H8/3048, H8/3067対応
AKI-H8マイコンCコンパイラ

動作環境

O　S：Windows95以上（DOS/V, PC98で動作します）

メモリ：16Mバイト以上

ＡＫＩ－Ｈ８マイコン専用
Ｃコンパイラ

Ｃコンパイラのインストール方法と
操作方法

－－－ＷＩＮＤＯＷＳ編－－－

■Ｃ言語の基礎知識の説明が当社インターネットホームページのダウンロードコーナーにございます。このフロッピーＤＩＳＫにａｂｏｕｔＣ．ｔｘｔが入っている場合は、それをごらんください。

■秋月電子通商ホームページ　http://www.tomakomai.or.jp/akizuki/

この文書では、H8/3048F C コンパイラのインストール方法と、操作方法を説明します。

C コンパイラについて
C ソースコードをコンパイルし、オブジェクトコードを生成します。
残念ながら、プログラムのすべてを C だけで書くことはできません。スタックの 初期化や、ベクタテーブルはアセンブラで書いて、C のオブジェクトコードとリンクする必要があります。
そのために必要なアセンブラ/リンカなどは H8/3048F 用 C コンパイラパッケー ジには含まれていません。
H8/3048F 用 アセンブラはＡＫＩ－Ｈ８マイコンキットに付属しています。

インストールの方法
NEC-98 フォーマットのフロッピーディスクに入っている自己解凍形式のファイル を起動すると、コンパイラやコンパイルに必要なインクルードファイル、それにライブラリファイルができあがります。
パスの設定がめんどうなので、それらのファイルをアセンブラやリンカのあるディレクトリに一緒に置いてしまいました。ちょっとごちゃごちゃしています。

コンパイラの実行
サンプルソースのコンパイルをしてみましょう。
CC38H.exe が C コンパイラです。良く似た名前の C38H.exe は、リンカ出力をモ トローラ S フォーマット形式に変換するユーティリティーツールです。間違えな いようにしましょう。
そのままコンパイルするとインクルードファイルが見つからないというエラーが発生します。コンパイラと同じ場所にインクルードファイルを置いているにも関わらず、エラーが発生します。
DOS 3.3 のころのインクルードファイルパスの設定を Win95 でもやってみたのですが、うまくいかなかったので、bat ファイルを作ることにしました。その内容をつぎに示します。

```
c:
cd c:¥h8¥asm¥
del err.txt
cc38h.exe -CPU=300HA -INCLUDE=c:¥h8¥asm %1 )) err.txt
type err..txt
```

C38H.BAT の内容を説明します。
最初の２行で、ドライブ C に移り、カレントディレクトリを「c:¥h8¥asm¥」にします。このディレクトリにアセンブラやコンパイラなどのファイル一式が置かれています。ソースもここにあります。皆さんの環境にあわせて下さい。
前回のエラーメッセージを削除します。

```
del err.txt
```

次の行が C コンパイラ CC38H.exe を起動しています。オプションが後ろについています。
CPU を 300H シリーズのアドバンストモードにします。

```
-CPU=300HA
```

インクルードファイル（～.h）の置いてあるパスを設定しています。

```
-INCLUDE=c:¥h8¥asm
```

%1 がソースファイル名に自動的に置き換わる部分です。
最後にエラーメッセージの処理です。エラーが発生したときに、そのエラーメッセージを err.txt に書き込みます。

```
)) err.txt
```

最後の行でエラーメッセージを表示しています。

```
type err.txt
```

この bat ファイルに C ソースファイルを Drag and Drop します。すると、コンパイルが始まります。
ソースは他のディレクトリにあってもかまいませんが、コンパイル結果の obj ファイルは、コンパイラの置かれているディレクトリに作られます。

サンプルとして入っている CTEST.c をコンパイルしてみましょう。先ほど作った bat ファイル C38H.bat に、CTEST.c をずるずると引きずって重ねます。DOS 窓が開いて、コンパイルが始まります。何もエラーがなければ
CTEST.obj がコンパイラが置かれているディレクトリ、この場合は c:¥h8¥asm¥ にできあがります。

コンパイルしたあとは、リンクです。
コンパイルしてできあがった obj ファイルは、サンプルとして付属しているリセットベクタや、スタックポインタ初期化部分を含んだアセンブラファイル RESETV.mar をアセンブルした RESETV.obj とリンクします。

リンクする
C コンパイラで CTEST.c をコンパイルすると、CTEST.obj が作られます。このファイルとリセットベクタとスタックポインタ初期化処理が含まれるアセンブラファイル RESETV.mar をアセンブルして作られた RESETV.obj をリンクします。
何と何をリンクするかという情報を、付属するサンプルの場合は「CTEST.sub」という名前のファイルに記述しています。このファイルをリンカに渡すことで、リンクを実行します。
リンカもそのままではダメで、～.sub を受け取るようにオプションをつけなければなりません。そのための bat ファイルを示します。

L38H.BAT の内容

```
c:
cd c:¥h8¥asm
2-L38H.EXE -SUBCOMMAND=%1
```

「C:¥h8¥asm¥」にコンパイラやリンカ、それにコンパイラ付属のインクルードファイル（～.h）などを置いています。皆さんの環境にあわせて下さい。途中で作成されるファイルは、すべてこのディレクトリに生成されます
Bソースファイルはどこにおいてもよいのですが、ソースファイルと同じ場所に、 出力ファイルが生成されることはありません。
「2-L38H.EXE -SUBCOMMAND=%1」は、リンカとそのオプションです。
リンカは「2-L38H.EXE」になっていますが、皆さんお使いのファイル名にしてください。そのままなら、「L38H.EXE」という名前になっています。
オプション「-SUBCOMMAND=%1」で、リンクするファイルを記述してあるサブコマンドファイルを指定しています。

サブコマンドファイルの例として、CTEST.sub の内容を示します。

```
OUTPUT CTEST
PRINT CTEST
INPUT resetv,CTEST
LIB c38hab
START P(200)
EXIT
```

アンダーラインのついた「CTEST」を変えることで、ほかのファイルのリンクにも 対応できます。
リンクで CTEST.abs ができます。これを「C38H.exe」でモトローラ形式に変換し 、H8/3048 CPU ボードに転送します。

リセットベクタ、割り込みベクタそれにスタックの設定
リセットベクタとスタックの設定はアセンブラで書いて、あとでリンクします。
例として、付属の「Resetv.mar」を次に示します。

```
        .CPU 300HA
        .SECTION A,DATA,LOCATE=H'000000
        .IMPORT _main

        .DATA.L H'00100          ;リセットベクトル

        .ORG    H'000100
        MOV.L   #H'FFF10,ER7     ;スタックポインタ設定
        jmp     @_main

        .END
```

[付録 A]
自己解凍形式のファイル H8C.exe を解凍すると、次のファイルが出来上がります 。

```
H8C       EXE       735,160  97-12-21  12:37 H8C.EXE


STRING    H             975  97-11-25  13:33 STRING.H
C38CGNB   EXE       418,816  97-05-29   7:49 C38CGNB.EXE
C38ASMB   EXE       211,456  97-05-29   7:52 C38ASMB.EXE
C38FRNTB  EXE       308,224  97-05-29   7:46 C38FRNTB.EXE
C38MIDB   EXE       196,096  97-05-29   7:45 C38MIDB.EXE
C38PEPB   EXE       125,952  97-05-29   7:50 C38PEPB.EXE
CC38H     EXE        72,192  97-06-10   7:35 CC38H.EXE
C38HNB    LIB       125,187  97-05-20  13:21 C38HNB.LIB
C38HAB    LIB       131,075  97-05-20  13:22 C38HAB.LIB
CTYPE     H           1,322  97-11-25  13:33 CTYPE.H
ERRNO     H             929  97-11-25  13:34 ERRNO.H
FLOAT     H           1,562  97-11-25  13:34 FLOAT.H
LIMITS    H             660  97-11-25  13:34 LIMITS.H
MATH      H             464  97-11-25  13:34 MATH.H
SETJMP    H             243  97-11-25  13:35 SETJMP.H
STDARG    H             760  97-11-25  13:35 STDARG.H
STDDEF    H             662  97-11-25  13:36 STDDEF.H
STDIO     H           4,448  97-11-25  13:36 STDIO.H
STDLIB    H           1,560  97-11-25  13:36 STDLIB.H
ASSERT    H             288  97-11-25  13:37 ASSERT.H
3048F     H          65,498  97-12-20  17:14 3048F.H
```

[付録B]プログラム例
タイマーとパラレルポートを使ったプログラム例です。
パラレルポート１には８個のＬＥＤが接続されています。そのＬＥＤを１秒間隔で順番に点滅させます。点滅の順番は整数配列 table[] に入っています。ここを修 正すると点滅の仕方が変わりますから、いろいろ試してみ ください。データの個数は(メモリが許す限り)いくつでもかまいません。最後のデータをマイナスにしておいてください。マイナスの値が、データが最後であるという印です。
点滅速度の１秒というのは、タイマーに設定する値で変更できます。計算方法は次のとおりです。

点滅間隔(秒) ＝ １／（ＣＰＵクロック周波数 16 ＭＨｚ／ 8） × 40000 × 50

I/O アクセスを便利にするインクルードファイル 3048f.h を使わないで、直接ア ドレスを設定しています。

```
/* Timer2.c */
/* #include "3048f.h" */
#include "myType.h"

/* Prototye */
void outPort(int xx);
void initio(void);
void wait(void);            /* Wait one second          */

/* protam */
main()
{
    int table[] = {
        0x80, 0x40, 0x20, 0x10,
        0x08, 0x04, 0x02, 0x01,
        0x02, 0x04, 0x08, 0x10,
        0x20, 0x40,
        -1
    };
    int xx = 0;
    int ii = 0;

    initio();
    do {
        xx = table[ii++];
        if (xx != 0) {
            outPort(xx);
            wait();
        } else {
            ii = 0;
        }
    } while (1);
}   /* main */

void outPort(int xx)
{
    /* P1.DR.BYTE = (unsigned char)(~xx & 0xff); */
    char *p1dr = (char *)0xfffc2;
    *p1dr = (char)(~xx & 0xff);
}   /* outPort */

void initio(void)
{
    /* P1.DDR = 0xff;          */ /* Set port 1 to output    */
    /* ITU0.TCR.BYTE = 0x23;   */ /* Timer Internal Clk 1/8  */
    /* ITU0.GRA = 0x9c40;      */ /* 40000                   */
    /* ITU.TSTR.BYTE = 0xe1;   */ /* Start Count             */
```

```
    char *p1ddr = (char *)0xfffc0;
    char *tstr  = (char *)0xffff60;
    char *tcr0  = (char *)0xffff64;
    char *tsr0  = (char *)0xffff67;
    int  *gra0  = (int  *)0xffff6a;

    *p1ddr = 0xff;          /* Set port 1 to output      */
    *tcr0  = 0x23;          /* Timer Internal Clk 1/8    */
    *gra0  = 0x9c40;        /* 40000                     */
    *tstr  = 0xe1;          /* Start Count               */
}   /* initio */

void wait(void)             /* Wait one second           */
{
    int ii;

    for (ii = 0; ii < 0x50; ii++) {
        /* do { */
        /* } while (ITU0.TSR.BIT.IMFA == 0); */
        /* ITU0.TSR.BIT.IMFA = 0; */
        char *tsr0  = (char *)0xffff67;
        Boolean fin = false;
        do {
            fin = (Boolean)(*tsr0 & 1);
        } while (!fin);
        *tsr0 &= 0xfe;
    }   /* for (ii = 0; ii < 50; ii++) */
}   /* wait */

/* end of timer.c */


/*
 *  myType.h
 *
 */

#ifndef __MYTYPE__
#define __MYTYPE__

typedef signed char     SInt8;
typedef unsigned char   UInt8;
typedef signed short    SInt16;
typedef unsigned short  UInt16;
typedef signed long     SInt32;
typedef unsigned long   UInt32;
```

```
typedef char Boolean;
enum { false = 0, true = 1 };

#endif

/* end of myType.h */
```

Timer2.c をコンパイルした Timer2.obj と resetv.obj をリンクするための Timer2.sub ファイルです。このファイルをつぎに示すリンカーの .bat ファイルにド ラッグ&ドロップすることで、リンクします。

```
OUTPUT sci
PRINT sci
INPUT resetv.sci
LIB c38hab
START P(200)
EXIT
```

sub ファイルを使ってリンクするための .bat ファイルです。
L38H.BAT の内容

```
c:
cd c:¥h8¥asm
2-L38H.EXE -SUBCOMMAND=%1
```

「C:¥h8¥asm¥」にコンパイラやリンカ、それにコンパイラ付属のインクルードファイル（～.h）などを置いています。皆さんの環境にあわせて下さい。途中で作成されるファイルは、すべてこのディレクトリに生成されます
ソースファイルはどこにおいてもよいのですが、ソースファイルと同じ場所に、出力ファイルが生成されることはありません。
2-L38H.exe は、リンカーです。元の名前は L-38H.exe でした。

順番が逆になりましたが、コンパイルのための .bat ファイルです。インクルードファイルの場所を設定しています。これも先ほどと同じで、「C:¥h8¥asm¥」にコンパイラやリンカ、それにコンパイラ付属のインクルードファイル（～.h）などを置いています。皆さんの環境にあわせて下さい。
CC38H.exe は C コンパイラ本体です。

```
c:
cd c:¥h8¥asm¥
del err.txt
cc38h.exe -CPU=300HA -INCLUDE=c:¥h8¥asm %1 )) err.txt
type err.txt
```

Ｈ８／３０４８Ｆ用　Ｃコンパイラ ユーザーマニュアル

このフロッピーＤＩＳＫは１．２Ｍバイトフォーマットになっています。
ＰＣＡＴ互換機の方は、１．２Ｍバイト対応の３モードドライブでご使用してください。

あらかじめファイルを圧縮してありますので、ご使用前に解凍してください。
（解凍方法はＤＩＳＫ内のREADME.TXTをごらんください。）

■動作環境■
ＯＳ　ＷＩＮＤＯＷＳ９５以上　（ＤＯＳ／Ｖ，ＰＣ９８で動作します。）
メモリ　１６Ｍバイト以上
ＷＩＮＤＯＷＳ９５のＤＯＳプロンプトまたはＤＯＳ窓にて動作

ＲＯＭ化のためのリンカー等はこのＤＩＳＫに付属していません。
ＡＫＩ－Ｈ８キットに付属していますので、それをご使用ください。

このＣコンパイラは「ＡＮＳＩ－Ｃ」に準拠していますので、このマニュアルは
３０４８Ｆに特有な事のみを記述してあります。それ以外のＣ言語の説明は各社
から、出版されています「Ｃ言語入門」等をごらんください。

１．Ｃコンパイラの実行

２．Ｃコンパイラの実行方式

付録

1　Cコンパイラの実行

本節ではCコンパイラの起動方法、オプションの指定方法、コンパイルリストの見方について解説します。

1.1　Cコンパイラの起動方法

Cコンパイラを起動するコマンドラインの形式は次のとおりです。

cc38h［△〈オプション〉．．．］［△〈ファイル名〉［△〈オプション〉．．．］．．．］

また、オプションの形式は次のとおりです。

〈オプション〉［＝〈サブオプション〉］［，〈サブオプション〉］．．．］

以下、Cコンパイラの基本的な操作方法を説明します。

(1) 起動コマンドの入力

```
cc38h (RET)
```

コンパイルは行なわず、コマンドライン形式、オプション一覧を出力します。

(2) プログラムのコンパイル

```
cc38h△test.c (RET)
```

Cソースプログラム「test.c」をコンパイルします。

(3) オプション指定方法

```
cc38h△-cpu=300ha△-debug△test.c (RET)
cc38h△-cpu=300ha△-debug△-speed=register,switch△test.c (RET)
```

