EPSON
EXCEED YOUR VISION

CMOS 16-BIT SINGLE CHIP MICROCONTROLLER
(S1C17 Family C コンパイラパッケージ) (Ver. 3.2)

S5U1C17001C
マニュアル

SEIKO EPSON CORPORATION

Rev.1.0

評価ボード・キット、開発ツールご使用上の注意事項

1. 本評価ボード・キット、開発ツールは、お客様での技術的評価、動作の確認および開発のみに用いられることを想定し設計されています。それらの技術評価・開発等の目的以外には使用しないで下さい。本品は、完成品に対する設計品質に適合していません。

2. 本評価ボード・キット、開発ツールは、電子エンジニア向けであり、消費者向け製品ではありません。お客様において、適切な使用と安全に配慮願います。弊社は、本品を用いることで発生する損害や火災に対し、いかなる責も負いかねます。通常の使用においても、異常がある場合は使用を中止して下さい。

3. 本評価ボード・キット、開発ツールに用いられる部品は、予告無く変更されることがあります。

本資料のご使用につきましては、次の点にご留意願います。

本資料の内容については、予告無く変更することがあります。

1. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りいたします。

2. 本資料に掲載される応用回路、プログラム、使用方法等はあくまでも参考情報であり、これらに起因する第三者の知的財産権およびその他の権利侵害あるいは損害の発生に対し、弊社はいかなる保証を行うものではありません。また、本資料によって第三者または弊社の知的財産権およびその他の権利の実施権の許諾を行うものではありません。

3. 特性値の数値の大小は、数直線上の大小関係で表しています。

4. 製品および弊社が提供する技術を輸出等するにあたっては「外国為替および外国貿易法」を遵守し、当該法令の定める手続きが必要です。大量破壊兵器の開発等およびその他の軍事用途に使用する目的をもって製品および弊社が提供する技術を費消、再販売または輸出等しないでください。

5. 本資料に掲載されている製品は、生命維持装置その他、きわめて高い信頼性が要求される用途を前提としていません。よって、弊社は本（当該）製品をこれらの用途に用いた場合のいかなる責任についても負いかねます。

6. 本資料に掲載されている会社名、商品名は、各社の商標または登録商標です。

S5U1C17001C Manual

はじめに

本書は S1C17 Family の全機種に共通するソフトウェア開発ツール「S1C17 Family C コンパイラパッケージ」による C ソースプログラムのコンパイルからデバッグ、最終的に提出していただく PA ファイル（提出用データ）の作成までの手順および各ツールの使用方法を説明します。

マニュアルの読み方

本書はプログラム開発に従事されている方を対象に編集されています。したがって、以下の内容が予備知識として必要です。

- C 言語（ANSI C 準拠）に関する知識および C ソースプログラムの作成方法
- Eclipse IDE for C/C++ Developers Package の基本的な操作方法
- GNU C、binutil、および GNU リンカ（ld）用リンカスクリプトに関する知識
- アセンブリ言語に関する一般的な知識
- C コンパイラおよびアセンブラを使用したプログラム開発全般の基礎知識
- Windows の基本的な操作方法

これらの内容につきましては、ANSI C を解説した一般の書籍、Eclipse IDE for C/C++ Developers Package を解説した一般の書籍、GNU ツール、Windows のマニュアル等を参照してください。

■ インストールの前に

readmeVxxx.txt をお読みください。パッケージの構成品や各ツールの概要が記載されています。

■ インストール

readmeVxxx.txt のインストール手順に従って、ツールのインストールを行ってください。

■ ソースプログラム作成時

２章の必要個所をお読みください。ソースファイル作成上の注意事項や、アセンブリソースファイルの内容等が記載されています。ソースファイルを作成する際には、以下のマニュアルも併せて参照してください。

S1C17xxx テクニカルマニュアル
　デバイスの仕様、周辺回路の動作と制御方法について
S1C17 コアマニュアル
　CPU コアの機能と動作、命令セットの詳細について

■ プログラムのデバッグについて

8 章でデバッガの詳細を解説しています。
なお、デバッグ前に、使用するデバッグ用ツールのマニュアルも参照してください。
S1C17 Family In-Circuit Debugger Manual
　ICDmini（S5U1C17001H）の取り扱い方法について

■ 各ツールの詳細について

3～10 の各章で、それぞれのツールの詳細を解説しています。必要に応じて参照してください。

マニュアルの表記

このマニュアルは、次の表記規則に従って記述されています。

■ サンプル

サンプルの画面はシステムや搭載されているフォントにより表示が異なる場合があります。

■ 各部の名称

ウィンドウ名、メニューおよびメニューコマンド名、ボタン名、ダイアログボックス名、キーの名称は、[ ]で囲んで記述されています。たとえば、[Command]ウィンドウ、[File]メニュー、[Stop]ボタン、[q]キーなどのように記述します。

■ 命令などの名称

CPU の命令やコマンド名など、大文字、小文字の両方が許されるものについては小文字を使用しています。ただし、ユーザが指定するシンボル例などは除きます。これらの記述には `fixed-width` フォントを使用しています。

■ 数値の表記

数値は次のように記述しています。
10 進数: 数値の前後に何も付けない。（例: 123、1000）
16 進数: 数値の前に"0x"を付ける。（例: 0x0110、0xffff）
2 進数: 数値の前に"0b"を付ける。（例: 0b0001、0b10）
ただし、表示例などには、16 進数、2 進数に何も付かない場合があります。

■ マウス操作

クリック:	目的の位置にカーソル（ポインタ）を合わせ、マウスの左ボタンを一度押す操作をすべて「クリック」で表現します。右ボタンのクリック操作は「右クリック」という言葉を使います。
ダブルクリック:	目的の位置にカーソル（ポインタ）を合わせ、マウスの左ボタンを二度連続的に押す操作をすべて「ダブルクリック」で表現します。
ドラッグ:	クリックにより選択したファイル（アイコン）などを、左ボタンを押したまま別の場所に移動させる操作を「ドラッグ」で表現します。
選択:	メニューコマンドなどをクリックして選択する操作を、「選択」のみで表現します。

■ キー操作

特定のキーを押す操作は「キーを入力」、または「キーを押す」と表現しています。
[Ctrl]+[C]キーのように+を使ったキーの組み合わせは、[Ctrl]キーを押しながら[C]キーを押す操作を表します。
キーボードからの入力例については特に[ ]で囲んでいません。
なお、このマニュアルでは、マウスで可能な操作は、すべてマウス操作方法のみを記述しています。キーボードによる操作については Windows のマニュアルやヘルプ等を参考にしてください。

■ コマンドや起動時オプション、メッセージの一般形

[ ]で囲まれた項目はユーザ選択項目で、入力しなくても動作することを表しています。
< >で囲まれた内容は、特定の名称が用いられることを表しています。たとえば、<*file name*>や<*ファイル名*>は実際のファイル名に置き換えることを示しています。

■ その他開発ツールの名称

ICD: ICDmini（S5U1C17001H）を示しています。

ー 目　次 ー

1 概要

1.1 特徴

S1C17 Family C コンパイラパッケージには、 C ソースプログラムのコンパイル、アセンブリソースプログラムのアセンブルからデバッグ、さらに PSA ファイル（ROM データ）や PA ファイル（提出用データ）の作成までを行う一連のソフトウェアツールやユーティリティが含まれています。

すべてのツール/ユーティリティは S1C17 Family の全機種で共通に使用することができます。

主な特長を以下に示します。

●強力なオプティマイズ機能

C コンパイラは S1C17 アーキテクチャに対して最適化されており、非常にコンパクトなコードを生成します。また、高いオプティマイズ能力にもかかわらず、デバッグ情報が失われることが少なく、C ソースレベルのデバッグを可能とします。

●使いやすい拡張命令セットにより S1C17 コア命令セットをサポート

S1C17 コア命令セットの即値拡張機能（ext 命令）や複数の命令を 1 命令で記述可能な拡張命令を提供します。これにより、データサイズを特に意識せずにアセンブリソースを作成することができます。

●シミュレータ機能を持つ、C ソースおよびアセンブリソースレベルデバッガ

デバッガは C ソースレベルデバッグおよびアセンブリソースレベルデバッグに対応しています。エミュレータである ICDmini (S5U1C17001H)を使用することにより、ターゲットボードを動作させた状態でデバッグが行えます。また、S1C17MCU コアのコアシミュレータを使用することができます。

●Windows に対応した統合開発環境

Microsoft Windows に対応した **GNU17 IDE** は、ソース作成からデバッグまでのシームレスな統合開発環境を提供します。

1.2 ソフトウェアツールの概要

本パッケージに含まれる主要なソフトウェアツールの概要を以下に示します。

(1) C コンパイラ（xgcc.exe）

ANSI C に準拠した C コンパイラで、GNU C コンパイラがベースとなっています。このツールは **cpp.exe** および **cc1.exe** を連続して呼び出し、C ソースファイルをコンパイルして S1C17 Family 用のアセンブリソースファイルを生成します。強力なオプティマイズ能力を持ち、非常にコンパクトなコードを生成します。**xgcc.exe** は、さらにアセンブラ **as.exe** を呼び出してオブジェクトファイルを生成することもできます。

(2) アセンブラ（as.exe）

C コンパイラが出力するアセンブリソースファイルをアセンブルし、ソースファイルのニーモニックを S1C17 コアのオブジェクト（機械語）コードに変換します。**as.exe** は **xgcc.exe** から **cpp.exe** に続けて実行させることも可能です。これにより、アセンブリソースファイル内でプリプロセッサ擬似命令を使用することも許されます。結果はリンクおよびライブラリ化が可能なオブジェクトファイルとして出力されます。

(3) リンカ（ld.exe）

C コンパイラおよびアセンブラによって生成したオブジェクトコードのメモリロケーションを決定し、実行可能なオブジェクトコードを生成します。複数のオブジェクトファイルやライブラリも、このツールによって 1 つにまとめられます。

(4) デバッガ（gdb.exe）

ICDmini を制御してソースレベルのデバッグを行うソフトウェアです。特別なツールを使用せずにパソコン上でデバッグを行うシミュレーション機能も持っています。

(5) ライブラリアン（ar.exe）

ライブラリの編集ツールです。**ar.exe** は C コンパイラおよびアセンブラが生成したオブジェクトファイルをライブラリに登録することができます。ライブラリからオブジェクトを削除したり、オブジェクトファイルに復元することもできます。

(6) GNU17 IDE（eclipse.exe）

ソース作成からデバッグまでの統合開発環境を提供する開発用ワークベンチです。

本パッケージには、その他の GNU ツール、サンプルプログラムおよびいくつかのユーティリティプログラムが含まれています。それらの内容につきましては、ディスク上の"readmeV*xxx*.txt"（*xxx* はバージョン番号を表します）をご覧ください。

注: 各ツールのコマンドオプションについては、各章で説明されているオプションのみ動作保証の対象となります。その他のオプションはお客さまの責任においてご使用ください。

2 ソースファイル

この章ではソースファイルを作成する際の規則、文法について解説します。

2.1 ファイル形式とファイル名

ソースファイルは **GNU17 IDE** のエディタ、あるいは汎用のエディタ等で作成してください。

ファイル形式

標準のテキストファイルとしてセーブしてください。

ファイル名

C ソースファイル　　＜*ファイル名*＞.c
アセンブラソースファイル　　＜*ファイル名*＞.s
＜*ファイル名*＞は、32 文字以内で、以下の英数記号で指定します。
a～z、A～Z、0～9、_(アンダースコア)
なお、S1C17 ツールのファイル名には、すべてこの規定が適用されます。

ディレクトリ名

ディレクトリ名もファイル名と同じ英数記号のみが使用可能です。スペースや漢字等は使用しないでください。ディレクトリ名を含むファイル名は 64 文字以下としてください。

グローバル変数/static 変数

グローバル変数/static 変数の命名文字数は最大 200 文字です。
またグローバル変数/static 変数は合わせて 32,000 個まで使用可能です。

ファイルサイズ

C ソースファイルの最大サイズについては以下を目安としてください。

- 変数、定数、配列等のみのソースファイルは 10 万行まで使用可能です。
- 実行コード（配列、変数等を含まない）のみのソースファイルは 2 万行まで使用可能です。
  ただしソースの密度によって使用可能な行数は変動します。
- 変数、定数、配列および実行コードが混在するソースは、上記 2 項目を参考にしてください。
- コンパイルする環境によっては、上記に挙げた行数は変動します。またリソース不足によりコンパイラが強制終了することがあります。この場合はリソースの豊富な環境でビルドするか、ソースファイルを分割してください（リソースは PC に装着されている RAM 容量よりも OS に依存します）。
- C ソースファイルの 1 行の文字数は、最大 512 文字です。
- アセンブラソースに許可されている行数は最大 3 万行です。

タブ設定

タブストップは 4 文字ごとを推奨します。この文字数は IDE がソース表示を行う場合のデフォルトのタブ設定です。

EOF

各ステートメントは必ず改行し、EOF は改行後に付くように（ファイルの最後に独立するように）してください。

2.2 C ソースの文法

本パッケージの C コンパイラ **xgcc** は ANSI C 準拠の GNU C コンパイラです。
C ソースは ANSI C の規定に従って作成してください。文法について不明な点がある場合は、ANSI C を解説している一般の書籍の参照をお願いいたします。

2.2.1 データ型

C コンパイラ **xgcc** は ANSI C のすべてのデータ型に対応しています。各型のサイズ（バイト数）、表現できる数値の有効範囲を表 2.2.1.1 に示します。

表 2.2.1.1 型の種類とサイズ

型	サイズ	数値の有効範囲
char	1	-128～127
unsigned char	1	0～255
short	2	-32768～32767
unsigned short	2	0～65535
int	2	-32768～32767
unsigned int	2	0～65535
long	4	-2147483648～2147483647
unsigned long	4	0～4294967295
pointer	4	0～16777215
float	4	1.175e-38～3.403e+38（正規化数）
double	8	2.225e-308～1.798e+308（正規化数）
long long	8	-9223372036854775808～9223372036854775807
unsigned long long	8	0～18446744073709551615
wchar_t	2	0～65535

`float` 型、`double` 型は IEEE 標準規格のフォーマットに準拠しています。

`long long` 型の定数を扱う場合は、接尾子 `LL` または `ll`(`long long` 型)/`ULL` または `ull`(`unsigned long long` 型)が必要になります。この接尾子がないと、コンパイラは `long long` 型の定数として認識できないため、ワーニングになります。

例：
```
long longll_val;
       ll_val = 0x1234567812345678;
            → warning:integer constant is too large for “long” type
       Ll_val = 0x1234567812345678LL;
             → OK
```

`wchar_t` 型はワイド文字を扱うためのデータ型で、`stdlib.h`/`stddef.h` 内に `unsigned short` 型として定義されています。

2.2.2 ライブラリ関数とヘッダファイル

本パッケージには、ANSI ライブラリと浮動小数点/整数剰余演算用のエミュレーションライブラリが用意されています。また、"include"ディレクトリ内のヘッダファイルには、ライブラリの関数宣言、マクロなどが定義されています。ライブラリ関数を使用する場合は、その宣言を含んだヘッダファイルを`#include`命令でインクルードしてください。
なお、一部の ANSI ライブラリ関数は本パッケージでは対応していないため、ANSI ライブラリに含まれていません。本パッケージで対応していない ANSI ライブラリ関数を使用する必要がある場合は、お客さまの責任で関数の実装及び、プロトタイプの宣言を行ってください。ただし、本パッケージで未対応の ANSI ライブラリ関数についても、ヘッダファイル内でプロトタイプ宣言のみされているものもありますので、この場合はプロトタイプ宣言をする代わりに、該当するヘッダファイルをインクルードした上で関数の実装を行ってください。
ライブラリファイルの種類およびヘッダファイルとの対応は次のとおりです。

表 2.2.2.1 ライブラリファイル/関数一覧

ANSIライブラリ

ファイル名	関数／マクロ	対応ヘッダファイル
libc.a	perror, getchar, fgetc, getc, gets, fgets, fscanf, scanf, sscanf, fread, putchar, fputc, putc, puts, fputs, ungetc, fprintf, printf, sprintf, vfprintf, vprintf, vsprintf, fwrite	stdio.h
	abort, exit, malloc, calloc, realloc, free, atoi, atol, atof, strtol, strtoul, strtod, abs, labs, div, ldiv, rand, srand, bsearch, qsort	stdlib.h
	setjmp, longjmp	setjmp.h
	time, mktime, gmtime	time.h
	acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan, tanh	math.h,errno.h,float.h,limits.h
	memchr, memmove, strchr, strcspn, strncat, strpbrk, strstr, memcmp, memset, strcmp, strerror, strncmp, strrchr, strtok, memcpy, strcat, strcpy, strlen, strncpy, strspn	string.h
	isalnum, iscntrl, isgraph, isprint, isspace, isxdigit, toupper, isalpha, isdigit, islower, ispunct, isupper, tolower	ctype.h
	va_start, va_arg, va_end	stdarg.h

エミュレーションライブラリ

ファイル名	関数
libgcc.a	___subdf3,__adddf3,__addsf3,__ashldi3,__ashlhi3,__ashlsi3,__ashrdi3,__ashrhi3,__ashrsi3, __cmpdi2,__divdf3,__divdi3,__divhi3,__divsf3,__divsi3,__eqdf2,__eqsf2,__extendsfdf2, __fixdfdi,__fixdfsi,__fixsfdi,__fixsfsi,__fixunsdfdi,__fixunsdfsi,__fixunssfdi,__fixunssfsi, __floatdidf,__floatdisf,__floatsidf,__floatsisf,__gedf2,__gesf2,__gtdf2,__gtsf2,__ledf2,__lesf2, __lshrdi3,__lshrhi3,__lshrsi3,__ltdf2,__ltsf2,__moddi3,__modhi3,__modsi3,__muldf3, __muldi3,__mulhi3,__mulsf3,__mulsi3,__nedf2,__negdf2,__negdi2,__negsf2,__nesf2, __subsf3,__truncdfsf2,__ucmpdi2,__udivdi3,__udivhi3,__udivsi3,__umoddi3,__umodhi3, __umodsi3,__cmpsi2,__ucmpsi2

プロトタイプ宣言のみされている関数

関数	対応ヘッダファイル
freopen, tmpfile, tmpnam, remove, rename, fopen, fclose, setbuf, setvbuf, fflush, clearerr, feof, ferror, fseek, fgetpos, fsetpos, ftell, rewind	stdio.h
atexit, getenv, system	stdlib.h
difftime, clock, localtime, asctime, ctime	time.h

ライブラリに含まれる関数の機能については"5 ライブラリ"を参照してください。また、ライブラリ関数を使用した場合は、リンク時にその関数を含むライブラリファイルを指定してください。リンカは指定されたライブラリファイルの中から必要なオブジェクトモジュールのみを取り出してリンクします。

2.2.3 インラインアセンブル

C コンパイラ **xgcc** はインラインアセンブルに対応しており、**asm** 文が使用可能です。これに伴い、"asm"は予約語となりますので注意してください。

形式： **asm**("***<文字列>***");

例 1：
```
/* HALT mode */
asm("halt");
```

例 2：
```
/* Trap Table*/
asm(".long        BOOT¥n¥
     .long        ADDR_ERR¥n¥
     .long        NMI¥n¥}
     .space       4¥n¥
     .long        EINT0¥n¥
     .long        EINT1");
```

例 3：
```
BOOT(){
     asm("xld.a %sp,0x3f00");  /* set SP */
          :
}
```

アセンブリソースの記述方法の詳細については"2.3　アセンブリソースの文法"を参照してください。

2.2.4 プロトタイプ宣言

●割り込み処理関数の宣言

割り込み処理関数は次の形式でプロトタイプ宣言します。

<型> <関数名> **__attribute__ ((interrupt_handler));**

例：
```
void foo(void) __attribute__ ((interrupt_handler));

int int_num;
void foo()
{

     int_num = 5;
}
```

アセンブラコード

```
foo:
     ld.a -[%sp],%r2
     ld %r2,5
     xld [int_num],%r2
     ld.a %r2,[%sp]+
     reti
```

2.3 アセンブリソースの文法

2.3.1 ステートメント

アセンブリソースの個々の命令や定義をステートメントと呼びます。基本的なステートメントの構成は次のとおりです。

●構文パターン

1	*<ニーモニック>*	*(<オペランド>)*	*(;<コメント>)*
2	*<アセンブラ擬似命令>*	*(<パラメータ>)*	*(;<コメント>)*
3	*<ラベル>:*	*(;<コメント>)*	
4	*;<コメント>*		
5	*<拡張命令>*	*<オペランド>*	*(;<コメント>)*
6	*<プリプロセッサ擬似命令>*	*(<パラメータ>)*	*(;<コメント>)*

例:

```
————————————————ステートメント———————————————— ―構文パターン―
; boot.s                                                    4
; boot program                                              4

#define SP_INI,0x3f00      ; Stack pointer value           6
                                                           2
        .text
        .long BOOT         ; BOOT VECTOR                   2
BOOT:                                                      3
        xld.a %sp,SP_INI   ; set SP                        5
        xcall main         ; goto main                     5
        jpr BOOT           ; infinity loop                 1
```

上記の例が一般的なソース記述方法です。視認性を高めるため、各ステートメントを構成する要素はタブやスペースで位置を揃えています。

●制限事項

- 各行には 1 つのステートメントのみ記述できます。1 行に 2 つ以上の命令を記述するとエラーとなります。ただし、コメントは命令やラベルの行にも記述することができます。

  例：
  ```
  ;OK
  BOOT:   ld %r1,%r2
          ld %r0,%r1
  ;Error
  BOOT:   ld %r1,%r2         ld %r0,%r1
  ```

- 1 つのステートメントを複数行に分けて記述することはできません。1 行で完結していないステートメントはエラーとなります。

  例：
  ```
  ;OK
          ld %r1,%r2
  ;Error
          ld %r1,
             %r2
  ```

- コメントを除き、使用可能な文字は ASCII キャラクタ（英数記号）に限られます。また、使用できる記号にも制限があります（詳細は後述）。
  コメントには ASCII キャラクタ以外の漢字なども使用できます。漢字をコメントにする場合は、`/* ... */`を使用してください。

(1) 命令（ニーモニック＆オペランド）

S1C17 コアに対する命令は<*ニーモニック*>＋<*オペランド*>の構成となります。命令によっては、オペランドを持たないものもあります。

●命令表記の一般形

一般形：***<ニーモニック>***
<ニーモニック> タブまたはスペース ***<オペランド>***
<ニーモニック> タブまたはスペース ***<オペランド 1>, <オペランド 2>***

例：
```
nop
call   SUB1
ld     %r0,0x4
```

行の中で、ニーモニックの記述開始位置に制限はありません。ニーモニックの前にあるタブやスペースは無視されます。一般的にはタブによってニーモニックの頭揃えを行います。

オペランドを持つ命令の場合、ニーモニックとオペランド間は 1 個以上のタブまたはスペースで区切る必要があります。また、オペランドが複数の場合は、オペランド間を 1 個のカンマ（,）で区切ります。オペランド間のスペースは無視されます。

オペランドの要素については後述します。

●ニーモニックの種類

S1C17 Family では、以下の S1C17 コア命令を使用できます。

ld.b	ld.ub	ld	ld.a		
add	add/c	add/nc	add.a	add.a/c	add.a/nc
adc	adc/c	adc/nc	sub	sub/c	sub/nc
sub.a	sub.a/c	sub.a/nc	sbc	sbc/c	sbc/nc
cmp	cmp/c	cmp/nc	cmp.a	cmp.a/c	cmp.a/nc
cmc	cmc/c	cmc/nc			
and	and/c	and/nc	or	or/c	or/nc
xor	xor/c	xor/nc	not	not/c	not/nc
sr	sa	sl	swap		
cv.ab	cv.as	cv.al	cv.la	cv.ls	
jpr	jpr.d	jpa	ipa.d	jrgt	jrgt.d
jrge	jrge.d	jrlt	jrlt.d	jrle	jrle.d
jrugt	jrugt.d	jruge	jruge.d	jrult	jrult.d
jrule	jrule.d	jreq	jreq.d	jrne	jrne.d
call	call.d	calla	calla.d	ret	ret.d
int	intl	reti	reti.d	brk	retd
ext	nop	halt	slp	ei	di
ld.cw	ld.ca	ld.cf			

各命令の詳細については S1C17 コアのマニュアルを参照してください。

●文字の制限

ニーモニックは大文字（A～Z）、小文字（a～z）のどちらで記述しても受け付けられます。
たとえば、"ld"、"LD"、"Ld"はすべて"ld"命令として受け付けられます。
シンボル等と区別するため、本マニュアルでは小文字を使用します。
オペランドについては後述します。

(2) アセンブラ疑似命令

アセンブラ **as** は、GNU アセンブラに標準で用意されている擬似命令をサポートしています。
標準の擬似命令については GNU アセンブラのマニュアルを参照してください。アセンブラ擬似命令はピリオド(.)で始まります。よく使われるアセンブラ擬似命令を以下に示します。

.text		.text セクションを宣言
.section .data		.data セクションを宣言
.section .rodata		.rodata セクションを宣言
.section .bss		.bss セクションを宣言
.long	<*data*>	4 バイトデータを定義
.short	<*data*>	2 バイトデータを定義
.byte	<*data*>	バイトデータを定義
.ascii	<*string*>	ASCII 文字列を定義
.space	<*length*>	空白領域（0x0）を定義
.zero	<*length*>	空白領域（0x0）を定義
.align	<*value*>	指定境界アドレスにアライメント
.global	<*symbol*>	シンボルのグローバル宣言
.set	<*symbol*>,<*address*>	シンボルに絶対アドレスを定義

(3) ラベル

ラベルはプログラム中の任意のアドレスを参照するための識別子です。プログラムの分岐先アドレスや.text/.data セクション内の任意のアドレスを、ラベルとして定義したシンボルを使って参照することができます。

●ラベルの定義

次の形式で記述したシンボルがラベルとみなされます。

<シンボル>:

先頭のスペースやタブは無視されます。一般的には行の先頭から記述します。
定義されたシンボルは記述された位置のアドレスを示します。
実際のアドレス値はリンク時に決定します。

●制限事項

使用できる文字は以下のものに限られます。
　A～Z a～z _ 0～9
ただし、数字で始まることはできません。大文字と小文字は区別されます。

例：
```
;OK        ;Error
FOO:       1label:
_Abcd:     0_ABC:
L1:
```

(4) コメント

コメントは一連のルーチン、あるいは各ステートメントの意味を記述しておくもので、コード化の対象とはなりません。

●コメントの定義

セミコロン（;）で始まり、改行で終わる文字列がコメントとみなされます。
"/*"で始まり"*/"で終わる文字列もコメントとして扱われます。
コメントには ASCII キャラクタのほかに漢字などの非 ASCII キャラクタも使用可能です。
また、ラベルや命令と同じ行に記述することもできます。

例：
```
; This line is a comment line.
LABEL:              ;Comment for LABEL.
      ld %a,%b      ;Comment for the instruction on the left.
/*
      This type of comment can include
      newline characters.
*/
```

●制限事項

コメントが複数行に渡る場合は、各行ごとにセミコロンで始めるか、"/*"と"*/"を使用する必要があります。

例：
```
;These are
    comment lines.          2行目はコメントとはみなされません。エラーとなります。

;These are
;   comment lines.          2行ともコメントとみなされます。

/*
    These are
    comment lines.          2行ともコメントとみなされます。
*/
```

(5) 空白行

本アセンブラでは、復帰/改行コードのみの空白行も許されます。一連のルーチンの区切りなど、コメント行にする必要はありません。

2.3.2 オペランドの表記

ここでは命令のオペランドに使用するレジスタ名、シンボル、定数の表記法を説明します。

(1) レジスタ名

S1C17 コアの内部レジスタ名には、すべてパーセント（%）を付けます。レジスタ名は大文字でも、小文字でも受け付けます（区別されません）。

汎用レジスタ(%rd, %rs, %rb)	表記
汎用レジスタ R0～R7	%r0～%r7 または%R0～%R7

特殊レジスタ	表記
スタックポインタ SP	%sp または%SP
プログラムカウンタ PC	%pc または%PC

[ ]を伴うレジスタ間接アドレッシングの指定にも、レジスタ名に%が必要です。

例：[%r7] [%r1]+ [%sp+imm7]

注: レジスタ名の先頭に%がない場合、シンボルとみなされます。逆に%で始まるものはすべてレジスタとみなされ、存在しないレジスタ名を使用するとエラーとなります。

(2) 数値表記

アセンブラ as は 10 進形式、16 進形式、2 進形式の 3 種類の数値表記に対応しています。

●10 進形式の数値表記

0～9 の数値のみで表記されたものは 10 進数とみなされます。負の数を指定する場合は、マイナス（-）符号を数値の前に付けてください。

例: 1　255　-3

0～9、および符号（-）以外の文字は使用できません。

●16 進形式の数値表記

16 進数を指定する場合は数値の前に"0x"を付けてください。

例: 0x1a　0xff00

"0x"の後ろには 0～9、a～f、A～F 以外の文字は使用できません。

●2 進形式の数値表記

2 進数を指定する場合は数値の前に"0b"を付けてください。

例: 0b1001　0b01001100

"0b"の後ろには 0 と 1 以外の文字は使用できません。

●数値の指定範囲

即値データのサイズ（指定範囲）は各命令により異なります。
各即値データの指定可能範囲は次のとおりです。

表 2.3.2.1 即値データの種類と指定可能範囲

記号	種類	10進数	16進数	2進数
imm3	3ビット即値	0～7	0x0～0x7	0b0～0b111
imm5	5ビット即値	0～31	0x0～0x1f	0b0～0b1 1111
imm7	7ビット即値	0～127	0x0～0x7f	0b0～0b111 1111
sign7	符号付き7ビット即値	-64～63	0x0～0x7f	0b0～0b111 1111
sign8	符号付き8ビット即値	-128～127	0x0～0xff	0b0～0b1111 1111
sign10	符号付き10ビット即値	-512～511	0x0～0x3ff	0b0～0b11 1111 1111
imm13	13ビット即値	0～8,191	0x0～0x1fff	0b0～0b1 1111 1111 1111
imm16	16ビット即値	0～65,535	0x0～0xffff	0b0～0b1111 1111 1111 1111
sign16	符号付き16ビット即値	-32,768～32,767	0x0～0xffff	0b0～0b1111 1111 1111 1111
imm20	20ビット即値	0～1,048,575	0x0～0xfffff	0b0～0b1111 1111 1111 1111 1111
sign21	符号付き21ビット即値	-1,048,576～1,048,575	0x0～0x1fffff	0b0～0b1 1111 1111 1111 1111 1111
sign23	符号付き23ビット即値	-4194304～4194303	0x0～0x7fffff	0b0～0b111 1111 1111 1111 1111 1111
imm24	24ビット即値	0～16,777,215	0x0～0xffffff	0b0～0b1111 1111 1111 1111 1111 1111
sign24	符号付き24ビット即値	-8,388,608～8,388,607	0x0～0xffffff	0b0～0b1111 1111 1111 1111 1111 1111

(3) シンボル

即値データによるアドレス指定には、他の場所で定義されているシンボルを使用することができます。

●シンボル定義

使用可能なシンボルは次のいずれかの方法によって 24 ビットの値として定義されます。

1. ラベルとして記述（text、data または bss セクション内）

   例： `LABEL1:`
   LABEL1 は`.text`、`.data` または`.bss` セクション内の記述された位置（アドレス）を示すシンボル

2. `.set` 擬似命令で定義

   例： `.set ADDR1,0xff00`
   ADDR1 は絶対アドレス 0x00ff00 を示すシンボル

●文字の制限

使用できる文字は以下のものに限られます。
A～Z a～z _ 0～9
ただし、数字で始まることはできません。大文字と小文字は区別されます。

●ローカルシンボルとグローバルシンボル

通常、定義したシンボルはローカルシンボルとなり、そのファイル内でのみ参照可能です。
このため、複数のファイルで同じ名称のシンボルを定義することもできます。
他のファイルで定義されているシンボルを参照するには、シンボルを定義したファイル内で、
`global` 擬似命令によるグローバル宣言が必要です。

●シンボルの拡張表記

通常、シンボルでアドレスを参照する場合、アドレスを指定するオペランドにそのシンボル名を記述します。

```
例： call    LABEL      ← LABEL = sign10
     ld.a    %rd,LABEL  ← LABEL = sign7
```

アセンブラ **as** では、次のようなディスプレースメント付きの参照も許可します。
LABEL + imm24　LABEL + sign24

```
例： xcall   LABEL+0x10
```

2.3.3 拡張命令

拡張命令は、通常 ext 命令を含む複数の命令で記述する内容を 1 つの命令として記述できるようにしたもので、アセンブラ **as** により必要最小限の基本命令に展開されます。

●拡張命令の種類

xadd	xadd.a	xadc	xsub	xsub.a	xsbc	xcmp
xcmp.a	xcmc					
sadd	sadd.a	sadc	ssub	ssub.a	ssbc	scmp
scmp.a	scmc					
xand	xoor	xxor				
sand	soor	sxor				
xld	xld.a	xld.b	xld.ub			
sld	sld.a	sld.b	sld.ub			
xjpr	xjpr.d	xjpa	xjpa.d	xjreq	xjreq.d	xjrne
xjrne.d	xjrgt	xjrgt.d	xjrge	xjrge.d	xjrlt	xjrlt.d
xjrle	xjrle.d	xjrugt	xjrugt.d	xjruge	xjruge.d	xjrult
xjrult.d	xjrule	xjrule.d	xcall	xcall.d	xcalla	xcalla.d
sjpr	sjpr.d	sjpa	sjpa.d	sjreq	sjreq.d	sjrne
sjrne.d	sjrgt	sjrgt.d	sjrge	sjrge.d	sjrlt	sjrlt.d
sjrle	sjrle.d	sjrugt	sjrugt.d	sjruge	sjruge.d	sjrult
sjrult.d	sjrule	sjrule.d	scall	scall.d	scalla	scalla.d
xld.cw	xld.ca	xld.cf				
sld.cw	sld.ca	sld.cf				

●拡張命令の使用方法

オペランドには即値拡張後のサイズの数値、シンボルを直接記述可能です。

```
例：  xcallLABEL          ; ext LABEL[23:10]
                          ; call LABEL[9:0]

      sld.a%r1,imm16      ; ext imm16[15:7]
                          ; ld.a %r1,imm16[6:0]

      xld.a %r1,imm24     ; ext imm24[23:20]
                          ; ext imm24[19:7]
                          ; ld.a %r1,imm24[6:0]
```

また、基本命令の即値拡張機能以外にも、以下のようなオペランドの指定も命令により可能です。

```
例：  xld.a  %r0,symbol + 0x10      ; R0 ← symbol + 0x10
      xjpa   LABEL + 5              ; LABEL+5 のアドレスにジャンプ
```

オペランドを含む拡張命令の詳細については"6.6 拡張命令"を参照してください。

2.3.4 プリプロセッサ疑似命令

アセンブリソースファイル内でも C プリプロセッサ **cpp** の擬似命令を使用できます。
主な擬似命令を以下に示します。

#include	ファイルの挿入
#define	文字列および数値の定義 / マクロ定義
#if-#else-#endif	条件アセンブル

例:
```
#include      "define.h"
#define       NULL          0
#ifdef        TYPE1
              ld            %r0,0
#else
              ld            %r0,-1
#endif
```

プリプロセッサ擬似命令の詳細については GNU C プリプロセッサのマニュアルを参照してください。

注: プリプロセッサ擬似命令を含むアセンブリソースはプリプロセッサを通す必要があり(xgcc の-c と -xassembler-with-cpp オプションで指定)、アセンブラ as に直接入力することはできません(直接入力するとエラーになります)。

2.4 ソース作成上の注意事項

(1) タブストップはできるだけ 4 文字ごとに設定してください。4 文字以外のタブ設定をしたソースをデバッガ **gdb** でソース表示/ミックス表示すると、ソース部分がずれて出力される場合があります。

(2) C ソース、アセンブリソースをデバッグ情報付きでコンパイル/アセンブルする場合には、他のソースファイルをインクルード（#include）しないでください。デバッガ **gdb** が正しく動作しなくなります。ソースを含まない通常のヘッダファイルは問題ありません。

(3) C モジュールとアセンブラモジュールを混在して使用する場合、引数や戻り値のサイズ、受け渡し方法には十分注意してください。

(4) C コンパイラのデフォルト設定ではアドレス空間が 24 ビットになります。これに対して int 型は 16 ビットのため、unsigned int/unsigned short 型変数とポインタを含む演算には注意が必要です。
たとえば、以下のコードでは意図した処理を行いません。

```
int* ip_Pt;
unsigned int i = 1;

ip_Pt += (-1)*i;
```

このコードは、ip_Pt += (-1);を意図していますが、実際には ip_Pt += 0xffff;として処理されます。
アドレス空間が 16 ビットの場合はこれで正しく処理されますが、24 ビットの場合には演算結果が不正なアドレスになってしまいます。

アドレス空間を 24 ビットとしたときにポインタとの演算を行う場合は、unsigned int/unsigned short 型変数を避けるか、あるいは次のように定数に接尾語'L' を付けて、long 型として演算されるようにする必要があります。

```
ip_Pt += (-1L)*i;
```

(5) C ソースでは関数名を関数へのポインタとしても使用できますが、関数名を使用してその値を実数型（float/double）の変数や配列に代入することはできません。
整数型の変数や配列には代入できます。
ただし、グローバル変数またはグローバル配列の宣言時に関数名で値を代入する場合は、アドレス空間のサイズによって代入可能な整数型が制限されます。

24 ビット空間（デフォルト設定）の場合
long/unsigned long 型のグローバル変数/配列にのみ、関数名による代入が行えます。

16 ビット空間の場合（-mpointer16 指定時）
short/unsigned short/int/unsigned int 型のグローバル変数/配列にのみ、関数名による代入が行えます。

宣言時以外であれば、任意の整数型のグローバル変数/配列に関数名で値を代入することができます。

ローカル変数またはローカル配列の場合は、整数型であれば宣言時か否かにかかわらず、いつでも関数名で値を代入することができます。

例:16 ビット空間の場合（-mpointer16 指定時）

1）`short s_Global_Val = (short)boot;`
→ short 型のグローバル変数 s_Global_Val の宣言時に、関数名による代入は行えます。

2）`long l_Global_Val = (long)boot;`
→ long 型のグローバル変数 l_Global_Val の宣言時に関数名による代入を行うと、エラーが発生します。
`error: initializer element is not constant`

3）`short s_local_val = (short)boot;`
→ short 型のローカル変数 s_local_val への関数名による代入は行えます。

4）`long l_local_val = (long)boot;`
→ long 型のローカル変数 l_local_val への関数名による代入は行えます。

5) char c_Global_Val;
```
void sub()
{
   c_Global_Val = (char)boot;
}
```
→ 宣言時以外であれば、char 型のグローバル変数 c_Global_Val への関数名による代入は行えます。

(6) C ソースでは関数ポインタも使用できますが、この関数ポインタも（5）と同様に実数型（float/double）の変数や配列には代入できません。
整数型の変数や配列には代入できます。
ただし、グローバル変数またはグローバル配列の宣言時は関数ポインタを代入することはできません。

宣言時以外であれば、整数型のグローバル変数/配列に関数ポインタを代入することができます。
整数型のローカル変数またはローカル配列の場合は、宣言時か否かにかかわらず、関数ポインタを代入することができます。

ただし、アドレス空間のサイズとグローバル/ローカル変数またはグローバル/ローカル配列の型によってはワーニングが発生します。

24 ビット空間（デフォルト設定）の場合
long/unsigned long 型以外の変数/配列に関数ポインタを代入するとワーニングが発生します。

16 ビット空間の場合（-mpointer16 指定時）
short/unsigned short/int/unsigned int 型以外の変数/配列に関数ポインタを代入するとワーニングが発生します。

例:16 ビット空間の場合（-mpointer16 指定時）
```
void (* fp_Pt)(void); // 戻り値と引数の型が共に void 型の関数ポインタの宣言
```

1) short s_Global_Val = (short)fp_Pt;
→ グローバル変数 s_Global_Val の宣言時に関数ポインタを代入すると、エラーが発生します。
```
   error: initializer element is not constant
```

2) short s_local_val = (short)fp_Pt;
→ short 型のローカル変数 s_local_val への関数ポインタの代入は行えます。

3) long l_local_val = (long)fp_Pt;
→ long 型のローカル変数 l_local_val への関数ポインタの代入はできますが、ワーニングが発生します。
```
   warning: cast from pointer to integer of different size
```

4) short s_Global_Val;
```
void sub()
{
   s_Global_Val = (short)fp_Pt;
}
```
→ 宣言時以外であれば、short 型のグローバル変数 s_Global_Val への関数ポインタの代入は行えます。

(7) 関数は必ずプロトタイプ宣言もしくは extern 宣言を行ってください。
プロトタイプ宣言もしくは extern 宣言がされてなく、かつ、同じファイル内の前方に定義部のない関数をコールした場合、関数をコールする側で想定している型と、実際にコールされる関数の型が一致せずに誤動作する可能性があります。この場合でも、コンパイル時にエラーは発生しません。ただし、同じファイル内にコールされる関数の定義部がある場合には、ワーニングが発生します。他のファイル内にコールされる関数の定義部がある場合は、-Wall オプションを付けない限りワーニングも発生しません。
戻り値に関しては、この場合、暗黙のうちに int 型と判断されます。そのため、int 型より大きい型の戻り値は正しく取得できなくなります。

例:
```
long l_Val=0x12345678,l_Val_2;

int main()
  {
    l_Val_2 = sub();              // l_Val_2 の値が 0x5678 になってしまいます。
    return 0;
  }

long sub()
  {
    long l_wk;

    l_wk = l_Val;
    return l_wk;
  }
```

(8) char 以外のポインタを使用して奇数番地のメモリへのリード/ライトを行わないでください。
この場合、アドレス不整例外が発生します。

例:
```
int *ip_Pt;

int sub()
{

  ip_Pt = (int *)0x3;
  (*ip_Pt) = 0x2;

  return 0;                       //ここでアドレス不整例外が発生します。

  }
```

(9) C 言語の規格上、動作が未定義の処理を行った場合、最適化オプションや、ローカル変数/外部変数などの違いにより演算結果が異なる場合がありますので、注意してください。
未定義の処理としては、以下のケースなどが挙げれらます。

- 浮動小数点型→整数型に型変換した時に、オーバーフローしていた場合。
- シフト演算を負数もしくは、型昇格された後の演算対象のビット長と同じかより大きい値で行う場合。

(10) C 言語の規格上、非互換の型のポインタ経由で変数にアクセスしようとした時に-Wall オプションが付いている場合、以下のワーニングが表示されることがあります。この場合、ポインタを経由した変数への参照及び代入は正しく行われない場合があります。

```
warning: dereferencing type-punned pointer will break strict-aliasing rules
```

例:-O3 オプションが指定されている場合
```
int sub()
{
        int p1 = 0 ;
        short *p2 = (short *)&p1 ;
```

p2 と&p1 は非互換の型のため、ワーニングが発生します。
この場合、ポインタ p2 を経由した変数 p1 への参照及び、代入は正しく行われない場合があります。

3 GNU17 IDE

この章では、**GNU17 IDE** の機能と操作方法を説明します。

3.1 概要

3.1.1 特長

GNU17 IDE（以下 **IDE**）は、S1C17 Family C コンパイラパッケージ（S5U1C17001C）を使用したプログラムの開発を支援する統合開発環境を提供します。
本 **IDE** は Eclipse IDE for C/C++ Developers Package に S1C17 用プログラム開発に必要な機能（以下 GNU17 独自プラグイン）を追加したものです。Eclipse の標準機能ついては、Eclipse IDE for C/C++ Developers Package を解説している一般の書籍をご覧ください。
IDE の主な機能は次のとおりです。

- **プロジェクトの生成・管理**
  GNU17 用のプロジェクトを作成し（GNU17 独自プラグインの機能）、1 つのアプリケーションの作成に必要な全ソースファイルなどをプロジェクトとして一元管理できます。
- **GNU17 version 2 プロジェクトのインポート**（GNU17 独自プラグインの機能）
  GNU17 version 2（GNU17 V2.3.0 など）で作成したプロジェクトをインポートできます。
- **プロジェクトの設定**
  S1C17 用プログラムのビルドに必要なプロパティをプロジェクトに設定できます（GNU17 独自プラグインの機能）。
- **GNU 準拠の C およびアセンブラをサポート**
  GNU 準拠の C およびアセンブリ言語でのソース作成/編集機能があります。他のエディタで作成したソースも読み込み可能です。
- **プログラムのビルド機能**
  プロジェクトの設定に従ってコンパイルからリンカまでのシーケンスを実行し、アプリケーション（*.elf/*.psa）をビルドできます。
- **デバッガ gdb を呼び出すためのランチャ機能**
  ビルド処理終了後、デバッガ **gdb** を呼び出し、デバッグを行うことができます。

3.1.2 IDE 使用上の注意

●IDE の動作保証について

IDE は開発プラットフォーム Eclipse 上で動作し、Eclipse の機能も利用しています。ただし、**IDE** では、本マニュアルに記載されていない機能については動作保証対象外としていますのでご注意ください。

●Eclipse プラグインのバージョン

IDE の基盤となっている Eclipse プラグインとそのバージョンは以下のとおりです。

表 3.1.2.1 Eclipse プラグインのバージョン

プラグイン	バージョン
Eclipse Platform	4.4.1
Eclipse CDT	8.5.0

GNU17 独自プラグインは、これらのプラグイン上で動作させることを想定して提供しています。

●IDE 上での日本語の使用について

IDE 上では、ファイルやフォルダ名、ファイル内の文字列に日本語（文字コード Shift-JIS/MS-932）を使用可能ですが、ビルドなどに使用する GNU17 ツールは日本語をサポートしておりません。したがって、ファイルやフォルダ名、およびソースファイル等の実行行では日本語を使用しないでください。
（ソースファイル内のコメント行のみ日本語を使用可能です。）

3.2 IDE の起動と終了

3.2.1 起動方法

IDE の起動方法は以下のとおりです。

(1) C:¥EPSON¥GNU17V3¥eclipse ディレクトリ内の **eclipse.exe** のアイコンをダブルクリックし、**IDE** を起動させます。
Windows のスタートアップメニューから[EPSON MCU] > [GNU17V3] > [GNU17V3 IDE]を選択しても起動します。
また、コマンドラインからも引数なしで起動可能です。

(2) Eclipse のスプラッシュに続き、[Workspace Launcher]ダイアログが表示されます。ここではプロジェクトの格納などに使用する作業用ディレクトリ（ワークスペース）を指定します。任意のディレクトリを選択または新規作成してワークスペースに設定可能です。ワークスペースは、起動後でも[File]メニューの[Switch workspace]で変更可能です。

※ ワークスペースディレクトリには、プロジェクトディレクトリ（.project ファイルが含まれるディレクトリ）を指定しないでください。プロジェクトインポート（[プロジェクトをワークスペースにコピー]が ON のとき）に失敗することがあります。現在のワークスペースディレクトリは、[File] > Switch workspace] > [other...]より[Workspace Launcher]ダイアログで確認できます。

(3) [OK]ボタンをクリックします。
IDE が起動します。

3.2.2 終了方法

IDE を終了させるには、 [File]メニューから[Exit]を選択してください。
エディタで開いているファイルが保存されていない場合は、保存するか否かを選択するダイアログが表示されます。いずれかを選択して終了させてください。ウィンドウの （閉じる）ボタンでも終了可能です。ただし、この場合は終了を確認するダイアログが表示されますので、終了する場合は[OK]ボタンを、終了を取り消す場合は[キャンセル]ボタンをクリックします。

3.3 プロジェクト

3.3.1 プロジェクトとは

IDE では個々のアプリケーション開発をプロジェクト名で管理します。具体的には、1 つのアプリケーション開発を始めるに当たって指定したプロジェクト名を持つディレクトリを作成し、そこでソースファイルなどのリソースやコンパイラなどのツールが生成するファイルを管理します。プロジェクトディレクトリにはプロジェクトの管理用ファイル（.cproject、.project）も生成され、**IDE** により随時更新されます。

注: プロジェクトディレクトリ内の管理用ファイルは、絶対に IDE 上の操作以外で編集、移動、削除しないでください。プロジェクトが再開できなくなることがあります。

3.3.2 新規プロジェクトの作成

IDE によるアプリケーション開発は、新規プロジェクトの作成から始めます。その手順は次のとおりです。

(1) 以下のいずれかの方法で[GNU17 Project] ウィザードを開始させます。

- [File]メニューから[New] > [GNU17 Project]を選択します。
- ツールバーの[New]ショートカットから[GNU17 Project]を選択します。
- ツールバーの[New C/C++ Project]ショートカットから[GNU17 Project]を選択します。
- [Project Explorer]ビューのコンテキストメニューから[New]>[GNU17 Project]を選択します。

(2) 起動したウィザード上で、[Project name:]にプロジェクト名を入力します。

- プロジェクト名には、半角英数字とアンダーバー以外の文字は使用できません。

(3) プロジェクトディレクトリを作成する場所を指定します。（特に必要な場合のみ）
デフォルト設定では[Use default location]チェックボックスが選択されており、**IDE** 起動時に指定したワークスペースディレクトリ内にプロジェクトフォルダが生成されます。通常はこのまま次に進んでください。

(4) GNU17 設定画面で、S1C17 用プログラムのビルドに必要な情報を設定します。

(5) [Program Type]コンボボックスから生成するプログラムを選択します。
Application(.elf/.psa): 実行可能プログラム
Library(.a): ライブラリファイル

(6) [Target CPU]コンボボックスからターゲットプロセッサの機種であるターゲット CPU を選択します（Program Type が Application の場合）。
目的のターゲット CPU がリストに存在しない場合は、機種別情報ファイル（gnu17_mcu_model_xxx.zip）をセイコーエプソンの Web サイトから入手するか、弊社営業窓口にお問い合わせください。

(7) [Memory Model]コンボボックスからメモリモデルを選択します。
REGULAR: 24 ビット（16M バイト空間を使用）
SMALL: 16 ビット（64K バイト空間を使用）

(8) [GCC Version] コンボボックスから GCC バージョンを選択します。
4.9：GCC4.9
6.4：GCC6.4

(9) [OK]ボタンをクリックします。
指定した名称のプロジェクトが作成されます。プロジェクトを新規作成すると、現在のワークスペースまたは（3）で指定したディレクトリ内にプロジェクトと同じ名称のディレクトリが作成されます。プロジェクト名のディレクトリがすでに存在する場合は、そのディレクトリをそのままプロジェクトディレクトリとして使用します。

3.3.3 ソースファイルの作成と追加

IDE は C、アセンブラをサポートしており、これらの言語のソースファイルからオブジェクトを生成可能です。
オブジェクトの生成に必要なソースファイルは src フォルダに入れてください。src フォルダ内のファイルはビルド対象になります。

●ソースファイルの作成

ソースファイルの作成の手順は次のとおりです。

(1) 以下のいずれかの操作を行います。

- [File]メニューから[New]>[SourceFile]（ソースファイルを作成する場合）または[HeaderFile]（ヘッダファイルを作成する場合）を選択します。
- [Project Explorer]ビューのコンテキストメニューから[New]>[SourceFile]または[HeaderFile]を選択します。
- ツールバーの[New]ショートカットから[SourceFile]または[HeaderFile]を選択します。
- ツールバーの[New C/C++ SourceFile]ショートカットから[SourceFile]または[HeaderFile]を選択します。
- ツールバーの[New C/C++ SourceFile]ボタンをクリックします（ソースファイルを作成する場合）。

[New Source File] (または[New HeaderFile]）ダイアログが表示されます。

(2) [SourceFile:](または[HeaderFile:]）に作成するファイルの名称を入力します。
C ソースファイルを作成する場合は拡張子を".c"、アセンブラソースファイルを作成する場合は拡張子を".s"として入力してください。対応しているソースファイルの拡張子が入力されないと、警告が表示されます。ただしこの場合でも、その名称でファイルを作成することは可能です。
[SourceFolder:]には、現在作業中のプロジェクトフォルダ名が表示されます。他のディレクトリに作成する場合は、パスを入力するか、[Browse...]ボタンでディレクトリを選択してください。

(3) [Finish]ボタンをクリックします。中止する場合は[Cancel]ボタンをクリックしてください。

作成したファイルがビューに表示され、エディタで開かれます。
メニューから[HeaderFile]を選択して新規のヘッダファイルを生成した場合は、ファイル内にマクロ定義（<ファイル名> _H_）が自動的に記述されます。

●ソースファイルの追加

Windows のエクスプローラ上でコピーしたソースファイルを、[Project Explorer]ビュー内にペーストすることでプロジェクトにソースファイルを追加できます。

また、次のようにソースファイルをインポートすることでプロジェクトに追加できます。

(1) [Project Explorer]ビューでファイルをインポートするプロジェクトまたはフォルダをクリックして選択します。

(2) 以下のいずれかの操作を行います。

- [File]メニューから[import...]を選択します。
- [Project Explorer]ビューのコンテキストメニューから[import...]を選択すると[import]ウィザードが起動します。

(3) 一覧から[File System]を選択し、[Next>]ボタンをクリックします。

(4) [From directory:]の[Browse...]ボタンをクリックして表示されるフォルダ選択ダイアログで、インポートするファイルがあるフォルダを選択してください。選択したフォルダへのパスが[From directory:]に入力されます。以前インポート元となったフォルダであれば、[From directory:]のボタンで表示される履歴から選択することも可能です。

(5) インポートするファイルを選択します。

(6) フォルダを選択してインポートすることも可能です。この場合はフォルダからのディレクトリ構造を含めてインポートされます（ファイルは選択したフォルダ内のもののみインポートされます）。

(7) [Finish]をクリックします。

(8) [Project Explorer]ビューにインポートしたファイルが表示されます。

3.3.4 割り込みベクタとブート処理の記述

ライブラリ crt0.o には、割り込みベクタテーブルと、ターゲット MCU がリセットされてから main 関数が呼び出されるまでの処理（ブート処理）が用意されています。
crt0.o の詳細については、"5.2 起動処理ライブラリ"を参照してください。

●ブート処理の追加

以下の関数を定義することで、ブート時に独自の処理を実行できます。定義しない場合は crt0.o が定義している内容が有効になります。

void _init_device (void)	デバイスを初期化する。_init_section、_init_lib、_init_sysの前に呼び出される。 crt0.oでは何もしない関数として定義されている。
void _start_device (void)	デバイスの動作を開始する。_init_section、_init_lib、_init_sysの後、mainの前に呼び出される。 crt0.oでは割り込み許可eiを実行する関数として定義されている。
void _stop_device (void)	デバイスの動作を停止する。mainの後に呼び出される。 crt0.oでは割り込み禁止diを実行する関数として定義されている。

●スタックポインタの初期値の設定

crt0.o は__START_stack の値に従ってスタックポインタの初期値を設定します。ターゲット機種に搭載されている RAM のサイズに合わせて__START_stack の値を設定してください。C ソースファイルで設定する場合は、任意の個所で、次のように記述します。

```
asm(".global __START_stack");

asm(".set __START_stack, 0x7c0"); /* initial value of SP register */
```

リンカシンボルファイル（ldsyms.ini）で設定する場合は、次のように記述します。

```
__START_stack = 0x7c0; /* initial value of SP register */
```

以上の例では 0x7c0 としています。記述しない場合はデフォルトリンカスクリプトが定義している値が有効になります。

●割り込みベクタの登録

crt0.o では割り込み処理関数の名前を_vectorXX_handler(XX は 10 進 2 桁のベクタ番号)に固定しています。したがって、次のような_vector08_handler 関数を用意すれば、ベクタ番号 8 の割り込み処理関数として割り込みベクタテーブルに登録されます。

```
void _vector08_handler(void) __attribute__((interrupt_handler));
```

既存の割り込み処理関数 sampleInterrupt をベクタ番号 8 に登録する場合、sampleInterrupt を定義している C ソースファイルに以下の記述を追加してください。

```
void _vector08_handler(void) __attribute__((alias("sampleInterrupt")));
```

この記述により、sampleInterrupt の別名として_vector08_handler 関数が定義されます。sampleInterrupt 関数は割り込み処理関数ですので、次のようにプロトタイプ宣言されている必要があります。

```
void sampleInterrupt(void) __attribute__((interrupt_handler));
```

3.3.5 既存プロジェクトのインポート

ここでは、本 **IDE** に同梱されているサンプルプロジェクトなどの既存のプロジェクトを現在のワークスペースに取り込む方法を説明します。GNU17 version2 で作成したプロジェクトをインポートする方法については 3.3.6. GNU17 Version 2 用プロジェクトのインポートを参照してください。

インポートの手順は次のとおりです。

(1) 以下のいずれかの操作を行います。

- [File]メニューから[Import...]を選択します。
- [Project Explorer]ビューのコンテキストメニューから[Import...]を選択します。

[Import]ウィザードが起動します。

(2) 一覧から[Existing Project into Workspace]を選択し、[Next>]ボタンをクリックします。

(3) [Select root directory]ラジオボタンを選択し、[Browse...]ボタンをクリックして表示されるフォルダ選択ダイアログで、インポートするプロジェクトフォルダが存在するディレクトリを選択してください。

(4) プロジェクトを現在のワークスペースにコピーする場合、Options リストの[Copy projects into workspace]にチェックをしてください。インポート後の編集操作等がワークスペース内のファイルに対して行われるようになります。[Copy projects into workspace]をチェックしない場合、指定したプロジェクトフォルダをワークスペースから参照して使用します。

(5) [Finish]ボタンをクリックします。
[Project Explorer]ビューにインポートしたプロジェクトが表示されます。

(6) インポートしたプロジェクトの Target CPU を選択します。
[Properties]ダイアログ＞GNU17 Setting を表示し、Target CPU を確認します。

他の環境（PC）で作成したプロジェクトについても、プロジェクトフォルダごと現在の PC 上にコピーしてインポートすると、ビルドを実行することができます。ただし、ビルドオプションで独自のインクルード検索パスやライブラリのパスを設定している場合は、インポート後に変更する必要があります。

3.3.6 GNU17 Version2 用プロジェクトのインポート

GNU17 Version 2（GNU17 v2.3.0 など）で作成したプロジェクトを、本パッケージでビルドする場合、プロジェクトのプロパティを全て設定し直す必要があります。本機能を利用することで設定し直す必要がなくなります。

(1) [File]メニューの[Import…]アイテムを選択します。
インポートダイアログ上で"GNU17 v2 Project"を選択して次へ進んでください。"GNU17 v2 Project Import Wizard"が起動します。

(2) 既存プロジェクトのフォルダを選択し、[Finish]を選択します。
新たなプロジェクトの src フォルダ以下に、既存プロジェクトのソースファイルがコピーされます。

(3) インポートしたプロジェクトの Target CPU を選択します。
[Properties]ダイアログ＞GNU17 Setting を表示し、Target CPU を確認します。

(4) 割り込み処理関数の名前を変更します。
起動処理ライブラリ crt0.o には、割り込みベクタテーブルと、ターゲット MCU がリセットされてから main 関数が呼び出されるまでの処理が用意されています。"3.3.4. 割り込みベクタとブート処理の記述"で説明しているように、crt0.o は割り込み処理関数の名前を_vectorXX_handler に固定しています。インポートしたプロジェクト中の割り込みベクタテーブルを参照し、割り込みベクタに登録された割り込み処理関数を、割り込みベクタ番号に対応する名前に変更してください。

(5) 既存ソースファイルから crt0.o と重複する処理を削除します。
既存ソースファイルから、crt0.o と重複する以下の処理を削除してください。

- ベクタテーブル
- data セクションの RAM へのコピーと bss セクションの初期化(_init_section)
- 標準ライブラリの初期化（_init_lib および_init_sys）

crt0.o の内容は、本パッケージに同梱されているソースコード utility/lib_src/crt0/crt0.c でご確認いただけます。

(6) ブート処理を整理します。
"3.3.4. 割り込みベクタとブート処理の記述"に従って、既存ソースファイルに定義されていたブート処理を整理し、スタックポインタの初期値を定義してください。

起動処理ライブラリ crt0.o を使用しない場合、(4)(5)(6)の手順は必要ありません。代わりに、次の作業を実施しなければなりません。

(a) プロジェクトにリンカスクリプトファイルを追加します。
既存のリンカスクリプトファイルをコピーしてプロジェクトに追加し、プロジェクトのプロパティのリンカオプション[Other options]で追加したリンカスクリプトファイルを指定してください。

(b) crt0.o をリンク対象から除外します。
プロジェクトのプロパティのリンカオプション[Do not use standard start files]をチェックして、crt0.o をリンク対象から除外してください。

(c) 既存プロジェクトの定義に従いリンカスクリプトファイルを変更します。
既存のリンカスクリプトの以下の設定を変更してください。

- エントリーポイント（ENTRY）にリセット割り込み処理関数を登録してください。
- .vector セクションに割り込みベクタテーブル（boot.o や vector.o の.rodata セクションに含まれている場合が多い）を配置してください。
- .vector セクションに配置した割り込みベクタテーブルを、.rodata セクションから除外してください。

(d) GDB コマンドファイルを変更します。
gdbsim.ini に c17 ttbr コマンドが記述されていれば、割り込みベクタテーブルを指すように変更します。例えば割り込みベクタテーブルが vector 配列であれば、c17 ttbr &vector と記述します。

(7) プロジェクトをビルドします。
プロジェクトをビルドし、コンソールビューでビルド結果を確認してください。

3.4 プロジェクトプロパティの設定

プロジェクトは、個々に各種のプロパティを持ち、[Properties]ダイアログでそれらを参照/設定できるようになっています。プロジェクトプロパティを設定することで、ビルドに必要なオプションやリンカスクリプトを指定することができます。

[Properties]ダイアログは次の手順で開きます。

(1) [Project Explorer]ビューで、プロジェクトを選択します。
[Project]メニューまたは上記ビューのコンテキストメニューから[Properties]を選択します。

(2) [Properties]ダイアログが表示されます。

3.4.1 GNU17 用プロジェクトプロパティの設定

S1C17 用プログラムをビルドするとき、開発するアプリケーションシステムのプロセッサの種類およびメモリ空間のサイズにより、ツールの起動コマンドオプションやリンクするライブラリを切り替える必要があります。IDE が自動で正しく切り替えるために、プロジェクトプロパティで正しいプロセッサの種類とメモリモデルを指定してください。
[Properties]ダイアログ＞GNU17 Setting で以下のように設定できます。通常はプロジェクトの新規作成時にターゲット CPU タイプ及び、メモリモデルを選択しますのでその後の選択は不要です。

<table>
<tr><th></th><th>変更箇所</th><th>設定方法</th></tr>
<tr><td>ターゲット機種の選択</td><td>"GNU17 Setting"
Target CPU</td><td>S1C17 MCUの名前を選択します。この項目の選択内容は環境変数GCC17_COPRO、GNU17_MODELXXXに反映されます。</td></tr>
<tr><td>メモリモデルの選択</td><td>"GNU17 Setting"
Memory Model</td><td>以下のメモリモデルから選択します。この項目の選択内容は環境変数GCC17_POINTERに反映されます。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>REGULAR</td><td>24ビットアドレスモード</td></tr>
<tr><td>SMALL</td><td>16ビットアドレスモード</td></tr>
</table></td></tr>
<tr><td>Cコンパイラの選択</td><td>"GNU17 Setting"
GCC Version</td><td>以下のバージョンから選択します。この項目の選択内容は環境変数GCC17_LOCに反映されます。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>4.9</td><td>GCC 4.9</td></tr>
<tr><td>6.4</td><td>GCC 6.4</td></tr>
</table></td></tr>
<tr><td>スタックポインタ初期値の設定</td><td>"GNU17 Setting"
SP register initial value</td><td>タックポインタの初期値を設定します。この項目に設定した値は、シンボル__START_stackの値として、シンボルファイルに反映されます。</td></tr>
<tr><td>フラッシュセキュリティ設定</td><td>"GNU17 Setting"
Flash Security Key</td><td>以下のパラメータを設定します。この項目の設定内容は環境変数GNU17_SECURITY_KEYに反映されます。
<table>
<tr><th>パラメータ</th><th>意味</th></tr>
<tr><td>Version</td><td>フラッシュセキュリティのバージョン</td></tr>
<tr><td>Password</td><td>ICにあらかじめ設定されているパスワード</td></tr>
</table></td></tr>
<tr><td>フラッシュプロテクト設定</td><td>"GNU17 Setting"
Flash Protect Bits</td><td>エリアごとにフラッシュプロテクトの有効･無効をチェックボックスで選択します。この項目の選択内容は環境変数GNU17_PTD_FILE、GNU17_PTD_OPTIONに反映されます。
<table>
<tr><th>状態</th><th>意味</th></tr>
<tr><td>Read protect : ON</td><td>データ読み出しを禁止します。ただし、CPUによる命令実行は可能です。</td></tr>
<tr><td>Read protect : OFF</td><td>データ読み出し可能です。</td></tr>
<tr><td>Write protect : ON</td><td>データ書き込みを禁止します。</td></tr>
<tr><td>Write protect : OFF</td><td>データ書き込み可能です。</td></tr>
</table></td></tr>
</table>

3.4.2 環境変数の設定

GNU17 Setting プロパティでの設定内容に応じて環境変数に値が設定されます。これらの値は直接変更することも可能です。
[Properties]ダイアログ＞C/C++ Build＞Environment に以下の値を設定します。

<table>
<tr><th></th><th>変更箇所</th><th>設定方法</th></tr>
<tr><td>Cコンパイラの選択</td><td>[Variable]
GCC17_LOC</td><td>以下の値を設定します。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>${GNU17_LOC}/gcc4</td><td>GCC 4を使用する</td></tr>
<tr><td>${GNU17_LOC}/gcc6</td><td>GCC 6を使用する</td></tr>
</table></td></tr>
<tr><td>メモリモデルの選択</td><td>[Variable]
GCC17_POINTER</td><td>以下の値を設定します。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td>24</td><td>REGULARモデル
（24ビットアドレスモード）</td></tr>
<tr><td>16</td><td>SMALLモデル
（16ビットアドレスモード）</td></tr>
</table></td></tr>
<tr><td>ターゲット機種の選択</td><td>[Variable]
GCC17_COPRO</td><td>以下の値を設定します。
<table>
<tr><th>値</th><th>意味</th></tr>
<tr><td></td><td>COPROを搭載しない機種
（S1C17701など）</td></tr>
<tr><td>¥¥M</td><td>乗算コプロセッサ搭載機種</td></tr>
<tr><td>¥¥MD</td><td>COPRO搭載機種</td></tr>
<tr><td>¥¥MD2</td><td>COPRO2搭載機種
（S1C17Wシリーズ）</td></tr>
</table></td></tr>
<tr><td>パス</td><td>[Variable]
GCC17_INC</td><td>ANSIライブラリのインクルードファイルのパスの設定をします。
${GCC17_LOC}¥include</td></tr>
<tr><td></td><td>GCC17_LIB</td><td>エミュレーションライブラリ、ANSIライブラリのパスの設定をします。
${GCC17_LOC}¥lib¥¥${GCC17_COPRO}¥¥${GCC17_POINTER}bit</td></tr>
<tr><td>ユーザーライブラリのリンク</td><td>[Variable]
GCC17_USER_LIBS</td><td>リンクするユーザーライブラリファイル名を登録します。
ライブラリは¥Project¥Debugフォルダへ置く必要があります。
この変数に登録されたライブラリのリンク順はプロジェクト中で生成したオブジェクトより、後になります。</td></tr>
<tr><td></td><td>GCC17_STARTUP_LIB</td><td>リンクするユーザーライブラリファイル名を登録します。
ライブラリは¥Project¥Debugフォルダへ置く必要があります。
この変数に登録されたライブラリのリンク順は、プロジェクト中で生成したオブジェクトより、前になります。</td></tr>
<tr><td>機種情報</td><td>[Variable]
GNU17_MODEL</td><td>GNU17Setting>Target CPUで選択した値が反映されます。</td></tr>
<tr><td></td><td>GNU17_MODEL_LOC</td><td>機種情報のパスの設定
${GNU17_LOC}¥mcu_model</td></tr>
<tr><td></td><td>GNU17_MODEL_NAME</td><td>GNU17 Settingプロパティの変更時に、GNU17_MODELに依存した値が設定されます。通常は変更しないでください。</td></tr>
<tr><td></td><td>GNU17_MODEL_RS</td><td>GNU17 Settingプロパティの変更時に、GNU17_MODELに依存した値が設定されます。通常は変更しないでください。</td></tr>
<tr><td></td><td>GNU17_MODEL_SIZE</td><td>GNU17 Settingプロパティの変更時に、GNU17_MODELに依存した値が設定されます。通常は変更しないでください。</td></tr>
<tr><td></td><td>GNU17_MODEL_TOP</td><td>GNU17 Settingプロパティの変更時に、GNU17_MODELに依存した値が設定されます。通常は変更しないでください。</td></tr>
<tr><td>フラッシュセキュリティ設定</td><td>[Variable]
GNU17_SECURITY_KEY</td><td>フラッシュセキュリティを使用するために、winmdc17に渡す起動時オプションを設定します。
<table>
<tr><th>起動時オプション</th><th>意味</th></tr>
<tr><td>-s1</td><td>フラッシュセキュリティのバージョンです。
GNU17_MODELに依存した値が設定されます。
通常は変更しないでください。</td></tr>
</table></td></tr>
</table>

		-s2 フラッシュセキュリティのパスワードを設定します。
フラッシュプロテクト設定	[Variable] GNU17_PTD_OPTION	フラッシュプロテクトを使用するために、ptd.exeに渡す起動時オプションを設定します。
	GNU17_PTD_FILE	フラッシュプロテクトを使用する場合に、プロテクトされたROMデータ（PSAファイル）のファイル名に追加される文字列（"_ptd"）を設定します。
Cygwin設定	CYGWIN	ファイルパスの形式についてCygwinが出力する警告を表示しない場合は、"nodosfilewarning"を設定します。

3.4.3 コンパイラパスの設定

コンパイラのパスは、次のように設定します。

[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings] > [Cross Settings]>[path]で以下のようにフォルダをフルパスで記述してください。

GCC4 を使用する場合	`C:¥EPSON¥GNU17V3¥gcc4`
GCC6 を使用する場合	`C:¥EPSON¥GNU17V3¥gcc6`

3.4.4 コンパイラオプションの設定

コンパイラのコマンドオプションは、次のように設定します。

[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings] > [Cross GCC Compiler]で以下のように設定できます。オプションの詳細は"4 C コンパイラ"を参照してください。

