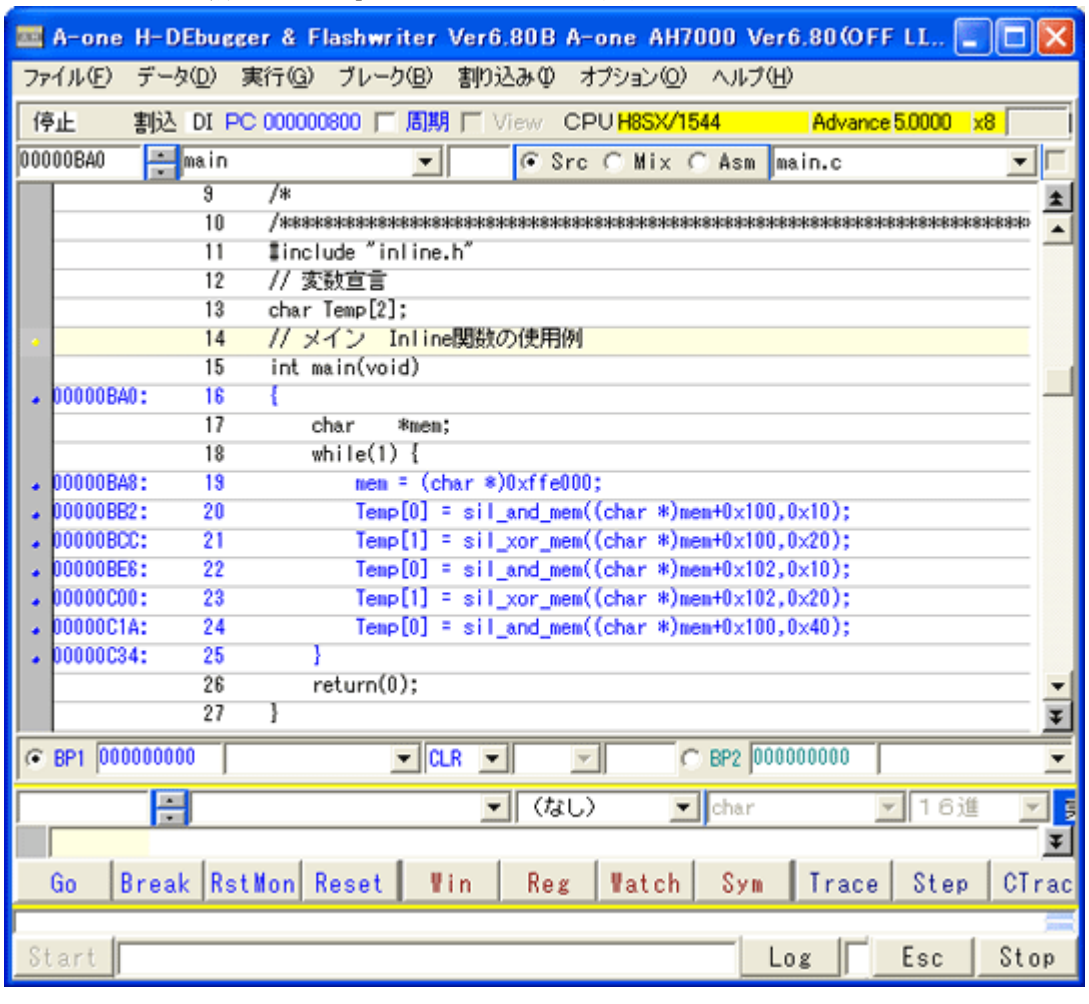
GNU の場合、Optimize＜None＞でコンパイルしますと全てStatic 関数として処理されます。(新機能)
DEF バージョン６．８０Ｂより、モジュール内にインクルードされたファイルにアドレスが存在した場合は、追加モジュールとして追加します。

[1-1-2]

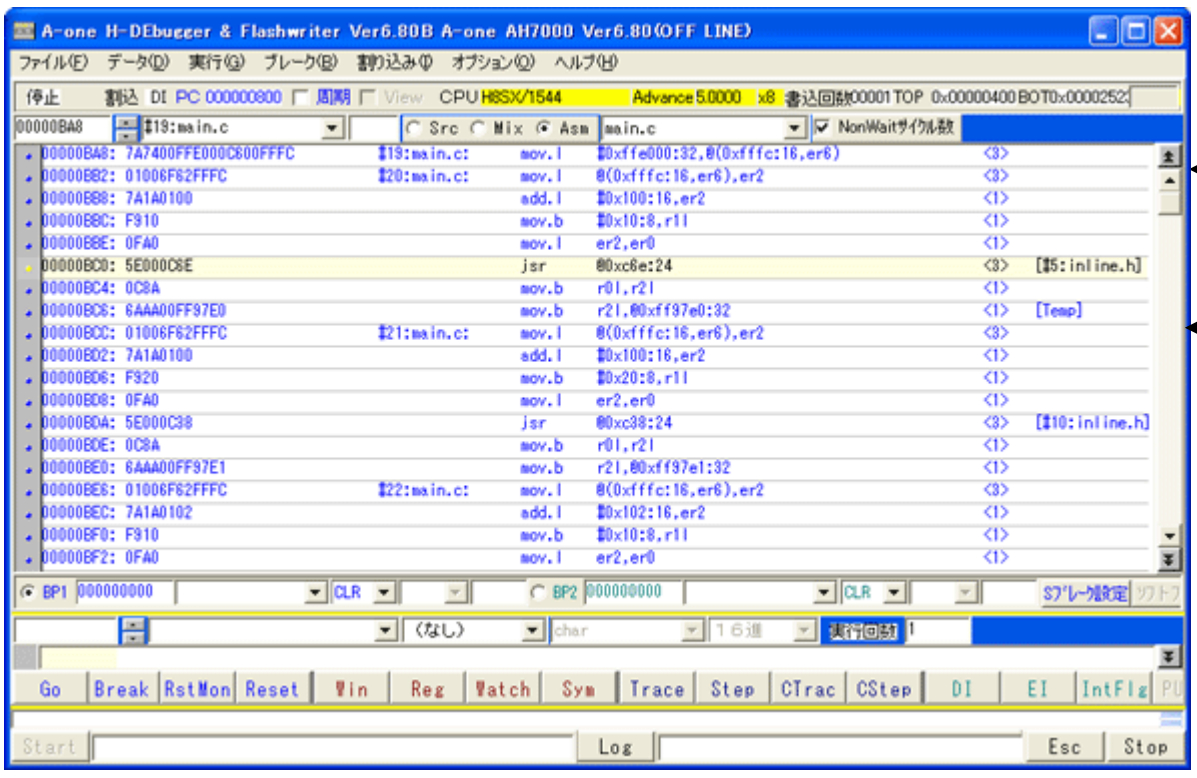


ここで Oprimize＜None＞の指定をします。

[1-1-3]　上記ライン情報で「main.c」をＣＶｉｅｗ表示した場合



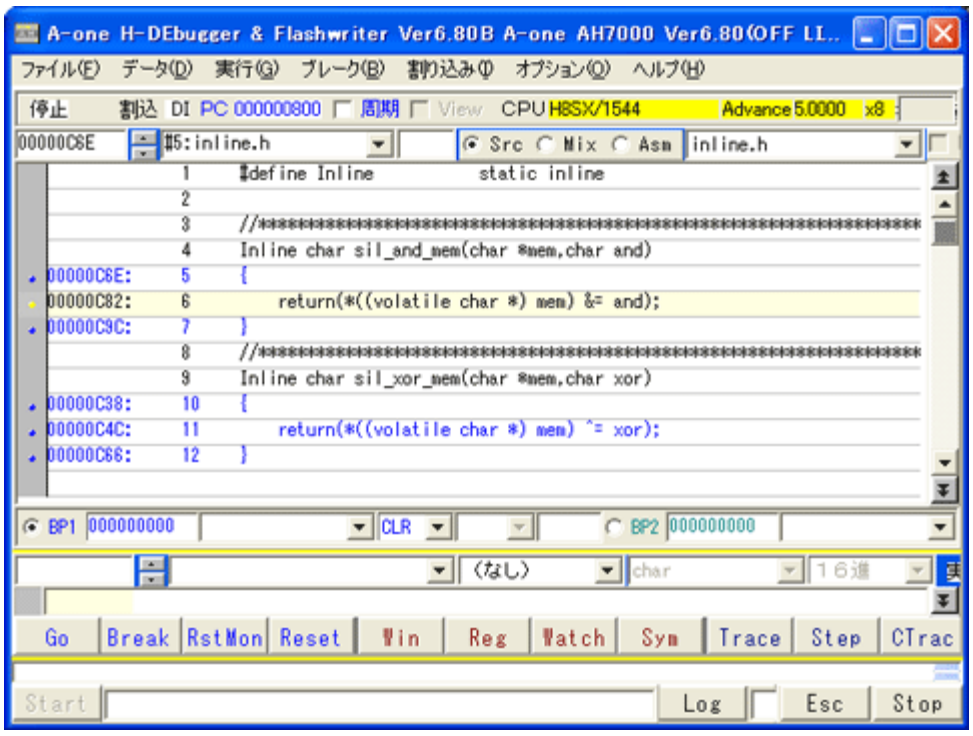
[1-1-4]　「main.c」のアセンブラ表示



⚠注意点（1）

「main.c」のライン（２０）と（２１）の間にインライン関数「inline.h」のライン（５）に Static 関数として呼ばれているのが確認できます。

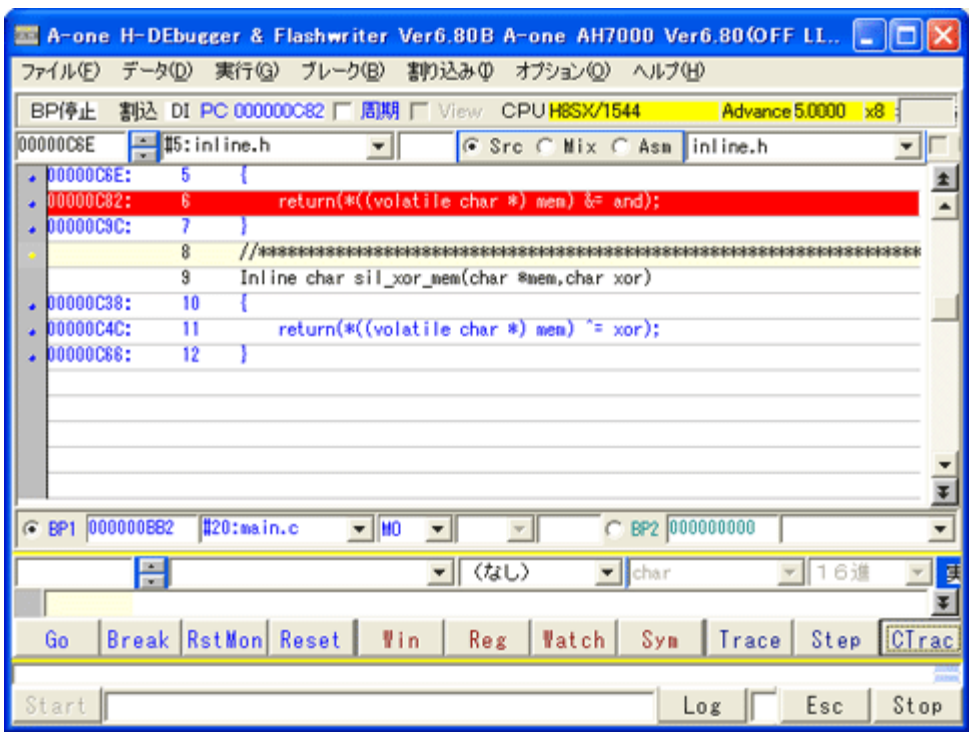
