AKI-H8/3048Fマイコンボード専用
BASICコンパイラ

★マザーボード16文字2行液晶専用
**　「PRINT」関数で表示が簡単にできます。**
★「ADINP」「DAOUT」関数でAD変換・DA変換も簡単にできます。
★インラインアセンブル機能付　割り込みも使用できます。
OS:Windows95/98, Windows NT4.0(x86)

AKI－H8/3048Fマイコンボード専用 BASICコンパイラ WINDOWS98/95版

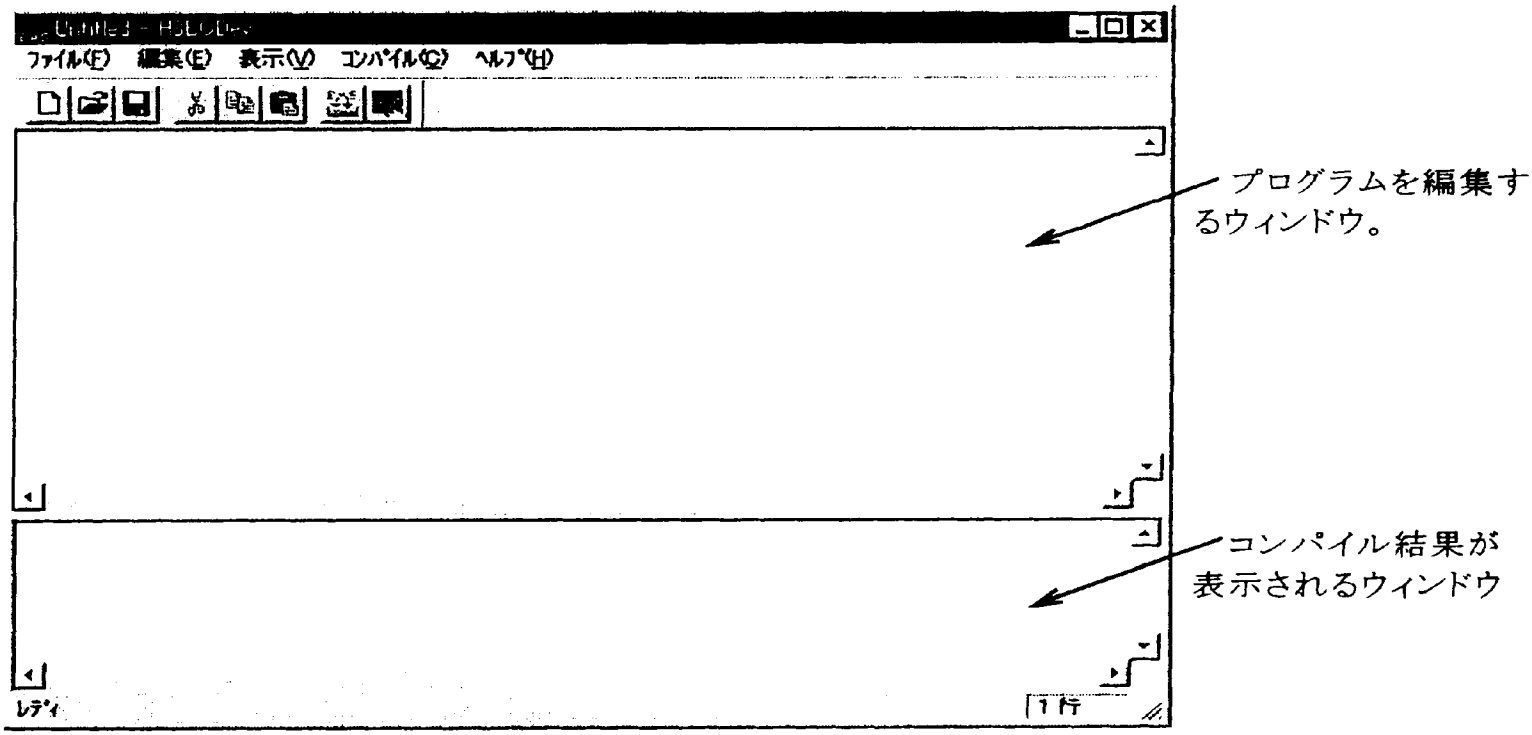
★マザーボード１６文字２行液晶専用「ＰＲＩＮＴ」関数で表示が簡単にできます
★「ＡＤＩＮＰ」「ＤＡＯＵＴ」関数でＡＤ変換・ＤＡ変換も簡単にできます。
★インラインアセンブル機能付　割り込みも使用できます。