●Symbols

このページではコンパイラのマクロ定義オプションを設定します。

[Defined symbols (-D)] （デフォルト: なし）
マクロ名と置き換え文字を指定します。

●Includes

このページではコンパイラの検索パスオプションを設定します。

[Include paths (-I)] （デフォルト:`"${GCC17_INC}", ../inc`）
インクルードファイルの検索パスを設定します。

●Optimization

このページではコンパイラの最適化オプションを選択します。

[Optimization Level] （デフォルト: `-O1`）
最適化のレベルを選択します。
- `-O0`： 最適化を行いません。C プログラムの動作を行単位で確認する場合などに選択してください。
- `-O1`： コードのサイズと実行速度の最適化を行います。
- `-O2`： コードのサイズと実行速度の最適化を行います。(gcc4 のみ)
- `-O3`： `-O1` よりも更に、コードの実行速度の最適化を行います。(gcc4 のみ)
- `-Os`: `-O1` よりも更に、コードのサイズの最適化を行います

C コンパイラごとに選択可能な最適化オプションが異なります。最適化の詳細については、"4.3.2 コマンドラインオプション"を参照してください。

●Miscellaneous

このページではその他のコンパイラオプションを選択します。

[Other flags] （デフォルト: `-c -mpointer${GCC17_POINTER} -B${GCC17_LOC}`）
コンパイラフラグを追加できます。

3.4.5 リンカオプションの設定

リンカのコマンドオプションは、次のように設定します。

[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings] > [Cross GCC Linker]で以下のように設定できます。オプションの詳細は"7 リンカ"を参照してください。

●General

このページではデフォルトライブラリの利用などについて設定します。

[Do not use standard start files]（デフォルト: チェックなし）
この項目をチェックすると crt0.o をリンクしません。
GNU17 Version2 用プロジェクトをインポートした場合や、アセンブラのみでプログラムを記述する場合など、独自の起動処理を実装する場合に選択します。

[Do not use default libraries]（デフォルト: チェックなし）
この項目をチェックするとデフォルトライブラリをリンクしません。
libc.a と libgcc.a と libg.a がデフォルトライブラリです。これらのライブラリをリンクしない場合に選択します。

●Libraries

このページではリンクするライブラリ、ライブラリの検索パスを設定します。
ユーザ独自のライブラリは環境変数 GCC17_USER_LIBS に設定してください。

[Libraries]
リンクするライブラリを指定します。
libc.a/libgcc.a/libg.a はリンカのデフォルトライブラリとしてあらかじめ指定されているため、特に設定する必要はありません。

[Library serach path]
ライブラリを検索するパスの指定を行います。
デフォルト設定で以下のパスが設定されています。

```
${GCC17_LIB}
```

これはエミュレーションライブラリ、ANSI ライブラリの場所を示しています。

●Miscellaneous

このページではその他のリンカオプションを指定します。

[Linker flags] （デフォルト: `-gstabs -B${GCC17_LOC} -mrelax`）
リンカフラグを追加できます。

[Other options]
その他のリンカオプションを指定します。
デフォルト設定で MAP ファイル生成オプションが設定されています。

```
-Map=${ProjName}.map
```

独自のリンカスクリプトを指定する場合は`-T` オプションにてリンカスクリプトファイルを指定してください。
指定しない場合は、デフォルトのリンカスクリプトに従いリンクします。

例：プロジェクトフォルダ上にあるリンカスクリプトファイル `elf32c17.x` を指定する場合

```
-T ../elf32c17.x
```

3.4.6 アセンブラオプションの設定

アセンブラのコマンドオプションは、次のように設定します。

[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings] > [Cross GCC Assembler]で以下のように設定できます。オプションの詳細は"6 アセンブラ"を参照してください。

注：IDE はアセンブラソースをアセンブルする際に、C プリプロセッサを通すために `xgcc(-c -xassembler-with-cpp`

オプション指定)を介してアセンブラを起動します。

●General

[Assembler flags]（デフォルト: `-B${GCC17_LOC} -c -mpointer${GCC17_POINTER} -x assembler-with-cpp -Wa,--gstabs`）
アセンブラフラグを追加できます。

[Include paths] （デフォルト: `../inc`）
インクルードファイルの検索パスを設定します。

3.5 プログラムのビルド

プログラムのビルドとは、必要なソースをコンパイル/アセンブルし、ライブラリ等も含めてリンクして実行形式のオブジェクトファイルを生成することをいいます。ここではビルドの実行方法を説明します。

3.5.1 リンカスクリプトの編集

リンカスクリプトファイルは、実行ファイル（.elf）を構成するオブジェクトファイルの配置情報をリンカに渡すためのファイルです。

通常は指定せず、デフォルトのリンカスクリプトを使用します。デフォルトのリンカスクリプトは、ソースファイルを C 言語で記述し起動処理ライブラリ（crt0.o）とリンクすることを想定しています。以下のように想定と異なる場合はリンカスクリプトを作成して指定する必要があります。

- 基本レイアウトを変更する場合
- RAM エリアを複数の変数で共用する場合
- RAM 上でプログラムを実行する場合
- プログラムのエントリポイント（リセット例外ハンドラ）を_start 以外に変更する場合
- ベクタテーブルを crt0.o 以外に定義する場合
- GCC Version を 3.3 に指定する場合

一般に、ソースファイルをすべてアセンブリで記述した場合は、リンカスクリプトを作成して指定する必要があります。

●リンカスクリプトの新規作成

以下のフォルダにデフォルトのリンカスクリプトがあります。

¥GNU17V3¥sample¥sample_gcc6¥elf32c17.x

本ファイルをコピーしてプロジェクトフォルダ上に保存し、変更してください。
テキストエディタで新規作成することも可能です。

●リンカスクリプトの編集

リンカスクリプトの内容についてはテキストエディタで編集します。リンカスクリプトの内容の詳細は"7 リンカ"の 7.4.2. リンク例を参照してください。

●リンカスクリプトの指定

[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings]＞[Cross GCC Linker]＞Miscellaneous で指定できます。
-T オプションにてリンカスクリプトファイルを指定してください。指定しない場合は、デフォルトのリンカスクリプトに従いリンクします。

例：プロジェクトフォルダ上にあるリンカスクリプトファイル elf32c17.x を指定する場合

```
-T ../elf32c17.x
```

3.5.2 ビルドの実行

ソースファイルの作成、ビルドオプションの設定が終わると、ビルドを実行できます。
ビルドの実行方法を以下に示します。

●ワークスペース内の全プロジェクトのビルド

ワークスペース内のすべてのプロジェクトをビルドするには、以下の操作のいずれかを行います。

- [Project]メニューから[Build All]を選択します。
- ウィンドウツールバーの[Build All]ボタンをクリックします。

●選択したプロジェクトのビルド

プロジェクト別のビルドは次のように行います。

(1) [Project Explorer]ビューで、ビルドするプロジェクトを選択します。

(2) 以下のいずれかの方法でビルドを実行します。

- [Project]メニューから[Build Project]を選択します。
- [Project Explorer]ビューのコンテキストメニューから[Build Project]を選択します。

●ビルド処理

ビルドを開始すると、**IDE** は次のような処理を行います。

(1) エディタ上に保存されていないファイルがある場合は保存します。

(2) Build を実行します。以下のファイルが生成されます。

- ソースごとのオブジェクトファイル（<*ソース名*> .o）
- 実行形式オブジェクトファイル（<*プロジェクト名*> .elf）
- S レコード形式の psa ファイル(<*プロジェクト名*> .psa)
- 提出用データの PA ファイル(<*プロジェクト名*> .PA)
- Gang Programmer 用ユーザ設定・プログラムデータファイル（gpdata.bin）
- リンクマップファイル（<*プロジェクト名*> .map）

ビルド実行中は、各ツールを実行するコマンドラインが[Console]ビューに表示されます。
ビルド中に発生したエラーは[Problems]ビューで確認でき、そこからエディタ上のエラー発生箇所にジャンプすることができます。

3.5.3 クリーンとリビルド

ソースまたはインクルードファイルが変更されていない場合はオブジェクトファイル（*.o）は再生成されません。生成されているオブジェクトファイル（*.o）をすべて消去すると、再度すべてのソースからのビルド（リビルド）を実行できます。

●クリーン処理

(1) [Project Explorer]ビューで、リビルドするプロジェクトを選択します。

(2) [Project]メニューから[Clean...]を選択すると[Clean]ダイアログが表示されます。

(3) [Clean projects selected below]ラジオボタンを選択し、リストからクリーン（リビルド）するプロジェクトを選択します。
ワークスペース内のすべてのプロジェクトをリビルドする場合は、[Clean all projects]を選択します。

(4) [OK]ボタンをクリックします。
選択したプロジェクト内の生成されたオブジェクトファイルがすべて削除されます。

上記の他に、次の方法でもクリーンを実行できます。

(1) [Project Explorer]ビューで、クリーンするプロジェクトを選択します。

(2) [Project Explorer]ビューのコンテキストメニューから[Clean project]を選択します。

この方法の場合、ダイアログは表示されずにクリーンのみが実行されます。

意図的にリビルドを行う以外、通常はソースファイルまたはヘッダファイルを変更してもビルドを行うだけで済みますが、以下の場合にはリビルドが必要になります。

- プロジェクトプロパティを変更した場合

インクルードしているヘッダファイルを変更した場合

3.5.4 スタック使用量静的解析機能

“C17StackCounter” は、スタックが使用するメモリサイズを静的に解析する機能です。Build 時に、[Console]ビューに結果を表示することが可能です。

●設定

(1) プロジェクトを選択し、[Project]>[Propaties]を選択してください。

(2) [C/C++ Build]>[Settings]>[Build Steps]タブを選択します。
Post-build steps の[Command:]ボックスに以下の内容を、追記してください。
※元々の設定は消さずに、追記してください。

;"${GNU17_LOC}¥utility¥stack¥C17StackCounter" ${ProjName}.elf

●出力

設定後、Build を実行した場合、[Console]ビューに実行結果を表示します。[Estimation of Total Stack Size is]に示される値が、スタックのサイズ(byte)です。
以下は、その例です。

```
"C:¥¥EPSON¥¥GNU17V3¥¥utility¥¥stack¥¥C17StackCounter" sample_gcc6.elf
Stack Size of Normal Call Trees:
- _start1                       : 64 = 8 + 56
   > main                       :    56 = 4 + 52
   >> puts                      :    52 = 20 + 32
   >>> putc                    :    32 = 8 + 24
   >>>> write                   :    24
- read                          : 12
- _crt0_start0                  : 0
- emu_copro_process             : 0
- _crt0_vector_handler          : 0
- _crt0_exit                    : 0

Estimation of Total Stack Size is: 64
  = Normal Call Tree Max: 64
     + Interrupt Disable Call Tree Max: 0
     + Interrupt Enable    Call Tree Sum: 0
```

●制限事項

以下の場合は、解析ができません。

- アセンブラを用いて、gcc と異なる関数出口処理を記述した場合
- 飛び先アドレスをテーブルや変数で管理している場合（ベクターテーブルを除く）

解析に失敗した場合

[Stack Size of Unsolved Functions and its Callees]に紐づけされなかった関数群が表示され、以下のメッセージが表示されます。

Note: This program contains some unsolved calls.

-

3.6 プログラムのデバッグ

ビルドによって実行ファイル（.elf）が生成され、前節までの準備が完了すると、デバッグを開始できます。
デバッグの開始は、起動構成画面から行います。
起動構成画面では、デバッグを開始するための各種設定を行い、デバッガ(GDB)を起動します。

3.6.1 GDB コマンドファイル

デバッガの起動時に実行するコマンドは、プロジェクト中の GDB コマンドファイルに記述しておくことができます。
プロジェクトには以下の２種類の GDB コマンドファイルが存在します。

gdbsim.ini	シミュレータ用 GDB コマンドファイル
gdbmini2.ini	ICDmini2(S5U1C17001H2)用 GDB コマンドファイル
gdbmini3.ini	ICDmini3(S5U1C17001H3)用 GDB コマンドファイル

どの GDB コマンドファイルを使用するかは、"Debug configuration"ウィンドウで選択できます。

例：GDB コマンドファイル［gdbmini3.ini］

```
c17 model_path c:/EPSON/GNU17V3/mcu_model
c17 model 17W23
target icd icdmini3
load
# Please uncomment following commented out lines to enable STDOUT while debugging.
# c17 stdout 1 WRITE_FLASH WRITE_BUF
# Please uncomment following commented out lines to enable STDIN while debugging.
# c17 stdin 1 READ_FLASH READ_BUF
# Please uncomment following commented out lines to enable LCD panel simulator while
debugging.
# c17 lcdsim on
```

ICDmini を使用してデバッグする場合、プロジェクト中の ICDmini に合った GDB コマンドファイル gdbminix.ini を編集して、ターゲット機種を指定してください。

c17 model_path コマンドでは MCU 機種別情報フォルダを指定します。GNU17 が c:¥EPSON¥GNU17V3 にインストールされている場合、次のように指定してください。

```
c17 model_path c:/EPSON/GNU17V3/mcu_model
```

c17 model コマンドではターゲット機種を指定します。ターゲット機種が S1C17W23 である場合、次のように指定してください。

```
c17 model 17W23
```

gdbsim.ini を選択して、シミュレータを使用してデバッグする場合、上記の設定は必要ありません。

ターゲット MCU にフラッシュセキュリティが設定されている場合、target コマンドでターゲット MCU と接続したのち、load コマンドを実行する前に、c17 pwul コマンドを実行して、フラッシュセキュリティのパスワードを解除しなければなりません。フラッシュセキュリティのバージョンが M03 で、設定されているパスワードが"ABCD1234"である場合、次のように指定してください。

```
C17 pwul M03 ABCD1234
```

このほかにデバッガ起動時に実行したいコマンドがあれば、ファイルを編集して追加してください。
デバッガコマンドについては"8 デバッガ"を参照してください。

3.6.2 標準入出力の設定

プログラム中からの入出力関数により GDB のコンソールウィンドウに文字列の出力、また専用の入力ウィンドウから文字列の入力をすることができます。
以下の設定が必要になります。

- GDB コマンドファイル内の赤字部分のコメント＃を外して有効にしてください。

例：GDB コマンドファイル［gdbmini3.ini］

```
# Please uncomment following commented out lines to enable STDOUT while debugging.
c17 stdout 1 WRITE_FLASH WRITE_BUF
# Please uncomment following commented out lines to enable STDIN while debugging.
c17 stdin 1 READ_FLASH READ_BUF
```

注：本機能を有効にするとハードウェアブレークポイントを 1 本占有します。

3.6.3 デバッガの操作

●デバッグ開始

現在のプロジェクトで最初にデバッガを起動する場合の手順は次のとおりです。

(1) [Run]メニューから[Debug Configration...]を選択して起動構成ダイアログを表示させます。ツールバーの [Debug]ボタンのメニューからも表示させることができます。

(2) C/C++ Application タイプを選択し、プロジェクト名に対応した Debug configuration を選択します。

(3) Debugger タブの GDB Command file にてプロジェクト中の GDB コマンドファイルを選択します。
gdbsim.ini： シミュレータモードでデバッグを行う場合
gdbmini2.ini： ICDmini2(S5U1C17001H2)を使用してデバッグを行う場合
gdbmini3.ini： ICDmini3(S5U1C17001H3)を使用してデバッグを行う場合

(4) Debugger タブの Stop on startup at でデバッグを開始するシンボルを指定します。C 言語で記述されたプログラムに対応するために、main 関数があらかじめ指定されています。

(5) [Debug]ボタンをクリックし、デバッグを開始します。

Debug configuration の詳細については、"3.6.4 Debug configuration の設定"を参照してください。

●プログラムのデバッグ

デバッグ中のプログラムの動作は[Run]メニューから操作します。
以下のような、GNU17 デバッガ独自の操作（gdb の c17 コマンド）については [C17]メニューの[Debug Command]から選択して実行します。

コマンド	機能
c17 rst	リセット
c17 rstt	ターゲットリセット
c17 int	割り込み
c17 intclear	割り込み解除
c17 tm	トレースモード設定
c17 chgclkmd	DCLK切替モード

注：c17 ttbr, c17 pwul, c17 model_path, c17 model コマンドは、デバッガ起動時に実行する GDB コマンドファイルに記述してください。

また、以下のビューを用いて、デバッグ中のプログラムの状態を調べることができます。

ビュー	機能
Breakpoints	ワークスペース内のすべてのプロジェクトのブレークポイントの設定を確認・変更できます。
Console	以下の出力を確認できます。 ・ ビルド実行中に各ツールが出力したメッセージ ・ GDBコマンドファイルの実行結果 ・ gdbのc17コマンドの実行結果 ・ 実行中のプログラムの標準出力
Disassembly	プログラムの逆アセンブリを確認できます。
EmbSys Registers	周辺回路レジスタの値を確認・変更できます。
Expressions	ワークスペース内のすべてのプロジェクトについて、任意の監視式（グローバルシンボルやレジスタ）を登録し、その値を確認できます。
Memory Browser	メモリ内容を確認・変更できます。
Registers	CPUレジスタの値を確認・変更できます。
Variables	変数の値を確認・変更できます。 ローカル変数は、プログラムの実行状態に応じて、自動的に表示されます。グローバル変数は、登録した場合に表示されます。

●デバッガの終了

以下のいずれかの方法でデバッグを終了することができます。
デバッグ終了後、[デバッグ]ビューの表示が終了状態に変わります。

- [Run]メニューから[Terminate]を選択します。
- [デバッグ]ビューの[Terminate]ボタンをクリックします。
- [コンソール]ビューの[Terminate]ボタンをクリックします。
- [デバッグ]ビューのコンテキストメニューから[Terminate]を選択します。

3.6.4 Debug configuration の設定

Program Type が Application である GNU17 プロジェクトを作成すると、生成するプログラムを実行するための Debug configuration が同時に作成されます。プロジェクトの作成後に、Debug configuration を追加作成するには、[File]メニューから[New] > [GNU17 Debug Configuration]を選択し、[GNU17 Debug Configuration] ウィザードを開始させます。

ワークスペース内のすべての Debug configuration は、[Run]メニューから[Debug configurations]を選択し Debug Configuration ダイアログを表示して、以下のように設定できます。

●Debugger タブ

[GDB debugger] （デフォルト: ..¥gdb）
使用するデバッガ（gdb）のパスを指定します。

[GDB command file] （デフォルト: gdbsim.ini）
デバッガの起動時に実行する GDB コマンドファイルを指定します。

●Source タブ

[Source Lookup Path]
シンボリックデバッグ時に用いるソースファイルの検索パスを設定します。デフォルトではプロジェクトフォルダ以下と GNU17 のライブラリソースコードが設定されています。

3.7 IDE がプロジェクト内に作成するファイル

表 3.7.1 IDE 生成ファイル一覧

ファイル名	ファイル内容	編集可	要管理ファイル
.project	IDEプロジェクトファイル	×	○
.cproject	IDEプロジェクトファイル（CDT）	×	○
<*プロジェクト名*> Debug.launch	GDB起動設定用ファイル	×	○
gdbmini2.ini	GDBコマンドファイル （ICDmini2）	○	○
gdbmini3.ini	GDBコマンドファイル （ICDmini3）	○	○
gdbsim.ini	GDBコマンドファイル（シミュレータ）	○	○
gpdata.ini	gpdataオプション設定（Gang Programmer）	○	○
ldsyms.ini	リンカシンボルファイル	○	○
¥.settings	プロジェクト設定格納ディレクトリ	×	○
¥Debug ¥<*プロジェクト名*>.elf	実行ファイル	×	×
¥Debug ¥<*プロジェクト名*>.map	マップファイル	×	×
¥Debug ¥<*プロジェクト名*>.psa	ROMデータ	×	×
¥Debug ¥<*プロジェクト名*>_ptd.psa	フラッシュプロテクト設定されたROMデータ	×	×
¥Debug ¥<*プロジェクト名*>.sa	S3形式実行ファイル	×	×
¥Debug ¥<*プロジェクト名*>.saf	S3形式実行ファイルの空き領域を0xff詰めしたファイル	×	×
¥Debug ¥<*プロジェクト名*>.PA	提出用データ	×	×
¥Debug¥gpdata.bin	Gang Programmer用ユーザ設定・プログラムデータファイル	×	×
¥Debug ¥<*プロジェクト名*>.o	オブジェクトファイル	×	×

要管理ファイルとは、ソース管理アプリケーションなどで管理する必要のあるファイル等を指します。

4 C コンパイラ

この章では C コンパイラ **xgcc** の使用方法とアセンブリソースとのインタフェースに関する内容を説明します。
C コンパイラの標準機能や C ソースの文法については、ANSI C を解説している一般の書籍をご覧ください。

4.1 機能

C コンパイラ **xgcc** は C ソースファイルをコンパイルし、S1C17 コア命令セットのニーモニック、拡張命令、アセンブラの擬似命令を含むアセンブリソースファイルを生成します。**xgcc** は ANSI C 準拠の GNU C コンパイラです。特殊な文法はサポートしていませんので、他機種用のプログラムの移植なども容易に行えます。
また、強力なオプティマイズ能力を持ち、非常にコンパクトなコードを生成しますので、組み込み用アプリケーションの開発に最適です。
C コンパイラは **xgcc.exe**、**cpp.exe**、**cc1.exe** の 3 つのファイルで構成されています。

xgcc は Free Software Foundation, Inc.の C コンパイラをベースとしています。GCC version 6 を移植した C コンパイラを本パッケージの gcc6 フォルダで、GCC version 4 を移植した C コンパイラを gcc4 フォルダで提供しています。ライセンスに関してはテキストファイル"COPYING3"、"COPYING"に記述されていますので、ご使用前にご覧ください。

4.2 入出力ファイル

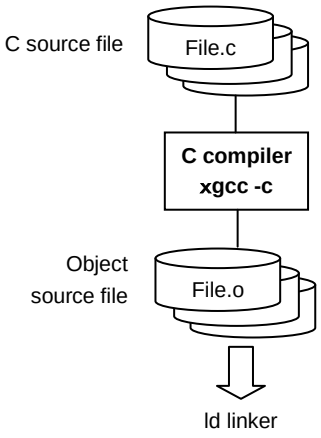


図 4.2.1 フローチャート

4.2.1 入力ファイル

●C ソースファイル

ファイル形式: テキストファイル
ファイル名: <*ファイル名*>.c
内容: C ソースプログラムを記述したファイルです。

4.2.2 出力ファイル

●アセンブリソースファイル

ファイル形式: テキストファイル
ファイル名: <*ファイル名*>.s
内容: アセンブラ as に入力するアセンブリソースファイルです。-S オプションを指定した場合に生成されます。

●オブジェクトファイル

ファイル形式: バイナリファイル
ファイル名: <*ファイル名*>.o
内容: リンカ **ld** に入力するリロケータブルオブジェクトファイルです。-c オプションを指定した場合に生成されます。

注: コマンドオプションの指定により、C コンパイラ xgcc は elf 形式の実行可能なオブジェクトファイルやプリプロセッサのみを通したソースファイルも出力できます。

4.3 起動方法

4.3.1 起動フォーマット

C コンパイラ **xgcc** は次のコマンドにより起動します。

xgcc *<オプション> <ファイル名>*

<*オプション*> 4.3.2 コマンドラインオプション参照
<*ファイル名*> C ソースファイルを拡張子（.c）も含めて指定します。

4.3.2 コマンドラインオプション

本パッケージの コンパイラは、以下のコマンドラインオプションを正式にサポートします。
ここに記載されているもの以外は動作保証対象外です。お客さまの責任で使用してください。

-c
機能: **リロケータブルオブジェクトファイルの出力**
説明: リロケータブルオブジェクトファイル（<*入力ファイル名*>.o）を出力するためのオプションです。
このオプションを指定すると、C コンパイラ **xgcc** はアセンブルが終了した時点で処理を中断し、リンクは行いません。このオプションを使用する場合、同時に-S または-E を指定しないでください。
デフォルト: C コンパイラ **xgcc** は elf 形式の実行可能なオブジェクトファイルを生成します。

-S
機能: **アセンブリコードの出力**
説明: アセンブリソースファイル（<*入力ファイル名*>.o）を出力するためのオプションです。
このオプションを指定すると、C コンパイラ **xgcc** はコンパイルが終了した時点で処理を中断し、アセンブルは行いません。このオプションを使用する場合、同時に-c または-E を指定しないでください。
デフォルト: C コンパイラ **xgcc** は elf 形式の実行可能なオブジェクトファイルを生成します。

-E
機能: **C プリプロセッサのみの実行**
説明: このオプションを指定すると、C コンパイラ **xgcc** はプリプロセッサの実行が終了した時点で処理を中断し、コンパイルやアセンブルは行いません。プリプロセッシング結果は標準出力に出力されます。
このオプションを使用する場合、同時に-S または-c を指定しないでください。
デフォルト: C コンパイラ **xgcc** は elf 形式の実行可能なオブジェクトファイルを生成します。

-B<*ディレクトリ*>
機能: **コンパイラの検索パス指定**
説明: C コンパイラ **xgcc** がサブプログラム（**cpp** や **cc1** など）やコンパイラ自体のデータファイルなどを探すための検索パスに<*ディレクトリ*>を追加します。
<*ディレクトリ*>は-B に続けて入力してください。複数のディレクトリを指定することができます。その場合は、-B<*ディレクトリ*>を必要数入力してください。検索は、コマンドライン上の出現順に行われます。ファイル検索の優先順位は、カレントディレクトリ、-B オプション、PATH の順です。
デフォルト: C コンパイラ **xgcc** はカレントディレクトリおよび PATH 指定ディレクトリ内のサブプログラムを検索します。

-I<*ディレクトリ*>

機能: **インクルードファイルディレクトリを指定**

説明: C ソース中にインクルードしたファイルが存在するディレクトリを指定します。
<*ディレクトリ*>は-I に続けて入力してください。複数のディレクトリを指定することができます。その場合は、-I<*ディレクトリ*>を必要数入力してください。インクルードファイルの検索は、コマンドライン上の出現順に行われます。
環境変数 C_INCLUDE_PATH にディレクトリが登録されていれば-I オプションは不要です。
ファイル検索の優先順位は、カレントディレクトリ、-I オプション、C_INCLUDE_PATH の順です。

デフォルト: C コンパイラ **xgcc** はカレントディレクトリおよび C_INCLUDE_PATH ディレクトリ内のインクルードファイルを検索します。

-D<*マクロ名*>[=<*置き換え文字*>]

機能: **マクロ名の定義**

説明: #define と同じ働きをします。**=**<*置き換え文字*>が指定されている場合はマクロにその値を定義します。指定されていない場合、マクロの値は 1 に設定されます。
<*マクロ名*>[**=**<*置き換え文字*>]は-D に続けて入力してください。複数のマクロ名を指定することができます。その場合は、-D<*マクロ名*>[**=**<*置き換え文字*>]を必要数入力してください。

※マクロ名の自動生成について

コンパイル時、以下のマクロ名が自動的に定義されます。これらのマクロ名はどのソースファイルからでも参照可能です。ただし、ユーザプログラムで同名のマクロを定義することはできません。

マクロ名	内容
__c17	S1C17 用にコンパイルされていることを示します。
__INT__	int 型のサイズ（16）を示します。
__POINTER24	コンパイラオプション-mpointer16 なしでコンパイルされていることを示します。
__POINTER16	コンパイラオプション-mpointer16 付きでコンパイルされていることを示します。

-O0,-O1,-O2,-O3,-Os

機能: **最適化**

説明: いずれか 1 つのスイッチを指定し、最適化処理を行います。
速度とサイズを重視して最適化されたコードが生成されます。 ただし、-O3 は速度のみを最適化の対象にしています。
-O の数値が上がるほど最適化機能は強化されますが、デバッグ情報が一部出力されないなどの問題が発生することがありますので、注意してください。
正常に実行できないときは、最適化の数値を下げてください。最適化の段階でレジスタのインターロックは考慮されません。-O3 は速度の最適化を目的としていますので、他の場合よりもサイズが大きくなることがあります。通常は、-O1 でコンパイルすることを推奨します。

以下に、各オプションの特徴を説明します。

-O0

最適化を行いません。
未使用のローカル変数が宣言された場合でも、スタックに領域を確保します。
コードはそのままコンパイルし、どこからも参照されないローカル変数に値を代入するというような、不要のコードもそのまま生成します。レジスタにロードした変数の値を再利用しません。ただし、register 宣言されたローカル変数は最適化され、不要であれば削除します。

-O1
コードのサイズと実行速度の最適化を行います。
実行される最適化としては、以下の点などが挙げられます。
不要なコードは削除します。（どこからも参照されないローカル変数に値を代入するなど）
変数の処理にレジスタを割り当て、その値を再利用することにより、メモリへのリード/ライトの回数を減らすようにします。
そのため、メモリへのアクセスの発生を保障できなくなりますので、確実にメモリのリード/ライトが必要な変数は volatile 宣言が必要になります。
ループ処理の最適化を実行します。
分岐条件を予測し最適化を行うことより、重複した比較命令を繰り返さないようにします。

-O2（gcc4 のみ）
-O1 より積極的に、コードのサイズと実行速度の最適化を行います。

-O3（gcc4 のみ）
-O1 よりも更に、コードの実行速度の最適化を行います。
-O1 よりも強化される最適化としては、以下の点などが挙げられます。
グローバル領域での共通の演算処理を一度の演算に置き換えます。（グローバル共通部分式の削除）
ループ処理の最適化を 2 回、実行します。
ld 命令などの単純な命令でのオペランドにおいて、レジスタの割り当ての最適化を行います。
到達することのない分岐条件ブロックを無視し、コードの生成を行いません。
inline 宣言のない関数のインライン展開を行い、単純なコードのサブルーチンはコールする代わりにその関数本体のコードをコピーします。これにより、関数コールのオーバーヘッドを削除します。
ただし、ソースコードの内容によっては、-O3 の結果が最速の実行速度にならないこともあります。

-Os
-O1 よりも更に、コードのサイズの最適化を行います。

デフォルト: コードの最適化が行われます。

-gstabs/-g

機能: **ソースファイルの相対パスを含むデバッグ情報の付加**

説明: デバッグ情報を含む出力ファイルが生成されます。
ソースファイルの位置情報は相対パスで出力されます。
Gcc4 の場合 :-gstabs
Gcc6 の場合 :-g

デフォルト: デバッグ情報が出力されます。

-fno-builtin

機能: **ビルトイン関数を無効化**

説明: 以下の関数がビルトイン関数としてコンパイルされず、必ず call されます。このオプションがない場合、コンパイラは以下の関数を状況に応じてインライン展開、もしくは他の関数への置き換えを行うことにより、コードの効率化を行う場合があります。
abort, abs, cos, exit, exp, fabs, fprintf, fputs, labs, log, memcpy, memmove, memset, printf, putchar, puts, scanf, sin, sprint, sqrt, sscanf, strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, vprintf, vsprintf

デフォルト: ビルトイン関数が有効になります。

-mpointer16

機能: **16 ビット（64KB）データ空間用コードの生成**

説明: データポインタを 16 ビット（データ空間は 64KB に限定）としてコードを生成します。
スタティック変数のポインタを格納する RAM サイズが節約できます。 ただし、これによってスタックを節約することはできません。

デフォルト: 24 ビット（16MB）空間にデータを配置可能なコードを生成します。

-mrelax

機能: **出力コードサイズの最適化**

説明: -mrelax オプションを指定すると、リンク時に ext 0 命令を削除して出力コードサイズを最適化します。

デフォルト: リンク時に ext 0 命令を削除しません。

-Wall

機能: **ワーニングオプションの有効化**

説明: 以下のワーニングオプションを全て有効にします。
これらのワーニングオプションは、"-Wno-" と付けることで個別に無効にすることができます。
例えば "-Wcomment" のワーニングのみを無効にしたい場合は、"-Wall" の後ろに
"-Wno-comment"を追加します。

"-Wchar-subscripts"
配列の添字の型が char 型である場合にワーニングを出力します。

"-Wcomment"
コメントの開始文字列である "/*" が "/*" で始まるコメントの中にある場合にワーニングを出力します。
または、"//" で始まるコメントがバックスラッシュで終わっている場合にワーニングを出力します。

"-Wformat"
printf 関数や scanf 関数などを呼び出した時に、変換文字列に対し引数が適切であるかを確認します。
また、変換文字列で指定された変換が妥当であるかを確認します。

"-Wimplicit-int"
変数や関数を宣言する時に型が指定されていない場合にワーニングを出力します。

"-Wimplicit-function-declaration"
宣言される前に関数を使用した場合にワーニングを出力します。

"-Wimplicit"
"-Wimplicit-int" 及び "-Wimplicit-function-declaration" を有効にしたものです。

"**-Wmain**"
main 関数の型が正しくない場合にワーニングを出力します。
main 関数は、外部リンケージを持ち、戻り値は int 型であり、0 個または 2 個または 3 個の適切な型の引数を持つべきとされています。

"-Wmissing-braces"
配列などを初期化する時に、十分に括弧でくくられていない場合にワーニングを出力します。
多次元配列を初期化する時に、各次元ごとに括弧でくくられていない場合などが該当します。
例：`long l_Array_1[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };`
（ワーニングを出力します）
`long l_Array_2[3][3] = { { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 } };`
（ワーニングを出力しません）

"-Wparentheses"
括弧が省略されて記述が混乱する場合に、ワーニングを出力します。
ネストした if 文などにおいて "{ }" が省略されている場合などが該当します。

"-Wsequence-point"
標準の C 言語仕様においては、実行する順序が正確に規定されていないため、振る舞いが未定義になる可能性のあるコードを記述した場合にワーニングを出力します。
例：`i_Array[i_Val++] = i_Val;`

"-Wreturn-type"
関数を定義する時に戻り値の型が指定されていない為、デフォルトの int 型として定義される場合にワーニングを出力します。または、戻り値が void 型以外の関数であるにも関わらず、値を返していない場合にワーニングを出力します。

"-Wswitch"
switch 文が enum 型の変数をインデックスとする場合に、enum 型の全ての値に対応する case 文がない場合にワーニングを出力します。(default ラベルがある場合は、このワーニングは出力されません。)または、enum 型の範囲を超えた値を case 文として指定した場合にワーニングを出力します。

"-Wunused-function"
static 関数が宣言されているにも関わらず定義されていない場合にワーニングを出力します。または、inline でない static 関数が定義されているにも関わらず未使用の場合にワーニングを出力します。

"-Wunused-label"
ラベルが宣言されているにも関わらず未使用の場合にワーニングを出力します。

"-Wunused-variable"
ローカル変数や const でない static 変数が宣言されているにも関わらず未使用の場合にワーニングを出力します。

"-Wunused-value"
使用されないことが明白にも関わらず、計算をする場合にワーニングを出力します。

"-Wunused"
上記の"-Wunused-xxxx" を全て有効にしたものです。

"-Wuninitialized"
ローカル変数が初期化されずに使用されている場合にワーニングを出力します。このワーニングは -O0 の時は出力されません。

デフォルト: 上記のワーニングオプションは無効になります。

-Werror-implicit-function-declaration

機能: **未宣言の関数をエラー出力**

説明: C ソースファイルで、宣言される前に関数を使用した場合にエラーを出力します。

デフォルト: C ソースファイルで、宣言される前に関数を使用した場合でも、エラーを出力しません。

-xassembler-with-cpp

機能: **C プリプロセッサの呼び出し**

説明: アセンブル前に C プリプロセッサ **cpp** を実行します。本オプションを付加することにより、アセンブラソース内で擬似命令（#define、#include など）を使用することが可能になります。

デフォルト: C プリプロセッサの呼び出しは行いません。

-Wa,*<オプション>*

機能: **アセンブラオプションの指定**

説明: 指定のオプションがアセンブラに渡されます。複数のオプションを指定することができます。その場合は、-Wa,*<オプション>*を必要数入力してください。

デフォルト: アセンブラに渡されるオプションはありません。

コマンドラインにオプションを入力する場合、オプションの前後には 1 個以上のスペースが必要です。

例: `xgcc -c -g test.c`

注: ・ 同じコンパイラオプションを異なる設定で複数指定した場合、動作は不定となります。

・ xgcc を呼び出す場合は、必ず -S、-E または -c オプションのいずれかを指定してください。これらのオプションが指定されない場合、xgcc はリンク処理まで実行しますので、必要なリンカのオプションも指定しなければなりません。

4.4 コンパイラの出力

ここではCコンパイラ **xgcc** が出力するアセンブリソースや使用するレジスタについて説明します。

4.4.1 出力内容

Cコンパイラ **xgcc** はCソースをコンパイルし、以下の内容を出力します。

- S1C17コア命令セットのニーモニック
- 拡張命令のニーモニック
- アセンブラ擬似命令

基本命令以外は拡張命令で出力されます。

システム制御命令などはCソースでは表現できませんので、asmによるインラインアセンブルまたはアセンブリソースファイルで対応してください。

例: `asm("halt");`

セクションやデータ定義にはアセンブラ擬似命令を出力します。
命令やデータが設定されるセクションは以下のとおりです。

命令

すべて.textセクションに配置されます。

定数

.rodataセクションに配置されます。

例: `const int i=1;`

```
        .global    i
        .section   .rodata
        .align     2
        .type      i,@object
        .size      i,4
i:
        .long      1
```

初期値を持つグローバル変数とスタティック変数

.dataセクションに配置されます。

例: `int i=1;`

```
        .global    i
        .section   .data
        .align     2
        .type      i,@object
        .size      i,4
i:
        .long      1
```

初期値を持たないグローバル変数とスタティック変数

.bssセクションに配置されます。

例: `int  i;`

```
        .global    i
        .section   .bss
        .align     2
        .type      i,@object
        .size      i,4
i:
        .zero      4
```

関数名やラベル等、すべてのシンボルにはアセンブラ擬似命令.stabによるシンボル情報が挿入されます(-gstabs/-gオプション指定時)。

4.4.2 データ表現

C コンパイラ **xgcc** は ANSI C のすべてのデータ型に対応しています。各型のサイズ（バイト数）、表現できる数値の有効範囲を表 4.4.2.1 に示します。

表 4.4.2.1 型の種類とサイズ

型	サイズ	数値の有効範囲
char	1	-128～127
unsigned char	1	0～255
short	2	-32768～32767
unsigned short	2	0～65535
int	2	-32768～32767
unsigned int	2	0～65535
long	4	-2147483648～2147483647
unsigned long	4	0～4294967295
pointer	4	0～16777215
float	4	1.175e-38～3.403e+38（正規化数）
double	8	2.225e-308～1.798e+308（正規化数）
long long	8	-9223372036854775808～9223372036854775807
unsigned long long	8	0～18446744073709551615
wchar_t	2	0～65535

`float` 型、`double` 型は IEEE 標準規格のフォーマットに準拠しています。

`long long` 型の定数を扱う場合は、接尾子 `LL` または `ll`(`long long` 型)／`ULL` または `ull`(`unsigned long long` 型)が必要になります。この接尾子がないと、コンパイラは `long long` 型の定数として認識できないため、ワーニングになります。

例：
```
long long ll_val;
    ll_val = 0x1234567812345678;
        → warning:integer constant is too large for “long” type
    Ll_val = 0x1234567812345678LL;
        → OK
```

`wchar_t` 型はワイド文字を扱うためのデータ型で、`stdlib.h`/`stddef.h` 内に `unsigned short` 型として定義されています。

●メモリ上の格納位置

データのメモリ上の格納位置はデータ型および格納する変数領域に依存します。各型の変数領域における格納位置を表 4.4.2.2 に示します。

表 4.4.2.2 型の種類と格納位置

型	グローバル変数領域	ローカル変数領域
char	1バイト境界	
short	2バイト境界	
int	2バイト境界	
long	4バイト境界	
pointer	4バイト境界	
long long	4バイト境界	
構造体	4バイト境界の型を含まない場合：2バイト境界	サイズが2バイト以下の場合：2バイト境界
	4バイト境界の型を含む場合：4バイト境界	サイズが3バイト以上の場合：4バイト境界
配列	4バイト境界の型を含まない場合：2バイト境界	要素数が1の場合：データ型の格納位置
	4バイト境界の型を含む場合：4バイト境界	要素数が2以上の場合：4バイト境界

構造体内での格納位置はデータ型に依存します。各型の構造体内での格納位置を表 4.4.2.3 に示します。

表 4.4.2.3 型の種類と格納位置

型	格納位置
char	1バイト境界
short	2バイト境界
int	2バイト境界
long	4バイト境界
pointer	4バイト境界
long long	4バイト境界
構造体	4バイト境界の型を含まない場合:2バイト境界
	4バイト境界の型を含む場合:4バイト境界
配列	データ型の格納位置

注:構造体及び配列の直後に配置した変数の格納位置は、上記の内容と一致しない場合があります。

●構造体データ

メンバは定義の出現順に各データ型のサイズに従って配置されます。
構造体の定義と配置の例を次に示します。

例:
```
struct Sample {
        char       cData;
        short      sData;
        char       cArray[3];
        long       lData;
};
```

図 4.4.2.1 構造体データのメモリ配置例

図のように、メンバのデータ型により未使用領域ができることがあります。
C言語の規格上、構造体又は共用体のメンバ変数の配置方法は処理系定義で調整することが許されています。
本パッケージのCコンパイラでは、処理系定義として構造体及び共用体のサイズは、必ず偶数バイトになるように調整されます。

●ビットフィールドのアクセス

ビットフィールドは定義された型に応じたサイズでアクセスされます。

4.4.3 レジスタ使用法

C コンパイラ **xgcc** は汎用レジスタを次のように使用します。

表 4.4.3.1 汎用レジスタの使用法

レジスタ	使用法
%r0	引数渡し用レジスタ（第1ワード） 戻り値格納用レジスタ（8/16ビットデータ、ポインタ、32ビットデータの下位16ビット）
%r1	引数渡し用レジスタ（第2ワード） 戻り値格納用レジスタ（32ビットデータの上位16ビット）
%r2	引数渡し用レジスタ（第3ワード）
%r3	引数渡し用レジスタ（第4ワード）
%r4	関数呼び出し時に値を保存する必要のあるレジスタ
%r5	
%r6	
%r7	

●引数渡し用レジスタ（%r0～%r3）

関数呼び出し時に引数を格納します。4ワードを超えた分はスタックに格納して渡します。引数を格納する前は、スクラッチレジスタとして使用されます。

%r0 ← 第1引数
%r1 ← 第2引数
%r2 ← 第3引数
%r3 ← 第4引数

32ビット（long）引数は、次のようにレジスタペアに格納します。
%r1（上位16ビット）と%r0（下位16ビット）
%r3（上位16ビット）と%r2（下位16ビット）

例:

- 第1引数がlong、第2引数がlongの場合

```
foo( long lData1, long lData2 );
```

%r0 ← lData1（下位16ビット）
%r1 ← lData1（上位16ビット）
%r2 ← lData2（下位16ビット）
%r3 ← lData2（上位16ビット）

- 第1引数がshort、第2引数がlongの場合

```
foo( short sData, long lData );
```

%r0	sData（16ビット）
%r1	lData（下位16ビット）
%r2	lData（上位16ビット）
%r3	未使用

- 第1引数がlong、第2引数がshort、第3引数がshortの場合

```
foo( long lData, short sData1, short sData2 );
```

%r0 ← lData（下位16ビット）
%r1 ← lData（上位16ビット）
%r2 ← sData1（16ビット）
%r3 ← sData2（16ビット）

- 第1引数がlong、第2引数がpointer、第3引数がpointerの場合
  ```
  foo( long lData, int *ip_Pt, char *cp_Pt );

  %r0 ← lData（下位16ビット）
  %r1 ← lData（上位16ビット）
  %r2 ← ip_Pt（24ビットまたは16ビット）
  %r3 ← cp_Pt（24ビットまたは16ビット）
  ```

64ビット（long longあるいはdouble）引数は、スタックに格納して渡します。

戻り値が64ビット（long long、double）の場合、呼び出し時に戻り値領域を確保し、その先頭アドレスを%r0に入れて関数に渡します。

●戻り値格納用レジスタ（%r0, %r1）
関数の戻り値を格納します。戻り値を格納する前は、スクラッチレジスタとして使用されます。

- 戻り値が8ビット/16ビットデータ、あるいはポインタ（24ビット）の場合
  ```
  %r0 ←  戻り値
  %r1    未使用
  ```

- 戻り値が32ビットデータの場合
  ```
  %r0 ← 戻り値（下位16ビット）
  %r1 ← 戻り値（上位16ビット）
  ```

●関数呼び出し時に保存するレジスタ（%r4～%r7）
式の計算結果やローカル変数が格納されます。これらのレジスタは、関数からリターンした際に呼び出し時の値を保持していなければなりません。したがって、呼び出された関数内でこれらのレジスタを変更する場合、その関数はsave/restoreによりレジスタの内容を退避/復帰する必要があります。

4.4.4 関数呼び出し

●引数の渡し方
関数呼び出しの際、引数は4つまでが引数渡し用レジスタ（%r0～%r3）に、それを超える分は呼び出し側関数のスタックフレーム（次節で説明）に格納されて呼び出される関数に渡されます。

●構造体引数の扱い
引数が64ビット以下の構造体の場合、引数渡し用レジスタ（%r0～%r3）を確保できるときは、構造体メンバの値を引数渡し用レジスタを介して渡します。引数渡し用レジスタ（%r0～%r3）を確保できないときは、構造体のメンバの値をスタックを介して渡します。
引数が64ビットを越える構造体の場合は、構造体のメンバの値をスタックを介して渡します。

4.4.5 スタックフレーム

C コンパイラ **xgcc** は関数呼び出し時に図 4.4.5.1 に示すスタックフレームを生成します。
スタックフレームの開始アドレスは常に 32 ビット境界アドレスとなります。

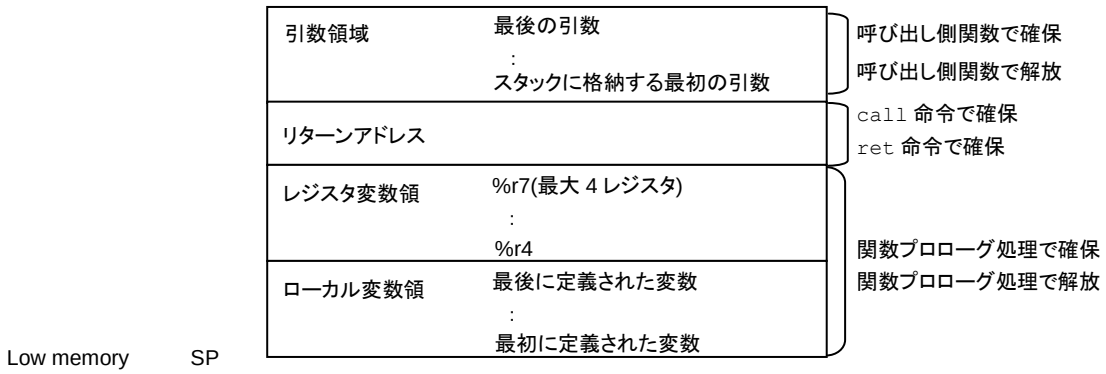


図 4.4.5.1 スタックフレーム

●引数領域
関数呼び出しの引数の中で、引数渡し用レジスタに格納できないものについては、スタックフレーム内に領域を確保します。引数はすべて4バイト境界に配置されます。

●リターンアドレス
呼び出し側の関数へのリターンアドレスです。

●レジスタ退避領域
%r4～%r7レジスタの中で、呼び出される側の関数が使用しているレジスタがあれば格納されます。
呼び出される側の関数が%r4～%r7レジスタを使用していない場合、この領域は確保されません。

●ローカル変数領域
呼び出された関数内で定義されているローカル変数で、レジスタに格納できないものについては、スタックフレーム内に領域を確保します。
スタックに割り当てるローカル変数がない場合、この領域は確保されません。

4.4.6 C ソースの文法

データ型、ライブラリ関数とヘッダファイル、インラインアセンブル、プロトタイプ宣言（割り込み処理関数の宣言）については、"2.2 C ソースの文法"を参照してください。

4.4.7 コンパイラの処理系定義

C 言語の規格上、構造体又は共用体のメンバ変数の配置方法は処理系定義で調整することが許されています。
本パッケージの C コンパイラでは、処理系定義として構造体及び共用体のサイズは、必ず偶数バイトになるように調整されます。

4.5 Shift JIS コードへの対応

GCC の基本的な文字コードは、UTF-8 です。そのため、SJIS として以下のような文字コードがあると、正しく処理が出来ません。

例：
SJIS として、"能" (= 0x945c) という文字があると、0x5c 部分を行連結子(¥) と誤って判断してしまいます。

```
i_Val = 0;      // 機能
i_Val = 1;      ← この行が上の行に連結されてしまい、コメントとして処理されてしまいます。
```

このような問題は、GCC が扱う文字コードとして SJIS を指定し対応させることで、解消可能です。

- **文字コードの指定方法**

[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings]＞[Cross GCC Compiler]＞[Dialect]＞[Other dialect flags] に **-finput-charset=CP932** を追加して下さい。

“-finput-charset”は、文字コードを指定するオプションになり、SJIS(CP932)を指定します。

4.6 xgcc の機能と注意事項について

- **xgcc** の詳細については GNU C コンパイラのドキュメントを参照してください。ドキュメントは、世界各地にある GNU のミラーサイトからインターネット等を利用して入手可能です。
- C コンパイラの既知の問題・制限事項などについては、本パッケージの readmeVxxx.txt およびリリース履歴（GNU17v3_release_history_j.pdf）を参照してください。
- C コンパイラが対応する C99 規格については、以下のサイトを参照してください。
  http://gcc.gnu.org/c99status.html

5 ライブラリ

この章では本パッケージに含まれるエミュレーションライブラリと ANSI ライブラリについて説明します。

5.1 ライブラリ概要

本パッケージ が提供するライブラリ全般についての概要について説明します。

5.1.1 ライブラリの構成

コンパイラのバージョン毎にライブラリを準備しています。

libc.a **ANSI ライブラリ**
ANSI 標準の関数を提供します。

libgcc.a **エミュレーションライブラリ**
単精度（32 ビット）および倍精度（64 ビット）浮動小数点の四則演算/比較/型変換、
整数の乗除算/シフト、long long 型の加減算/シフト関数を提供します。

libg.a **デバッグ用ライブラリ**
デバッガを補助する機能を提供します。

各コンパイラのライブラリは、コプロセッサ仕様とメモリモデルに応じて以下のフォルダにインストールされます。

```
¥EPSON
    ¥GNU17V3¥gcc*x*
        ¥lib¥24bit          全機種対応 24 ビットメモリ空間用ライブラリ
        ¥lib¥16bit          全機種対応 16 ビットメモリ空間用ライブラリ
        ¥lib¥M¥24bit        乗算コプロセッサ搭載機種対応 24 ビットメモリ空間用ライブラリ
        ¥lib¥M¥16bit        乗算コプロセッサ搭載機種対応 16 ビットメモリ空間用ライブラリ
        ¥lib¥MD¥24bit       COPRO 搭載機種対応 24 ビットメモリ空間用ライブラリ
        ¥lib¥MD¥16bit       COPRO 搭載機種対応 16 ビットメモリ空間用ライブラリ
        ¥lib¥MD2¥24bit      COPRO2 搭載機種対応 24 ビットメモリ空間用ライブラリ
        ¥lib¥MD2¥16bit      COPRO2 搭載機種対応 16 ビットメモリ空間用ライブラリ
```

C コンパイラおよびアセンブラの-mpointer16 オプションを指定したときは、16 ビット版のライブラリをリンクしてください。

5.1.2 ライブラリ追加時の注意事項

ライブラリには依存関係があります。
リンク対象のライブラリを記述する場合は、以下の順序で指定してください。

1. 追加ライブラリ
2. libc.a
3. libgcc.a
4. libc.a　　（2 との重複はエラーとはなりません。相互参照が可能です。）

オブジェクトファイル（もしくはライブラリ）は、リンカに渡すファイル順で自身よりも後方にあるライブラリのみ参照可能です。追加ライブラリを最後に指定した場合、追加ライブラリ内では他のライブラリをいっさい使用することができません。float や double 演算、ANSI ライブラリといった基本的な関数が使用できませんので、追加ライブラリは必ずエミュレーションライブラリ、ANSI ライブラリよりも前に配置するようにしてください。

例: 1. NG

```
ld.exe -T withmylib.x -o withmylib.elf boot.o libc.a libgcc.a libc.a mylib.a
```

mylib.a がエミュレーションライブラリ、ANSI ライブラリを使用している場合、必ずリンク時にエラーが出ます。

2. OK

```
ld.exe -T withmylib.x -o withmylib.elf boot.o mylib.a libc.a libgcc.a libc.a
```

リンク時にエラーは発生せず、mylib.a は正常にエミュレーションライブラリ、ANSI ライブラリを使用できます。

追加したライブラリ同士で依存関係を持つ場合は、基本的なライブラリを最後に配置するようにしてください。

例:　lib1.a　エミュレーションライブラリ、ANSI ライブラリのみを呼び出す
　　lib2.a　エミュレーションライブラリ、ANSI ライブラリに加え、 lib1.a を呼び出す
　　lib3.a　エミュレーションライブラリ、ANSI ライブラリに加え、 lib1.a、lib2.a を呼び出す

```
ld.exe -T withmylib.x -o withmylib.elf boot.o lib3.a lib2.a lib1.a libc.a libgcc.a libc.a
```

5.2 起動処理ライブラリ

5.2.1 概要

ターゲット MCU がリセットされてから main 関数が呼び出されるまでの処理およびベクタテーブルを、ライブラリ crt0.o にあらかじめ用意しています。crt0.o の処理内容は、本パッケージに同梱されているソースコード utility/lib_src/crt0/crt0.c を参照してください。

5.2.2 ベクタテーブル

ベクタテーブルは crt0.o に以下のようにあらかじめ登録されています。

```
void * const _vector[] = {
        VECTOR(_start),
        VECTOR(_vector01_handler),
        VECTOR(_vector02_handler),
        VECTOR(emu_copro_process),
        VECTOR(_vector04_handler),
        VECTOR(_vector05_handler),
        VECTOR(_vector06_handler),
        VECTOR(_vector07_handler),
         :
        VECTOR(_vector30_handler),
        VECTOR(_vector31_handler),
};
```

crt0.o はベクタテーブルに登録する割り込み処理関数の名前を_vectorXX_handler（XX は 10 進 2 桁のベクタ番号）に固定しています。"2.2 C ソースの文法"に従い、以下のように割り込み処理関数を宣言してください。

例：ベクタ番号 8 の割り込み処理関数を宣言する。

```
void _vector08_handler (void) __attribute__ ((interrupt_handler));
```

異なる名前で定義した割り込み処理関数を crt0.o のベクタテーブルに登録するには、以下のように宣言してください。

例：割り込み処理関数 sampleInterrupt をベクタ番号 8 に登録する。

```
void sampleInterrupt (void) __attribute__ ((interrupt_handler));
void _vector08_handler(void) __attribute__((alias("sampleInterrupt")));
```

5.2.3 スタックポインタ初期値

crt0.o はプログラムの開始直後に、シンボル__START_stack の値をスタックポインタの初期値に設定します。本パッケージの環境では、通常、__START_stack の値はリンカスクリプトで定義されています。
スタックポインタ初期値を変更するには、リンカスクリプトの定義を変更するか、次のように、C ソースファイルで設定するか、リンカシンボルファイルで設定します。

例：C ソースファイル上の任意の箇所でスタックポインタ初期値を 0x700 に設定する。

```
asm (".global __START_stack");
asm (".set __START_stack, 0x700"); /* initial value of SP register */
```

例：リンカシンボルファイルでスタックポインタ初期値を 0x700 に設定する。

```
__START_stack = 0x700; /* initial value of SP register */
```

スタックポインタ初期値は、プログラムのターゲット機種に適した値に設定してください。設定しない場合はリンカスクリプトの定義が、リンカスクリプトも指定しない場合はデフォルトリンカスクリプトの定義が有効になります。

5.2.4 起動処理

crt0.o は次に示す関数を順次呼び出します。これらの関数を独自に定義することで、起動処理を変更することが可能です。独自に定義しない場合は、crt0.o にデフォルトとして定義されている動作が実行されます。定義されている動作の概要を次の表に示します。具体的な実装は、本パッケージに同梱されているソースコード utility/lib_src/crt0/crt0.c を参照してください。

表 5.2.4.1 起動処理関数

実行順	起動処理関数	デフォルトの動作
1	void _init_device (void)	何もしない。
2	void _init_section (void)	dataセクションをRAMへコピーし、bssセクションを初期化する。
3	void _init_lib (void)	標準ライブラリを初期化する。
4	void _init_sys (void)	標準ライブラリ（入出力）を初期化する。
5	void _start_device (void)	ei命令により割り込みを許可する。
6	int main (void)	（なし）
7	void _stop_device (void)	di命令により割り込みを禁止する。
8	void _exit (int)	引数はmain関数の戻り値。 無限ループで停止する。

5.3 エミュレーションライブラリ

5.3.1 概要

本パッケージには、IEEE 形式に準拠した単精度（32 ビット）および倍精度（64 ビット）浮動小数点数の四則演算/比較/型変換、整数の乗除算/シフト、long long 型の加減算をサポートするエミュレーションライブラリ libgcc.a が含まれています。C コンパイラ **xgcc** は、浮動小数点演算、long long 演算、整数演算が行われると、これらのライブラリ内の関数を呼び出します。ライブラリ関数は、既定の汎用レジスタ/スタックを介してデータのやり取りを行いますので、アセンブリソースから呼び出すこともできます。エミュレーションライブラリ関数を使用する場合は、リンク時に libgcc.a と libc.a を指定してください。

ライブラリで使用しているレジスタ

%r0 から%r7 を使用しています。
%r4 から%r7 は関数の開始時にスタックに保存、終了時に復帰して保護されます。

5.3.2 浮動小数点演算関数

●関数一覧

表 5.3.2.1 に浮動小数点演算関数を示します。

表 5.3.2.1 浮動小数点演算関数一覧

分類	関数名	機能	
倍精度浮動小数点演算	__adddf3	加算	x ← a + b
	__subdf3	減算	x ← a - b
	__muldf3	乗算	x ← a * b
	__divdf3	除算	x ← a / b
	__negdf2	符号反転	x ← -a
単精度浮動小数点演算	__addsf3	加算	x ← a + b
	__subsf3	減算	x ← a - b
	__mulsf3	乗算	x ← a * b
	__divsf3	除算	x ← a / b
	__negsf2	符号反転	x ← -a
型変換	__fixunsdfsi	double → unsigned int	x ← a
	__fixdfsi	double → int	x ← a
	__floatsidf	int → double	x ← a
	__fixunssfsi	float → unsigned int	x ← a
	__fixsfsi	float → int	x ← a
	__floatsisf	int → float	x ← a
	__truncdfsf2	double → float	x ← a
	__extendsfdf2	float → double	x ← a
倍精度浮動小数点比較	__fcmpd	double型比較	PSR変更 ← a - b
	__eqdf2	double型比較 (a=b)	PSR変更 ← a - b, x ← 0 | 1 *
	__nedf2	double型比較 (a≠b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gtdf2	double型比較 (a>b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gedf2	double型比較 (a≧b)	PSR変更 ← a - b, x ← 0 | -1 *
	__ltdf2	double型比較 (a<b)	PSR変更 ← a - b, x ← -1 | 0 *
	__ledf2	double型比較 (a≦b)	PSR変更 ← a - b, x ← 0 | 1 *
単精度浮動小数点比較	__fcmps	float型比較	PSR変更 ← a - b
	__eqsf2	float型比較 (a=b)	PSR変更 ← a - b, x ← 0 | 1 *
	__nesf2	float型比較 (a≠b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gtsf2	float型比較 (a>b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gesf2	float型比較 (a≧b)	PSR変更 ← a - b, x ← 0 | -1 *
	__ltsf2	float型比較 (a<b)	PSR変更 ← a - b, x ← -1 | 0 *
	__lesf2	float型比較 (a≦b)	PSR変更 ← a - b, x ← 0 | 1 *

* 条件成立の場合x=左側の値、条件不成立の場合x=右側の値

- 演算結果がオーバーフローまたはアンダーフローした場合は、無限大または-無限大（次節参照）を返します。
- 比較関数は *op1 - op2*（a - b）の結果により、PSR の C、V、Z、N を次のように変更します。他のフラグは変更されません。

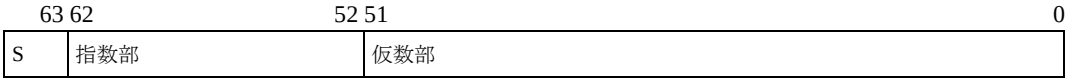
比較結果	C	V	Z	N
op1 > op2	0	0	0	0
op1 = op2	0	0	1	0
op1 < op2	1	0	0	1

●浮動小数点フォーマット

C コンパイラ **xgcc** は IEEE 標準規格に準拠した `float` 型（単精度浮動小数点、32 ビット）と `double` 型（倍精度浮動小数点、64 ビット）をサポートしています。
浮動小数点数の内部形式を以下に示します。

倍精度浮動小数点数のフォーマット

`double` 型の実数は次のように 64 ビットで構成されます。



ビット63:　　符号ビット（1ビット）
ビット62～52: 指数部（11ビット）
ビット51～0:　仮数部（52ビット）

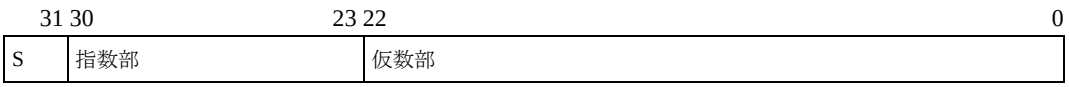
浮動小数点演算の結果は`%r0`が示すアドレスを先頭とする64ビット領域に格納されます。
`double`型の有効範囲、浮動小数点表記と内部データの対応は次のとおりです。

+0:	`0.0e+0`	`0x00000000 00000000`
-0:	`-0.0e+0`	`0x80000000 00000000`
最大正規化数:	`1.79769e+308`	`0x7fefffff ffffffff`
最小正規化数:	`2.22507e-308`	`0x00100000 00000000`
最大非正規化数:	`2.22507e-308`	`0x000fffff ffffffff`
最小非正規化数:	`4.94065e-324`	`0x00000000 00000001`
無限大:		`0x7ff00000 00000000`
-無限大:		`0xfff00000 00000000`

`0x7ff00000 00000001`～`0x7fffffff ffffffff`および`0xfff00000 00000001`～`0xffffffff ffffffff`は数値としては認められません。

単精度浮動小数点数のフォーマット

`float` 型の実数は次のように 32 ビットで構成されます。



ビット31:　　符号ビット（1ビット）
ビット30～23: 指数部（8ビット）
ビット22～0:　仮数部（23ビット）

レジスタに格納する場合は2つのレジスタを占有します。たとえば、浮動小数点演算の結果は次のように`%r1`および`%r0`レジスタに格納されます。

`%r1`レジスタ: 符号ビット、指数部および仮数部の上位7ビット（22:16）
`%r0`レジスタ: 仮数部の下位16ビット（15:0）

`float`型の有効範囲、浮動小数点表記と内部データの対応は次のとおりです。

+0:	`0.0e+0f`	`0x00000000`
-0:	`-0.0e+0f`	`0x80000000`
最大正規化数:	`3.40282e+38f`	`0x7f7fffff`
最小正規化数:	`1.17549e-38f`	`0x00800000`
最大非正規化数:	`1.17549e-38f`	`0x007fffff`
最小非正規化数:	`1.40129e-45f`	`0x00000001`
無限大:		`0x7f800000`
-無限大:		`0xff800000`

`0x7f800001`～`0x7fffffff`および`0xff800001`～`0xffffffff`は数値としては認められません。

注：無限大の扱い等を含め、IEEE 準拠の FPU とは精度および機能的な差異があります。

5.3.3 浮動小数点数処理の処理系定義

以下の処理はC言語の規格上、処理系定義となっており本パッケージのエミュレーションライブラリでは次のように処理しています。

浮動小数点丸め方法

整数型から浮動小数点型への型変換や、浮動小数点型から別の浮動小数点型への型変換、浮動小数点型の演算時において、目的の値が目的の型で表しうる二つの隣接した値の中間にある場合、どちらの値に丸めるかは処理系定義となります。
本パッケージでは、偶数となるように丸めています。
つまり、丸め前の値のLSBが0の場合は何も処理せず、LSBが1の場合は+1して切り上げます。

浮動小数点型から整数型への変換

浮動小数点型から整数型へ変換する際、小数部分は切り捨てとなります。
小数部分切捨て後、元の値が目的の型で表現できない場合の挙動は処理系定義となります。

・単精度/倍精度浮動小数点型からsigned / unsigned longへの変換
元の値が+NaNのとき → 目的の型がsignedであれば0x0、unsignedであれば0x80000000とします。
元の値が-NaNのとき → 0x0とします。
元の値が大きすぎるとき → 目的の型で表現可能な最大値とします。
元の値が小さすぎるとき → 0x80000000とします。

・単精度/倍精度浮動小数点型からsigned / unsigned long longへの変換
元の値が+NaNのとき → 0x80000000 80000000とします。
元の値が-NaNのとき → 目的の値がsignedであれば0x7fffffff 80000000、unsignedであれば0x0とします。
元の値が大きすぎるとき → 0xffffffff ffffffffとします。
元の値が小さすぎるとき → 目的の型がsignedであれば0x1、unsignedであれば0x0とします。

浮動小数点型から浮動小数点型への変換

浮動小数点型から別の浮動小数点型へ変換する際、元の値が目的の型で表現できない場合の挙動は処理系定義となります。

・倍精度浮動小数点型から単精度浮動小数点への変換
元の値が+NaNのとき → 倍精度浮動小数点の仮数部を2ビット左シフトし、上位32ビットを取得し、切り捨てた仮数部下位32ビットが0x0でなければさらに取得した仮数部32ビットのLSBを1とし、その値と0x7f900000を論理和したもの（+NaN）とします。
元の値が-NaNのとき → 倍精度浮動小数点の仮数部を2ビット左シフトし、上位32ビットを取得し、切り捨てた仮数部下位32ビットが0x0でなければさらに取得した仮数部32ビットのLSBを1とし、その値と0xff900000を論理和したもの（-NaN）とします。
元の値が大きすぎるとき → 0x7f800000（+∞）とします。
元の値が小さすぎるとき → 0xff800000（-∞）とします。
元の値が0に近すぎるとき（0より大） → 0x00000000（+0）とします。
元の値が0に近すぎるとき（0より小） → 0x80000000（-0）とします。

・単精度浮動小数点型から倍精度浮動小数点への変換
元の値が+NaNのとき → 単精度浮動小数点の仮数部を2ビット右シフトし、その値と0x7ff8000000000000を論理和したものとします。
元の値が-NaNのとき → 単精度浮動小数点の仮数部を2ビット右シフトし、その値と0xfff8000000000000を論理和したものとします。

5.3.4 整数演算/シフト関数

表 5.3.4.1 に整数演算/シフト関数を示します。

表 5.3.4.1 整数演算/シフト関数一覧

分類	関数名	機能	
整数演算	__divsi3	符号付き32ビット整数除算	x ← a / b
	__modsi3	符号付き32ビット整数剰余演算	x ← a % b
	__udivsi3	符号なし32ビット整数除算	x ← a / b
	__umodsi3	符号なし32ビット整数剰余演算	x ← a % b
	__mulsi3	32ビット乗算	x ← a * b
	__divhi3	符号付き16ビット整数除算	x ← a / b
	__modhi3	符号付き16ビット整数剰余演算	x ← a % b
	__udivhi3	符号なし16ビット整数除算	x ← a / b
	__umodhi3	符号なし16ビット整数剰余演算	x ← a % b
	__mulhi3	16ビット乗算	x ← a / b
整数シフト	__ashlsi3	32ビット算術左シフト	x ← a >> bビット
	__ashrsi3	32ビット算術右シフト	x ← a << bビット
	__lshrsi3	32ビット論理右シフト	x ← a << bビット
	__ashlhi3	16ビット算術左シフト	x ← a >> bビット
	__ashrhi3	16ビット算術右シフト	x ← a << bビット
	__lshrhi3	16ビット論理右シフト	x ← a << bビット
整数比較	__cmpsi2	比較(long)	x ← 2 | 1 | 0 *1
	__ucmpsi2	比較(unsigned long)	x ← 2 | 1 | 0 *1

5.3.5 long long 型演算関数

表 5.3.5.1 に long long 型演算関数を示します。

表 5.3.5.1 long long 型演算関数一覧

分類	関数名	機能	
long long型演算	__muldi3	符号付き64ビット乗算	x <-a * b
	__divdi3	符号付き64ビット除算	x <-a / b
	__udivdi3	符号なし64ビット除算	x <-a / b
	__moddi3	符号付き64ビット剰余演算	x <-a % b
	__umoddi3	符号なし64ビット剰余演算	x <-a % b
	__negdi2	符号反転	x <- -a
long long型シフト	__lshrdi3	64ビット論理右シフト	x <-a >> bビット
	__ashldi3	64ビット算術左シフト	x ← a << bビット
	__ashrdi3	64ビット算術右シフト	x <- a >> bビット
型変換	__fixunsdfdi	double → unsigned long long	x ←a
	__fixdfdi	double → long long	x ←a
	__floatdidf	long long → double	x ←a
	__fixunssfdi	float → unsigned long long	x ←a
	__fixsfdi	float → long long	x ←a
	__floatdisf	long long → float	x ←a
long long型比較	__cmpdi2	比較(long long)	x ←2 | 1 | 0 *1
	__ucmpdi2	比較(unsigned long long)	x ←2 | 1 | 0 *1

*1 整数比較関数及び、long long 比較関数は *op1 - op2* の結果により、以下の値を返します。
op1 > *op2* →2
op1 = *op2* →1
op1 < *op2* →0

5.3.6 コプロセッサ命令対応

S1C17 コアはコプロセッサ命令をサポートしています。
コプロセッサ命令に対応したライブラリを使用する場合は、以下のようにベクタテーブルの No.3 に"emu_copro_process" 関数を配置してください。

例：ベクタテーブルの指定方法(vector.c)

```
func *const vector[] = {
        VECTOR(boot),                   // 0
        VECTOR(unalign),                // 1
        VECTOR(dummy),                  // 2
        VECTOR(emu_copro_process)       // 3
};
```