オプションcpu、debug、speedの前に、-を付加します。
複数のオプションを指定するときは、スペース（△）で区切ります。
また、複数のサブオプションを指定するときはカンマ（，）で区切ります。

(4) 複数のプログラムのコンパイル

複数のCソースプログラムを一度にコンパイルできます。

例1：複数プログラムの指定方法

```
cc38h△-cpu=300ha△test1.c△test2.c (RET)
```

例2：オプションの指定（Cソースプログラムすべてに有効なオプション指定例）

```
cc38h△-cpu=300ha△-indirect△test1.c △test2.c (RET)
```

test1.c、test2.c ともに indirect オプションが有効になります。

例3：オプションの指定（各Cソースプログラムごとに有効なオプション指定例）

```
cc38h△-cpu=300ha△test1.c△test2.c△-indirect (RET)
```

indirect オプションは test2.c のみ有効になります。Cソースプログラムごとのオプション指定は、Cソースプログラム全体に対するオプション指定よりも優先します。

1.2 コンパイラオプション

コンパイラオプションの形式と短縮形および省略時解釈の一覧を表 1.1 に示します。下線部（＿）は短縮形指定時の文字を示します。

オプション、サブオプションの指定方法は「1.1 Cコンパイラの起動方法」を参照してください。

表 1.1 コンパイラオプション一覧

No.	項 目	オプション形式	省略時解釈	関連拡張言語仕様
1	短絶対アドレスの指定	abs8 abs16	なし	#pragma abs8 #pragma abs16 #pragma abs8 section #pragma abs16 section
2	列挙型のサイズ	byteenum	なし	なし
3	switch 文展開方法	case= ifthen | table	なし	なし
4	CPU の指定	cpu= 300hn | 300ha	300ha	なし
5	デバッグ情報	debug nodebug	nodebug	なし
6	ブロック転送命令	eepmov	なし	なし

No.	項　目	オプション形式	省略時解釈	関連拡張言語仕様
7	インクルードファイル	include＝〈パス名〉	なし	なし
8	リストファイル出力	ｌｉｓｔオプションはサポートしていません。	nolist	なし
9	メモリ間接形式	indirect	なし	#pragma indirect #pragma indirect section
10	最適化レベル	optimize＝0 | 1	optimize＝1	なし
11	レジスタ変数拡張	regexpansion noregexpansion	regexpansion	なし
12	最適化内容	speed [=register | shift | loop | switch | inline | struct]	なし	#pragma inline

以下に各オプションの意味を示します。

(1) 短絶対アドレスの指定

■形式

abs8

abs16

■説明

静的領域に割り付けるデータを、短絶対アドレッシングモードでアクセスします。

abs8 オプションは、char 型、unsigned char 型および char 型、unsigned char 型の要素、メンバを含む複合型データを 8 ビット絶対アドレス（@aa:8）でアクセスするコードを生成します。

abs16 オプションは、CPU が 300ha のとき、データを 16 ビット絶対アドレス（@aa:16）でアクセスするコードを生成します。CPU が 300hn のとき、本オプションの指定は無効です。

abs8 オプションにより、8 ビット絶対アドレスでアクセスされるデータは、セクション名“$ABS8+C セクション名”、“$ABS8+D セクション名”または“$ABS8+B セクション名”に出力されます。また、abs16 オプションにより、16 ビット絶対アドレスでアクセスされるデータは、セクション名“$ABS16+C セクション名”、“$ABS16+D セクション名”または“$ABS16+B セクション名”に出力されます。リンク時には、本オプションにより出力されたセクションを短絶対アドレス領域に割り付ける必要があります。

(2) 列挙型のサイズ

■形式

byteenum

■説明

enum 宣言した列挙型のデータを char 型として扱います。

本オプションが指定された場合で、かつ、enum 宣言した列挙型のメンバの値が全て-128～127 の範囲のとき、列挙型データを char 型として扱います。

本オプションを省略した場合、および、本オプションが指定されても列挙型のメンバの値が 1 つでも-128～127 の範囲外の場合は、列挙型データを int 型として扱います。

(3) switch 文展開方式

■形式

case=ifthen | table

■説明

switch 文のコード展開方式を指定します。

case=ifthen オプションは、switch 文を if_then 方式で展開します。if_then 方式は、switch 文の評価式の値と case ラベルの値を比較し、一致すれば case ラベルの文へ飛ぶ処理を case ラベルの回数繰り返す展開方式です。この展開方式は、switch 文に含まれる case ラベルの数に比例してオブジェクトコードのサイズが増大します。

case=table オプションは、switch 文をテーブル方式で展開します。テーブル方式は、case ラベルの飛び先をジャンプテーブルに確保し、1 回のジャンプテーブルの参照で switch 文の評価式と一致する case ラベルの文へ飛び越す展開方式です。この方式は、switch 文に含まれる case ラベルの数に比例して定数領域に確保されるジャンプテーブルのサイズが増えますが、実行速度は常に一定です。

本オプションを省略した場合は、オブジェクトサイズの縮小を優先したいずれかの展開方式を C コンパイラが自動的に選択します。

また、本オプション省略時に speed オプションあるいは speed＝switch オプションを指定した場合は、実行速度を優先した展開方式を C コンパイラが自動的に選択します。

■例

```
int a, b;
  :
switch(a){
          case 1:       b=3; break;
          case 2:       b=2; break;
          case 3:       b=1; break;
          default:      b=0; break;
}
```

上記の C ソースプログラムのコード展開例を次に示します。（cpu＝300ha の場合）

```
MOV.W       @_a,R0          MOV.W   @_a,R0
MOV.B       R0H,R0H         SUB.W   #1,R0
BNE         Ld              CMP.W   #2,R0
CMP.B       #1,R0L          BHI     Ld
BEQ         L1              ADD.N   R0,R0
CMP.B       #2,R0L          MOV.W   @(L,ER0),R0
BEQ         L,2             JMP     @ER0
CMP.B       #3,R0L          :
BEQ         L3              L:（ジャンプテーブル）
BRA         Ld
```

case=ifthen 時　　case=table 時

case 値	if_then 方式		テーブル方式	
	オブジェクトサイズ	実行サイクル	オブジェクトサイズ	実行サイクル
1	22 バイト	18	28 (22+6) バイト	28
3		30		

(4) CPU の指定

■形式

CPU=300hn | 300ha

■説明

作成するオブジェクトプログラムの CPU 種別を指定します。サブオプションの一覧を表 1.2 に示します。

表 1.2 cpu オプションのサブオプション一覧

項番	サブオプション名	意　味
1	300hn	H8/300H 用ノーマルモードのオブジェクトを作成します。
2	300ha	H8/300H 用アドバンストモードのオブジェクトを作成します。

■注意

cpu オプションを省略した場合は、300ha とみなします。

(5) デバッグ情報

■形式

debug

nodebug

■説明

debug オプションは、デバッグ時にC ソースプログラムレベルでのデバッグができるようにオブジェクトファイル中にデバッグ情報を出力することを指定します。

オブジェクトファイルがリロケータブルオブジェクトプログラムの時は、直接デバッグ情報が出力されます。

本オプションは、最適化オプションを指定した場合でも有効です。

nodebug オプションは、デバッグ情報をオブジェクトファイル中に出力しないことを指定します。

本オプションの省略時解釈は、nodebug です。

■注意

C ソースレベルのデバッグを行なうためには、アセンブルおよびリンク時にもdebug オプションを指定する必要があります。

(6) ブロック転送命令

■形式

eepmov

■説明

構造体の代入文や局所変数で宣言された配列の初期値代入式を、ブロック転送命令 EEPMOV にコード展開します。

本オプションを省略した場合は、構造体の代入文などをMOV 命令または、実行時ルーチンに展開します。

■注意

EEPMOV 命令実行中にNMI 割り込みが発生すると、割り込み処理終了後、次の命令に制御が移るため動作結果が保証されません。本オプション使用時にはNMI 割り込みに注意してください。

(7) インクルードファイル

■形式

include＝〈パス名〉

■説明

include オプションは、コンパイルするＣソースプログラムが参照するインクルードファイルの存在するパス名を指定します。

パス名が複数ある場合にはカンマ（,）で区切って指定することができます。

(8) メモリ間接形式

■形式

indirect

■説明

Ｃソースプログラム内で呼び出す関数を全てメモリ間接（@@aa:8）で呼び出します。

本オプションの指定により、Ｃソースプログラム中に定義されている関数は、関数本体以外にセクション名“$INDIRECT+セクション名”にメモリ間接呼び出しのためのアドレステーブルが出力されます。

■注意

indirect オプションを指定すると、ソースプログラム内で呼び出している標準ライブラリ関数も全てメモリ間接で呼び出すようにコード生成されます。この場合、当該標準ライブラリ関数のアドレスをアドレステーブルに設定する必要があります。

例：アドレステーブルのアセンブリプログラム

```
                .IMPORT   _printf        ;     標準関数の外部参照宣言
                                         ; (_＋標準ライブラリ関数名)
                .EXPORT   $printf        ;     アドレステーブルの外部定義宣言
                                         ; ($＋標準ライブラリ関数名)
                .SECTION  $INDIRECT,DATA,ALIGN=2
$printf         .DATA.L   _printf        ;アドレステーブルの定義
                .END
```

アドレステーブルを格納できるエリアは、0x0000～0x00FF 番地に制限されています。リンク時には、リンケージマップによりアドレステーブルのセクションが0x0000～0x00FF 番地の範囲内であることを確認してください。

(10) 最適化レベル

■形式

```
optimize＝0 | 1
```

■説明

オブジェクトプログラムの最適化レベルを指定します。

optimize＝0 オプションは、C コンパイラがオブジェクトプログラムの最適化を行なわないことを示します。

optimize＝1 オプションは、C コンパイラが最適化を行なうことを示します。

本オプションの省略時解釈は、optimize＝1 とみなします。

■注意

optimize＝0 オプションを指定したとき、speed＝inline、loop オプションは無効となります。

(11) レジスタ変数拡張

■形式

```
regexpansion
noregexpansion
```

■説明

regexpansion オプションは、レジスタ変数を割り付けるレジスタの数を拡張することを指定します。

noregexpansion オプションは、レジスタ変数を割り付けるレジスタの数の拡張を行わないことを指定します。

レジスタの数を拡張した場合、一般にレジスタに割り付く変数の数が多くなり、変数のアクセススピードが速くなります。

本オプションの省略時解釈は、regexpansion です。

(12) 最適化内容の指定

■形式

```
speed [=        register |
                shift |
                loop |
                switch |
                inline
                struct]
```

■説明

コンパイラが生成するオブジェクトに対し、実行速度の高速化を図る最適化を指定します。

speed＝register オプションは、関数の入口／出口でレジスタを退避／回復するコードとして実行時ルーチンを使用せずにPUSH、POP 命令で展開します。

speed＝shift オプションは、シフト演算をより高速なオブジェクトコードで展開します。

speed＝loop、speed＝switch オプションは、for および switch 文のコード生成の高速化を指定します。

speed＝inline オプションは、呼び出す関数をインライン展開することを示します。

speed＝struct オプションは、構造体型や double 型の代入文を実行時ルーチンを用いず、直接インライン展開します。

speed のみを指定した場合は、これら全ての実行速度優先の最適化を行います。本オプションを省略した場合は、実行速度よりもオブジェクトコードのサイズ縮小を重視したオブジェクトを生成します。

■注意

最適化なし（optimize＝0）を指定したとき、speed＝loop、inline は無効となります。

2　C プログラムの実行方式

本節では、C コンパイラが生成するオブジェクトプログラムについて説明します。特に、C プログラムとアセンブリプログラムを結合する場合について詳しく説明します。

本節で述べる項目は、以下のとおりです。

2.1　アセンブリプログラムとの結合

C プログラムで使用する変数名や関数名のうち、他のオブジェクトプログラムとの間で相互に参照できる名前の規則について述べます。また、C プログラムの関数呼び出しでの引数やリターン値の受け渡し方法、レジスタの使用方法に関する規則について述べます。これらは、C プログラムの関数とアセンブリプログラムのルーチン間で相互に呼び出しや参照を行なうときに必要です。

本節では、H8/300H のハードウェアの知識を必要としますので、「ハードウェアマニュアル」をあわせてお読みください。

2.1　アセンブリプログラムとの結合

C 言語はシステムプログラムの記述に適しており、マイコン組み込み用の応用システムのほとんどの処理を C 言語で記述することができます。特に本C コンパイラでは、

組み込み関数などをサポートすることにより、全ての機能をC 言語で記述できるようになっています。

しかしながら、ハードウェアのタイミング要求やメモリサイズの制限などのように性能要求が厳しい場合、アセンブリ言語で記述し、C プログラムと結合する必要があります。

ここでは、C プログラムとアセンブリプログラムの結合時に留意すべき以下の内容について述べます。

(1) 外部名の相互参照方法
(2) 関数呼び出しのインタフェース

(1) 外部名の相互参照方法

C プログラムの中で外部名として宣言されたものは、アセンブリプログラムとの間で相互に参照あるいは更新することができます。C コンパイラは、次のものを外部名として扱います。

- 大域変数であって、かつ static 記憶クラスでないもの
- extern 記憶クラスで宣言されている変数名
- static 記憶クラスを指定されていない関数名

外部名となる変数名、関数名についての詳細は、「H シリーズ C 言語マニュアル 6.1 記憶クラス指定子 識別子の結合」を参照してください。

外部名となる変数名、関数名をアセンブリプログラムで指定する場合は、C プログラム内での名前（31 文字までが有効です）の先頭に下線（_）をつけたものになります。

■例 1 アセンブリプログラムの外部名を C プログラムで参照する方法

・アセンブリプログラムでは、「.EXPORT」制御命令を用いてシンボル名（先頭に下線（_）を付与）を外部定義宣言します。

・C プログラムでは、シンボル名（先頭に下線（_）がない）を「extern」宣言します。

アセンブリプログラム（定義する側）

```
     .EXPORT  _a, _b
     .SECTION D,DATA,ALIGN=2
_a:  .DATA.W  1
_b:  .DATA.W  1
     .END
```

Cプログラム（参照する側）

```
extern int a,b;


f()
{
        a+=b;
}
```

■例2　Cプログラムの外部名をアセンブリプログラムから参照する方法

・Cプログラムでは、シンボル名（先頭に下線（_）がない）を外部定義します。

・アセンブリプログラムでは、「.IMPORT」制御命令を用いてシンボル名（先頭に下線（_）を付与）を外部参照宣言します。

Cプログラム（定義する側）

```
char a,b;
```

アセンブリプログラム（参照する側）

```
    .IMPORT   _a,_b
    .SECTION P,CODE,ALIGN=2
    MOV.B     @_a,R5L
    MOV.B     R5L,@_b
    RTS
    .END
```

(2) 関数呼び出し

Cプログラムとアセンブリプログラム間で相互に関数呼び出しを行なうときに、アセンブリプログラム側で守るべき次の4つの規則について説明します。

(a) スタックポインタに関する規則
(b) スタックフレームの割り付け、解放に関する規則
(c) レジスタに関する規則
(d) 引数、リターン値の設定、参照に関する規則

(a) スタックポインタに関する規則

スタックポインタの指すアドレスよりも下位（0番地の方向）のスタック領域に、有効なデータを格納してはいけません。スタックポインタより下位アドレスに格納されたデータは、割り込み処理で破壊される可能性があります。

(b) スタックフレームの割り付け、解放に関する規則

関数呼び出しが行なわれた時点（JSR または BSR 命令の実行直後）では、スタックポインタはリターンアドレスの領域を指しています。この領域より上位アドレスのデータの割り付け、設定は呼び出す側の関数の役目です。