コンパイラを走らせるのに必要なシステム：Windows 95/98, Windows NT 4.0(x86)

ファイル構成：
H8BCDev.EXE　　統合開発環境
H8BASCOM.EXE　BASICコンパイラ本体
RUNTIME.RTL　　ランタイムライブラリ
（すべて同じディレクトリにおきます。）

準備（インストール）：
フロッピーディスクにあるH8BASICというフォルダをハードディスクにコピーしてください。

使用法：
H8BASICフォルダの中にあるH8BCDev.EXEをエクスプローラ（など）から起動します。



この編集ウィンドウでBASICのプログラムを入力します。または、メニューの「ファイル」－「開く...」を選んですでに存在するBASICのプログラムファイルを開きます。

コンパイルを行うときはメニューの「コンパイル」－「コンパイル」を選びます。
書いたプログラムを保存していないときは先に「ファイル」－「名前を付けて保存...」で保存してからコンパイルを行います。

そうすると下側のウィンドウに次のような表示を出してコンパイルが開始されます。

```
------------------------------------------------------------------------
BASIC Compiler for AKI-H8(H8/3048)
(C) 1999 (有)秋月電子通商 / (株)エイペックス

コンパイル中...[BASICソースファイル名]
------------------------------------------------------------------------
```

そしてソースにエラーがなくコンパイルが終わると

```
------------------------------------------------------------------------
*** コンパイル情報 ***

START ADDRESS   : 000100

USER CODE       : 000D80 ～ 0012F1
STRING TABLE    : 0012F2 ～ 001314

VARIABLE AREA   : FFEF1A ～ FFEF35
HEAP AREA       : FFEF36 ～ FFF735

STACK INIT      : FFFF00
------------------------------------------------------------------------
```

のように表示され、MOTファイルが出力されます。
このMOTファイルはそのままROMライタープログラム(FLASH.EXE)でAKI-H8に転送することができます。

文法は Microsoft BASICを参考にしてありますので NECのPC-8001についていたN-BASIC, PC-8801, PC-9801についていたN88-BASICや 富士通のFM-8, FM-7についていたF-BASICなどに似ています。ただ、行番号は用いずに GOTO, GOSUBの宛先はすべてラベルです。
また、変数(定数)は整数(-2147483648～ 2147483647)及び文字列のみです。変数は一文字目がA～Zで二文字目以降がA～Z,0～9,_(アンダースコア)からなる文字列です(例えばA, ABC, A0, HEN_SUU等)。また、文字列変数は最後に"$"がつきます(例えば、A$, ABC$, A0$, HEN_SUU$)。大文字小文字は区別しません。
ラベルは !で始まり、二文字目がA～Z,_(アンダースコア)で三文字目以降がA～Z,0～9,_(アンダースコア)からなる文字列です。大文字小文字は区別しません。
なお、定義位置には:(コロン)を書いてからラベル名を書きます。

例:

```
        .....
:!LOOP
        A=A+3
        IF A<10000 THEN GOTO !LOOP
        .....
```

使用できる演算子は以下の通りです。

+(加算)	A= 56 + 3 でAには59が入ります。
-(減算)	A= 56 - 3 でAには53が入ります。

*(乗算)	A= 56 * 3 でAには168が入ります。
/(除算)	A= 56 / 3 でAには18が入ります。
^(べき乗)	A = 56 ^ 3 でAには175616が入ります。
MOD(剰余)	A = 56 MOD 3 でAには2が入ります。
AND(論理積)	A = 56 AND 3 でAには2が入ります。
OR(論理和)	A = 56 AND 3 でAには57が入ります。
XOR(排他的論理和)	A = 56 XOR 3 でAには55が入ります。
NOT(1の補数)	A = NOT 56 でAには -57が入ります。
-(2の補数)	A = -56 でAには-56が入ります。

また、文中にインラインアセンブラを使用することができます。
書式は次の通りです。

```
ASM BEGIN
        MOV.B   @(H'00003:16,ER0),R0L
:!LABEL
        CMP.B   #H'33:8,R0L
        BNE     H'FF212:8                       ; コメント
ASM END
```