起動処理ライブラリ crt0.o はベクタテーブルの No.3 に"emu_copro_process" 関数を配置済みです。

また、コプロセッサ命令には対応してある機種と対応してない機種がありますので、注意してください。
lib/M フォルダの libgcc.a は、乗算のコプロセッサ命令に対応したライブラリです。lib/MD フォルダの libgcc.a は、乗算/除算/剰余演算のコプロセッサ命令に対応したライブラリです。lib/MD2 フォルダの libgcc.a も同様ですが、対応機種が異なります。
表 5.3.6.1 にエミュレーションライブラリ内でコプロセッサ命令を使用している関数を示します。コプロセッサ命令を使用する場合、各関数は emu_copro_process を呼び出しています。emu_copro_process を呼び出す場合には、15～40 サイクル程度の割り込み禁止区間があります。
割り込み禁止区間の詳細は機種により異なりますので、必要に応じて、お使いの機種で emu_copro_process を動作させてご確認ください。

表 5.3.6.1 エミュレーションライブラリ内でコプロセッサ命令を使用している関数

関数名	機能	libgcc.a	M/libgcc.a	MD/libgcc.a	MD2/libgcc.a
__mulhi3	16ビット乗算	-	✓	✓	✓
__mulsi3	32ビット乗算	-	✓	✓	✓
__divhi3	符号付き16ビット除算	-	-	✓	✓
__modhi3	符号付き16ビット剰余演算	-	-	✓	✓
__udivhi3	符号なし16ビット除算	-	-	✓	✓
__umodhi3	符号なし16ビット剰余演算	-	-	✓	✓
__divsi3	符号付き32ビット除算	-	-	-	✓
__modsi3	符号付き32ビット剰余演算	-	-	-	✓
__udivsi3	符号なし32ビット除算	-	-	-	✓
__umodsi3	16ビット乗算	-	-	-	✓

5.4 ANSI ライブラリ

5.4.1 概要

本パッケージには、ANSI ライブラリが含まれています。

各関数は ANSI 標準の機能を持っています。ただし、一部の ANSI ライブラリ関数は本パッケージで対応していないため、ANSI ライブラリに含まれていません。"5.4.2 ANSI ライブラリ関数一覧"に記載されていない ANSI ライブラリ関数を使用する場合は、お客様の責任で関数の実装及び、プロトタイプ宣言を行ってください。

なお、本パッケージで未対応の ANSI ライブラリ関数についても、ヘッダファイル内でプロトタイプ宣言のみされているものもありますので、この場合はプロトタイプ宣言をする代わりに、該当するヘッダファイルをインクルードした上で関数の実装を行ってください。

プロトタイプ宣言のみされている ANSI ライブラリ関数については"2.2.2 ライブラリ関数とヘッダファイル"の表を参照してください。

ANSI ライブラリファイルは libc.a で、メモリモデル別に 24bit、16bit ディレクトリにインストールされます。

また、long long 型の ANSI ライブラリが libgcc.a に含まれています。

各関数に関する定義が記録されている以下のヘッダファイルが、include ディレクトリにインストールされます。

stdio.h stdlib.h time.h math.h errno.h float.h limits.h ctype.h string.h stdarg.h setjmp.h smcvals.h stddef.h

ライブラリで使用しているレジスタ

- %r0 から%r7 を使用しています。
- %r4 から%r7 は関数の開始時にスタックに保存、終了時に復帰して保護されます。

5.4.2 ANSI ライブラリ関数一覧

一覧表のリエントラント欄に記載されている記号の意味は以下のとおりです。

○ リエントラントな関数
× ノンリエントラントな関数
△ ノンリエントラントな関数（グローバル変数を参照しているものです。グローバル変数の変更がなく、お客さまが記述するread()およびwrite()がリエントラントな関数である場合、リエントラントな関数として使えます。）

●入出力関数

以下、libc.a に含まれる入出力関数の一覧を示します。

表 5.4.2.1 入出力関数一覧

ヘッダファイル: stdio.h

関数	機能	リエントラント	備考
size_t **fread**(void *ptr, size_t size, size_t count, FILE *stream);	stdinから配列要素を入力	△	グローバル変数stdin, _iobを参照、read関数をコール
size_t **fwrite**(void *ptr, size_t size, size_t count, FILE *stream);	stdoutから配列要素を出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int **fgetc**(FILE *stream);	stdinから1文字入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int **getc**(FILE *stream);	stdinから1文字入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int **getchar**(void);	stdinから1文字入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int **ungetc**(int c, FILE *stream);	入力バッファに1文字押し戻す	×	グローバル変数stdin, stdout, stderr, _iobを参照、戻した文字が上書きされる

関数	機能	リエントラント	備考
char *fgets(char *s, int n, FILE *stream);	stdinから文字列を入力	△	グローバル変数stdin, _iobを参照、read関数をコール
char *gets(char *s);	stdinから文字列を入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int fputc(int c, FILE *stream);	stdoutに1文字出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int putc(int c, FILE *stream);	stdoutに1文字出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int putchar(int c);	stdoutに1文字出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int fputs(char *s, FILE *stream);	stdoutへ文字列を出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int puts(char *s);	stdoutへ文字列を出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
void perror(const char *s);	stdoutへのエラー情報出力	×	グローバル変数stdout, _iobを参照、errnoを変更、read関数をコール
int fscanf (FILE *stream, const char *format,...);	stdinからの書式指定付き入力	×	グローバル変数stdout, _iobを参照、errnoを変更、read関数をコール
int scanf(const char *format, ...);	stdinからの書式指定付き入力	×	グローバル変数stdout, _iobを参照、errnoを変更、read関数をコール
int sscanf (const char *s, const char *format, ...);	文字列からの書式指定付き入力	×	グローバル変数errnoを変更
int fprintf(FILE *stream, const char *format,...);	stdoutへの書式指定付き出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int printf(const char *format, ...);	stdoutへの書式指定付き出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int sprintf(char *s, const char *format, ...);	配列への書式指定付き出力	○	write関数をコール
int vfprintf(FILE *stream, const char *format, va_list arg);	stdoutへの変換出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int vprintf(const char *format, va_list arg);	stdoutへの変換出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int vsprintf(char *s, const char *format, va_list arg);	配列への変換出力	○	write関数をコール

注: ファイルシステムは扱えません。stdin、stdout のみです。stdin は read、stdout は write 関数が必要です。詳細は 5.4.4 節を参照してください。

●ユーティリティ関数

以下、libc.a に含まれるユーティリティ関数の一覧を示します。

表 5.4.2.2 ユーティリティ関数一覧

ヘッダファイル: stdlib.h

関数	機能	リエントラント	備考
void *malloc(size_t size);	領域の確保	×	グローバル変数errno, ansi_ucStartAlloc, ansi_ucEndAlloc, ansi_ucNxtAlcP, ansi_ucTblPtr, ansi_ulRowを変更
void *calloc(size_t elt_count, size_t elt_size);	配列領域の確保	×	メモリアロケート中から呼び出しのため不正
void free(void *ptr);	領域の解放	×	メモリアロケート中から呼び出しのため不正
void *realloc(void *ptr, size_t size);	領域サイズの変更	×	メモリアロケート中から呼び出しのため不正
void exit(int status);	プログラムの正常終了	○	exitを参照 後から呼び出された側で終了
void abort(void);	プログラムの異常終了	○	exitを参照 後から呼び出された側で終了
void * bsearch(const void *key, const void *base, size_t count,size_t size, int (*compare)(const void *, const void *));	バイナリサーチ	○	
void qsort qsort(void *base, size_t count, size_t size,int (*compare)(const void *, const void *));	クイックソート	○	
int abs(int x);	絶対値を返す (int型)	○	
long labs(long x);	絶対値を返す (long型)	○	
div_t div(int n, int d);	int型除算	×	グローバル変数errnoを変更
ldiv_t ldiv(int n, int d);	long型除算	×	グローバル変数errnoを変更
int rand(void);	擬似乱数を返す	×	グローバル変数errnoを変更
void srand(unsigned int seed);	擬似乱数種の設定	×	グローバル変数errnoを変更
long atol(const char *str);	文字列をlong型に変換	×	グローバル変数errnoを変更
int atoi(const char *str);	文字列をint型に変換	×	グローバル変数errnoを変更
double atof (const char *str);	文字列をdouble型に変換	×	グローバル変数errnoを変更
double strtod(const char *str, char **ptr);	文字列をdouble型に変換	×	グローバル変数errnoを変更
long strtol(const char *str, char **ptr, int base);	文字列をlong型に変換	×	グローバル変数errnoを変更
unsigned long strtoul(const char *str, char **ptr, int base);	文字列をunsigned long型に変換	×	グローバル変数errnoを変更

●非局所分岐関数

以下、libc.a に含まれる非局所分岐関数の一覧を示します。

表 5.4.2.3 非局所分岐関数一覧

ヘッダファイル: setjmp.h

関数	機能	リエントラント	備考
int setjmp(jmp_buf env);	非局所分岐	○	
void longjmp(jmp_buf env, int status);	非局所分岐	○	

●日付と時刻関数

以下、libc.a に含まれる日付と時刻関数の一覧を示します。

表 5.4.2.4 日付と時刻関数一覧

ヘッダファイル: time.h

関数	機能	リエントラント	備考
struct tm ***gmtime**(const time_t *t);	カレンダー時間を標準時間に変換	×	スタティック変数を変更
time_t **mktime**(struct tm *tmptr);	標準時間をカレンダー時間に変換	×	ロケール情報と夏時間の設定は反映されません。
time_t **time**(time_t *tptr);	現在のカレンダー時間を返す	△	グローバル変数gm_secを参照

●数学関数

以下、libc.a に含まれる数学関数の一覧を示します。

表 5.4.2.5 数学関数一覧

ヘッダファイル: math.h, errno.h, float.h, limits.h

関数	機能	リエントラント	備考
double **fabs**(double x);	絶対値を返す(double型)	○	
double **ceil**(double x);	double型小数部の切り上げ	×	グローバル変数errnoを変更
double **floor**(double x);	double型小数部の切り捨て	×	グローバル変数errnoを変更
double **fmod**(double x, double y);	double型の剰余計算	×	グローバル変数errnoを変更
double **exp**(double x);	指数計算(e^x)	×	グローバル変数errnoを変更
double **log**(double x);	自然対数の計算	×	グローバル変数errnoを変更
double **log10**(double x);	常用対数の計算	×	グローバル変数errnoを変更
double **frexp**(double x, int *nptr);	浮動小数点数の仮数と指数を返す	×	グローバル変数errnoを変更
double **ldexp**(double x, int n);	仮数と指数から浮動小数点数を返す	×	グローバル変数errnoを変更
double **modf**(double x, double *nptr);	浮動小数点数の整数部と小数部を返す	×	グローバル変数errnoを変更
double **pow**(double x, double y);	x^yを計算	×	グローバル変数errnoを変更
double **sqrt**(double x);	平方根を計算	×	グローバル変数errnoを変更
double **sin**(double x);	正弦(サイン)計算	×	グローバル変数errnoを変更
double **cos**(double x);	余弦(コサイン)計算	×	グローバル変数errnoを変更
double **tan**(double x);	正接(タンジェント)計算	×	グローバル変数errnoを変更
double **asin**(double x);	逆正弦(アークサイン)計算	×	グローバル変数errnoを変更
double **acos**(double x);	逆余弦(アークコサイン)計算	×	グローバル変数errnoを変更
double **atan**(double x);	逆正接(アークタンジェント)計算	×	
double **atan2**(double y, double x);	y/xの逆正接(アークタンジェント)計算	×	グローバル変数errnoを変更
double **sinh**(double x);	双曲線正弦（ハイパボリックサイン）計算	×	グローバル変数errnoを変更
double **cosh**(double x);	双曲線余弦(ハイパボリックコサイン)計算	×	グローバル変数errnoを変更
double **tanh**(double x);	双曲線正接(ハイパボリックタンジェント)計算	×	

●文字関数

以下、libc.a に含まれる文字関数の一覧を示します。

表 5.4.2.6 文字関数一覧

ヘッダファイル: string.h

関数	機能	リエントラント	備考
void *memchr(const void *s, int c, size_t n);	記憶域内の指定文字の位置を返す	○	
int memcmp(const void *s1, const void *s2, size_t n);	記憶域の比較	○	
void ***memcpy**(void *s1, const void *s2, size_t n);	記憶域のコピー	○	
void ***memmove**(void *s1, const void *s2, size_t n);	記憶域のコピー(オーバーラップ可能)	○	
void *memset(void *s, int c, size_t n);	記憶域に文字を設定	○	
char *strcat(char *s1, const char *s2);	文字列の連結	○	
char *strchr(const char *s, int c);	文字列内で最初に現れた指定文字の位置を返す	○	
int strcmp(const char *s1, const char *s2);	文字列の比較	○	
char *strcpy(char *s1, const char *s2);	文字列のコピー	○	
size_t **strspn**(const char *s1, const char *s2);	文字列の先頭から指定文字(複数候補)が現れるまでの文字数を返す	○	
char ***strerror**(int code);	エラーメッセージの文字列を返す	○	
size_t strlen(const char *s);	文字列の長さを返す	○	
char ***strncat**(char *s1, const char *s2, size_t n);	文字列の連結(文字数指定付き)	○	
int **strncmp**(const char *s1, const char *s2, size_t n);	文字列の比較(文字数指定付き)	○	
char ***strncpy**(char *s1, const char *s2, size_t n);	文字列のコピー(文字数指定付き)	○	
char ***strpbrk**(const char *s1, const char *s2);	文字列内で最初に現れた指定文字(複数候補)の位置を返す	○	
char *strrchr(const char *str, int c);	文字列内で最後に現れた指定文字の位置を返す	○	
size_t **strcspn**(const char *s1, const char *s2);	文字列の先頭から指定文字(複数候補)以外が現れるまでの文字数を返す	○	
char ***strstr**(const char *s1, const char *s2);	指定文字列が最初に現れた位置を返す	○	
char *strtok(char *s1, const char *s2);	文字列をトークンに分割	×	スタティック変数を変更

●文字種判定/変換関数

以下、libc.a に含まれる文字種判定/変換関数の一覧を示します。

表 5.4.2.7 文字種判定/変換関数一覧

ヘッダファイル: ctype.h

関数	機能	リエントラント	備考
int **isalnum**(int c);	文字種判定(10進数またはアルファベット)	○	
int **isalpha**(int c);	文字種判定(アルファベット)	○	
int **iscntrl**(int c);	文字種判定(制御文字)	○	
int **isdigit**(int c);	文字種判定(10進数)	○	
int **isgraph**(int c);	文字種判定(図形文字)	○	
int **islower**(int c);	文字種判定(アルファベット小文字)	○	
int **isprint**(int c);	文字種判定(印刷可能文字)	○	
int **ispunct**(int c);	文字種判定(区切り文字)	○	
int **isspace**(int c);	文字種判定(空白文字)	○	
int **isupper**(int c);	文字種判定(アルファベット大文字)	○	
int **isxdigit**(int c);	文字種判定(16進数)	○	
int **tolower**(int c);	文字種変換(アルファベット大文字 → 小文字)	○	
int **toupper**(int c);	文字種変換(アルファベット小文字 → 大文字)	○	

●可変引数マクロ

以下、stdarg.h に定義された可変引数マクロの一覧を示します。

表 5.4.2.8 可変引数マクロ一覧

ヘッダファイル: stdarg.h

マクロ	機能
void **va_start**(va_list ap, type lastarg);	可変個引数集の初期化
type **va_arg**(va_list ap, type);	実引数の値を返す
void **va_end**(va_list ap);	可変個引数関数からの正常復帰

5.4.3 グローバル変数の宣言と初期化

ANSI ライブラリ関数は表 5.4.3.1 に示すグローバル変数を参照します。これらの変数は、crt0.o ライブラリ内に定義されており、同様に crt0.o に定義されている_init_lib()により起動時に初期化されます。

表 5.4.3.1 宣言が必要なグローバル変数

グローバル変数	初期設定	関連するヘッダファイル/関数

FILE _iob[FOPEN_MAX +1];
FOPEN_MAX=3, stdio.hにて定義
標準入出力ストリーム用
ファイル構造体データ
	_iob[*N*]._flg = _UGETN;
	_iob[*N*]._buf = 0;
	_iob[*N*]._fd = *N*;
	(*N*=0~2)
	_iob[0]: stdin用入力データ
	_iob[1]: stdout用出力データ
	_iob[2]: stderr用出力データ
		stdio.h, smcvals.h
		fgets, fread, fscanf, getc, getchar, gets, scanf, ungetc, perror, fprintf, fputs, fwrite, printf, putc, putchar, puts, vfprintf, vprintf

FILE *stdin; 標準入出力ファイル構造体 データ_iob[0]へのポインタ	stdin = &_iob[0];	**stdio.h** fgets, fread, fscanf, getc, getchar, gets, scanf, ungetc
FILE *stdout; 標準入出力ファイル構造体 データ_iob[1]へのポインタ	stdout = &_iob[1];	**stdio.h** fprintf, fputs, fwrite, printf, putc, putchar, puts, vfprintf, vprintf
FILE *stderr; 標準入出力ファイル構造体 データ_iob[2]へのポインタ	stderr = &_iob[2];	**stdio.h** fprintf, fputs, fwrite, printf, perror, putc, putchar, puts, vfprintf, vprintf
int errno; エラー番号を格納する変数	errno = 0;	**errno.h** fopen, freopen, fseek, fsetpos, perror, remove, rename, tmpfile, tmpnam, fprintf, printf, sprintf, vprintf, vfprintf, fscanf, scanf, sscanf atof, atoi, calloc, div, ldiv, malloc, realloc, strtod, strtol, strtoul acos, asin, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan
unsigned int seed; 乱数の種を格納する変数	seed = 1;	**stdlib.h** rand, srand
time_t gm_sec; タイマー関数の70年1月1日 0時0分0秒からの秒数	gm_sec = -1;	**time.h** time

ANSI ライブラリ関数より参照されるグローバル変数のうち、malloc、calloc、realloc、free の各関数で使用するグローバル変数は次の初期化関数をコールすることにより初期化されます。この関数は stdlib.h で宣言されています。

```
int ansi_InitMalloc(unsigned long START_ADDRESS, unsigned long END_ADDRESS);
```

START_ADDRESS には使用するメモリの先頭アドレスを、END_ADDRESS には終了アドレスを設定します。これらのアドレスは関数内で 4 バイト境界に調整されます。

この関数により、以下のグローバル変数が初期化されます。これらの変数は stdlib.h で宣言されています。

```
unsigned char *ansi_ucStartAlloc;     ヒープ領域の先頭アドレスを指すポインタ
unsigned char *ansi_ucEndAlloc;       ヒープ領域の終端アドレスを指すポインタ
unsigned char *ansi_ucNxtAlcP;        次に割り当てられる新規領域の先頭を指すアドレスポインタ
unsigned char *ansi_ucTblPtr;         次に割り当てられる管理領域の先頭を指すアドレスポインタ
unsigned long ansi_ulRow;             次に割り当てられる管理領域を指す行ポインタ
```

ヒープ領域を確保するごとに、終端アドレス（ansi_ucEndAlloc）から始まって、先頭方向に 8 バイトのヒープ領域の管理データが配置されていきます。したがって、ansi_InitMalloc()関数によりヒープ領域として指定したエリアは、スタックポインタなどによりデータが書き換えられないように注意してください。

* ansi_InitMalloc()関数は crt0.o 及び libstdio.a ライブラリに含まれていません。malloc()などをコールする前にユーザールーチンからコールする必要がありますので注意してください。
（ansi_InitMalloc()がコールされないと、ヒープ領域が確保されません。）

5.4.4 下位レベル関数

以下に示す 3 つの関数（read、write、_exit）はライブラリ関数が呼び出す下位レベル関数です。これらの関数についてはデバッグ用の実装を libg.a ライブラリ内に定義されています。

●read 関数

read 関数の内容

形式: **int read(int fd, char *buf, int nbytes);**

引数: int fd; 入力を示すファイル記述子
ライブラリ関数から呼ばれる場合、0（stdin）が渡されます。
char *buf; 入力データを格納するバッファへのポインタ
int nbytes; 転送バイト数

機能: ユーザが定義する入力用バッファからデータを最大 nbytes 分読み込み、buf で示されるバッファに格納します。

戻り値: 実際に入力用バッファから読み込んだデータのバイト数
入力用バッファが空（EOF）または nbytes が 0 のときは、0 を返します。
エラー発生時は、-1 を返します。

read 関数を呼び出すライブラリ関数:
直接コール: fread, getc, _doscan（_doscan は scanf 系内部関数）
間接コール: fgetc, fgets, getchar, gets（getc をコール）
scanf, fscanf, sscanf（_doscan をコール）

●write 関数

write 関数の内容

形式: **int write(int fd, char *buf, int nbytes);**

引数: int fd; 出力を示すファイル記述子
ライブラリ関数から呼ばれる場合、1（stdout）または 2（stderr）が渡されます。
char *buf; 出力データを格納するバッファへのポインタ
int nbytes; 転送バイト数

機能: buf で示されるバッファに格納されたデータを、ユーザが定義する出力用バッファに nbytes 分書き込みます。

戻り値: 実際に出力用バッファに書き込んだデータのバイト数
書き込みが正常に行われたときは nbytes を返します。
書き込みエラー発生時は、nbytes 以外の数値を返します。

write 関数を呼び出すライブラリ関数:
直接コール: fwrite, putc, _doprint（_doprint は printf 系内部関数）
間接コール: fputc, fputs, putchar, puts（putcc をコール）
printf, fprintf, sprintf, vprintf, vfprintf（_doprint をコール）
perror（fprintf をコール）

●_exit 関数

_exit 関数の内容

形式: **`void _exit(void);`**

機能: プログラムの終了処理を行います。

引数/戻り値:
なし

`_exit` 関数を呼び出すライブラリ関数:
直接コール: `abort`, `exit`

6 アセンブラ

この章ではアセンブラ as の機能を説明します。アセンブリソースの文法については"2.3 アセンブリソースの文法"を参照してください。

6.1 機能

アセンブラ **as** は C コンパイラが出力したアセンブリソースファイルをアセンブル（翻訳）し、機械語のオブジェクトファイルを生成します。シンボリックデバッグ用のデバッグ情報も出力可能です。

このアセンブラは GNU アセンブラ（as）をベースとしています。アセンブラ **as** の詳細については、GNU アセンブラのドキュメントを参照してください。ドキュメントは、世界各地にある GNU のミラーサイトからインターネット等を利用して入手可能です。

6.2 入出力ファイル

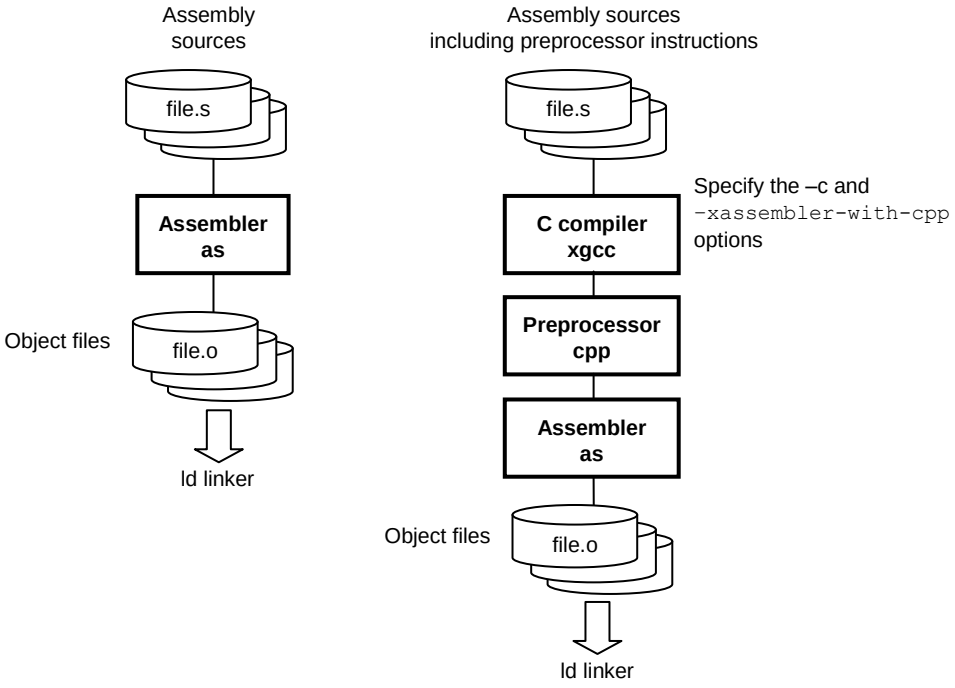


図 6.2.1 フローチャート

6.2.1 入力ファイル

●アセンブリソースファイル

ファイル形式: テキストファイル
ファイル名: <*ファイル名*>.s（".s"以外の拡張子が使用可能、パス指定も可能）
内容: 基本命令とアセンブラ擬似命令を記述したファイルです。通常、C コンパイラ **xgcc** が出力したファイルを入力します。
基本命令とアセンブラ擬似命令のみを記述したソースファイルを作成した場合は、直接アセンブラ **as** に入力することも可能です。

6.2.2 出力ファイル

●オブジェクトファイル

ファイル形式: elf 形式のバイナリファイル
ファイル名: <*ファイル名*>.o （<*ファイル名*>は入力ファイルと同じです。）
内容: プログラムコード（機械語）にシンボル情報やデバッグ情報等を付加したファイルです。

6.3 起動方法

6.3.1 起動フォーマット

アセンブラ **as** は次のコマンドにより起動します。

as ***<オプション> <ファイル名>***

<オプション> 6.3.2 項参照
<ファイル名> アセンブリソースファイルを拡張子（.s）も含めて指定します。

6.3.2 コマンドラインオプション

アセンブラ **as** は GNU アセンブラ標準のコマンドオプションを受け付けます。
ここでは、よく使用するオプションのみを説明しますので、その他のオプションも含め、詳細については GNU アセンブラのマニュアルを参照してください。

-o*<ファイル名>*

機能: **出力ファイル名の指定**
説明: アセンブラ **as** が出力するオブジェクトファイルの名称を指定します。
*<ファイル名>*は-o に続けて入力してください。
デフォルト: a.out という名称のファイルを生成します。

-a[*<サブオプション>*]

機能: **アセンブリリストファイルの出力**
説明: アセンブリリストファイルを出力します。*<サブオプション>*の指定により、出力内容も選択できます。
例: -adhl ハイレベルな出力と、デバッグ擬似命令の削除を指定します。
デフォルト: アセンブリリストファイルは出力されません。

--gstabs

機能: **ソースファイルの相対パスを含むデバッグ情報の付加**
説明: デバッグ情報を含む出力ファイルが生成されます。
ソースファイルの位置情報は相対パスで出力されます。
デフォルト: デバッグ情報は出力されません。

標準オプションに加え、以下の S1C17 専用のオプションも用意されています。

-mpointer16

機能: **16 ビットポインタモードの指定**
説明: 16 ビットポインタモード（64KB メモリモデル）用のオブジェクトファイルを出力します。実際には、16 ビットポインタモードを判別するためのフラグをセットするのみで、生成されるオブジェクトコードには影響を与えません。
デフォルト: 24 ビットポインタモード（16MB メモリモデル）用のオブジェクトファイルを出力します。

コマンドラインにオプションを入力する場合、オプションの前後には 1 個以上のスペースが必要です。

例: `as -otest.o -adhl test.s`

6.4 スコープ

各ソースファイル内で定義されたシンボルは、そのファイル内では自由に参照することができます。このようなシンボルの参照範囲をスコープといいます。

通常シンボルは定義されたファイル内でのみ参照可能です。そのファイル内に見つからないシンボルが参照されるとアセンブラ **as** はそのシンボルを未定義シンボルとしてオブジェクトファイルを作成し、解決をリンカ **ld** に任せます。

複数のソースファイルを使用する開発ではスコープを他のソースファイルにも広げる必要があり、アセンブラ **as** にはそのための擬似命令が用意されています。それによってシンボルのグローバル宣言を行い、他のソースファイルでも参照可能にします。

定義したファイル内でのみ参照可能なシンボルをローカルシンボル、グローバル宣言されたシンボルをグローバルシンボルと呼びます。ローカルシンボルの場合、複数のファイルで同一のシンボル名を指定しても、それぞれ別のシンボルとして扱われます。グローバルシンボルは複数のファイルで多重定義するとリンカ **ld** でワーニングとなります。

例: **file1**（グローバルシンボルを定義するファイル）

```
          .global  SYMBOL       ...シンボルを定義するファイルでグローバル宣言
          .global  VAR1
SYMBOL:
                   :
                   :
LABEL:                           ...ローカルシンボル
                   :                （このファイル内でのみ参照）
          .section .bss
          .align   2
VAR1:
          .zero    4
```

file2（グローバルシンボルを参照するファイル）

```
          xcall  SYMBOL         ...シンボルを外部参照
                   :
          xld.a  %r1,VAR1       ...シンボルを外部参照
LABEL:                           ...ローカルシンボル
                   :                （file1のLABELとは別のシンボルとして扱われます。）
```

アセンブラ **as** は file2 のシンボル SYMBOL および VAR1 をアドレス未定としてアセンブルし、その情報を出力するオブジェクトファイルに含めます。それらのアドレスはリンクの処理によって最終的に決定します。

6.5 アセンブラ擬似命令

アセンブラ擬似命令は、実行コードに変換される命令ではなく、アセンブルを制御したりデータを設定する命令です。

他の命令と区別するため、アセンブラ擬似命令はすべてピリオド（.）で始まります。

命令はすべて小文字で記述してください。パラメータは大文字と小文字が区別されます。

アセンブラ **as** では GNU アセンブラの擬似命令がすべて使用可能です。アセンブラの擬似命令の詳細については、GNU アセンブラのドキュメントを参照してください。

6.5.1 Text セクション定義擬似命令（.text）

●命令の形式

```
.text
```

●機能

.text セクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.text セクションとしてアセンブルされます。

6.5.2 Data セクション定義擬似命令（.rodata, .data）

●命令一覧

.rodata　定数を配置する.rodata セクションを宣言
.data　初期値を持つデータを配置する.data セクションを宣言

●命令の形式

```
.section .rodata
.section .data
```

●機能

(1) .section .rodata

定数データセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.rodata セクションに配置されるものとしてアセンブルされます。通常、このセクションはリンク時に読み出し専用メモリ（ROM）に配置します。

例: .section .rodata　　.rodata セクションを設定します。

(2) .section .data

初期値付きデータセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.data セクションに配置されるものとしてアセンブルされます。通常、このセクションはリンク処理で読み出し専用メモリ（ROM）に配置し、実行時はデータを使用する前にユーザプログラムで RAM などのリード/ライト可能なメモリに転送する必要があります。

例: .section .data　　.data セクションを設定します。

●注意事項

データ定義擬似命令で 1 つのデータに割り当てられるメモリのサイズは以下のとおりです。
1 バイト: .byte
2 バイト: .short, .hword, .word, .int
4 バイト: .long

6.5.3 Bss セクション定義擬似命令（.bss）

●命令一覧

.bss　　初期値を持たないデータを配置する.bss セクションを宣言

●命令の形式

```
.section .bss
```

●機能

非初期化データセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.bss セクションに配置されるものとしてアセンブルされます。

例: .section .bss　　.bss セクションを設定します。

●注意事項

- デフォルト設定では、.bss セクションに記載したラベルはすべてローカルシンボルとして定義されます。グローバルシンボルとして定義するには、.global 擬似命令を使用してください。

例:
```
        .section .bss
        .align    2
    VAR1:
        .skip     4     4バイトのローカル変数VAR1を定義

        .section .bss
        .global  VAR2
        .align    2
    VAR2:
        .skip     4     4バイトのグローバル変数VAR2を定義
```

- .bss セクション内では、.skip 擬似命令で領域が確保できます。.space 擬似命令は初期値を持つため使用できません。

6.5.4 データ定義擬似命令（.long, .short, .byte, .ascii, .space）

ここで説明する擬似命令は、.data セクションまたは.text セクションにデータを定義するために使用します。

●命令一覧

.long	4 バイトのデータを定義
.short	2 バイトのデータを定義
.byte	1 バイトのデータを定義
.ascii	ASCII 文字列を定義
.space	領域へのバイトデータ埋め込み

●命令の形式

.long	***<4 バイトデータ>***[, ***<4 バイトデータ>*** … , ***<4 バイトデータ>***]
.short	***<2 バイトデータ>***[, ***<2 バイトデータ>*** … , ***<2 バイトデータ>***]
.byte	***<1 バイトデータ>***[, ***<1 バイトデータ>*** … , ***<1 バイトデータ>***]
.ascii	"***<文字列>***"[, "***<文字列>***" … , "***<文字列>***"]
.space	***<バイト長>***[, ***<1 バイトデータ>***]

<4 バイトデータ>	0x0～0xffffffff
<2 バイトデータ>	0x0～0xffff
<1 バイトデータ>	0x0～0xff
<文字列>	ASCII 文字列
<バイト長>	埋め込む領域のサイズ

●機能

(1) .long, .short, .byte

4 バイトデータ、2 バイトデータ、1 バイトデータを定義します。2 つ以上のデータを指定する場合は、データ間をカンマで区切ります。定義したデータは直前に.align 擬似命令がない限り、定義するデータのサイズに従った境界アドレスから配置されます。現在位置が境界アドレス以外の場合、データを配置する境界アドレスまでの間は 0x00 が設定されます。

```
例: .long  0x0,0x1,0x2
    .byte  0xff
```

これらの擬似命令以外にも、以下の擬似命令が使用可能です。

.hword	.short と同機能
.word	.short と同機能
.int	.short と同機能

(2) .ascii

ASCII 文字列を定義します。文字列は二重引用符（"）で囲んで指定します。文字列には ASCII 文字および¥記号で始まるエスケープシーケンスが記述可能です。たとえば、文字列中に二重引用符を設定する場合は¥"、¥を設定する場合は¥¥と記述します。2 つ以上のデータを指定する場合は、データ間をカンマで区切ります。定義したデータは直前に.align 擬似命令がない限り、現在のアドレスから配置されます。

```
例: .ascii "abc","xyz"
    .ascii "abc¥"D¥"efg"  (= abc"D"efg)
```

(3) .space

*<バイト長>*で指定される領域に*<1 バイトデータ>*を埋め込みます。データを埋め込む領域は直前に.align 擬似命令がない限り、現在のアドレスから始まります。
*<1 バイトデータ>*の指定を省略すると 0x0 を埋め込みます。0x0 の埋め込みは.zero 擬似命令（次ページ参照）でも行えます。

```
例: .space  4,0xff      現在のアドレスから始まる 4 バイトの領域に 0xff を埋め込みます。
    .zero   4           (= .space 4,0x0)
```

6.5.5 領域確保擬似命令(.zero)

●命令の形式

.zero ***<バイト長>***

<バイト長> 領域のサイズ

●機能

現在の.bss セクションに*<バイト長>*で指定されるサイズの空白領域を確保します。直前に.align 擬似命令がない限り、現在のアドレスから領域を確保します。

例:
```
        .section .bss
        .global  VAR1
        .align   2
VAR1:
        .zero    4       4バイトのグローバル変数VAR1用の領域を確保
```

6.5.6 アライメント擬似命令（.align）

●命令の形式

.align *〈アライメント〉*

<*アライメント*>　境界を指定する数値

●機能

この擬似命令の直後に出現するデータを 2^n バイト境界にアライメントします（n=<*アライメント*>）。

例:.align 2　　4 バイト境界へのアライメント

●注意事項

.align 擬似命令は直後にあるデータ定義または領域確保擬似命令に対してのみ有効です。したがって、アライメントが必要なデータを定義する場合には、データ定義擬似命令個々に.align 擬似命令を使用する必要があります。

6.5.7 グローバル宣言擬似命令（.global）

●命令の形式

.global ***〈シンボル〉***

<*シンボル*>　現在のファイルで定義するグローバルシンボル

●機能

シンボルのグローバル宣言を行います。指定のシンボルは、他のモジュールから参照可能なグローバルシンボルに設定されます。

例:`.global  SUB1`

●注意事項

本擬似命令で宣言されないシンボルは、すべてローカルシンボルになります。

6.5.8 シンボル定義擬似命令（.set）

●命令の形式

.set ***〈シンボル〉,〈アドレス〉***

<*シンボル*>　メモリアクセス（アドレス参照）用のシンボル
<*アドレス*>　絶対アドレス

●機能

シンボルに絶対アドレス（24 ビット）を定義します。

例：.set　DATA1,0x80000　絶対アドレス 0x80000 を示すシンボル DATA1 を定義します。

●注意事項

設定したシンボルはローカルシンボルとなります。グローバルシンボルとして使用する場合は、.global 擬似命令によるグローバル宣言を行ってください。

6.6 拡張命令

アセンブラ **as** は以下に説明する拡張命令に対応しています。拡張命令は、通常 ext 命令を含む複数の命令で記述する内容を 1 つの命令として記述できるようにしたもので、命令の機能およびオペランドの即値サイズに従って必要最小限の基本命令に展開されます。

説明に使用するシンボル

immX	符号なし *X* ビット即値
signX	符号付き *X* ビット即値
symbol	メモリアドレスを示すシンボル
label	分岐先ラベル
(X:Y)	ビット *X* からビット *Y* のビットフィールド

6.6.1 算術演算命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
sadd %rd,*imm16*	%rd ← %rd+*imm16*	(1)
sadc %rd,*imm16*	%rd ← %rd+*imm16*+C	(1)
sadd.a %rd,*imm20*	%rd ← %rd+*imm20*	(2)
sadd.a %sp,*imm20*	%sp ← %sp+*imm20*	(2)
ssub %rd,*imm16*	%rd ← %rd-*imm16*	(1)
ssbc %rd,*imm16*	%rd ← %rd-*imm16*-C	(1)
ssub.a %rd,*imm20*	%rd ← %rd-*imm20*	(2)
ssub.a %sp,*imm20*	%sp ← %sp-*imm20*	(2)
xadd %rd,*imm16*	%rd ← %rd+*imm16*	(1)
xadc %rd,*imm16*	%rd ← %rd+*imm16*+C	(1)
xadd.a %rd,*imm24*	%rd ← %rd+*imm24*	(3)
xadd.a %sp,*imm24*	%sp ← %sp+*imm24*	(3)
xsub %rd,*imm16*	%rd ← %rd-*imm16*	(1)
xsbc %rd,*imm16*	%rd ← %rd-*imm16*-C	(1)
xsub.a %rd,*imm24*	%rd ← %rd-*imm24*	(3)
xsub.a %sp,*imm24*	%sp ← %sp-*imm24*	(3)

これらの拡張命令により、加減算で 16 ビット/20 ビット/24 ビット即値が直接指定可能となります。
拡張命令に、条件演算オプション（/c、/nc）は指定できません。

●展開後の基本命令

sadd, **xadd**	add 命令に展開
sadc, **xadc**	adc 命令に展開
sadd.a, **xadd.a**	add.a 命令に展開
ssub, **xsub**	sub 命令に展開
ssbc, **xsbc**	sbc 命令に展開
ssub.a, **xsub.a**	sub.a 命令に展開

●展開形式

(1) **s*OP* %rd,*imm16* / x*OP* %rd,*imm16*** （*OP* = add, adc, sub, sbc）

例: xadd %rd,*imm16*

imm16 ≤ 0x7f	0x7f < *imm16*
add %rd,*imm16*(6:0)	ext *imm16*(15:7) add %rd,*imm16*(6:0)

(2) **s*OP*.a %rd,*imm20* / s*OP*.a %sp,*imm20*** （*OP* = add, sub）

例: sadd.a %rd,*imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
add.a %rd,*imm20*(6:0)	ext *imm20*(19:7) add.a %rd,*imm20*(6:0)

(3) **x*OP*.a %rd,*imm24* / x*OP*.a %sp,*imm24*** （*OP* = add, sub）

例: xadd.a %rd,*imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
add.a %rd,*imm24*(6:0)	ext *imm24*(19:7) add.a %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) add.a %rd,*imm24*(6:0)

6.6.2 比較命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
scmp %rd,imm*16*	%rd-*imm16* （PSRのC, V, Z, Nフラグを変更）	(1)
scmc %rd,*imm16*	%rd-*imm16*-C （PSRのC, V, Z, Nフラグを変更）	(1)
scmp.a %rd,*imm20*	%rd-*imm20* （PSRのC, V, Z, Nフラグを変更）	(2)
xcmp %rd,*imm16*	%rd-*imm16* （PSRのC, V, Z, Nフラグを変更）	(1)
xcmc %rd,*imm16*	%rd-*imm16*-C （PSRのC, V, Z, Nフラグを変更）	(1)
xcmp.a %rd,*imm24*	%rd-*imm24* （PSRのC, V, Z, Nフラグを変更）	(3)

この拡張命令により、汎用レジスタと符号付き 16 ビット/20 ビット/24 ビット即値との比較が行えます。
拡張命令に、条件演算オプション（/c、/nc）は指定できません。

●展開後の基本命令

scmp, **xcmp** cmp 命令に展開
scmc, **xcmc** cmc 命令に展開
scmp.a, **xcmp.a** cmp.a 命令に展開

●展開形式

(1) **s*OP* %rd,*imm16* / x*OP* %rd,*imm16*** （*OP* = cmp, cmc）

例: xcmp %rd,*imm16*

imm16 ≤ 0x7f	0x7f < imm16
cmp %rd,*imm16*(6:0)	ext *imm16*(15:7) cmp %rd,*imm16*(6:0)

(2) **scmp.a %rd,*imm20***

imm20 ≤ 0x7f	*0x7f < imm20*
cmp.a %rd,*imm20*(6:0)	ext *imm20*(19:7) cmp.a %rd,*imm20*(6:0)

(3) **xcmp.a %rd,*imm24***

imm24 ≤ 0x7f	*0x7f < imm24* ≤ 0xfffff	*imm24* > 0xfffff
cmp.a %rd,*imm24*(6:0)	ext *imm24*(19:7) cmp.a %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) cmp.a %rd,*imm24*(6:0)

6.6.3 論理演算命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
sand %rd,*imm16*	%rd ← %rd&*imm16*	(1)
soor %rd,*imm16*	%rd ← %rd|*imm16*	(1)
sxor %rd,*imm16*	%rd ← %rd^*imm16*	(1)
snot %rd,*imm16*	%rd ← !*imm16*	(1)
xand %rd,*imm16*	%rd ← %rd&*imm16*	(1)
xoor %rd,*imm16*	%rd ← %rd|*imm16*	(1)
xxor %rd,*imm16*	%rd ← %rd^*imm16*	(1)
xnot %rd,*imm16*	%rd ← !*imm16*	(1)

これらの拡張命令により、論理演算で符号付き 16 ビット即値が直接指定可能となります。
拡張命令に、条件演算オプション（/c、/nc）は指定できません。

●展開後の基本命令

sand, **xand**　and 命令に展開
soor, **xoor**　or 命令に展開
sxor, **xxor**　xor 命令に展開
snot, **xnot**　not 命令に展開

●展開形式

(1) **s*OP* %rd,imm*16* / x*OP* %rd,imm*16* (*OP* = and, oor, xor, not)

例: xand %rd,*imm16*

imm*16* ≤ 0x7f	0x7f < imm16
and %rd,*imm16*(6:0)	ext *imm16*(15:7) and %rd,*imm16*(6:0)

6.6.4 スタック～レジスタ間データ転送命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
sld.b %rd,[%sp+*imm20*]	%rd ← B[%sp+*imm20*]（**符号拡張**）	(1)
sld.ub %rd,[%sp+*imm20*]	%rd ← B[%sp+*imm20*]（**ゼロ拡張**）	(1)
sld %rd,[%sp+*imm20*]	%rd ← W[%sp+*imm20*]	(1)
sld.a %rd,[%sp+*imm20*]	%rd ← A[%sp+*imm20*](23:0), **無視** ← A[%sp+ *imm20*](31:24)	(1)
sld.b [%sp+*imm20*],%rs	B[%sp+*imm20*] ← %rs(7:0)	(1)
sld [%sp+*imm20*],%rs	W[%sp+*imm20*] ← %rs(15:0)	(1)
sld.a [%sp+*imm20*],%rs	A[%sp+*imm20*](23:0) ← %rs(23:0), A[%sp+ *imm20*](31:24) ← 0	(1)
xld.b %rd,[%sp+*imm24*]	%rd ← B[%sp+*imm24*]（**符号拡張**）	(2)
xld.ub %rd,[%sp+*imm24*]	%rd ← B[%sp+*imm24*]（**ゼロ拡張**）	(2)
xld %rd,[%sp+*imm24*]	%rd ← W[%sp+*imm24*]	(2)
xld.a %rd,[%sp+*imm24*]	%rd ← A[%sp+*imm24*](23:0), **無視** ← A[%sp+ *imm20*](31:24)	(2)
xld.b [%sp+*imm24*],%rs	B[%sp+*imm24*] ← %rs(7:0)	(2)
xld [%sp+*imm24*],%rs	W[%sp+*imm24*] ← %rs(15:0)	(2)
xld.a [%sp+*imm24*],%rs	A[%sp+*imm24*](23:0) ← %rs(23:0), A[%sp+ *imm20*](31:24) ← 0	(2)

これらの拡張命令により、20 ビット/24 ビットまでのディスプレースメントが直接指定可能となります。*imm20*/*imm24* は省略可能です。

●展開後の基本命令

sld.b, **xld.b** ld.b 命令に展開
sld.ub, **xld.ub** ld.ub 命令に展開
sld, **xld** ld 命令に展開
sld.a, **xld.a** ld.a 命令に展開

●展開形式

imm20、*imm24* を省略した場合、[%sp+0x0]を指定したものとして展開されます。

(1) **s*OP* %rd,[%sp+*imm20*]** (*OP* = ld.b, ld.ub, ld, ld.a)
s*OP* [%sp+*imm20*],%rs (*OP* = ld.b, ld, ld.a)

例: sld.a %rd,[%sp+*imm20*]

imm20 ≤ 0x7f	0x7f < *imm20*
ld.a %rd,[%sp+*imm20*(6:0)]	ext *imm20*(19:7) ld.a %rd,[%sp+*imm20*(6:0)]

(2) **x*OP* %rd,[%sp+*imm24*]** (*OP* = ld.b, ld.ub, ld, ld.a)
x*OP* [%sp+*imm24*],%rs (*OP* = ld.b, ld, ld.a)

例: xld.a %rd,[%sp+*imm24*]

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.a %rd,[%sp+*imm24*(6:0)]	ext *imm24*(19:7) ld.a %rd,[%sp+*imm24*(6:0)]	ext *imm24*(23:20) ext *imm24*(19:7) ld.a %rd,[%sp+*imm24*(6:0)]

6.6.5 メモリ～レジスタ間データ転送命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
sld.b %rd,[*imm20*]	%rd ← B[*imm20*]（**符号拡張**）	(1)
sld.ub %rd,[*imm20*]	%rd ← B[*imm20*]（**ゼロ拡張**）	(1)
sld %rd,[*imm20*]	%rd ← W[*imm20*]	(1)
sld.a %rd,[*imm20*]	%rd ← A[*imm20*](23:0), **無視** ← A[*imm20*](31:24)	(1)
sld.b [*imm20*],%rs	B[*imm20*] ← %rs(7:0)	(1)
sld [*imm20*],%rs	W[*imm20*] ← %rs(15:0)	(1)
sld.a [*imm20*],%rs	A[*imm20*](23:0) ← %rs(23:0), A[*imm20*](31:24) ← 0	(1)
xld.b %rd,[*imm24*]	%rd ← B[*imm24*]（**符号拡張**）	(2)
xld.ub %rd,[*imm24*]	%rd ← B[*imm24*]（**ゼロ拡張**）	(2)
xld %rd,[*imm24*]	%rd ← W[*imm24*]	(2)
xld.a %rd,[*imm24*]	%rd ← A[*imm24*](23:0), **無視** ← A[*imm24*](31:24)	(2)
xld.b [*imm24*],%rs	B[*imm24*] ← %rs(7:0)	(2)
xld [*imm24*],%rs	W[*imm24*] ← %rs(15:0)	(2)
xld.a [*imm24*],%rs	A[*imm24*](23:0) ← %rs(23:0), A[*imm24*](31:24) ← 0	(2)

これらの拡張命令により、20 ビット/24 ビット即値でアドレスを指定するメモリアクセスが可能となります。ただし、ポストインクリメント機能（[]+）は使用できません。

●展開後の基本命令

sld.b, **xld.b**　　ld.b 命令に展開
sld.ub, **xld.ub**　　ld.ub 命令に展開
sld, **xld**　　ld 命令に展開
sld.a, **xld.a**　　ld.a 命令に展開

●展開形式

(1) **s*OP* %rd,[*imm20*]**　　(*OP* = ld.b, ld.ub, ld, ld.a)
s*OP* [*imm20*],%rs　　(*OP* = ld.b, ld, ld.a)

例: sld.a %rd,[*imm20*]

imm20 ≤ 0x7f	0x7f < *imm20*
ld.a %rd,[*imm20*(6:0)]	ext *imm20*(19:7) ld.a %rd,[*imm20*(6:0)]

(2) **x*OP* %rd,[*imm24*]**　　(*OP* = ld.b, ld.ub, ld, ld.a)
x*OP* [*imm24*],%rs　　(*OP* = ld.b, ld, ld.a)

例: xld.a %rd,[*imm24*]

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.a %rd,[*imm24*(6:0)]	ext *imm24*(19:7) ld.a %rd,[*imm24*(6:0)]	ext *imm24*(23:20) ext *imm24*(19:7) ld.a %rd,[*imm24*(6:0)]

6.6.6 即値ロード命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
sld %rd,*imm16*	%rd ← *imm16*	(1)
sld.a %rd,*imm20*	%rd ← *imm20*	(2)
sld.a %sp,*imm20*	%sp ← *imm20*	(2)
sld %rd,*symbol*±*imm16*	%rd ← *symbol*±*imm16*(15:0)	(4)
sld.a %rd,*symbol*±*imm20*	%rd ← *symbol*±*imm20*(19:0)	(5)
sld.a %sp,*symbol*±*imm20*	%sp ← *symbol*±*imm20*(19:0)	(5)
xld %rd,*imm16*	%rd ← *imm16*	(1)
xld.a %rd,*imm24*	%rd ← *imm24*	(3)
xld.a %sp,*imm24*	%sp ← *imm24*	(3)
xld %rd,*symbol*±*imm16*	%rd ← *symbol*±*imm16*(15:0)	(4)
xld.a %rd,*symbol*±*imm24*	%rd ← *symbol*±*imm24*(23:0)	(6)
xld.a %sp,*symbol*±*imm24*	%sp ← *symbol*±*imm24*(23:0)	(6)

これらの拡張命令により、16 ビット/20 ビット/24 ビットの即値を直接汎用レジスタにロードすることができます。即値指定にはシンボルも使用可能です。

●展開後の基本命令

sld, **xld**　　ld 命令に展開
sld.a, **xld.a**　　ld.a 命令に展開

●展開形式

(1) **sld %rd,*imm16*** / **xld %rd,*imm16***

例: xld %rd,*imm16*

imm16 ≤ 0x7f	0x7f < *imm16*
ld %rd,*imm16*(6:0)	ext *imm16*(15:7) ld %rd,*imm16*(6:0)

(2) **sld.a %rd,*imm20*** / **sld.a %sp,*imm20***

例: sld.a %rd,*imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
ld.a %rd,*imm20*(6:0)	ext *imm20*(19:7) ld.a %rd,*imm20*(6:0)

(3) **xld.a %rd,*imm24*** / **xld.a %sp,*imm24***

例: xld.a %rd,*imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.a %rd,*imm24*(6:0)	ext *imm24*(19:7) ld.a %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) ld.a %rd,*imm24*(6:0)

(4) **sld %rd,*symbol±imm16*** / **xld %rd,*symbol±imm16***

例: sld %rd,*symbol±imm16*

無条件
ext (*symbol±imm16*)(15:7) ld %rd,(*symbol±imm16*)(6:0)

(5) **sld.a %rd,*symbol±imm20*** / **sld.a %sp,*symbol±imm20***

例: sld.a %rd,*symbol±imm20*

無条件
ext (*symbol±imm20*)(19:7) ld.a %rd,(*symbol±imm20*)(6:0)

(6) **xld.a %rd,*symbol±imm24*** / **xld.a %sp,*symbol±imm24***

例: xld.a %rd,*symbol±imm24*

無条件
ext (*symbol±imm24*)(23:20) ext (*symbol±imm24*)(19:7) ld.a %rd,(*symbol±imm24*)(6:0)

6.6.7 分岐命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
scall *label±imm20*	PC相対サブルーチンコール	(1)
sjpr *label±imm20*	PC相対無条件ジャンプ	(1)
sjreq *label±imm20*	PC相対条件付きジャンプ	(2)
sjrne *label±imm20*	PC相対条件付きジャンプ	(2)
sjrgt *label±imm20*	PC相対条件付きジャンプ	(2)
sjrge *label±imm20*	PC相対条件付きジャンプ	(2)
sjrlt *label±imm20*	PC相対条件付きジャンプ	(2)
sjrle *label±imm20*	PC相対条件付きジャンプ	(2)
sjrugt *label±imm20*	PC相対条件付きジャンプ	(2)
sjruge *label±imm20*	PC相対条件付きジャンプ	(2)
sjrult *label±imm20*	PC相対条件付きジャンプ	(2)
sjrule *label±imm20*	PC相対条件付きジャンプ	(2)
scalla *label±imm20*	PC絶対サブルーチンコール	(3)
sjpa *label±imm20*	PC絶対無条件ジャンプ	(3)
scall *sign20*	PC相対サブルーチンコール	(4)
sjpr *sign20*	PC相対無条件ジャンプ	(4)
sjreq *sign20*	PC相対条件付きジャンプ	(5)
sjrne *sign20*	PC相対条件付きジャンプ	(5)
sjrgt *sign20*	PC相対条件付きジャンプ	(5)
sjrge *sign20*	PC相対条件付きジャンプ	(5)
sjrlt *sign20*	PC相対条件付きジャンプ	(5)
sjrle *sign20*	PC相対条件付きジャンプ	(5)
sjrugt *sign20*	PC相対条件付きジャンプ	(5)
sjruge *sign20*	PC相対条件付きジャンプ	(5)
sjrult *sign20*	PC相対条件付きジャンプ	(5)
sjrule *sign20*	PC相対条件付きジャンプ	(5)
scalla *imm20*	PC絶対サブルーチンコール	(6)
sjpa *imm20*	PC絶対無条件ジャンプ	(6)
xcall *label±imm24*	PC相対サブルーチンコール	(7)
xjpr *label±imm24*	PC相対無条件ジャンプ	(7)
xjreq *label±imm24*	PC相対条件付きジャンプ	(8)
xjrne *label±imm24*	PC相対条件付きジャンプ	(8)
xjrgt *label±imm24*	PC相対条件付きジャンプ	(8)
xjrge *label±imm24*	PC相対条件付きジャンプ	(8)
xjrlt *label±imm24*	PC相対条件付きジャンプ	(8)
xjrle *label±imm24*	PC相対条件付きジャンプ	(8)
xjrugt *label±imm24*	PC相対条件付きジャンプ	(8)
xjruge *label±imm24*	PC相対条件付きジャンプ	(8)
xjrult *label±imm24*	PC相対条件付きジャンプ	(8)
xjrule *label±imm24*	PC相対条件付きジャンプ	(8)
xcalla *label±imm24*	PC絶対サブルーチンコール	(9)
xjpa *label±imm24*	PC絶対無条件ジャンプ	(9)
xcall *sign24*	PC相対サブルーチンコール	(10)
xjpr *sign24*	PC相対無条件ジャンプ	(10)
xjreq *sign24*	PC相対条件付きジャンプ	(11)
xjrne *sign24*	PC相対条件付きジャンプ	(11)
xjrgt *sign24*	PC相対条件付きジャンプ	(11)

xjrge *sign24*	PC相対条件付きジャンプ	(11)
xjrlt *sign24*	PC相対条件付きジャンプ	(11)
xjrle *sign24*	PC相対条件付きジャンプ	(11)
xjrugt *sign24*	PC相対条件付きジャンプ	(11)
xjruge *sign24*	PC相対条件付きジャンプ	(11)
xjrult *sign24*	PC相対条件付きジャンプ	(11)
xjrule *sign24*	PC相対条件付きジャンプ	(11)
xcalla *imm24*	PC絶対サブルーチンコール	(12)
xjpa *imm24*	PC絶対無条件ジャンプ	(12)

これらの拡張命令により、20 ビット/24 ビット即値、またはラベルで分岐先を指定することができます。条件付きジャンプ命令の分岐条件は基本命令と同じです。
拡張命令も、".d"を付加してディレイド分岐命令として使用可能です。

例：xcall.d *sign24*

●展開後の基本命令

scall, **scall.d**, **xcall**, **xcall.d** call/call.d 命令に展開
scalla, **scalla.d**, **xcalla**, **xcalla.d** calla/calla.d 命令に展開
sjpa, **sjpa.d**, **xjpa**, **xjpa.d** jpa/jpa.d 命令に展開
sjpr, **sjpr.d**, **xjpr**, **xjpr.d** jpr/jpr.d 命令に展開
sjreq, **sjreq.d**, **xjreq**, **xjreq.d** jreq/jreq.d 命令に展開
sjrne, **sjrne.d**, **xjrne**, **xjrne.d** jrne/jrne.d 命令に展開
sjrgt, **sjrgt.d**, **xjrgt**, **xjrgt.d** jrgt/jrgt.d 命令に展開
sjrge, **sjrge.d**, **xjrge**, **xjrge.d** jrge/jrge.d 命令に展開
sjrlt, **sjrlt.d**, **xjrlt**, **xjrlt.d** jrlt/jrlt.d 命令に展開
sjrle, **sjrle.d**, **xjrle**, **xjrle.d** jrle/jrle.d 命令に展開
sjrugt, **sjrugt.d**, **xjrugt**, **xjrugt.d** jrugt/jrugt.d 命令に展開
sjruge, **sjruge.d**, **xjruge**, **xjruge.d** jruge/jruge.d 命令に展開
sjrult, **sjrult.d**, **xjrult**, **xjrult.d** jrult/jrult.d 命令に展開
sjrule, **sjrule.d**, **xjrule**, **xjrule.d** jrule/jrule.d 命令に展開

●展開形式

(1) ***sOP label±imm20*** (*OP* = call, call.d, jpr, jpr.d)

例: scall *label±imm20*

無条件
ext (*label±imm20*)(19:12) call (*label±imm20*)(11:1)

(2) ***sOP label±imm20*** (*OP* = jr*, jr*.d)

例: sjreq *label±imm20*

無条件
ext (*label±imm20*)(19:8) jreq (*label±imm20*)(7:1)

(3) ***sOP label±imm20*** (*OP* = calla, calla.d, jpa, jpa.d)

例: scalla *label±imm20*

無条件
ext (*label±imm20*)(19:7) calla (*label±imm20*)(6:0)

(4) ***sOP sign20*** (*OP* = call, call.d, jpr, jpr.d)

例: scall *sign20*

-1024 ≤ *sign20* ≤ 1023	*sign20* < -1024 または 1023 < *sign20*
call *sign20*(11:1)	ext *sign20*(23:12) call *sign20*(11:1)

(5) ***sOP sign20*** (*OP* = jr*, jr*.d)

例: sjreq *sign20*

-128 ≤ *sign20* ≤ 127	*sign20* < -128 または 127 < *sign20*
jreq *sign20*(7:1)	ext *sign20*(19:8) jreq *sign20*(7:1)

(6) ***sOP imm20*** (*OP* = calla, calla.d, jpa, jpa.d)

例: scalla *imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
calla *imm20*(6:0)	ext *imm20*(19:7) calla *imm20*(6:0)

(7) ***xOP label±imm24*** (*OP* = call, call.d, jpr, jpr.d)

例: xcall *label±imm24*

無条件
ext (*label±imm24*)(23:12) call (*label±imm24*)(11:1)

(8) ***xOP label±imm24*** (*OP* = jr*, jr*.d)

例: xjreq *label±imm24*

無条件
ext (*label±imm24*)(23:21) ext (*label±imm24*)(20:8) jreq (*label±imm24*)(7:1)

(9) ***xOP label±imm24*** (*OP* = calla, calla.d, jpa, jpa.d)

例: xcalla *label±imm24*

無条件
ext (*label±imm24*)(23:20) ext (*label±imm24*)(19:7) calla (*label±imm24*)(6:0)

(10) ***xOP sign24*** (*OP* = call, call.d, jpr, jpr.d)

例: xcall *sign24*

-1024 ≤ *sign24* ≤ 1023	*sign24* < -1024 または 1023 < *sign24*
call *sign24*(11:1)	ext *sign24*(23:12) call *sign24*(11:1)

(11) **x*OP* *sign24*** (*OP* = jr*, jr*.d)

例: xjreq *sign24*

-128 ≤ *sign24* ≤ 127	-1048576 ≤ *sign24* < -128 または 127 < *sign24* ≤ 1048575	*sign24* < -1048576 または 1048575 < *sign24*
jreq *sign24*(7:1)	ext *sign24*(20:8) jreq *sign24*(7:1)	ext *sign24*(23:21) ext *sign24*(20:8) jreq *sign24*(7:1)

(12) **x*OP* *imm24*** (*OP* = calla, calla.d, jpa, jpa.d)

例: xcalla *imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
calla *imm24*(6:0)	ext *imm24*(19:7) calla *imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) calla *imm24*(6:0)

6.6.8 コプロセッサ命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
sld.cw %rd,*imm20*	コプロセッサ ← %rd & *imm20*	(1)
sld.ca %rd,*imm20*	コプロセッサ ← %rd & *imm20*, 結果&フラグ取得	(1)
sld.cf %rd,*imm20*	コプロセッサ ← %rd & *imm20*, フラグ取得	(1)
sld.cw %rd,*symbol*±*imm20*	コプロセッサ ← %rd & *symbol*±*imm20*	(2)
sld.ca %rd,*symbol*±*imm20*	コプロセッサ ← %rd & *symbol*±*imm20*, 結果&フラグ取得	(2)
sld.cf %sp,*symbol*±*imm20*	コプロセッサ ← %rd & *symbol*±*imm20*, フラグ取得	(2)
xld.cw %rd,*imm24*	コプロセッサ ← %rd & *imm24*	(3)
xld.ca %rd,*imm24*	コプロセッサ ← %rd & *imm24*, 結果&フラグ取得	(3)
xld.cf %rd,*imm24*	コプロセッサ ← %rd & *imm24*, フラグ取得	(3)
xld.cw %rd,*symbol*±*imm24*	コプロセッサ ← %rd & *symbol*±*imm24*	(4)
xld.ca %rd,*symbol*±*imm24*	コプロセッサ ← %rd & *symbol*±*imm24*, 結果&フラグ取得	(4)
xld.cf %rd,*symbol*±*imm24*	コプロセッサ ← %rd & *symbol*±*imm24*, フラグ取得	(4)

これらの拡張命令により、20 ビット/24 ビットの即値をコプロセッサに転送することができます。即値指定にはシンボルも使用可能です。

●展開後の基本命令

sld.cw, **xld.cw** ld.cw 命令に展開
sld.ca, **xld.ca** ld.ca 命令に展開
sld.cf, **xld.cf** ld.cf 命令に展開

●展開形式

(1) ***sOP %rd,imm20*** (*OP* = ld.cw, ld.ca, ld.cf)

例: sld.ca %rd,imm20

imm20 ≤ 0x7f	0x7f < *imm20*
ld.ca %rd,*imm20*(6:0)	ext *imm20*(19:7) ld.ca %rd,*imm20*(6:0)

(2) ***sOP %rd,symbol±imm20*** (*OP* = ld.cw, ld.ca, ld.cf)

例: sld.ca %rd,*symbol*±*imm20*

無条件
ext (*symbol*±*imm20*)(19:7) ld.ca %rd,(*symbol*±*imm20*)(6:0)

(3) ***xOP %rd,imm24*** (*OP* = ld.cw, ld.ca, ld.cf)

例: xld.ca %rd,*imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.ca %rd,*imm24*(6:0)	ext *imm24*(19:7) ld.ca %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) ld.ca %rd,*imm24*(6:0)

(4) **xOP %rd,*symbol±imm24*** （*OP* = ld.cw, ld.ca, ld.cf）

例: xld.ca %rd,*symbol±imm24*

無条件
ext (*symbol±imm24*)(23:20) ext (*symbol±imm24*)(19:7) ld.ca %rd,(*symbol±imm24*)(6:0)

6.6.9 Xext 命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
Xext *imm24*	ext 命令に展開	(1)

以下の命令と組み合わせることにより、オフセットとして機能します。

Xext imm24
OP [%rd], %rs **==>** [%rd+imm24] ← %rs として機能します。

Xext imm24
OP %rd, [%rs] **==>** %rd ← [%rs+imm24] として機能します。

（*OP* = ld.b, ld.ub, ld, ld.a）

●展開後の基本命令

Xext ext 命令に展開

●展開形式

(1) **Xext *imm24***

imm24 ≤ 0x1fff	0x1fff < *imm24* ≤ 0xffffff
ext *imm24*(12:0)	ext *imm24*(23:13) ext *imm24*(12:0)

6.7 エラー/ワーニングメッセージ

以下に、アセンブラ **as** が出力する主なエラーおよびワーニングメッセージを示します。

表 6.7.1 エラーメッセージ

エラーメッセージ	内容
Error: Unrecognized opcode: *'XXXXX'*	*XXXXX*は未定義のオペコードです。
Error: junk at end of line: *'XXXXX'*	オペランド書式のエラーです。
Error: *XXXXXX*: invalid register name	使用できないレジスタを指定しています。

表 6.7.2 ワーニングメッセージ

ワーニングメッセージ	内容
Warning: Unrecognized .section attribute: want a, w, x	セクションの属性がa、w、x以外です。
Warning: Bignum truncated to *AAA* bytes	定数宣言(`.long`、`.int`など)のサイズが最大値を超えています。値を*AAA*バイトサイズに補正します。 例:`0x100000012 → 0x12`
Warning: Value *XXXX* truncated to *AAA*	定数宣言の値が最大値を超えています。値を*AAA*に補正します。 例:`.byte 0x10000012 → .byte 0xff`
Warning: operand out of range (*XXXXXX*: *XXX* not between *AAA* and *BBB*)	オペランドで指定されている値は有効範囲外です。

6.8 注意事項

- デバッガ **gdb** でアセンブラソースレベルデバッグを行う場合には、アセンブラ **as** のオプション --gstabs を指定し、ソース情報を付加してください。
- ソースファイル内のデバッグ情報（.stab 擬似命令）は、**xgcc** および **as** 以外で作成しないでください。また出力されたデバッグ情報を修正しないでください。**as**、**ld**、**gdb** の誤動作につながります。
- リンク時の不具合を防止するため、.section 擬似命令を記述する場合は必ず.align 命令も併記し、セクションの境界を明確にしてください。

例:

```
            .section .rodata
            .align 2            ; ←必須
    .long   data1
    .long   data2
```

7 リンカ

この章ではリンカ ld の機能を説明します。

7.1 機能

リンカ **ld** は実行可能なオブジェクトファイルを生成するソフトウェアで、以下の機能があります。

- ライブラリを含め、複数のオブジェクトモジュールをリンクし、1 つの実行可能なオブジェクトファイルを生成
- モジュール間の外部参照を解決
- リロケータブルアドレスをアブソリュートアドレスに再配置
- リンク後のオブジェクトファイルに行番号やシンボル情報などのデバッグ情報を出力
- リンクマップファイルを出力可能

このリンカは GNU リンカ（ld）をベースとしています。リンカ **ld** の詳細およびリンカスクリプトの記述方法については、GNU リンカのドキュメントを参照してください。ドキュメントは、世界各地にある GNU のミラーサイトからインターネット等を利用して入手可能です。

7.2 入出力ファイル

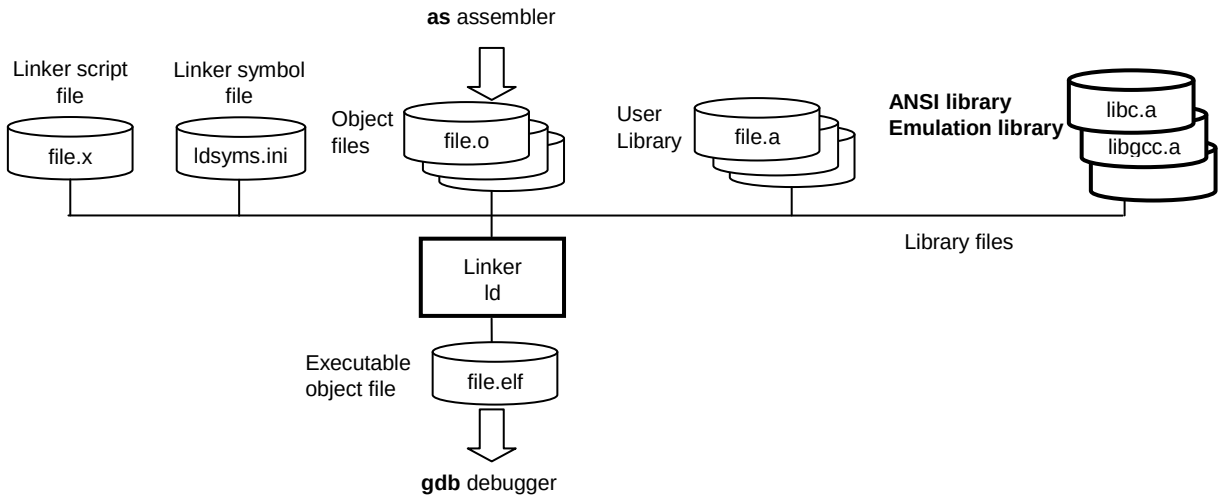


図 7.2.1 フローチャート

7.2.1 入力ファイル

●オブジェクトファイル

ファイル形式: elf 形式のバイナリファイル
ファイル名: <*ファイル名*>.o
内容: アセンブラ **as** で生成したモジュール個々のオブジェクトファイルです。

●ライブラリファイル

ファイル形式: ライブラリ形式のバイナリファイル
ファイル名: <*ファイル名*>.a
内容: ANSI ライブラリ、エミュレーションライブラリ、およびユーザのライブラリファイルです。

●リンカスクリプトファイル

ファイル形式: テキストファイル
ファイル名: <*ファイル名*>.x
内容: 各セクションの開始アドレスなどリンクに必要な情報を指定するファイルです。
このファイルは-T オプションが指定された場合にリンカ **ld** に入力されます。

●リンカシンボルファイル

ファイル形式: テキストファイル
ファイル名: ldsyms.ini
内容: シンボル名に対応するアドレスを指定するファイルです。
このファイルは-R オプションが指定された場合にリンカ **ld** に入力されます。