[1-1-5]　Static 関数化された「inline.h」のＣＶｉｅｗ表示



⚠ポイント（新機能）

インクルードされた「inline.h」を単独の追加モジュールとして登録されます。

[1-1-6]　「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>　　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);
}
//*********************************************************************/
// 変数宣言
char Temp[2];
// メイン　Inline 関数の使用例
int 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);
        }
        return(0);
}
```

[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
【対象】　SH-2/SH-2E/SH-2A の全品種

【説明】

ＫＰＩＴ－ＧＮＵの場合、全ＣＰＵ品種において Inline 関数のライン情報(ELF/Dwarf2)出力仕様が全て同じでしたので代表品種のみで説明します。

【ＫＰＩＴ－GNU　パターン１】

＜ＳＨ７０５１Ｆの例＞

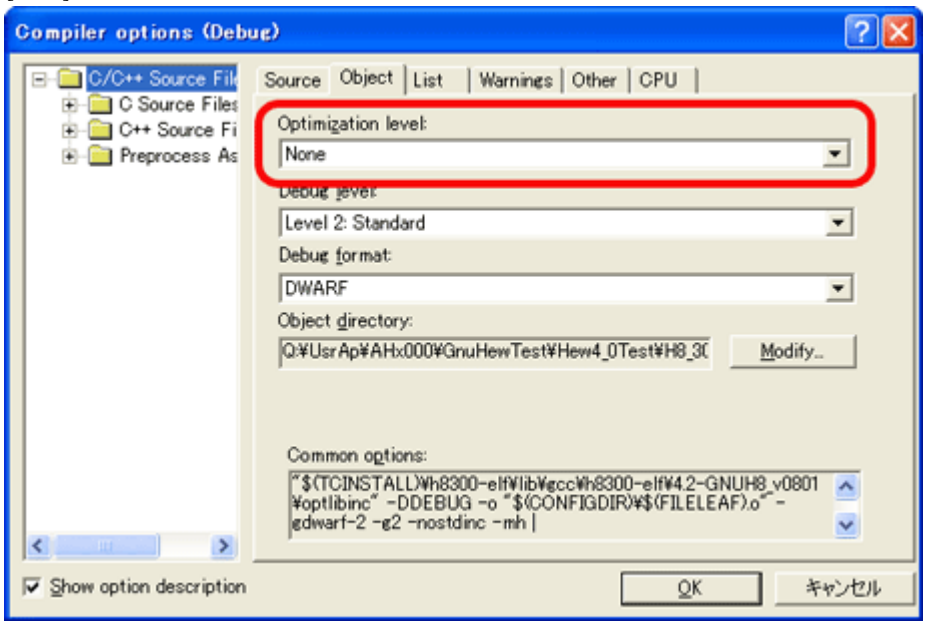
・KPIT GNUSH[ELF] Toolchain(v0801)　・Compiler(v0801)　・Assmebler(v0801)　・Linker(v0801)

[2-1-1]

[ELF/Dwarf2 .debug_line section Code]		(address)	(line)	(address)	(line)
[ELF/Dwarf2 .debug_line section Code]		(address)	(line)	(address)	(line)
/ DW_LNS_extended(0x00)	set_address(0x02) adr(0x1528)	+**** main.c			