ニーモニック、オペランドなどの基本的な書き方はH8のアセンブラに準拠します。
但し、以下のような制限があります。

- ラベルについてはBASICのラベルと同じ制限があります。即ち、!で始まり、二文字目がA～Z,_(アンダースコア)で三文字目以降がA～Z,0～9,_(アンダースコア)からなる文字列です。大文字小文字は区別しません。そして、定義位置には:(コロン)を書いてからラベル名を書きます。
- 数値を直接書くには十進数(例:235)または16進数(例:H'12AF)の書き方しか使えません。
- 条件ジャンプのオペランドに数値を書く場合は"@@"をつけずに、直接数値を書きます。

なお、インラインアセンブラ内ではBASICの変数をオペランドに用いることができます。

```
        SW = 12
ASM BEGIN
        mov.l   er0, SW
        shal.l  er0
        mov.l   SW, er0
ASM END
        PRINT SW
```

といった具合です。

割込みルーチンについて

このBASICコンパイラでは割込みルーチンを登録することができます。
ON IRQ(...) GOTO xxx
プログラムの最初でこのステートメントを用いて宣言します。
IRQ(...)のかっこの中にはベクタ番号を書きます。例えば、TRAPA #0なら8。ITUのIMIA0(チャネル0)なら24です。GOTOの次には割込みルーチンのラベルを書きます。

例えば、TRAPA #0割込みで!WARIKOMIへ飛ぶようにするなら

```
ON IRQ(8) GOTO !WARIKOMI
```

となります。
割込みルーチン内ではER0～ER6のレジスタを保存してください。そして、ルーチンの最後にはRETURNIRQを書いてください。

例:

```
:!WARIKOMI
        '割込みルーチン内では必ずレジスタをPUSHしておいてください。
        ASM BEGIN
                PUSH.L  ER0
                PUSH.L  ER1
                PUSH.L  ER2
                PUSH.L  ER3
                PUSH.L  ER4
                PUSH.L  ER5
                PUSH.L  ER6
        ASM END
        ......
        割込みルーチンの処理
        ......
        'PUSHしておいたレジスタをPOPする。
        ASM BEGIN
                POP.L   ER6
                POP.L   ER5
                POP.L   ER4
                POP.L   ER3
                POP.L   ER2
                POP.L   ER1
                POP.L   ER0
        ASM END
        '割込みルーチンから復帰
        RETURNIRQ
```

また、BASICのTIMERステートメントを用いるときは、割込みを起こすための設定をBASICのステートメントで行いますが、それ以外の時には割込みを起こさせるための設定を自分でプログラムする必要があります。
例：ＳＣＩ（１）で受信割り込みをしたい場合

```
        SCR = PEEK(&HFFFFBA)
        SCR = SCR OR &H50
        POKE &HFFFFBA, SCR
```

なおかつ、CCR(コンディションコードレジスタ)の割込みマスクビット(Iビット)を0にする必要があります。
例:

```
ASM BEGIN
        ANDC    #H'7F,CCR
ASM END
```

また、BASICのステートメントからI/Oを操作するような命令を実行した場合(POKEなどで直接操作

する場合は除く)は、そのステートメントによってI/Oの設定を操作していますので、それ以前に自分で設定したものがクリアされていることがあります。例えば、INPUT #-1, PRINT #-1などを実行した後はSCIのレジスタの設定などはこれらのステートメントによって操作されています

★サンプルプログラム　AD. BASをごらんください。

タイマーの使用について

タイマーを使用するために以下のステートメントが用意されています。
TIMERSET, TIMER ... START, TIMER ... STOP, TIMER ... CONTINUE

まずは、タイマーの通知をうけたときに飛んでいくルーチンを作成します。
このルーチンの中で必ずTIMER ... CONTINUEを書いてください。(タイマーを終わらせる場合はその代わりにTIMER ... STOPでもよいです。)
例えばタイマーチャンネル0の割込みルーチン内なら

```
TIMER 0 CONTINUE
```

です。
その他割込みルーチンでの注意については「**割込みルーチンについて**」の項を参照してください。
また、タイマーはインテグレーテッドタイマーユニット(ITU)のアウトプットコンペアAを用いていますので、割込みルーチンの登録は
チャンネル0ならON IRQ(24) GOTO ...
チャンネル1ならON IRQ(28) GOTO ...
チャンネル2ならON IRQ(32) GOTO ...
チャンネル3ならON IRQ(36) GOTO ...
チャンネル4ならON IRQ(40) GOTO ...
と、なります。