7.2.2 出力ファイル

●実行形式オブジェクトファイル

ファイル形式: elf 形式のバイナリファイル
ファイル名: <*ファイル名*>.elf
内容: デバッガ **gdb** に入力可能な実行形式のオブジェクトファイルです。1 つのプログラムを構成するすべてのモジュールがリンクされ、すべてのコードが配置される絶対アドレスが決定した内容となっています。デバッグに必要な情報も elf 形式で含まれています。
-o オプションによる出力ファイル名の指定がない場合、オブジェクトファイルは a.out という名称で生成されます。

●リンクマップファイル

ファイル形式: テキストファイル
ファイル名: <*ファイル名*>.map
内容: 各入力ファイルが各セクションのどのアドレスから配置されたかを示すマップ情報ファイルです。起動時オプションでの-M または-Map が指定された場合に出力されます。

7.3 起動方法

7.3.1 起動フォーマット

リンカ **ld** は次のコマンドにより起動します。

ld ***<オプション> <ファイル名>***

<オプション> 7.3.2 項参照

<ファイル名> リンクするオブジェクトファイルおよびライブラリファイルを指定します。

例: `ld -o sample.elf sample.o ..¥lib¥24bit¥libc.a ..¥lib¥24bit¥libgcc.a`

7.3.2 コマンドラインオプション

リンカ **ld** は GNU リンカ標準のコマンドオプションを受け付けます。

ここでは、よく使用するオプションのみを説明しますので、その他のオプションも含め、詳細については GNU リンカのマニュアルを参照してください。

-o ***<ファイル名>***

機能: **出力ファイル名の指定**
説明: リンカ **ld** が出力するオブジェクトファイルの名称を指定します。
デフォルト: `a.out` という名称のファイルを生成します。

-T ***<リンカスクリプトファイル名>***

機能: **リンカスクリプトファイルの読み込み**
説明: リンカスクリプトファイルから配置情報を入力する場合に指定します。
デフォルト: デフォルトリンカスクリプト（7.4.1 項参照）が使用されます。

-R ***<リンカシンボルファイル名>***

機能: **リンカシンボルファイルの読み込み**
説明: シンボル名に対応するアドレスをリンカシンボルファイルで指定します。
デフォルト: リンカシンボルファイルを読み込みません。

-M

-Map ***<ファイル名>***

機能: **リンクマップファイルの出力**
説明: `-M` オプションを指定するとリンクマップ情報が `stdio` デバイスに出力されます。
`-Map` オプションはリンクマップ情報を指定のファイルに出力します。
デフォルト: リンクマップ情報は出力されません。

-N

機能: **データセグメントのアライメントチェックを禁止**
説明: `-N` オプションを指定すると、データセグメントのアライメントチェックを行いません。
デフォルト: リンカはデータセグメントのアライメントチェックを行います。

--relax

機能: **出力コードサイズの最適化**
説明: `--relax` オプションを指定すると、ext 0 命令を削除して出力コードサイズを最適化します。
デフォルト: リンカは ext 0 命令を削除しません。

コマンドラインにオプションを入力する場合、オプションの前後には 1 個以上のスペースが必要です。

例: `ld -o sample.elf -T sample.x sample.o ..¥lib¥24bit¥libc.a`

7.4 リンク処理

7.4.1 デフォルトリンカスクリプト

-T オプション未指定時のデフォルトリンカスクリプト

-T オプションが指定されないと、リンカ **ld** は下記のリンカスクリプトを使用してリンクの処理をします。

```
OUTPUT_FORMAT("elf32-c17")
OUTPUT_ARCH(c17)
ENTRY(_start)
SEARCH_DIR(.);
MEMORY
{
 iram    : ORIGIN = 0,         LENGTH = 32K
 irom    : ORIGIN = 0x8000,    LENGTH = 4064K
}
SECTIONS
{
 .bss (NOLOAD) :
 {
    PROVIDE (__START_bss = .) ;
    *(.bss)
    *(.bss.*)
    *(COMMON)
    PROVIDE (__END_bss = .) ;
 }  > iram
 .vector :
 {
    PROVIDE (__START_vector = .) ;
    KEEP (*crt0.o(.rodata))
    PROVIDE (__END_vector = .) ;
 }  > irom
 .text :
 {
    PROVIDE (__START_text = .) ;
    *(.text.*)
    *(.text)
    PROVIDE (__END_text = .) ;
 }  > irom
 .data :
 {
    PROVIDE (__START_data = .) ;
    *(.data)
    *(.data.*)
    PROVIDE (__END_data = .) ;
 }  > iram AT > irom
 .rodata :
 {
    PROVIDE (__START_rodata = .) ;
    *(EXCLUDE_FILE (*crt0.o ) .rodata)
    *(.rodata.*)
    PROVIDE (__END_rodata = .) ;
 }  > irom
```

```
 PROVIDE (__START_data_lma = LOADADDR(.data));
 PROVIDE (__END_data_lma = LOADADDR(.data) + SIZEOF (.data));
 PROVIDE (__START_stack = 0x0007C0);
}
```

このスクリプトでは、データが.bss、.data の順にアドレス 0 から配置され、ベクタテーブル、プログラムコード、および定数データはアドレス 0x8000 から配置されます。
図 7.4.1.1 にリンク後のメモリマップを示します。

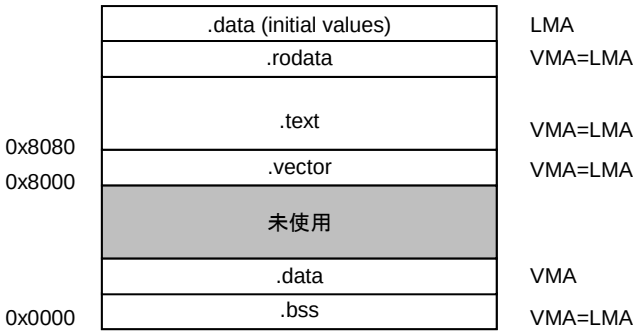


図 7.4.1.1 デフォルトリンカスクリプトによるメモリマップ

7.4.2 リンク例

仮想・共用セクションを使用する場合

仮想セクションおよび共用セクションを使用する場合のリンカスクリプト例を以下に示します。

```
OUTPUT_FORMAT("elf32-c17")
OUTPUT_ARCH(c17)
ENTRY(_start)
SEARCH_DIR(.);
MEMORY
{
 iram    : ORIGIN = 0,         LENGTH = 32K
 irom    : ORIGIN = 0x8000,    LENGTH = 4064K
}
SECTIONS
{
 .bss (NOLOAD) :
 {
    PROVIDE (__START_bss = .) ;
    *(.bss)
    *(.bss.*)
    *(COMMON)
    PROVIDE (__END_bss = .) ;
 } > iram
 .vector :
 {
    PROVIDE (__START_vector = .) ;
    KEEP (*crt0.o(.rodata))
    PROVIDE (__END_vector = .) ;
 } > irom
 .text :
 {
    PROVIDE (__START_text = .) ;
    *(EXCLUDE_FILE (*foo1.o *foo2.o *foo3.o) .text.*)
    *(EXCLUDE_FILE (*foo1.o *foo2.o *foo3.o) .text)
    PROVIDE (__END_text = .) ;
 } > irom
 .data :
 {
    PROVIDE (__START_data = .) ;
    *(.data)
    *(.data.*)
    PROVIDE (__END_data = .) ;
 } > iram AT > irom
 OVERLAY ".":
 {
    .text_foo1 { *foo1.o(.text.*) *foo1.o(.text) }
    .text_foo2 { *foo2.o(.text.*) *foo2.o(.text) }
    .text_foo3 { *foo3.o(.text.*) *foo3.o(.text) }
 } > iram AT > irom
 .rodata :
 {
    PROVIDE (__START_rodata = .) ;
    *(EXCLUDE_FILE (*crt0.o ) .rodata)
    *(.rodata.*)
```

```
      PROVIDE (__END_rodata = .) ;
   }  > irom
   PROVIDE (__START_data_lma = LOADADDR(.data));
   PROVIDE (__END_data_lma = LOADADDR(.data) + SIZEOF (.data));
   PROVIDE (__START_stack = 0x0007C0);
}
```

図 7.4.2.1 にセクションの配置を示します。

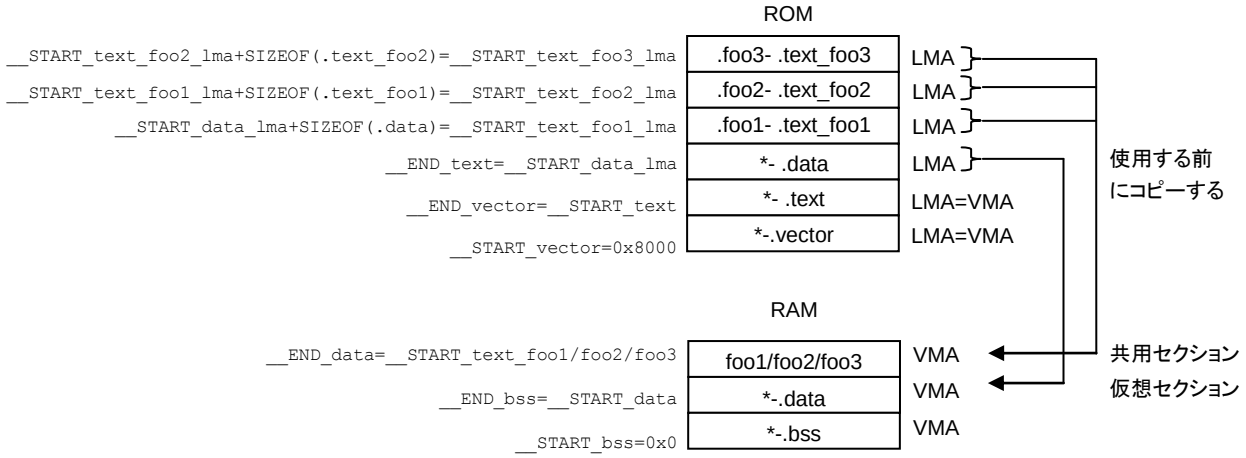


図 7.4.2.1 メモリマップ

.data セクションは ROM 上の LMA に実体を持ち、RAM 上の VMA（.bss セクションの直後）にコピーして使用するように設定しています。RAM 上の.data セクション（VMA）はプログラムの実行開始時にはデータが存在しない仮想セクションと見ることができます。初期値を持つ変数などは、このように使用する必要があります。この例の場合、すべてのファイルの.data セクションが 1 つにまとめられています。

.text_foo1 は foo1.o ファイルの.text セクションで、ROM 上の LMA に実コードを持ち、RAM 上の VMA で実行するように設定しています。.text_foo2 セクションと.text_foo3 セクションも同様ですが、これら 3 つのセクションは同じ VMA が設定されています。同じ実行アドレスを共用しているため、RAM 上の.text_foo1/2/3 セクションは共用セクションと見ることができます。プログラムを高速実行させるためのキャッシュのような使い方はこのように実現します。これら 3 つ以外のファイルの.text セクションは.vector セクションに続く.text セクションに配置され、そのまま ROM 上で実行されます。

7.4.3 リンクマップ

コマンドラインオプションに-M（または-Map）を指定すると、リンカ**ld**はリンクマップ情報を出力します。
7.4.1に示したデフォルトリンカスクリプトを用いたとき、出力されるリンクマップの例を以下に示します。

```
.bss     0x00000000   0xba
         0x00000000         PROVIDE (__START_bss, .)
*(.bss)
.bss     0x00000000   0x30 crt0.o
         0x00000000         gm_sec
         0x00000004         seed
         0x00000008         stderr
         0x0000000c         stdout
         0x00000010         stdin
         0x00000014         _iob
         0x0000002c         errno
.bss     0x00000030   0x0 src¥sample_gcc6.o
.bss     0x00000030   0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libgcc.a(emu_copro_process.o)
.bss     0x00000030   0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(puts.o)
.bss     0x00000030   0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(memcpy.o)
.bss     0x00000030   0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(memset.o)
.bss     0x00000030   0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(putc.o)
.bss     0x00000030   0x8a C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libg.a(debuglib.o)
         0x00000030         WRITE_SIZE
         0x00000032         WRITE_BUF
         0x00000074         READ_BUF_POS
         0x00000076         READ_EOF
         0x00000078         READ_BUF
*(.bss.*)
*(COMMON)
         0x000000ba         PROVIDE (__END_bss, .)

.vector   0x00008000   0x80
         [!provide]         PROVIDE (__START_vector, .)
*crt0.o(.rodata)
.rodata   0x00008000   0x80 crt0.o
         0x00008000         _vector
         [!provide]         PROVIDE (__END_vector, .)

.text    0x00008080    0x3b6
         [!provide]         PROVIDE (__START_text, .)
*(.text.*)
*(.text)
.text    0x00008080    0xac crt0.o
         0x00008080         _crt0_start0
         0x00008080         _start
         0x0000808a         _vector20_handler
         0x0000808a         _vector06_handler
         0x0000808a         _vector28_handler
         0x0000808a         _vector31_handler
         0x0000808a         _vector27_handler
         0x0000808a         _vector07_handler
         0x0000808a         _vector09_handler
         0x0000808a         _vector18_handler
         0x0000808a         _vector21_handler
         0x0000808a         _vector24_handler
         0x0000808a         _vector26_handler
         0x0000808a         _vector22_handler
         0x0000808a         _vector12_handler
         0x0000808a         _vector17_handler
         0x0000808a         _vector08_handler
         0x0000808a         _vector19_handler
         0x0000808a         _vector13_handler
         0x0000808a         _vector25_handler
         0x0000808a         _vector01_handler
         0x0000808a         _vector11_handler
         0x0000808a         _vector14_handler
```

```
         0x0000808a         _vector05_handler
         0x0000808a         _vector29_handler
         0x0000808a         _vector16_handler
         0x0000808a         _vector04_handler
         0x0000808a         _vector02_handler
         0x0000808a         _vector23_handler
         0x0000808a         _vector30_handler
         0x0000808a         _vector15_handler
         0x0000808a         _vector10_handler
         0x0000808c         _crt0_init_dummy
         0x0000808c         _init_device
         0x0000808e         _start_device
         0x0000808e         _crt0_start_device
         0x00008092         _crt0_stop_device
         0x00008092         _stop_device
         0x00008096         _init_lib
         0x00008096         _crt0_init_lib
         0x000080de         _crt0_init_section
         0x000080de         _init_section
         0x00008114         _crt0_exit
         0x00008116         _crt0_start1
         0x00008116         _start1
.text    0x0000812c    0xa  src¥sample_gcc6.o
         0x0000812c         main
.text    0x00008136    0x2 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libgcc.a(emu_copro_process.o)
         0x00008136         emu_copro_process
.text    0x00008138   0x76 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(puts.o)
         0x00008138         puts
.text    0x000081ae   0x10 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(memcpy.o)
         0x000081ae         memcpy
.text    0x000081be   0xe  C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(memset.o)
         0x000081be         memset
.text    0x000081cc   0x30 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(putc.o)
         0x000081cc         putc
.text    0x000081fc   0x23a C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libg.a(debuglib.o)
         0x000081fc         write
         0x0000823a         WRITE_FLASH
         0x00008330         _init_sys
         0x00008340         read
         0x000083d2         READ_FLASH
         0x00008432         _exit
         [!provide]         PROVIDE (__END_text, .)

.data    0x000000ba    0x0  load address 0x00008436
         0x000000ba         PROVIDE (__START_data, .)
*(.data)
.data    0x000000ba    0x0 crt0.o
.data    0x000000ba    0x0 src¥sample_gcc6.o
.data    0x000000ba    0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libgcc.a(emu_copro_process.o)
.data    0x000000ba    0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(puts.o)
.data    0x000000ba    0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(memcpy.o)
.data    0x000000ba    0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(memset.o)
.data    0x000000ba    0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(putc.o)
.data    0x000000ba    0x0 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libg.a(debuglib.o)
*(.data.*)
         0x000000ba         PROVIDE (__END_data, .)

.rodata  0x00008436    0xc
         [!provide]         PROVIDE (__START_rodata, .)
*(EXCLUDE_FILE(*crt0.o) .rodata)
.rodata  0x00008436    0xc src¥sample_gcc6.o
*(.rodata.*)
         [!provide]         PROVIDE (__END_rodata, .)
         0x00008436         PROVIDE (__START_data_lma, LOADADDR (.data))
         [!provide]         PROVIDE (__END_data_lma, (LOADADDR (.data) + SIZEOF (.data)))
         [!provide]         PROVIDE (__START_stack, 0x7c0)
OUTPUT(sample_gcc6.elf elf32-c17)
```

```
.stab      0x00000000   0xf78
.stab      0x00000000   0x570 crt0.o
.stab      0x00000570   0x21c src¥sample_gcc6.o
                        0x228 (size before relaxing)
.stab      0x0000078c 0x24 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libgcc.a(emu_copro_process.o)
                        0x30 (size before relaxing)
.stab      0x000007b0   0x7c8 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libg.a(debuglib.o)
                        0x7d4 (size before relaxing)

.comment    0x00000000   0x11
.comment    0x00000000   0x11  crt0.o
                         0x12 (size before relaxing)
.comment    0x00000011   0x12 src¥sample_gcc6.o
.comment    0x00000011   0x12 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(puts.o)
.comment    0x00000011   0x12 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libc.a(putc.o)
.comment    0x00000011   0x12 C:¥EPSON¥GNU17V3¥gcc6¥lib¥¥24bit¥libg.a(debuglib.o)

.stabstr    0x00000000   0xecb
.stabstr    0x00000000   0xecb crt0.o
```

リンカスクリプトによる指定に従い、.bss セクションがアドレス 0x00000000 から始まり、それに続いて .data セクションが配置されています。さらに、.vector セクションが 0x00008000 から、.text セクションが 0x00008080 から始まり、それに続いて .data セクションの初期値と .rodata セクションが配置されています。

シンボル __START_stack は、リンカスクリプトによる指定に従い、0x7c0 に設定されています。
その他のシンボル __START_bss, __END_bss, __START_data, __END_data, __START_vector, __END_vector, __START_text, __END_text, __START_rodata, __END_rodata, __START_data_lma, __END_data_lma についても、リンカにより配置されたアドレスが示されています。これらのシンボルを参照することで、プログラム中から各セクションのアドレスやサイズを知ることが可能です。

.stab セクションと .stabstr セクションは、デバッガが参照するデバッグ情報をプログラムに埋め込むために使用されています。これらのセクションの情報は、ターゲットにロードされません。

7.5 エラー/ワーニングメッセージ

エラーおよびワーニングメッセージは標準出力（stdout）に表示/出力されます。
リンカ **ld** では、以下に示すエラーおよびワーニングメッセージが GNU リンカの標準エラーメッセージに追加されています。

表 7.5.1 エラーメッセージ

エラーメッセージ	内容
Warning: out of range error.	シンボルのアドレスが16ビットアドレス（-mpointer16指定時）,24ビットアドレス空間を超えています。
Error: The offset value of a symbol is over 24bit.	シンボルのアドレスが24ビットアドレス空間を超えています。
Error: section *XXX* is not within 16bit address.	*XXX*セクションのアドレスが16ビットアドレス空間を越えています。
Error: section *XXX* is not within 24bit address.	*XXX*セクションのアドレスが24ビットアドレス空間を越えています。
Error: Input object file <objectfile> [included from <archivefile>] is not for C17.	オブジェクトファイルはC17用ではありません
Error: Input object file <objectfile> is not 16bit nor 24bit address mode.	オブジェクトファイルが16ビットモードでも24ビットモードでもありません
Error: Cannot link 16bit object <objectfile16> [included from <archivefile16>] with 24bit object <objectfile24> [included from <archivefile24>]	16ビットポインタモードで作成されたオブジェクトファイルと、24ビットポインタモードで作成されたオブジェクトファイルはリンクできません

7.6 リンカスクリプト生成ウィザード

本 IDE には、リンカスクリプトファイルを作成するためのリンカスクリプト生成ウィザードが組み込まれています。

7.6.1 出力ファイル

●リンカスクリプトファイル

ファイル形式: テキストファイル
ファイル名: *<ファイル名>*.x
内容: 各セクションの開始アドレスなどリンクに必要な情報を指定するファイルです。
デフォルトのファイル名はプロジェクト名が設定されていますが任意の名前に変更可能です。
作成したリンカスクリプトファイルは、プロジェクトの[Properties]ダイアログボックスの[GNU17 Setting]ページで選択してください。

7.6.2 起動と終了

●リンカスクリプト生成ウィザードの起動

リンカスクリプト生成ウィザードは以下のいずれかの方法で起動します。

(1) ツールバーの[New]ショートカットから[GNU17 Linker Script]を選択します。

(2) [File]メニュー > [New] > [GNU17 Linker Script]を選択します。

(3) [File]メニュー > [New] > [Project…] > [C/C++] > [GNU17 Linker Script]を選択します。

(4) プロジェクトを選択して右クリック > [New] > [Project…] > [C/C++] > [GNU17 Linker Script]を選択します。

●リンカスクリプト生成ウィザードの終了

リンカスクリプト生成ウィザードは以下のいずれかの操作で終了します。

(1) [Finish]ボタンをクリックする。この場合はリンカスクリプトファイルが生成されます。

(2) [Cancel]ボタンをクリックする。この場合はリンカスクリプトファイルが生成されません。

7.6.3 メニュー

●Linker Script File:
リンカスクリプトファイル名を入力します。デフォルトではプロジェクト名となっています。リンカスクリプトファイルは、現在選択されているプロジェクトフォルダ内に“<指定ファイル名>.x”として生成されます。

●Entry routine:
ENTRY コマンドで設定するプログラム開始アドレスを入力します。起動処理用ライブラリ“crt0.o”を使用する場合は、[use crt0.o]チェックボックスを選択し、“_start”を設定します。

●MCU memory regions
MEMORY コマンドで記述するメモリ構成(領域名称、開始アドレス、容量)を入力します。デフォルトでは、プロジェクトプロパティのターゲット機種で選択した MCU の RAM、ROM 領域が iram, irom として定義されています。
ORIGIN と LENGTH を変更する場合は、表示されている[Start Address]と[Length]の数値をクリックして選択し、新たな数値を入力します。新たなメモリ領域を追加するには、[Add Region]ボタンをクリックします。メモリ領域のリストに“regionX”という名称の領域が追加されますので、この内容を変更してください。
メモリ領域を削除したい場合は該当する領域を選択して[Del Region]ボタンをクリックしてください。

●Output sections and their input patterns
SECTIONS コマンドで記述する出力セクションと入力セクションを入力します。デフォルトでは、“7.4.1 デフォルトリンカスクリプト”に記述してあるデフォルトリンカスクリプトの入出力セクションが定義されています。
新たな出力セクションを追加するには、[Add Output Section]ボタンをクリックします。出力セクション名 =“.sectionX”、VMA 領域名 =“iram”という出力セクション定義が追加されます。
出力セクションを削除したい場合は該当する出力セクションを選択してから[Del Output Section]ボタンをクリックしてください。出力セクションを削除するとその出力セクションに定義した入力セクションも一緒に削除されますので注意してください。
出力セクションをダブルクリックすると[Output Section Description]ダイアログボックスが表示されますので、ここで出力セクションに関する以下の設定を行います。

	変更箇所	設定方法
出力セクション名	Section name	出力セクション名を入力します。
VMA領域名	VMA region	リストからVMA領域を選択します。リストは“MCU memory regions”で定義した領域です。
LMA領域名 （オプション）	LMA region	LMA領域を使用する場合はチェックボックスを選択して、リストからLMA領域を選択します。リストは“MCU memory regions”で定義した領域です。
アライメント値 （オプション）	Alignment	アラインメントを設定する場合はチェックボックスを選択して、リストから値を選択します。
埋め込み値 (オプション)	Fill value	空き領域の埋め込み値を設定する場合はチェックボックスを選択して、値を入力します。
NOLOAD設定 （オプション）	This section is not loaded at run time (NOLOAD).	NOLOADセクションに設定する場合はチェックボックスを選択します。
OVERLAY設定 （オプション）	This section is overlay section(OVERLAY).	OVERLAYセクションに設定する場合はチェックボックスを選択します。 設定されたセクションをOVERLAYセクションとしてまとめます。

新たな入力セクションを追加するには、出力セクションを選択して[Add Input Pattern]ボタンをクリックします。選択した出力セクションに対する入力セクションが追加されます。入力セクションを削除したい場合は該当する入力セクションを選択して[Del Input Pattern]ボタンをクリックしてください。
入力セクションをダブルクリックすると[Input Section Description]ダイアログボックスが表示されますので、ここで入力セクションに関する以下の設定を行います。

	変更箇所	設定方法
出力セクション名	Output Section	出力セクション名をリストから選択します。
入力セクション名	Input Sections	入力セクション名をリストから選択します。複数のセクションを指定する場合は直接セクション名を入力します。
入力ファイル名	Input Files	入力ファイル名を入力します。
除外ファイル名 （オプション）	Exclude Files	除外ファイルを設定する場合はチェックボックスを選択して、除外ファイル名を入力します。
KEEP設定 (オプション)	This section should not be eliminated (KEEP).	KEEPセクションに設定する場合はチェックボックスを選択します。

ウィザードで定義できない内容については、生成されたリンカスクリプトファイルをエディタで開き編集してください。

7.7 注意事項

- リンカ実行時、以下のようなエラーメッセージが表示されることがあります。
  `ld: test.elf: Not enough room for program header, try linking with -N`
  これは、リンカがデータセグメントのアライメントチェックを行っているために発生します。このエラーは、リンカ **ld** のオプションに`-N` を指定することにより回避できますので、`-N` を指定してリンクを行ってください。
- **ld** のコマンドラインで指定するオブジェクトファイル名とリンカスクリプトに記述する同じオブジェクトファイル名は大文字、小文字も含め、完全に一致させてください。大文字、小文字のみの違いでも、異なるファイルと見なされます。

例:

コマンドライン

```
ld -T sample.x -o sample.elf prg1.o prg2.o
```

リンカスクリプトファイル（sample.x）

```
                :
.text 0xc00000:
{
PRG1.o (.text) ← 小文字の prg1.o に変更する必要があります。
prg2.o (.text)
}
     :
```

- 同じ関数を含むライブラリファイル（`*.a`）をリンクしてもエラーになりません（2 重にリンクしません）。同じ関数を含むオブジェクトファイル（`*.o`）をリンクした場合はエラーになります。
- 変数または変数との演算により、配置されるアドレスが 24 ビット(0xffffff)を越える場合、もしくは 0x0 より小さくなる場合はエラーにならず、アドレスの 24 ビット以上は 0 にマスクされます。

例: `xadd.a %r0,symbol-5`
`symbol` のアドレスが 0 番地の時、絶対アドレスは 0 – 5 = 0xfffffb (-5)となります。したがって、
`xadd.a %r0,0xfffffb` として出力コードが生成されます。

8 デバッガ

この章ではデバッガ **gdb** の操作方法を説明します。

8.1 特徴

デバッガ **gdb** はリンカによって生成した elf 形式のオブジェクトファイルを読み込み、プログラムのデバッグを行うためのソフトウェアです。
以下の特長と機能を持ちます。

- 統合開発環境(IDE)のデバッグ機能でデバッグ
- マルチウィンドウにより各種のデータを一度に参照可能
- ICDmini (S5U1C17001H)を使用したデバッグのほかに、パソコン上でデバッグ可能なソフトウェアシミュレータ機能を搭載
- C ソースコードおよびアセンブリソースコードに対応したソースレベルデバッグ機能
- プログラムの連続実行と C ソースレベル、アセンブラレベルのステップ実行機能
- ハードウェア、ソフトウェア PC ブレーク機能
- 周辺回路制御レジスタを参照可能な EmbSysRegView 機能
- LCD パネルが未搭載の実機に対し、PC 上で LCD パネルの表示をシミュレート可能な LCD パネルシミュレータ機能

8.2 入出力ファイル

8.2.1 入力ファイル

●オブジェクトファイル

ファイル形式: elf 形式のバイナリファイル
ファイル名: <*ファイル名*>.elf
内容: リンカ **ld** で生成した elf 形式のアブソリュートオブジェクトファイルです。`file` および `load` コマンドにより読み込みます。デバッグ情報を含んだオブジェクトファイルを読み込みことで、ソース表示およびシンボリックデバッグが行えます。

●ソースファイル

ファイル形式: テキストファイル
ファイル名: <*ファイル名*>.c （C ソース）
<*ファイル名*>.s （アセンブリソース）
内容: 上記オブジェクトファイルのソースファイルで、ソース表示を行うときに読み込まれます。

●起動コマンドファイル

ファイル形式: テキストファイル
ファイル名: <*ファイル名*>.ini
内容: 起動時に実行させるデバッグコマンドを記述したファイルです。このファイルは起動オプション`-x`により読み込み、実行させます。

●ROM データ

ファイル形式: モトローラ（S1～S3）形式
ファイル名: 任意のファイル名（`.psa`、`.sa`、`.saf` など）
内容: オブジェクトファイル（`.elf`）から作成された、デバッグ情報を含まないオブジェクトファイルです。デバッグ情報を含まないため、ROM データではソースレベルのデバッグは行えません。ターゲット MCU のメモリへプログラム/データをロードする `load` コマンドで使用します。

8.2.2 出力ファイル

●ログファイル

ファイル形式: テキストファイル
ファイル名: gdb.txt
内容: 実行したコマンドと実行結果が出力されます。起動コマンドファイル中で set logging on コマンドを実行することにより出力します。

8.3 起動方法

8.3.1 起動フォーマット

●コマンドラインの一般形

gdb[<*起動オプション*>]
[]は省略可能なことを示します。

●IDE の操作

現在のプロジェクトで最初にデバッガを起動する場合の手順は次のとおりです。

(1) [Run]メニューから[Debug Configration...]を選択して起動構成ダイアログを表示させます。ツールバーの [Debug]ボタンのメニューからも表示させることができます。

(2) C/C++ Application タイプを選択し、プロジェクト名に対応した Debug configuration を選択します。

(3) Debugger タブの GDB Command file にてプロジェクト中の GDB コマンドファイルを選択します。
gdbsim.ini： シミュレータモードでデバッグを行う場合
gdbmini2.ini： ICDmini2(S5U1C17001H2)を使用してデバッグを行う場合
gdbmini3.ini： ICDmini3(S5U1C17001H3)を使用してデバッグを行う場合

(4) [Debug]ボタンをクリックし、デバッグを開始します。

デバッグ開始/起動構成ダイアログは、ツールバーの[デバッグ]ボタンのメニューからも開くことができます。
デバッガの起動後、**IDE** 上でデバッグ操作するには、デバッグパースペクティブを開いてください。

●コネクトモードの選択

gdb は 2 種類のコネクトモードに対応しており、target コマンドで使用するモードを設定します。
IDE 上で作成した GNU17 プロジェクトでは、コネクトモードに対応した起動コマンドファイルとデバッグの構成が用意されています。

ICD Mini モード
ICDmini (S5U1C17001H)を使用してデバッグを行うモードです。プログラムはターゲットボード上で実行されます。

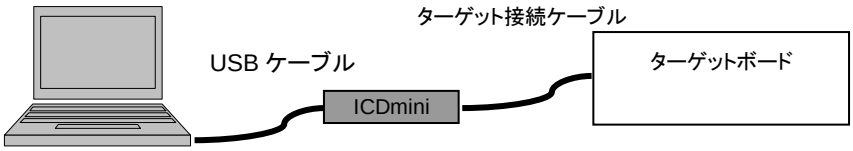


図 8.3.1. ICD を使用するデバッグシステム例

指定方法
コマンド: (gdb) target icd icdmini*x*
IDE での指定方法:
Debugger タブの GDB Command file にて使用する ICDmini に合った gdbmini*x*.ini を選択します。

ICD Mini モードで起動する場合、ホスト PC と ICDmini を USB ケーブルで接続し、ICDmini およびターゲットボードを正しく接続し、これらの電源も投入しておく必要があります。

[ICDmini3 を使用する場合]
ターゲットボードの RESET 端子を ICDmini3 と接続していない場合に、デバッガを起動した後に ICDmini3 の EMU LED(赤色)が消灯したならば、消灯している間にターゲット MCU をリセットしてください。EMU LED(赤色)の消灯時間は 2 秒間です。その間にリセットする必要があります。

シミュレータ(SIM)モード
シミュレータモードは、パソコンのメモリ上でターゲットプログラムの実行をシミュレートするモードで、ほかのツールを必要としません。ただし、ICDmini に依存した機能の使用はできません。

指定方法

コマンド： (gdb) target sim

IDE での指定方法:

Debugger タブの GDB Command file にて gdbsim.ini を選択します。

8.3.2 起動時オプション

デバッガには起動時オプションが用意されています。**IDE** 上で実行する場合、これらの起動時オプションを指定する必要はありません。

--command=*<コマンドファイル名>*
-x *<コマンドファイル名>*

機能: コマンドファイルの指定

説明: このオプションを指定すると、デバッガは起動時に指定のコマンドファイルを読み込んで実行します。

--cd=*<ディレクトリパス文字列>*

機能: カレントディレクトリの変更

説明: このオプションを指定すると、デバッガは起動時に指定のパスをカレントディレクトリに設定します。
省略時は **gdb.exe** が存在するディレクトリに設定されます。

--directory=*<ディレクトリパス文字列>*

機能: ソースファイルディレクトリの変更

説明: このオプションを使用して、ソースファイルのあるディレクトリを指定できます。
このオプションは複数指定することができます。

--model_path=*<ディレクトリパス文字列>*

機能: 機種情報ファイルの格納されているディレクトリの設定

説明: このオプションを使用して、デバッガが MCU モデルに応じた動作をするための機種情報ファイルのあるディレクトリを指定できます。
省略時は **gdb.exe** が存在するディレクトリのサブディレクトリ mcu_model に設定されます。

--model=*<MCU モデル名文字列>*

機能: MCU モデル名の設定

説明: このオプションを指定すると、デバッガは MCU モデルに応じて動作します。MCU の内蔵する Flash ROM へプログラムをロードするには、このオプションを指定するか、c17 model コマンドを実行する必要があります。MCU モデルが指定されていない場合、Flash ROM への書き込みはできません。

8.3.3 コマンドファイルの実行

一連のデバッグコマンドを記述したコマンドファイルを読み込んで、それらのコマンドを実行させることができます。IDE の操作でデバッグを開始する場合、c17model_path コマンド、c17_model コマンド、target コマンド、load コマンドを起動時にコマンドファイルで実行してください。

●コマンドファイルの作成

コマンドファイルは汎用エディタ等でテキストファイルとして作成してください。

●コマンドファイル例

コマンドは 1 行にひとつのみ記述可能です。

例:

```
c17 model_path c:/EPSON/GNU17V3/mcu_model   機種情報ファイルのディレクトリを指定
c17 model 17W23@NOVCCIN                     機種名を指定（電圧レベル 3.3V）
target icd icdmini3                         ターゲット MCU の接続
load                                        プログラムのロード
```

●コマンドファイルの読み込み/実行

コマンドファイルを読み込み、実行させる方法は次の 3 通りです。

1. 起動時オプションによる実行
   デバッガの起動コマンドで-x オプション（または—command オプション）を指定すると、1 つのコマンドファイルをデバッガの起動時に実行させることができます。

   例: `c:¥EPSON¥GNU17V3¥gdb -x icdmini3.ini`

2. IDE 上のデバッグの構成による設定
   IDE のメニュー[Run]>[Debug Configurations…]の Debagger タブの GDB command file にコマンドファイルを指定する。

3. コマンドによる実行
   コマンドファイルの実行用に、source コマンドが用意されています。
   source コマンドは指定されたファイルを読み込み、その中のコマンドを記述順に実行します。

   例: (gdb) source gdbmini3.ini

8.3.4 終了方法

●コマンドの操作

quit コマンド入力によって終了します。
(gdb)quit

●IDE の操作

以下のいずれかの方法でデバッグを終了することができます。
デバッグ終了後、[デバッグ]ビューの表示が終了状態に変わります。

- [実行メニューから[終了]を選択します。
- [デバッグ]ビューの[終了]ボタンをクリックします。
- [コンソール]ビューの[終了]ボタンをクリックします。
- [デバッグ]ビューのコンテキストメニューから終了を選択します。

注: ICDmini を使用してデバッグを行っている場合は、ICDmini の電源を切る前に必ずデバッガを終了させてください。デバッガを実行中に ICDmini の電源を切ると再接続ができなくなることがあります。

8.4 コマンド実行方法

デバッグ機能は IDE のデバッグパースペクティブ上の操作によって実行できます。
ここでは、IDE を使わずに、コマンドを実行させる方法を説明します。コマンドのパラメータなど、詳細については各コマンド説明を参照してください。

8.4.1 コマンドのキーボード入力

デバッガのプロンプト"(gdb)"が表示され、その後にカーソルが点滅していればコマンドが入力できる状態にあります。
そこに、デバッグコマンドを小文字で入力してください。

●コマンド入力の一般形

（gdb）コマンド [パラメータ [パラメータ ... パラメータ]]

コマンドとパラメータ間、およびパラメータ間にはスペースが必要です。

入力ミスの修正には矢印キー（←、→）、[Back Space]キー、[Delete]キーが使用できます。
最後に[Enter]キーを押すと、そのコマンドを実行します。

例: `(gdb) continue` （コマンドのみの入力）
`(gdb) target icd icdmini3` （コマンドとパラメータの入力）

8.4.2 パラメータの入力形式

●数値入力

コマンドでアドレスやデータを指定するパラメータは、デフォルトで 10 進数を入力するように設定されています。16 進数を入力するには、数値の先頭に 0x（または 0X）を付けてください。16 進数として認められる文字は 0～9、a～f、A～F のみです。

ブレーク系コマンドで、即値アドレスを指定する場合は、次のように*をアドレス値の前に付けてください。

例: `(gdb) break *0xc00040`

各コマンドの書式説明で、アドレスパラメータの前に*が付いていないものについては、この必要はありません。

●ソース行番号の指定

ブレーク系コマンドではソース行番号でブレークポイントを指定することができます。ただし、ソース行番号情報を含む elf 形式のオブジェクトファイルをデバッグする場合に限られます。
行番号の指定は次の形式で行います。

Filename:*LineNo.*

Filename ソースファイル名
Filename:は、カレントファイル（現在の PC に対応するコードを含むファイル）内の行番号を指定する場合は省略可能です。

LineNo. 行番号
行番号は 10 進数でのみ指定可能です。

例: `main.c:100`

●シンボルによるアドレス指定

アドレスの指定には、シンボルを使用することもできます。ただし、シンボル情報を含む elf 形式のオブジェクトファイルをデバッグする場合に限られます。

●ファイル名の入力

カレントディレクトリ以外のファイル名は必ずパスを指定してください。
使用可能な文字は、`a～z`、`A～Z`、`0～9`、`/`、`_`のみです。大文字と小文字は区別されません。
また、ドライブ名は/<*ドライブ名*>/形式で指定し、パスの区切りは¥ではなく/を使用してください。

例: `(gdb) file /c/EPSON/gnu17v3/sample/txt/sample.elf`

8.5 コマンドリファレンス

8.5.1 コマンド一覧表

表 8.5.1.1 コマンド一覧表

分類	コマンド	機能	モード対応	
			ICD Mini	SIM
メモリ操作	**c17 fb**	領域のフィル(バイト単位)	○	○
	c17 fh	領域のフィル(16ビット単位)	○	○
	c17 fw	領域のフィル(32ビット単位)	○	○
	x /b	メモリダンプ(バイト単位)	○	○
	x /h	メモリダンプ(16ビット単位)	○	○
	x /w	メモリダンプ(32ビット単位)	○	○
	set {char}	データ入力(バイト単位)	○	○
	set {short}	データ入力(16ビット単位)	○	○
	set {long}	データ入力(32ビット単位)	○	○
	c17 mvb	領域のコピー(バイト単位)	○	○
	c17 mvh	領域のコピー(16ビット単位)	○	○
	c17 mvw	領域のコピー(32ビット単位)	○	○
	c17 df	メモリ内容の保存	○	○
レジスタ操作	**info reg**	レジスタ表示	○	○
	set $	レジスタ変更	○	○
プログラム実行	**continue**	連続実行	○	○
	until	テンポラリブレーク付き連続実行	○	○
	step	ステップ実行(行単位)	○	○
	stepi	ステップ実行(ニーモニック単位)	○	○
	next	スキップ付きステップ実行(行単位)	○	○
	nexti	スキップ付きステップ実行(ニーモニック単位)	○	○
	finish	関数終了	○	○
CPUリセット	**c17 rst**	リセット	○	○
	c17 rstt	ターゲットリセット	○	−
割り込み	**c17 int**	割り込み	−	○
	c17 intclear	割り込み解除	−	○
ブレーク	**break**	ソフトウェアPCブレーク設定	○	○
	tbreak	テンポラリソフトウェアPCブレーク設定	○	○
	hbreak	ハードウェアPCブレーク設定	○	○
	thbreak	テンポラリハードウェアPCブレーク設定	○	○
	delete	ブレーク番号によるブレークの解除	○	○
	clear	ブレーク位置によるブレークの解除	○	○
	enable	ブレークポイントの有効化	○	○
	disable	ブレークポイントの無効化	○	○
	ignore	回数指定付きブレークの無効化	○	○
	info breakpoints	ブレークポイントリストの表示	○	○
	commands	ブレーク時に実行するコマンドの設定	○	○
シンボル情報	**info locals**	ローカルシンボル表示	○	○
	info var	グローバルシンボル表示	○	○
	print	シンボル値の変更	○	○

分類	コマンド	機能	モード対応	
			ICD Mini	SIM
ファイル読み込み	**file**	デバッグ情報の読み込み	○	○
	load	プログラムのロード	○	○
トレース	**c17 tm**	トレースモード設定	－	○
その他	**set output-radix**	変数表示形式の変更	○	○
	set logging	ログ出力の設定	○	○
	source	コマンドファイルの実行	○	○
	target	ターゲットの接続	○	○
	detach	ターゲットの切断	○	○
	pwd	カレントディレクトリの表示	○	○
	cd	カレントディレクトリの変更	○	○
	c17 ttbr	TTBRの設定	－	○
	c17 cpu	CPUタイプの設定	－	○
	c17 chgclkmd	DCLK切替えモード	○	－
	c17 pwul	Flashセキュリティ・パスワードの解除	○	－
	c17 help	ヘルプ	○	○
	c17 model_path	機種別情報ファイルのディレクトリの設定	○	－
	c17 model	MCUモデル名の設定	○	－
	c17 flv	Flashプログラミング電源設定	○	－
	c17 flvs	Flashプログラミング電源設定解除	○	－
	c17 stdin	入出力関数を用いたデータ入力	○	○
	c17 stdout	入出力関数を用いたデータ出力	○	○
	c17 lcdsim	LCDパネルシミュレータの設定	○	－
	quit	デバッガ終了	○	○

モード対応：○＝使用可、－＝使用不可
※上記以外のコマンドは、GNU17V3でサポートされていません。

8.5.2 コマンド詳細説明の見方

コマンド個別の詳細説明は、以下の形式で記述されています。

コマンド名（コマンド機能） [対応モード]

各コマンドの詳細説明は、この形式のコマンド名の見出しから始まります。
[対応モード]は[ICD Mini / SIM]のようにコマンドが使用できるモードを示します。ここに記載されていないモードでは、そのページのコマンドを使用することはできません。
基本的にコマンドは個別に説明しますが、同機能で一部の違いのみにとどまる場合や、同時に説明した方が望ましいコマンドについては、複数をまとめて説明していることもあります。

機　能

コマンドの機能が説明されています。

書　式

コマンドプロンプトに入力する際のコマンド書式とパラメータの内容が記載されています。[ ]で囲まれたパラメータは省略可能です。それ以外は省略できません。*Italic* 文字は数値やシンボルで指定するパラメータを表します。

使用例

コマンドの入力例とその結果などが記載されています。

注　意

コマンドの制限事項や使用上の注意事項などが記載されています。

コマンドによっては、必要に応じて上記以外の項目を設けて説明しているものもあります。

8.5.3 メモリ操作コマンド

c17 fb（領域のフィル, バイト単位）
c17 fh（領域のフィル, 16 ビット単位）
c17 fw（領域のフィル, 32 ビット単位） [ICD Mini / SIM]

機 能

c17 fb 指定のメモリ領域を指定のバイトデータで書き換えます。
c17 fh 指定のメモリ領域を指定の 16 ビットデータで書き換えます。
c17 fw 指定のメモリ領域を指定の 32 ビットデータで書き換えます。

書 式

c17 fb ***StartAddr EndAddr Data***
c17 fh ***StartAddr EndAddr Data***
c17 fw ***StartAddr EndAddr Data***

StartAddr: フィル領域先頭アドレス（10 進数、16 進数、またはシンボル）
EndAddr: フィル領域終了アドレス（10 進数、16 進数、またはシンボル）
Data: 書き込みデータ（10 進数または 16 進数）
条件: 0≦*StartAddr*≦*EndAddr*≦0xffffff、0≦*Data*≦0xff（`c17 fb`）、0≦*Data*≦0xffff（`c17 fh`）、0≦*Data*≦0xffffffff（`c17 fw`）

使用例

■例 1

```
(gdb) c17 fb 0x0 0xf 0x1
Start address = 0x0, End address = 0xf, Fill data = 0x1 .....done
(gdb) x /16b 0x0        (メモリダンプコマンド)
0x0: 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x8: 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
```

アドレス 0x0～0xf のメモリ領域をすべて 0x01 のバイトデータで書き換えます。

■例 2

```
(gdb) c17 fh 0x0 0xf 0x1
Start address = 0x0, End address = 0xe, Fill data = 0x1 .....done
(gdb) x /8h 0x0         (メモリダンプコマンド)
0x0: 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001
```

アドレス 0x0～0xf のメモリ領域をすべて 0x0001 の 16 ビットデータで書き換えます。
（リトルエンディアンの場合）

■例 3

```
(gdb) c17 fw 0x0 0xf 0x1
Start address = 0x0, End address = 0xc, Fill data = 0x1 .....done
(gdb) x /4w 0x0         (メモリダンプコマンド)
0x0: 0x00000001 0x00000001 0x00000001 0x00000001
```

アドレス 0x0～0xf のメモリ領域をすべて 0x00000001 の 32 ビットデータで書き換えます。
（リトルエンディアンの場合）

注 意

- 16 ビットおよび 32 ビットデータの書き込みは、リトルエンディアン形式で行われます。
- データ書き込み範囲はデータサイズに従った境界アドレスに調整されます。
  ```
  (gdb) c17 fw 0x3 0x9 0x0
  ```
  たとえば、上記のように書き込み範囲を指定した場合、先頭アドレス 0x3 と終了アドレス 0x9 は 32 ビットデータ境界アドレスではないため、下位 2 ビットを 00 として（16 ビットの場合は最下位ビットを 0 として）境界アドレスに調整されます。実際に実行されるコマンドは次のとおりで、32 ビットデータアドレス 0x0～0x8（バイトデータアドレス 0x0～0xb）が 0x00000000 のデータで書き換えられます。
  ```
  (gdb) c17 fw 0x0 0x8 0x0
  ```
- 24 ビットを超えるアドレスを指定した場合はエラーとなります。
- データのパラメータは、`c17 fb` は下位 8 ビット、`c17 fh` は下位 16 ビット、`c17 fw` は下位 32 ビットのみが有効で、それを超えた分は無視されます。たとえば、`c17 fb` でデータを 0x100 と指定した場合、0x00 として処理されます。
- 終了アドレスが先頭アドレスより小さい場合、エラーとなります。

X（メモリダンプ） [ICD Mini / SIM]

機 能

メモリの内容を 16 進数でダンプ表示します。データサイズ、表示開始アドレス、表示データ数が指定できます。

書 式

x［/［***Length***］***Size***］［***Address***］

Length: 表示するデータ数（10 進数）
省略時は 1 です。
Size: データサイズ（表示単位）を指定する以下のシンボル
b バイト単位
h 16 ビット単位
w 32 ビット単位（デフォルト）
i 逆アセンブル
Address: 表示開始アドレス（10 進数、16 進数、シンボル、またはレジスタ名）
省略時は、前回の x コマンド実行時に表示した最後のアドレスの次になります。
gdb 起動時のデフォルトは 0x0 です。
条件: 0≦*Length*≦2147483647、0≦*Address*≦0xffffff

表 示

メモリ内容は次のように表示されます。(*Size:*が b/h/w のとき)
Address[<*Symbol*>]: *Data* [*Data* ...]

Address: 各行のデータの先頭アドレスを 16 進数で表示します。
Symbol: 行先頭に表示されるアドレスにシンボル/ラベルが定義されている場合にそのシンボル名が表示されます。また、関数や変数の途中のアドレスを指定した場合には、そのシンボルと 10 進数のオフセット値（<*Symbol*+*n*>）が表示されます。
Data: *Address* から始まるデータが、1 行に最大 16 バイトまで表示されます。

逆アセンブルは次のように表示されます。(*Size:*が i のとき)
Address[<*Symbol*>]: *insn* [ext insn]

Address: 各行のコードのアドレスを 16 進数で表示します。
Symbol: 行先頭に表示されるアドレスにシンボル/ラベルが定義されている場合にそのシンボル名が表示されます。また、関数や変数の途中のアドレスを指定した場合には、そのシンボルと 10 進数のオフセット値（<*Symbol*+*n*>）が表示されます。
insn: *Address* から始まるアセンブラ基本命令が表示されます。
ext insn: *ext* 拡張命令があれば表示されます。

使用例

■例 1

```
(gdb) x
0x0:  0x00000000
```

起動後に、パラメータをすべて省略した場合、"x /1w 0x0"として実行されます。

■例 2

```
(gdb) x /b 0
0x0 <i>:    0xe3
(gdb) x /b 1
0x1 <i+1>: 0xa1
```

Length を省略して *Size* を指定すると、指定データサイズ 1 単位分を表示します。
i はアドレス 0x0 に定義されているシンボルです。変数などの先頭以外のアドレスを指定すると<*シンボル+オフセット*>がシンボルとして表示されます。

■例 3

```
(gdb) x /16h _START_text
0xc00000 <_START_text>: 0x0004 0x00c0 0xc020 0x6c0f 0xa0f1 0xc000 0xc000 0x6c0f
0xc00010 <boot+12>:     0xc000 0xc000 0x1c04 0xdff8 0xdfff 0x1ef5 0x0200 0x6c04
```

Length を指定すると、そのデータ数分表示されます。
コード領域を表示すると、シンボルが定義されていないアドレスでも、<*ラベル+オフセット*>がシンボルとして表示されます。

■例 4

```
(gdb) x /4w 0
0x0 <i>: 0x00001ae3 0x00000000 0x00000000 0x00000000
(gdb) x
0x10:    0x00000000
(gdb) x
0x14:    0x00000000
```

一度 x コマンドを実行した後は、x のみの入力で、前回に続くアドレスから前回と同じデータサイズ 1 単位分をダンプして表示します。

■例 5

```
(gdb) x /w &i
0x0 <i>: 0x00000010
(gdb) x /w i
0x10:    0x00000000
```

アドレスをデータのシンボルで指定する場合、そのシンボルが割り付けられたアドレスを参照する場合は&を付けて入力します。シンボルのみを指定すると、そのデータの値がアドレスとして用いられます。プログラムコード中のラベルは割り付けられたアドレスを示しますので&は不要です。

■例 6

```
(gdb)  x /10i boot
0x8004 <boot>:      ext    0x20
0x8006 <boot+2>:    ld.a   %sp,0x0       sld.a  %sp,0x1000
0x8008 <boot+4>:    call   0x1    call   0x1    (0x00800C)    <main>
0x800a <boot+6>:    nop                  nop
0x800c <main>:      ld     %r0,[0x0]     ld     %r0,[0x0]     <p1>
0x800e <main+2>:    ld     %r1,[0x2]     ld     %r1,[0x2]     <p2>
0x8010 <main+4>:    ext    0x0
0x8012 <main+6>:    ld     %r2,0x64      sld    %r2,0x64
0x8014 <main+8>:    call   0x16   call   0x16   (0x008042)    <memcpy>
0x8016 <main+10>:   ld     %r0,[0x0]     ld     %r0,[0x0]     <p1>
```

指定した 10 命令を表示します。

注 意

- 表示は、リトルエンディアン形式で行われます。
- データサイズに従った境界アドレス以外を指定しても、x コマンドはそのアドレスから表示します。
- 24 ビットを超えるアドレスを指定した場合はエラーとなります。
- Length に 2147483648 以上を指定した場合でもエラーにはなりません。Length は 2147483647 に設定されます。

set { }（データ入力） [ICD Mini / SIM]

機 能

指定のアドレスに指定のデータを書き込みます。

書 式

set {***Size***}***Address***=***Data***

Size: データサイズを指定する以下のシンボル
- **char** バイト単位
- **short** 16 ビット単位（デフォルト）
- **int** 16 ビット単位
- **long** 32 ビット単位

Address: データを書き込むアドレス（10 進数、16 進数、またはシンボル）
Data: 書き込むデータ（10 進数、16 進数、またはシンボル）
条件: 0≦*Address*≦0xffffff、0≦*Data*≦0xff（`set {char}`）、
0≦*Data*≦0xffff（`set {short/int}`）、
0≦*Data*≦0xffffffff （`set {long}`）

使用例

■例 1

```
(gdb) set {char}0x1000=0x55
(gdb) x /b 0x1000
0x1000:  0x55
```

アドレス 0x1000 にバイトデータ 0x55 を書き込みます。

■例 2

```
(gdb) set {short}0x1000=0x5555
(gdb) x /h 0x1000
0x1000:  0x5555
```

アドレス 0x1000 に 16 ビットデータ 0x5555 を書き込みます。

■例 3

```
(gdb) set {long}&i=0x55555555
(gdb) x /w &i
0x0 <i>: 0x55555555
```

long 変数 i に 32 ビットデータ 0x55555555 を書き込みます。

注 意

- 16 ビットおよび 32 ビットデータの書き込みは、リトルエンディアン形式で行われます。
- 24 ビットを超えるアドレスを指定した場合はエラーとなります。
- データのパラメータは、`set {char}`は下位 8 ビット、`set {short}`および `set {int}`は下位 16 ビット、`set {long}`は下位 32 ビットのみが有効で、それを超えた分は無視されます。
  たとえば、`set {char}`でデータを 0x100 と指定した場合、0x00 として処理されます。

c17 mvb（領域のコピー, バイト単位）
c17 mvh（領域のコピー, 16 ビット単位）
c17 mvw（領域のコピー, 32 ビット単位） [ICD Mini / SIM]

機 能

c17 mvb 指定のメモリ領域の内容をバイト単位で別の領域にコピーします。
c17 mvh 指定のメモリ領域の内容を 16 ビット単位で別の領域にコピーします。
c17 mvw 指定のメモリ領域の内容を 32 ビット単位で別の領域にコピーします。

書 式

c17 mvb ***SourceStart SourceEnd Destination***
c17 mvh ***SourceStart SourceEnd Destination***
c17 mvw ***SourceStart SourceEnd Destination***

SourceStart:コピー元の先頭アドレス（10 進数、16 進数、またはシンボル）
SourceEnd: コピー元の終了アドレス（10 進数、16 進数、またはシンボル）
Destination:コピー先の先頭アドレス（10 進数、16 進数、またはシンボル）
条件: 0≦*SourceStart*≦*SourceEnd*≦0xffffff、0≦*Destination*≦0xffffff

使用例

■例 1

```
(gdb) x /16b 0
0x0:  0x00  0x01  0x02  0x03  0x04  0x05  0x06  0x07
0x8:  0x08  0x09  0x0a  0x0b  0x0c  0x0d  0x0e  0x0f
(gdb) c17 mvb 0x0 0x7 0x8
Start address = 0x0, End address = 0x7, Destination address = 0x8 .....done
(gdb) x /16b 0
0x0:  0x00  0x01  0x02  0x03  0x04  0x05  0x06  0x07
0x8:  0x00  0x01  0x02  0x03  0x04  0x05  0x06  0x07
```

アドレス 0x0～0x7 のメモリ領域の内容を、0x8 以降の領域にコピーします。

■例 2

```
(gdb) x /4w 0
0x0 <i>: 0x00000000 0x11111111 0x22222222 0x33333333
(gdb) c17 mvw i i i+4
Start address = 0x0, End address = 0x0, Destination address = 0x4 .....done
(gdb) x /4w 0
0x0 <i>: 0x00000000 0x00000000 0x22222222 0x33333333
```

long 変数 i の内容を 4 バイト後ろにコピーします。

注 意

- コピー元とコピー先のエンディアン形式が異なる場合は、コピーの際にデータ形式が変換されます。
- 24 ビットを超えるアドレスを指定した場合はエラーとなります。
- c17 mvh と c17 mvw では、アドレスはデータサイズに従った境界アドレスに調整されます。
  c17 mvh: アドレスの最下位ビットを 0 として処理します。
  c17 mvw: アドレスの下位 2 ビットを 00 として処理します。
- コピー元終了アドレスがコピー元先頭アドレスより小さい場合、エラーとなります。
- コピー先アドレスがコピー元の先頭アドレスより小さい場合は、先頭アドレスから順にコピーします。逆に、コピー先アドレスがコピー元の先頭アドレスより大きい場合は、終了アドレスから順にコピーします。したがって、コピー元の領域内にコピー先アドレスを設定した場合でもコピーされます。
- コピー先の終了アドレスが 0xffffff を超える場合、0xffffff までがコピーの対象となります。

c17 df（メモリ内容の保存） [ICD Mini / SIM]

機 能

指定範囲のメモリの内容をバイナリ形式､テキスト形式､またはモトローラ S1/S2/S3 形式でファイルに出力します。

書 式

c17 df ***StartAddr EndAddr Type Filename*** [***Append***]

StartAddr: 先頭アドレス（10 進数、16 進数、またはシンボル）
EndAddr: 終了アドレス（10 進数、16 進数、またはシンボル）
Type: ファイルの種類を指定する以下の数値
- **1** バイナリファイル
- **2** テキストファイル
- **3** モトローラ S1 ファイル
- **4** モトローラ S2 ファイル
- **5** モトローラ S3 ファイル

Filename: ファイル名
Append: **a** 追記モード有効
Type = 1 のとき、バイナリファイルの最後にダンプデータを追加出力します。
Type = 2 のとき、テキストファイルの最後にダンプデータを追加出力します。
Type = 3,4,5 のとき、モトローラファイルの最後にフッタレコードは付加しません。
f 追記モード有効
Type = 1 のとき、バイナリファイルの最後にダンプデータを追加出力します。
Type = 2 のとき、テキストファイルの最後にダンプデータを追加出力します。
Type = 3,4,5 のとき、モトローラファイルの最後にフッタレコードを付加します。
省略時、ファイルは新規作成されます。
条件: 0≦*StartAddr*≦*EndAddr*≦0xffff(モトローラ S1 ファイルの場合)
0≦*StartAddr*≦*EndAddr*≦0xffffff(バイナリ/テキスト/モトローラ S2/S3 ファイルの場合)

使用例

■例 1

```
(gdb) c17 df 0x0 0xf 2 dump.txt
Start address = 0x0, End address = 0xf, File type = Text
Processing 00000000-0000000F address.
```

アドレス 0x0～0xf の内容をファイル dump.txt にテキスト形式で書き出します。
（dump.txt の内容）

```
  addr    +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00000000  00 01 02 03 04 05 06 07 00 01 02 03 04 0D 0E 0F
```

■例 2

```
(gdb) c17 df 0x80000 0x80103 5 dump.mot
Start address = 0x80000, End address = 0x80103, File type = Motorola-S3
Processing 00080000-00080103 address.
```

アドレス 0x80000～0x80103 の内容をファイル dump.mot にモトローラ S3 形式で書き出します。
モトローラ S3 形式では 1 行に 32 バイト分が出力されます。一行が 32 バイトに満たない場合は、指定されたアドレスの範囲文バイト数を書き出します。
（dump.mot の内容）

```
S3250008000094D4BA020FCA086120800961881C6F0AA4D4BA020FCA086120800961881C6F0AA8
S3250008002008D3730A0000000000000000000000008D3730A09CA026139A505610CD309613F
                                                :
S30800080100A4D5BABB
S70500000000FA
```

■例 3

```
(gdb) c17 df 0x10 0x1f 2 dump.txt a
```

アドレス 0x10～0x1f の内容をファイル dump.txt の最後にテキスト形式で追加出力します。

■例 4

```
(gdb) c17 df 0x1000 0x1fff 5 dump.mot a ;フッタは出力しません。(最初)
(gdb) c17 df 0x3000 0x3fff 5 dump.mot a ;フッタは出力しません。
(gdb) c17 df 0x5000 0x5fff 5 dump.mot a ;フッタは出力しません。
(gdb) c17 df 0x7000 0x7fff 5 dump.mot f ;フッタを出力します。 (最後)
```

アドレス 0x1000～0x7fff（0x1000 番地ごと）の内容をファイル dump.mot にモトローラ S3 形式で書き出します。*Append* パラメータがない場合、または f を指定されるとモトローラ S3 形式ファイルにはフッタレコードが出力されます。

注 意

- 24 ビットを超えるアドレスを指定した場合はエラーとなります。
- 終了アドレスが先頭アドレスより小さい場合、エラーとなります。

8.5.4 レジスタ操作コマンド

info reg（レジスタ表示） [ICD Mini / SIM]

機 能

CPU レジスタの内容を表示します。

書 式

info reg [***RegisterName***]

RegisterName: 表示するレジスタ名（小文字で指定）
r0-r7、sp、pc、psr
省略時は、すべてのレジスタの内容を表示します。

表 示

レジスタの内容は次のように表示されます。
Register Hexadecimal Decimal

Register: レジスタ名です。
Hexadecimal: レジスタ値を 16 進数で表示します。
Decimal: レジスタ値を 10 進数で表示します。

使用例

■例 1

```
(gdb) info reg r1
r1                  0xaaaaaa     1184810
(gdb) info reg pc
pc                  0x4090       16528
```

レジスタ名を指定すると、そのレジスタの内容のみを表示します。

■例 2

```
(gdb) info reg
r0           0xd20           3360
r1           0xaaaaaa        1184810
r2           0xaaaaaa        1184810
r3           0xaaaaaa        1184810
r4           0x690           1680
r5           0xaaaaaa        1184810
r6           0x0             0
r7           0xaaaaaa        1184810
sp           0x7f8           2040
pc           0xc00030        12582960
psr          0x2             2
```

注 意

レジスタ名は小文字で指定してください。大文字で指定したり、存在しないレジスタ名を指定するとエラーとなります。

set $（レジスタ変更） [ICD Mini / SIM]

機 能

CPU レジスタの値を変更します。

書 式

set $*RegisterName*=*Value*

RegisterName: 変更するレジスタ名（小文字で指定）
r0-r7、sp、pc、psr
Value: 設定する 24 ビットデータ（10 進数、16 進数、またはシンボル）
条件: 0≦*Value*≦0xffffff

使用例

```
(gdb) set $r1=0x10000
(gdb) info reg r1
r1             0x10000  65536
(gdb) set $pc=main
```

設定値には数値以外にシンボルも使用可能です。

注 意

- 24 ビットを超える値を指定した場合は、下位 24bit のみ有効となります。
- 設定値の内容はチェックしていません。PC に 16 ビット境界アドレス以外、同様に SP に 32 ビット境界アドレス以外の値を設定してもエラーとはなりません。ただし、実際にレジスタを変更する時点では下位ビットが切り捨てられ、強制的に境界アドレスに設定されます。

8.5.5 プログラム実行コマンド

continue（連続実行） [ICD Mini / SIM]

機 能

現在の PC アドレスからターゲットプログラムを実行します。
プログラムの実行は次のいずれかの要因によってブレークするまで継続します。

- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック

また、ブレーク条件の成立により停止しているターゲットプログラムを再実行させる場合、パラメータの指定により、そのブレークを指定回数だけ無効にすることができます。

書 式

continue [***IgnoreCount***]
cont [***IgnoreCount***] （省略形）

IgnoreCount: ブレーク回数指定（10 進数または 16 進数）
設定した回数のブレーク条件が成立するまで実行を継続します。

使用例

■例 1

```
(gdb) continue
Continuing.

Breakpoint 1, main () at main.c:13
```

IgnoreCount を省略して continue を実行すると、ターゲットプログラムは現在の PC アドレスから実行を開始し、最初のブレーク条件成立により停止します。

■例 2

```
(gdb) cont 5
Breakpoint 1, main () at main.c:13
```

IgnoreCount に 5 を指定しているため、実行開始後に現れる 5 - 1 = 4 回のブレーク条件成立を無視し、5 回目の条件成立でブレークします。この例では、main.c の 13 行目に設定した PC ブレークポイント（ブレーク番号 1）で停止していたターゲットプログラムの実行を再開したため、その PC ブレークポイントに 5 回ヒットしたところで停止します。
これは、次のコマンドを実行した場合と同じです。

```
(gdb) ignore 1 4
(gdb) continue
```

注 意

- プログラムの先頭から実行するには、continue の前に c17 rst（リセット）を実行してください。
- *IgnoreCount* を指定した continue の実行は、ターゲットプログラムが一度以上実行され、ブレーク条件の成立により停止していることが条件です。なお、[中断]ボタンによるブレークはこの場合のブレーク条件の成立とはみなされません。ターゲットプログラムが一度もブレークしていない状態で *IgnoreCount* を指定しても無視されます。
- ターゲットプログラムがひとつのブレーク要因で停止しているとき、そのブレーク設定を解除してから *IgnoreCount* を指定して continue を実行するとエラーになります。これは、他のブレーク条件が設定されていても同じです。
- ターゲットプログラムが停止したブレーク以外のブレーク条件を指定回数分無視したいときには、ignore コマンドでブレーク条件とブレークヒットを無視する回数を指定し、continue コマンドをパラメータなしで実行してください。

until（テンポラリブレーク付き連続実行） [ICD Mini / SIM]

機 能

現在の PC アドレスからターゲットプログラムを実行します。
テンポラリブレークを 1 カ所指定でき、そのブレークポイントを実行前に停止させることができます。
このテンポラリブレークにはハードウェア PC ブレークが使用され、一度ブレークすると解除されます。
テンポラリブレークを指定した場合は、C ソース以外も連続実行します。

指定したブレークポイントにヒットしない場合、プログラムの実行は次のいずれかの要因によってブレークするまで継続します。
- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック
- 現在のレベル（関数内）から上位レベルにリターンした時点
- アセンブリソースまたはソース情報がない場合は現在の命令のみを実行

書 式

`until` ***Breakpoint***

Breakpoint: テンポラリブレークポイント
以下の 3 種類で指定可能です。
- 関数名
- ソースファイル名:行番号、または行番号のみ
- *アドレス（10 進数、16 進数、またはシンボル）

条件: 0≦アドレス≦0xffffff

使用例

■例 1

```
(gdb) until main
main () at main.c:10
```

テンポラリブレークポイントを関数名で指定し、ターゲットプログラムを実行します。
`main()` 内の最初の C 命令（ニーモニックに展開されるもの）を実行前にブレークします。ブレーク時の PC はその命令の先頭アドレス（展開後の先頭ニーモニックのアドレス）を示します。

■例 2

```
(gdb) until main.c:10
main () at main.c:10
```

テンポラリブレークポイントを行番号で指定し、ターゲットプログラムを実行します。
ここでは、"ソースファイル名:行番号"の書式で指定していますが、現在の PC アドレスがある C ソース内であれば `until 10` のように行番号のみで指定できます。アセンブリソースの場合は常にソースファイル名が必要です。
このコマンドの実行により、`main.c` 内の 10 行目の C 命令を実行前にブレークします。
ブレーク時の PC はその命令の先頭アドレス（展開後の先頭ニーモニックのアドレス）を示します。10 行目に実コードを持つ命令がない（ニーモニックに展開されない）場合は、それ以降最初に現れる実コードを持つ命令の先頭でブレークします。

■例 3

```
(gdb) until *0xc0001e
main () at main.c:10
```

テンポラリブレークポイントをアドレスで指定し、ターゲットプログラムを実行します。
そのアドレスの命令を実行前にブレークします。
次のようにシンボルも使用可能です。

```
(gdb) until *main
main () at main.c:7
```

*を付加すると、関数名でもアドレスと見なされます。

注 意

- プログラムの先頭から実行するには、until の前に c17 rst（リセット）を実行してください。
- ニーモニックに展開されない C ソース行をテンポラリブレークポイントとして設定してもその行ではブレークしません。それ以降の最初に実行されるニーモニックのアドレスでブレークします。
- テンポラリブレークポイントは、以下の行には設定できません。エラーとなります。
  - 先頭の ext 命令を除く拡張命令の行
  - 遅延命令の行（遅延分岐命令の次の行）
- テンポラリブレークポイントを、存在しない関数名や行番号で指定するとエラーとなります。
- テンポラリブレークポイントをアドレス値で指定する場合、24 ビットを超える値を指定するとエラーとなります。
- テンポラリブレークポイントをアドレス値で指定する場合、アドレスを奇数の値で指定すると最下位ビットを 0 として 16 ビット境界に補正されます。

step（ステップ実行、行単位）
stepi（ステップ実行、ニーモニック単位） [ICD Mini / SIM]

機 能

現在の PC からターゲットプログラムをシングルステップ実行します。呼び出される関数やサブルーチン内もステップ実行します。
step: ソースの行単位にシングルステップ実行します。C ソースは 1 行の C 命令（展開された複数のニーモニックすべて）を 1 ステップとして実行します。アセンブリソースの場合は stepi と同機能です。
stepi: ニーモニック単位にシングルステップ実行します。

実行するステップ数を指定できます。
その全ステップが終了する前でも、プログラムの実行は次のいずれかの要因によって停止します。
- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック

書 式

step [*Count*]
stepi [*Count*]

Count: 実行ステップ数（10 進数、または 16 進数）
省略時は 1 ステップです。
条件: 1≦*Count*≦0x7fffffff

使用例

■例 1
```
(gdb) step
```
現在の PC が示すソース行を実行します。

■例 2
```
(gdb) stepi
```
現在の PC が示すアドレスの命令（ニーモニック単位）を実行します。

■例 3
```
(gdb) step 10
sub (k=5) at main.c:20
```
現在の PC が示すソース行から 10 行分を実行します。

■例 4
```
(gdb) stepi 10
main () at main.c:13
```
現在の PC が示すアドレスから 10 命令（ニーモニック単位）を実行します。