/ DW_advence_line(0x03)	op(27) line(28)				
/ DW_copy(0x01)	Lines(28) Address(0x1528)	+0x00001528	28		
/ DW_special(0x4D)	dl(3) line(31) da(0x8) adr(0x1530)	+0x00001530	31		
/ DW_special(0x4B)	dl(1) line(32) da(0x8) adr(0x1538)	+0x00001538	32		
/ DW_special(0xC9)	dl(1) line(33) da(0x1A) adr(0x1552)	+0x00001552	33		
/ DW_special(0xD7)	dl(1) line(34) da(0x1C) adr(0x156E)	+0x0000156E	34		
/ DW_special(0xC9)	dl(1) line(35) da(0x1A) adr(0x1588)	+0x00001588	35		
/ DW_special(0xD7)	dl(1) line(36) da(0x1C) adr(0x15A4)	+0x000015A4	36		
/ DW_special(0xC9)	dl(1) line(37) da(0x1A) adr(0x15BE)	+0x000015BE	37		
/ DW_advence_line(0x03)	op(-17) line(20)				
/ DW_special(0xC8)	dl(0) line(20) da(0x1A) adr(0x15D8)	+0x000015D8	20		
/ DW_special(0x91)	dl(1) line(21) da(0x12) adr(0x15EA)	+0x000015EA	21		
/ DW_special(0xF3)	dl(1) line(22) da(0x20) adr(0x160A)	+0x0000160A	22		
/ DW_advence_line(0x03)	op(-7) line(15)				