関数のリターン時は、リターンアドレスの領域の解放を呼び出される側の関数で行ないます。これは、通常 RTS 命令を用いて行ないます。これより上位アドレスの領域（リターン値アドレスおよび引数領域）は、呼び出した側の関数で解放します。

(c) レジスタに関する規則

関数呼び出し前後において、値を保証するレジスタと保証しないレジスタがあります。

各 CPU 種類におけるレジスタの保証規則を表 2.1 に示します。

表 2.1　関数呼び出し前後のレジスタ保証規則

No.	項　目	CPU 種類と対象レジスタ	プログラミングにおける留意点
		H8/300H 用	
1	保証しないレジスタ	ER0、ER1	関数呼び出し時に対象レジスタに有効な値があれば、呼び出し側で値を退避する。呼び出される側の関数では、退避せずに使用可能。
2	保証するレジスタ	ER2～ER6	対象レジスタのうち関数内で使用するレジスタの値を退避し、リターン時に回復する。

以下、レジスタ保証規則についてH8/300H用アドバンストモードの場合の具体例を示します。

■例1：アセンブリプログラムのサブルーチンをCプログラムから呼び出す場合

アセンブリプログラム（呼び出される側）

```
        .EXPORT   _sub
        .SECTION  P,CODE,ALIGN=2
_sub:   STM.L     (ER4-ER6),@-SP
        SUB.L     #10,SP
                :
        ADD.L     #10,SP
        LDM.L     @SP+,(ER4-ER6)
        RTS
        .END
```

} 関数内で使用するレジスタの退避
} 関数本体の処理
（ER0，ER1は保証しないレジスタのため、退避せずに使用可能）
} 退避したレジスタの回復

Cプログラム（呼び出す側）

```
extern void sub();
f()
{
        sub();
}
```

■例2：Cプログラムのサブルーチンをアセンブリプログラムから呼び出す場合

Cプログラム（呼び出される側）

```
void sub()
{
            .
            .
            .
}
```

アセンブリプログラム（呼び出す側）

```
      .IMPORT   _sub
      .SECTION  P,CODE,ALIGN=2
            .
            .
            .
      MOV.L     ER1,@(4,SP)
      MOV.L     ER0,ER6
      JSR       @_sub
            .
            .
            .
      .END
```

} レジスタER0，ER1に有効な値があれば空きレジスタまたはスタックに退避
関数「sub」の呼び出し

(d) 引数とリターン値の設定、参照に関する規則

以下、引数とリターン値の設定、参照法について説明します。引数とリターン値の規則は、関数の宣言において、個々の引数とリターン値の型が明示的に宣言されているかどうかによって異なります。引数とリターン値の型を明示的に宣言するには、関数の原型宣言を用います。

以下の解説では、まず引数とリターン値に対する一般的な規則について述べたあと、引数の割り付け方とリターン値の設定場所について述べます。

(i) 引数とリターン値に対する一般的な規則

引数の渡し方

引数の値を、必ず引数の割り付け領域にコピーしたあとで関数を呼び出します。呼び出した側の関数では、リターン後に引数の割り付け領域を参照することはありませんので、呼び出された側の関数で引数の値を変更しても呼び出した側の処理は直接には影響を受けません。

型変換の規則

引数を渡す場合、またはリターン値を返す場合、自動的に型変換を行なう場合があります。以下、この型変換の規則について説明します。

●リターン値の型変換

リターン値は、その関数の返す型に変換します。

●型の宣言された引数の型変換

原型宣言によって型が宣言されている引数は、宣言された型に変換します。

●型の宣言されていない引数の型変換

原型宣言によって型が宣言されていない引数の型変換は、以下の規則に従って変換します。

- char 型、unsigned char 型の引数は、int 型に変換します。
- float 型の引数は、double 型に変換します。
- 上記以外の型は、変換しません。

■注意

C コンパイラでは、原型宣言によって引数の型を宣言していない場合、正しく引数が渡されるように呼び出される側と呼び出す側で同じ型を指定しないと、動作を保証しません。

動作を保証しない指定例では、関数「f」の引数の原型宣言がないため、関数「main」の側で呼び出すときに引数 x を double 型に変換します。

一方、関数「f」の側では引数を float 型として宣言していますので正しく引数を受け渡しすることはできません。

原型宣言によって引数の型を宣言するか、関数「f」の側の引数宣言を double 型にする必要があります。

正しい指定例は、原型宣言によって引数の型を宣言した例です。

(ii) 引数の割り付け領域

引数は、スタック上の引数領域に割り付ける場合と、レジスタに割り付ける場合があります。

オブジェクト種類ごとの引数の割り付け領域を図 2.2 に、引数割り付け領域の一般規則を表 2.2 にそれぞれ示します。

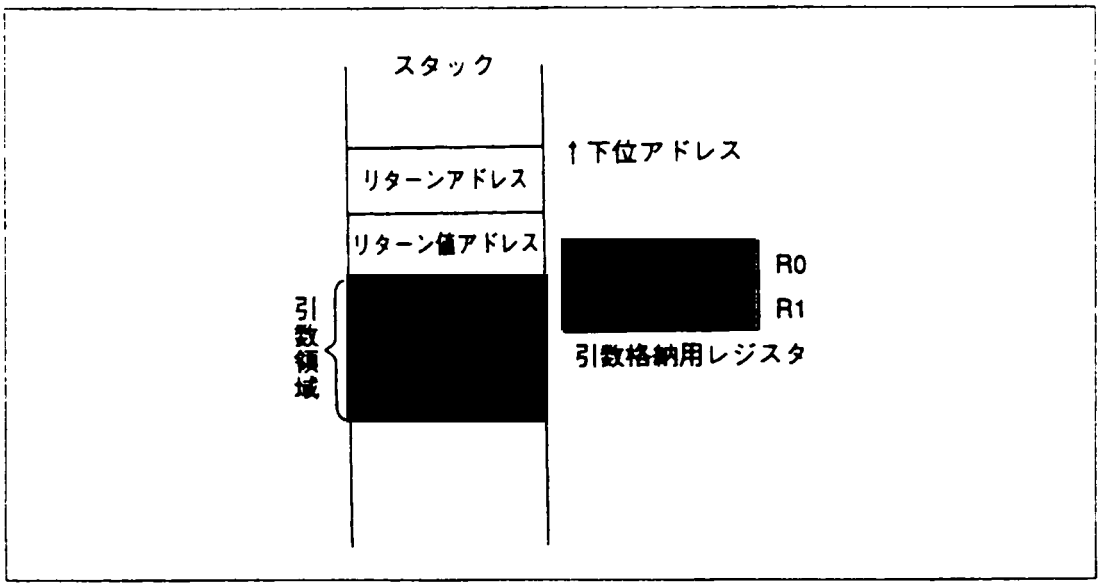


図 2.2 引数の割り付け領域

表 2.2　引数割り付け領域の一般規則

No.	CPU 種類	割り付け規則		
		レジスタに割り付ける引数		スタックに割り付ける引数
		引数格納用レジスタ	対象の型	
1	H8/300H 用	ER0、ER1	char、unsigned char、short、unsigned short、int、unsigned int、long、unsigned long、float、ポインタ	[1] 引数の型がレジスタ渡しの対象の型以外のもの [2] 原型宣言により可変個の引数をもつ関数として宣言しているもの *1 [3] 引数の数が多いため、レジスタに割り付かなかったもの

【注】　*1　原型宣言により可変個の引数をもつ関数として宣言している場合、宣言の中で対応する型のない引数およびその直前の引数はスタックに割り付けます。

例：

```
int f2(int,int,...);
        :
        f2(x,y,z);    →  y,z はスタックに割り付けます。
        :
```

(iii) 引数の割り付け

引数格納用レジスタへの割り付け

引数格納用レジスタには、ソースプログラムの宣言順に番号の小さい、LSB 側のレジスタから割り付けます。

スタック上の引数領域への割り付け

スタック上の引数領域には、ソースプログラム上で指定した順に下位アドレスから割り付けます。

■注意

構造体型、共用体型引数に関する注意

構造体型、共用体型の引数を設定する場合は、その型の本来の境界調整にかかわらず2 バイト境界に割り付け、しかもその領域として偶数バイトの領域を使用します。これは、H8S、H8/300 シリーズのスタックポインタが2 バイト単位で変化するためです。

(iV) リターン値の設定場所

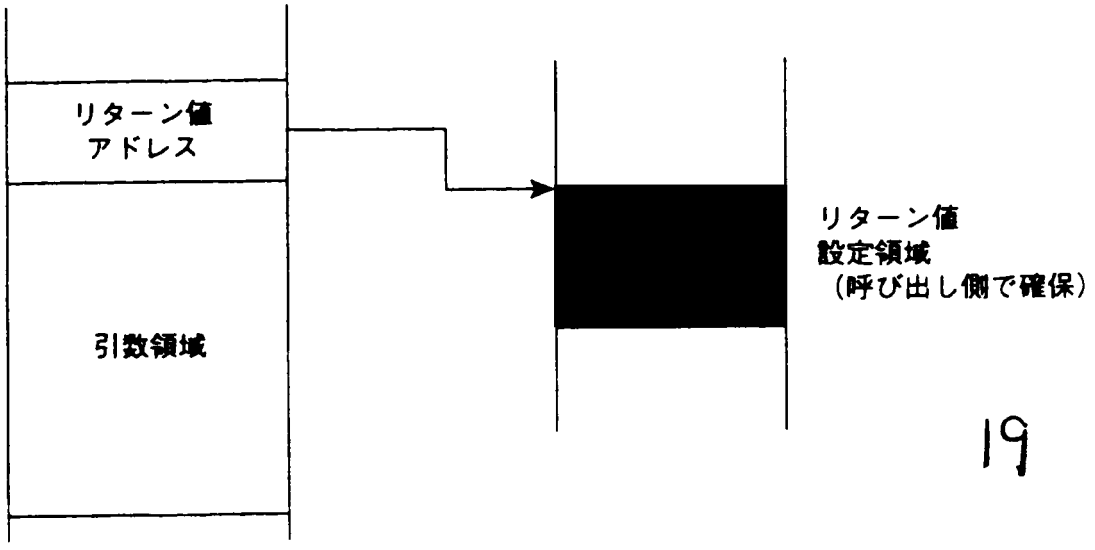
関数のリターン値の型により、リターン値をレジスタに設定する場合とメモリに設定する場合があります。リターン値の型と設定場所の関係は表 2.3 を参照してください。

関数のリターン値をメモリに設定する場合、リターン値はリターン値アドレスの指す領域に設定します。呼び出す側では、引数領域のほかにリターン値設定領域を確保し、そのアドレスをリターン値アドレスの領域に設定してから関数を呼び出します（図 2.4 参照）。関数のリターン値が void 型の場合、リターン値を設定しません。

表 2.3　リターン値の型と設定場所

No.	リターン値の型	リターン値の設定場所
		H8/300H 用
1	char、 unsigned char	レジスタ（R0L）
2	short、 unsigned short、 int、 unsigned int	レジスタ（R0）
3	ポインタ	レジスタ ノーマルモード：（R0） アドバンストモード：（ER0）
4	long、 unsigned long、 float	レジスタ（ER0）
5	double、 long double、 構造体、共用体	リターン値設定領域（メモリ）

図 2.4　リターン値をメモリに設定する場合の設定領域



メモリ割り付け例とリンク時のアドレス指定方法

アブソリュートロードモジュール作成時に、リンケージエディタのオプションまたはサブコマンドで各セクションごとに割り付ける領域のアドレスを指定します。以下、静的領域のメモリ割り付け例とリンク時の指定方法について説明します。

ROM、RAM の割り付け

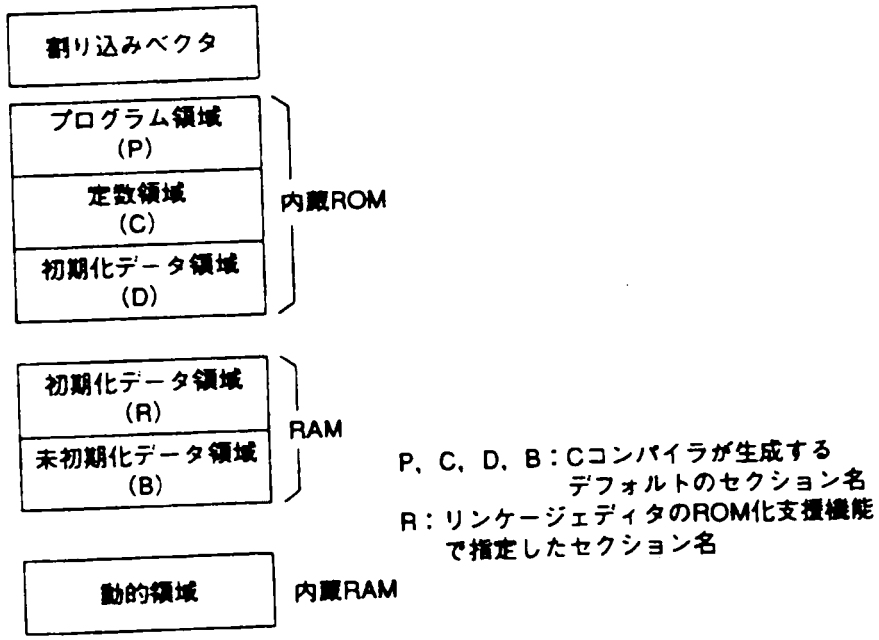
プログラムを ROM 化する場合は、静的な領域を以下のように ROM と RAM に分けて割り付けます。

プログラム領域	(セクション P) →	ROM
定数領域	(セクション C) →	ROM
未初期化データ領域	(セクション B) →	RAM
初期化データ領域	(セクション D) →	ROM、RAM

初期化データ領域の割り付け

初期化データ領域は、初期値を持ったデータ領域ですが、値の変更が可能なので、リンク時には ROM 上に置き、プログラムの実行開始時に RAM 上にコピーします。したがって、初期化データの領域については、ROM 上と RAM 上に、二重に領域をとる必要があります。

ただし、初期値を指定した静的変数を変更しないようにプログラムを作成すれば、初期化データの領域は ROM 上に置くだけでよく、二重に割り付ける必要はありません。



本編では，C言語の中で標準的に利用できる関数であるライブラリ関数の仕様について説明します。「1. ライブラリの概要」では，ライブラリの構成を概説し，本編の読み方および用語について説明します。以下の章ではライブラリの構成に従って各ライブラリ関数の仕様を説明します。

1. ライブラリの概要

(1) ライブラリの種類

ライブラリとは，入出力，文字列操作等の標準的な処理をC言語の関数の形式で実現したものです。また，これらのライブラリは，各処理単位ごとに対応した標準インクルードファイルを取り込むことによって使用可能となります。

標準インクルードファイルには，対応するライブラリの宣言とそれらを使用するために必要なマクロ名が定義されています。

表1－1にライブラリの種類と対応する標準インクルードファイルを示します。

表1－1 ライブラリの種類と対応する標準インクルードファイル

項番	ライブラリの種類	内容	標準インクルードファイル
1	プログラム診断用ライブラリ	プログラムの診断情報の出力を行なうライブラリです。	<assert.h>
2	文字操作用ライブラリ	文字の操作およびチェックを行なうライブラリです。	<ctype.h>
3	数値計算用ライブラリ	三角関数等の数値計算を行なうライブラリです。	<math.h>
4	プログラムの制御移動用ライブラリ	関数間の制御の移動をサポートするライブラリです。	<setjmp.h>
6	可変個の実引数アクセス用ライブラリ	可変個の実引数を持つ関数に対し，その実引数へのアクセスをサポートするライブラリです。	<stdarg.h>
7	入出力用ライブラリ	入出力操作を行なうライブラリです。	<stdio.h>
8	標準処理用ライブラリ	記憶域管理等のCプログラムでの標準的処理を行なうライブラリです。	<stdlib.h>
9	文字列操作用ライブラリ	文字列の比較，複写等を行なうライブラリです。	<string.h>