注 意

- ソース情報（オブジェクトファイルに含まれるデバッグ情報）のないアドレスからステップ実行することはできません。continue などにより連続実行することは可能です。
- プログラムの先頭から実行するには、step/stepi の前に c17 rst（リセット）を実行してください。
- stepi であっても、ext による拡張命令は、拡張命令全体（2 または 3 命令）を 1 ステップとして実行します。
- シングルステップ動作時も割り込みを受け付けます。
  また、halt 命令および slp 命令はシングルステップ動作時でも実行され、それによってスタンバイモードになります。これは、外部割り込みを発生させることによって解除されます。また、[中断]ボタンで解除することもできます。

next（スキップ付きステップ実行、行単位）
nexti（スキップ付きステップ実行、ニーモニック単位） [ICD Mini / SIM]

機 能

現在の PC からターゲットプログラムをシングルステップ実行します。基本的な動作は step/stepi と同じですが、関数コールやサブルーチンコールは、それより下位レベルの呼び出される関数やサブルーチンをすべて含め 1 ステップとして実行します（呼び出される関数やサブルーチン内はリターンするまで連続実行されます）。
next: ソースの行単位にシングルステップ実行します。C ソースは 1 行の C 命令（展開された複数のニーモニックすべて）を 1 ステップとして実行します。アセンブリソースの場合は nexti と同機能です。
nexti: ニーモニック単位にシングルステップ実行します。

実行するステップ数を指定できます。
その全ステップが終了する前でも、プログラムの実行は次のいずれかの要因によって停止します。
- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック

書 式

next [*Count*]
nexti [*Count*]

Count: 実行ステップ数（10 進数、または 16 進数）
省略時は 1 ステップです。
条件: 1≦*Count*≦0x7fffffff

使用例

■例 1
```
(gdb) next
```
現在の PC が示すソース行を実行します。ソースが関数コールまたはサブルーチンコールの場合は、呼び出す関数/サブルーチンもリターンするまで実行します。

■例 2
```
(gdb) nexti
```
現在の PC が示すアドレスの命令（ニーモニック単位）を実行します。命令がサブルーチンコールの場合は、呼び出すサブルーチンもリターンするまで実行します。

■例 3
```
(gdb) next 10
sub (k=5) at main.c:20
```
現在の PC が示すソース行から 10 行分を実行します。

■例 4
```
(gdb) nexti 10
main () at main.c:13
```
現在の PC が示すアドレスから 10 命令（ニーモニック単位）を実行します。

注 意

- ソース情報（オブジェクトファイルに含まれるデバッグ情報）のないアドレスからステップ実行することはできません。continue などにより連続実行することは可能です。
- プログラムの先頭から実行するには、next/nexti の前に c17 rst（リセット）を実行してください。
- nexti であっても、ext による拡張命令は、拡張命令全体（2 または 3 命令）を 1 ステップとして実行します。
- シングルステップ動作時も割り込みを受け付けます。
  また、halt 命令および slp 命令はシングルステップ動作時でも実行され、それによってスタンバイモードになります。これは、外部割り込みを発生させることによって解除されます。また、[中断]ボタンで解除することもできます。

finish（関数終了） [ICD Mini / SIM]

機 能

現在の PC からターゲットプログラムを実行し、現在の関数から上位レベルにリターンした時点で停止します。リターン位置の命令は実行しません。
ただし、リターンの前でも、プログラムの実行は次のいずれかの要因によって停止します。

- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック

書 式

finish

使用例

(gdb) **finish**
現在の PC アドレスからターゲットプログラムを実行し、リターン後に停止します。

注 意

ブートルーチンのような最上位レベルで finish を実行した場合、プログラムは停止しません。ブレークが設定されていない場合は、[中断]ボタンで停止させてください。

8.5.6 CPU リセットコマンド

c17 rst（リセット） [ICD Mini / SIM]

機 能

CPU をリセットします。
ICD Mini モードの場合、ターゲットリセットを行います。
これによる初期設定内容は以下のとおりです。

(1) CPU の内部レジスタ
r0～r7: 0x000000
pc: ブートアドレス（トラップテーブルのリセットベクタ）
sp: 0xffffffc
psr: 0x00（IL = 000、IE = 0、CVZN = 0000）

(2) 実行カウンタを 0 に設定

書 式

```
c17 rst
```

使用例

```
(gdb) c17 rst
```

CPU をリセットします。

注 意

- メモリ内容、ブレークやトレースなどのデバッグステータスはリセットされません。
- ICD Mini モードで使用する場合、バスや I/O の状態は保持されます。

c17 rstt（ターゲットリセット） [ICD Mini]

機 能

ターゲットのリセット入力端子へリセット信号を出力します。

書 式

c17 rstt

使用例

```
(gdb) c17 rstt
TARGET resetting ..................... done
```

ターゲットをリセットします。

ターゲットリセット失敗時、のメッセージ

```
TARGET resetting ..................... failure
```

注 意

- c17 rstt コマンドは ICD Mini モード時のみ使用可能です。
- このコマンドを実行するには、ターゲットボードにリセット入力端子を実装する必要があります。

8.5.7 割り込みコマンド

c17 int（割り込み） [SIM]

機 能

割り込みの発生をシミュレートします。
本コマンドで割り込み番号を指定すると、次のプログラム実行開始時にその割り込みが発生します。

書 式

c17 int [***No Level***]

No: 割り込み番号（10 進数、16 進数、またはシンボル）
Level: 割り込みレベル（10 進数、16 進数、またはシンボル）
条件: 0≦*No*≦0x1f、0≦*Level*≦7

使用例

■例 1

```
(gdb) c17 int
```

パラメータを指定しない場合、NMI を発生します。

■例 2

```
(gdb) c17 int 3 6
```

マスク可能な割り込みの番号と割り込みレベルが設定できます。

注 意

- `c17 int` コマンドはシミュレータモード時のみ使用可能です。
- 割り込み番号は 0〜31 の範囲内で指定してください。これを超えた場合、エラーとなります。
- 割り込みレベルは 0〜7 の範囲内で指定してください。これを超えた場合、エラーとなります。
- シミュレータモードでも TTBR は有効です。

c17 intclear（割り込み解除） [SIM]

機　能

割り込みの解除をシミュレートします。
本コマンドで割り込み番号を指定すると、その割り込みが解除されます。

書　式

c17 intclear [***No***]

No: 割り込み番号（10 進数、16 進数、またはシンボル）
条件: 0≦*No*≦0x1f

使用例

```
(gdb) c17 int 3 6
(gdb) continue
(gdb) c17 intclear 3
```

割り込み番号 3 の割り込みを解除します。

注　意

- c17 intclear コマンドはシミュレータモード時のみ使用可能です。
- 割り込み番号は 0～31 の範囲内で指定してください。これを超えた場合、エラーとなります。

8.5.8 ブレーク設定コマンド

break（ソフトウェア PC ブレーク設定）
tbreak（テンポラリソフトウェア PC ブレーク設定） [ICD Mini / SIM]

機 能

ソフトウェア PC ブレークポイントを設定します。最大 200 カ所のソフトウェア PC ブレークポイントが設定可能です。ただし、ICD Mini モードでターゲットボード上の ROM にブレークポイントを設定する場合、本コマンドは hbreak/thbreak コマンドと同様に動作し、ハードウェア PC ブレークポイントを設定します。
プログラムの実行により PC が設定したアドレスに一致すると、そのアドレスの命令を実行前にブレークします。
ブレークポイントは関数名、行番号、アドレスで指定できます。
break コマンドと tbreak コマンドのブレーク機能はどちらも同じで、違いは次のとおりです。
break: break で設定したブレークポイントは、プログラムの実行によるブレークの発生では解除されません。
tbreak: tbreak で設定したブレークポイントは、一度ブレークすると解除されます。

書 式

break ［***Breakpoint***］
tbreak ［***Breakpoint***］

Breakpoint: ブレークポイント
以下の 3 種類で指定可能です。
- 関数名
- ソースファイル名:行番号、または行番号のみ
- *アドレス（10 進数、16 進数、またはシンボル）

省略時は、現在の PC が示すアドレスに設定されます。
条件: 0≦アドレス≦0xfffffe

使用例

■例 1

```
(gdb) break main
Breakpoint 1 at 0xc0001e: file main.c, line 10.
(gdb) continue
Continuing.
Breakpoint 1, main () at main.c:10
```

関数名を指定し、ソフトウェア PC ブレークポイントを設定します。
ターゲットプログラムを実行すると、main()内の最初の C 命令（ニーモニックに展開されるもの）を実行前にブレークします。ブレーク時の PC はその命令の先頭アドレス（展開後の先頭ニーモニックのアドレス）を示します。

■例 2

```
(gdb) tbreak main.c:10
Breakpoint 1 at 0xc0001e: file main.c, line 10.
```

行番号を指定し、テンポラリソフトウェア PC ブレークポイントを設定します。
ここでは、"ソースファイル名:行番号"の書式で指定していますが、現在の PC アドレスがある C ソース内であれば tbreak 10 のように行番号のみで指定できます。アセンブリソースの場合は常にソースファイル名が必要です。
指定の行に実コードを持つ命令がない（ニーモニックに展開されない）場合は、それ以降最初に現れる実コードを持つ命令の先頭にブレークポイントが設定されます。
ターゲットプログラムを実行すると、main.c の 10 行目の C 命令を実行前にブレークします。ブレーク時の PC はその命令の先頭アドレス（展開後の先頭ニーモニックのアドレス）を示します。10 行目に実コードを持つ命令がない場合は、それ以降最初に現れる実コードを持つ命令の先頭でブレークします。tbreak で設定したため、ブレークポイントはブレーク後に解除されます。

■例 3

```
(gdb) break *0xc0001e
Note: breakpoint 1 also set at pc 0xc0001e.
Breakpoint 2 at 0xc0001e: file main.c, line 10.
```

アドレスを指定し、ソフトウェア PC ブレークポイントを設定します。
ターゲットプログラムを実行すると、そのアドレスの命令を実行前にブレークします。
次のようにシンボルも使用可能です。

```
(gdb) tbreak *main
Breakpoint 3 at 0xc0001c: file main.c, line 7.
```

*を付加すると、関数名でもアドレスと見なされます。

ブレークポイントの管理方法

ブレークポイントを設定すると、ブレークの種類にかかわらず、1 から順にブレーク番号が付けられます（上記例参照）。この番号は、後からブレークポイントを個別に無効化/有効化、あるいは削除するときに必要となります。なお、ブレークポイントを削除しても、デバッガを終了するまで、番号の繰り上げ（消えた番号の再利用）は起きません。
設定したブレークポイントを操作するには、以下のコマンドを使用します。

`disable`	ブレークポイントの無効化
`enable`	ブレークポイントの有効化
`delete` または `clear`	ブレークポイントの削除
`ignore`	ブレークを無効にする回数を指定
`info breakpoints`	ブレークポイントリストの表示

詳細は、各コマンドの説明を参照してください。

注 意

- ソフトウェア PC ブレークポイントは、最大 200 カ所まで設定可能です。これを超えるとエラーになります。デバッガが他の機能で使用するソフトウェア PC ブレークについてもこのブレーク本数に含まれますので注意してください。
- ニーモニックに展開されない C ソース行をソフトウェア PC ブレークポイントに指定しても、その行はブレークポイントとして設定されません。ソフトウェア PC ブレークポイントは、それ以降の最初に実行される命令のアドレスに設定されます。
- 関数名や関数内先頭の C ソース行をソフトウェア PC ブレークポイントに設定した場合、プログラムの実行がブレークするのは、関数内の最初の C ソース（ニーモニックに展開される命令）の先頭アドレスです。関数の先頭にはコンパイル時にレジスタを退避させる `ld` 命令が挿入されますが、その命令はブレーク前に実行されます。この命令の実行前にブレークさせるには、そのアドレス値でソフトウェア PC ブレークポイントを指定してください。
- ソフトウェア PC ブレークポイントは、以下の行には設定できません。
  - 先頭の `ext` 命令を除く拡張命令の行
  - 遅延命令の行（遅延分岐命令の次の行）
- 存在しない関数名や行番号でソフトウェア PC ブレークポイントを指定するとエラーとなります。
- ソフトウェア PC ブレークポイントをアドレス値で指定する場合、アドレスを奇数の値で指定すると最下位ビットを 0 として 16 ビット境界に補正されます。また、24 ビットを超えるアドレスを指定した場合はエラーとなります。
- ソフトウェア PC ブレークは、`brk` 命令の埋め込みにより実現しています。そのため、命令の埋め込みができないターゲットボード上の ROM に対しては使用できません。brk 命令を埋め込みできない場合、hbreak/thbreak コマンドと同様に、ハードウェア PC ブレークを設定します。brk 命令を埋め込みできないことがあらかじめ明らかであれば、hbreak/thbreak コマンドを使用してください。

- BRK 命令をソースに埋め込んだ時の処理
  break コマンドではなく、ユーザーアプリケーションソースに埋め込まれた BRK 命令を実行した場合には、ソフトブレーク後、自動的に PC+=2 を行います（以下の例では①+2 で②になる）。Flash メモリ等 ROM で実行する場合、BRK 命令をソースに埋め込むことにより、ハードブレークの数を越える個所でブレークが可能になります。

例）

```
sample.c
void main()
{
    ・                  ;   < ここで continue
    ・
    ・
    a = b + 1           ;
    iRet = sub( a )     ;   < ③
    asm( "brk" )        ;   < ①brk 命令埋め込み
    if ( iRet   1 ) {   ;   < ②ここで停止（BRK 命令のアドレス+2）
    b -= 2              ;
}
    ・
    ・
    ・
```

上記例で①にソフトブレーク設定した場合は①で停止します。これは、メモリに埋め込まれた BRK 命令が、break コマンドによるものか、ソースに埋め込まれていたものかが、デバッガには判断できないためです。また、次に実行する前に、ブレークポイント設定を解除する必要があります。①にハードブレーク設定した場合は②で停止します。③を next 後は②で停止します。

hbreak（ハードウェア PC ブレーク設定）
thbreak（テンポラリハードウェア PC ブレーク設定） [ICD Mini / SIM]

機 能

ハードウェア PC ブレークポイントを設定します。ハードウェア PC ブレークポイントは ICD Mini モードでは、機種により 1 ～ 4 箇所設定可能です。SIM モードでは、1 箇所のみ設定可能です。
プログラムの実行により PC が設定したアドレスに一致すると、そのアドレスの命令を実行前にブレークします。
ブレークポイントは関数名、行番号、アドレスで指定できます。
hbreak コマンドと thbreak コマンドのブレーク機能はどちらも同じで、違いは次のとおりです。
hbreak: hbreak で設定したブレークポイントは、プログラムの実行によるブレークの発生では解除されません。
thbreak: thbreak で設定したブレークポイントは、一度ブレークすると解除されます。

書 式

hbreak [***Breakpoint***]
thbreak [***Breakpoint***]

Breakpoint: ブレークポイント
以下の 3 種類で指定可能です。
- 関数名
- ソースファイル名:行番号、または行番号のみ
- *アドレス（10 進数、16 進数、またはシンボル）

省略時は、現在の PC が示すアドレスに設定されます。
条件: 0≦アドレス≦0xfffffe

使用例

■例 1

```
(gdb) hbreak main
Hardware assisted breakpoint 1 at 0xc0001e: file main.c, line 10.
(gdb) continue
Continuing.

Breakpoint 1, main () at main.c:10
```

関数名を指定し、ハードウェア PC ブレークポイントを設定します。
ターゲットプログラムを実行すると、main()内の最初の C 命令（ニーモニックに展開されるもの）を実行前にブレークします。ブレーク時の PC はその命令の先頭アドレス（展開後の先頭ニーモニックのアドレス）を示します。

■例 2

```
(gdb) thbreak main.c:10
Hardware assisted breakpoint 1 at 0xc0001e: file main.c, line 10.
```

行番号を指定し、テンポラリハードウェア PC ブレークポイントを設定します。
ここでは、"ソースファイル名:行番号"の書式で指定していますが、現在の PC アドレスがある C ソース内であれば thbreak 10 のように行番号のみで指定できます。アセンブリソースの場合は常にソースファイル名が必要です。
指定の行に実コードを持つ命令がない（ニーモニックに展開されない）場合は、それ以降最初に現れる実コードを持つ命令の先頭にブレークポイントが設定されます。
ターゲットプログラムを実行すると、main.c 内の 10 行目の C 命令を実行前にブレークします。ブレーク時の PC はその命令の先頭アドレス（展開後の先頭ニーモニックのアドレス）を示します。
10 行目に実コードを持つ命令がない場合は、それ以降最初に現れる実コードを持つ命令の先頭でブレークします。
thbreak で設定したため、ブレークポイントはブレーク後に解除されます。

■例 3

```
(gdb) hbreak *0xc0001e
Note: breakpoint 1 also set at pc 0xc0001e.
Hardware assisted breakpoint 2 at 0xc0001e: file main.c, line 10.
```

アドレスを指定し、ハードウェア PC ブレークポイントを設定します。
ターゲットプログラムを実行すると、そのアドレスの命令を実行前にブレークします。
次のようにシンボルも使用可能です。

```
(gdb) thbreak *main
Hardware assisted breakpoint 3 at 0xc0001c: file main.c, line 7.
```

*を付加すると、関数名でもアドレスと見なされます。

ブレークポイントの管理方法

ブレークポイントを設定すると、ブレークの種類にかかわらず、1 から順にブレーク番号が付けられます（上記例参照）。
この番号は、後からブレークポイントを個別に無効化/有効化、あるいは削除するときに必要となります。
なお、ブレークポイントを削除しても、デバッガを終了するまで、番号の繰り上げ（消えた番号の再利用）は起きません。
設定したブレークポイントを操作するには、以下のコマンドを使用します。

`disable`	ブレークポイントの無効化
`enable`	ブレークポイントの有効化
`delete` または `clear`	ブレークポイントの削除
`ignore`	ブレークを無効にする回数を指定
`info breakpoints`	ブレークポイントリストの表示

詳細は、各コマンドの説明を参照してください。

注 意

- 有効なハードウェア PC ブレークポイントは、ICD Mini モードでは機種により 1 ～4 箇所に、SIM モードでは 1 箇所に設定可能です。これ以上に設定する場合は、無効化の状態で設定可能です。デバッガがテンポラリブレーク付きで使用するハードウェア PC ブレークについてもこのブレーク本数に含まれますので注意してください。
- ニーモニックに展開されない C ソース行をハードウェア PC ブレークポイントに指定しても、その行はブレークポイントとして設定されません。ハードウェア PC ブレークポイントは、それ以降の最初に実行される命令のアドレスに設定されます。
- 関数名や関数内先頭の C ソース行をハードウェア PC ブレークポイントに設定した場合、プログラムの実行がブレークするのは、関数内の最初の C ソース（ニーモニックに展開される命令）の先頭アドレスです。

  関数の先頭にはコンパイル時にレジスタを退避させる `ld` 命令が挿入されますが、その命令はブレーク前に実行されます。この命令の実行前にブレークさせるには、そのアドレス値でブレークポイントを指定してください。
- ハードウェア PC ブレークポイントは、以下の行には設定できません。設定すると、ターゲットプログラムが実行できなくなります（解除すれば問題ありません）。
  - 先頭の `ext` 命令を除く拡張命令の行
  - 遅延命令の行（遅延分岐命令の次の行）
- 存在しない関数名や行番号でハードウェア PC ブレークポイントを指定するとエラーとなります。
- ハードウェア PC ブレークポイントをアドレス値で指定する場合、アドレスを奇数の値で指定すると最下位ビットを 0 として 16 ビット境界に補正されます。また、24 ビットを超えるアドレスを指定した場合はエラーとなります。

delete（ブレーク番号によるブレークの解除） [ICD Mini / SIM]

機 能

設定されているブレークポイントをすべて、あるいはブレーク番号を指定して個別に削除します。

書 式

delete [***BreakNo***]

BreakNo: ブレーク番号（10 進数）
省略すると、すべてのブレークポイントが削除されます。

使用例

```
(gdb) info breakpoints  (ブレークポイントリストの表示)
Num  Type         Disp   Enb   Address     What
1    breakpoint   keep   y     0x00c00038  in sub at main.c:20
2    breakpoint   keep   y     0x00c00030  in main at main.c:14
3    breakpoint   keep   y     0x00c0003c  in sub at main.c:22
```

上記のとおり、ブレークポイントが設定されているものとします。

■例 1

```
(gdb) delete 1 2
(gdb) info breakpoints
Num  Type         Disp   Enb   Address     What
3    breakpoint   keep   y     0x00c0003c  in sub at main.c:22
```

ブレーク番号を指定すると、そのブレークのみ解除できます。複数のブレーク番号を一度に指定可能です。

■例 2

```
(gdb) delete
(gdb) info breakpoints
No breakpoints or watchpoints.
```

ブレーク番号を省略すると、すべてのブレークを解除します。

注 意

- ブレーク番号は、ブレーク設定時に個々のブレークポイントに対して 1 から順に割り当てられます。削除する時点で不明な場合は、上記例のとおり `info breakpoints` コマンドで確認してください。
- `delete` はブレークの設定を完全に削除します。一時的にブレークポイントを無効にしたい場合は、`disable` コマンドや `ignore` コマンドを使用してください。
- 設定されていないブレーク番号が指定された場合、"No breakpoint number N."を表示して削除は行いません。

clear（ブレーク位置によるブレークの解除） [ICD Mini / SIM]

機 能

設定されている PC ブレークポイント（テンポラリブレークを含む）を、設定位置（関数名、行番号、アドレス）を指定して個別に削除します。

書 式

clear ***Breakpoint***

Breakpoint: ブレークポイント
以下の 3 種類で指定可能です。
- 関数名
- ソースファイル名:行番号、または行番号のみ
- *アドレス（10 進数、16 進数、またはシンボル）

条件: 0≦アドレス≦0xffffe

使用例

```
(gdb) info breakpoints（ブレークポイントリストの表示）
Num  Type          Disp   Enb   Address     What
1    breakpoint    keep   y     0x00c0001e  in main at main.c:10
2    breakpoint    keep   y     0x00c00038  in sub at main.c:20
3    breakpoint    keep   y     0x00c0003c  in sub at main.c:22
4    breakpoint    keep   y     0x00c00042  in sub at main.c:22
```

上記のとおり、ブレークポイントが設定されているものとします。ブレーク番号 3 と 4 は異なるアドレスですが、C ソースで見ると 1 行に設定されています（ASSEMBLY 表示のアドレスにブレークポイントを設定したケース）。

■例 1

```
(gdb) clear main.c:22
Deleted breakpoints 4 3
(gdb) info breakpoints
Num  Type          Disp   Enb   Address     What
1    breakpoint    keep   y     0x00c0001e  in main at main.c:10
2    breakpoint    keep   y     0x00c00038  in sub at main.c:20
```

行番号を指定すると、そのソース行に設定されているすべてのブレークポイントを解除します。

■例 2

```
(gdb) clear main
Deleted breakpoint 1
(gdb) info breakpoints
Num  Type          Disp   Enb   Address     What
2    breakpoint    keep   y     0x00c00038  in sub at main.c:20
```

関数名を指定すると、その関数内の最初の C 命令（ニーモニックに展開されるもの）に設定されているブレークポイントを解除します。"break 関数名"等で設定したブレークポイントの削除に使用します。

注 意

- clear はブレークの設定を完全に削除します。一時的にブレークポイントを無効にしたい場合は、disable コマンドや ignore コマンドを使用してください。
- ブレークポイントに設定されていない関数名、行番号、アドレスを指定するとエラーとなります。

enable（ブレークポイントの有効化）
disable（ブレークポイントの無効化） [ICD Mini / SIM]

機 能

enable: 無効に設定されているブレークポイントを有効な状態に切り換えます。
disable: 有効に設定されているブレークポイントを無効な状態に切り換えます。

ブレークコマンドでブレークポイントを設定した段階では、ブレークは有効となります。disable コマンドはこれを、ブレークポイントを削除せずに無効にします。
enable コマンドで有効に戻すまでは、そのブレークポイントではブレークしません。

書 式

enable [***BreakNo***]
disable [***BreakNo***]

BreakNo: ブレーク番号（10 進数）
省略すると、すべてのブレークポイントが対象になります。

使用例

```
(gdb) info breakpoints（ブレークポイントリストの表示）
Num  Type          Disp   Enb   Address     What
1    breakpoint    keep   y     0x00c0001c  in main at main.c:7
2    breakpoint    keep   y     0x00c0001e  in main at main.c:10
3    breakpoint    keep   y     0x00c00028  in main at main.c:13
4    breakpoint    keep   y     0x00c00038  in sub at main.c:20
```

上記のとおり、ブレークポイントが設定されているものとします。有効なブレークポイントは Enb が y と表示されます。

■例 1

```
(gdb) disable 1 3
(gdb) info breakpoints
Num  Type          Disp   Enb   Address     What
1    breakpoint    keep   n     0x00c0001c  in main at main.c:7
2    breakpoint    keep   y     0x00c0001e  in main at main.c:10
3    breakpoint    keep   n     0x00c00028  in main at main.c:13
4    breakpoint    keep   y     0x00c00038  in sub at main.c:20
```

ブレーク番号を付けて disable コマンドを実行すると、そのブレークのみ無効にできます。
複数のブレーク番号を一度に指定可能です。無効なブレークポイントは Enb が n と表示されます。

■例 2

```
(gdb) enable
(gdb) info breakpoints
Num  Type          Disp   Enb   Address     What
1    breakpoint    keep   y     0x00c0001c  in main at main.c:7
2    breakpoint    keep   y     0x00c0001e  in main at main.c:10
3    breakpoint    keep   y     0x00c00028  in main at main.c:13
4    breakpoint    keep   y     0x00c00038  in sub at main.c:20
```

ブレーク番号を省略すると、すべてのブレークポイントが一度に有効（無効）に設定されます。

注 意

- ブレーク番号は、ブレーク設定時に個々のブレークポイントに対して 1 から順に割り当てられます。有効/無効にする時点で不明な場合は、上記例のとおり info breakpoints コマンドで確認してください。
- ブレークポイントは設定数に制限がありますので、使用しないブレークポイントは delete コマンドで削除してください。
- 設定されていないブレーク番号が指定された場合、"No breakpoint number N."を表示して設定は行いません。

ignore（回数指定付きブレークの無効化） [ICD Mini / SIM]

機 能

ブレークヒット数を指定し、その回数だけ特定のブレークを無効にします。

書 式

ignore ***BreakNo Count***

BreakNo: ブレーク番号（10 進数）
Count: 無効にするブレークヒット数（10 進数または 16 進数）

使用例

```
(gdb) info breakpoints
Num  Type          Disp   Enb   Address     What
1    breakpoint    keep   y     0x00c0003c  in sub at main.c:22
2    breakpoint    keep   y     0x00c00030  in main at main.c:14
(gdb) ignore 2 2
```

ブレーク番号 2 を 2 回無効にします。

```
(gdb) continue
Continuing.
Breakpoint 1, sub (k=1) at main.c:22
(gdb) continue
Continuing.
Breakpoint 1, sub (k=1) at main.c:22
(gdb) continue
Continuing.
Breakpoint 2, main () at main.c:14
```

上記 2 回の実行（continue）でターゲットプログラムはブレークポイント 2 を通過していますが、ブレークは発生しません。3 度目の実行では無効設定が解除されているため、ブレークが発生します。

注 意

- ブレーク番号は、ブレーク設定時に個々のブレークポイントに対して 1 から順に割り当てられます。削除する時点で不明な場合は、上記例のとおり info breakpoints コマンドで確認してください。
- ブレーク無効回数は指定ブレークのヒット数をカウントする値で、ターゲットプログラムの実行回数をカウントするものではありません。指定のブレークポイントを通過しない場合、無効回数は減りません。
- ignore コマンドでは、複数のブレークポイントをまとめて無効にすることはできません。
- 設定されていないブレーク番号が指定された場合、"No breakpoint number N."を表示して実行を中断します。

info breakpoints（ブレークポイントリストの表示） [ICD Mini / SIM]

機 能

設定されているブレークポイントの一覧を表示します。

書 式

info breakpoints

表 示

一覧は次のように表示されます。

```
(gdb) info breakpoints
Num  Type           Disp   Enb  Address     What
1    breakpoint     keep   y    0x00c00026  in main at main.c:11
     breakpoint already hit 1 time
2    hw breakpoint  del    n    0x00c00038  in sub at main.c:20
```

Num: ブレーク番号を表示します。
Type: ブレークポイントの種類を表示します。
`breakpoint` ソフトウェア PC ブレークポイント
`hw breakpoint` ハードウェア PC ブレークポイント
Disp: ブレークヒット後のブレークポイントの状態を表示します。
`keep` ブレークポイントは削除されません。
`del` ブレークポイントが削除されます。テンポラリブレークであることを示します。
Enb: ブレークポイントが有効か無効かを表示します。
`y` 有効
`n` 無効
Address: ブレークポイントに設定されたアドレスを 16 進数で表示します。
What: ブレークポイントが設定された位置を表示します。
ブレークポイントは、"`in` *関数名* `at` *ソースファイル名*`:`*行番号*"の形式で表示されます。

その他、各ブレークポイントごとに、現在までのヒット回数を"breakpoint already hit N times"の形で表示します。
ブレークポイントが 1 カ所も設定されていないときは、次のように表示されます。

```
(gdb) info breakpoints
No breakpoints or watchpoints.
```

commands（ブレーク後に実行するコマンドの設定） [ICD Mini / SIM]

機 能

指定したブレークポイントで停止したときに実行するコマンド（複数行）の設定及び、解除を行います。

書 式

commands [*BreakNo*]
コマンド入力後、プロンプトが">"に変わります。ここで、ブレーク時に設定するコマンド行を入力します。
コマンド行は、複数行可能で、最後に"end"を入力すると設定を終了します。
設定したコマンド行は、info breakpoint コマンドで確認可能です。
ブレーク番号省略時は、直前に設定したブレークポイントの番号が指定されます。

コマンド行を解除する方法：
プロンプトが">"になったら 1 行目に"end"を入力します。

使用例

```
(gdb) break boot.s:16
(gdb) commands 1
>x /4b 0x100
>break main
>continue
>x /4b sub
>end
(gdb) continue
Continuing.

Breakpoint 1, boot () at boot.s:16
0x100: 0xaa 0xaa 0xaa 0xaa
Breakpoint 2 at 0x632: file main.c, line 18

Breakpoint 2, main () at main.c:18
Current language: auto; currentry c

0x658 <sub>: 0x00 0x40 0x25 0x18

(gdb)
```

注 意

- 存在しない数値をブレーク番号に指定するとエラーになります。
- ブレーク番号省略時は、直前に設定したブレークポイントの番号が指定されます。
- コマンド行は最大 50 行まで動作保証されます。50 行を越えてもエラーにはなりません。51 行以上は AS IS で使用となります。
- コマンド実行中にエラーが発生した場合は、そこで停止します。
- テンポラリブレークポイント（tbreak、thbreak）でブレークした場合は、コマンド行は実行しません。
- commands コマンドのネストはできません。コマンド行に commands コマンドがあると、ブレーク時にその commands コマンドでのコマンド行が入力できなくなります。

8.5.9 シンボル情報表示コマンド

info locals（ローカルシンボル表示）
info var（グローバルシンボル表示） [ICD Mini / SIM]

機 能

シンボルの一覧を表示します。

info locals: 現在の関数内で定義されているローカル変数の一覧を表示します。
info var: グローバルおよびスタティック変数の一覧を表示します。

書 式

```
info locals
info var
```

使用例

■例 1

```
(gdb) info locals
i = 0
j = 2
```

現在の PC アドレスを含む関数内のローカルシンボルをすべて、その内容と共に表示します。

■例 2

```
(gdb) info var
All defined variables:

File main.c:
int i;

Non-debugging symbols:
0x00000000  __START_bss
0x00000004  __END_bss
0x00000004  __END_data
0x00000004  __START_data
```

定義されているすべてのグローバルおよびスタティック変数を、ソースファイルごとに一覧表示します。
"Non-debugging symbols:"には、セクションシンボル等、ソースファイル以外で定義されたグローバルシンボルが表示されます。

注 意

PC アドレスで示される現在位置が関数(スタックフレーム)外の場合(アセンブリソースのブートルーチン内など)、ローカルシンボルは表示されません。

```
(gdb) info locals
No frame selected.
```

print（シンボル値の変更） [ICD Mini / SIM]

機 能

シンボルの値を変更します。

書 式

print ***Symbol*** [=***Value***]

Symbol: 変数名
Value: 変更する値（10 進数、16 進数、またはシンボル）
省略時は現在のシンボル値を表示します。
条件: 0≦*Value*≦型の有効範囲

使用例

■例 1

```
(gdb) info local
j = 0
(gdb) print j
$1 = 0
```

変数名のみを指定すると、その値を表示します。$*N* は後からこの値を参照するための番号です。`print $1` で、ここで表示した内容を参照できます。

■例 2

```
(gdb) print j=5
$2 = 5
(gdb) info local
j = 5
```

値を指定すると、変数がその値に変更されます。

注 意

- 定義されていないシンボルを指定すると、エラーとなります。
- 変更する変数の型の範囲を超えた値を指定してもエラーとはなりません。変数のサイズに相当する下位側のビット数分のみが有効となり、それを超えた分は無視されます。たとえば、int 変数に 0x10000 を指定すると、0x0000 として処理されます。

8.5.10 ファイル読み込みコマンド

file（デバッグ情報の読み込み） [ICD Mini / SIM]

機 能

elf 形式のオブジェクトファイルからデバッグ情報のみを読み込みます。
オブジェクトコードは load コマンドを使用して読み込みます。

書 式

file ***Filename***

Filename: デバッグする elf 形式のオブジェクトファイル名（パスも指定可能）

使用例

(gdb) **file sample.elf**

カレントディレクトリの sample.elf からデバッグ情報を読み込みます。

注 意

- file コマンドはデバッグ情報のみを読み込み、オブジェクトコードは読み込みません。したがって、ターゲット MCU の ROM にプログラムが書き込まれている場合を除き、file コマンドの実行のみではデバッグを開始することはできません。
- file コマンドは、target コマンド、load コマンドより先に実行してください。基本的な実行順序は次のとおりです。

```
(gdb) file sample.elf       (本コマンド)
(gdb) target sim            (ターゲット MCU の接続)
(gdb) load                  (プログラムのロード)
(gdb) c17 rst               (リセット)
```

- file コマンドでは実行形式の elf オブジェクトファイル（リンカにより生成）以外はエラーとなり読み込めません。また、読み込んだファイルにデバッグ情報がない場合もエラーとなります。
- elf 形式のオブジェクトファイルは、ディレクトリ構造も含めたソースファイル情報を持っています。このため、ソースファイルがカレントディレクトリから見てオブジェクトファイル内の指定ディレクトリ上にないと、ソースファイルが読み込めません。基本的にはコンパイルからデバッグまでの操作を、同一のディレクトリで行ってください。
- file コマンド実行後は、ファイルの読み込みを終了するまで中断はできません。
- 非サポート（C17 用フラグの無い）の elf ファイルを指定するとエラーになります。

load（プログラムのロード） [ICD Mini / SIM]

機 能

ファイルからプログラム/データを読み込み、ターゲット MCU のメモリにロードします。

書 式

load ***Filename***

Filename: デバッグする elf または ROM データ (モトローラ形式)
ファイル名（パスも指定可能）
省略すると、先に file コマンドで指定したファイルが読み込まれます。通常は省略します。

使用例

```
(gdb) file sample.elf
(gdb) target sim
(gdb) load
```

カレントディレクトリの sample.elf（file コマンドで指定）からプログラム/データを読み込み、ターゲットメモリ（この例ではシミュレータモードのため、パソコンのメモリ）にロードします。

注 意

- load コマンドは、file コマンド、target コマンドより後に実行してください。基本的な実行順序は次のとおりです。

```
(gdb) file sample.elf      (デバッグ情報の読み込み)
(gdb) target sim           (ターゲット MCU の接続)
(gdb) load                 (本コマンド)
(gdb) c17 rst              (リセット)
```

- load コマンドはオブジェクトファイル内のコードおよびデータが存在する領域のみを読み込みます。それ以外の領域は load コマンド実行前の状態のまま残りますので注意してください。

8.5.11 トレースコマンド

c17 tm（トレースモード設定） [SIM]

機 能

以下の選択を行います。

トレース機能の ON/OFF
ON に設定すると、プログラムの実行に従ってトレース情報が採取されます。

トレース情報の表示項目
トレース情報の中で表示させる項目を選択できます。

トレース情報の出力先
採取したトレース情報の出力先として、コンソールまたはファイルを選択できます。
ファイル選択時は、ファイル名も指定します。

書 式

c17 tm on ***Mode*** [***Filename***] （トレースモード設定）
c17 tm off （トレースモード解除）

Mode: トレースモード（トレース情報表示内容）
0x00～0xff の範囲で指定します。ビットが 1 の情報を表示します。
ビット 0 トレース番号
ビット 1 クロック数
ビット 2 PC 値と命令コード
ビット 3 バス情報（アドレス、R/W およびアクセスサイズ、データ）
ビット 4 レジスタ値（R0～R7、SP）
ビット 5 PSR 値（IE、IL、CVZN）
ビット 6 逆アセンブル内容とソース
ビット 7 クロック数の積算表示指定（0 の場合は命令個別のクロック数表示）

Filename: トレース情報出力ファイル名
指定すると採取したトレース情報はファイルに出力され、コンソールには表示されません。
省略するとコンソールに表示され、ファイルには出力されません。

使用例

■例 1

```
(gdb) c17 tm on 0xff trace.log
```

全情報を表示させるトレースモードを設定し、情報を保存するファイル `trace.log` を指定します。この設定以降にプログラムを実行すると、実行した命令ごとにトレース情報をファイルに出力します。ファイル名を省略すると、コンソールに表示されます。

■例 2

```
(gdb) c17 tm off
```

トレースモードを解除します。これ以降は、プログラムを実行してもトレース情報は採取しません。

トレース情報

本コマンドによるトレースモードの設定後、ターゲットプログラムを実行すると、実行した命令ごとにコンソールに表示、またはファイルに出力します。
表示/出力されるトレース情報の内容は以下のとおりです。

各行の形式

num clk pc code bus_addr/type/data r0 r1 r2 r3 r4 r5 r6 r7 sp ie/il/cvzn src_mix

num: 実行命令 No.（10 進数）
CPU をリセットしてからの実行命令数
clk: 実行クロック数(10 進数)
CPU をリセットしてからの実行クロック数
pc: 実行アドレス
code: 命令コード
bus_addr: アクセスしたメモリアドレス（16 進数）
type: バスオペレーションタイプ
`r8`:バイトデータリード
`r16`: 16 ビットデータリード
`r32`: 32 ビットデータリード
`w8`: バイトデータライト
`w16`: 16 ビットデータライト
`w32`: 32 ビットデータライト
data: リード/ライトデータ
r0～r7: r0～r7 レジスタ値（16 進数）
sp: sp レジスタ値
ie: psr の IE ビット値
il: psr の IL ビット値
cvzn: psr の C、V、Z、N ビット値
src_mix: 実行命令の逆アセンブル内容とソースコード

表示例

各行の前半（トレース番号～レジスタ値）

```
num    clk   pc      code    bus addr/type/data       r0       r1       r2       r3       r4       r5       r6       r7
652    1445  0040dc  9900    ------ --- --------     000094   000000   000000   00ffff   000000   000000   000000   000000
653    1446  0040de  4000    ------ --- --------     000094   000000   000000   00ffff   000000   000000   000000   000000
654    1447  0040e0  4000    ------ --- --------     000094   000000   000000   00ffff   000000   000000   000000   000000
655    1449  0040e2  d900    000000 w16 00000000     000094   000000   000000   00ffff   000000   000000   000000   000000
656    1450  0040e4  2a12    ------ --- --------     000094   000000   000000   00ffff   000000   000000   000000   000000
657    1451  0040e6  2814    ------ --- --------     000000   000000   000000   00ffff   000000   000000   000000   000000
658    1452  0040e8  4000    ------ --- --------     000000   000000   000000   00ffff   000000   000000   000000   000000
659    1457  0040ea  1805    003ef4 w32 000040ec     000000   000000   000000   00ffff   000000   000000   000000   000000
660    1458  0040f6  a001    ------ --- --------     000000   000000   000000   00ffff   000000   000000   000000   000000
661    1459  0040f8  9000    ------ --- --------     000000   000000   000000   00ffff   000000   000000   000000   000000
662    1462  0040fa  0e0e    ------ --- --------     000000   000000   000000   00ffff   000000   000000   000000   000000
663    1466  004118  0120    003ef4 r32 000040ec     000000   000000   000000   00ffff   000000   000000   000000   000000
664    1467  0040ec  8201    ------ --- --------     000000   000000   000000   00ffff   000001   000000   000000   000000
665    1468  0040ee  9205    ------ --- --------     000000   000000   000000   00ffff   000001   000000   000000   000000
```

各行の後半（SP 値～ソースコード）

```
  sp ie/il/cvzn    src     mix
003ef8 0 0 0010    ld      %r2,0x0          (main.c)      00012         i = 0;
003ef8 0 0 0010    ext     0x0
003ef8 0 0 0010    ext     0x0
003ef8 0 0 0010    ld      [0x0],%r2
003ef8 0 0 0010    ld      %r4,%r2          (main.c)      00014         for( j = 0; j < 6;  ++j )
003ef8 0 0 0010    ld      %r0,%r4          (main.c)      00016         sub(j);
003ef8 0 0 0010    ext     0x0
003ef4 0 0 0010    call    0x5
003ef4 0 0 0010    and     %r0,0x1          (main.c)      00022         if(k & 0x1)
003ef4 0 0 0010    cmp     %r0,0x0
003ef4 0 0 0010    jreq    0xe
003ef8 0 0 0010    ret     (main.c)         00027 }
003ef8 0 0 0000    add     %r4,0x1          (main.c)      00014         for( j = 0; j < 6; ++j )
003ef8 0 0 1001    cmp     %r4,0x5
```

注 意

- 本コマンドは ICD Mini モードでは使用できません。
- トレースモード（トレース情報表示内容）を変更する場合は、トレースモードを一旦解除（`c17 tm off` を実行）してから再設定してください。

8.5.12 その他のコマンド

set output-radix（変数表示形式の変更） [ICD Mini/SIM]

機 能

print コマンドで変数を表示する形式を変更します。

変更可能な形式は、16 進数/10 進数/8 進数です。
ただし、変数が浮動小数点やポインタの場合は、表示形式は変更しません。
変更した形式は、デバッガ終了時記憶されず、次回 GDB 起動時にはデフォルト(10 進数)の表示形式で変数を表示します。

書 式

set output-radix ***Type***

Type: 表示形式
16=16 進数
10=10 進数（デフォルト）
8=8 進数

使用例

```
(gdb)print i
$1 = -21846
(gdb)set output-radix 16
(gdb)print i
$2 = 0xaaaa
(gdb)set output-radix 8
(gdb)print i
$3 = 0125252
```

注 意

- 2 進数を設定した場合（`set output-radix 2`）はデバッガの表示が正しく表示されません。

set logging（ログ出力の設定） [ICD Mini / SIM]

機 能

デバッガコマンドのログをファイルに保存します。

書 式

set ***logging on*** （ログ出力有効）
set ***logging off*** （ログ出力無効）

使用例

■例 1

```
(gdb) set logging on
```

デバッガコマンドログを出力します。ログはファイル gdb.txt に保存されます。

■例 2

```
(gdb) set logging off
```

ログ出力を無効にします。

source （コマンドファイルの実行） [ICD Mini / SIM]

機 能

コマンドファイルを読み込み、その中に記述されたデバッグコマンドを連続実行します。

書 式

source ***Filename***

Filename: コマンドファイル名

使用例

ファイル名 = src.cmd

```
# load symbol information
file c:/EPSON/gnu17v3/sample/tst/sample.elf
#decide debugger mode and its port
target sim
# load to memory
load c:/EPSON/gnu17v3/sample/tst/sample.elf
# reset
c17 rst
```

#から行の終わりまではコメントとみなされます。

```
(gdb) source src.cmd
(gdb)
(gdb) file c:/EPSON/gnu17v3/sample/tst/sample.elf
(gdb)
(gdb) target sim
boot () at boot.s:9
Connected to the simulator.
Current language:  auto; currently asm
(gdb)
(gdb) load c:/EPSON/gnu17v3/sample/tst/sample.elf
Loading section .text, size 0xbc lma 0xc00000
Start address 0xc00000
Transfer rate: 1504 bits in <1 sec.
(gdb)
(gdb) c17 rst
CPU resetting ..................... done
```

指定のコマンドファイルを読み込み、連続的に実行します。

注 意

- コマンドファイル内の記述に誤りがあると、そこでコマンドファイルの実行を中止します。この場合、エラーメッセージは表示されませんので、コマンドファイルは注意して作成してください。
- コマンドファイル内に source コマンドがあるというように source コマンドのネストが可能です。ネスト数に制限はありません。
- コマンドファイル等で if 文等の制御命令はサポートしません。

target（ターゲット MCU の接続） [ICD Mini / SIM]

機 能

ターゲット MCU との接続を確立し、コネクトモードを設定します。

ICD Mini モード: ICDmini と USB インタフェースで接続します。
シミュレータモード: デバッガをシミュレータモードに設定します。

書 式

target ***Type*** **[*SubType*]**

Type: コネクトモードを指定する以下のシンボル
icd ICDmini と USB インタフェースで接続（ICD Mini モード）
sim シミュレータを起動（SIM モード）
SubType: コネクトモードが icd の場合に指定する以下のシンボル
Icdmini2 ICDmini2 を使用するとき指定します。
icdmini3 ICDmini3 を使用するとき指定します。

使用例

■例 1

```
(gdb) target sim
Connected to the simulator.
```

シミュレータモードに設定します。

■例 2

```
(gdb) target icd icdmini3
ICD hardware version ..... 1.0
ICD software version ..... 1.0
Hardware break MAX .... ..... 1
```

ICDmini3 を使用し、ICD Mini モードに設定します。

注 意

target コマンドは load コマンドの前、file コマンドは target コマンドより前に実行してください。基本的な実行順序は次のとおりです。

```
(gdb) target sim            (本コマンド)
(gdb) c17 ttbr 0x20000      (TTBR の設定)
(gdb) load sample.elf       (プログラムのロード)
(gdb) c17 rst               (リセット)
```

detach（ターゲット MCU の切断） [ICD Mini / SIM]

機　能

ターゲット MCU との通信ポートを閉じ、現在のデバッグモードを終了します。

書　式

detach

使用例

```
(gdb) target icd icdmini3
        :
     デバッグ
        :
(gdb) detach
```

ICD Mini モードを終了します。

注　意

本コマンドはシミュレータモードと他のモードの切り換えや、ターゲットボードの操作のために ICDmini を OFF する場合などに使用します。デバッグを終了する場合に実行させる必要はありません。

pwd（カレントディレクトリの表示）
cd（カレントディレクトリの変更） [ICD Mini / SIM]

機 能

pwd: カレントディレクトリを表示します。
cd: カレントディレクトリを変更します。

書 式

pwd （カレントディレクトリの表示）
cd ***Directory*** （カレントディレクトリの変更）

Directory: ディレクトリを指定する文字列

使用例

```
(gdb) pwd
Working directory c:/EPSON/gnu17/sample/tst.
(gdb) cd c:/EPSON/gnu17/sample/ansilib
Working directory c:/EPSON/gnu17/sample/ansilib.
```

カレントディレクトリを確認後 c:¥EPSON¥gnu17¥sample¥ansilib に変更します。

c17 ttbr（TTBR 設定） [SIM]

機 能

TTBR にアドレスを設定します。
リセットコマンド（c17 rst）を実行したときに、TTBR の示すアドレス内の値（リセットベクタ）が PC に設定されます。

書 式

c17 ttbr ***Address***

Address: TTBR に設定するアドレス（10 進数、16 進数、またはシンボル）
条件: 0≦*Address*≦0xffff00（*Address* の下位 8 ビットは 0x00 である必要があります。）

使用例

(gdb) **c17 ttbr 0x8000**

TTBR に 0x8000 番地を設定します。

注 意

- 本コマンドは、シミュレータモード時のみ使用可能です。
- 本コマンドは、target コマンドの前に実行する必要があります。

c17 cpu（CPU タイプ設定） [SIM]

機 能

C17 コアシミュレータに CPU タイプを設定します。
C17 コアシミュレータ上でコプロセッサインタフェース命令の動作が切り替わります。

書 式

c17 cpu [*CpuType*]

CpuType: CPU タイプを設定する以下のシンボル

copro0	コプロセッサなし（デフォルト）
coprom	乗算コプロセッサ搭載
copro	COPRO 搭載
copro2	COPRO2 搭載

省略した場合は、現在の CPU タイプを表示します。

使用例

(gdb) **c17 cpu copro2**

CPU タイプを乗除算器 COPRO2 搭載に設定します。

注 意

- 本コマンドは、シミュレータモード時のみ使用可能です。
- 本コマンドは、target コマンドの前に実行する必要があります。
- デバッガは、c17 model コマンド経由で取得した機種の情報か、本コマンドにより、CPU タイプを設定します。c17 model コマンドと本コマンドの両方が実行された場合、より後に実行したコマンドの設定を優先します。

c17 chgclkmd（DCLK 切替えモード） [ICD Mini]

機 能

ブレーク時の DCLK（デバッグクロック）切替えモードを設定します。

切替有効の場合： ブレーク時に、DCLK が低速クロックの場合に高速クロックへ自動的に変更します。
プログラム再開する場合に元の DCLK に戻します。
切替無効の場合： ブレーク時、DCLK は変更しません。

書 式

c17 chgclkmd [*Mode*]

Mode： 0 DCLK 切替モード有効（デフォルト）
1 DCLK 切替モード無効
省略した場合は、現在のモードを表示します。
Mode=0 のとき："DCLK change mode ON"
Mode=1 のとき："DCLK change mode OFF"

使用例

<ターゲット CPU が S1C17702 の場合の例>

```
(gdb)continue
   プログラム内で OSC1 に設定
   プログラム内でブレーク発生          切替えモード有効なので HSCLK に切替える。
(gdb)finish                          OSC1 に戻して実行する。
   finish 終了                       HSCLK に切替える。
(gdb)c17 chgclkmd 1                  切替えモード無効にして、OSC1 に戻す。
(gdb)continue                        切替えモード無効なので HSCLK のまま実行。
   プログラム内でブレーク              切替えモード無効なのでクロック切替えしない。
(gdb)
```

注 意

- c17 chgclkmd コマンドは ICD Mini モード時のみ使用可能です。
- c17 chgclkmd コマンドは対応するターゲット CPU のみ使用可能です。
- Mode=0 のとき、以下の使い方をするとクロック切替えが正常に動作しなくなることがあります。

  1. ブレーク中のクロック制御レジスタ、クロックソースレジスタの書き替え
  2. ターゲットプログラム内でクロック切り替え中にブレーク
  3. ターゲットプログラム内でクロック切り替え部分をステップ実行

  以上のような使い方をする場合は *Mode*=1 にしてください。

c17 pwul（Flash セキュリティ・パスワードの解除） [ICD Mini]

機 能

Flash セキュリティ対応機種でパスワードが設定されているとき、パスワードの解除を行います。

書 式

`c17 pwul` ***Version Password***

Version: Flash セキュリティのバージョン（文字列）
例：
`M03`：Flash セキュリティのバージョン 3 を表す値
Password: パスワードの値。
英数字(`A-Z`、`a-z`、`0-9`)を指定します。
有効な文字数は、*Version* の値により変わります。

使用例

```
(gdb)c17 pwul M03 ABCD1234
Unlock flash security password was setted.
```

設定済のパスワード"ABCD1234"でパスワードの解除を行います。

注 意

- Flash セキュリティ未対応機種のとき、エラーになります。
- 未定義の *Version* を指定するとエラーになり、パスワード解除はできません。
- *Password* に無効な文字列（英数字以外や無効な文字数）を指定するとエラーになり、パスワードの解除はできません。

c17 help（ヘルプ） [ICD Mini / SIM]

機 能

コマンド説明を表示します。

書 式

c17 help [*Command*]
c17 help [*GroupNo.*]

Command: コマンド名
GroupNo.: コマンドグループ番号

使用例

■例 1

```
(gdb) c17 help
group  0: memory ...........  c17 fb,c17 fh,c17 fw,x /b,x /h,x /w,set {char},
                              set {short},set {int},c17 mvb,c17 mvh, ¥n¥
                              c17 mvw,c17 df
group  1: register .........  info reg,set $
group  2: execution ........  continue,until,step,stepi,next,nexti,finish
group  3: CPU reset ........  c17 rst, c17 rstt
group  4: interrupt ........  c17 int,c17 intclear
group  5: break ............  break,tbreak,hbreak,thbreak,delete,clear,enable,
                              disable,info breakpoints
group  6: symbol ...........  info locals,info var,print
group  7: file .............  file,load
group  8: trace ............  c17 tm
group  9: others ...........  source,target,detach,pwd,cd,set output-radix
                              c17 ttbr,c17 cpu,c17 chgclkmd,c17 help
                              c17 model_path,c17_model,quit
Please type ¥"c17 help 1¥" to show group 1 or type "c17 help c17 fb" to get usage of command
"c17 fb"
```

"パラメータを省略すると、コマンドグループの一覧を表示します。

■例 2

```
(gdb) c17 help 2
group 2:  execution¥n¥
          continue            Execute continously
          until               Execute continously with temporary break
          step                Single-step every line
          stepi               Single-step every mnemonic
          next                Single-step with skip every line
          nexti               Single-step with skip every mnemonic
          finish              Quit function
Please type "c17 help continue" to get usage of command "continu"
```

コマンドグループ番号を指定すると、そのグループに属するコマンドの一覧を表示します。

■例 3

```
(gdb) c17 help step
step: Single-step,every line                              [ICD/SIM]

usage:     step [Count]
Count:     Number of steps to execute (decimal or hexadecimal)
           One step is assumed if ommitted.
Conditions:1-0x7fffffff

example:
(gdb)step
(gdb)step 10
```

コマンドを指定すると、詳細説明を表示します。

■例 4

```
(gdb) c17 help c17 rst
c17 rst: Reset                                                  [ICD/SIM]

usage: c17 rst

example:
(gdb)c17 rst
The CPU is reset.
```

C17 系のコマンドの詳細説明を表示する場合は、"c17"の後に続けてコマンドを指定します。

注 意

- c17 help コマンドではなく gnu 標準の help コマンドを実行すると、gnu デバッガに設定されたコマンドクラスとコマンドのヘルプが表示され、その中には本デバッガがサポートしていないコマンドも含まれます。
  本マニュアルで説明していないコマンドは、動作が保証されませんので注意してください。
- 詳細説明（例 3、例 4）で表示された[ICD/SIM]は、コマンドが有効なモードを示しています。
  ICD: ICD Mini モードで有効（ICDmini 使用時）
  SIM: シミュレータモードで有効（PC のみでのデバッグ時）
  "[ICD]"のような表示は、そのコマンドが ICD Mini モード以外では使用できないことを意味します。

c17 model_path（機種別情報ファイルのディレクトリの設定） [ICD Mini / SIM]

機 能

機種別情報ファイルの格納されているディレクトリを設定します。
起動時オプションの--model_path と同じ機能です。

書 式

c17 model_path ***ModelPath***

ModelPath: 機種別情報ファイルのあるディレクトリを指定する文字列

使用例

```
(gdb) c17 model_path C:/EPSON/GNU17V3/mcu_model
(gdb) c17  model  17W23
(gdb) target icd icdmini3
```

機種別情報ファイルの格納されているディレクトリを、c:¥EPSON¥GNU17V3¥mcu_model に設定します。

注 意

- 本コマンドを実行しない場合、機種別情報ファイルの格納されているディレクトリは、デバッガ gdb.exe が存在するディレクトリのサブディレクトリ mcu_model に設定されています。通常、デバッガ gdb.exe は c:¥EPSON¥GNU17V3 に存在しますので、 使用例 の設定は、本コマンドを実行しない場合と同じです。
- 本コマンドによる設定は、c17 model コマンド実行時に参照されます。本コマンドを実行する場合、c17 model コマンドの前に実行してください。

c17 model（MCU モデル名の設定） [ICD Mini / SIM]

機 能

デバッグターゲットの MCU モデル名を設定します。
起動時オプションの`--model` と同じ機能です。

書 式

`c17 model` *ModelName* [*@Detail*]

ModelName: ターゲットとなる MCU モデルを指定する文字列
例：`17W23`
Detail : 以下の Detail オプションを指定する文字列

ModelName と *Detail* は空白ではなく@で区切ります。

Detail オプション	内容
NOVCCIN	ICDmini3 の POWER I/F の TARGET VCC IN 端子が未接続の場合に指定します。 ■ 省略時の動作 TARGET VCC IN 端子の電圧レベルで Target CPU と通信します。 TARGET VCC IN 端子にターゲット電源が接続されている必要があります。 ■ パラメータ なし
FLS	使用する FLS(Flash 消去・書換え)プログラムを指定します。 ■ 省略時の動作 標準の FLS プログラム（昇圧回路未使用）を使用します。 ■ パラメータ FLS プログラムファイル名(*.saf) ■ 指定例 FLS=fwr17W36_32bv11.saf ; 昇圧回路使用 FLS プログラムを指定します。
NOREAD	書き込みデータと、フラッシュメモリに書き込まれているデータの比較を禁止します。 ■ 省略時の動作 書き込みデータと、フラッシュメモリに書き込まれているデータを比較し、一致する場合は書き込みを省略します。 ■ パラメータ なし
NOWRITE	フラッシュメモリへの書き込みを禁止します。 ■ 省略時の動作 デバッガの load コマンド実行時にフラッシュメモリへデータを書き込みます。(書き込みデータと、フラッシュメモリに書き込まれているデータが一致する場合を除きます) ■ パラメータ なし
NOERASE	フラッシュメモリの消去を禁止します。 ■ 省略時の動作 フラッシュメモリにデータを書き込む前に消去を実行します。 ■ パラメータ なし
VPP	フラッシュメモリの消去・書き込み電圧を設定します。Flash プログラミング電源端子を持つターゲット MCU にのみ使用可能です。 ■ 省略時の動作] 機種の仕様に従い電圧を印加します。 ■ パラメータ 7.5[V]、7.0[V] 無効な値を指定すると、省略時と同じ動作になります。 ■ 指定例 VPP=7.5; 7.5V に設定します。

BREAKWAIT	強制ブレーク時の最大待ち時間[msec]を設定します。 ■ 省略時の動作 デフォルト値（3000msec）が設定されます。 ■ パラメータ 5[msec]以上、300000[msec]以下の値 有効範囲外の値を指定すると、デフォルト値が設定されます。 ■ 指定例 BREAKWAIT＝3000; 3000msec に設定します。
TIMEOUT	ICDmini3 と Target CPU の通信のタイムアウト[msec]を設定します。 ■ 省略時の動作 デフォルト値（10msec）が設定されます。 ■ パラメータ 5[msec]以上、300000[msec]以下の値 有効範囲外の値を指定すると、デフォルト値が設定されます。 ■ 指定例 TIMEOUT＝10; 10msec に設定します。

使用例

■例 1:ターゲットの MCU モデルが S1C17W23 のとき

```
(gdb) c17 model 17W23
(gdb) target icd icdmini3
```

デバッグ I/F 電圧をターゲットボードの電圧にする場合も同様の指定になります。
＊ICDmini の POWER I/F（TARGET VCC IN 端子）にターゲット電源が接続されている必要があります。

■例 2: FLS(フラッシュ書き込みルーチン)を指定するとき

```
(gdb) c17 model 17W23@FLS=FLS17W23.saf
(gdb) target icd icdmini3
```

■例 3:デバッグ I/F 電圧が 3.3V のとき

```
(gdb) c17 model 17W23@NOVCCIN
(gdb) target icd icdmini3
```

■例 4:デバッグ I/F 電圧が 3.3V で、FLS も指定するとき

```
(gdb) c17 model 17W23@FLS=FLS17W23.saf,NOVCCIN
(gdb) target icd icdmini3
```

Detail に複数の Detail オプションを指定する場合は、カンマで区切ります。

注 意

- 本コマンドは target コマンドの前に実行してください。

c17 flv（Flash プログラミング電源設定） [ICD Mini]

機 能

Flashプログラミング電源端子があるマイコンの消去、及び書き込み時に必要な電圧をICDminiから供給します。このコマンドは、loadコマンドの前に使用します。通常は、loadコマンド内において自動で必要な電圧制御を行いますので、指定は不要です。

書 式

c17 flv 7.5（設定）

使用例

```
(gdb) target icd icdmini3
(gdb) c17 flv 7.5
(gdb) load
(gdb) c17 flvs
```

Loadコマンド前にFlashプログラミング電源として7.5Vを供給開始し、loadコマンド後にFlashプログラミング電源の供給を停止します。

注 意

- 本コマンドを使用するには、Flash プログラミング電源端子があるマイコンの機種別情報ファイルが必要です。
- 本コマンドはシミュレータモードでは使用できません。
- フラッシュメモリの書き込み・消去後は、c17 flvs コマンドで書き込み・消去電圧の設定を解除してください。
- load コマンドが失敗した場合、自動的に電圧供給を解除します。（c17 flvs と同等な処理を行います。）

c17 flvs（Flash プログラミング電源設定解除） [ICD Mini]

機 能

c17 flvで設定したFlashプログラミング電源の供給を停止します。このコマンドは、loadコマンドの後に使用します。C17 flvを実行した場合は、本コマンドを実行してください。

書 式

c17 flvs（解除）

使用例

```
(gdb) target icd icdmini3
(gdb) c17 flv 7.5
(gdb) load
(gdb) c17 flvs
```

Loadコマンド前にFlashプログラミング電源として7.5Vを供給開始し、loadコマンド後にFlashプログラミング電源の供給を停止します。

注 意

- 本コマンドを使用するには、Flash プログラミング電源端子があるマイコンの機種別情報ファイルが必要です。
- 本コマンドはシミュレータモードでは使用できません。

c17 stdin（入出力関数を用いたデータ入力） [ICD Mini / SIM]

機 能

データをファイルまたは標準入力アプリケーション(stdinput.exe)から入力して、プログラムに渡すための設定を行います。入出力関数を用いることで、ユーザプログラムにデータを取り込むことが可能です。

書 式

c17 stdin 1 READ_FLASH READ_BUF［***Filename***］（設定）
c17 stdin 2 （解除）

***Filename*:** 入力ファイル名、省略すると、[コンソール]ビューからの入力となります。

使用例

■例1

(gdb) **c17 stdin 1 READ_FLASH READ_BUF input.txt**

ファイルからのデータ入力機能を設定します。
この設定の後でプログラムを連続実行させると、デバッガはlibg.aライブラリに記述されているラベルREAD_FLASHの位置で実行を中断します。そこでinput.txtファイルから1行分のデータを入力バッファ（READ_BUF）に取り込み、プログラムの実行を再開します。

■例2

(gdb) **c17 stdin 1 READ_FLASH READ_BUF**

標準入力アプリケーションを使用するデータ入力機能を設定します。
この設定の後でプログラムを連続実行させると、デバッガはlibg.aライブラリに記述されているラベルREAD_FLASHの位置で実行を中断し、標準入力アプリケーションを起動します。標準入力アプリケーションにデータを入力して[OK]ボタンを押すと、デバッガは入力したデータを入力バッファ（READ_BUF）に取り込み、プログラムの実行を再開します。

■例3

(gdb) **c17 stdin 2**

データ入力機能を解除します。ファイル入力の場合、指定ファイルは閉じられます。

注 意

- c17 stdin コマンドを使用する場合、ユーザプログラムは libg.a がリンクされている必要があります。
- c17 stdin コマンドでは、ハードウェアブレークポイントを 1 つ使用します。そのため、ハードウェアブレークポイントの最大数を超える場合は、エラーが発生します。
- 入力バッファ(READ_BUF)は 62 文字です。62 文字を越える入力は、破棄します。
- 標準入力アプリケーションは、英数字・記号のみ入力を許可します。
- c17 stdin 1 コマンドの連続実行は出来ません。必ず、c17 stdin 1/c17 stdin 2 をセットで使用してください。

c17 stdout（入出力関数を用いたデータ出力） [ICD Mini / SIM]

機 能

ユーザプログラムから、データをファイルまたは[Console]ビューに出力するための設定を行います。入出力関数を用いることで、ユーザプログラムからデータを出力することが可能です。

書 式

c17 stdout 1 WRITE_FLASH WRITE_BUF［***Filename***］（設定）
c17 stdout 2 （解除）

***Filename*:** 入力ファイル名、省略すると、[コンソール]ビューへの出力となります。

使用例

■例1

(gdb) **c17 stdout 1 WRITE_FLASH WRITE_BUF output.txt**

ファイルへのデータ出力機能を設定します。
この設定の後でプログラムを連続実行させると、デバッガはlibg.aライブラリに記述されているラベルWRITE_FLASHの位置で実行を中断します。ここで、指定のバッファ（WRITE_BUF）内のデータを指定のファイルに出力し、プログラムの実行を再開します。

■例2

(gdb) **c17 stdout 1 WRITE_FLASH WRITE_BUF**

[コンソール]ビューを使用するデータ出力機能を設定します。
この設定の後でプログラムを連続実行させると、デバッガはlibg.aライブラリに記述されているラベルWRITE_FLASHの位置で実行を中断します。ここで、[コンソール]ビューを開き、指定のバッファ（WRITE_BUF）内のデータをウィンドウに表示してプログラムの実行を再開します。

■例3

(gdb) **c17 stdout 2**

データ出力機能を解除します。ファイル出力の場合、指定ファイルは閉じられます。

注 意

- c17 stdout コマンドを使用する場合、ユーザプログラムは libg.a がリンクされている必要があります。
- c17 stdout コマンドでは、ハードウェアブレークポイントを 1 つ使用します。そのため、ハードウェアブレークポイントの最大数を超える場合は、エラーが発生します。
- 出力バッファ(READ_BUF)は 62 文字です。62 文字を越える出力は、破棄します。
- c17 stdout 1 コマンドの連続実行は出来ません。必ず、c17 stdout 1/c17 stdout 2 をセットで使用してください。

c17 lcdsim（LCD パネルシミュレータの設定・解除） [ICD Mini]

機 能

LCDパネルシミュレータ機能の設定、または解除を行います。

書 式

c17 lcdsim on （設定）
c17 lcdsim off（解除）

使用例

■例1

(gdb) **c17 lcdsim on**

LCDパネルシミュレータ機能を設定し、[ES-Sim]ウィンドウを起動します。
この設定の後でプログラムを連続実行させると、デバッガはliblcdsim.aライブラリに記述されているラベルLCD_SIMの位置で実行を中断します。

■例2

(gdb) **c17 lcdsim off**

LCDパネルシミュレータ機能を解除し、[ES-Sim]ウィンドウを終了します。

注 意

- c17 lcdsim コマンドを使用する場合、ユーザプログラムは liblcdsim.a がリンクされている必要があります。
- c17 lcdsim コマンドでは、ハードウェアブレークポイントを 1 つ使用します。そのため、ハードウェアブレークポイントの最大数を超える場合は、エラーが発生します。
- LCD パネルシミュレータに対応していない機種の場合、以下のエラーが発生します。

quit（デバッガ終了） [ICD Mini / SIM]

機　能

デバッガを終了します。
デバッガが使用したポートやファイルが開いているときは、すべて閉じます。

書　式

quit
q　　（省略形）

使用例

(gdb) **q**

8.6 ステータス/エラーメッセージ

8.6.1 ステータスメッセージ

ターゲットプログラムがブレークすると、コマンド入力待ちになる直前に以下のブレーク要因を示すメッセージを表示します。

表 8.6.1.1 ステータスメッセージ

メッセージ	内容
Breakpoint #, *function* at *file:line*	設定したブレークポイントでブレーク
Illegal instruction.	シミュレータモードで不当命令の実行によりブレーク
Illegal delayed instruction.	シミュレータモードで不当な遅延命令の実行によりブレーク
Break by key break.	[中断]ボタンによる強制ブレーク(シミュレータモード)
Break by key break. Program received signal SIGINT, Interrupt.	[中断]ボタンによる強制ブレーク(ICD Miniモード)

8.6.2 エラーメッセージ

表 8.6.2.1 エラーメッセージ(アルファベット順)

メッセージ	内容
Address is 24bit over.	指定アドレスは、24bitを超えています。S1C17のアドレスは、最大24bit(0xFFFFFF)です。
Address(0x%lx) is ext or delayed instruction	指定アドレスは、`ext`命令または遅延命令のため設定できません。
C17 command error, command is not supported in present mode.	指定したコマンドは,現在のモード(ICD MINI or SIMモードまたはどちらでもない)では対応していません。
C17 command error, invalid command.	コマンドに誤りがあります。
C17 command error, invalid parameter.	指定したコマンドの引数に誤りがあります。
C17 command error, number of parameter.	指定したコマンドの引数の数に誤りがあります。
C17 command error, start address > end address.	開始アドレスが終了アドレスを超えて指定されています。
Cannot set hard pc break.	指定したアドレスにハードブレークは設定できません。
Cannot set hard pc break any more.	ハードウェアPCブレークポイントの設定数が制限を超えました。
Cannot set soft pc break.	指定したアドレスにソフトブレークは設定できません。
Cannot set soft pc break any more.	ソフトウェアPCブレークポイントの設定数が制限を超えました(Max. 200)。
Cannot write file	ファイルへの書き込みができません。
command result error!	未定義のコマンド実行結果がエラーとなりました。
icdmini3 dll open failure.	ICD mini Ver3 との接続に失敗しました。
C17 command error, command is not supported in present target CPU.	選択している機種は、LCDパネルシミュレータに対応していません。

8.7 実行時間計測

ユーザプログラムの実行開始からブレークまでのプログラム実行時間を計測します。

8.7.1 表示方法

[Expressions]ビューから、以下の 4 つのシンボルを登録します。

- $icdLastLapTime … ラップタイム (時、分、秒、us)
- $icdLastLapUs … ラップタイム (us)
- $icdTotalLapTime … 積算ラップタイム (時、分、秒、us)
- $icdTotalLapUs … 積算ラップタイム (us)

8.7.2 制限事項

- シミュレータモードには対応していません。
- ブレーク⇔連続実行(再開)の切り替え時に誤差が生じるため、数命令の短い実行時間時間計測ではなく、出来るだけ広い範囲の計測に使用してください。
- LCD パネルシミュレータ機能には対応していません。