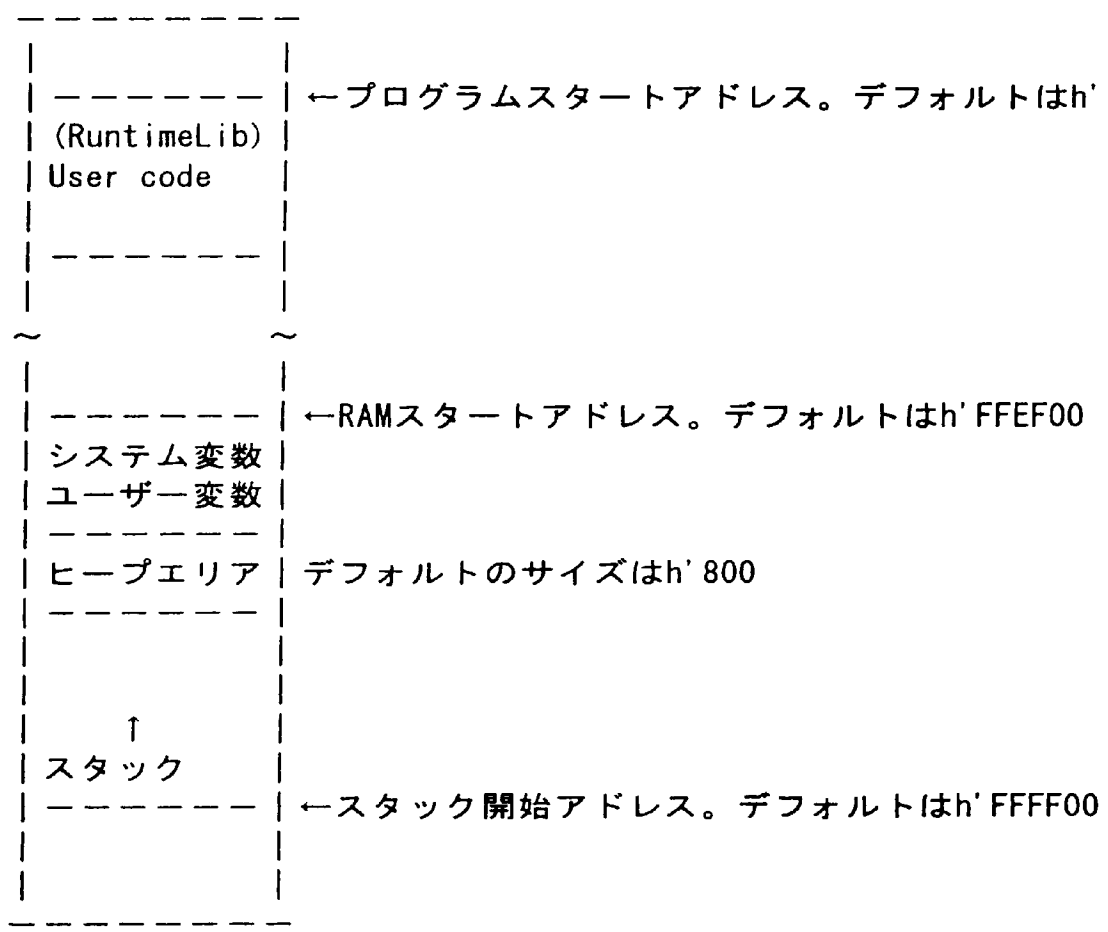
プログラムの最初でTIMERSETでチャンネルとタイマーの間隔を指定してタイマーの設定を行います。
チャンネル0で20ms毎なら

```
TIMERSET 0, 20
```