表1－2 マクロ名定義からなる標準インクルードファイル

項番	標準インクルードファイル	内容
1	<stddef.h>	各標準インクルードファイルで共通に使用するマクロ名を定義します。
2	<float.h>	浮動小数点数の内部表現に関する各種制限値を定義します。
3	<limits.h>	コンパイラの内部処理に関する各種制限値を定義します。

2. <stddef.h>

機能概要

標準インクルードファイルの中で共通に使用されるマクロ名を定義します。

定義名一覧

定義名	種類	説明
ptrdiff_t	マクロ名	二つのポインタを減算した結果の型を示します。
size_t	マクロ名	sizeof演算子による演算結果の型を示します。
NULL	マクロ名	ポインタが何も指していない時の値を示します。 この値は，0と等値演算子（＝＝）による比較結果が真になるような値です。
errno	マクロ名	ライブラリ関数の処理中にエラーが発生した場合，そのライブラリ関数に定義されたエラーコードがこのerrnoに設定されます。ライブラリ関数を呼び出す前にerrnoに0を設定しておき，ライブラリ関数の処理終了後にerrnoに設定されているコードを調べることによってライブラリ関数の処理中に発生したエラーをチェックすることができます。

上記のマクロ名は，すべて処理系定義です。

3. <assert.h>

機能概要

プログラム中に診断機能を付け加えます。

定義名一覧

定義名	種類	説明
assert	マクロ	プログラム中に診断機能を付け加えます。

<assert.h>で定義される診断機能を無効にするためには，<assert.h>を取り込む前にNDEBUGというマクロ名を #define文で定義してください（#define NDEBUG）。

assert関数はマクロとして実現されています。

assertというマクロ名に対して#undef文を使用すると，それ以降のassertの呼び出しの効果は保証されません。

3.1 assertマクロ	マクロ

機能	プログラム中に診断機能を付け加えます。			
呼び出し手順	#include <assert.h> int expression; assert (expression);			
パラメタ	No.	名前	型	意味
	1	expression	int	評価する式
リターン値	型	void		
	正常			
	異常			

4. <ctype.h>

機能概要

文字に対して，その種類の判定や変換を行ないます。

定義名一覧

定義名	種類	説明
isalnum	関数	英字または10進数字かどうかを判定します。
isalpha	関数	英字かどうかを判定します。
iscntrl	関数	制御文字かどうかを判定します。
isdigit	関数	10進数字かどうかを判定します。
isgraph	関数	空白を除く印字文字かどうかを判定します。
islower	関数	英小文字かどうかを判定します。
isprint	関数	空白を含む印字文字かどうかを判定します。
ispunct	関数	特殊文字かどうかを判定します。
isspace	関数	空白類文字かどうかを判定します。
isupper	関数	英大文字かどうかを判定します。
isxdigit	関数	16進数字かどうかを判定します。
tolower	関数	英大文字を英小文字に変換します。
toupper	関数	英小文字を英大文字に変換します。

表４－１　文字の種類

項番	文字の種類	内　　容
1	英大文字	以下の26文字のいずれかの文字です。 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'
2	英小文字	以下の26文字のいずれかの文字です。 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'
3	英　　字	英大文字と英小文字のいずれかの文字です。
4	10進数字	以下の10文字のいずれかの文字です。 '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
5	印字文字	空白（' '）を含む，ディスプレイ上に表示される文字のことです。 ＡＳＣＩＩコードの０x20～０x7Fに対応します。
6	制御文字	印字文字以外の文字のことです。
7	空白類文字	以下の６文字のいずれかの文字です。 空白（' '），書式送り（'＼f'），改行（'＼n'）， 復帰（'＼r'），水平タブ（'＼t'），垂直タブ（'＼v'），.
8	16進数字	以下の22文字のいずれかの文字です。 '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'a', 'b', 'c', 'd', 'e', 'f'
9	特殊文字	空白（' '），英字，及び10進数字を除く任意の印字文字のことです。

4.1　ｉｓａｌｎｕｍ関数				関　数
機能	文字が英字または１０進数字であるかどうか判定します。			
呼び出し手順	#include <ctype.h> int c , ret; ret=isalnum(c);			
パラメタ	No.	名　前	型	意　　味
	1	c	int	判定する文字
リターン値	型	int		
	正常	文字ｃが英字または10進数字の時：０以外 文字ｃが英字または10進数字以外の時：０		
	異常			

4.2 isalpha関数

関数

機能

文字が英字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int  c, ret;

        ret=isalpha(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが英字の時：0以外 文字cが英字以外の時：0
異常	

4.3 iscntrl関数

関数

機能

文字が制御文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=iscntrl(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが制御文字の時：0以外 文字cが制御文字以外の時：0
異常	

4.4 isdigit関数	関数

機能

文字が10進数字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=isdigit(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが10進数字の時：0以外 文字cが10進数字以外の時：0
異常	

4.5 isgraph関数	関数

機能

文字が空白(' ')を除く任意の印字文字かどうかを判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=isgraph(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが空白を除く印字文字の時：0以外 文字cが空白を除く印字文字以外の時：0
異常	

4.6 islower関数

関数

機能

文字が英小文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=islower(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが英小文字の時：0以外 文字cが英小文字以外の時：0
異常	

4.7 isprint関数

関数

機能

文字が空白文字(' ')を含む印字文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=isprint(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが空白文字を含む印字文字の時：0以外 文字cが空白文字を含む印字文字以外の時：0
異常	

4.8 ｉｓｐｕｎｃｔ関数

関　数

機能

文字が特殊文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=ispunct(c);
```

パラメタ

No	名　前	型	意　味
1	c	int	判定する文字

リターン値

型	int
正常	文字ｃが特殊文字の時：０以外 文字ｃが特殊文字以外の時：０
異常	

4.9 ｉｓｓｐａｃｅ関数

関　数

機能

文字が空白類文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=isspace(c);
```

パラメタ

No	名　前	型	意　味
1	c	int	判定する文字

リターン値

型	int
正常	文字ｃが空白類文字の時：０以外 文字ｃが空白類文字以外の時：０
異常	

4.10 isupper関数

関数

機能

文字が英大文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=isupper(c);
```

パラメタ

No	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが英大文字の時：0以外 文字cが英大文字以外の時：0
異常	

4.11 isxdigit関数

関数

機能

文字が16進数字かどうか判定します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=isxdigit(c);
```

パラメタ

No	名前	型	意味
1	c	int	判定する文字

リターン値

型	int
正常	文字cが16進数字の時：0以外 文字cが16進数字以外の時：0
異常	

4.12 tolower関数

関数

機能

英大文字を対応する英小文字に変換します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=tolower(c);
```

パラメタ

No.	名前	型	意味
1	c	int	変換する文字

リターン値

型	int
正常	文字cが英大文字の時：文字cに対応する英小文字 文字cが英大文字以外の時：文字c
異常	

4.13 toupper関数

関数

機能

英小文字を対応する英大文字に変換します。

呼び出し手順

```
#include <ctype.h>
int c, ret;

        ret=toupper(c);
```

パラメタ

No.	名前	型	意味
1	c	int	変換する文字

リターン値

型	int
正常	文字cが英小文字の時：文字cに対応する英大文字 文字cが英小文字以外の時：文字c
異常	

5. <float.h>

機能概要

浮動小数点数の内部表現に関する各種制限値を定義します。

定義名一覧

定義名	種類	説明
FLT_RADIX	マクロ名	指数部表現における基数を示します。
FLT_ROUNDS	マクロ名	加算演算結果が丸められるかどうかを示します。 本マクロの定義の意味は以下のとおりです。 ① 加算演算結果を丸める場合：正の値 ② 加算演算結果を切り捨てる場合：0 ③ 特に規定しない場合：－1 丸め，切り捨ての方法は，処理系定義です。
FLT_GUARD	マクロ名	乗算演算結果においてガードビットが用いられるかどうかを示します。 本マクロの定義の意味は以下のとおりです。 ① ガードビットが用いられる場合：1 ② ガードビットが用いられない場合：0
FLT_NORMALIZE	マクロ名	浮動小数点数の値が正規化されているかどうかを示します。 本マクロの定義の意味は以下のとおりです。 ① 正規化されている場合：1 ② 正規化されていない場合：0
FLT_MAX	マクロ名	float 型の浮動小数点数値として表現できる最大値を示します。
DBL_MAX	マクロ名	double型の浮動小数点数値として表現できる最大値を示します。
LDBL_MAX	マクロ名	long double 型の浮動小数点数値として表現できる最大値を示します。
FLT_MIN	マクロ名	float 型の浮動小数点数値として表現できる正の値での最小値を示します。
DBL_MIN	マクロ名	double型の浮動小数点数値として表現できる正の値での最小値を示します。
LDBL_MIN	マクロ名	long double 型の浮動小数点数値として表現できる正の値での最小値を示します。
FLT_MAX_EXP	マクロ名	float 型の浮動小数点数値として表現できる基数のべき乗の最大値を示します。
DBL_MAX_EXP	マクロ名	double型の浮動小数点数値として表現できる基数のべき乗の最大値を示します。
LDBL_MAX_EXP	マクロ名	long double 型の浮動小数点数値として表現できる基数のべき乗の最大値を示します。
FLT_MIN_EXP	マクロ名	float 型の正の値として表現できる浮動小数点数値の基数のべき乗の最小値を示します。
DBL_MIN_EXP	マクロ名	double型の正の値として表現できる浮動小数点数値の基数のべき乗の最小値を示します。
LDBL_MIN_EXP	マクロ名	long double 型の正の値として表現できる浮動小数点数値の基数のべき乗の最小値を示します。
FLT_MAX_10_EXP	マクロ名	float 型の浮動小数点数値として表現できる10のべき乗の最大値を示します。
DBL_MAX_10_EXP	マクロ名	double型の浮動小数点数値として表現できる10のべき乗の最大値を示します。
LDBL_MAX_10_EXP	マクロ名	long double 型の浮動小数点数値として表現できる10のべき乗の最大値を示します。

定義名	種類	説明
FLT_MIN_10_EXP	マクロ名	float 型の正の値として表現できる浮動小数点数値の10のべき乗の最小値を示します。
DBL_MIN_10_EXP	マクロ名	double型の正の値として表現できる浮動小数点数値の10のべき乗の最小値を示します。
LDBL_MIN_10_EXP	マクロ名	long double 型の正の値として表現できる浮動小数点数値の10のべき乗の最小値を示します。
FLT_DIG	マクロ名	float 型の浮動小数点数値の10進精度の最大桁数を示します。
DBL_DIG	マクロ名	double型の浮動小数点数値の10進精度の最大桁数を示します。
LDBL_DIG	マクロ名	long double 型の浮動小数点数値の10進精度の最大桁数を示します。
FLT_MANT_DIG	マクロ名	float 型の浮動小数点数値を基数に合わせて表現した時の仮数部の最大桁数を示します。
DBL_MANT_DIG	マクロ名	double型の浮動小数点数値を基数に合わせて表現した時の仮数部の最大桁数を示します。
LDBL_MANT_DIG	マクロ名	long double 型の浮動小数点数値を基数に合わせて表現した時の仮数部の最大桁数を示します。
FLT_EXP_DIG	マクロ名	float 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数を示します。
DBL_EXP_DIG	マクロ名	double型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数を示します。
LDBL_EXP_DIG	マクロ名	long double 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数を示します。
FLT_POS_EPS	マクロ名	float 型において，1.0 + x ≠ 1.0 である最小の浮動小数点数値 x を示します。
DBL_POS_EPS	マクロ名	double型において，1.0 + x ≠ 1.0 である最小の浮動小数点数値 x を示します。
LDBL_POS_EPS	マクロ名	long double 型において，1.0 + x ≠ 1.0 である最小の浮動小数点数値 x を示します。
FLT_NEG_EPS	マクロ名	float 型において，1.0 − x ≠ 1.0 である最小の浮動小数点数値 x を示します。
DBL_NEG_EPS	マクロ名	double型において，1.0 − x ≠ 1.0 である最小の浮動小数点数値 x を示します。
LDBL_NEG_EPS	マクロ名	long double 型において，1.0 − x ≠ 1.0 である最小の浮動小数点数値 x を示します。
FLT_POS_EPS_EXP	マクロ名	float 型において，1.0 + (基数)^n ≠ 1.0 となる最小の整数 n を示します。
DBL_POS_EPS_EXP	マクロ名	double型において，1.0 + (基数)^n ≠ 1.0 となる最小の整数 n を示します。
LDBL_POS_EPS_EXP	マクロ名	long double 型において，1.0 + (基数)^n ≠ 1.0 となる最小の整数 n を示します。
FLT_NEG_EPS_EXP	マクロ名	float 型において，1.0 − (基数)^n ≠ 1.0 となる最小の整数 n を示します。
DBL_NEG_EPS_EXP	マクロ名	double型において，1.0 − (基数)^n ≠ 1.0 となる最小の整数 n を示します。
LDBL_NEG_EPS_EXP	マクロ名	long double 型において，1.0 − (基数)^n ≠ 1.0 となる最小の整数 n を示します。

上記のマクロ名はすべて処理系定義です。

6. <limits.h>

機能概要

整数型データの内部表現に関する各種制限値を定義します。

定義名一覧

定義名	種類	説明
CHAR_BIT	マクロ名	char型が何ビットから構成されるかを示します。
CHAR_MAX	マクロ名	char型の変数が値として持つことのできる最大値を示します。
CHAR_MIN	マクロ名	char型の変数が値として持つことのできる最小値を示します。
SCHAR_MAX	マクロ名	signed char 型の変数が値として持つことのできる最大値を示します。
SCHAR_MIN	マクロ名	signed char 型の変数が値として持つことのできる最小値を示します。
UCHAR_MAX	マクロ名	unsigned char 型の変数が値として持つことのできる最大値を示します。
SHRT_MAX	マクロ名	short int 型の変数が値として持つことのできる最大値を示します。
SHRT_MIN	マクロ名	short int 型の変数が値として持つことのできる最小値を示します。
USHRT_MAX	マクロ名	unsigned short int型の変数が値として持つことのできる最大値を示します。
INT_MAX	マクロ名	int 型の変数が値として持つことのできる最大値を示します。
INT_MIN	マクロ名	int 型の変数が値として持つことのできる最小値を示します。
UINT_MAX	マクロ名	unsigned int型の変数が値として持つことのできる最大値を示します。
LONG_MAX	マクロ名	long型の変数が値として持つことのできる最大値を示します。
LONG_MIN	マクロ名	long型の変数が値として持つことのできる最小値を示します。
ULONG_MAX	マクロ名	unsigned long 型の変数が値として持つことのできる最大値を示します。

上記のマクロ名はすべて処理系定義です。

7. <math.h>

機能概要

各種の数値計算を行ないます。

定義名	種類	説明
EDOM	マクロ名	関数に入力するパラメタの値が関数内で定義している値の範囲を超える時，errno に設定する値を示しています。
ERANGE	マクロ名	関数の計算結果がdouble型の値として表わせない時，あるいはオーバーフロー／アンダフローとなった時，errno に設定する値を示しています。
HUGE_VAL	マクロ名	関数の計算結果がオーバーフローした時に，関数のリターン値として返す値を示しています。
modf	関数	浮動小数点数を整数部分と小数部分に分解します。
frexp	関数	浮動小数点数を［0.5, 1.0）の値として2のべき乗の積に分解します。
ldexp	関数	浮動小数点数と2のべき乗の乗算を計算します。
ceil	関数	浮動小数点数の小数点以下を切り上げた整数値を求めます。
fabs	関数	浮動小数点数の絶対値を計算します。
floor	関数	浮動小数点数の小数点以下を切り捨てた整数値を求めます。
fmod	関数	浮動小数点数どうしを除算した結果の余りを計算します。

上記のマクロ名はすべて処理系定義です。

frexp関数				関数
機能	浮動小数点数を［0.5, 1.0）の値と2のべき乗の積に分解します。			
呼び出し手順	#include <math.h> double ret, value; int *e; ret=frexp(value, e);			
パラメタ	No.	名前	型	意味
	1	value	double	［0.5, 1.0）の値と2のべき乗の積に分解する浮動小数点数
	2	e	int 型を指すポインタ	2のべき乗値を格納する記憶域へのポインタ
リターン値	型	double		
	正常	value が0.0 の時：0.0 value が0.0 でない時：ret＊2 eの指している記憶域の値 =valueで定義されるretの値		
	異常			

ｌｄｅｘｐ関数

関　数

機能

浮動小数点数と２のべき乗の積を計算します。

呼び出し手順