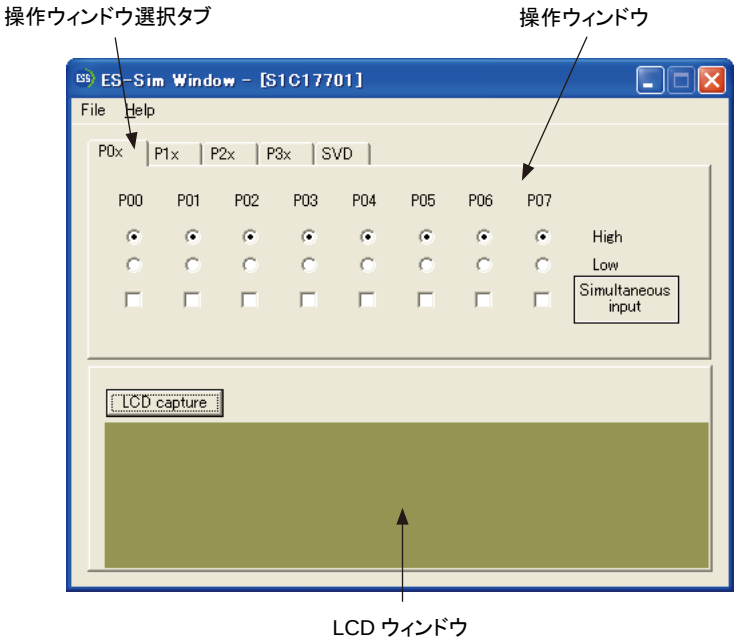
8.8 周辺回路シミュレータ（ES-Sim17）

組み込みシステムシミュレータ（**ES-Sim17**）は、PC 上で S1C17 チップのハードウェアをシミュレートする機能を提供します。デバッガ **gdb** のシミュレータモードに連動して動作しますので、より実際的なアプリケーションシステムのデバッグが PC のみで行えます。

ES-Sim17 の機能は以下のとおりです。

1. 汎用ポートの出力状態を表示するとともに入力状態をシミュレート
2. SVD 動作を評価するため、電源電圧値を設定可能
3. ターゲット機種の内蔵 LCD ドライバによる表示をシミュレート
   ※ポート数、SVD 有り/無しは機種によって決定されます。

操作と表示は、すべて次の[ES-Sim]ウィンドウで行います。



[ES-Sim]ウィンドウ（S1C17701 の例）

OSC1 クロック動作はリアルタイムに実行可能です。OSC3 クロック動作については、"simulator_readme. txt"を参照してください。

注: PC 上でのシミュレーションのため、制限事項があります。"8.7.8 制限事項"および"simulator_readme.txt"を参照してください。

8.8.1 入出力ファイル

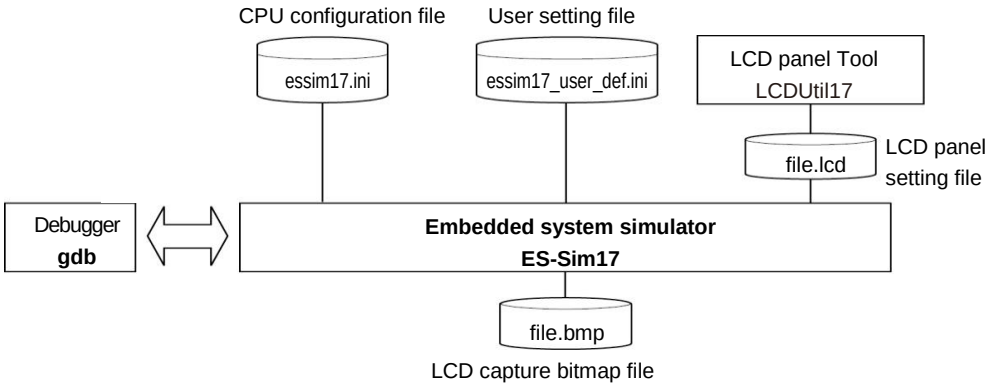


図 8.7.1.1 入出力ファイル

●入力ファイル

CPU 構成ファイル

ファイル形式: テキストファイル
ファイル名: `essim17.ini`（固定）
内容: **ES-Sim17** でシミュレートするターゲット機種のハードウェア構成を記述したファイルです。このファイルは、ターゲット MCU の機種別情報ファイルに含まれている必要があります。含まれていない場合は、最新の機種別情報ファイル（gnu17_mcu_model_xxx.zip）をセイコーエプソンの Web サイトから入手するか、弊社営業窓口にお問い合わせください。

注: ES-Sim17 が正常に動作しなくなりますので、このファイルは変更しないでください。

ユーザ設定ファイル

ファイル形式: テキストファイル
ファイル名: `essim17_user_def.ini`（固定）
内容: **ES-Sim17** で使用したターゲット機種の設定、ユーザの設定を保持するファイルです。このファイルは、ターゲット MCU の機種別情報ファイルに含まれている必要があります。含まれていない場合は、最新の機種別情報ファイル（gnu17_mcu_model_xxx.zip）をセイコーエプソンの Web サイトから入手するか、弊社営業窓口にお問い合わせください。

LCD パネル設定ファイル

ファイル形式: バイナリファイル
ファイル名: `<`*ファイル名*`>.lcd`
内容: LcdUtil17 で作成した、ES-Sim17 用 LCD パネル設定ファイルです。
ES-Sim17 でドットマトリクス LCD、セグメント LCD のシミュレートを行えます。

●出力ファイル

LCD キャプチャビットマップファイル

ファイル形式: ビットマップファイル
ファイル名: `<`*ファイル名*`> .bmp`
内容: **ES-Sim17** で作成する、シミュレートした LCD 画面のビットマップファイルです。

8.8.2 起動と終了

●ES-Sim17 の起動

ES-Sim17 は以下の 2 つの条件が成立した場合にデバッガによって起動します。

1. c17 model コマンドで指定された MCU モデルに essim17.ini ファイルが存在する。
2. `target sim`（シミュレータモード設定）コマンドが実行される。

ES-Sim17 が起動すると、[ES-Sim]ウィンドウが表示されます。

●ES-Sim17 の終了

ES-Sim17 が終了するのは、次の 2 つの場合です。

1. デバッガの `detach` コマンド実行時
2. デバッガの終了時

●[ES-Sim]ウィンドウのオープン/クローズ

[ES-Sim]ウィンドウは[閉じる]ボタンで、閉じることができます（この操作で **ES-Sim17** は終了しません）。
再度開くには、**¥essim17¥EssWnd.exe** を実行します。ただし、**ES-Sim17** が終了していないことが前提です。**ES-Sim17** が終了している場合は、再度 `target sim` コマンドを実行する必要があります。
[ES-Sim]ウィンドウは 1 つのみ開くことができます。すでに開いている場合に再度開く操作を行うと、ウィンドウが最前面に移動しますが、新たなウィンドウは開きません。

8.8.3 メニュー

[File]メニュー

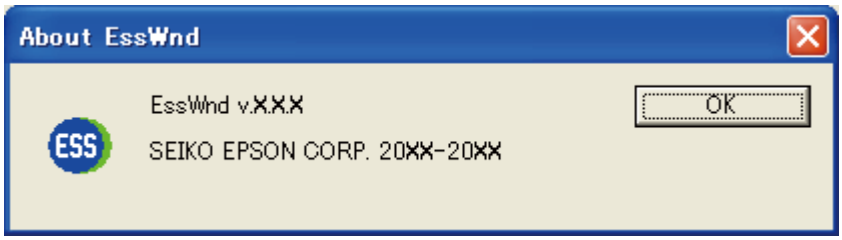
[Load lcd file]

LCDUtil17 で作成した LCD ファイル(.lcd)を開きます。
LCDUtil17、LCD ファイルについては"10.8 LCDUtil17（LCD パネルカスタマイズツール）"を参照ください。

[Help]メニュー

[About EssWnd]

ES-Sim Window のバージョン情報を表示します。

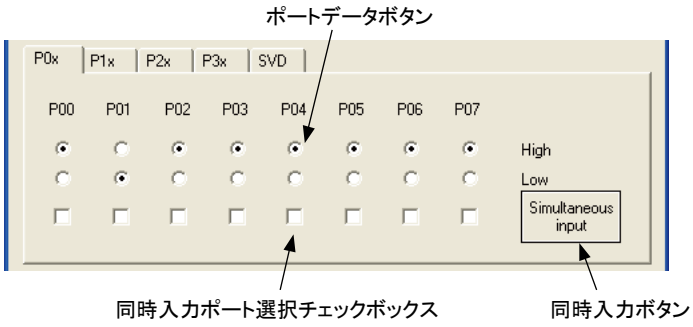


8.8.4 入出力ポートシミュレーション

汎用入力に設定したポートの入力状態を[ES-Sim]ウィンドウで制御することができます。また、汎用出力に設定したポートについては、出力状態が確認できます。

●ポートデータ操作ウィンドウ

操作ウィンドウ選択タブで、操作/表示するポート系列（P0x、P1x、P2x、P3x）を選択します。



ポートデータ操作ウィンドウ（P0x ポート）

ES-Sim17 は、デバッガを介して PC 上のエミュレーションメモリから入出力ポートの機能選択および入出力方向の情報を取得し、ポートデータ操作ウィンドウ上のポートの構成を決定します。

汎用入力に設定されたポートは、ポートデータボタン、同時入力ポート選択チェックボックスが有 効になり、入力レベルの設定が可能となります。ウィンドウ上の操作によって入力レベルが変化すると、**ES-Sim17** はデバッガを介してエミュレーションメモリの入力データレジスタを更新します。

汎用出力に設定したポートのポートデータボタンと同時入力ポート選択チェックボックスは、操作が無効なグレー表示となります。ただし、ポートデータボタンは現在の出力状態を示します。汎用出力に設定したポートの出力データレジスタが変更されると、その状態がポートデータボタンに反映されます。

内蔵周辺回路用入出力に設定されたポートのポートデータボタン、同時入力ポート選択チェックボックスは表示されません。

ターゲット機種に存在しないポートのポートデータボタン、同時入力ポート選択チェックボックスは表示されません。

●ポート入力状態の設定

ポートデータボタンで High または Low を選択します。これが現在のポート入力レベルになります。

●複数キーの同時入力

複数のキーを同時に押す操作をシミュレートするには、同時入力ポート選択チェックボックスで同時入力を行うポートをすべて選択します。その状態で同時入力ボタンをクリックすると、該当ポートの入力レベル（ポートデータボタンの設定）が反転します。

この操作は、現在表示している以外のタブのポート系列にも有効です。同時入力ポート選択ボタンで選択されているポートは、そのタブのページが表示されているか否かにかかわらず、いずれかのページの同時入力ボタンで入力レベルが反転します。

同時入力ポート選択チェックボックスで複数のポートを選択している場合でも、ポートデータボタンで個々のポートの制御は行えます。

●ポートの出力状態

デバッガによるプログラムの実行でポート出力が変化した場合は、その状態が即時ポートデータボタンの表示に反映されます。
なお、汎用出力に設定したポートのポートデータボタンをマウスで操作することはできません。

●P0 ポートキー入力リセット

P0 ポートの同時入力リセットに対応している機種の場合、ソフトウェアで指定したポートの同時入力によってリセットが行えます。この機能を評価する場合は、上記の複数キーの同時入力と同じ方法を使用してください。

●ポート入力割り込み

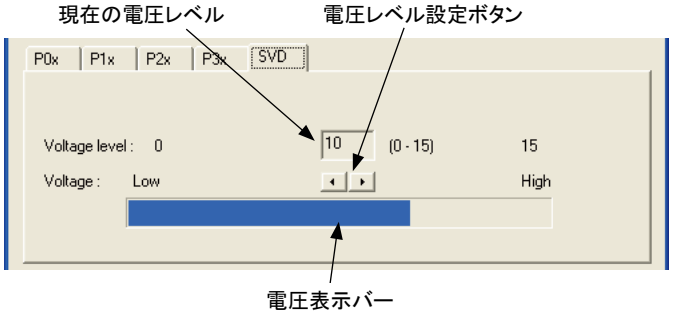
ポートデータ操作ウィンドウ上の入力状態の操作により、ポート入力割り込みを発生させることができます。

8.8.5 SVD シミュレーション

SVD 動作を確認するため、電源電圧のレベルを[ES-Sim]ウィンドウで制御することができます。

●SVD 操作ウィンドウ

操作ウィンドウ選択タブで、SVD を選択します。



SVD 操作ウィンドウ

SVD 操作ウィンドウは、電圧レベル 15（最高レベル）に初期設定されます。

●電圧レベルの設定

電圧レベル設定ボタンで電圧レベルを 0（低）～15（高）の 16 段階*に設定できます。

* この電圧レベルは、ターゲット機種の SVD 比較電圧の有効設定数に相当します。機種により設定可能な電圧レベルが変わる場合があります。

ボタンの操作により、現在の電圧レベルと電圧表示バーが変化します。同時に、エミュレーションメモリ内の SVD 制御レジスタに設定されている比較電圧とここで設定した電圧レベルが比較され、結果が SVD 検出結果レジスタに書き込まれます。

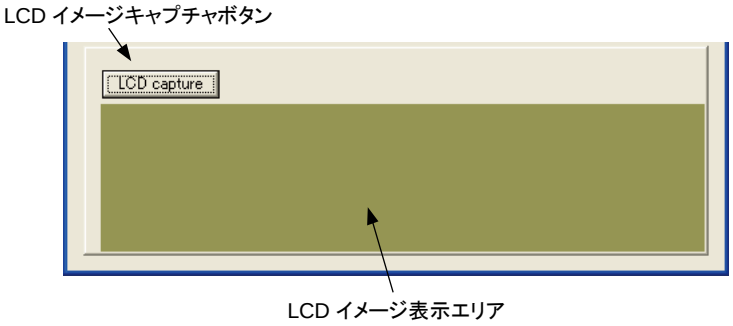
●SVD 割り込み

SVD 割り込みに対応している機種の場合、このウィンドウで SVD 比較電圧より低い電圧レベルを設定することで割り込みを発生可能です。

8.8.6 LCD ドライバシミュレーション

LCD ドライバの制御と表示メモリの操作に従った LCD パネルの表示をシミュレートできます。

●LCD ウィンドウ



LCD ウィンドウ

ドットマトリクスタイプの LCD ドライバの表示をシミュレートします。
ES-Sim17 は 32Hz の周期で表示メモリの内容を取得し、このウィンドウを再描画しています。
駆動デューティの設定や表示の制御（表示 On/Off、コントラスト、表示エリアの選択など）も反映されます。

●LCD 画面の保存

現在、LCD ウィンドウに表示されている画面をビットマップデータとしてファイル（.bmp）に保存することができます。
取り込みたい画面になったところで[LCD capture]ボタンをクリックします。ファイル保存ダイアログボックスが表示されますので、保存先の選択とファイル名の入力を行い、ファイルに保存します。
画面データの取り込みは[LCD capture]ボタンをクリックした時点で行われ、保存が完了するまで画面の更新は停止します。
LCD ウィンドウに LCD パネル全体が表示されていない場合でも、パネル全体の画像が保存されます。
生成されるファイルは Windows 標準のビットマップファイル（.bmp）です。

●制限事項

- 実際の LCD パネルとはドットのサイズやコントラスト、背景色が異なります。
- 実際の LCD パネルとは LCD 表示の更新タイミングが異なります。

8.8.7 ES-Sim17 のエラーメッセージ

表 8.7.7.1 エラーメッセージ（**gdb** の[Console]ウィンドウに表示）

メッセージ	内容
ES-Sim error 01 : Loading the dll file was failed.	**ES-Sim17**のdllファイルが既定の場所にないか、あるいは正常にロードできません。
ES-Sim error 02 : Opening the CPU construction file was failed.	CPU構成ファイルが指定場所にないか、あるいは正常に開けません。
ES-Sim error 03 : Generating the CPU components was failed.	CPUモジュール定義に誤りがあるか、必要なdllがないためCPUモジュールの生成に失敗しました。
ES-Sim error 04 : Connecting the CPU components was failed.	CPUモジュール定義の接続先に誤りがあるため、CPUモジュールの接続に失敗しました。
ES-Sim error 05 : Opening the "user.ini" was failed.	ユーザ設定ファイルが指定場所にないか、あるいは正常に開けません。
ES-Sim error 06 : Setting of the "user.ini" is invalid.	ユーザ設定ファイルに記述されている設定値が不正です。

表 8.7.7.2 エラーメッセージ（ダイアログボックスに表示）

メッセージ	内容
"*path*¥*file*"の保存に失敗しました。	キャプチャ画像の保存に失敗しました。
Failed to open "path¥file". Please confirm the file is fitting with the CPU type, or the file is existing.	LCDファイルのオープンに失敗しました。

8.8.8 制限事項

- 外部デバイスは、モノクロドットマトリクス LCD パネル、およびモノクロセグメント LCD パネルをサポートしています。
- 実際の LCD パネルとはドットのサイズやコントラスト、パネルの色が異なります。
- 本シミュレータはインストラクションレベルでシミュレーションを行っています。インストラクションサイクルよりも短いサイクルのシミュレーションを行うことはできません。
- インストラクションサイクルを基に動作クロックをシミュレートしていますので、実際のハードウェアとタイミングが同じではありません。
- 以下の機能はシミュレートできません。

  1. タイマクロックの外部出力、発振クロックの外部出力
  2. UART、I^2C、SPI 等のデータ送受信機能
  3. ノイズ、チャタリング除去フィルタ機能

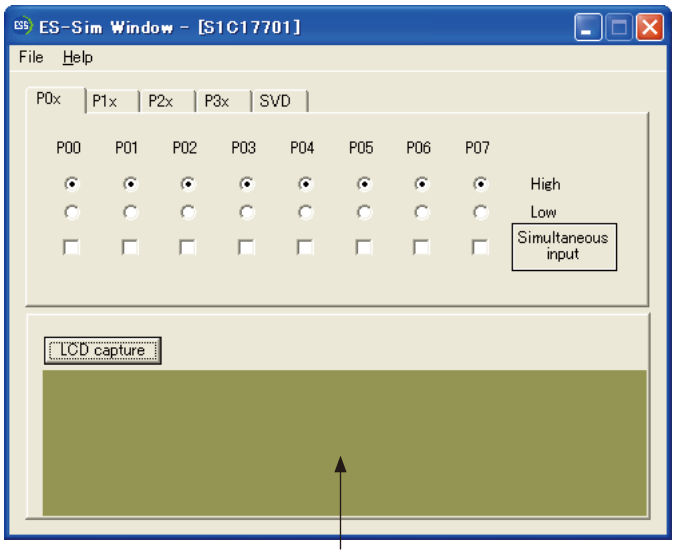
- PC 上で複数のシミュレータを同時に起動して、シミュレーションを行うことはできません。
- 発振回路の周波数設定値を大きくするとシミュレーションの速度が遅くなります。
- 対応していない回路の I/O 制御レジスタは、すべてリード/ライト可能レジスタとなります。また、リセット時に初期化はされません。
- 発振回路、SVD 等の回路は安定動作までに時間が必要ですが、シミュレータでは動作開始直後から安定した動作を行います。

※最新バージョンのシミュレータの制限事項、および機種に依存する制限事項については"simulator_readme.txt"を参照してください。

8.9 LCD パネルシミュレータ

LCD パネルシミュレータは、PC 上で LCD パネルの表示をシミュレートする機能を提供します。この機能は、プログラムを実機上で動作させ、LCD パネルの表示部分のみ PC 上でシミュレートします。そのため、LCD パネルが搭載されていない実機に対し、PC 上で LCD パネル表示の確認が行えます。

表示は、次の[ES-Sim]ウィンドウの中の LCD ウィンドウで行います。



[ES-Sim]ウィンドウ

LCD ウィンドウの操作に関しましては“8.7.6 LCD ドライバシミュレーション”を参照してください。

注: PC 上でのシミュレーションのため、制限事項があります。"8.8.4 制限事項"を参照してください。

8.9.1 入力ファイル

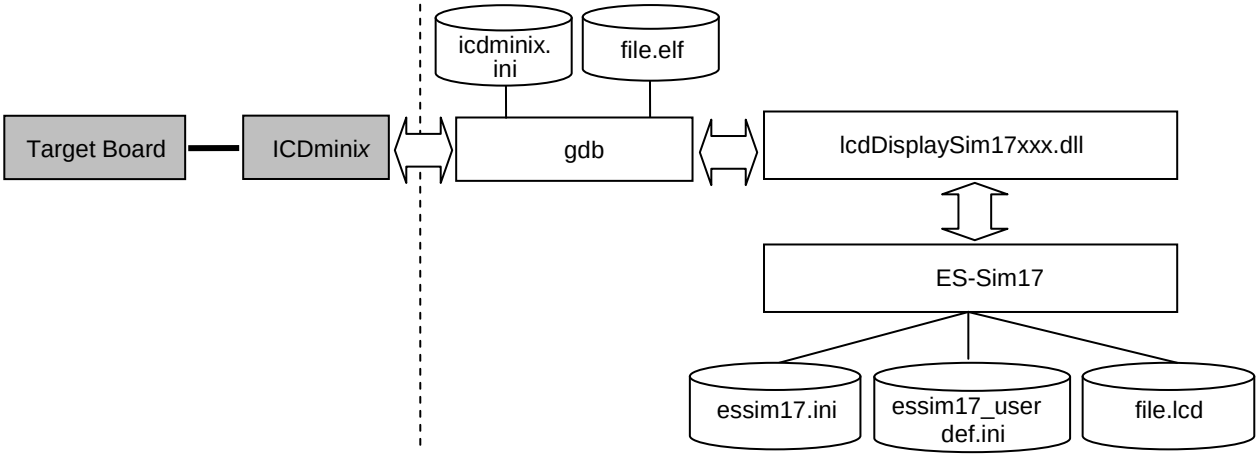


図 8.9.1.1 入出力ファイル

CPU 構成ファイル

ファイル形式: テキストファイル
ファイル名: `essim17.ini`
内容: LCD パネルシミュレータでシミュレートするターゲット機種のハードウェア構成を記述したファイルです。機種情報に含まれています。

注: LCD パネルシミュレータが正常に動作しなくなりますので、このファイルは変更しないでください。

ユーザ設定ファイル

ファイル形式: テキストファイル
ファイル名: `essim17_user_def.ini`
内容: LCD パネルシミュレータでシミュレートするターゲット機種のユーザが設定可能な値を記述するファイルです。機種情報に含まれています。

GDB コマンドファイル

ファイル形式: テキストファイル
ファイル名: `gdbminix.ini`
内容: ICDmini*x* 用 GDB コマンドファイルです。デバッガの起動と共に LCD パネルシミュレータを起動する場合は以下のコマンドを追記してください。

`c17 lcdsim on`	LCD パネルシミュレータを有効にします。

LCD パネル設定ファイル

ファイル形式: バイナリファイル
ファイル名: `<`*ファイル名*`>.lcd`
内容: LcdUtil17 で作成した、LCD パネル設定ファイルです。
ドットマトリクス LCD、セグメント LCD のシミュレートを行えます。

8.9.2 起動と終了

●LCD パネルシミュレータライブラリの起動

LCD パネルシミュレータは以下の条件が成立した場合に起動します。

1. c17 model コマンドで指定された MCU モデルに `lcdDisplaySim17xxx.DLL` ファイルが存在する。
2. LCD ファイルが存在し、ファイルパスが `essim17_user_def.ini` で設定されている。
3. c17 lcdsim on コマンドが実行される。

LCD パネルシミュレータが起動すると、[ES-Sim]ウィンドウが表示されます。

●LCD パネルシミュレータの終了

LCD パネルシミュレータが終了するのは、次の 2 つの場合です。

1. c17 lcdsim off コマンド実行時
2. デバッガの終了時

●LCD パネルの表示

[ES-Sim]ウィンドウの[File]>[Load lcd file] から、LCDUtil17 で作成した LCD ファイル(.lcd)を開きます。

LCDUtil17、LCD ファイルについては"10.8 LCDUtil17（LCD パネルカスタマイズツール）"を参照ください。

●[ES-Sim]ウィンドウのオープン/クローズ

[ES-Sim]ウィンドウは[閉じる]ボタンで、閉じることができます（この操作で LCD パネルシミュレータは終了しません）。
再度開くには、**¥essim17¥EssWnd.exe** を実行します。ただし、LCD パネルシミュレータが終了していないことが前提です。LCD パネルシミュレータが終了している場合は、再度 `c17 lcdsim on` コマンドを実行する必要があります。
[ES-Sim]ウィンドウは 1 つのみ開くことができます。すでに開いている場合に再度開く操作を行うと、ウィンドウが最前面に移動しますが、新たなウィンドウは開きません。

8.9.3 プログラム変更手順

LCD パネルシミュレータを動作させるには、プログラムの変更が必要となります。LCD パネルシミュレータライブラリをインクルードして、LCD ウィンドウの表示を更新したいポイントで LCD パネルシミュレータ表示更新関数を呼び出してください。以下にその手順を記述します。

●LCD パネルシミュレータライブラリのインクルード

LCD パネルシミュレータライブラリ関数を使用するプログラムファイルで、ヘッダーファイル（lcdsim.h）を#include 命令でインクルードしてください。

例： #include "lcdsim.h"

●LCD パネルシミュレータ表示更新関数の挿入

LCD パネルシミュレータの LCD ウィンドウの表示が更新されるのは LCD パネルシミュレータ表示更新関数（lcdsimUpdate）が実行された時です。プログラムの中で LCD の表示を更新したい箇所に LCD パネルシミュレータ表示更新関数を挿入してください。

```
例：  LCD24DSP.DSPC = 0x2   // LCD 全点灯
      lcdsimUpdate();       // LCD ウィンドウの表示更新を行う
      LCD24DSP.DSPC = 0x0   // LCD 全消灯
      lcdsimUpdate();       // LCD ウィンドウの表示更新を行う
      LCD24DSP.DSPC = 0x1   // LCD 通常表示
      lcdsimUpdate();       // LCD ウィンドウの表示更新を行う
```

●リンカオプションの設定

LCD パネルシミュレータライブラリをリンクしてプログラムをビルドするには、プロジェクトプロパティのリンカオプションを設定する必要があります。

[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings]＞[Cross GCC Linker]＞[Libraries] に LCD パネルシミュレータライブラリを指定します。“lcdsim”を追加してください。

プロジェクトを新規作成する時にターゲット機種の選択で LCD パネルシミュレータライブラリをサポートしている機種を選んだ場合は、自動的にリンカオプションの設定が行われます。

注: 実際に LCD パネルを搭載して動作させる場合は LCD パネルシミュレータライブラリのインクルード命令、LCD パネルシミュレータ表示更新関数はプログラム中から削除してください。残したままだと不要な処理が発生します。

8.9.4 制限事項

- target sim によるシミュレータモードは、サポートしていません。
- モノクロドットマトリクス LCD パネル、およびモノクロセグメント LCD パネルのみサポートしています。
- 実際の LCD パネルとはドットのサイズやコントラスト、パネルの色が異なります。
- 実際の LCD パネルとは LCD 表示の更新タイミングが異なります。
- 本機能を有効にするとハードウェアブレイクポイントを 1 本占有します。
- 周辺デバイスの状態（GPIO や CLG 等）は参照していません。

8.10 プロファイラ・カバレッジ

プロファイラ・カバレッジ機能は、プログラム中で性能を落とす原因としてボトルネックとなる関数の検出を主な目的とします。プロファイラ・カバレッジ機能は、本パッケージに同梱されている c17debug.exe により、実機とシュミレータを使用し計測を行います。c17debug.exe は、デバッグ機能の 1 つですが、デバッガ gdb とは別の外部ツールです。

8.10.1 入力出力ファイル

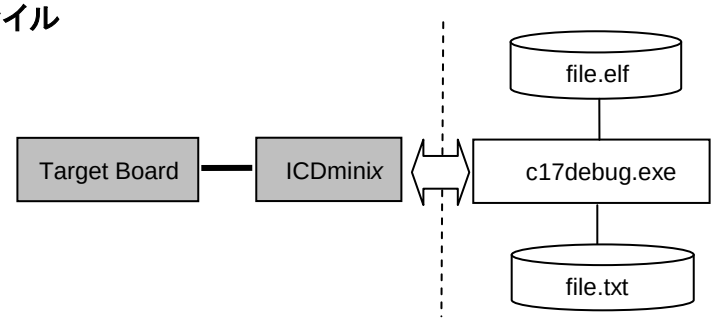


図 8.10.1.1　入出力ファイル

●入力ファイル

オブジェクトファイル

ファイル形式:　elf 形式のバイナリファイル
ファイル名:　<*ファイル名*>.elf
内容:　リンカ **ld** で生成した elf 形式のアブソリュートオブジェクトファイルです。シンボル等のデバッグ情報が含まれている必要があります。

●出力ファイル

ログファイル

ファイル形式:　テキストファイル
ファイル名:　<*ファイル名*>.txt
内容:　[Console]ビューに出力されている内容をログとして保存します。

8.10.2 起動と終了

プロファイラ・カバレッジ機能は、c17debug.exe より提供します。c17debug.exe は外部ツールとして起動可能です。

●c17debug.exe の起動

デバッガ gdb を起動している場合は、終了してください。以下の手順により、起動します。

1. [Run]メニューから[External Tools]＞[External Tools Configurations]を選択。
2. [New launch configurations]をクリックし、”New_configuration”を生成。
3. “New_configuration”の以下を変更。

タブ	設定項目	設計内容
-	Name:	任意のファイル名を入力。
[Main]タブ	Location:	[Browse File System]から以下のフォルダ直下にある c17debug.exe を選択。 フォルダ名：C:¥EPSON¥GNU17V3
	Working Directory:	[Browse File System]から作業フォルダである対象プロジェクトフォルダを選択。
	Arguments:	ログを出力するための任意のファイル名を入力。
[Common]タブ	Save as:	“Shared file”にチェックし、[Browse]から対象プロジェクトを選択。

4. [Apply]をクリックし、設定を反映。
5. [Run]をクリックし、c17debug.exe を起動。

c17debug.exe は、[Console]ビューを介し起動します。c17debug.exe は、[Console]ビュー上にプロンプト(“$”)が出力し、ユーザからの入力待機状態となります。

●c17debug.exe の終了

[Console]ビューの[Terminate]ボタンより、終了します。

8.10.3 事前準備

プロファイラ・カバレッジ機能を実行する事前準備として、以下の手順でプログラムのロードを行います。

●手順

1. PC に、ICDmini3 とターゲットボードを接続。

   ※1 c17debug.exe がサポートするのは ICDmini3 のみです。

2. c17debug.exe が起動した[Console]ビューに以下のコマンドを入力し実行。

   ```
   $ model 17xxx                        … 対象機種を指定
   $ icd                                … ICDmini3とターゲットMCUを接続
   $ load Debug/project_name.elf        … 対象オブジェクトファイル(.elf)と検出に使用するサポート関数
                                          をターゲット MCU にロード
   ```

   ※2 icd コマンド実行後、応答が無い場合は、ICDmini3 とターゲット MCU の接続に失敗しています。
   ※3 load コマンドでは、対象オブジェクトファイルがあるパスを指定してください。
   ※4 対象オブジェクトファイル(.elf)には、シンボル情報を含めてください。

以下の応答が得られれば、準備完了です。

●応答例

```
$ load Debug/sample_gcc6.elf
    VAddr    LAddr    FileSize MemSize
00: 0x000000 0x000000 0x000000 0x000050
01: 0x008000 0x008000 0x000280 0x000280
* Control flash:
  Successfully erased.
  Successfully writed.
* Support routines: 0x000050 - 0x0000e9
  0x000090: resume,  0x00008c: step  ,  0x0000b8: go1   ,  0x0000b4: go2
  0x0000bc: go0   ,  0x000074: read  ,  0x00005e: write
```

メモリ配置

オブジェクトファイルのロード結果

検出に使用する
サポート関数のロード結果

8.10.4 カバレッジ機能

カバレッジ(コードカバレッジ)は、コード網羅率を指します。ユーザプログラムの内、実行されたコード/回数を把握することが出来る機能です。入力されたシンボル名またはアドレスから、対象関数終端の命令まで実行し、関数内の区間ごとの実行回数を示します。

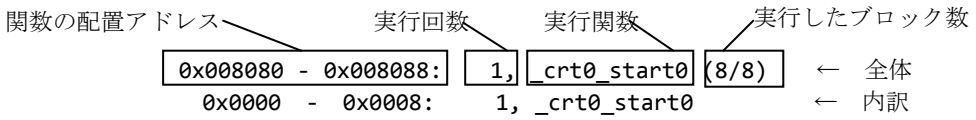
事前準備を行った[Console]ビューに、以下のコマンドを入力し実行します。

```
$ coverage symbol                          … 実行するシンボル名またはアドレスを指定
```

●実行例

```
$ coverage _crt0_start0
target paused
register    PC: 0x008116                                    … 実行終了後のレジスタ情報
          PSR: 0x00
           R0: 0x000000
      cycle  : 2                                            … 実行開始から関数終了までのサイクル数
      status : 9 (9: break, 10: timeout, 11: uncertain PC, 12/13: wrong PC)  … ターゲット MCU の状態
0x008080 - 0x008088:    1, _crt0_start0 (8/8)               … 計測結果
  0x0000  -  0x0008:     1, _crt0_start0
```

●計測結果の読み方



8.10.5 プロファイラ機能

プロファイラは､性能分析を指します｡本機能は､統計的プロファイラ(サンプリング型プロファイラ)を採用しています。サンプリングから、各関数の実行時間を把握することが出来る機能です。入力されたシンボル名またはアドレスから 10 秒間実行し、50ms 間隔で PC(プログラムカウンタ)をサンプリングした結果を示します。

事前準備を行った[Console]ビューに、以下のコマンドを入力し実行します。

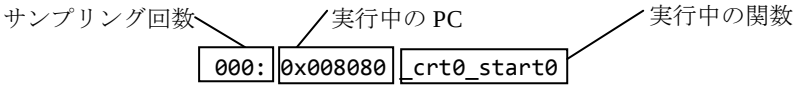
```
$ clear                                  … 計測途中にブレークしないようにブレークポイントをクリア
$ profile symbol                         … 実行するシンボル名またはアドレスを指定
```

※1 サンプリング中の 10 秒間は何も表示しません。
※2 10 秒経過すると、ターゲット MCU をブレークさせ、サンプリング結果を表示します。

●実行例

```
$ clear
$ profile _crt0_start0
000: 0x008080 _crt0_start0               … サンプリング中の現在位置
001: 0x008270 _exit
002: 0x008270 _exit
003: 0x008270 _exit
  :
197: 0x008270 _exit
198: 0x008270 _exit
199: 0x008270 _exit
target paused
register   PC: 0x008272                  … 実行終了後のレジスタ情報
         PSR: 0x0a
          R0: 0x000000
```

●計測結果の読み方



上記実行例からは、サンプリング回数 001～199 は、_exit 関数にいることが分かります。従って、ほとんどの時間を_exit 関数中に費やしたことが分かります。

8.10.6 制限事項

- ユーザプログラムは、RAM に対し 200byte 以上の空き容量を用意してください。本機能は計測のためにサブルーチン関数を RAM にロードするため、空き容量が必要です。
- 割り込み処理を含む関数計測は、サポートしていません。
- 対象オブジェクトファイル(.elf)には、シンボル情報を含めてください。
- 使用可能な ICDmini は、ICDmini3 のみです。

9 提出用データの作成

弊社工場にてターゲット CPU の内蔵 ROM 又は内蔵 FLASH メモリにユーザプログラムを書き込むサービスをご利用の場合、PA ファイル(提出用データ)を作成し、弊社へ提出していただく必要があります。
PA ファイルを作成するには、以下のファイルが必要です。

- FDC ファイル(ファンクションオプションドキュメント)
  (ファンクションオプションを選択する必要がある CPU のみ)
- PSA ファイル(ROM データ)

本章では、PA ファイルの作成方法について、説明します。

9.1　提出用データ作成ツールの概要

PA ファイル(提出用データ)の作成において、以下の 2 種類のツールを使用します。

1. ファンクションオプションジェネレーター（winfog17.exe）
   本ツールは、ファンクションオプションを選択する必要がある CPU で使用します。
   winfog17 は IC のマスクパターンを生成するための FDC ファイル(ファンクションオプションドキュメント)を作成するツールです。ウィンドウに表示された選択項目をチェックボックスで選択するだけでファンクションオプションを設定することができます。
   ファンクションオプションを設定する必要がない CPU では、FDC ファイルは不要です。
2. データチェッカ（winmdc17.exe）
   winmdc17 は開発が完了した PSA ファイル、FDC ファイルのデータをチェックし、弊社へ提出するための PA ファイルを作成するツールです。

9.2 提出用データの作成手順

図 9.2.1 に PA ファイル(提出用データ)作成のフローを示します。

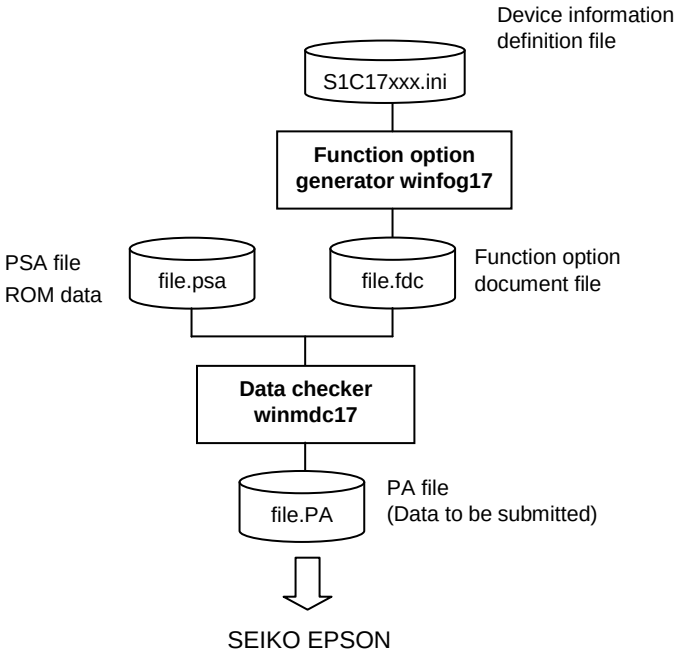


図 9.2.1 ファイル作成フロー

9.2.1 winfog17 による FDC ファイル(ファンクションオプションドキュメント)の作成

本作業は、ファンクションオプションを選択する必要がある CPU で必要となります。
FDC ファイル(ファンクションオプションドキュメント)は、ターゲット CPU 固有のファンクションオプションを選択したファイルです。選択可能なファンクションオプションについては、ターゲット CPU のテクニカルマニュアルを参照してください。
以下の手順で、winfog17.exe を使用してファンクションオプションを選択し、FDC ファイルを生成します。

(1) winfog17 を起動
ユーザーフォルダ¥EPSON¥GNU17V3¥dev¥Bin にある winfog17.exe をダブルクリックします。

前回の実行時に機種情報定義ファイル（S1C17xxx.ini）を読み込んでいる場合は、winfog17 起動時に同じファイルを自動的に読み込みます。

(2) S1C17xxx.ini をロード
winfog17.exe 上の[Device INI Select]ボタンをクリック、もしくは[Tool]メニューから[Device INI Select]を選択します。

ダイアログが表示され、テキストボックスにパスを含むファイル名を入力、もしくは[Ref]ボタンをクリックしてターゲット CPU の S1C17xxx.ini をロードします。各機種の S1C17xxx.ini は、以下のフォルダを参照してください。
ユーザーフォルダ¥EPSON¥GNU17V3¥mcu_model
ターゲット CPU に対応した S1C17xxx.ini が存在しない場合は、営業窓口にお問い合わせください。

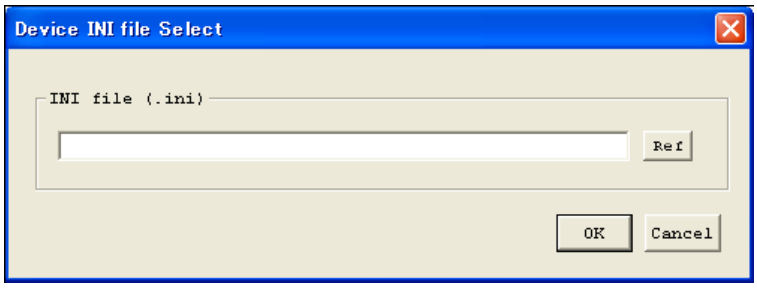


図 9.2.1.1　機種情報定義ファイルのロード画面

ファンクションオプションを選択する必要がない CPU については、[INI file does not include FOG information]とダイアログが表示されます。

(3)　ファンクションオプションを選択
オプションリスト領域のチェックボックスをクリックして必要なオプションを選択します。オプションリスト領域で選択項目を変更すると、選択されたファンクションオプションがファンクションオプションドキュメント領域に表示されます。

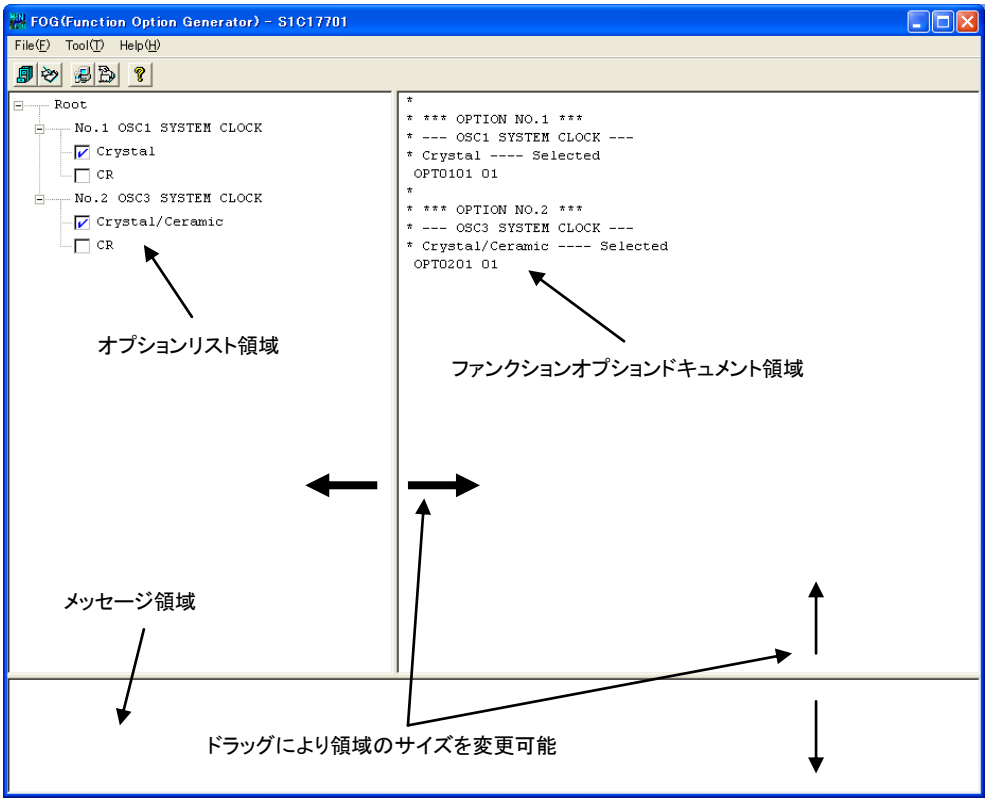


図 9.2.1.2 ファンクションオプション表示画面

(4)　FDC ファイルを生成
Winfog17 上の[Generate]ボタンをクリック、もしくは[Tool]メニューから[Generate]を選択します。

ファイル生成が正常に終了した場合は、メッセージ領域に"Making file(s) is completed"が表示されます。初期設定では、FDC ファイルは以下のフォルダに生成されます。

ユーザーフォルダ¥EPSON¥GNU17V¥dev

その他 winfog17.exe の機能

上記以外の winfog17.exe の機能を説明します。下記の機能は必要に合わせてご使用ください。

[File]メニュー

Open
FDC ファイルを開きます。既存のファイルを修正する場合に使用します。[Open]ボタンも同機能です。

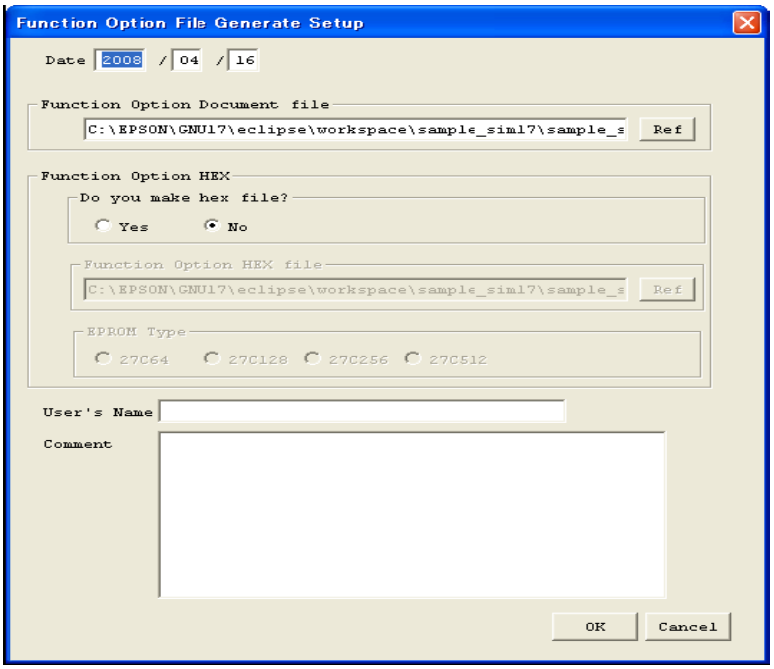
 [Open]ボタン

End
winfog17 を終了します。

[Tool]メニュー

Setup
作成日や出力ファイル名、FDC ファイルに含めるコメントなどを設定します。[Setup]ボタンも同機能です。

 [Setup]ボタン
次のダイアログボックスが表示されます。



Date
　現在の日付が表示されます。必要に応じて変更してください。

Function Option Document file
　作成する FDC ファイル名を指定できます。初期設定で表示される名前を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。

Function Option HEX
　S1C17 Family では本機能は使用しません。"No"を選択してください。

User's Name
　お客さまの会社名を入力します。最大 40 文字まで入力することができ、それを超えた場合は無視されます。英文字、数字、記号およびスペースが入力可能です。ここに入力した内容は、FDC ファイルの USER'S NAME フィールドに記録されます。

Comment
コメントを入力します。1 行に入力可能な文字数は 50 文字まで、最大 10 行まで入力することができます。英文字、数字、記号およびスペースが入力可能です。また、[Enter]キーで改行できます。ここに入力した内容は、FDC ファイルの COMMENT フィールドに記録されます。

上記の必要な項目を入力後、[OK]をクリックすると設定内容が保存され、ダイアログボックスが閉じます。[Cancel]をクリックした場合、現在の設定は変更されずにダイアログボックスが閉じます。

注: •ファイル名の指定には以下の制限があります。
1. **パスを含めたファイル名指定の文字数は最大 2048 文字です。**
2. **ファイル名（拡張子を除く）は最大 15 文字、拡張子は最大 3 文字です。**
3. **ファイル名の先頭にハイフン(-)は使用できません。また、ディレクトリ名（フォルダ名）、ファイル名、拡張子に、以下の記号の使用を禁止します。**
   / : , ; * ? " < > |

•User's Name と Comment に以下の記号は使用できません。
$ ¥ | `

9.2.2 PSA ファイル(ROM データ)の作成

PSA ファイル(ROM データ)は、elf 形式オブジェクトファイルと同じファイル名で、拡張子".psa"のモトローラ S2 形式ファイルです。以下の手順で、PSA ファイルを作成します。
なお、生成した PSA ファイルを使用して、必ず実機上で動作確認を行ってください。

(1) ターゲット CPU 機種の選択確認
IDE 上で、対象プロジェクトの Target CPU が目的の機種であることを確認します。目的の機種が選択されていない場合は、目的の機種を選択してください。また、目的の機種がリストに存在しない場合は、機種別情報ファイル（gnu17_mcu_model_xxx.zip）をセイコーエプソンの Web サイトから入手するか、弊社営業窓口にお問い合わせください。

(2) PSA ファイルを生成
対象プロジェクトをビルドします。対象プロジェクトフォルダ下に、PSA ファイルが生成されます。

IDE 上のプロジェクトのビルドでは、elf 形式オブジェクトファイルを生成すると、PSA ファイルの生成と、PA ファイルの生成も実施します。

9.2.3 winmdc17 による PA ファイル(提出用データ)の作成

PA ファイルの構成は、以下の 2 種類あります。

- FDC ファイル(ファンクションオプションドキュメント)が存在しない場合
  PSA ファイル(ROM データ)から PA ファイルを生成します。
- FDC ファイルが存在する場合
  PSA ファイルと FDC ファイルから、1 つの PA ファイルを生成します。

どちらの構成であっても、IDE 上でプロジェクトをビルドすると、ターゲット CPU の設定に応じて、winmdc17 が PA ファイルを生成します。

弊社工場にてターゲット CPU の内蔵 ROM 又は内蔵 FLASH メモリにユーザプログラムを書き込むサービスをご利用の場合、生成した PA ファイルを弊社営業窓口へ提出してください。

9.2.4　PA ファイル(提出用データ)の分離方法

winmdc17 で生成(パック)した PA ファイル(提出用データ)をファンクションオプションドキュメントと ROM データに分離(アンパック)することが可能です。

以下の手順で、winmdc17.exe を使用し、PA ファイルを UFD ファイル(ファンクションオプションドキュメント)と USA ファイル(ROM データ)に分離します。

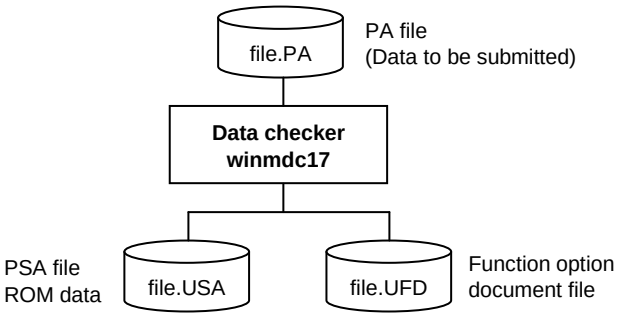


図 9.2.4.1　PA ファイルのアンパックフロー

(1)　winmdc17.exe を起動
ユーザーフォルダ¥EPSON¥GNU17V3¥dev¥Bin にある winmdc17.exe をダブルクリックします。

前回の実行時に機種情報定義ファイル（S1C17xxx.ini）を読み込んでいる場合は、winmdc17 起動時に同じファイルを自動的に読み込みます。

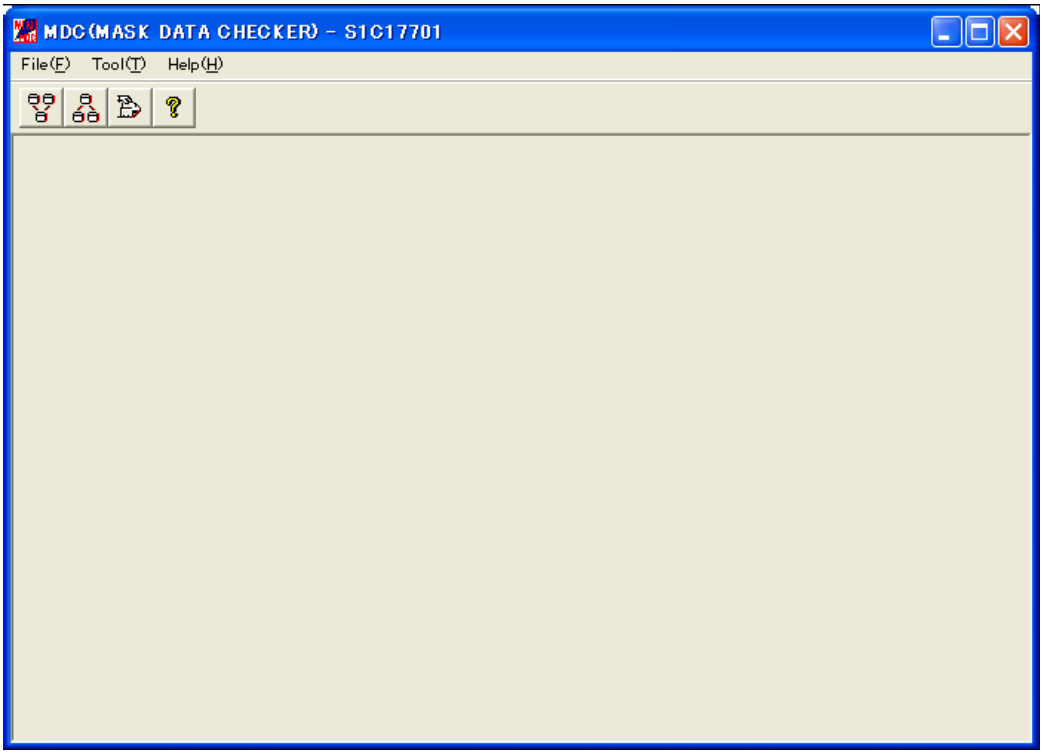


図 9.2.4.2　初期画面

(2) S1C17xxx.ini をロード

winmdc17.exe 上の[Device INI Select]ボタンをクリック、もしくは[Tool]メニューから[Device INI Select]を選択します。ダイアログが表示され、テキストボックスにパスを含むファイル名を入力、もしくは[Ref]ボタンをクリックしてターゲット CPU の S1C17xxx.ini をロードします。各機種の S1C17xxx.ini は、以下のフォルダを参照してください。

ユーザーフォルダ¥EPSON¥GNU17V3¥mcu_model

ターゲット CUP に対応した S1C17xxx.ini が存在しない場合は、営業窓口にお問い合わせください。

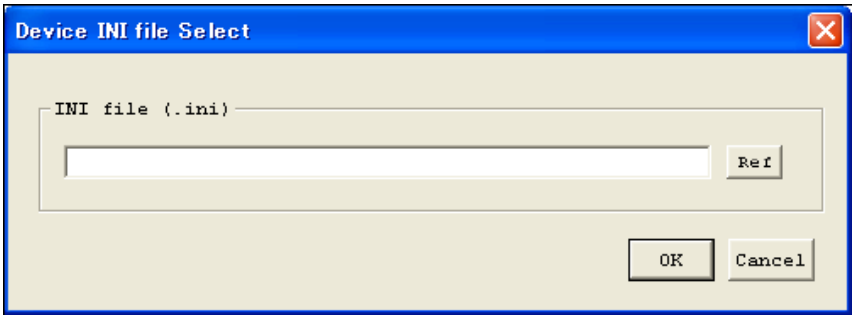


図 9.2.4.3　機種情報定義ファイルのロード画面

(3) 入力ファイルの選択

winmdc17.exe 上の[Unpack]ボタンをクリック、もしくは[Tool]メニューから[Unpack]を選択します。

アンパックするファイルを選択します。[Packed Input Files]の[Ref]ボタンをクリックし、アンパックする PA ファイルを選択してください。

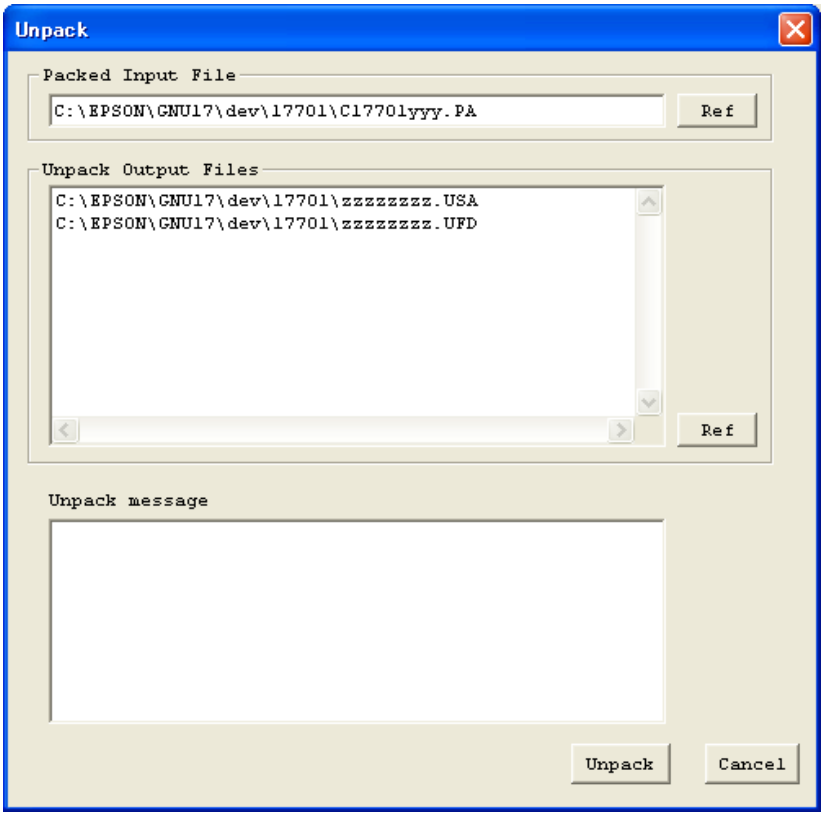


図 9.2.4.4　入力ファイル選択画面

[Unpack Output Files]から、出力するファイル名を指定できます。ファイル名をリストされているファイルに対し、[Ref]ボタンをクリックし、出力するフォルダの選択、拡張子付きでファイル名の入力を行ってください。拡張子は変更できません。

(4) PA ファイルを分離
入力ファイル選択画面の[Unpack]ボタンをクリックし、アンパックを実行します。アンパックが正常に終了した場合は、[Unpack message]領域に"Unpack completed !"が表示されます。
初期設定では、UFD ファイル、USA ファイルは以下のフォルダに生成されます。
ユーザーフォルダ¥EPSON¥GNU17V3¥dev
[Cancel]ボタンをクリックしてダイアログボックスを閉じます。

9.3 提出用データ作成ツールのエラーメッセージ

9.3.1 winfog17 のエラーメッセージ

winfog17 のエラーメッセージの一覧を示します。表示の"Dialog"はダイアログボックスに表示されるメッセージを、"Message"は[FOG]ウィンドウのメッセージ領域に表示されるメッセージを示します。

表 9.3.1.1 エラーメッセージ一覧

メッセージ	説 明	表示
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。	Dialog
Illegal character	入力禁止文字が入力された。	Dialog
Please input file name	ファイル名が未入力。	Dialog
Can't open File : xxxx	ファイル(xxxx)がオープンできない。 異常時メッセージ(デバッグ中にファイルを削除したときなどに出力される)。	Dialog
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。	Dialog
INI file does not include FOG information	指定した機種情報定義ファイル(.ini)にファンクションオプション情報が含まれていない。	Dialog
Function Option document file is not found	指定したファンクションオプションドキュメントファイルが存在しない。	Dialog
Function Option document file does not match INI file	指定したファンクションオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。	Dialog
A lot of parameter	コマンドラインの引数が多すぎる。	Dialog
Making file(s) is completed [xxxx is no data exist]	ファイル作成完了。ただし、作成したファイル(xxxx)にはデータが含まれていない。	Message
Can't open File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)がオープンできない。	Message
Can't write File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)に書き込みができない。	Message

表 9.3.1.2 ワーニングメッセージ

メッセージ	説 明	表示
Are you file update? xxxx is already exist	上書き確認メッセージ (指定したファイルは既に存在する。)	Dialog

9.3.2 winmdc17 のエラーメッセージ

winmdc17 のエラーメッセージの一覧を示します。表示の"Dialog"はダイアログボックスに表示されるメッセージを、"Message"は[Pack]または[Unpack]ダイアログボックスのメッセージ領域に表示されるメッセージを示します。

表 9.3.2.1 I/O エラーメッセージ一覧

メッセージ	説 明	表示
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。	Dialog
Illegal character	入力禁止文字が入力された。	Dialog
Please input file name	ファイル名が未入力。	Dialog
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。	Dialog
INI file does not include MDC information	指定した機種情報定義ファイル(.ini)にMDC情報が含まれていない。	Dialog
Can't open file : xxxx	ファイル(xxxx)がオープンできない。	Dialog
Can't write file: xxxx	ファイル(xxxx)に書き込みができない。	Dialog

表 9.3.2.2 ROM データエラーメッセージ一覧

メッセージ	説 明	表示
Hex data error: Not S record.	データが"S"で始まっていない。	Message
Hex data error: Data is not sequential.	データが昇順に並んでいない。	Message
Hex data error: Illegal data.	不当なキャラクタがある。	Message
Hex data error: Too many data in one line.	1行中のデータ数が多すぎる。	Message
Hex data error: Check sum error.	チェックサムが合わない。	Message
Hex data error: ROM capacity over.	データ容量が大きい。(データサイズ>ROMサイズ)	Message
Hex data error: Not enough the ROM data.	データ容量が少ない。(データサイズ<ROMサイズ)	Message
Hex data error: Illegal start mark.	スタートマークが不当である。	Message
Hex data error: Illegal end mark.	エンドマークが不当である。	Message
Hex data error: Illegal comment.	データの最初の機種名表示が不当である。	Message

表 9.3.2.3 ファンクションオプションデータエラーメッセージ一覧

メッセージ	説 明	表示
Option data error : Illegal model name.	機種名が不当である。	Message
Option data error : Illegal version.	バージョンが不当である。	Message
Option data error : Illegal option number.	オプションNo.が不当である。	Message
Option data error : Illegal select number.	選択肢No.が不当である。	Message
Option data error : Mask data is not enough.	ROMデータが充分でない。	Message
Option data error : Illegal start mark.	スタートマークが不当である。	Message
Option data error : Illegal end mark.	エンドマークが不当である。	Message

9.4 提出用データ作成ツールの出力例

winfog17.exe、winmdc17.exe から生成される FDC ファイル(ファンクションオプションドキュメント)、PA ファイル（提出用データ）のファイルフォーマットを以下に示します。

注: データの構成および内容は機種により異なります。

●FDC ファイル例

```
* S1C17xxx_xxKB FUNCTION OPTION DOCUMENT Vx.xx        ←バージョン
*
* FILE NAME zzzzzzzz.FDC                              ←ファイル名（[Setup]で指定）
* USER'S NAME SEIKO EPSON CORPORATION                 ←ユーザ名（[Setup]で指定）
* INPUT DATE yyyy/mm/dd                               ←作成年月日（[Setup]で指定）
* COMMENT SAMPLE DATA                                 ←コメント（[Setup]で指定）
*
* *** OPTION NO.1 ***                                 ←オプション番号
* --- OSC1 SYSTEM CLOCK ---                           ←オプション名
* Crystal(32.768KHz) ---- Selected                    ←選択した仕様
OPT0101 01                                           ←マスクデータ
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K04 With Resistor ---- Selected
* K05 With Resistor ---- Selected
* K06 With Resistor ---- Selected
* K07 With Resistor ---- Selected
OPT0301 01
OPT0302 01
OPT0303 01
OPT0304 01
OPT0305 01
OPT0306 01
OPT0307 01
OPT0308 01
*
* *** OPTION NO.4 ***
* --- OUTPUT PORT OUTPUT SPECIFICATION ---
* R00 Complementary ---- Selected
* R01 Complementary ---- Selected
* R02 Complementary ---- Selected
* R03 Complementary ---- Selected
OPT0401 01
OPT0402 01
OPT0403 01
OPT0404 01
*
:
*
```

```
* *** OPTION NO.8 ***
* --- SOUND GENERATOR POLARITY ---
* NEGATIVE ---- Selected
OPT0801 01
*EOF                                                    ←エンドマーク
```

●PA ファイル例

```
*
* S1C17xxx_xxKB xPCS xBITW xBITR MASK DATA VER x.xx  ←バージョン
*
¥ROM1                                                  ←ROM データスタートマーク
S1C17xxxyyy PROGRAM ROM                                ←機種名
S224008000
: : : : : "xxxxxxxx.psa"
S804000000FB
¥END                                                   ←ROM データエンドマーク
¥OPTION1                                               ←ファンクションオプションスタートマーク
* S1C17xxx FUNCTION OPTION DOCUMENT V x.xx             ←機種名/バージョン
*
* FILE NAME zzzzzzzz.FDC
* USER'S NAME SEIKO EPSON CORPORATION
* INPUT DATE yyyy/mm/dd
* COMMENT SAMPLE DATA
* "xxxxxxxx.fdc"
* *** OPTION NO.1 ***
* --- OSC1 SYSTEM CLOCK ---
* Crystal(32.768KHz) ---- Selected
OPT0101 01
: : : : :
OPTnn01 01
*EOF
¥END                                                   ←ファンクションオプションエンドマーク
```

10 その他のツール

この章では本パッケージに含まれるその他のツールについて説明します。

10.1 objdump.exe

10.1.1 機能

objdump は elf 形式バイナリファイルの内部データを表示するツールです。逆アセンブルコード、生データ、セクション構成、セクション配置アドレスとデータサイズ、リロケータブル情報シンボルテーブルを表示することができます。
objdump の詳細については、gnu のユーティリティに関する資料を参照してください。

10.1.2 入力ファイル

●実行形式オブジェクトファイル

ファイル形式:　elf 形式のバイナリファイル
ファイル名:　<*ファイル名*> `.elf`
内容:　リンカにてリンク処理が終了した後の実行形式ファイルです。
ダンプ時のアドレスは、絶対アドレス表示になります。

●オブジェクトファイル

ファイル形式:　elf 形式のバイナリファイル
ファイル名:　<*ファイル名*> `.o`
内容:　アセンブル後のオブジェクトファイルです。
ダンプ時のアドレスは、ファイル先頭またはセクション先頭からの相対アドレス表示になります。

10.1.3 使用方法

●起動フォーマット

`objdump` <*オプション*> <*入力ファイル名*>
<*入力ファイル名*>：ダンプするオブジェクトファイル名を指定します。

●オプション

以下の起動時オプションが指定可能です。

`-d`
機能: **逆アセンブル表示**
説明: ファイル内の実行可能なすべてのセクションを逆アセンブルして表示します。ソースファイルの混合表示は行いません。

`-h`
機能: **セクション情報表示**
説明: セクション構成とそのサイズ、およびアドレスを表示します。

`-g`(gcc4 のみ)
機能: **デバッグ情報の変換表示**
説明: デバッグ情報を元にソースとアドレスとの関連を表示します。また、グローバルシンボルのデータ型も表示します。

`-t`
機能: **グローバルシンボル情報表示**
説明: グローバルシンボルの一覧を表示します。これにはローカルラベルも含まれます。

-s
機能: **16 進ダンプ表示**
説明: すべてのセクション情報を 16 進数で表示します。未解決シンボルに相当するデータの場合、正しい値は表示されません。

-D
機能: **全セクションの逆アセンブル表示**
説明: すべてのセクションの逆アセンブル情報を表示します。

-G
機能: **デバッグ情報の生データ表示**
説明: stab 形式のデバッグ情報の生データを表示します。

-S
機能: **ミックス表示**
説明: ファイル内の実行可能なすべてのセクションを逆アセンブルして表示します。可能な場合はソースとの混合表示を行います。

オプションの前後には 1 個以上のスペースが必要です。

例: `c:¥EPSON¥GNU17V3¥GCC6¥objdump -S test.elf`

10.1.4 エラーメッセージ

objdump のエラーメッセージは次のとおりです。

表 10.1.4.1 エラーメッセージ

エラーメッセージ内容	内容
/cygdrive/X/*path to objdump*/objdump: *filename*: File format not recognized	認識できないファイル(*filename*)が指定されました。 elf形式のファイルを指定してください。

10.1.5 注意事項

- 逆アセンブル表示を行った場合、表示する情報量が極端に多いと表示は途中で終了します。
- リンク前の.o ファイルのダンプを行う場合、表示されるアドレスは絶対アドレスでなく、各セクションの先頭アドレスからの相対アドレス表示となります。また、各セクションの先頭アドレスは 0x0 番地として扱われます。

10.2　objcopy.exe

10.2.1 機能

objcopy は gnu 標準のオブジェクトファイル形式変換ユーティリティで、オブジェクトファイルのコピーやデータ形式の変換に使用します。
S1C17 Family のアプリケーション開発においては、elf 形式オブジェクトファイルを ROM 書き込み用のモトローラ S3 形式ファイルに変換するのに使用します。
objcopy は多種の機能（オプション）、ファイル形式に対応していますが、ここでは elf 形式ファイル→モトローラ S3 形式ファイル変換に絞って説明を行います。**objcopy** の詳細については、gnu のユーティリティに関する資料を参照してください。

10.2.2 入出力ファイル

入力ファイル

●オブジェクトファイル

ファイル形式:　elf 形式のバイナリファイル
ファイル名:　<*ファイル名*> .elf
内容:　リンカにてリンク処理が終了した後の実行形式ファイルです。

出力ファイル

●SA ファイル(ROM データ)

ファイル形式:　モトローラ S3 形式ファイル
ファイル名:　<*ファイル名*> .sa
内容:　ROM に書き込むためのファイルです。複数の ROM を使用するシステムでは、elf 形式オブジェクトファイル内の、各 ROM に書き込むセクションのみをそれぞれ変換してファイルを作成します。

10.2.3 使用方法

● 起動フォーマット

objcopy ***<オプション> <入力ファイル名>*** [***<出力ファイル名>***]

[ ]は省略可能なことを示します。

<*入力ファイル名*>: 変換する elf 形式オブジェクトファイル名を指定します。
<*出力ファイル名*>: 変換後のモトローラ S3 形式ファイル名を指定します。

注: *<出力ファイル名>*を省略すると、objcopy は一時ファイルを作成して処理を実行後、そのファイルの名称を入力ファイルの名称に変更します。したがって、入力ファイルは削除されます。

●オプション

S1C17 Family のアプリケーション開発においては、主に以下のオプションを使用します。

-I elf32-little
機能: **入力ファイル形式指定**
説明: 入力ファイルのフォーマットを elf に指定します。

-O srec
機能: **モトローラ形式出力**
説明: 出力ファイルのフォーマットをモトローラ形式に指定します。'-I elf32-little'と一緒に指定してください。

-O binary
機能: **バイナリ形式出力**
説明: 出力ファイルのフォーマットをバイナリ形式に指定します。'-I elf32-little'と一緒に指定してください。

--srec-forceS3
機能: **モトローラ S3 形式指定**
説明: 出力ファイルのレコードフォーマットをモトローラ S3 形式にします。`-O srec` オプションと共に指定します。

例: `-O srec --srec-forceS3 ...`

-R ***SectionName***
機能: **セクションの削除**
説明: *SectionName* という名称のセクションを出力ファイルに含めないことを指定します。このオプションは複数回の指定が可能です。'-I elf32-little'と一緒に指定してください。

-v(または**--verbose**)
機能: **詳細出力モード**
説明: 修正されたすべてのオブジェクトファイルを表示します。

-V(または**--version**)
機能: **バージョン表示**
説明: **objcopy** のバージョンを表示して終了します。

--help
機能: **Usage 出力**
説明: **objcopy** の usage を表示して終了します。

10.2.4 SA ファイル(ROM データ)の作成方法

コマンドプロンプトウィンドウを開き、次のコマンドラインで **objcopy** を実行します。
```
C:¥EPSON¥GNU17V3¥gcc6¥objcopy -I elf32-little -O srec -R SectionName --srec-forceS3
Input  File OutputFile
```