/ DW_special(0x58)	dl(0) line(15) da(0xA) adr(0x1614)	+0x00001614	15		
/ DW_special(0x91)	dl(1) line(16) da(0x12) adr(0x1626)	+0x00001626	16		
/ DW_special(0xF3)	dl(1) line(17) da(0x20) adr(0x1646)	+0x00001646	17		
/ DW_advence_pc(0x02)	pc(0x5) adr(0x1650)	+0x00001650	38		

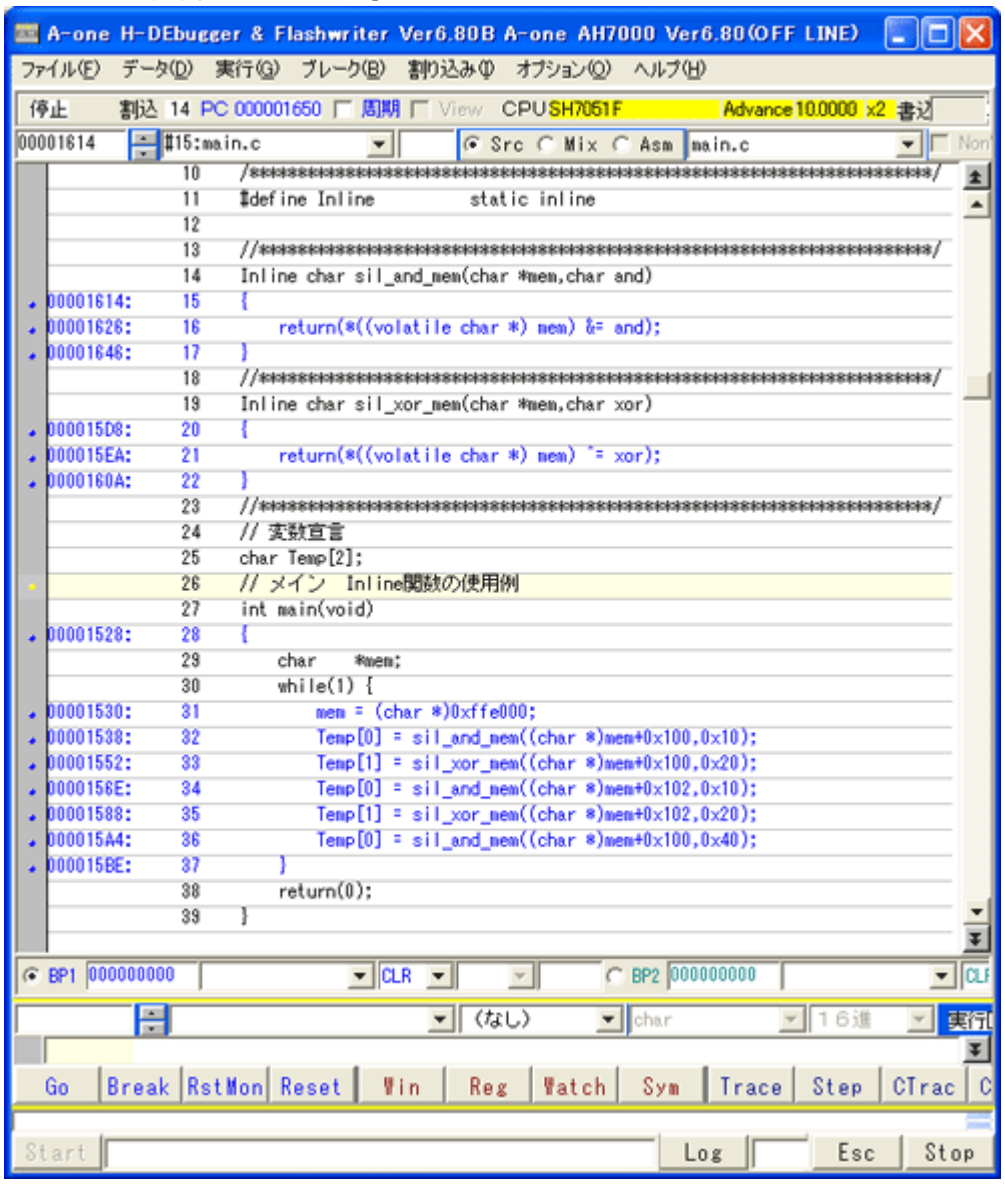
⚠ GNU の場合、Optimize＜None＞でコンパイルしますとInline関数のライン（１５）～（２２）がstatic関数として作成されます。

[2-1-2]



ここでOprimize＜None＞の指定をします。

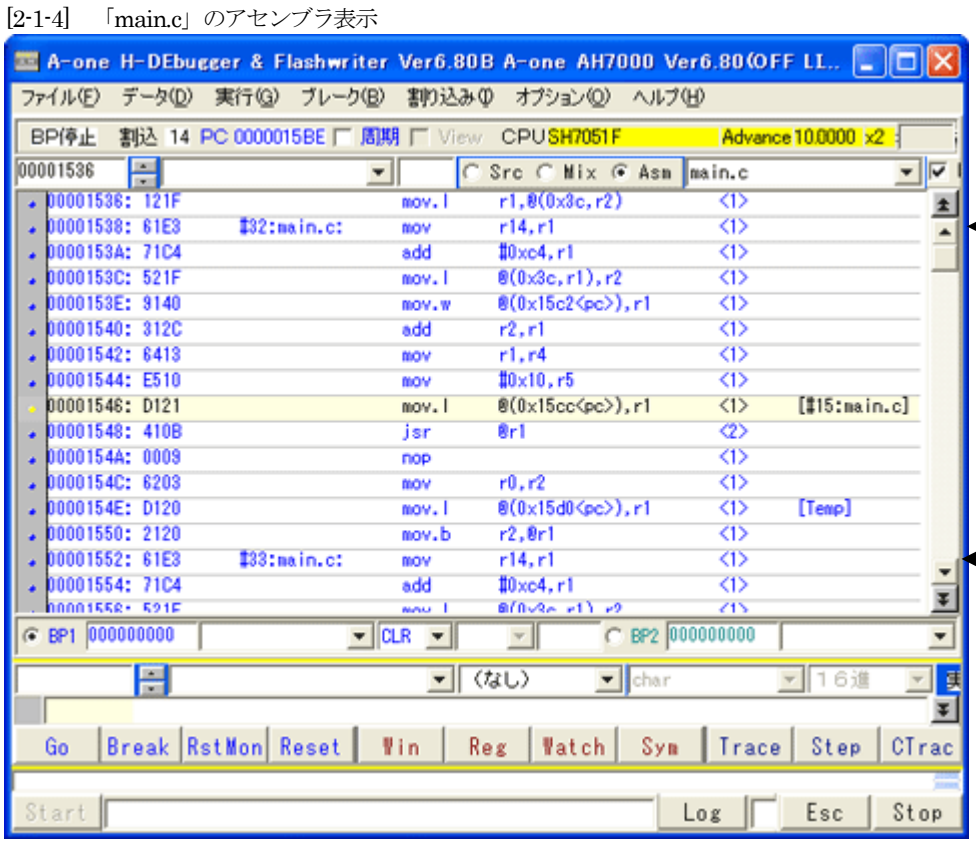
[2-1-3]　static 関数化された「main.c」のＣＶｉｅｗ表示



⚠ポイント（新機能）