```
#include <math.h>
double ret, e;
int f;

        ret=ldexp(e, f);
```

パラメタ

No.	名　前	型	意　味
1	e	double	２のべき乗値を求める浮動小数点数
2	f	int	２のべき乗値

リターン値

型	double
正常	e*2^f の演算結果の値
異常	

ｍｏｄｆ関数

関　数

機能

浮動小数点数を整数部分と小数部分に分解します。

呼び出し手順

```
#include <math.h>
double a, *b, ret;

        ret=modf(a, b);
```

パラメタ

No.	名　前	型	意　味
1	a	double	整数部分と小数部分に分解する浮動小数点数
2	b	double型を指すポインタ	整数部分を格納する記憶域を指すポインタ

リターン値

型	double
正常	aの小数部分
異常	

7.19 ceil関数

関数

機能

浮動小数点数の小数点以下を切り上げた整数値を求めます。

呼び出し手順

```
#include <math.h>
double d, ret;

        ret=ceil(d);
```

パラメタ

No	名前	型	意味
1	d	double	小数点以下を切り上げる浮動小数点数

リターン値

型	double
正常	dの小数点以下を切り上げた整数値
異常	

7.20 fabs関数

関数

機能

浮動小数点数の絶対値を計算します。

呼び出し手順

```
#include <math.h>
double d, ret;

        ret=fabs(d);
```

パラメタ

No	名前	型	意味
1	d	double	絶対値を求める浮動小数点数

リターン値

型	double
正常	dの絶対値
異常	

7.2.1 floor関数

関数

機能

浮動小数点数の小数点以下を切り捨てた整数値を求めます。

呼び出し手順

```
#include <math.h>
double d, ret;

        ret=floor(d);
```

パラメタ

No.	名前	型	意味
1	d	double	小数点以下を切り捨てる浮動小数点数

リターン値

型	double
正常	dの小数点以下を切り捨てた整数値
異常	

7.2.2 fmod関数

関数

機能

浮動小数点数どうしを除算した結果の余りを計算します。

呼び出し手順

```
#include <math.h>
double x, y, ret;

        ret=fmod(x, y);
```

パラメタ

No.	名前	型	意味
1	x	double	被除数
2	y	double	除数

リターン値

型	double
正常	yの値が0.0 の時：x yの値が0.0 でない時：xをyで除算した結果の余り
異常	

8. <setjmp.h>

機能概要

関数間の制御の移動をサポートします。

定義名一覧

定義名	種類	説明
jmp_buf	マクロ名	関数間の制御の移動を可能とする情報を保存しておくための記憶域に対応する型名を示しています。
setjmp	関数	現在実行中の関数のjmp_buf で定義した実行環境を指定した記憶域に退避します。
longjmp	関数	setjmp関数で退避していた関数の実行環境を回復し，setjmp関数を呼び出したプログラムの位置に制御を移動します。

上記のマクロ名は，処理系定義です。

setjmp関数は現在の関数の実行環境を退避します。その後longjmp 関数を呼び出すことにより，setjmp関数を呼び出したプログラム上の位置にもどることができます。以下にsetjmp，longjmp 関数を使用して関数間の制御の移動をサポートした例を示します。

例

```
1   #include <stdio.h>
2   #include <setjmp.h>
3   jmp_buf env;
4   main( )
5   {
6
7
8           if (setjmp(env)!=0)
9               printf("return from longjmp\n");
10              exit(0);
11          }
12          sub( );
13   }
14
15  sub( )
16  {
17        printf("subroutine is running\n");
18        longjmp(env, 1);
19   }
```

説　明

8行目でsetjmp関数を呼んでいます。この時，setjmp関数の呼び出された環境を，jmp_buf型の変数env に退避します。この時のリターン値は0なので，次に関数sub が呼び出されます。

関数sub の中で呼び出されるlongjmp 関数により，変数env に退避した環境を回復します。その結果，プログラムは，あたかも8行目のsetjmp関数からリターンしたかのようにふるまいます。ただし，この時のリターン値はlongjmp 関数の第2パラメタで指定した値(1)になります。その結果，9行目以降が実行されます。

8.1 setjmp関数

関数

機能

現在実行中の関数の実行環境を，指定した記憶域に退避します。

呼び出し手順

```
#include <setjmp.h>
int ret;
jmp_buf env;

        ret=setjmp(env);
```

パラメタ

No.	名前	型	意味
1	env	jmp_buf	実行環境を退避する記憶域へのポインタ

リターン値

型	int
正常	setjmp関数を呼び出した時：0 longjmp 関数からのリターン時：0以外
異常	

8.2 longjmp関数

関数

機能

setjmp関数で退避していた関数の実行環境を回復し，setjmp関数を呼び出したプログラムの位置に制御を移動します。

呼び出し手順

```
#include <setjmp.h>
int ret;
jmp_buf env;

        longjmp(env, ret);
```

パラメタ

No.	名前	型	意味
1	env	jmp_buf	実行環境を退避した記憶域へのポインタ
2	ret	int	setjmp関数へのリターンコード

リターン値

型	void
正常	
異常	

10. <stdarg.h>

機能概要

可変個の引数を持つ関数に対し，その引数の参照を可能にします。

定義名一覧

定義名	種類	説　　明
va_list	マクロ名	可変個の引数を参照するために，va_start，va_arg，va_end マクロで共通に使用される変数の型を示しています。
va_start	マクロ	可変個の引数の参照を行なうため，初期設定処理を行ないます。
va_arg	マクロ	可変個の引数を持つ関数に対して，現在参照中引数の次の引数への参照を可能とします。
va_end	マクロ	可変個の引数を持つ関数の引数への参照を終了させます。

上記のマクロ名はすべて処理系定義です。

本標準インクルードファイルで定義しているマクロを使用したプログラムの例を以下に示します。

例

```
1   #include <stdio.h>
2   #include <stdarg.h>
3
4   extern void prlist(int count, ...);
5
6   main( )
7   {
8           prlist(1, 1);
9           prlist(3, 4, 5, 6);
10          prlist(5, 1, 2, 3, 4, 5);
11  }
12
13  void  prlist(int count, ...)
14  {
15          va_list ap;
16          int i;
17
18          va_start(ap, count);
19          for(i=0; i<count; i++)
20             printf("%d", va_arg(ap, int));
21          putchar('\n');
22          va_end(ap);
    }
```

説　明

この例では，第1引数に出力するデータの数を指定し，以下の引数をその数だけ出力する関数prlistを実現しています。

18行目で，可変個の引数への参照をva_start で初期化します。その後引数を一つ出力するたびに，va_arg マクロによって次の引数を参照します（20行目）。va_arg マクロでは，引数の型名（この場合はint 型）を第2引数に指定します。

引数の参照が終了したら，va_end マクロを呼び出します（22行目）。

10.1 va_startマクロ

マクロ

機能

可変個のパラメタへの参照を行なうため，初期設定処理を行ないます。

呼び出し手順

```
#include <stdarg.h>
va_list ap;

        va_start(ap, parmN);
```

パラメタ

No.	名　前	型	意　味
1	ap	va_list	可変個のパラメタにアクセスするための変数
2	parmN	parmN の型	最右端の引数の識別子

リターン値

型	void
正常	
異常	

10.2 va_argマクロ

マクロ

機能

可変個のパラメタを持つ関数に対して，現在参照中のパラメタの次のパラメタへの参照を可能とします。

呼び出し手順

```
#include <stdarg.h>
va_list ap;
type    ret;

        ret=va_arg(ap,type);
```

パラメタ

No.	名　前	型	意　味
1	ap	va_list	可変個のパラメタにアクセスするための変数
2	type	型名	アクセスするパラメタの型

リターン値

型	パラメタの第2引数で指定した型type
正常	パラメタの値
異常	

10.3 va_endマクロ	マクロ

機能	可変個の引数を持つ関数の引数への参照を終了させます。			
呼び出し手順	#include <stdarg.h> va_list ap; va_end(ap);			
パラメタ	No.	名前	型	意味
	1	ap	va_list	可変個の引数を参照するための変数
リターン値	型			
	正常			
	異常			

11. <stdio.h>

機能概要

ストリーム入出力用ファイルの入出力に関する処理を行ないます。

定義名一覧

定義名	種類	説明
FILE	マクロ名	ストリーム入出力処理で必要とするバッファへのポインタやエラー指示子，終了指示子などの各種制御情報を保存しておく構造体の型を示しています。
_IOFBF	マクロ名	バッファ領域の使用方法として，入出力処理はすべてバッファ領域を使用することを示しています。
_IOLBF	マクロ名	バッファ領域の使用方法として，入出力処理は行単位でバッファ領域を使用することを示しています。
_IONBF	マクロ名	バッファ領域の使用方法として，入出力処理はバッファ領域を使用しないことを示しています。
BUFSIZ	マクロ名	入出力処理において必要とするバッファの大きさを示しています。
EOF	マクロ名	ファイルの終わり(end of file) すなわちファイルからそれ以上の入力が無いことを示しています。
L_tmpnam	マクロ名	tmpnam関数によって生成される一時ファイル名の文字列を格納するのに十分な大きさの配列のサイズを示しています。
SEEK_CUR	マクロ名	ファイルの現在の読み書き位置を現在の位置からのオフセットに移すことを示しています。
SEEK_END	マクロ名	ファイルの現在の読み書き位置をファイルの終了位置からのオフセットに移すことを示しています。
SEEK_SET	マクロ名	ファイルの現在の読み書き位置をファイルの先頭位置からのオフセットに移すことを示しています。
SYS_OPEN	マクロ名	処理系が同時にオープンすることができることを保証するファイルの数を示しています。
TMP_MAX	マクロ名	tmpnam関数によって生成される一意なファイル名の個数の最小値を示します。
stderr	マクロ名	標準エラーファイルに対するファイルポインタを示します。
stdin	マクロ名	標準入力ファイルに対するファイルポインタを示します。
stdout	マクロ名	標準出力ファイルに対するファイルポインタを示します。
fclose	関　数	ストリーム入出力用ファイルをクローズします。
fflush	関　数	ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。
fopen	関　数	ストリーム入出力用ファイルを指定したファイル名によってオープンします。
freopen	関　数	現在オープンされているストリーム入力出用ファイルをクローズし，新しいファイルを指定したファイル名で再オープンします。
setbuf	関　数	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。
setvbuf	関　数	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。

fprintf	関　数	書式に従ってストリーム入出力用ファイルへデータを出力します。
fscanf	関　数	ストリーム入出力用ファイルからデータを入力し，書式に従って変換します。
printf	関　数	データを書式に従って変換し，標準出力ファイル(stdout)へ出力します。
scanf	関　数	標準入力ファイル(stdin) からデータを入力し，書式に従って変換します。
sprintf	関　数	データを書式に従って変換し，指定した領域へ出力します。
sscanf	関　数	指定した記憶域からデータを入力し，書式に従って変換します。
vfprintf	関　数	可変個のパラメタリストを書式に従って指定したストリーム入出力用ファイルに出力します。
vprintf	関　数	可変個のパラメタリストを書式に従って標準出力ファイルに出力します。
vsprintf	関　数	可変個のパラメタリストを書式に従って指定した記憶域に出力します。
fgetc	関　数	ストリーム入出力用ファイルから1文字入力します。
fgets	関　数	ストリーム入出力用ファイルから文字列を入力します。
fputc	関　数	ストリーム入出力用ファイルへ1文字出力します。
fputs	関　数	ストリーム入出力用ファイルへ文字列を出力します。
getc	マクロ名	ストリーム入出力用ファイルから1文字入力します。
getchar	マクロ名	標準入力ファイルから1文字入力します。
gets	関　数	標準入力ファイルから文字列を入力します。
putc	マクロ名	ストリーム入出力用ファイルへ1文字出力します。
putchar	マクロ名	標準出力ファイルへ1文字出力します。
puts	関　数	標準出力ファイルへ文字列を出力します。
ungetc	関　数	ストリーム入出力用ファイルへ1文字をもどします。
fread	関　数	ストリーム入出力用ファイルから指定した記憶域にデータを入力します。
fwrite	関　数	記憶域からストリーム入出力用ファイルにデータを出力します。
fseek	関　数	ストリーム入出力用ファイルの現在の読み書き位置を移動させます。
ftell	関　数	ストリーム入出力用ファイルの現在の読み書き位置を求めます。
rewind	関　数	ストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。
clearerr	関　数	ストリーム入出力用ファイルのエラー状態をクリアします。
feof	関　数	ストリーム入出力用ファイルが終わりであるかどうかを判定します。
ferror	関　数	ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。
perror	関　数	標準エラーファイル(stderr)に，エラー番号に対応したエラーメッセージを出力します。

11.5 fclose関数

関数

機能

ストリーム入出力用ファイルをクローズします。

呼び出し手順

```
#include <stdio.h>
FILE  *fp;
int   ret;

        ret=fclose(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	0
異常	0以外

11.6 fflush関数

関数

機能

ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp;
int   ret;

        ret=fflush(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	0
異常	0以外

11.7 fopen関数

関数

機能

ストリーム入出力用ファイルを，指定したファイル名によってオープンします。

呼び出し手順

```
#include <stdio.h>
FILE  *ret;
const char  *fname, *mode;

        ret=fopen(fname, mode);
```

パラメタ

No.	名前	型	意味
1	fname	const char型を指すポインタ	ファイル名を示す文字列へのポインタ
2	mode	const char型を指すポインタ	ファイルアクセスモードを示す文字列へのポインタ

リターン値

型	FILE型へのポインタ
正常	オープンしたファイルのファイル情報を指すファイルポインタ
異常	NULL

11.8 freopen関数

関数

機能

現在オープンされているストリーム入出力用ファイルをクローズし，新しいファイルを指定したファイル名で再オープンします。

呼び出し手順

```
#include <stdio.h>
const char  *fname, *mode;
FILE  *ret, *fp;

        ret=freopen(fname, mode, fp);
```

パラメタ

No.	名前	型	意味
1	fname	const char型を指すポインタ	新しいファイル名を示す文字列へのポインタ
2	mode	const char型を指すポインタ	ファイルアクセスモードを示す文字列へのポインタ
3	fp	FILE型を指すポインタ	現在オープンされているストリーム入出力用ファイルのファイルポインタ

リターン値

型	FILE型へのポインタ
正常	fp
異常	NULL

11.9 setbuf関数

関数

機能

ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp;
char buf[BUFSIZ];

        setbuf(fp, buf);
```

パラメタ

No	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ
2	buf	char型の配列を指すポインタ	バッファ領域へのポインタ

リターン値

型	void
正常	
異常	

11.10 setvbuf関数

関数

機能

ストリーム入出力用のバッファ領域をユーザプログラムの側で定義して設定します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp;
char  *buf;
int   type, ret;
size_t size;

        ret=setvbuf(fp, buf, type, size);
```