上記のコマンドにより、-R オプションで指定した以外のセクションが S3 レコードに変換され、出力ファイルが生成されます。

例: input.elf からすべてのセクションのデータを抜き出し、output.sa に書き出します。
```
C:¥EPSON¥GNU17V3¥GCC6¥objcopy -I elf32-little -O srec --srec-forceS3 input.elf output.sa
```

10.3 ar.exe

10.3.1 機能

ar はアーカイブファイルとしてまとめられたファイル群を保守するための gnu 標準のユーティリティです。通常、リンカ **ld** が使用するライブラリファイルの作成、更新に使用します。**ar** の詳細については、gnu のユーティリティに関する資料を参照してください。

10.3.2 入出力ファイル

●オブジェクトファイル

ファイル形式: elf 形式のバイナリファイル
ファイル名: <*ファイル名*>.o
内容: リロケータブルオブジェクトファイルです。
ar はこの形式のオブジェクトファイルを入力してアーカイブ（ライブラリ）に追加、あるいはアーカイブからオブジェクトを抽出してこの形式のファイルを作成します。

●アーカイブファイル（ライブラリファイル）

ファイル形式: バイナリ形式のアーカイブファイル
ファイル名: <*ファイル名*>.a
内容: リンカ **ld** に入力可能なライブラリファイルです。

10.3.3 使用方法

●起動フォーマット

ar ***<キー>*** [***<修飾子>***] [***<追加位置>***] ***<アーカイブ>*** [***<オブジェクト>***]

[ ]は省略可能なことを示します。

<キー>、<*修飾子*>: **ar** の処理内容を指定します。
<*追加位置*>: <*オブジェクト*>を追加する位置をアーカイブ中のオブジェクト名で指定します。
<*アーカイブ*>: 編集するアーカイブファイル名を指定します。
<*オブジェクト*>: 追加、抽出、移動、削除するオブジェクトファイル名を指定します。複数のオブジェクトファイルをスペースで区切って指定可能です。

●キー

d <*オブジェクト*>をアーカイブファイルから削除します。

m <*オブジェクト*>をアーカイブファイルの最後に移動します。修飾子 a または b と併用してアーカイブ中の指定位置に移動することも可能です。

q <*オブジェクト*>をアーカイブファイルの最後に追加します。アーカイブファイル内のシンボルテーブルは更新されません。

r アーカイブファイル中の<*オブジェクト*>を置き換えます。アーカイブファイルに同名のオブジェクトが存在しない場合はファイルの最後に追加されます（修飾子 a または b と併用して追加位置の指定も可能）。

t アーカイブファイルの内容一覧、または指定した<*オブジェクト*>の一覧を表示します。

x アーカイブファイルから指定した<*オブジェクト*>を抽出してファイルを作成します。<*オブジェクト*>が指定されない場合、アーカイブファイル内のすべてのオブジェクトが取り出されます。

●修飾子

a r または m キーと併用して<*オブジェクト*>をアーカイブファイル中の<*追加位置*>の後に置きます。
<*追加位置*>にはアーカイブファイル内のオブジェクト名を指定します。

b a と同様ですが、<*オブジェクト*>を<*追加位置*>の前に置きます。

s アーカイブファイルのシンボルテーブルを強制的に更新します。

u r キーと併用し、<*オブジェクト*>の中でアーカイブファイルに収めた後に変更があったもののみを置き換えます。

v 冗長モード。実行した動作を表示します。

キーと修飾子は、間に空白を入れずにつなげて記述してください。

●使用例

(1) アーカイブファイルの新規作成

```
ar rs mylib.a func1.o func3.o

(mylib.a: func1.o + func3.o)
```

アーカイブファイル（mylib.a）が存在しない場合は新規に作成され、指定のオブジェクトファイル（func1.o、func3.o）が指定順にすべて追加されます。

(2) オブジェクトの追加

```
ar rs mylib.a func4.o func5.o

(mylib.a: func1.o + func3.o + func4.o + func5.o)
```

mylib.a の最後に func4.o、func5.o を追加します。

(3) 指定位置へのオブジェクトの追加

```
ar ras func1.o mylib.a func2.o

(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
```

mylib.a 内の func1.o の後ろに func2.o を追加します。

(4) オブジェクトの置き換え

```
ar rus mylib.a func1.o func2.o func3.o func4.o func5.o

(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
```

func1.o、func2.o、func3.o、func4.o、func5.o の中で、mylib.a に追加した後に更新されたファイルがあれば、そのオブジェクトのみ置き換えられます。

(5) オブジェクトの抽出

```
ar x mylib.a func5.o

(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
```

mylib.a から func5.o を取り出してオブジェクトファイルを作成します。アーカイブファイルの内容は変更されません。

(6) オブジェクトの削除

```
ar ds mylib.a func5.o

(mylib.a: func1.o + func2.o + func3.o + func4.o)
```

mylib.a から func5.o を削除します。

10.4 moto2ff.exe

10.4.1 機能

moto2ff は開始アドレスとブロックサイズを指定して、モトローラ S3 形式ファイルを読み込み、ファイル内のデータが存在しない空きアドレスを 0xff のデータで埋めた出力ファイルを生成します。
S1C17 Family のアプリケーション開発においては、objcopy により生成されたモトローラ S3 形式ファイルから ROM エリアのデータを抽出するために使用します。
ここで生成した ROM エリアのデータを sconv32、winmdc17 で処理し、最終的にセイコーエプソンに提出する PA ファイル(提出用データ)を生成します。PA ファイルの生成手順については、"9.2 提出用データの作成"を参照してください。

10.4.2 入出力ファイル

入力ファイル

●SA ファイル(ROM データ)

ファイル形式: モトローラ S3 形式ファイル
ファイル名: <*ファイル名*> .sa
内容: 実行形式の elf ファイルを **objcopy** で変換したモトローラ S3 形式のファイルです。

出力ファイル

●SAF ファイル(ROM データ)

ファイル形式: モトローラ S3 形式ファイル
ファイル名: <*ファイル名*> .saf
内容: 指定したアドレスの ROM データです。未使用アドレスには 0xff が埋め込まれます。

10.4.3 起動フォーマット

moto2ff *〈データの開始アドレス〉〈データのブロックサイズ〉〈入力ファイル名〉*

<*データの開始アドレス*> : 入力ファイル内のアドレスで、ここからデータの出力を開始します。16 進数値で指定します。
<*データのブロックサイズ*> : 出力するデータのブロックサイズ（バイト単位）です。16 進数値で指定します。
<*入力ファイル名*> : 0xff 埋めを行うモトローラ S3 形式ファイル名を指定します。
入力ファイル名はパス名、拡張子含めて 128 文字以内です。入力ファイルはパスの指定ができますが、出力ファイルはカレントディレクトリに出力されます。

- 引数の指定がない場合は使用法メッセージが表示されます。
- 同名の出力ファイルがすでにあった場合は上書きされます。
- エラー発生時はエラーメッセージが表示され、出力ファイルは生成されません。
- 入力したモトローラ S3 形式ファイルに、開始アドレスとブロックサイズによって指定した範囲外のデータがある場合、以下のエラーメッセージが出力され、出力ファイルは生成されません。
  エラーメッセージ :
  `Error: FILENAME contains data outside of specified range (STARTADDR:SIZE)`
- 入力したモトローラ S3 形式ファイルにアドレスが重複したデータがあった場合は、後のデータで上書きされ出力されます。

- データの開始アドレスとデータのブロックサイズは、各機種のテクニカルマニュアルを参照し、機種に合わせて正確に入力してください。正しく入力されていない場合、最終的に PA ファイルを生成する **winmdc17** の処理でエラーとなります。
- 正常に終了した場合、以下のメッセージが標準出力に出力されます。
  `moto2ff : Convert Completed`
- [-f] : 強制出力オプションです。入力したモトローラ S3 形式ファイルに、開始アドレスとブロックサイズで指定した範囲外のデータがある場合でも、指定した範囲のみを切り出して 0xff 埋めを行った出力ファイルを生成します。範囲外のデータがある場合はワーニングメッセージを出力します。

10.4.4 エラー/ワーニングメッセージ

moto2ff が出力するエラー/ワーニングメッセージを以下に示します。

表 10.4.4.1 エラーメッセージ

エラーメッセージ	内容
Input filename is over 128 letters.	入力ファイル名が128文字を超えています。
Cannot open input file "*FILENAME*".	入力ファイル*FILENAME*が開けません。
Cannot open output file "*FILENAME*".	出力ファイル*FILENAME*が開けません。
Motorola S3 checksum error.	モトローラS3形式ファイルの読み込みでチェックサムエラーが発生しました。
Cannot allocate memory.	メモリが確保できません。
FILENAME contains data outside of specified range ("*STARTADDR*":"*SIZE*")	*FILENAME*には、指定された範囲(*STARTADDR*から*SIZE*)以外のデータも含まれています。出力ファイルは生成されません。

表 10.4.4.2 ワーニングメッセージ

エラーメッセージ	内容
Invalid file format in "*FILENAME*" line "*NUMBER*".	入力ファイル*FILENAME*の*NUMBER*行目に不正な形式のデータがあります。
FILENAME contains data outside of specified range ("*STARTADDR*":"*SIZE*")	*FILENAME*には、指定された範囲(*STARTADDR*から*SIZE*)以外のデータも含まれていますが、強制出力オプション -f によって出力ファイルが生成されます。

10.4.5 SAF ファイル(ROM データ)の作成方法

objcopy によりモトローラ S3 フォーマットの SA ファイル(ROM データ)を作成後、**moto2ff** により SAF ファイルを作成します。
コマンドプロンプトウィンドウを開き、次のように **moto2ff** を実行します。

例: `C:¥EPSON¥GNU17V3¥>moto2ff 8000 10000 input.sa`

このコマンドは `input.sa` 内のアドレス 0x8000 から始まる 0x10000 バイト分のデータを `input.saf` に出力します。
`input.sa` のアドレス 0x8000 から 0x17fff の範囲にある空きアドレスは、0xff のデータで埋められます。
以上の操作により、内蔵 ROM 用の SAF ファイルが作成されます。

この後、ここで生成した SAF ファイルを **sconv32** を使用してモトローラ S2 フォーマットの PSA ファイル(ROM データ)に変換します。そのファイルを使用して実機上での最終動作確認を行ってください。

最終的には、動作確認済みの PSA ファイルと **winfog17** で生成した FDC ファイル(ファンクションオプションドキュメント)を **winmdc17** で 1 つの PA ファイルにパックし、セイコーエプソンに提出していただきます。

10.5　sconv32.exe

10.5.1 機能

sconv32 はモトローラ S 形式ファイルの変換ツールです。S1C17 Family のアプリケーション開発においては、**moto2ff** により生成されたモトローラ S3 形式の SAF ファイル(ROM データ)をモトローラ S2 形式に変換するために使用します。

ここで変換した PSA ファイル(ROM データ)を使用して実機上での動作確認を行った後、ファイルを **winmdc17** で処理し、最終的にセイコーエプソンに提出する PA ファイル(提出用データ)を生成します。PA ファイルの生成手順については、"9.2 提出用データの作成"を参照してください。

10.5.2 入出力ファイル

入力ファイル

●SAF ファイル

ファイル形式:　モトローラ S3 形式ファイル
ファイル名:　　<*ファイル名*> .saf
内容:　　**moto2ff** で生成したモトローラ S3 形式のファイルです。

出力ファイル

●PSA ファイル

ファイル形式:　モトローラ S2 形式ファイル
ファイル名:　　<*ファイル名*> .psa
内容:　　入力データファイルをモトローラ S2 形式に変換したファイルです。

10.5.3 起動フォーマット

sconv32 S2 <入力ファイル> <出力ファイル名>

S2:　　モトローラ S2 形式に変換することを指定するスイッチです。
<*入力ファイル名*>:　**moto2ff** で生成した SAF ファイル名を指定します。
<*出力ファイル名*>:　モトローラ S2 形式に変換後の出力ファイル名を指定します。
　　拡張子は必ず".psa"としてください。

- 引数の指定がない場合は使用法メッセージが表示されます。
- 同名の出力ファイルがすでにあった場合は上書きされます。
- エラー発生時はエラーメッセージが標準エラーに出力され、出力ファイルは生成されません。
- [Esc]キーにより、変換動作中でも強制停止できます。
- 正常に終了した場合、以下のようなメッセージが出力されます。

```
Sconv32 : Convert Completed. 終了メッセージ        *1
```

*1: 標準出力に出力

10.5.4 エラーメッセージ

sconv32 が出力するエラーメッセージを以下に示します。

表 10.5.4.1 エラーメッセージ

エラーメッセージ	内容
INVALID SWITCH.	無効なスイッチが指定されました。
COMPLEMENT SWITCH ERROR.	出力ファイルのチェックサム補数の指定に誤りがあります。
S FORMAT TYPE ERROR.	出力ファイルのSフォーマットの指定に誤りがあります。
NO INPUT FILE NAME.	入力ファイルが指定されていません。
NO OUTPUT FILE NAME.	出力ファイルが指定されていません。
INPUT SAME FILE.	入力と出力に同じファイルを指定しました。
CANNOT OPEN SOURCE FILE (*filename*).	入力ファイルが見つからないか、開けません。
CANNOT OPEN DESTINATION FILE (*filename*).	出力ファイルが開けません。
SOURCE RECORD TYPE NOT SUPPORT.	入力ファイルのレコードタイプは未対応です。
ADDRESS LENGTH RANGE OVER.	入力ファイルのアドレスが変換後のS形式のアドレス長を超えています。
OTHER ERROR.	その他のエラーが発生しました。

10.6 gpdata.exe

10.6.1 機能

gpdata はバイナリデータ（バイナリ形式）を読み込み、Gang Programmer（S5U1C17001W2）用のユーザ設定情報を付加したファイル"gpdata.bin"を生成します。

gpdata の詳細については Gang Programmer のユーザマニュアルまたは **gpdata** に付属する文書を参照してください。

10.6.2 入出力ファイル

入力ファイル

●BIN ファイル

ファイル形式: バイナリ形式ファイル
ファイル名: *<ファイル名>*.bin
内容: **objcopy** などで生成したバイナリ形式のファイルです。

出力ファイル

●gpdata.bin ファイル

ファイル形式: Gang Programmer ユーザ設定・プログラムデータファイル
ファイル名: gpdata.bin
内容: バイナリデータにユーザ設定情報を付加したファイルです。

10.6.3 使用方法

●起動フォーマット

gpdata **<入力ファイル名> <オプション>**

<入力ファイル名> : **objcopy** などで生成したバイナリデータを指定します。

●オプション

以下の起動時オプションが指定可能です。詳細については Gang Programmer のユーザマニュアルまたは **gpdata** に付属する文書を参照してください。

-v **ベリファイ方式選択**

-d **インタフェース電圧レベル選択（必須）**

-b **プログラム完了時のブザー音有無選択**

-t **機種名設定（必須）**

-a **ユーザプログラムの配置先アドレス（必須）**

-i **シリアル番号初期値設定**

-s **シリアル番号書込み先先頭アドレス設定**

-p **フラッシュメモリセキュリティパスワード**

-f **パラメータ入力ファイル指定**

10.7 ptd.exe

10.7.1 機能

ptd はモトローラ S 形式ファイルを読み込み、指定されたアドレスのデータを変更し、モトローラ S 形式で出力しなおします。

S1C17 Family のアプリケーション開発においては、**sconv32** により生成されたモトローラ S2 形式の PSA ファイル（ROM データ）にフラッシュプロテクトビットを埋め込み、フラッシュプロテクトが設定された PSA ファイルを生成するために使用します。

10.7.2 入出力ファイル

入力ファイル

●PSA ファイル

ファイル形式: モトローラ S2 形式ファイル
ファイル名: <*ファイル名*>.psa
内容: **sconv32** で生成したモトローラ S2 形式のファイルです。

出力ファイル

●PSA ファイル

ファイル形式: モトローラ S2 形式ファイル
ファイル名: <*ファイル名*>_ptd.psa
内容: 入力データファイルにフラッシュプロテクトを設定したファイルです。

10.7.3 使用方法

●起動フォーマット

ptd **<オプション> <入力ファイル名>**

<*入力ファイル名*> : **sconv32** で生成した PSA ファイル名を指定します。

- 引数の指定がない場合は使用法メッセージが表示されます。
- 入力ファイルに対する変更がない場合、処理は正常に終了しますが、出力ファイルを生成しません。
- エラーが発生した場合、エラーメッセージを標準エラーへ出力し、出力ファイルを生成しません。
- 正常に終了した場合、以下のようなメッセージを標準出力へ出力します。

```
ptd:Convert Completed: 終了メッセージ
```

●オプション

以下の起動時オプションが指定可能です。未知のオプションが指定された場合は使用法メッセージが表示されます。

-o <出力ファイル名>
機能: **出力ファイル名を指定**
説明: データ変更結果を出力するファイル名を指定します。指定しなかった場合、入力ファイル"<input>.psa"に対して、出力ファイル"<input>_ptd.psa"を生成します。同名の出力ファイルがすでにあった場合は上書きします。

-w <アドレス>=<データ>[,<データ>]
機能: **アドレスに書き込むデータを指定**
説明: アドレスに書き込むデータを 16 ビット単位で指定します。複数のデータを指定した場合、続くアドレスへデータを書き込みます。指定されたアドレスが入力ファイル上に存在しない場合、エラーとなります。

10.7.4 エラーメッセージ

ptd が出力するエラーメッセージを以下に示します。

表 10.7.4.1 エラーメッセージ

エラーメッセージ	内容
Cannot open input file "FILENAME".	入力ファイルFILENAMEが開けません。
Cannot open output file "FILENAME".	出力ファイルFILENAMEが開けません。
S-format checksum error.	S形式ファイルの読み込みでチェックサムエラーが発生しました。
Unknown address format "ADDRESS"	指定されたアドレスADDRESSの書式が正しくありません。
Unknown data format "DATA"	指定されたデータDATAの書式が正しくありません。
Address "ADDRESS" is out of range	指定されたアドレスADDRESSは入力データに含まれていません。

10.7.5 フラッシュプロテクトの設定方法

S1C17 Family には、内蔵 Flash メモリの内容を保護するために、フラッシュプロテクトを設定できる機種（S1C17554 など）があります。フラッシュプロテクトを設定するには、**sconv32** によりモトローラ S2 フォーマットの PSA ファイル(ROM データ)を作成した後、**ptd** を用いてフラッシュプロテクトが設定された PSA ファイルを作成します。
コマンドプロンプトウィンドウを開き、次のように **ptd** を実行します。

例: `C:¥EPSON¥GNU17V3¥>ptd s1c17554.psa -e 0x27ffc=0xff80,0xffff`

このコマンドは、`s1c17554.psa` 内のアドレス 0x27ffc の値を 0xff80 に、アドレス 0x27ffe の値を 0xffff に書き換え、s1c17554_ptd.psa に出力します。S1C17754 における Flash Protect Bits のアドレスは 0x27ffc（write-protect）と 0x27ffe（data-read-protect）です。この例のように Flash Protect Bits を書き換えた s1c17554_ptd.psa を S1C17554 の内蔵 Flash メモリへ書き込むと、0x8000-0x23fff の領域のデータ書き込みとセクタ消去が禁止された状態になります。

注: フラッシュプロテクト設定の有無、Flash Protect Bits のアドレスは、機種により異なります。詳細は各機種のテクニカルマニュアルを参照してください。

10.8 LCDUtil17（LCD パネルカスタマイズツール）

10.8.1 概要

LCD パネルカスタマイズツール（以下 LcdUtil17 と記述）は組み込みシステムシミュレータ ES-Sim17（v.1.2 以降）でモノクロ LCD パネルの表示をシミュレートするための、LCD パネルレイアウトおよび COM/SEG ポートの割り付けを記載した LCD ファイルを作成します。セグメント LCD のレイアウトはビットマップファイル(.bmp)から作成でき、ES-Sim17 上で実際の製品と同様の画面をシミュレートすることができます。また、本ツールはドットマトリクス LCD のレイアウト作成にも対応しています。

10.8.2 入出力ファイル

図 10.8.2.1 に LcdUtil17 の入出力ファイルを示します。

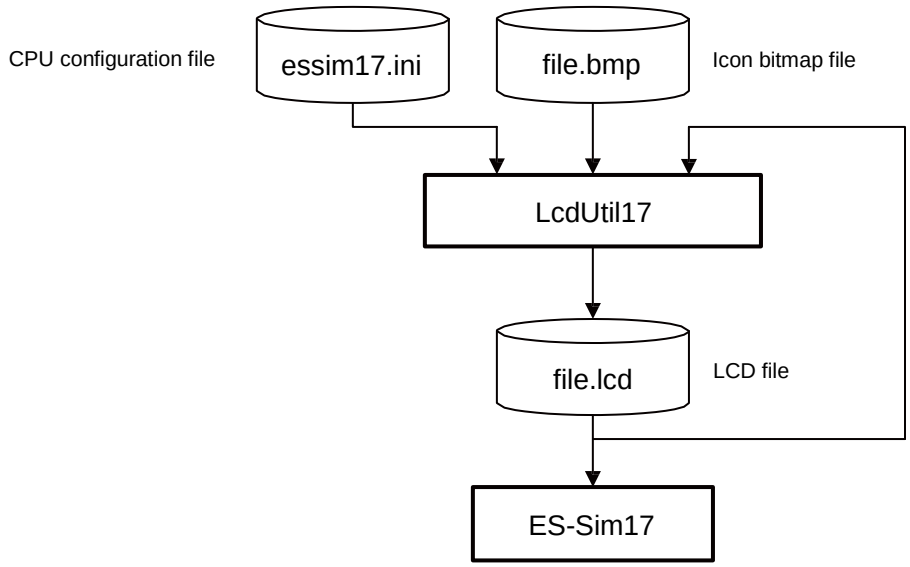


図 10.8.2.1 LcdUtil17 の入出力ファイル

●CPU 構成ファイル（essim17.ini）

シミュレータの機種情報が記録されています。必ずエプソンが提供する設定ファイルを使用してください。ファイルの内容を編集すると、本ツールならびに、ES-Sim17 が正常に動作しなくなる恐れがあります。

●ビットマップファイル（file_name.bmp）

LCD パネルをイメージした（白黒）ビットマップファイルです。ビットマップファイルを読み込んで、それぞれパーツごとにアイコンとして取り込み、LcdUtil17 上でレイアウトを編集することができます。

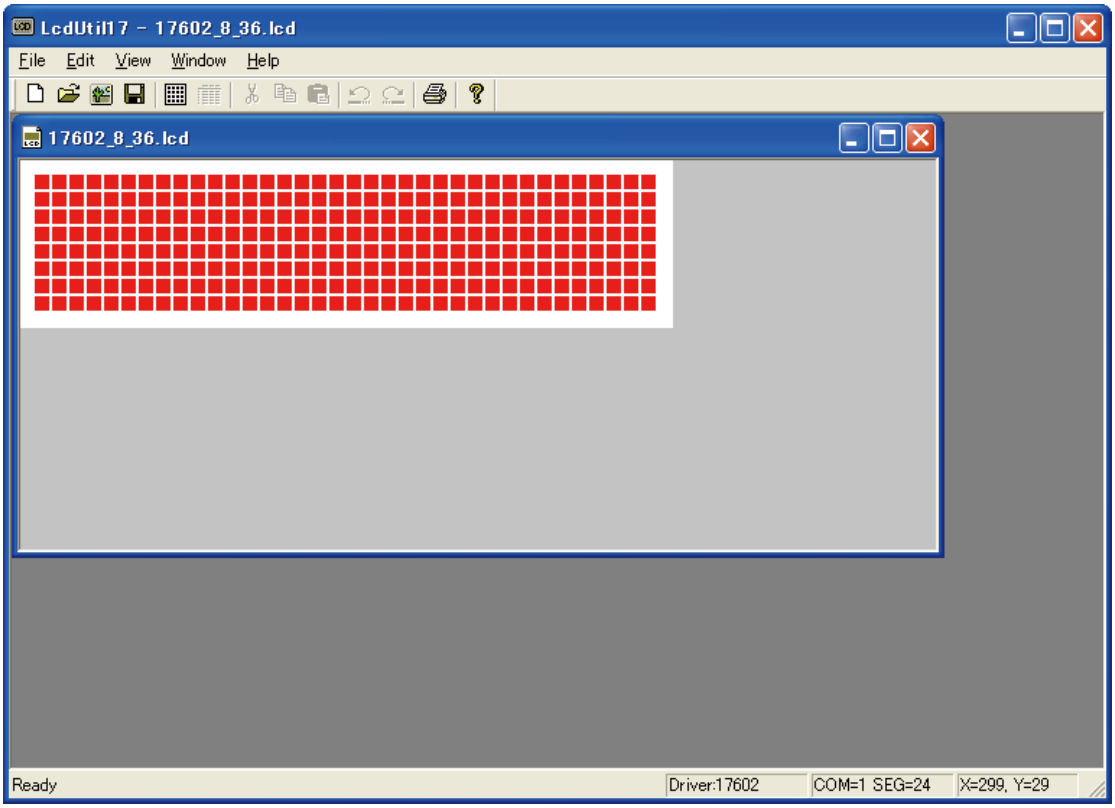
●LCD ファイル

LCD パネルレイアウトと COM/SEG 割り当て情報が記載されたファイルです。このファイルをシミュレータにロードして LCD 表示のシミュレーションを行います。

10.8.3 起動と終了

IDE の[C17]メニューから[Launch LcdUtility]を選択すると LcdUtil17 が起動します。
終了するには、[File]メニューから[Exit]を選択します。

10.8.4 ウィンドウ



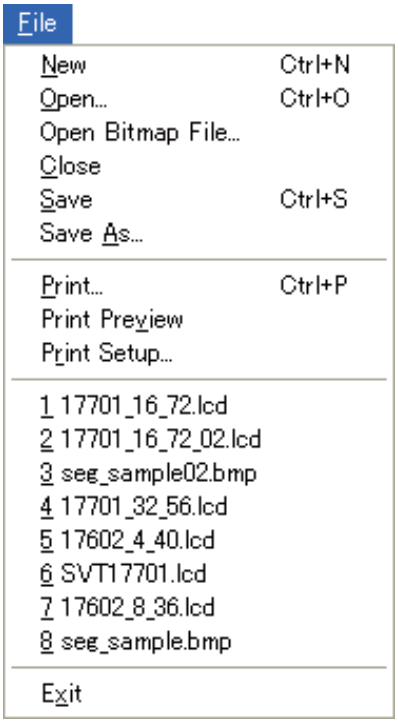
●パネル編集ウィンドウ

ビットマップファイル(.bmp)または LCD ファイル(.lcd)を開くと、このウィンドウに表示されます。このウィンドウ上で LCD パネルのレイアウト、COM/SEG の割り付けを行うことができます。
ウィンドウは同時に複数開くことができ、2 つのウィンドウ間でのアイコンおよびドットマトリクスのドラッグ&ドロップにも対応しています。

10.8.5 メニューとツールバー

10.8.5.1 メニュー

[File]メニュー



[New]（[Ctrl]+[N])
新しいパネル編集ウィンドウを開きます。
[Open…]（[Ctrl]+[O])
LCD ファイル(.lcd)を開きます。
[Open Bitmap File…]
ビットマップファイル(.bmp)を開きます。
[Close]
アクティブなパネル編集ウィンドウを閉じます。
[Save]（[Ctrl]+[S])
アクティブなパネル編集ウィンドウの内容を LCD ファイル(.lcd)に保存（上書き）します。
[Save As…]
アクティブなパネル編集ウィンドウの内容を別の名称で LCD ファイル(.lcd)に保存します。
[Print…]（[Ctrl]+[P])
アクティブなパネル編集ウィンドウのビットマップを印刷します。
[Print Preview]
アクティブなパネル編集ウィンドウの印刷イメージを表示します。
[Print Setup…]
用紙やプリンタを選択するダイアログボックスを表示します。
ファイルリスト
8 件まで開いたファイルの履歴を表示し、開くことができます。
[Exit]
LcdUtil17 を終了します。

[Edit]メニュー

[Cut]　([Ctrl]+[X])
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへカットします。
[Copy]　([Ctrl]+[C])
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへコピーします。
[Paste]　([Ctrl]+[V])
クリップボードにコピーされているパーツをパネル編集ウィンドウの左上端にペーストします。
[Insert dot matrix]　([Ctrl]+[M])
パネル編集ウィンドウにドットマトリクスを挿入します。ダイアログボックスでサイズなどを編集できます。
[Icon List]　([Ctrl]+[I])
アクティブなパネル編集ウィンドウにあるアイコンの一覧を表示します。この一覧の中でも COM/SEG の割り付けを行えます。
[Resize LCD]
LCD パネルのサイズを設定します。新規のパネル編集ウィンドウのサイズは 640×480 です。
[Group Icon]
複数のアイコンを 1 つにグループ化します。
[Release Group]
グループを解除し、個々のアイコンに戻します。

[View]メニュー

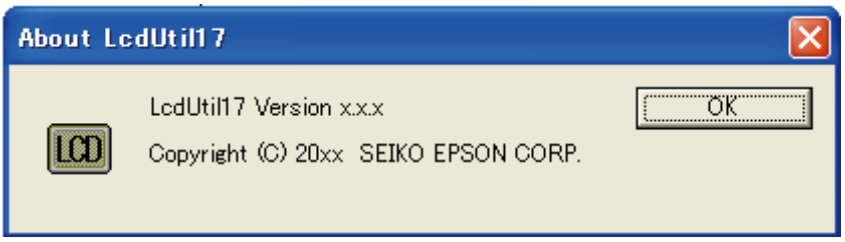
[Toolbar]
ツールバーの表示/非表示を切り替えます。
[Status Bar]
ステータスバーの表示/非表示を切り替えます。

[Window]メニュー

[Cascade]
パネル編集ウィンドウをカスケード表示します。
[Tile]
パネル編集ウィンドウをタイル表示します。
[Arrange Icons]
パネル編集ウィンドウを最小化してウィンドウ下側へ整列させます。
ウィンドウリスト
現在開いているパネル編集ウィンドウ名がリストされます。
ここでパネル編集ウィンドウを選択すると、選択されたパネル編集ウィンドウがアクティブになります。

[Help]メニュー

[About LcdUtil…]
LcdUtil17 のバージョン情報を表示します。



10.8.5.2 ツールバーボタン

[New] ボタン
新しいパネル編集ウィンドウを開きます。

[Open] ボタン
LCD ファイル(.lcd)を開きます。

[Bitmap] ボタン
ビットマップファイル(.bmp)を開きます。

[Save] ボタン
アクティブなパネル編集ウィンドウの内容を LCD ファイル(.lcd)に保存（上書き）します。

[Dot Matrix] ボタン
パネル編集ウィンドウにドットマトリクスを挿入します。

[Icon List] ボタン：[Edit]-[Icon List]
アクティブなパネル編集ウィンドウにあるアイコンの一覧を表示します。

[Cut] ボタン
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへカットします。

[Copy] ボタン
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへコピーします。

[Paste] ボタン
クリップボードにコピーされているパーツをパネル編集ウィンドウの左上端にペーストします。

[Undo] ボタン
3 つ前までの操作へ戻ることができます。
Undo 可能な操作：アイコン/ドットマトリクスの移動、カット、ペースト、SEG/COM 変更、グルー プ化、グループ化解除

[Redo] ボタン
Undo で戻した操作をやり直すことができます。

[Print] ボタン
アクティブなパネル編集ウィンドウのビットマップを印刷します。

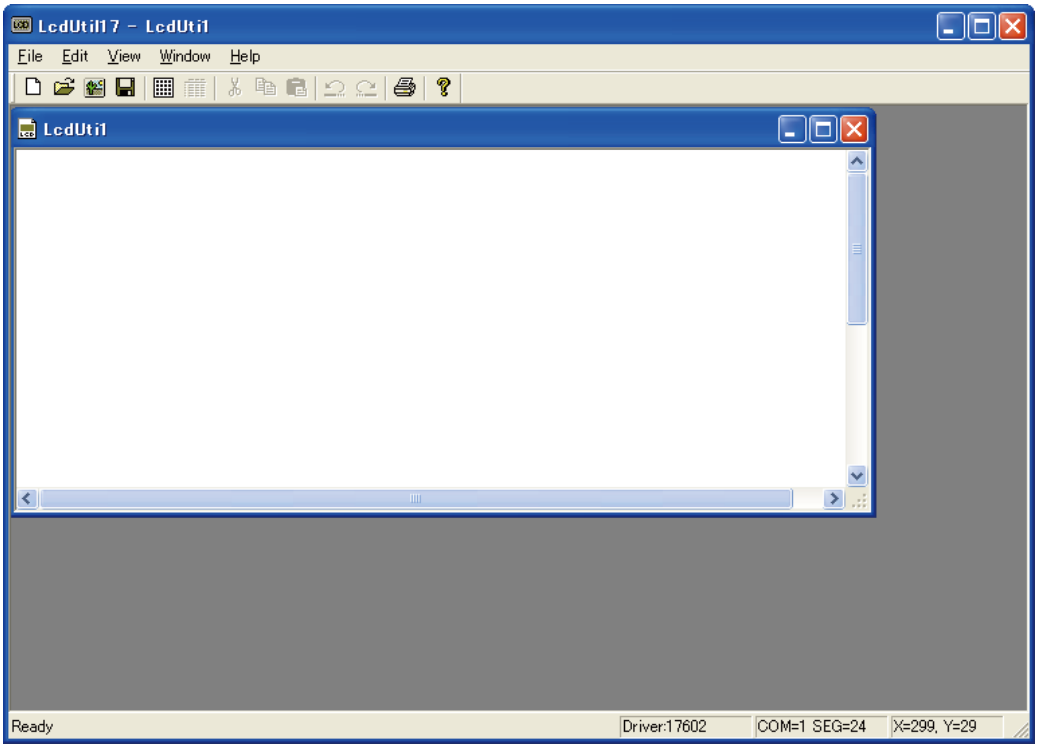
[About] ボタン
LcdUtil17 のバージョン情報を表示します。

10.8.6 LCD ファイルの作成

パネル編集ウィンドウにはアイコンとドットマトリクスを実際の LCD パネルのようにレイアウトすることができ、COM/SEG の割り付けも行えます。以下、その方法を説明します。

10.8.6.1 ドットマトリクス LCD パネルの作成

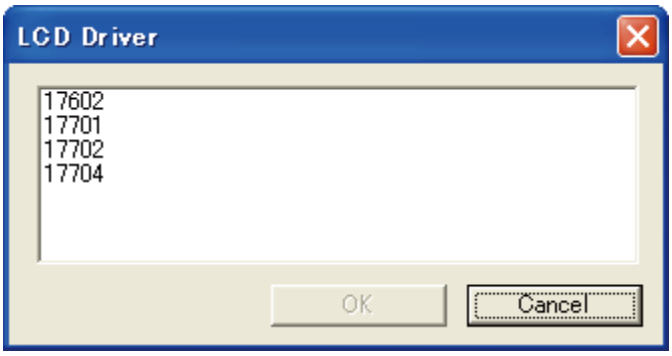
1)　[File]メニューから[New]を選択します。
　　空白のパネル編集ウィンドウが開きます。



2)　[Edit]メニューから[Resize LCD]を選択します。
　　[Resize LCD image]ダイアログボックスが開きます。

　　ここで LCD パネルのサイズを入力し、[OK]ボタンをクリックします。
　　新規作成時のサイズは 640×480 ドットです。
　　ドットマトリクスのサイズを計算し、LCD パネルのサイズを設定してください。

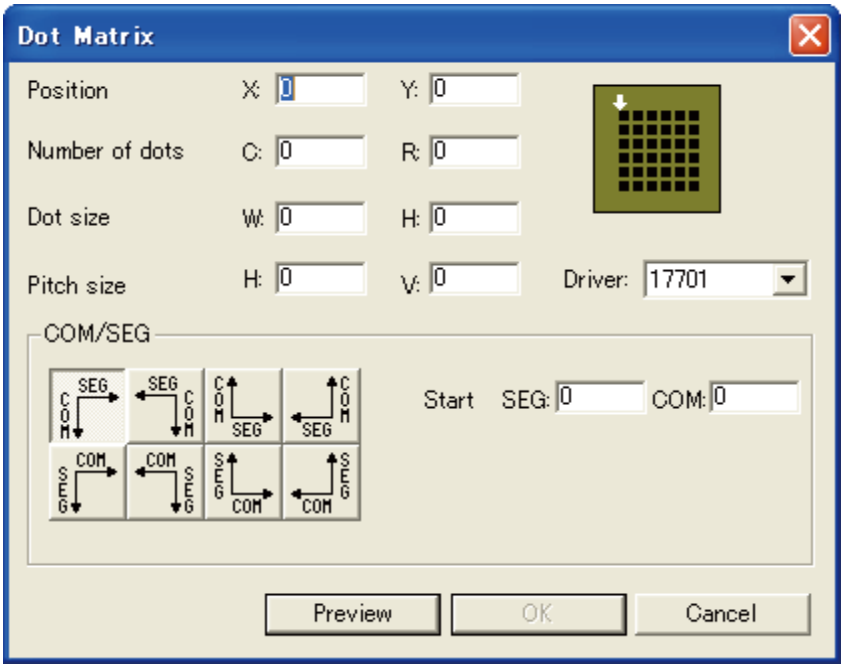
3)　[Edit]メニューから[Insert dot matrix]を選択します。
LCD ドライバが設定されていない場合は[LCD Driver]ダイアログボックスが表示されます。
リストから開発する機種を選択して[OK]ボタンをクリックします。



※　このダイアログボックスは、LCD ドライバが設定されていない場合にのみ表示されます。すでに作成済みの LCD ファイルを読み込んだ場合や、IDE でプロジェクトを選択した状態で IDE の[LCDUtility を起動]ボタンをクリックした場合は、LCD ドライバがすでに設定されており、このダイアログボックスは表示されません。

注：LCD ドライバは1度設定すると変更することができませんので、選択には注意してください。

4)　[Dot matrix]ダイアログボックスが表示されます。



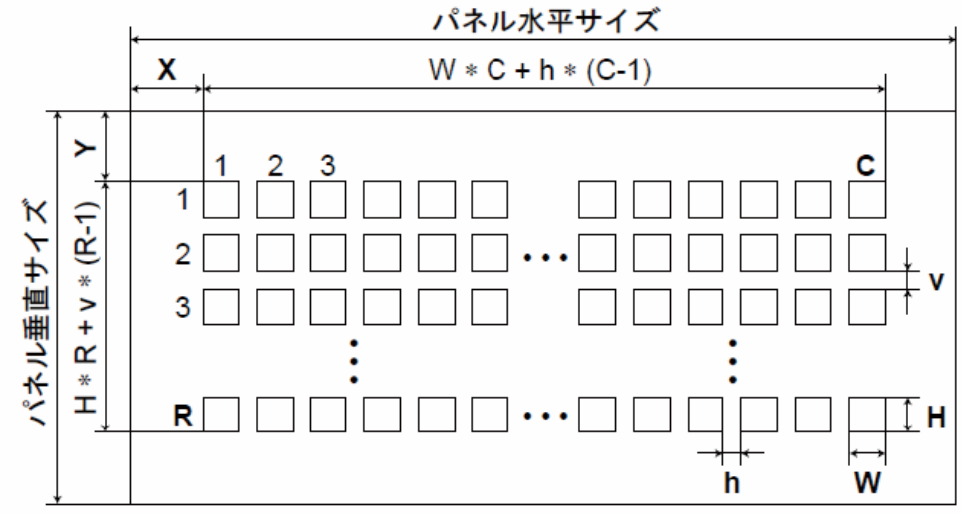


図 10.8.6.1.1　ドットマトリクスの設定

[Dot matrix]ダイアログボックスで以下の設定を行ってください。

Position

ドットマトリクスの左上端の座標を指定します。（図の X と Y）

Number of dots

ドットマトリクスの水平方向、垂直方向のドット数を指定します。（図の C と R）

Dot size

ドットマトリクスの 1 つのドットの大きさをドット数で指定します。（図の W と H）

Pitch size

ドットマトリクスのドットの間隔をドット数で指定します。（図の h と v）

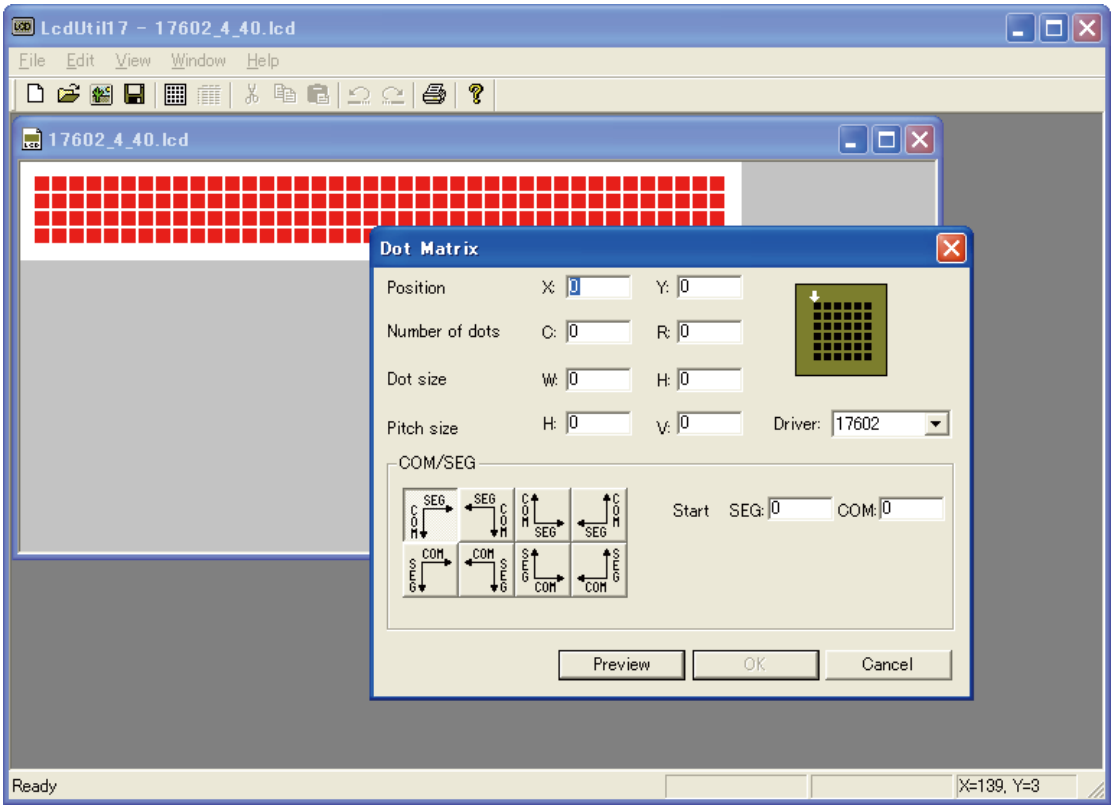
Driver

現在設定されている LCD ドライバ名を表示します。変更はできません。
COM/SEG 値の上限は LCD ドライバで設定されています。

COM/SEG

COM/SEG ポートの割り付け方向を選択します。[Start]テキストボックスに指定した COM/SEG の値から順にポート番号を割り付けます。

上記の各設定を行った後、[Preview]ボタンをクリックすると、パネル編集ウィンドウ上に確認用のドットマトリクスが表示されます。[OK]ボタンをクリックするとこの設定でドットマトリクスが作成されます。



COM/SEG ポートを割り付けされたドットマトリクスは赤で表示されます。
ドットマトリクスをダブルクリックすると、[Dot matrix]ダイアログボックスが表され、ダイアログボックスの設定を変更することができます。

注：ドットマトリクスをコピー＆ペーストした場合、位置とサイズ情報は保持されますが、ポート割り付け情報は破棄されます。割り付け情報のないドットマトリクスは黒で表示されます。

10.8.6.2　セグメント ICD パネルの作成

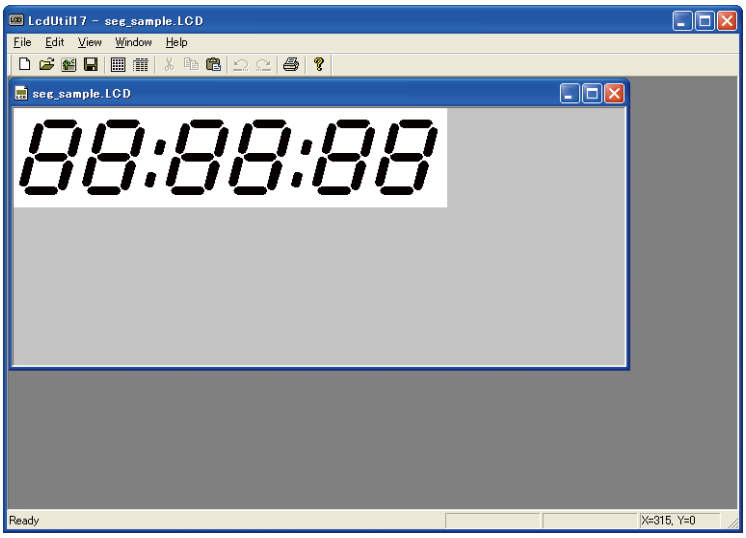
1) セグメント LCD 用にアイコンのビットマップファイルを用意します。
ビットマップファイルは一般のペイントソフトで作成できますが、以下の点に注意してください。

色数とファイル形式
背景を白、アイコンを黒で作成し、白黒ビットマップ形式(.bmp)で保存してください。
※カラーのビットマップファイルを読み込むことは可能ですが、正常に 2 値化されない場合があります。

サイズ
ビットマップファイルのサイズは 1280×1024 以下で作成してください。

C ファイル数
アイコンのいくつかを別のビットマップファイルに分けて作成し、LCDUtil17 上で 1 つのセグメント LCD パネルにすることができます。ただし、LCDUtil17 には簡易的な編集機能しかありませんので、1 つのビットマップファイルに作成しておくことを推奨します。

2) ビットマップファイルを読み込みます。
LCDUtil17 を起動し、[File]メニューから[Open Bitmap File]を選択します。
作成したビットマップファイルを選択してください。
パネル編集ウィンドウが開き、読み込んだビットマップを表示します。



3) アイコンのレイアウトを編集します。

アイコンの状態
アイコンの状態は以下の 3 種類があります。
　黒：COM/SEG 情報　未設定
　赤：COM/SEG 情報　設定済
　青：選択状態

アイコンの編集
選択状態のアイコンに対して、以下の操作ができます。

位置変更
マウスによるドラッグで、アイコンの移動ができます。また、別のパネル編集ウィンドウからアイコンをドラッグアンドドロップすることでアイコンの移動が行えます。

カット、コピー
[Edit]メニューまたはツールバーにあるボタンをクリックしてください。

ペースト
[Edit]メニューまたはツールバーにあるボタンをクリックしてください。アイコンは LCD パネルの左上の位置にペーストされます。

削除
[Delete]キーで削除できます。

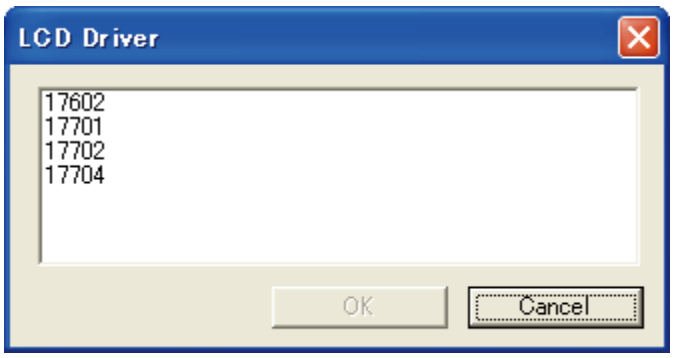
グループ化
[Ctrl]キーを押しながらアイコンをクリックすることで、複数のアイコンを選択できます。この状態で[Edit]メニューから[Group Icon]を選択すると、選択されたパーツがグループ化され、1 つのアイコンとして選択できます。グループ化を解除するには、[Edit]メニューから[Release Group]を選択してください。

注：グループ化したときの COM/SEG 情報は、最後に選択したアイコンの COM/SEG 情報が反映されます。グループ化解除したときのアイコンの COM/SEG 情報は、グループ化していた時の COM/SEG 情報が残ります。

注：LCDUtil17 の編集機能は簡易的なものです。できるだけビットマップ作成時にレイアウトを決定しておくことを推奨します。

4)　アイコンにポートの割り付けを行います。
アイコンをダブルクリックします。
[LCD Driver]ダイアログボックスが表示されます。

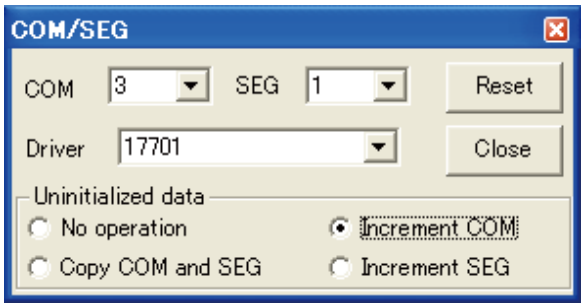
4-1)　LCD ドライバが設定されていない場合は[LCD Driver]ダイアログボックスが表示されます。
リストから開発する機種を選択して[OK]ボタンをクリックします。



※このダイアログボックスは、LCD ドライバが設定されていない場合にのみ表示されます。すでに作成済みの LCD ファイルを読み込んだ場合や、IDE でプロジェクトを選択した状態で IDE の[LCDUtility を起動]ボタンをクリックした場合は、LCD ドライバがすでに設定されており、このダイアログボックスは表示されません。

注：LCD ドライバは1度設定すると変更することができませんので、選択には注意してください。

4-2)　[COM/SEG]ダイアログボックスが表示されます。



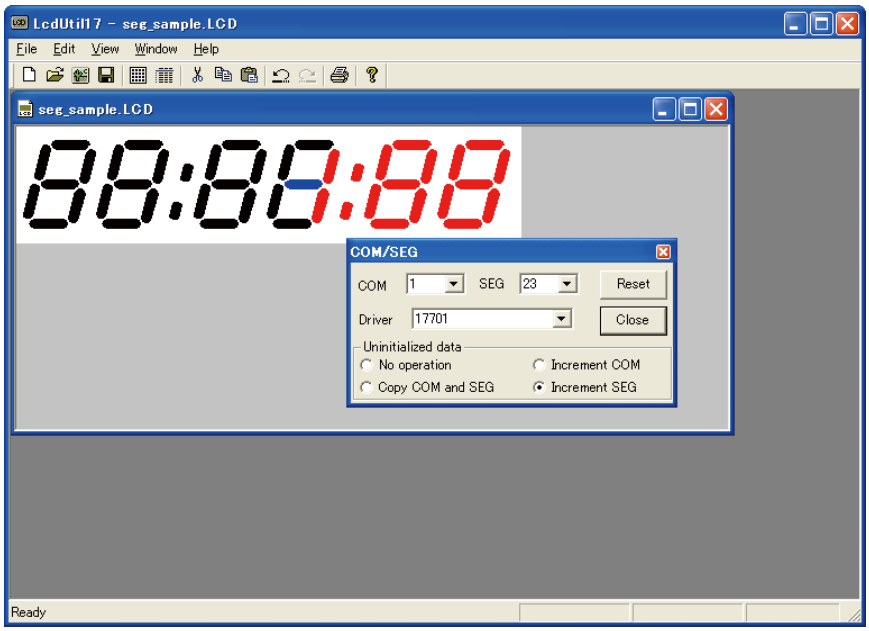
COM、SEG
プルダウンリストから選択します。COM/SEG の割り付けは変更直後に反映されます。

[Reset]ボタン
クリックすると COM/SEG の内容がクリアされ、COM/SEG 未設定状態となります。

Driver
現在設定されている LCD ドライバ名を表示します。変更はできません。COM/SEG 値の上限は LCD ドライバで設定されています。

Uninitialized data
COM/SEG 未設定のアイコンをクリックしたときの設定を決めることができる。

No operation	COM/SEG 情報を変更しません。
Copy COM and SEG	表示している COM/SEG 情報をアイコンに設定します。
Increment COM	表示している COM/SEG 情報に、COM をインクリメント（+1）して設定します。
Increment SEG	表示している COM/SEG 情報に、SEG をインクリメント（+1）して設定します。



注：アイコンをコピー＆ペーストした場合や、他のパネル編集ウィンドウから移動したアイコンは、ポート割り付け情報は破棄されます。割り付け情報のないドットマトリクスは黒で表示されます。

5)　アイコンリストの表示を行います。
[Edit]メニューから[Icon List]を選択します。
現在アクティブなパネル編集ウィンドウ内のアイコンの一覧が表示されます。

リスト内のアイコンをクリックして選択すると、パネル編集ウィンドウ内のアイコンも青く表示され、どのアイコンに対応しているか分かります。
このウィンドウでもアイコンの COM/SEG 情報を編集することができます。

10.8.7 ショートカットキーリスト

LCDUtil17 で使用できるショートカットキーの一覧を示します。

表 10.8.7.1 ショートカットキー一覧

機能	ショートカットキー
コピー	Ctrl ＋ C Ctrl ＋ Insert
カット	Ctrl ＋ X Delete
ペースト	Ctrl ＋ V Shift ＋ Insert
アイコンリスト	Ctrl ＋ I
ドットマトリクス	Ctrl ＋ M
新規作成	Ctrl ＋ N
オープン	Ctrl ＋ O
上書き保存	Ctrl ＋ S
プリント	Ctrl ＋ P
Undo	Ctrl ＋ Z Alt ＋ BS
Redo	Ctrl ＋ Y

10.8.8 ワーニングメッセージ/エラーメッセージ

LCDUtil17 で表示されるワーニングメッセージの一覧を示します。

表 10.8.8.1 ワーニングメッセージ一覧

No.	新規作成/ファイルオープン時
1	・空のドキュメントの作成に失敗しました。 ・新しいドキュメントを作成できません。
	原因: 100ウィンドウを超えたため、新たにウィンドウを作成できなかった。メモリ不足で新たにウィンドウを作成できなかった。
	対応: ウィンドウ数を99以下にしてください。他のアプリケーションを閉じてメモリを解放してください。
2	予期しないファイル形式です。
	原因: 正常なlcd形式、bmp形式のファイルが読み込まれなかった。サポートしていないドライバのlcdファイルだった。
	対応: LcdUtil17対応のファイルを指定してください。
	値入力時
3	・????から????までの整数を入力してください。 ・????から????までの有効な数値を入力してください。
	原因: 範囲外の値が入力されている。
	対応: 正しい値を入力してください。
4	・整数を入力してください。 ・有効な数字を入力してください。次の値は無効です:空白スペース、小数、0、+、-
	原因: 整数入力のエディットボックスに数字以外が入力されている。
	対応: 正しい値を入力してください。
	ドットマトリクス設定画面（[Preview]ボタンクリック時）
5	Invalid ????（???? = position、number of dots、dot size、pitch size、start COM/SEG）
	原因: ???? に不正な値が入っている。
	対応: 正しい値を入力してください。
6	Invalid ??? start number or Invalid number of dots （??? = COM、SEG）
	原因: 設定されたStart ??? とドット数、COM/SEG割り付け方向では、ドライバのCOM/SEG数を超える。
	対応: Start ??? とドット数、COM/SEG割り付け方向を、ドライバのCOM/SEG数に合わせてください。
No.	Save as時
7	Some icons/matrixes are out of LCD panel. Remove them? [OK][キャンセル]
	原因: LCDパネルからはみ出したアイコン/ドットマトリクスがある状態で、保存しようとした。
	対応: アイコン/ドットマトリクスをはみ出さないようにするか、[OK]ボタンをクリックしてはみ出したアイコン/ドットマトリクスを削除して保存してください。
	アイコン作成時/ファイル読み込み時
8	The number of the icon is exceeding 4096 pieces.
	原因: アイコンの総数が上限の4096を超えた。
	対応: アイコンの総数が4096個を超えないようにしてください。
9	The number of the icon is exceeding 4096 pieces. Loaded the icons to 4096 pieces.
	原因: アイコンの総数が上限の4096を超えた。
	対応: アイコンの総数が4096個を超えないようにしてください。
	マトリクス作成時/ファイル読み込み時
10	The number of the matrix is exceeding 10 pieces.
	原因: ドットマトリクスの総数が上限の10を超えた。
	対応: ドットマトリクスの総数が10個を超えないようにしてください。
	Save時/変更を保存せず閉じるときのSave時
11	This file was not saved.
	原因: はみ出したアイコン/ドットマトリクスの削除をしなかった。
	対応: アイコン/ドットマトリクスがLCDパネルからはみ出した状態で保存はできません。

LCDUtil17 で表示されるエラーメッセージを示します。

表 10.8.8.2　エラーメッセージ一覧

No.	起動時
1	Cannot open bitmap file. The size of the panel is too large.
	原因：ビットマップファイルの縦横サイズが大きすぎる。
	対応：横1280、縦1024以下のビットマップファイルを使用してください。

11 Quick Reference

メモリマップとトラップテーブル (S1C17コア)

S1C17 Core

メモリマップ

トラップテーブル

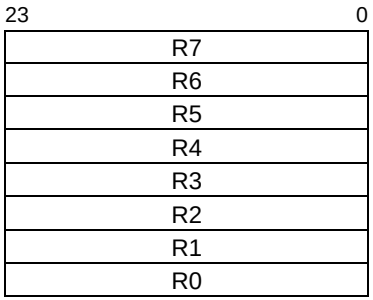
No.		ベクタアドレス
0 (0x00)	リセット	TTBR + 0x00
1 (0x01)	アドレス不整割り込み	TTBR + 0x04
2 (0x02)	NMI	TTBR + 0x08
3 (0x03)	マスク可能な外部割り込み3	TTBR + 0x0c
:	:	:
31 (0x1f)	マスク可能な外部割り込み31	TTBR + 0x7c

TTBR:トラップテーブル先頭アドレス
(0xffff80番地から読み出し可能)

レジスタ (S1C17 コア)

S1C17 Core

汎用レジスタ (8)

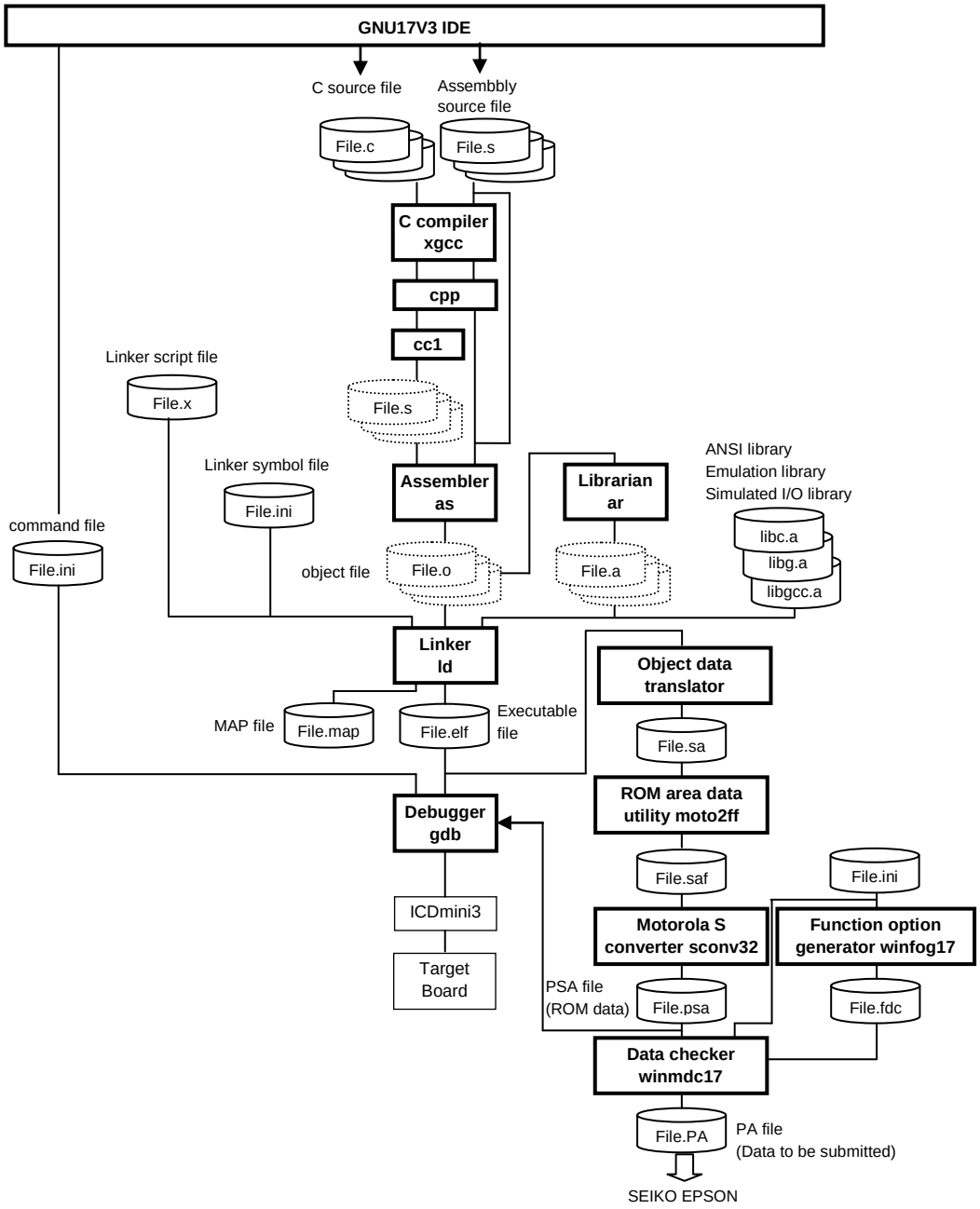


特殊レジスタ (3)

PSR

	7	6	5	4	3	2	1	0
	IL[2:0]			IE	C	V	Z	N
初期値	0	0	0	0	0	0	0	0

IL[2:0]: 割り込みレベル (0~7: 割り込み許可レベル)
IE: 割り込み許可 (1: 許可, 0: 禁止)
Z: ゼロフラグ (1: ゼロ, 0: ゼロ以外)
N: ネガティブフラグ (1: 負, 0: 正)
C: キャリーフラグ (1: キャリー/ボローあり, 0: なし)
V: オーバーフローフラグ (1: オーバーフローあり, 0: なし)



1. プロジェクトの作成
   IDE でプロジェクトを新規作成、または既存プロジェクトをインポートします。

2. ソースの編集
   IDE または汎用のエディタを使用してソースファイルなどのリソースを編集します。

3. ビルド(コンパイル、アセンブル、リンク)
   3-1) IDE のプロジェクトプロパティでビルドオプションとリンカスクリプトを編集します。
   3-2) IDE でビルドを実行します。ビルダーにより C コンパイラ **xgcc**、アセンブラ **as**、およびリンカ **ld** が順次実行され、実行形式のオブジェクトファイル(`.elf`)が生成されます。

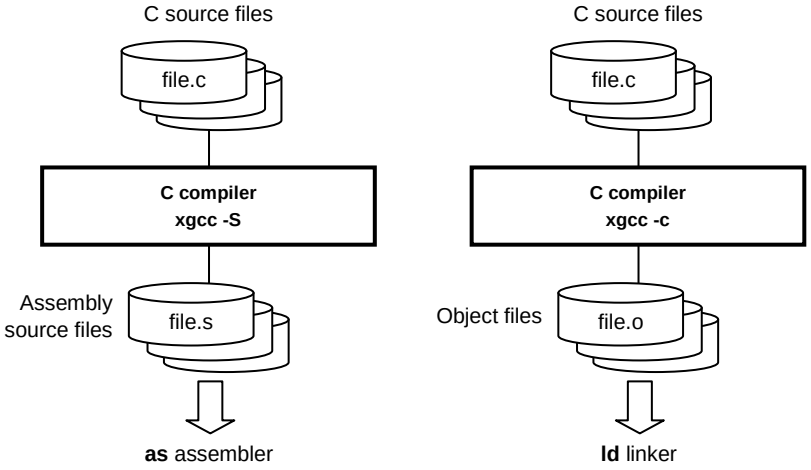
4. デバッグ
   4-1) デバッガ起動時に実行するコマンドファイルを編集します。
   4-2) IDE からデバッガ **gdb** を起動します。
   4-3) IDE を使用してデバッグを行います。

5. PA ファイル(提出用データ)の作成
   プログラムの完成後、提出用データを作成します。
   5-1) objcopy、moto2ff、sconv32 を使用して、PSA ファイル(ROM データ)を生成します。
   5-2) winfog17 を使用して、FDC ファイル(ファンクションオプションドキュメント)を生成します。
   5-3) PSA ファイルと FDC ファイルを **winmdc17** で 1 つの PA ファイルにパックします。
   5-4) できあがった PA ファイルをセイコーエプソンに提出してください。

概要

ANSI C に準拠した C コンパイラで、GNU C コンパイラがベースとなっています。このツールは **cpp.exe** および **cc1.exe** を連続して呼び出し、C ソースファイルをコンパイルして S1C17 Family 用のアセンブリソースファイルを生成します。強力なオプティマイズ能力を持ち、非常にコンパクトなコードを生成します。**xgcc.exe** は、さらにアセンブラ **as.exe** を呼び出してオブジェクトファイルを生成することもできます。

フローチャート



起動コマンド

xgcc <*options*> <*filename*>

<*filename*> C ソースファイル名

例: `xgcc -c -g test.c`

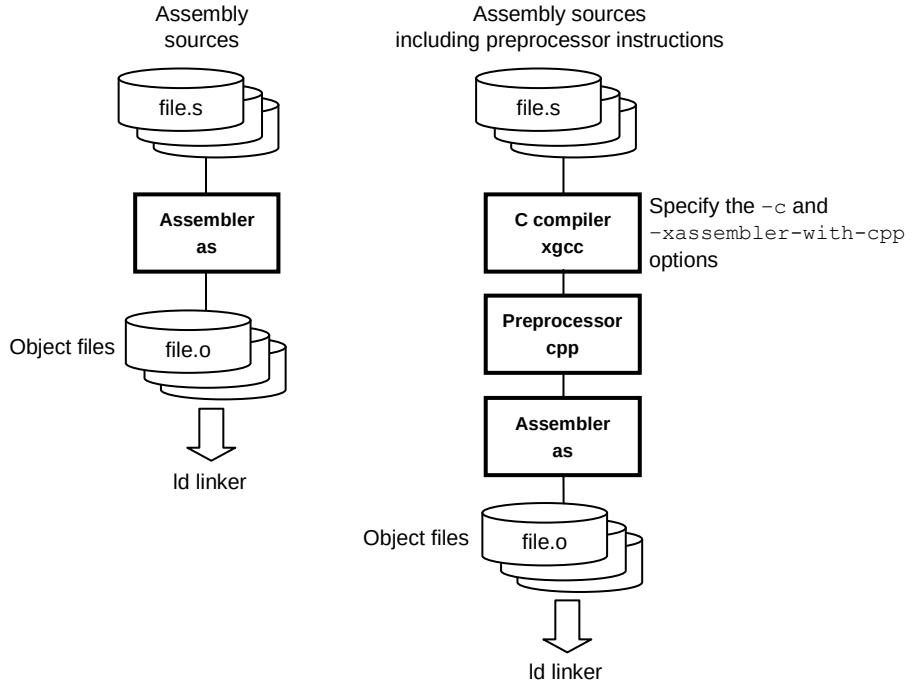
主要なコマンドラインオプション

`-S`	アセンブリコード(.s)の出力
`-c`	リロケータブルオブジェクトファイル(.o)の出力
`-E`	Cプリプロセッサのみの実行
`-B`<*path*>	コンパイラの検索パス指定
`-I`<*path*>	インクルードファイルディレクトリを指定
`-fno-builtin`	ビルトイン関数を無効化
`-D`<*macro*>`[=`<*string*>`]`	マクロ名の定義
`-O0, -O, -O1, -O2, -O3, -Os`	最適化
`-g`	ソースファイルの相対パスを含むデ バッグ情 報の付加。gcc6の場合。
`-gstabs`	ソースファイルの相対パスを含むデ バッグ情 報の付加。gcc4の場合。
`-mpointer16`	16ビット(64KB)データ空間用コードの生成
`-mrelax`	出力コードサイズの最適化
`-Wall`	ワーニングオプションの有効化
`-Werror-implicit-function-declaration`	未宣言の関数をエラー出力
`-xassembler-with-cpp`	Cプリプロセッサの呼び出し
`-Wa,`<*option*>	アセンブラオプションの指定

概要

C コンパイラが出力するアセンブリソースファイルをアセンブルし、ソースファイルのニーモニックを S1C17 のオブジェクト(機械語)コードに変換します。**as.exe** は **xgcc.exe** から **cpp.exe** に続けて実行させることも可能です。これにより、アセンブリソースファイル内でプリプロセッサ擬似命令を使用することも許されます。結果はリンクおよびライブラリ化が可能なオブジェクトファイルとして出力されます。

フローチャート



起動コマンド

as <*options*> <*filename*>

<*filename*> アセンブリソースファイル名

例: `as -o test.o -adhl test.sAssemblerasfile.`

主要なコマンドラインオプション

`-o`<*filename*>	出力ファイル名の指定
`-a[`<*suboption*>`]`	アセンブリリストファイルの出力例: `-adhl` (ハイレベルな出力とデバッグ擬似命令の削除)
`--gstabs`	ソースファイルの相対パスを含むデバッグ情報の付加
`-mpointer16`	16ビットポインタモードの指定

主要なプリプロセッサ擬似命令

`#include`	ファイルの挿入
`#define`	文字列および数値の定義 / マクロ定義
`#if – #else – #endif`	条件アセンブル (**xgcc**の`-c -xassembler-with-cpp`オプション指定時に使用可能)

主要なアセンブラ擬似命令

`.text`	.textセクションを宣言
`.section .data`	.dataセクションを宣言
`.section .rodata`	.rodataセクションを宣言
`.section .bss`	.bssセクションを宣言
`.long` <*data*>	4バイトデータを定義
`.short` <*data*>	2バイトデータを定義
`.byte` <*data*>	1バイトデータを定義
`.ascii` <*string*>	ASCII文字列を定義
`.space` <*length*>	空白領域(0x0)を定義
`.zero` <*length*>	空白領域(0x0)を定義
`.align` <*value*>	指定境界アドレスにアライメント
`.global` <*symbol*>	シンボルのグローバル宣言
`.set` <*symbol*>`,`<*address*>	シンボルに絶対アドレスを定義