ライン（１５）～（１７）とライン（２０）～（２２）の割付アドレスが昇順になっていなくても正しくＣＶｉｅｗ表示できるように改善しました。

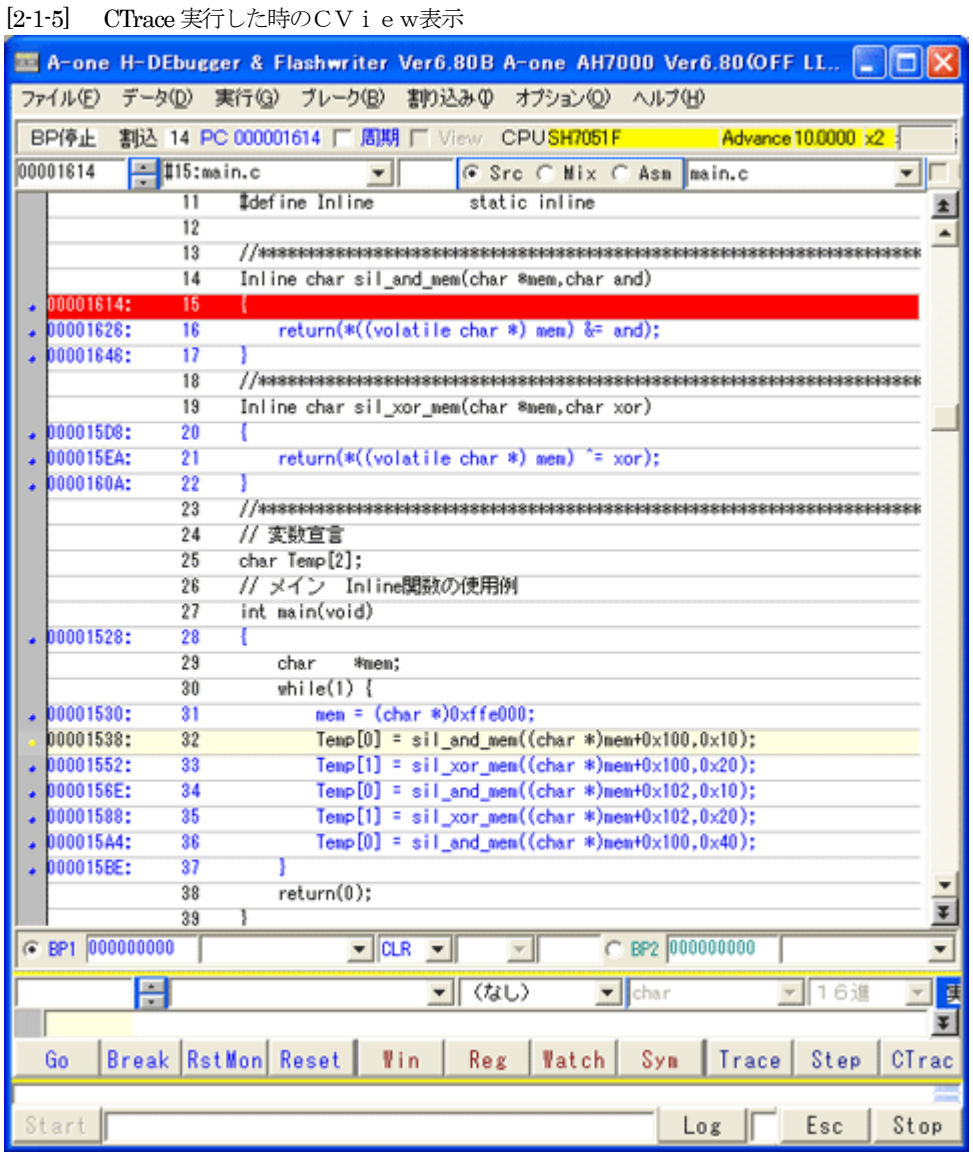
[2-1-4]　「main.c」のアセンブラ表示



⚠注意点（１）

「main.c」のライン（３２）と（３３）の間にインライン関数として宣言したライン（１５）が Static 関数として呼ばれているのが確認できます。

[2-1-5]　CTrace 実行した時のＣＶｉｅｗ表示



ライン（３２）から CTrace した場合のＣＶｉｅｗ表示です。

⚠【ＫＰＩＴ－ＧＮＵ　小括】　対象：全ＣＰＵ品種

・ＧＮＵの場合、コンパイラオプションの Optimize＜None＞の設定でコンパイルしますと Inline 関数は、全て Static 関数として処理されます。なお、現バージョンで Optimize＜Speed＞等のように有効設定にした場合のライン情報は不可解な情報になりますのでデバッグ中は有効にしないで下さい。

以上