パラメタ

No	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ
2	buf	char型を指すポインタ	バッファ領域へのポインタ
3	type	int	バッファの管理方式
4	size	size_t	バッファ領域の大きさ

リターン値

型	int
正常	0
異常	0以外

11.11 fprintf関数

関数

機能

書式に従って，ストリーム入出力用ファイルへデータを出力します。

呼び出し手順

```
#include <stdio.h>
FILE *fp;
const char *control;
int  ret;

        ret=fprintf(fp, control 〔,arg〕 …);
```

パラメタ

No	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ
2	control	const char型を指すポインタ	書式を示す文字列へのポインタ
3	arg, …	特に規定せず	書式に従って出力されるデータの並び

リターン値

型	int
正常	変換し出力した文字数
異常	負の値

11.12 fscanf関数

関数

機能

ストリーム入出力用ファイルからデータを入力し，書式に従って変換します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp;
const char  *control;
int ret;

        ret=fscanf(fp, control 〔,ptr〕 …);
```

パラメタ

No	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ
2	control	const char型を指すポインタ	書式を示す文字へのポインタ
3	ptr	データ型を指すポインタ	入力したデータを格納する記憶域へのポインタ

リターン値

型	int
正常	入力変換に成功したデータの個数
異常	入力データの変換を行なう前に入力データが終了した時：EOF

11.13 printf関数　　関数

機能

データを書式に従って変換し，標準出力ファイル（stdout）へ出力します。

呼び出し手順

```
#include <stdio.h>
const char *control;
int ret;

        ret=printf(control [ ,arg ] …);
```

パラメタ

No.	名前	型	意味
1	control	const char型を指すポインタ	書式を示す文字列へのポインタ
2	arg	書式に従った型	書式に従って出力されるデータ

リターン値

型	int
正常	変換し出力した文字数
異常	負の値

11.14 scanf関数　　関数

機能

標準入力ファイル（stdin)からデータを入力し，書式に従って変換します。

呼び出し手順

```
#include <stdio.h>
const char *control;
int ret;

        ret=scanf(control [ ,ptr ] …);
```

パラメタ

No.	名前	型	意味
1	control	const char型を指すポインタ	書式を示す文字列へのポインタ
2	ptr	任意のデータを指すポインタ	入力変換したデータを格納する記憶域へのポインタ

リターン値

型	int
正常	入力変換に成功したデータの個数
異常	EOF

11.15 sprintf関数

関数

機能

データを書式に従って変換し，指定した領域へ出力します。

呼び出し手順

```
#include <stdio.h>
char *s ;
const char *control;
int ret;
        ret=sprintf(s, control [, arg ] …) ;
```

パラメタ

No.	名前	型	意味
1	s	char型を指すポインタ	データを出力する記憶域へのポインタ
2	control	const char型を指すポインタ	書式を示す文字列へのポインタ
3	arg	書式に従った型	書式に従って出力されるデータ

リターン値

型	int
正常	変換した文字数
異常	

11.16 sscanf関数

関数

機能

指定した記憶域からデータを入力し，書式に従って変換します。

呼び出し手順

```
#include <stdio.h>
const char *s, *control;
int ret;
        ret=sscanf(s, control [, ptr ] …) ;
```

パラメタ

No.	名前	型	意味
1	s	const char型を指すポインタ	入力するデータがある記憶域
2	control	const char型を指すポインタ	書式を示す文字列へのポインタ
3	ptr	データ型を指すポインタ	入力変換したデータを格納する記憶域へのポインタ

リターン値

型	int
正常	入力変換に成功したデータの個数
異常	EOF

11.17 vfprintf関数

関数

機能

可変個のパラメタリストを書式に従って，指定したストリーム入出力用ファイルに出力します。

呼び出し手順

```
#include <stdarg.h>
#include <stdio.h>
FILE  *fp ;
const char *control;
va_list arg;
int       ret;
          ret=vfprintf(fp, control, arg) ;
```

パラメタ

No.	名　前	型	意　　味
1	fp	FILE型を指すポインタ	ファイルポインタ
2	control	const char型を指すポインタ	書式を示す文字列へのポインタ
3	arg	va_list	引数リスト

リターン値

型	int
正常	変換し出力した文字数
異常	負の値

11.18 vprintf関数

関数

機能

可変個のパラメタリストを書式に従って標準出力ファイル（stdout）に出力します。

呼び出し手順

```
#include <stdarg.h>
#include <stdio.h>
const char *control;
va_list   arg;
int         ret;
          ret=vprintf(control, arg)  ;
```

パラメタ

No.	名　前	型	意　　味
1	control	const char型を指すポインタ	書式を示す文字列へのポインタ
2	arg	va_list	引数リスト

リターン値

型	int
正常	変換し出力した文字数
異常	負の値

11.19 vsprintf関数

関数

機能

可変個のパラメタリストを書式に従って，指定した記憶域に出力します。

呼び出し手順

```
#include <stdarg.h>
#include <stdio.h>
char   *s;
const char *control;
va_list    arg;
int        ret;
           ret=vsprintf(s, control, arg);
```

パラメタ

No.	名前	型	意味
1	s	char型を指すポインタ	データを出力する記憶域へのポインタ
2	control	const char型を指すポインタ	書式を示す文字列へのポインタ
3	arg	va_list	引数リスト

リターン値

型	int
正常	変換した文字数
異常	負の数

11.20 fgetc関数

関数

機能

ストリーム入出力用ファイルから1文字入力します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp;
int   ret;

          ret=fgetc(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	ファイルの終了の時：EOF ファイルの終了でない時：入力した文字
異常	EOF

11.21 fgets関数

関数

機能

ストリーム入出力用ファイルから文字列を入力します。

呼び出し手順

```
#include <stdio.h>
char  *s,    *ret;
int    n;
FILE  *fp;
          ret=fgets(s, n, fp);
```

パラメタ

No.	名前	型	意味
1	s	char型を指すポインタ	文字列を入力する記憶域へのポインタ
2	n	int	文字列を入力する記憶域のバイト数
3	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	char型へのポインタ
正常	ファイルの終了の時；NULL ファイルの終了でない時；S
異常	NULL

11.22 fputc関数

関数

機能

ストリーム入出力用ファイルへ1文字出力します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;
int    c, ret ;

          ret=fputc(c, fp);
```

パラメタ

No.	名前	型	意味
1	c	int	出力する文字
2	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	出力した文字
異常	EOF

11.23 fputs関数

関数

機能

ストリーム入出力用ファイルへ文字列を出力します。

呼び出し手順

```
#include <stdio.h>
const char *s ;
int ret;
FILE  *fp;
        ret=fputs(s, fp) ;
```

パラメタ

No	名前	型	意味
1	s	const char型を指すポインタ	出力する文字列へのポインタ
2	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	0
異常	0以外

11.24 getcマクロ

マクロ

機能

ストリーム入出力用ファイルから1文字入力します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;
int   ret ;

        ret=getc(fp) ;
```

パラメタ

No	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	ファイルの終了の時：EOF ファイルの終了でない時：入力した文字
異常	EOF

11.25 getcharマクロ

マクロ

機能

標準入力ファイル (stdin)から，1文字入力します。

呼び出し手順

```
#include <stdio.h>
int ret ;

        ret=getchar ( ) ;
```

パラメタ

No.	名　前	型	意　味

リターン値

型	int
正常	ファイルの終了の時：EOF ファイルの終了でない時：入力した文字
異常	EOF

11.26 gets関数

関　数

機能

標準入力ファイル (stdin)から文字列を入力します。

呼び出し手順

```
#include <stdio.h>
char  *ret, *s ;

        ret=gets(s) ;
```

パラメタ

No.	名　前	型	意　味
1	s	char型を指すポインタ	文字列を入力する記憶域へのポインタ

リターン値

型	char型へのポインタ
正常	ファイルの終了の時：NULL ファイルの終了でない時：s
異常	NULL

11.27 putcマクロ

マクロ

機能

ストリーム入出力用ファイルへ1文字出力します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;
int   c,ret ;

        ret=putc(c, fp) ;
```

パラメタ

No.	名前	型	意味
1	c	int	出力する文字
2	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	出力した文字
異常	EOF

11.28 putcharマクロ

マクロ

機能

標準出力ファイル (stdout) へ1文字出力します。

呼び出し手順

```
#include <stdio.h>
int   c,ret ;

        ret=putchar(c);
```

パラメタ

No.	名前	型	意味
1	c	int	出力する文字

リターン値

型	int
正常	出力した文字
異常	EOF

11.29 puts関数

関数

機能

標準出力ファイル（stdout）へ文字列を出力します。

呼び出し手順

```
#include <stdio.h>
const  char  *s ;
int    ret ;

          ret=puts(s);
```

パラメタ

No.	名前	型	意味
1	s	const char型を指すポインタ	出力する文字列へのポインタ

リターン値

型	int
正常	0
異常	0以外

11.30 ungetc関数

関数

機能

ストリーム入出力用ファイルへ1文字をもどします。

呼び出し手順

```
#include <stdio.h>
int   c, ret ;
FILE  *fp ;

          ret=ungetc(c, fp) ;
```

パラメタ

No.	名前	型	意味
1	c	int	もどす文字
2	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	もどした文字
異常	EOF

11.31 fread関数

関数

機能

ストリーム入出力用ファイルから，指定した記憶域にデータを入力します。

呼び出し手順

```
#include <stdio.h>
void   *ptr ;
size_t   size ;
size_t   n, ret ;
FILE  *fp  ;

        ret=fread(ptr, size, n, fp)  ;
```

パラメタ

No.	名　前	型	意　　味
1	ptr	void型を指すポインタ	データを入力する記憶域へのポインタ
2	size	size_t	1メンバのバイト数
3	n	size_t	入力するメンバの数
4	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	size_t
正常	sizeもしくはnが0の時：0 size，nがともに0でない時：入力に成功したメンバ数
異常	

11.32 fwrite関数

関数

機能

メモリ領域からストリーム入出力用ファイルにデータを出力します。

呼び出し手順

```
#include <stdio.h>
const void   *ptr ;
size_t   size  ;
size_t   n, ret  ;
FILE  *fp  ;

        ret=fwrite(ptr, size, n, fp)  ;
```

パラメタ

No.	名　前	型	意　　味
1	ptr	const void型を指すポインタ	出力するデータを格納している記憶域へのポインタ
2	size	size_t	1メンバのバイト数
3	n	size_t	出力するメンバの数
4	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	size_t
正常	出力に成功したメンバ数
異常	

11.33 fseek関数

関数

機能

ストリーム入出力用ファイルの現在の読み書き位置を移動させます。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;
long  offset ;
int   type, ret ;

        ret=fseek(fp, offset, type) ;
```

パラメタ

No	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ
2	offset	long	オフセットの種類で指定された位置からのオフセット
3	type	int	オフセットの種類

リターン値

型	int
正常	0
異常	0以外

11.34 ftell関数

関数

機能

ストリーム入出力用ファイルの現在の読み書き位置を求めます。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;
long  ret ;

        ret=ftell(fp) ;
```

パラメタ

No	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	long
正常	現在の位置指示子の位置（テキストファイル） ファイルの先頭から現在位置までのバイト数（バイナリファイル）
異常	

11.35 rewind関数

関数

機能

ストリーム入出力用ファイルの現在の読み書き位置を，ファイルの先頭に移動します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;

        rewind(fp) ;
```

パラメタ

No.	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	void
正常	
異常	

11.36 clearerr関数

関数

機能

ストリーム入出力用ファイルのエラー状態をクリアします。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;

        clearerr(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	void
正常	
異常	

11.37 feof関数

関数

機能

ストリーム入出力用ファイルが終わりであるかどうかを判定します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;
int   ret ;

      ret=feof(fp) ;
```

パラメタ

No.	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	ファイルが終わりの時　：0以外 ファイルが終わりでない時：0
異常	

11.38 ferror関数

関数

機能

ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。

呼び出し手順

```
#include <stdio.h>
FILE  *fp ;
int   ret ;

      ret=ferror(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE型を指すポインタ	ファイルポインタ

リターン値

型	int
正常	ファイルがエラー状態の時　：0以外 ファイルがエラー状態でない時：0
異常	

11.39 perror関数	関 数

機能	標準エラーファイル（stderr）に，エラー番号に対応したエラーメッセージを出力します。
呼び出し手順	$include <stdio.h> const char *s ; perror(s) ;

パラメタ	No.	名 前	型	意 味
	1	s	const char型を指すポインタ	エラーメッセージへのポインタ

リターン値		
	型	void
	正常	
	異常	

12. <stdlib.h>

機能概要

Cプログラムでの標準的処理を行なう関数を定義しています。

定義名一覧

定 義 名	型	説 明
onexit_t	マクロ名	onexit関数で登録する関数の返す型およびonexit関数のリターン値の型を示しています。
div_t	マクロ名	div 関数のリターン値の構造体の型を示しています。
ldiv_t	マクロ名	ldiv関数のリターン値の構造体の型を示しています。
RAND_MAX	マクロ名	rand関数において生成する擬似乱数整数の最大値を示しています。
atof	関 数	数を表現する文字列をdouble型の浮動小数点数値に変換します。
atoi	関 数	10進数を表現する文字列をint 型の整数値に変換します。
atol	関 数	10進数を表現する文字列をlong型の整数値に変換します。
strtod	関 数	数を表現する文字列をdouble型の浮動小数点数値に変換します。
strtol	関 数	数を表現する文字列をlong型の整数値に変換します。
rand	関 数	0からRAND_MAX の間の擬似乱数整数を生成します。
srand	関 数	rand関数で生成する擬似乱数列の初期値を設定します。

calloc	関　数	記憶域を割り当てて，すべての割当てられた記憶域を0によって初期化します。
free	関　数	指定された記憶域を解放します。
malloc	関　数	記憶域を割り当てます。
realloc	関　数	記憶域の大きさを指定された大きさに変更します。
bsearch	関　数	2分割検索を行ないます。
qsort	関　数	ソートを行ないます。
abs	関　数	int 型整数の絶対値を計算します。
div	関　数	int 型整数の除算の商と余りを計算します。
labs	関　数	long型整数の絶対値を計算します。
ldiv	関　数	long型整数の除算の商と余りを計算します。

上記のマクロ名は，処理系定義です。

12.1　atof関数				関　数
機能	数を表現する文字列を，double型の浮動小数点数値に変換します。			
呼び出し手順	#include <stdlib.h> const char *nptr ; double ret ; ret=atof(nptr) ;			
パラメタ	No	名　前	型	意　味
	1	nptr	const char型を指すポインタ	変換する数を表現する文字列のポインタ
リターン値	型	double		
	正常	変換されたdouble型の浮動小数点数値		
	異常			

12.2 atoi関数

関数

機能

10進数を表現する文字列を，int 型の整数値に変換します。

呼び出し手順

```
#include <stdlib.h>
const char *nptr ;
int  ret ;

        ret=atoi(nptr) ;
```

パラメタ

No	名前	型	意味
1	nptr	const char型を指すポインタ	変換する数を表現する文字列のポインタ

リターン値

型	int
正常	変換されたint 型の整数値
異常	

12.3 atol関数

関数

機能

10進数を表現する文字列を，long型の整数値に変換します。

呼び出し手順