エラー/ワーニングメッセージ

エラーメッセージ

Error: Unrecognized opcode: *'XXXXX'*	*XXXXX*は未定義のオペコードです。
Error: junk at end of line: *'XXXXX'*	オペランド書式のエラーです。
Error: *XXXXXX*: invalid register name	使用できないレジスタを指定しています。

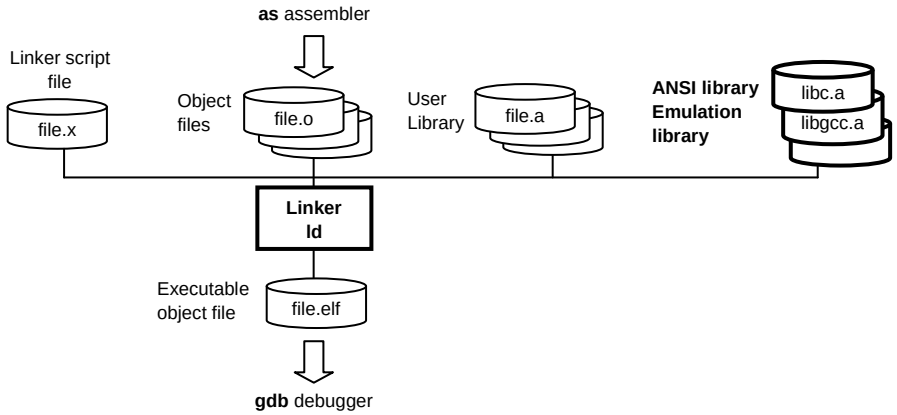
ワーニングメッセージ

Warning: Unrecognized .section attribute: want a, w, x	セクションの属性がa、w、x以外です。
Warning: Bignum truncated to *AAA* bytes	定数宣言(`.long`、`.int`など)のサイズが最大値を超えています。値を*AAA*バイトサイズに補正します。 例:`0x100000012` → `0x12`
Warning: Value *XXXX* truncated to *AAA*	定数宣言の値が最大値を超えています。値を*AAA*に補正します。 例:`.byte 0x10000012` → `.byte 0xff`
Warning: operand out of range (*XXXXXX*: *XXX* not between *AAA* and *BBB*)	オペランドで指定されている値は有効範囲外です。

概要

C コンパイラおよびアセンブラによって生成したオブジェクトコードのメモリロケーションを決定し、実行可能なオブジェクトコードを生成します。複数のオブジェクトファイルやライブラリも、このツールによって 1 つにまとめられます。

フローチャート



起動コマンド

ld <*options*> <*filename*>

<*filename*> リンクするオブジェクトファイルおよびライブラリファイル

例: `ld -o sample.elf boot.o sample.o ..¥lib¥24bit¥libc.a ..¥lib¥24bit¥libgcc.a ..¥lib¥24bit¥libc.a`

主要なコマンドラインオプション

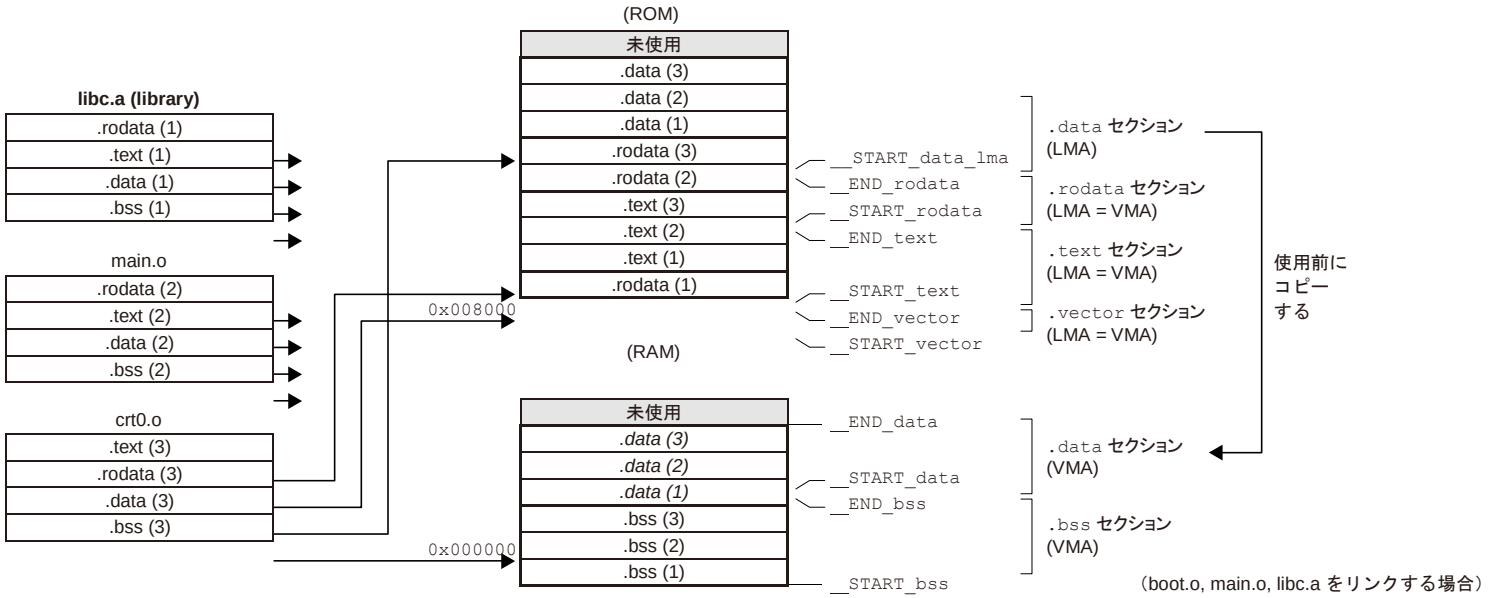
`-o` <*filename*>	出力ファイル名の指定
`-T` <*filename*>	リンカスクリプトファイルの読み込み
`-M`	リンクマップのstdout出力
`-Map` <*filename*>	リンクマップのファイル出力
`-N`	データセグメントのアライメントチェックを禁止
`-R`	リンカシンボルファイルの読み込み
`--relax`	出力コードサイズの最適化

エラーメッセージ

warning: out of range error.	シンボルのアドレスが16ビットアドレス(-mpointer16指定時),24ビットアドレス空間を超えています。
Error: The offset value of a symbol is over 24bit.	シンボルのアドレスが24ビットアドレス空間を超えています。
Error: section *XXX* is not within 16bit address.	*XXX*セクションのアドレスが16ビットアドレス空間を越えています。
Error: section *XXX* is not within 24bit address.	*XXX*セクションのアドレスが24ビットアドレス空間を越えています。
Error: Input object file <objectfile> [included from <archivefile>] is not for C17.	オブジェクトファイルはC17用ではありません
Error: Input object file <objectfile> is not 16bit nor 24bit address mode.	オブジェクトファイルが16ビットモードでも24ビットモードでもありません
Error: Cannot link 16bit object <objectfile16> [included from <archivefile16>] with 24bit object <objectfile24> [included from <archivefile24>]	16ビットポインタモードで作成されたオブジェクトファイルと、24ビットポインタモードで作成されたオブジェクトファイルはリンクできません

IDE が作成するデフォルトリンカスクリプトファイル

```
OUTPUT_FORMAT("elf32-c17")
OUTPUT_ARCH(c17)
ENTRY(_start)
SEARCH_DIR(.);
MEMORY
{
        iram        : ORIGIN = 0,           LENGTH = 32K
        irom        : ORIGIN = 0x8000,      LENGTH = 4064K
}
SECTIONS
{
        .bss (NOLOAD) :
        {
                PROVIDE (__START_bss = .) ;
                *(.bss)
                *(.bss.*)
                *(COMMON)
                PROVIDE (__END_bss = .) ;
        } > iram
        .vector :
        {
                PROVIDE (__START_vector = .) ;
                KEEP (*crt0.o(.rodata))
                PROVIDE (__END_vector = .) ;
        } > irom
        .text :
        {
                PROVIDE (__START_text = .) ;
                *(.text.*)
                *(.text)
                PROVIDE (__END_text = .) ;
        } > irom
        .data :
        {
                PROVIDE (__START_data = .) ;
                *(.data)
                *(.data.*)
                PROVIDE (__END_data = .) ;
        } > iram AT > irom
        .rodata :
        {
                PROVIDE (__START_rodata = .) ;
                *(EXCLUDE_FILE (*crt0.o ) .rodata)
                *(.rodata.*)
                PROVIDE (__END_rodata = .) ;
        } > irom
        PROVIDE (__START_data_lma = LOADADDR(.data));
        PROVIDE (__END_data_lma = LOADADDR(.data) + SIZEOF (.data));
        PROVIDE (__START_stack = 0x0007C0);
}
```

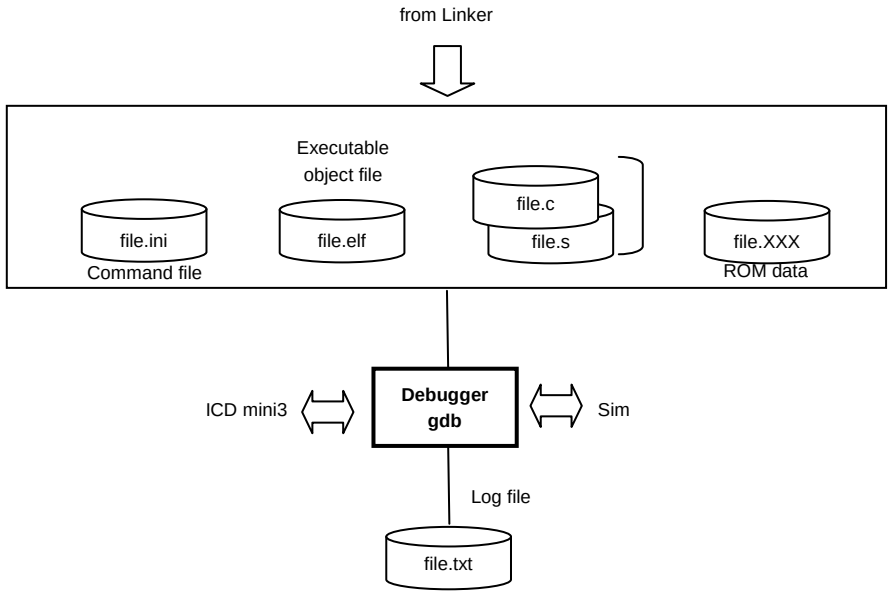


（boot.o, main.o, libc.a をリンクする場合）

概要

ICD を制御してソースレベルのデバッグを行うソフトウェアです。特別なツールを使用せずにパソコン上でデバッグを行うシミュレーション機能も持っています。

フローチャート



起動時オプション

gdb[<起動オプション>]

IDE 上で実行する場合、起動時オプションでコマンドファイルを指定します。
例:`gdb -x gdbmini3.ini`

コマンドファイル

例:

`c17 model_path c:/EPSON/GNU17V3/mcu_model`	機種情報ファイルのディレクトリを指定
`c17 model 17W23@NOVCCIN`	機種名を指定(電圧レベル 3.3V)
`target icd icdmini3`	ターゲットの接続
`load`	プログラムのロード
`thbreak main`	ハードウェア PC ブレークポイントの設定

デバッグコマンド

メモリ操作

c17 fb *addr1 addr2 data*	領域のフィル（8ビット単位）	ICD Mini/SIM
c17 fh *addr1 addr2 data*	領域のフィル（16ビット単位）	ICD Mini/SIM
c17 fw *addr1 addr2 data*	領域のフィル（32ビット単位）	ICD Mini/SIM
x /[*length*]**b** [*addr*]	メモリダンプ（8ビット単位）	ICD Mini/SIM
x /[*length*]**h** [*addr*]	メモリダンプ（16ビット単位）	ICD Mini/SIM
x /[*length*]**w** [*addr*]	メモリダンプ（32ビット単位）	ICD Mini/SIM
set {char} *addr=data*	データ入力（8ビット単位）	ICD Mini/SIM
set {short} *addr=data*	データ入力（16ビット単位）	ICD Mini/SIM
set {long} *addr=data*	データ入力（32ビット単位）	ICD Mini/SIM
c17 mvb *addr1 addr2 addr3*	領域のコピー（8ビット単位）	ICD Mini/SIM
c17 mvh *addr1 addr2 addr3*	領域のコピー（16ビット単位）	ICD Mini/SIM
c17 mvw *addr1 addr2 addr3*	領域のコピー（32ビット単位）	ICD Mini/SIM
c17 df *addr1 addr2 type file* [*append*]	メモリ内容の保存	ICD Mini/SIM

レジスタ操作

info reg [*register*]	レジスタ表示	ICD Mini/SIM
set $*register=data*	レジスタ変更	ICD Mini/SIM

プログラム実行

continue [*Ignore*]	連続実行	ICD Mini/SIM
until *addr*	テンポラリブレーク付き連続実行	ICD Mini/SIM
step [*count*]	ステップ実行(行単位)	ICD Mini/SIM
stepi [*count*]	ステップ実行(ニーモニック単位)	ICD Mini/SIM
next [*count*]	スキップ付きステップ実行(行単位)	ICD Mini/SIM
nexti [*count*]	スキップ付きステップ実行(ニーモニック単位)	ICD Mini/SIM
finish	関数終了	ICD Mini/SIM

CPU リセット

c17 rst	リセット(reset.gdbを実行)	ICD Mini/SIM
c17 rstt	ターゲットのリセット	ICD Mini

割り込み

c17 int [*intNo. level*]	割り込み発生	SIM
c17 intclear [*intNo.*]	割り込み解除	SIM

ブレーク

break [*addr*]	ソフトウェアPCブレーク設定	ICD Mini/SIM
tbreak [*addr*]	テンポラリソフトウェアPCブレーク設定	ICD Mini/SIM
hbreak [*addr*]	ハードウェアPCブレーク設定	ICD Mini/SIM
thbreak [*addr*]	テンポラリハードウェアPCブレーク設定	ICD Mini/SIM
delete [*breakNo.*]	ブレーク番号によるブレークの解除	ICD Mini/SIM
clear *addr*	ブレーク位置によるブレークの解除	ICD Mini/SIM
enable [*breakNo.*]	ブレークポイントの有効化	ICD Mini/SIM
disable [*breakNo.*]	ブレークポイントの無効化	ICD Mini/SIM
ignore *breakNo. count*	回数指定付きブレークの無効化	ICD Mini/SIM
info breakpoints	ブレークポイントリストの表示	ICD Mini/SIM
commands	ブレーク後に実行するコマンドの設定	ICD Mini/Sim

シンボル情報

info locals	ローカルシンボル表示	ICD Mini/SIM
info var	グローバルシンボル表示	ICD Mini/SIM
print *symbol*[=*value*]	シンボル値の変更	ICD Mini/SIM

ファイル読み込み

file *file*	デバッグ情報の読み込み	ICD Mini/SIM
load [*file*]	プログラムのロード	ICD Mini/SIM

トレース

c17 tm on/off *mode* [*file*]	トレースモードの設定	SIM

デバッグコマンド

その他

set output-rad x	変数表示形式の変更	ICD Mini/SIM
set logging on/off	ログ出力の設定	ICD Mini/SIM
source *file*	コマンドファイルの実行	ICD Mini/SIM
target *type*	ターゲットの接続	ICD Mini/SIM
detach	ターゲットの切断	ICD Mini/SIM
pwd	カレントディレクトリの表示	ICD Mini/SIM
cd *directory*	カレントディレクトリの変更	ICD Mini/SIM
c17 ttbr *addr*	TTBR設定	SIM
c17 cpu *type*	CPUタイプの設定	SIM
c17 chgclkmd 0/1	DLCK切替モード	ICD Mini
c17 pwul	Flashセキュリティ・パスワードの解除	ICD Mini
c17 help [*command/groupNo.*]	ヘルプ	ICD Mini/SIM
c17 model_path *addr*	機種別情報ファイルのディレクトリの設定	ICD Mini/SIM
c17 model *name*	MCUモデル名の設定	ICD Mini/SIM
quit	デバッガ終了	ICD Mini/SIM

ステータス/エラーメッセージ

ステータスメッセージ

Breakpoint *#*, *function* at *file:line*	設定したブレークポイントでブレーク
Illegal instruction.	シミュレータモードで不当命令の実行によりブレーク
Illegal delayed instruction.	シミュレータモードで不当な遅延命令の実行によりブレーク
Break by key break.	[中断]ボタンによる強制ブレーク(シミュレータモード)
Break by key break. Program received signal SIGINT, Interrupt.	[中断]ボタンによる強制ブレーク(ICD Miniモード)

エラーメッセージ

Address is 24bit over.	指定アドレスは、24bitを超えています。S1C17のアドレスは、最大24bit(0xFFFFFF)です。
Address(0x%lx) is ext or delayed instruction	指定アドレスは、ext命令または遅延命令のため設定できません。
C17 command error, command is not supported in present mode.	指定したコマンドは,現在のモード(ICD or SIMモードまたはどちらでもない)では対応していません。
C17 command error, invalid command.	コマンドに誤りがあります。
C17 command error, invalid parameter.	指定したコマンドの引数に誤りがあります。
C17 command error, number of parameter.	指定したコマンドの引数の数に誤りがあります。
C17 command error, start address > end address.	開始アドレスが終了アドレスを超えて指定されています。
Cannot set hard pc break.	指定したアドレスにハードブレークは設定できません。
Cannot set hard pc break any more.	ハードウェアPCブレークポイントの設定数が制限を超えました。
Cannot set soft pc break.	指定したアドレスにソフトブレークは設定できません。
Cannot set soft pc break any more.	ソフトウェアPCブレークポイントの設定数が制限を超えました(Max. 200)。
Cannot write file	ファイルへの書き込みができません。
command result error!	未定義のコマンド実行結果がエラーとなりました。
icdmini3 dll open failure.	ICD mini Ver3 との接続に失敗しました。

浮動小数点演算関数

Double 型演算

__adddf3	加算	x ← a + b
__subdf3	減算	x ← a - b
__muldf3	乗算	x ← a * b
__divdf3	除算	x ← a / b
__negdf2	符号反転	x ← -a

Float 型演算

__addsf3	加算	x ← a + b
__subsf3	減算	x ← a - b
__mulsf3	乗算	x ← a * b
__divsf3	除算	x ← a / b
__negsf2	符号反転	x ← -a

型変換

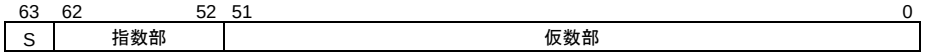
__fixunsdfsi	double → unsigned long	x ← a
__fixdfsi	double → long	x ← a
__floatsidf	long → double	x ← a
__fixunssfsi	float → unsigned long	x ← a
__fixsfsi	float → long	x ← a
__floatsisf	long → float	x ← a
__truncdfsf2	double → float	x ← a
__extendsfdf2	float → double	x ← a

比較

__**df2	double type	Changes %psr and x by a - b **=eq, ne, gt, ge, lt, le
__**sf2	float type	Changes %psr and x by a - %13 **=eq, ne, gt, ge, lt, le

浮動小数点データフォーマット

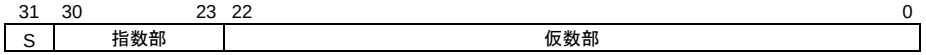
Double 型データ



Double 型有効範囲

+0:	0.0e+0 0x00000000 00000000
-0:	-0.0e+0 0x80000000 00000000
最大正規化数:	1.79769e+308 0x7fefffff ffffffff
最小正規化数:	2.22507e-308 0x00100000 00000000
最大非正規化数:	2.22507e-308 0x000fffff ffffffff
最小非正規化数:	4.94065e-324 0x00000000 00000001
無限大:	0x7ff00000 00000000
-無限大:	0xfff00000 00000000

Float 型データ



Float 型有効範囲

+0:	0.0e+0f 0x00000000
-0:	-0.0e+0f 0x80000000
最大正規化数:	3.40282e+38f 0x7f7fffff
最小正規化数:	1.17549e-38f 0x00800000
最大非正規化数:	1.17549e-38f 0x007fffff
最小非正規化数:	1.40129e-45f 0x00000001
無限大:	0x7f800000
-無限大:	0xff800000

整数演算関数

整数演算

__divsi3	符号付き32ビット整数除算	x ← a / b
__modsi3	符号付き32ビット整数剰余演算	x ← a % b
__udivsi3	符号なし32ビット整数除算	x ← a / b
__umodsi3	符号なし32ビット整数剰余演算	x ← a % b
__mulsi3	32ビット乗算	x ← a * b
__divhi3	符号付き16ビット整数除算	x ← a / b
__modhi3	符号付き16ビット整数剰余演算	x ← a % b
__udivhi3	符号なし16ビット整数除算	x ← a / b
__umodhi3	符号なし16ビット整数剰余演算	x ← a % b
__mulhi3	16ビット乗算	x ← a * b

整数シフト

__ashlsi3	32ビット算術左シフト	x ← a << bビット
__ashrsi3	32ビット算術右シフト	x ← a >> bビット
__lshrsi3	32ビット論理右シフト	x ← a >> bビット
__ashlhi3	16ビット算術左シフト	x ← a << bビット
__ashrhi3	16ビット算術右シフト	x ← a >> bビット
__lshrhi3	16ビット論理右シフト	x ← a >> bビット

整数比較

__cmpsi2	比較(long)	x ← 2 | 1 | 0
__ucmpsi2	比較(unsigned long)	x ← 2 | 1 | 0

long long 型演算関数

long long 型演算

__muldi3	符号付き64ビット乗算	x ← a * b
__divdi3	符号付き64ビット除算	x ← a / b
__udivdi3	符号なし64ビット除算	x ← a / b
__moddi3	符号付き64ビット剰余演算	x ← a % b
__umoddi3	符号なし64ビット剰余演算	x ← a % b
__negdi2	符号反転	x ← -a

long long 型シフト

__lshrdi3	64ビット論理右シフト	x ← a >> bビット
__ashldi3	64ビット算術左シフト	x ← a << bビット
__ashrdi3	64ビット算術右シフト	x ← a >> bビット

型変換

__fixunsdfdi	double → unsigned long long	x ← a
__fixdfdi	double → long long	x ← a
__floatdidf	long long → double	x ← a
__fixunssfdi	float → unsigned long long	x ← a
__fixsfdi	float → long long	x ← a
__floatdisf	long long → float	x ← a

long long 型比較

__cmpdi2	比較(long long)	x ← 2 | 1 | 0
__ucmpdi2	比較(unsigned long long)	x ← 2 | 1 | 0

入出力関数(ヘッダファイル: stdio.h)

fread( )	size_t fread(void *ptr, size_t size, size_t count, FILE *stream);	*1, *2
fwrite( )	size_t fwrite(const void *ptr, size_t size, size_t count, FILE *stream);	*1, *2
fgetc( )	int fgetc(FILE *stream);	*2
getc( )	int getc(FILE *stream);	*1, *2
getchar( )	int getchar(void);	*1, *2
ungetc( )	int ungetc(int c, FILE *stream);	*1
fgets( )	char *fgets(char *s, int n, FILE *stream);	*1, *2
gets( )	char *gets(char *s);	*1, *2
fputc( )	int fputc(int c, FILE *stream);	*2
putc( )	int putc(int c, FILE *stream);	*1, *2
putchar( )	int putchar(int c);	*1, *2
fputs( )	int fputs(char *s, FILE *stream);	*1, *2
puts( )	int puts(char *s);	*1, *2
perror( )	void perror(const char *s);	*1, *2
fscanf( )	int fscanf(FILE *stream, const char *format, ...);	*1, *2
scanf( )	int scanf(const char *format, ...);	*1, *2
sscanf( )	int sscanf(const char *s, const char *format, ...);	*1, *2
fprintf( )	int fprintf(FILE *stream, const char *format, ...);	*1, *2
printf( )	int printf(const char *format, ...);	*1, *2
sprintf( )	int sprintf(char *s, const char *format, ...);	*1, *2
vfprintf( )	int vfprintf(FILE *stream, const char *format, va_list arg);	*1, *2
vprintf( )	int vprintf(const char *format, va_list arg);	*1, *2
vsprintf( )	int vsprintf(char *s, const char *format, va_list arg);	

ユーティリティ関数(ヘッダファイル: stdlib.h)

malloc( )	void *malloc(size_t size);	*1
calloc( )	void *calloc(size_t elt_count, size_t elt_size);	*1
free( )	void free(void *ptr);	*1
realloc( )	void *realloc(void *ptr, size_t size);	*1
exit( )	void exit(int status);	
abort( )	void abort(void);	
bsearch( )	void *bsearch(const void *key, const void *base, size_t count,size_t size, int (*compare)(const void *, const void *));	
qsort( )	void qsort(void *base, size_t count, size_t size,int (*compare)(const void *, const void *));	
abs( )	int abs(int x);	
labs( )	long labs(long x);	
div( )	div_t div(int n, int d);	*1
ldiv( )	ldiv_t ldiv(long n, long d);	*1
rand( )	int rand(void);	
srand( )	void srand(unsigned int seed);	
atol( )	long atol(const char *str);	
atoi( )	int atoi(const char *str);	*1
atof( )	double atof(const char *str);	*1
strtod( )	double strtod(const char *str, char **ptr);	*1
strtol( )	long strtol(const char *str, char **ptr, int base);	*1
strtoul( )	unsigned long strtoul(const char *str, char **ptr, int base);	*1

日付と時刻関数(ヘッダファイル: time.h)

gmtime( )	struct tm *gmtime(const time_t *t);	
mktime( )	time_t mktime(struct tm *tmptr);	
time( )	time_t time(time_t *tptr);	*1

非局所分岐関数(ヘッダファイル: setjmp.h)

setjmp( )	int setjmp(jmp_buf env);
longjmp( )	void longjmp(jmp_buf env, int status);

*1 グローバル変数の宣言と初期化が必要です。
*2 下位レベル関数と入出力バッファの定義が必要です。

数学関数(ヘッダファイル: math.h, errno.h, float.h, limits.h)

fabs( )	double fabs(double x);	*1
ceil( )	double ceil(double x);	*1
floor( )	double floor(double x);	*1
fmod( )	double fmod(double x, double y);	*1
exp( )	double exp(double x);	*1
log( )	double log(double x);	*1
log10( )	double log10(double x);	*1
frexp( )	double frexp(double x, int *nptr);	*1
ldexp( )	double ldexp(double x, int n);	*1
modf( )	double modf(double x, double *nptr);	*1
pow( )	double pow(double x, double y);	*1
sqrt( )	double sqrt(double x);	*1
sin( )	double sin(double x);	*1
cos( )	double cos(double x);	*1
tan( )	double tan(double x);	*1
asin( )	double asin(double x);	*1
acos( )	double acos(double x);	*1
atan( )	double atan(double x);	
atan2( )	double atan2(double y, double x);	*1
sinh( )	double sinh(double x);	*1
cosh( )	double cosh(double x);	*1
tanh( )	double tanh(double x);	

文字種判定/変換関数(ヘッダファイル: ctype.h)

isalnum( )	int isalnum(int c);
isalpha( )	int isalpha(int c);
iscntrl( )	int iscntrl(int c);
isdigit( )	int isdigit(int c);
isgraph( )	int isgraph(int c);
islower( )	int islower(int c);
isprint( )	int isprint(int c);
ispunct( )	int ispunct(int c);
isspace( )	int isspace(int c);
isupper( )	int isupper(int c);
isxdigit( )	int isxdigit(int c);
tolower( )	int tolower(int c);
toupper( )	int toupper(int c);

可変引数マクロ(ヘッダファイル: stdarg.h)

va_start( )	void va_start(va_list ap, type lastarg);
va_arg( )	type va_arg(va_list ap, type);
va_end( )	void va_end(va_list ap);

*1 グローバル変数の宣言と初期化が必要です。

文字関数(ヘッダファイル: string.h)

memchr( )	void *memchr(const void *s, int c, size_t n);
memcmp( )	int memcmp(const void *s1, const void *s2, size_t n);
memcpy( )	void *memcpy(void *s1, const void *s2, size_t n);
memmove( )	void *memmove(void *s1, const void *s2, size_t n);
memset( )	void *memset(void *s, int c, size_t n);
strcat( )	char *strcat(char *s1, const char *s2);
strchr( )	char *strchr(const char *s, int c);
strcmp( )	int strcmp(const char *s1, const char *s2);
strcpy( )	char *strcpy(char *s1, const char *s2);
strcspn( )	size_t strcspn(const char *s1, const char *s2);
strerror( )	char *strerror(int code);
strlen( )	size_t strlen(const char *s);
strncat( )	char *strncat(char *s1, const char *s2, size_t n);
strncmp( )	int strncmp(const char *s1, const char *s2, size_t n);
strncpy( )	char *strncpy(char *s1, const char *s2, size_t n);
strpbrk( )	char *strpbrk(const char *s1, const char *s2);
strrchr( )	char *strrchr(const char *str, int c);
strspn( )	size_t strspn(const char *s1, const char *s2);
strstr( )	char *strstr(const char *s1, const char *s2);
strtok( )	char *strtok(char *s1, const char *s2);

*1 グローバル変数の宣言と初期化

FILE _iob[FOPEN_MAX+1];	_iob[N]._flg=_UGETN; _iob[N]._buf=0; _iob[N]._fd=N; (N=0: stdin, N=1: stdout, N=2: stderr)
FILE *stdin;	stdin=&_iob[0];
FILE *stdout;	stdout=&_iob[1];
FILE *stderr;	stderr=&_iob[2];
int errno;	errno=0;
unsigned int seed;	seed=1;
time_t gm_sec;	gm_sec=-1;

*2 下位レベル関数の定義

read( )	int read(int fd, char *buf, int nbytes); unsigned char READ_BUF[65]; (他の名称も可) unsigned char READ_EOF;
write( )	int write(int fd, char *buf, int nbytes); unsigned char WRITE_BUF[65]; (他の名称も可)

命令一覧中のシンボル

レジスタ/レジスタデータ

%rd, rd:	ディスティネーションとなる汎用レジスタ(R0~R7)、またはレジスタの内容
%rs, rs:	ソースとなる汎用レジスタ(R0~R7)、またはレジスタの内容
%rb, rb:	レジスタ間接アドレッシングのベースレジスタを保持している汎用レジスタ(R0~R7)、またはレジスタの内容
%sp, sp:	スタックポインタ(SP)またはその内容
%pc, pc:	プログラムカウンタ(PC)またはその内容

メモリ/アドレス/メモリデータ

[%rb], [%sp]:	レジスタ間接アドレッシング指定
[%rb]+, [%sp]+:	ポストインクリメント付きレジスタ間接アドレッシング指定
[%rb]-, [%sp]-:	ポストデクリメント付きレジスタ間接アドレッシング指定
-[%rb], -[%sp]:	プリデクリメント付きレジスタ間接アドレッシング指定
[%sp+immX]:	ディスプレースメント付きレジスタ間接アドレッシング指定
[imm7]:	即値によるメモリアドレス指定
B[XXX]:	XXXで指定されるアドレス、またはそのアドレスにストアされているバイトデータ
W[XXX]:	XXXで指定される16ビット境界アドレス、またはそのアドレスにストアされている ワードデータ
A[XXX]:	XXXで指定される32ビット境界アドレス、またはそのアドレスにストアされている 24ビットデータもしくは32ビットデータ

即値

immX:	符号なしXビット即値
signX:	符号付きXビット即値

シンボル/ラベル

Symbol:	アドレスを示すシンボル
Label:	分岐先ラベル

ビットフィールド

(X):	データのビットX
(X:Y):	ビットXからビットYまでのビットフィールド
{X, Y···}:	ビット構成

機能

←:	右側の内容が左側の項目にロードされることを示します。
+:	加算
-:	減算
&:	論理積
|:	論理和
^:	排他的論理和
!:	論理否定

フラグ

IL:	割り込みレベル
IE:	割り込みイネーブルフラグ
C:	キャリーフラグ
V:	オーバーフローフラグ
Z:	ゼロフラグ
N:	ネガティブフラグ
−:	変更なし
↔:	セット(1)、リセット(0)、または変更なし
1:	セット(1)
0:	リセット(0)

D

○:	ディレイド命令として使用可能なことを示します。
−:	ディレイド命令として使用できないことを示します。

注意

- **命令一覧には S1C17 命令セットの基本命令の他に、拡張命令(s で始まる命令および xor を除く x で始まる命令)も含まれています。**
- "*Italic*" の文字で記載されている命令は、上位互換の拡張命令が用意されていることを示します。

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
符号付き8ビット データ転送	ld.b	%rd, %rs	rd(7:0)←rs(7:0), rd(15:8)←rs(7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)+1	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)-1	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(7:0)←B[sp+imm7], rd(15:8)←B[sp+imm7](7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [imm7]	rd(7:0)←B[imm7], rd(15:8)←B[imm7](7), rd(23:16)←0	–	–	–	–	–	–	○
		[%rb], %rs	B[rb]←rs(7:0)	–	–	–	–	–	–	○
		[%rb]+, %rs	B[rb]←rs(7:0), rb(23:0)←rb(23:0)+1	–	–	–	–	–	–	○
		[%rb]-, %rs	B[rb]←rs(7:0), rb(23:0)←rb(23:0)-1	–	–	–	–	–	–	○
		-[%rb], %rs	rb(23:0)←rb(23:0)-1, B[rb]←rs(7:0)	–	–	–	–	–	–	○
		[%sp+imm7], %rs	B[sp+imm7]←rs(7:0)	–	–	–	–	–	–	○
		[imm7], %rs	B[imm7]←rs(7:0)	–	–	–	–	–	–	○
	sld.b	%rd, [%sp+imm20]	%rd←B[%sp+imm20](符号拡張)	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←B[imm20](符号拡張)	–	–	–	–	–	–	–
		[%sp+imm20], %rs	B[%sp+imm20]←%rs(7:0)	–	–	–	–	–	–	–
		[imm20], %rs	B[imm20]←%rs(7:0)	–	–	–	–	–	–	–
	xld.b	%rd, [%sp+imm24]	%rd←B[%sp+imm24](符号拡張)	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←B[imm24](符号拡張)	–	–	–	–	–	–	–
		[%sp+imm24], %rs	B[%sp+imm24]←%rs(7:0)	–	–	–	–	–	–	–
		[imm24], %rs	B[imm24]←%rs(7:0)	–	–	–	–	–	–	–
符号なし8ビット データ転送	ld.ub	%rd, %rs	rd(7:0)←rs(7:0), rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←rb(23:0)+1	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←rb(23:0)-1	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(7:0)←B[sp+imm7], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [imm7]	rd(7:0)←B[imm7], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
	sld.ub	%rd, [%sp+imm20]	%rd←B[%sp+imm20](ゼロ拡張)	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←B[imm20](ゼロ拡張)	–	–	–	–	–	–	–
	xld.ub	%rd, [%sp+imm24]	%rd←B[%sp+imm24](ゼロ拡張)	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←B[imm24](ゼロ拡張)	–	–	–	–	–	–	–

備　　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
16ビット データ転送	ld	%rd, %rs	rd(15:0)←rs(15:0), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, sign7	rd(6:0)←sign7(6:0), rd(15:7)←sign7(6), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(15:0)←W[rb], rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)+2	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)-2	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-2, rd(15:0)←W[rb], rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(15:0)←W[sp+imm7], rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [imm7]	rd(15:0)←W[imm7], rd(23:16)←0	–	–	–	–	–	–	○
		[%rb], %rs	W[rb]←rs(15:0)	–	–	–	–	–	–	○
		[%rb]+, %rs	W[rb]←rs(15:0), rb(23:0)←rb(23:0)+2	–	–	–	–	–	–	○
		[%rb]-, %rs	W[rb]←rs(15:0), rb(23:0)←rb(23:0)-2	–	–	–	–	–	–	○
		-[%rb], %rs	rb(23:0)←rb(23:0)-2, W[rb]←rs(15:0)	–	–	–	–	–	–	○
		[%sp+imm7], %rs	W[sp+imm7]←rs(15:0)	–	–	–	–	–	–	○
		[imm7], %rs	W[imm7]←rs(15:0)	–	–	–	–	–	–	○
	sld	%rd, imm16	%rd←imm16	–	–	–	–	–	–	–
		%rd, symbol±imm16	%rd←symbol±imm16(15:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm20]	%rd←W[%sp+imm20]	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←W[imm20]	–	–	–	–	–	–	–
		[%sp+imm20], %rs	W[%sp+imm20]←%rs(15:0)	–	–	–	–	–	–	–
		[imm20], %rs	W[imm20]←%rs(15:0)	–	–	–	–	–	–	–
	xld	%rd, imm16	%rd←imm16	–	–	–	–	–	–	–
		%rd, symbol±imm16	%rd←symbol±imm16(15:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm24]	%rd←W[%sp+imm24]	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←W[imm24]	–	–	–	–	–	–	–
		[%sp+imm24], %rs	W[%sp+imm24]←%rs(15:0)	–	–	–	–	–	–	–
		[imm24], %rs	W[imm24]←%rs(15:0)	–	–	–	–	–	–	–
32ビット データ転送	ld.a	%rd, %rs	rd(23:0)←rs(23:0)	–	–	–	–	–	–	○
		%rd, imm7	rd(6:0)←imm7(6:0), rd(23:7)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24)	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24), rb(23:0)←rb(23:0)+4	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24), rb(23:0)←rb(23:0)-4	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-4, rd(23:0)←A[rb](23:0), 無視←A[rb](31:24)	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(23:0)←A[sp+imm7](23:0), 無視←A[sp+imm7](31:24)	–	–	–	–	–	–	○
		%rd, [imm7]	rd(23:0)←A[imm7](23:0), 無視←A[imm7](31:24)	–	–	–	–	–	–	○

備　　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
32ビット データ転送	ld.a	[%rb], %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0	–	–	–	–	–	–	○
		[%rb]+, %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)+4	–	–	–	–	–	–	○
		[%rb]-, %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)-4	–	–	–	–	–	–	○
		-[%rb], %rs	rb(23:0)←rb(23:0)-4, A[rb](23:0)←rs(23:0), A[rb](31:24)←0	–	–	–	–	–	–	○
		[%sp+imm7], %rs	A[sp+imm7](23:0)←rs(23:0), A[sp+imm7](31:24)←0	–	–	–	–	–	–	○
		[imm7], %rs	A[imm7](23:0)←rs(23:0), A[imm7](31:24)←0	–	–	–	–	–	–	○
		%rd, %sp	rd(23:2)←sp(23:2), rd(1:0)←0	–	–	–	–	–	–	○
		%rd, %pc	rd(23:0)←pc(23:0)+2	–	–	–	–	–	–	○
		%rd, [%sp]	rd(23:0)←A[sp](23:0), 無視←A[sp](31:24)	–	–	–	–	–	–	○
		%rd, [%sp]+	rd(23:0)←A[sp](23:0), 無視←A[sp](31:24), sp(23:0)←sp(23:0)+4	–	–	–	–	–	–	○
		%rd, [%sp]-	rd(23:0)←A[sp](23:0), 無視←A[sp](31:24), sp(23:0)←sp(23:0)-4	–	–	–	–	–	–	○
		%rd, -[%sp]	sp(23:0)←sp(23:0)-4, rd(23:0)←A[sp](23:0), 無視←A[sp](31:24)	–	–	–	–	–	–	○
		[%sp], %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0	–	–	–	–	–	–	○
		[%sp]+, %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)+4	–	–	–	–	–	–	○
		[%sp]-, %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)-4	–	–	–	–	–	–	○
		-[%sp], %rs	sp(23:0)←sp(23:0)-4, A[sp](23:0)←rs(23:0), A[sp](31:24)←0	–	–	–	–	–	–	○
		%sp, %rs	sp(23:2)←rs(23:2)	–	–	–	–	–	–	○
		%sp, imm7	sp(6:2)←imm7(6:2), sp(23:7)←0	–	–	–	–	–	–	○
	sld.a	%rd, imm20	%rd←imm20	–	–	–	–	–	–	–
		%sp, imm20	%sp←imm20	–	–	–	–	–	–	–
		%rd, symbol±imm20	%rd←symbol±imm20(19:0)	–	–	–	–	–	–	–
		%sp, symbol±imm20	%sp←symbol±imm20(19:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm20]	%rd←A[%sp+imm20](23:0), 無視←A[%sp+imm20](31:24)	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←A[imm20](23:0), 無視←A[imm20](31:24)	–	–	–	–	–	–	–
		[%sp+imm20], %rs	A[%sp+imm20](23:0)←%rs(23:0), A[%sp+imm20](31:24)←0	–	–	–	–	–	–	–
		[imm20], %rs	A[imm20](23:0)←%rs(23:0), A[imm20](31:24)←0	–	–	–	–	–	–	–
	xld.a	%rd, imm24	%rd←imm24	–	–	–	–	–	–	–
		%sp, imm24	%sp←imm24	–	–	–	–	–	–	–
		%rd, symbol±imm24	%rd←symbol±imm24(23:0)	–	–	–	–	–	–	–
		%sp, symbol±imm24	%sp←symbol±imm24(23:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm24]	%rd←A[%sp+imm24](23:0), 無視←A[%sp+imm24](31:24)	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←A[imm24](23:0), 無視←A[imm24](31:24)	–	–	–	–	–	–	–
		[%sp+imm24], %rs	A[%sp+imm24](23:0)←%rs(23:0), A[imm24](31:24)←0	–	–	–	–	–	–	–
		[imm24], %rs	A[imm24](23:0)←%rs(23:0), A[%sp+imm24](31:24)←0	–	–	–	–	–	–	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
算術演算	add	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	add/c	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	add/nc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	add	*%rd, imm7*	rd(15:0)←rd(15:0)+imm7(ゼロ拡張), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sadd	%rd, imm16	rd(15:0)←rd(15:0)+imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xadd	%rd, imm16	rd(15:0)←rd(15:0)+imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	add.a	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0)	–	–	–	–	–	–	○
	add.a/c	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0) if C = 1 (nop if C = 0)	–	–	–	–	–	–	○
	add.a/nc	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0) if C = 0 (nop if C = 1)	–	–	–	–	–	–	○
	add.a	%sp, %rs	sp(23:0)←sp(23:0)+rs(23:0)	–	–	–	–	–	–	○
		%rd, imm7	rd(23:0)←rd(23:0)+imm7(ゼロ拡張)	–	–	–	–	–	–	○
		%sp, imm7	sp(23:0)←sp(23:0)+imm7(ゼロ拡張)	–	–	–	–	–	–	○
	sadd.a	%rd, imm20	rd(23:0)←rd(23:0)+imm20(ゼロ拡張)	–	–	–	–	–	–	–
		%sp, imm20	sp(23:0)←sp(23:0)+imm20(ゼロ拡張)	–	–	–	–	–	–	–
	xadd.a	%rd, imm24	rd(23:0)←rd(23:0)+imm24	–	–	–	–	–	–	–
		%sp, imm24	sp(23:0)←sp(23:0)+imm24	–	–	–	–	–	–	–
	adc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	adc/c	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	adc/nc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	adc	*%rd, imm7*	rd(15:0)←rd(15:0)+imm7(ゼロ拡張)+C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sadc	%rd, imm16	rd(15:0)←rd(15:0)+imm16+C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xadc	%rd, imm16	rd(15:0)←rd(15:0)+imm16+C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	sub	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sub/c	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	sub/nc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	sub	*%rd, imm7*	rd(15:0)←rd(15:0)-imm7(ゼロ拡張), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	ssub	%rd, imm16	rd(15:0)←rd(15:0)-imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xsub	%rd, imm16	rd(15:0)←rd(15:0)-imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	sub.a	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0)	–	–	–	–	–	–	○
	sub.a/c	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	–	–	–	–	–	–	○
	sub.a/nc	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	–	–	–	–	–	–	○
	sub.a	%sp, %rs	sp(23:0)←sp(23:0)-rs(23:0)	–	–	–	–	–	–	○
		%rd, imm7	rd(23:0)←rd(23:0)-imm7(ゼロ拡張)	–	–	–	–	–	–	○
		%sp, imm7	sp(23:0)←sp(23:0)-imm7(ゼロ拡張)	–	–	–	–	–	–	○

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
算術演算	ssub.a	%rd, imm20	rd(23:0)←rd(23:0)-imm20(ゼロ拡張)	–	–	–	–	–	–	–
		%sp, imm20	sp(23:0)←sp(23:0)-imm20(ゼロ拡張)	–	–	–	–	–	–	–
	xsub.a	%rd, imm24	rd(23:0)←rd(23:0)-imm24	–	–	–	–	–	–	–
		%sp, imm24	sp(23:0)←sp(23:0)-imm24	–	–	–	–	–	–	–
	sbc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sbc/c	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	sbc/nc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	sbc	*%rd, imm7*	rd(15:0)←rd(15:0)-imm7(ゼロ拡張)-C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	ssbc	%rd, imm16	rd(15:0)←rd(15:0)-imm16-C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xsbc	%rd, imm16	rd(15:0)←rd(15:0)-imm16-C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	cmp	%rd, %rs	rd(15:0)-rs(15:0)	–	–	↔	↔	↔	↔	○
	cmp/c	%rd, %rs	rd(15:0)-rs(15:0) if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	cmp/nc	%rd, %rs	rd(15:0)-rs(15:0) if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	cmp	*%rd, sign7*	rd(15:0)-sign7(符号拡張)	–	–	↔	↔	↔	↔	○
	scmp	%rd, imm16	rd(15:0)-imm16	–	–	↔	↔	↔	↔	–
	xcmp	%rd, imm16	rd(15:0)-imm16	–	–	↔	↔	↔	↔	–
	cmp.a	%rd, %rs	d(23:0)-rs(23:0)	–	–	↔	–	↔	–	○
	cmp.a/c	%rd, %rs	rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	–	–	–	–	↔	–	○
	cmp.a/nc	%rd, %rs	rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	–	–	–	–	↔	–	○
	cmp.a	*%rd, imm7*	rd(23:0)-imm7(ゼロ拡張)	–	–	↔	–	↔	–	○
	scmp.a	%rd, imm20	rd(23:0)-imm20(ゼロ拡張)	–	–	↔	–	↔	–	–
	xcmp.a	%rd, imm24	rd(23:0)-imm24	–	–	↔	–	↔	–	–
	cmc	%rd, %rs	rd(15:0)-rs(15:0)-C	–	–	↔	↔	↔	↔	○
	cmc/c	%rd, %rs	rd(15:0)-rs(15:0)-C if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	cmc/nc	%rd, %rs	rd(15:0)-rs(15:0)-C if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	cmc	*%rd, sign7*	rd(15:0)-sign7(符号拡張)-C	–	–	↔	↔	↔	↔	○
	scmc	%rd, imm16	rd(15:0)-imm16-C	–	–	↔	↔	↔	↔	–
	xcmc	%rd, imm16	rd(15:0)-imm16-C	–	–	↔	↔	↔	↔	–
論理演算	and	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	and/c	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	and/nc	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	and	*%rd, sign7*	rd(15:0)←rd(15:0)&sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	sand	%rd, imm16	rd(15:0)←rd(15:0)&imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xand	%rd, imm16	rd(15:0)←rd(15:0)&imm16, rd(23:16)←0	–	–	–	0	↔	↔	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
論理演算	or	%rd, %rs	d(15:0)←rd(15:0) | rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	or/c	%rd, %rs	rd(15:0)←rd(15:0) | rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	or/nc	%rd, %rs	rd(15:0)←rd(15:0) | rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	or	*%rd, sign7*	rd(15:0)←rd(15:0) | sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	soor	%rd, imm16	rd(15:0)←rd(15:0) | imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xoor	%rd, imm16	rd(15:0)←rd(15:0) | imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xor	%rd, %rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	xor/c	%rd, %rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	xor/nc	%rd, %rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	xor	*%rd, sign7*	rd(15:0)←rd(15:0)^sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	sxor	%rd, imm16	rd(15:0)←rd(15:0)^imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xxor	%rd, imm16	rd(15:0)←rd(15:0)^imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	not	%rd, %rs	rd(15:0)←!rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	not/c	%rd, %rs	rd(15:0)←!rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	not/nc	%rd, %rs	rd(15:0)←!rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	not	*%rd, sign7*	rd(15:0)←!sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	snot	%rd, imm16	rd(15:0)←!imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xnot	%rd, imm16	rd(15:0)←!imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
分岐	jpr / jpr.d	%rb	pc←pc+2+rb	–	–	–	–	–	–	–
		sign10	pc←pc+2+sign11; sign11={sign10,0}	–	–	–	–	–	–	–
	sjpr / sjpr.d	label±imm20	pc←label±imm20	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20	–	–	–	–	–	–	–
	xjpr / xjpr.d	label±imm24	pc←label±imm24	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24	–	–	–	–	–	–	–
	jpa / jpa.d	%rb	pc←rb	–	–	–	–	–	–	–
		imm7	pc←imm7	–	–	–	–	–	–	–
	sjpa / sjpa.d	label±imm20	pc←label±imm20	–	–	–	–	–	–	–
		imm20	pc←imm20	–	–	–	–	–	–	–
	xjpa / xjpa.d	label±imm24	pc←label±imm24	–	–	–	–	–	–	–
		imm24	pc←imm24	–	–	–	–	–	–	–
	jrgt / jrgt.d	*sign7*	pc←pc+2+sign8 if !Z&!(N^V) is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrgt / sjrgt.d	label±imm20	pc←label±imm20 if !Z&!(N^V) is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !Z&!(N^V) is true	–	–	–	–	–	–	–
	xjrgt / xjrgt.d	label±imm24	pc←label±imm24 if !Z&!(N^V) is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !Z&!(N^V) is true	–	–	–	–	–	–	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
分岐	*jrge / jrge.d*	*sign7*	pc←pc+2+sign8 if !(N^V) is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrge / sjrge.d	label±imm20	pc←label±imm20 if !(N^V) is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !(N^V) is true	–	–	–	–	–	–	–
	xjrge / xjrge.d	label±imm24	pc←label±imm24 if !(N^V) is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !(N^V) is true	–	–	–	–	–	–	–
	jrlt / jrlt.d	*sign7*	pc←pc+2+sign8 if N^V is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrlt / sjrlt.d	label±imm20	pc←label±imm20 if N^V is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if N^V is true	–	–	–	–	–	–	–
	xjrlt / xjrlt.d	label±imm24	pc←label±imm24 if N^V is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if N^V is true	–	–	–	–	–	–	–
	jrle / jrle.d	*sign7*	pc←pc+2+sign8 if Z | (N^V) is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrle / sjrle.d	label±imm20	pc←label±imm20 if Z | (N^V) is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if Z | (N^V) is true	–	–	–	–	–	–	–
	xjrle / xjrle.d	label±imm24	pc←label±imm24 if Z | (N^V) is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if Z | (N^V) is true	–	–	–	–	–	–	–
	jrugt / jrugt.d	*sign7*	pc←pc+2+sign8 if !Z&!C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrugt / sjrugt.d	label±imm20	pc←label±imm20 if !Z&!C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !Z&!C is true	–	–	–	–	–	–	–
	xjrugt / xjrugt.d	label±imm24	pc←label±imm24 if !Z&!C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !Z&!C is true	–	–	–	–	–	–	–
	jruge / jruge.d	*sign7*	pc←pc+2+sign8 if !C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjruge / sjruge.d	label±imm20	pc←label±imm20 if !C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !C is true	–	–	–	–	–	–	–
	xjruge / xjruge.d	label±imm24	pc←label±imm24 if !C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !C is true	–	–	–	–	–	–	–
	jrult / jrult.d	*sign7*	pc←pc+2+sign8 if C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrult / sjrult.d	label±imm20	pc←label±imm20 if C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if C is true	–	–	–	–	–	–	–
	xjrult / xjrult.d	label±imm24	pc←label±imm24 if C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if C is true	–	–	–	–	–	–	–
	jrule / jrule.d	*sign7*	pc←pc+2+sign8 if Z | C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrule / sjrule.d	label±imm20	pc←label±imm20 if Z | C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if Z | C is true	–	–	–	–	–	–	–
	xjrule / xjrule.d	label±imm24	pc←label±imm24 if Z | C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if Z | C is true	–	–	–	–	–	–	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
分岐	*jreq / jreq.d*	*sign7*	pc←pc+2+sign8 if Z is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjreq / sjreq.d	label±imm20	pc←label±imm20 if Z is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if Z is true	–	–	–	–	–	–	–
	xjreq / xjreq.d	label±imm24	pc←label±imm24 if Z is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if Z is true	–	–	–	–	–	–	–
	jrne / jrne.d	*sign7*	pc←pc+2+sign8 if !Z is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrne / sjrne.d	label±imm20	pc←label±imm20 if !Z is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !Z is true	–	–	–	–	–	–	–
	xjrne / xjrne.d	label±imm24	pc←label±imm24 if !Z is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !Z is true	–	–	–	–	–	–	–
	call / call.d	%rb	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+rb	–	–	–	–	–	–	–
		sign10	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+sign11; sign11={sign10,0}	–	–	–	–	–	–	–
	scall / scall.d	label±imm20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm20	–	–	–	–	–	–	–
		sign20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+sign20	–	–	–	–	–	–	–
	xcall / xcall.d	label±imm24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm24	–	–	–	–	–	–	–
		sign24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+sign24	–	–	–	–	–	–	–
	calla / calla.d	%rb	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←rb	–	–	–	–	–	–	–
		imm7	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←imm7	–	–	–	–	–	–	–
	scalla / scalla.d	label±imm20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm20	–	–	–	–	–	–	–
		imm20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←imm20	–	–	–	–	–	–	–
	xcalla / xcalla.d	label±imm24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm24	–	–	–	–	–	–	–
		imm24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←imm24	–	–	–	–	–	–	–
	ret / ret.d		pc←A[sp](23:0), sp←sp+4	–	–	–	–	–	–	–
	int	imm5	sp←sp-4, A[sp]←{psr, pc+2}, pc←ベクタ(TTBR+imm5×4)	–	0	–	–	–	–	–
	intl	imm5, imm3	sp←sp-4, A[sp]←{psr, pc+2}, pc←ベクタ(TTBR+imm5×4), psr(IL)←imm3	↔	0	–	–	–	–	–
	reti / reti.d		{psr, pc}←A[sp], sp←sp+4	↔	↔	↔	↔	↔	↔	–
	brk		A[DBRAM]←{psr, pc+2}, A[DBRAM+4]←r0, pc←0xfffc00	–	0	–	–	–	–	–
	retd		r0←A[DBRAM+4](23:0), {psr, pc}←A[DBRAM]	↔	↔	↔	↔	↔	↔	–
シフト&スワップ	sr	%rd, %rs	右論理シフト: rd(15:0)←rd(15:0)>>rs(15:0), rd(23:16)←0, MSB←0 (∗1)	–	–	↔	–	↔	↔	○
		%rd, imm7	右論理シフト: rd(15:0)←rd(15:0)>>imm7, rd(23:16)←0, MSB←0 (∗1)	–	–	↔	–	↔	↔	○
	sa	%rd, %rs	右算術シフト: rd(15:0)←rd(15:0)>>rs(15:0), rd(23:16)←0, MSB←符号 (∗1)	–	–	↔	–	↔	↔	○
		%rd, imm7	右算術シフト: rd(15:0)←rd(15:0)>>imm7, rd(23:16)←0, MSB←符号 (∗1)	–	–	↔	–	↔	↔	○
	sl	%rd, %rs	左論理シフト: rd(15:0)←rd(15:0)<<rs(15:0), rd(23:16)←0, LSB←0 (∗1)	–	–	↔	–	↔	↔	○
		%rd, imm7	左論理シフト: rd(15:0)←rd(15:0)<<imm7, rd(23:16)←0, LSB←0 (∗1)	–	–	↔	–	↔	↔	○
	swap	%rd, %rs	rd(15:8)←rs(7:0), rd(7:0)←rs(15:8), rd(23:16)←0	–	–	–	–	–	–	○

備　　考

∗1)　シフト量: rs/imm7 = 0～3の場合は0～3ビット, rs/imm7 = 4～7の場合は4ビット, rs/imm7 = 8以上の場合は8ビット

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
コンバージョン	cv.ab	%rd, %rs	rd(23:8)←rs(7), rd(7:0)←rs(7:0)	–	–	–	–	–	–	○
	cv.as	%rd, %rs	rd(23:16)←rs(15), rd(15:0)←rs(15:0)	–	–	–	–	–	–	○
	cv.al	%rd, %rs	rd(23:16)←rs(7:0), rd(15:0)←rd(15:0)	–	–	–	–	–	–	○
	cv.la	%rd, %rs	rd(23:8)←0, rd(7:0)←rs(23:16)	–	–	–	–	–	–	○
	cv.ls	%rd, %rs	rd(23:16)←0, rd(15:0)←rs(15)	–	–	–	–	–	–	○
即値拡張	ext	imm13	次の命令の即値またはオペランドを拡張	–	–	–	–	–	–	–
システム制御	nop		ノーオペレーション	–	–	–	–	–	–	○
	halt		HALT	–	–	–	–	–	–	–
	slp		SLEEP	–	–	–	–	–	–	–
	ei		psr(IE)←1	–	1	–	–	–	–	○
	di		psr(IE)←0	–	0	–	–	–	–	○
コプロセッサ	ld.cw	%rd, %rs	co_dout0←rd, co_dout1←rs	–	–	–	–	–	–	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7	–	–	–	–	–	–	○
	sld.cw	%rd, imm20	co_dout0←rd, co_dout1←imm20	–	–	–	–	–	–	–
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20	–	–	–	–	–	–	–
	xld.cw	%rd, imm24	co_dout0←rd, co_dout1←imm24	–	–	–	–	–	–	–
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24	–	–	–	–	–	–	–
	ld.ca	%rd, %rs	co_dout0←rd, co_dout1←rs, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
	sld.ca	%rd, imm20	co_dout0←rd, co_dout1←imm20, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
	xld.ca	%rd, imm24	co_dout0←rd, co_dout1←imm24, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
	ld.cf	%rd, %rs	co_dout0←rd, co_dout1←rs, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
	sld.cf	%rd, imm20	co_dout0←rd, co_dout1←imm20, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
	xld.cf	%rd, imm24	co_dout0←rd, co_dout1←imm24, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–

備　考

拡張命令の展開形式 (1)

拡張命令		展開形式		
オペコード	オペランド	条件1	条件2	条件3
sld.b **sld.ub** **sld** **sld.a**	**%rd, [%sp+imm20]** 例) sld.b %rd, [%sp+imm20]	imm20≤0x7f	0x7f<imm20	—
		ld.b %rd, [%sp+imm20(6:0)]	ext imm20(19:7) ld.b %rd, [%sp+imm20(6:0)]	
	%rd, [imm20] 例) sld %rd, [imm20]	imm20≤0x7f	0x7f<imm20	—
		ld %rd, [imm20(6:0)]	ext imm20(19:7) ld %rd, [imm20(6:0)]	
sld.b **sld** **sld.a**	**[%sp+imm20], %rs** 例) sld.b [%sp+imm20], %rs	imm20≤0x7f	0x7f<imm20	—
		ld.b [%sp+imm20(6:0)], %rs	ext imm20(19:7) ld.b [%sp+imm20(6:0)], %rs	
	[imm20], %rs 例) sld [imm20], %rs	imm20≤0x7f	0x7f<imm20	—
		ld [imm20(6:0)], %rs	ext imm20(19:7) ld [imm20(6:0)], %rs	
sld	**%rd, imm16** 例) sld %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		ld %rd, imm16(6:0)	ext imm16(15:7) ld %rd, imm16(6:0)	
	%rd, symbol±imm16 例) sld %rd, symbol+imm16	無条件	—	—
		ext (symbol+imm16)(15:7) ld %rd, (symbol+imm16)(6:0)		
sld.a	**%rd, imm20** 例) sld.a %rd, imm20	imm20≤0x7f	0x7f<imm20	—
		ld.a %rd, imm20(6:0)	ext imm20(19:7) ld.a %rd, imm20(6:0)	
	%sp, imm20 例) sld.a %sp, imm20	imm20≤0x7f	0x7f<imm20	—
		ld.a %sp, imm20(6:0)	ext imm20(19:7) ld.a %sp, imm20(6:0)	

備　考

拡張命令の展開形式 (2)

拡張命令		展開形式		
オペコード	オペランド	条件1	条件2	条件3
sld.a	**%rd, symbol±imm20** 例) sld.a %rd, symbol+imm20	無条件	—	—
		ext (symbol+imm20)(19:7) ld.a %rd, (symbol+imm20)(6:0)		
	%sp, symbol±imm20 例) sld.a %sp, symbol-imm20	無条件	—	—
		ext (symbol-imm20)(19:7) ld.a %sp, (symbol-imm20)(6:0)		
xld.b **xld.ub** **xld** **xld.a**	**%rd, [%sp+imm24]** 例) xld.b %rd, [%sp+imm24]	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		ld.b %rd, [%sp+imm24(6:0)]	ext imm24(19:7) ld.b %rd, [%sp+imm24(6:0)]	ext imm24(23:20) ext imm24(19:7) ld.b %rd, [%sp+imm24(6:0)]
	%rd, [imm24] 例) xld %rd, [imm24]	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		ld %rd, [imm24(6:0)]	ext imm24(19:7) ld %rd, [imm24(6:0)]	ext imm24(23:20) ext imm24(19:7) ld %rd, [imm24(6:0)]
xld.b **xld** **xld.a**	**[%sp+imm24], %rs** 例) xld.b [%sp+imm24], %rs	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		ld.b [%sp+imm24(6:0)], %rs	ext imm24(19:7) ld.b [%sp+imm24(6:0)], %rs	ext imm24(23:20) ext imm24(19:7) ld.b [%sp+imm24(6:0)], %rs
	[imm24], %rs 例) xld [imm24], %rs	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		ld [imm24(6:0)], %rs	ext imm24(19:7) ld [imm24(6:0)], %rs	ext imm24(23:20) ext imm24(19:7) ld [imm24(6:0)], %rs
xld	**%rd, imm16** 例) xld %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		ld %rd, imm16(6:0)	ext imm16(15:7) ld %rd, imm16(6:0)	
	%rd, symbol±imm16 例) xld %rd, symbol+imm16	無条件	—	—
		ext (symbol+imm16)(15:7) ld %rd, (symbol+imm16)(6:0)		

備　考

拡張命令の展開形式 (3)

拡張命令		展開形式		
オペコード	オペランド	条件1	条件2	条件3
xld.a	**%rd, imm24** 例) xld.a %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		ld.a %rd, imm24(6:0)	ext imm24(19:7) ld.a %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) ld.a %rd, imm24(6:0)
	%sp, imm24 例) xld.a %sp, imm24	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		ld.a %sp, imm24(6:0)	ext imm24(19:7) ld.a %sp, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) ld.a %sp, imm24(6:0)
	%rd, symbol±imm24 例) xld.a %rd, symbol+imm24	無条件	—	—
		ext (symbol+imm24)(23:20) ext (symbol+imm24)(19:7) ld.a %rd, (symbol+imm24)(6:0)		
	%sp, symbol±imm24 例) xld.a %sp, symbol-imm24	無条件	—	—
		ext (symbol-imm24)(23:20) ext (symbol-imm24)(19:7) ld.a %sp, (symbol-imm24)(6:0)		
sadd **sadc** **ssub** **ssbc**	**%rd, imm16** 例) sadd %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		add %rd, imm16(6:0)	ext imm16(15:7) add %rd, imm16(6:0)	
sadd.a **ssub.a**	**%rd, imm20** 例) ssub.a %rd, imm20	imm20≤0x7f	0x7f<imm20	—
		sub.a %rd, imm20(6:0)	ext imm20(19:7) sub.a %rd, imm20(6:0)	
	%sp, imm20 例) sadd.a %sp, imm20	imm20≤0x7f	0x7f<imm20	—
		add.a %sp, imm20(6:0)	ext imm20(19:7) add.a %sp, imm20(6:0)	
xadd **xadc** **xsub** **xsbc**	**%rd, imm16** 例) xadc %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		adc %rd, imm16(6:0)	ext imm16(15:7) adc %rd, imm16(6:0)	

備　考

拡張命令の展開形式 (4)

拡張命令		展開形式		
オペコード	オペランド	条件1	条件2	条件3
xadd.a **xsub.a**	**%rd, imm24** 例) xsub.a %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		sub.a %rd, imm24(6:0)	ext imm24(19:7) sub.a %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) sub.a %rd, imm24(6:0)
	%sp, imm24 例) xadd.a %sp, imm24	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		add.a %sp, imm24(6:0)	ext imm24(19:7) add.a %sp, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) add.a %sp, imm24(6:0)
scmp **scmc**	**%rd, imm16** 例) scmp %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		cmp %rd, imm16(6:0)	ext imm16(15:7) cmp %rd, imm16(6:0)	
scmp.a	**%rd, imm20** 例) scmp.a %rd, imm20	imm20≤0x7f	0x7f<imm20	—
		cmp.a %rd, imm20(6:0)	ext imm20(19:7) cmp.a %rd, imm20(6:0)	
xcmp **xcmc**	**%rd, imm16** 例) xcmc %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		cmc %rd, imm16(6:0)	ext imm16(15:7) cmc %rd, imm16(6:0)	
xcmp.a	**%rd, imm24** 例) xcmp.a %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		cmp.a %rd, imm24(6:0)	ext imm24(19:7) cmp.a %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) cmp.a %rd, imm24(6:0)
sand **soor** **sxor** **snot**	**%rd, imm16** 例) sand %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		and %rd, imm16(6:0)	ext imm16(15:7) and %rd, imm16(6:0)	
xand **xoor** **xxor** **xnot**	**%rd, imm16** 例) xoor %rd, imm16	imm16≤0x7f	0x7f<imm16	—
		or %rd, imm16(6:0)	ext imm16(15:7) or %rd, imm16(6:0)	

備　考

拡張命令の展開形式 (5)

拡張命令		展開形式		
オペコード	オペランド	条件1	条件2	条件3
scall **scall.d** **sjpr** **sjpr.d**	**label±imm20** 例) scall label+imm20	無条件	—	—
		ext (label+imm20)(19:12) call (label+imm20)(11:1)		
	sign20 例) sjpr sign20	-1024≤sign20≤1023	sign20<-1024 or 1023<sign20	—
		jpr sign20(11:1)	ext sign20(19:12) jpr sign20(11:1)	
sjr*1 **sjr***1**.d**	**label±imm20** 例) sjreq label+imm20	無条件	—	—
		ext (labe+imm20)(19:8) jreq (label+imm20)(7:1)		
	sign20 例) sjrne sign20	-128≤sign20≤127	sign20<-128 or 127<sign20	—
		jrne sign20(7:1)	ext sign20(19:8) jrne sign20(7:1)	
scalla **scalla.d** **sjpa** **sjpa.d**	**label±imm20** 例) scalla label+imm20	無条件	—	—
		ext (label+imm20)(19:7) calla (label+imm20)(6:0)		
	imm20 例) sjpa imm20	imm20≤0x7f	0x7f<imm20	—
		jpa imm20(6:0)	ext imm20(19:7) jpa imm20(6:0)	
xcall **xcall.d** **xjpr** **xjpr.d**	**label±imm24** 例) xcall label+imm24	無条件	—	—
		ext (label+imm24)(23:12) call (label+imm24)(11:1)		
	sign24 例) xjpr sign24	-1024≤sign24≤1023	sign24<-1024 or 1023<sign24	—
		jpr sign24(11:1)	ext sign24(23:12) jpr sign24(11:1)	

備　考

*1)　sjreq, sjreq.d, sjrne, sjrne.d, sjrgt, sjrgt.d, sjrge, sjrge.d, sjrlt, sjrlt.d, sjrle, sjrle.d, sjrugt, sjrugt.d, sjruge, sjruge.d, sjrult, sjrult.d, sjrule, sjrule.d

拡張命令の展開形式 (6)

拡張命令		展開形式		
オペコード	オペランド	条件1	条件2	条件3
xjr*1 **xjr***1**.d**	**label±imm24** 例) xjreq label+imm24	無条件	—	—
		ext (label+imm24)(23:21) ext (label+imm24)(20:8) jreq (label+imm24)(7:1)		
	sign24 例) xjrne sign24	-128≤sign24≤127	-1048576≤sign24<-128 or 127<sign24≤1048575	sign24<-1048576 or 1048575<sign24
		jrne sign24(7:1)	ext sign24(20:8) jrne sign24(7:1)	ext sign24(23:21) ext sign24(20:8) jrne sign24(7:1)
xcalla **xcalla.d** **xjpa** **xjpa.d**	**label±imm24** 例) xcalla label+imm24	無条件	—	—
		ext (label+imm24)(23:20) ext (label+imm24)(19:7) calla (label+imm24)(6:0)		
	imm24 例) xjpa imm24	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		jpa imm24(6:0)	ext imm24(19:7) jpa imm24(6:0)	ext imm24(23:20) ext imm24(19:7) jpa imm24(6:0)
sld.cw **sld.ca** **sld.cf**	**%rd, imm20** 例) sld.cw %rd, imm20	imm20≤0x7f	0x7f<imm20	—
		ld.cw %rd, imm20(6:0)	ext imm20(19:7) ld.cw %rd, imm20(6:0)	
	%rd, symbol±imm20 例) sld.ca %rd, symbol+imm20	無条件	—	—
		ext (symbol+imm20)(19:7) ld.ca %rd, (symbol+imm20)(6:0)		
xld.cw **xld.ca** **xld.cf**	**%rd, imm24** 例) xld.cw %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xfffff	0xfffff<imm24
		ld.cw %rd, imm24(6:0)	ext imm24(19:7) ld.cw %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) ld.cw %rd, imm24(6:0)
	%rd, symbol±imm24 例) xld.ca %rd, symbol+imm24	無条件	—	—
		ext (symbol+imm24)(23:20) ext (symbol+imm24)(19:7) ld.ca %rd, (symbol+imm24)(6:0)		

備　考

*1) xjreq, xjreq.d, xjrne, xjrne.d, xjrgt, xjrgt.d, xjrge, xjrge.d, xjrlt, xjrlt.d, xjrle, xjrle.d, xjrugt, xjrugt.d, xjruge, xjruge.d, xjrult, xjrult.d, xjrule, xjrule.d

セイコーエプソン株式会社
営業本部　デバイス営業部

東京　〒191-8501　東京都日野市日野 421-8
TEL（042）587-5313（直通）　FAX（042）587-5116

大阪　〒530-6122　大阪市北区中之島 3-3-23　中之島ダイビル 22F
TEL（06）7711-6770（代表）　FAX（06）7711-6771

ドキュメントコード：413810200
2019 年 3 月作成