```
#include <stdlib.h>
const char *nptr ;
long ret ;

        ret=atol(nptr) ;
```

パラメタ

No	名前	型	意味
1	nptr	const char型を指すポインタ	変換する数を表現する文字列のポインタ

リターン値

型	long
正常	変換されたlong型の整数値
異常	

12.4 strtod関数

関数

機能

数を表現する文字列をdouble型の浮動小数点数値に変換します。

呼び出し手順

```
#include <stdlib.h>
const char *nptr ;
char       **endptr ;
double     ret  ;

           ret=strtod(nptr, endptr) ;
```

パラメタ

No	名前	型	意味
1	nptr	const char型を指すポインタ	変換する数を表現する文字列へのポインタ
2	endptr	char型を指すポインタへのポインタ	浮動小数点数値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

リターン値

型	double
正常	nptrが指している文字列が浮動小数点数を構成しない文字で始まっている時：0 nptrが指している文字列が浮動小数点数を構成する文字で始まっている時：変換されたdouble型の浮動小数点数値
異常	変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつHUGE VAL 変換後の値がアンダフローの時：0

12.5 strtol関数

関数

機能

数を表現する文字列をlong型の整数値に変換します。

呼び出し手順

```
#include <stdlib.h>
long  ret  ;
const  char  *nptr ;
char   **endptr  ;
int    base  ;

          ret=strtol(nptr, endptr, base) ;
```

パラメタ

No	名前	型	意味
1	nptr	const char型を指すポインタ	変換する数を表現する文字列へのポインタ
2	endptr	char型を指すポインタへのポインタ	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
3	base	int	変換の基数（0又は2～36）

リターン値

型	long
正常	nptrが指している文字列が整数を構成しない文字で始まっている時：0 nptrが指している文字列が整数を構成する文字で始まっている時：変換されたlong型の整数値
異常	変換後の値がオーバフローの時：変換する文字列の符号に従ってLONG_MAX あるいはLONG_MIN

12.6 rand関数

関数

機能

0からRAND_MAX の間の擬似乱数整数を生成します。

呼び出し手順

```
#include <stdlib.h>
int ret;
        ret=rand( );
```

パラメタ

No.	名前	型	意味

リターン値

型	int
正常	擬似乱数整数値
異常	

12.7 srand関数

関数

機能

rand関数で生成する擬似乱数列の初期値を設定します。

呼び出し手順

```
#include <stdlib.h>
unsigned int  seed;
        srand(seed);
```

パラメタ

No.	名前	型	意味
1	seed	unsigned int	擬似乱数列生成の初期値

リターン値

型	void
正常	
異常	

12.9 free関数

関数

機能

指定された記憶域を解放します。

呼び出し手順

```
#include <stdlib.h>
void  *ptr;

        free(ptr);
```

パラメタ

No.	名前	型	意味
1	ptr	void型を指すポインタ	解放する記憶域のアドレス

リターン値

型	void
正常	
異常	

12.8 calloc関数

関数

機能

記憶域を割り当てて，すべての割り当てられた記憶域を0によって初期化します。

呼び出し手順

```
#include <stdlib.h>
size_t  nelem, elsize;
void  *ret;

        ret=calloc(nelem, elsize) ;
```

パラメタ

No.	名前	型	意味
1	nelem	size_t	要素の数
2	elsize	size_t	ひとつの要素の占めるバイト数

リターン値

型	void型へのポインタ
正常	割り当てられた記憶域の先頭のアドレス
異常	記憶域の割り当てができなかった時，またはパラメタのいずれかが0の時：NULL

12.10 malloc関数

関数

機能

記憶域を割り当てます。

呼び出し手順

```
#include <stdlib.h>
size_t  size ;
void  *ret ;

        ret=malloc(size)  ;
```

パラメタ

No	名前	型	意味
1	size	size_t	割り当てる記憶域のバイト数

リターン値

型	void型へのポインタ
正常	割り当てられた記憶域の先頭アドレス
異常	記憶域の割り当てができなかった時，またはsizeが0の時：NULL

12.11 realloc関数

関数

機能

記憶域の大きさを指定された大きさに変更します。

呼び出し手順

```
#include <stdlib.h>
size_t  size ;
void  *ptr, *ret;

        ret=realloc(ptr, size)  ;
```

パラメタ

No	名前	型	意味
1	ptr	void型を指すポインタ	変更する記憶域の先頭アドレス
2	size	size_t	変更後の記憶域のバイト数

リターン値

型	void型へのポインタ
正常	変更した記憶域の先頭アドレス
異常	記憶域の割り当てができなかった時，またはsizeが0の時：NULL

12.17 bsearch関数

関数

機能

二分割検索を行ないます。

呼び出し手順

```
#include <stdlib.h>
const  void  *key,  *base ;
size_t  nmemb,  size ;
int (*compar) (const  void *,  const  void *) ;
void  *ret ;

            ret=bsearch(key, base, nmemb, size, compar) ;
```

パラメタ

No.	名前	型	意味
1	key	const void型を指すポインタ	検索するデータへのポインタ
2	base	const voidを指すポインタ	検索対象となるテーブルへのポインタ
3	nmemb	size_t	検索対象のメンバの数
4	size	size_t	検索対象のメンバのバイト数
5	compar	int 型を返す関数へのポインタ	比較を行なう関数へのポインタ

リターン値

型	void型へのポインタ
正常	一致するメンバが検索できた時：一致したメンバへのポインタ 一致するメンバが検索できなかった時：NULL
異常	

12.18 qsort関数

関数

機能

ソートを行ないます。

呼び出し手順

```
#include <stdlib.h>
const  void  *base;
size_t  nmemb,  size ;
int (*compar) (const  void *,  const  void *) ;

            qsort(base, nmemb, size, compar) ;
```

パラメタ

No.	名前	型	意味
1	base	const void型を指すポインタ	ソート対象となるテーブルへのポインタ
2	nmemb	size_t	ソート対象のメンバの数
3	size	size_t	ソート対象のメンバのバイト数
4	compar	int 型を返す関数へのポインタ	比較を行なう関数へのポインタ

リターン値

型	void
正常	
異常	

12.19 abs関数

関数

機能

int 型整数の絶対値を計算します。

呼び出し手順

```
#include <stdlib.h>
int  i, ret ;

        ret=abs(i) ;
```

パラメタ

No	名前	型	意味
1	i	int	絶対値を求める整数

リターン値

型	int
正常	i の絶対値
異常	

12.20 div関数

関数

機能

int 型整数の除算の商と余りを計算します。

呼び出し手順

```
#include <stdlib.h>
int  numer,  denom ;
div_t  ret;

        ret=div(numer, denom) ;
```

パラメタ

No	名前	型	意味
1	numer	int	被除数
2	denom	int	除数

リターン値

型	div_t
正常	numer をdenom で除算した結果の商と余り。
異常	

12.2.1 labs関数

関数

機能

long型整数の絶対値を計算します。

呼び出し手順

```
#include <stdlib.h>
long  j ;
long  ret ;

        ret=labs(j);
```

パラメタ

No.	名前	型	意味
1	j	long	絶対値を求める整数

リターン値

型	long
正常	j の絶対値
異常	

12.2.2 ldiv関数

関数

機能

long型整数の除算の商と余りを計算します。

呼び出し手順

```
#include <stdlib.h>
long  numer, denom ;
ldiv_t  ret ;

        ret=ldiv(numer, denom);
```

パラメタ

No.	名前	型	意味
1	numer	long	被除数
2	denom	long	除数

リターン値

型	ldiv_t
正常	numer をdenom で除算した結果の商と余り。
異常	

13. <string.h>

機能概要

文字配列の操作に必要な種々の関数を定義します。

定義名一覧

定義名	種類	説明
memcpy	関数	複写元の記憶域の内容を指定した大きさ分，複写先の記憶域に複写します。
strcpy	関数	複写元の文字列の内容を，複写先の記憶域にヌル文字も含めて複写します。
strncpy	関数	複写元の文字列を指定された文字数分，複写先の記憶域に複写します。
strcat	関数	文字列の後に，文字列を連結します。
strncat	関数	文字列に文字列を指定した文字数分，連結します。
memcmp	関数	指定された二つの記憶域の比較を行ないます。
strcmp	関数	指定された二つの文字列を比較します。
strncmp	関数	指定された二つの文字列を指定された文字数分まで比較します。
memchr	関数	指定された記憶域において，指定された文字が最初に現われる位置を検索します。
strchr	関数	指定された文字列において，指定された文字が最初に現われる位置を検索します。
strcspn	関数	指定された文字列を先頭から調べ，別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。
strpbrk	関数	指定された文字列において，別に指定された文字列中の文字が最初に現われる位置を検索します。
strrchr	関数	指定された文字列において指定された文字が最後に現われる位置を検索します。
strspn	関数	指定された文字列を先頭から調べ別に指定した文字列中の文字が先頭から何文字続くかを求めます。
strstr	関数	指定された文字列において，別に指定した文字列が最初に現われる位置を検索します。
strtok	関数	指定した文字列をいくつかの字句に切り分けます。
memset	関数	指定された記憶域の先頭から指定された文字を指定された文字数分設定します。
strerror	関数	エラーメッセージを設定します。
strlen	関数	文字列の長さを計算します。

13.1 memcpy関数

関数

機能

複写元の記憶域の内容を，指定した大きさ分，複写先の記憶域に複写します。

呼び出し手順

```
#include <string.h>
void    *ret, *s1;
const   void  *s2;
size_t   n;

        ret=memcpy(s1, s2, n) ;
```

パラメタ

No.	名前	型	意味
1	s1	void型を指すポインタ	複写先の記憶域へのポインタ
2	s2	const void型を指すポインタ	複写元の記憶域へのポインタ
3	n	size_t	複写する文字数

リターン値

型	void型へのポインタ
正常	s1の値
異常	

13.2 strcpy関数

関数

機能

複写元の文字列の内容を，複写先の記憶域にヌル文字も含めて複写します。

呼び出し手順

```
#include <string.h>
char    *s1, *ret;
const   char *s2 ;

        ret=strcpy(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	char型を指すポインタ	複写先の記憶域へのポインタ
2	s2	const char型を指すポインタ	複写元の文字列へのポインタ

リターン値

型	char型へのポインタ
正常	s1の値
異常	

13.3 strncpy関数

関数

機能

複写元の文字列を指定された文字数分，複写先の記憶域に複写します。

呼び出し手順

```
#include <string.h>
char   *s1, *ret ;
const  char *s2  ;
size_t  n ;

        ret=strncpy(s1, s2, n )  ;
```

パラメタ

No.	名　前	型	意　　味
1	s1	char型を指すポインタ	複写先の記憶域へのポインタ
2	s2	const char型を指すポインタ	複写元の文字列へのポインタ
3	n	size_t	複写する文字数

リターン値

型	char型へのポインタ
正常	s1の値
異常	

13.4 strcat関数

関数

機能

文字列の後に，文字列を連結します。

呼び出し手順

```
#include <string.h>
char   *s1, *ret ;
const  char *s2  ;

        ret=strcat(s1, s2)  ;
```

パラメタ

No.	名　前	型	意　　味
1	s1	char型を指すポインタ	連結される文字列へのポインタ
2	s2	const char型を指すポインタ	連結する文字列へのポインタ

リターン値

型	char型へのポインタ
正常	s1の値
異常	

13.5 strncat関数

関数

機能

文字列に文字列を指定した文字数分連結します。

呼び出し手順

```
#include <string.h>
char   *s1, *ret ;
const  char *s2 ;
size_t n ;

        ret=strncat(s1, s2, n);
```

パラメタ

No.	名前	型	意味
1	s1	char型を指すポインタ	連結される文字列へのポインタ
2	s2	const char型を指すポインタ	連結する文字列へのポインタ
3	n	size_t	連結する文字数

リターン値

型	char型へのポインタ
正常	s1の値
異常	

13.6 memcmp関数

関数

機能

指定された二つの記憶域の内容を比較します。

呼び出し手順

```
# include <string.h>
const  void  *s1, *s2 ;
size_t n;
int  ret ;

        ret=memcmp(s1, s2, n);
```

パラメタ

No.	名前	型	意味
1	s1	const void型を指すポインタ	比較される記憶域へのポインタ
2	s2	const void型を指すポインタ	比較する記憶域へのポインタ
3	n	size_t	比較する記憶域の文字数

リターン値

型	int
正常	s1で指された記憶域 >s2で指された記憶域の時：正の値 s2で指された記憶域 ==s2で指された記憶域の時：0 s1で指された記憶域 <s2で指された記憶域の時：負の値
異常	

13.7 strcmp関数

関数

機能

指定された二つの文字列の内容を比較します。

呼び出し手順

```
#include <string.h>
const  char  *s1, *s2 ;
int  ret ;

        ret=strcmp(s1, s2) ;
```

パラメタ

No	名前	型	意味
1	s1	const char型を指すポインタ	比較される文字列へのポインタ
2	s2	const char型を指すポインタ	比較する文字列へのポインタ

リターン値

型	int
正常	s1で指された文字列 >s2で指された文字列の時：正の値 s1で指された文字列==s2で指された文字列の時：0 s1で指された文字列 <s2で指された文字列の時：負の値
異常	

13.8 strncmp関数

関数

機能

指定された二つの文字列を指定された文字分まで比較します。

呼び出し手順

```
#include <string.h>
const  char *s1, *s2 ;
size_t n;
int  ret ;

        ret=strncmp(s1, s2, n ) ;
```

パラメタ

No	名前	型	意味
1	s1	const char型を指すポインタ	比較される文字列へのポインタ
2	s2	const char型を指すポインタ	比較する文字列へのポインタ
3	n	size_t	比較する文字数の最大値

リターン値

型	int
正常	s1で指された文字列 >s2で指された文字列の時：正の値 s1で指された文字列==s2で指された文字列の時：0 s1で指された文字列 <s2で指された文字列の時：負の値
異常	

13.9 memchr関数

関数

機能

指定された記憶域において，指定された文字が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>
const  void  *s ;
int  c ;
size_t  n ;
void  *ret ;

        ret=memchr(s, c, n ) ;
```

パラメタ

No.	名前	型	意味
1	s	const void型を指すポインタ	検索を行なう記憶域へのポインタ
2	c	int	検索する文字
3	n	size_t	検索を行なう文字数

リターン値

型	void型へのポインタ
正常	検索の結果見つかった時：見つけられた文字へのポインタ 検索の結果見つからなかった時：NULL
異常	

13.10 strchr関数

関数

機能

指定された文字列において，指定された文字が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>
const char *s ;
int  c ;
char  *ret ;

        ret=strchr(s, c) ;
```

パラメタ

No.	名前	型	意味
1	s	const char型を指すポインタ	検索を行なう文字列へのポインタ
2	c	int	検索する文字

リターン値

型	char型へのポインタ
正常	検索の結果見つかった時：見つけられた文字へのポインタ 検索の結果見つからなかった時：NULL
異常	

13.11 strcspn関数

関数

機能

指定された文字列を先頭から調べ，別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。

呼び出し手順

```
#include <string.h>
const  char  *s1, *s2 ;
size_t  ret ;

        ret=strcspn(s1, s2) ;
```

パラメタ

No.	名前	型	意味
1	s1	const char型を指すポインタ	調べられる文字列へのポインタ
2	s2	const char型を指すポインタ	s1を調べるための文字列へのポインタ

リターン値

型	size_t
正常	s2が指す文字列を構成する文字以外の文字が構成される文字列s1の先頭からの長さ
異常	

13.12 strpbrk関数

関数

機能

指定された文字列内において，別に指定された文字列中の文字が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>
const  char  *s1, *s2 ;
char   *ret ;

        ret=strpbrk(s1, s2) ;
```

パラメタ

No.	名前	型	意味
1	s1	const char型を指すポインタ	検索を行なう文字列へのポインタ
2	s2	const char型を指すポインタ	s1内で検索する文字を示す文字列へのポインタ

リターン値

型	char型へのポインタ
正常	検索の結果見つかった時：見つかった文字へのポインタ 検索の結果見つからなかった時：NULL
異常	

13.13 strrchr関数

関数

機能

指定された文字列において，指定された文字が最後に現われる位置を検索します。

呼び出し手順

```
#include <string.h>
const  char  *s ;
int    c;
char  *ret ;

          ret=strrchr(s, c) ;
```

パラメタ

No	名前	型	意味
1	s	const char型を指すポインタ	検索を行なう文字列へのポインタ
2	c	int	検索する文字

リターン値

型	char型へのポインタ
正常	検索の結果見つかった時：見つかった文字へのポインタ 検索の結果見つからなかった時：NULL
異常	

13.14 strspn関数

関数

機能

指定された文字列を先頭から調べ，別に指定した文字列中の文字が先頭から何文字続くかを求めます。

呼び出し手順

```
#include <string.h>
const  char  *s1, *s2 ;
size_t       ret ;

          ret=strspn(s1, s2) ;
```

パラメタ

No	名前	型	意味
1	s1	const char型を指すポインタ	調べられる文字列へのポインタ
2	s2	const char型を指すポインタ	s1を調べるための文字列へのポインタ

リターン値

型	size_t
正常	s1の先頭から，s2で指定した文字が続いている文字数
異常	

13.16 strtok関数

関数

機能

指定した文字列をいくつかの字句に切り分けます。

呼び出し手順

```
#include <string.h>
char  *s1, *ret ;
const  char  *s2 ;

        ret=strtok(s1, s2) ;
```

パラメタ

No	名前	型	意味
1	s1	char型を指すポインタ	いくつかの字句に切り分ける文字列へのポインタ
2	s2	const char型を指すポインタ	文字列を切り分けるための文字からなる文字列へのポインタ

リターン値

型	char型へのポインタ
正常	字句に切り分けられた時：切り分けた字句の先頭へのポインタ 字句に切り分けられなかった時：NULL
異常	

13.15 strstr関数

関数

機能

指定された文字列において，別に指定した文字列が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>
const  char  *s1, *s2 ;
char   *ret ;

        ret=strstr(s1, s2) ;
```

パラメタ

No	名前	型	意味
1	s1	const char型を指すポインタ	検索を行なう文字列へのポインタ
2	s2	const char型を指すポインタ	検索する文字列へのポインタ

リターン値

型	char型へのポインタ
正常	検索の結果見つかった時：見つけられた文字へのポインタ 検索の結果見つからなかった時：NULL
異常	

13.17 memset関数

関数

機能

指定された記憶域の先頭から，指定された文字を指定された文字数分設定します。

呼び出し手順

```
#include <string.h>
void  *s,  *ret ;
int   c ;
size_t  n ;

        ret=memset(s, c, n) ;
```

パラメタ

No.	名前	型	意味
1	s	void型を指すポインタ	文字が設定される記憶域へのポインタ
2	c	int	設定する文字
3	n	size_t	設定する文字数

リターン値

型	void型へのポインタ
正常	s の値
異常	

13.18 strerror関数

関数

機能

エラー番号を指定して，それに対するエラーメッセージを返します。

呼び出し手順

```
#include <string.h>
char  *ret ;
int  s ;

        ret=strerror(s) ;
```

パラメタ

No.	名前	型	意味
1	s	int	エラー番号

リターン値

型	char型へのポインタ
正常	エラー番号に対応するエラーメッセージ（文字列）へのポインタ
異常	

13.19 strlen関数	関数

機能	文字列の長さを計算します。			
呼び出し手順	#include <string.h> const char *s ; size_t ret ; ret=strlen(s) ;			
パラメタ	No	名前	型	意味
	1	s	const char型を返すポインタ	長さを求める文字列へのポインタ
リターン値	型	size_t		
	正常	文字列の文字数		
	異常			

memmove				
機能	複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。また、複写元と複写先の記憶域が、重なっている部分があっても、複写元の重なっている部分を上書きする前に複写するので正しく複写されます。			
呼び出し手順	#include <string.h> void *ret, *s1; const void *s2; size_t n; ret=memmove(s1,s2,n) ;			
パラメタ	No.	名前	型	意味
	1	s1	void 型を指すポインタ	複写先の記憶域へのポインタ
	2	s2	const void 型を指すポインタ	複写元の記憶域へのポインタ
	3	n	size_t	複写する文字数
リターン値	型	void 型へのポインタ		
	正常	s1 の値		
	異常			

３０４８Ｆ．Ｈ

Ｉ／Ｏのアクセス

Ｃ言語から、各Ｉ／Ｏをアクセスするには、＃ｄｅｆｉｎｅで　Ｉ／Ｏアドレスをｐ１等に置き換え、直接Ｉ／Ｏアドレスにアクセスする必要があります。

（１）ＡＤＬＣＤ．Ｃの例

```
#define p1 (volatile unsigned char *)0xffffc2
 (p1が、ポート１のデータアドレス0xffffc2を表す)
 ・・・
*p1=code;
 (ポート１にcodeの内容を出力する。)
```

（２）３０４８Ｆ．Ｈを使う例

３０４８Ｆ．Ｈが、各Ｉ／Ｏアドレスを Ｉ／Ｏ名にまとめた物です。
このファイルをインクルードすることにより、上記の様な宣言を
いちいち書く必要はありません。
各Ｉ／Ｏ名は、３０４８Ｆ．Ｈをごらんください。

ＣＴＥＳＴ．Ｃ

```
#include (3048f.h)
(3048f.hをインクルードする。)
・・・
P1.DDR = 0xff;
(ポート１のコマンドアドレスにＦＦを入れ、出力に初期設定する。)
・・・
P1.DR.BYTE = i;
(ポート１にｉの内容を出力する。)（バイトアクセス例）
・・・
P1.DR.BIT.B1 = 1;
(ポート１のビット１を１にする。)（ビットアクセス例）
```

Cライブラリ関数のスタック使用量一覧

No.	ヘッダ名	関数名	スタックサイズ（バイト） H8/300H ノーマルモード	H8/300H アドバンストモード
1	stddef.h	offsetof	—	—
2	assert.h	assert	—	—
3	ctype.h	isalnum	4	8
		isalpha	4	8
		iscntrl	4	8
		isdigit	4	8
		isgraph	4	8
		islower	4	8
		isprint	4	8
		ispunct	4	8
		isspace	4	8
		isupper	4	8
		isxdigit	4	8
		tolower	2	4
		toupper	2	4
4	math.h	frexp	54	60
		ldexp	30	52
		modf	46	52
		ceil	110	156
		fabs	28	36
		floor	78	104
		fmod	110	120
5	setjmp.h	setjmp	2	4
		longjmp	2	4
6	stdarg.h	va_start	—	—
		va_arg	—	—
		va_end	2	4
7	stdio.h	fclose	48	56
		fflush	22	24
		fopen	54	88
		freopen	66	92
		setbuf	4	8
		setvbuf	6	30
		fprintf	626	734
		fscanf	426	518
		printf	624	718
		scanf	424	502
		sprintf	626	734
		sscanf	426	518
		vfprintf	642	738
		vprintf	626	734
		vsprintf	642	738
		fgetc	66	140
		fgets	88	160
		fputc	84	156
7	stdio.h	fputs	90	180
		getc	64	124
		getchar	62	124
		gets	86	148
		putc	84	156
		putchar	100	156
		puts	106	180
		ungetc	78	132
		fread	86	152
		fwrite	78	148
		fseek	22	32
		ftell	22	24
		rewind	6	26
		clearerr	2	4
		feof	4	8
		ferror	4	8
		perror	648	758
8	stdlib.h	atof	392	466
		atoi	38	82
		atol	38	82
		strtod	384	438
		strtol	34	74
		rand	4	8
		srand	6	8
		calloc	72	108
		free	22	24
		malloc	50	84
		realloc	72	116
		bsearch	24	48
		qsort	28	64
		abs	2	4
		div	134	168
		labs	6	8
		ldiv	144	172
9	string.h	memcpy	22	28
		strcpy	6	24
		strncpy	22	28
		strcat	6	24
		strncat	22	28
		memcmp	6	28
		strcmp	6	24
		strncmp	6	28
		memchr	6	28
		strchr	4	8
		strcspn	22	24
		strpbrk	22	24
		strrchr	24	48
9	string.h	strspn	22	24
		strstr	28	52
		strtok	44	48
		memset	22	30
		strerror	2	24
		strlen	2	24
		memmove	22	28

実行時ルーチンスタック使用量一覧

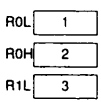
No.	ルーチン名	スタックサイズ（バイト） H8/300H ノーマルモード	H8/300H アドバンストモード
1	$ADDD$3	28	30
2	$ADDF$3	18	20
3	$ADDL$3	—	—
4	$ANDL$3	—	—
5	$BFINC$3	6	8
6	$BFINI$3	8	10
7	$BFSC$3	4	6
8	$BFSI$3	4	6
9	$BFUC$3	4	6
10	$BFUI$3	4	6
11	$CDVC$3	—	—
12	$CDVI$3	—	—
13	$CDVUI$3	—	—
14	$CMDC$3	—	—
15	$CMDI$3	—	—
16	$CMDUI$3	—	—
17	$CMLI$3	—	—
18	$CMPD$3	24	24
19	$CMPF$3	6	8
20	$CMPL$3	—	—
21	$CTOL$3	—	—
22	$DIVC$3	—	—
23	$DIVD$3	30	32
24	$DIVF$3	22	24
25	$DIVI$3	—	—
26	$DIVL$3	10	14
27	$DIVUI$3	—	—
28	$DIVUL$3	6	8
29	$DIVUX$3	—	—
30	$DIVXSB$3	—	—
31	$DIVXSW$3	—	—
32	$DIVXUW$3	—	—
33	$DSLC$3	6	8
34	$DSLI$3	6	8
35	$DSLL$3	8	10
36	$DSRC$3	6	8
37	$DSRI$3	6	8
38	$DSRL$3	8	10
39	$DSRUC$3	6	8
40	$DSRUI$3	6	8
41	$DSRUL$3	8	10
42	$DTOF$3	14	16
43	$DTOI$3	—	—
44	$DTOL$3	10	12
45	EQD3	26	28
46	EQF3	8	12
47	fp_regld3	2	4
48	fp_regsv3	18	20
49	$FTOD$3	10	12
50	$FTOI$3	—	—
51	$FTOL$3	6	8
52	GED3	26	28
53	GEF3	8	12
54	GTD3	26	28
55	GTF3	8	12
56	$ITOD$3	6	8
57	$ITOF$3	4	6
58	$ITOL$3	—	—
59	LED3	26	28
60	LEF3	8	12
61	LTD3	26	28
62	LTF3	8	12
63	$LTOD$3	10	12
64	$LTOF$3	8	10
65	$MODL$3	—	—
66	$MODUL$3	—	—
67	$MULD$3	54	56
68	$MULF$3	18	20
69	$MULI$3	—	—
70	$MULL$3	10	12
71	$MULXSB$3	—	—
72	$MULXSW$3	—	—
73	$MULXUW$3	—	—
74	$MV8$3	6	8
75	MVN3	14	18
76	NED3	26	28
77	NEF3	8	12
78	$NEGD$3	6	8
79	$NEGF$3	—	—
80	$NEGL$3	—	—
81	ORL3	—	—
82	$PODD$3	36	42
83	$POID$3	36	42
84	$PODF$3	—	—
85	$POIF$3	—	—
86	$PODL$3	—	—
87	$POIL$3	—	—
88	$PRDD$3	34	42
89	$PRDF$3	24	28
90	$PRDL$3	—	—
91	$PRID$3	34	42
92	$PRIF$3	24	28
93	$PRIL$3	—	—
94	SLI3	—	—
95	SLL3	—	—
96	sp_regld3	2	4
97	sp_regsv3	22	24
98	SRI3	—	—
99	SRL3	—	—
100	$SRUI$3	—	—
101	$SRUL$3	—	—
102	$SUBD$3	46	50
103	$SUBF$3	18	20
104	$SUBL$3	—	—
105	SWI3	—	—
106	$ULTOD$3	10	12
107	$ULTOF$3	6	8
108	$UTOD$3	6	8
109	$UTOF$3	2	4
110	$XORL$3	—	—
111	_INITSCT	16	18

引数割り付けの具体例

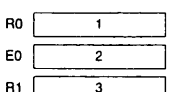
■例1

レジスタ渡しの対象の型である引数は、宣言順にレジスタER0、ER1に割り付けます。

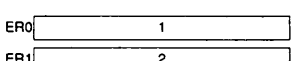
```
[1] int f(char,char,char);
        :
    f(1,2,3);
      :
```



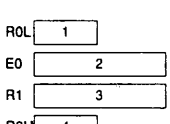

```
[2] int f(int,int,int);
        :
    f(1,2,3);
      :
```




```
[3] int f(long,long);
        :
    f(1,2);
      :
```




```
[4] int f(char,int,int,char);
        :
    f(1,2,3,4);
      :
```

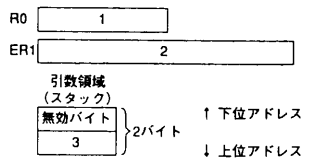


■例2

レジスタに割り付けることができなかった引数は、スタックに割り付けます。

また、引数の型がchar型で、スタック上の引数領域に割り付けた場合、下位アドレスに無効バイトができます。

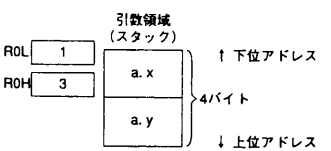
```
int f(int,long,char);
        :
    f(1,2,3);
      :
```



■例3

レジスタに割り付けられない型の引数は、スタックに割り付けます。

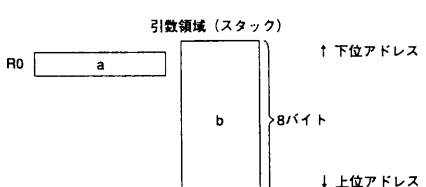
```
struct s{int x,y;}a;
int f(char,struct s,char);
      :
    f(1,a,3);
      :
```



■例4

原型宣言がない場合、char型はint型に、float型はdouble型に拡張して渡します。

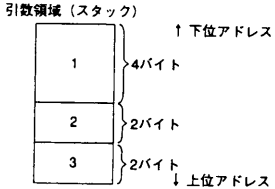
```
int f( );
char a;
float b;
        :
    f(a,b);
      :
```



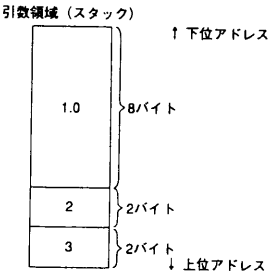
■例5

原型宣言により可変個の引数を持つ関数として宣言している場合、対応する型のない引数およびその直前の引数は、宣言順にスタックに割り付けます。

[1]
```
int f(long,...);
    :
f(1,2,3);
    :
```



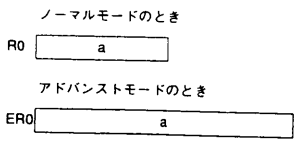
[2]
```
int f(double,int,...);
    :
f(1.0,2,3);
    :
```



■例6

ポインタ型は、ノーマルモードでは2バイト、アドバンストモードでは4バイトの領域に割り付けられます。

```
int f(int *);
    :
f(a);
    :
    :
```



■例7

関数の返す型が4バイトをこえる場合または構造体の場合、引数領域の直前にリターン値アドレスを設定します。また、構造体のサイズが奇数バイトのとき、1バイトの未使用領域が生じます。

```
struct s{char x,y,z;}a,b;
float f(struct s);
    :
f(a);
    :
    :
```