とします。このタイマーの間隔はクロックが16MHzの時で約30msが最大値となりますので注意してください。

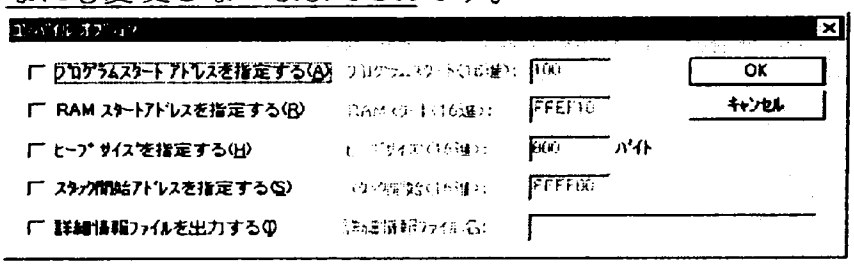
TIMER STARTでタイマーを開始し、TIMER STOPで停止します。

このBASICコンパイラで作成した実行形のメモリマップは以下のようになっています。



・プログラム開始アドレス…プログラムの開始アドレスです。このアドレスからプログラムが始まるようにコンパイルされます。
・RAMスタートアドレス…変数格納や作業に使うRAMのエリア開始アドレスです。
・ヒープサイズ…ヒープエリアのサイズを指定します。ヒープエリアは文字列操作等に使用します。
・スタック開始アドレス…スタックエリアの開始アドレスを指定します。サブルーチンの中からGOSUBを入れ子にして呼んでいる場合などここを大きくとる必要があります。

それぞれのアドレスは「コンパイル」－「設定」のダイアログで設定できるようになっています。通常はなにも変更しないままでOKです。



指定する場合には、それぞれチェックボックスにチェックをつけて右のEditTextに指定する数値等を入力します。

コンパイル時のエラー一覧

＜警告＞

演算によりオーバーフローが発生します。

演算でオーバーフローが起きています。(四則演算の結果が-2147483648～ 2147483647を越えた等)

配列の添え字が宣言範囲内にありません。

DIMによって宣言した配列変数の範囲を超えた添字で変数を使用しています。例えば、DIM A(10)と宣言した配列変数に対し A(20)=30としています。

＜エラー＞

・ファイルの読込、書込み起きたエラー

ソースファイルがオープンできません。

指定されたBASICソースファイルをオープンすることができませんでした。
指定したBASICソースファイルが存在するか確認してください。

出力ファイルの作成に失敗しました。

出力ファイルはBASICソースファイルと同じフォルダに作成されます。ディスクがいっぱいでないか、そのフォルダに書き込むことができるかのチェックをしてください。

出力ファイルの書き出し中にエラーが発生しました。

出力ファイルはBASICソースファイルと同じフォルダに作成されます。ディスクがいっぱいでないかのチェックをしてください。

ランタイムライブラリがオープンできません。

ランタイムライブラリ、ファイルRUNTIME.RTLをオープンできませんでした。
RUNTIME.RTLが同じフォルダにあるかチェックしてください。

詳細情報ファイルの作成に失敗しました。
詳細情報ファイルの書き出し中にエラーが発生しました。

詳細情報ファイルを作成するディスクがいっぱいでないか、そのフォルダに書き込むことができるかのチェックをしてください。

・文法上のエラー

文法が正しくありません。

タイプミスなどによってコンパイラが解釈できない部分があります。ステートメントや関数の打ち間違いはないかパラメータの数や型(数値か文字列か)は合っているかなどのチェックをしてください。

配列変数として定義されていません。

変数が配列変数として使用されていますが、DIMによって配列変数の宣言をされていません。プログラムの最初でDIMによる配列変数の宣言を行ってください。

配列の次元数が異なります。

DIMによって宣言した配列変数の次元数と異なる次元数で変数を使用しています。例えば、DIM A(10,20)と宣言した配列変数に対し A(0)=30とかA(0,0,0)=12とかしています。

添え字が多すぎます。

DIMで4次元を越える次元の配列変数を定義しようとしてます。

この変数はすでに別の場所で定義されています。

すでに配列宣言されている変数あるいは、通常の変数で使用している変数をDIM文で配列宣言しようとしました。

0で除算することはできません。

0を除数(割る数)としてわり算をしている箇所があります。あるいは0が底で負の数が指数の箇所があります。

このラベルはすでに別の場所で定義されています。

このラベルはすでに別の場所で定義されています。別のラベル名をつけて下さい。

ラベルが定義されていません。

GOTO, GOSUB, RESTORE等で指定されたラベルが見つかりません。

対応するWHILE文が見つかりません。

現れたWENDに対応するWHILEが見つかりません。

対応するWEND文が見つかりません。

現れたWHILEに対応するWENDが見つかりません。

対応するFOR文が見つかりません。

現れたNEXTに対応するFORが見つかりません。

対応するNEXT文が見つかりません。

現れたFORに対応するNEXTが見つかりません。

DATA文が見つかりません。

RESTORE文で指定されたラベルにDATAがありません。

添え字は定数で指定してください。

DIM文による配列変数の宣言で、配列の個数は定数で指定してください。

・インラインアセンブラで起きるエラー

オペランドサイズを指定してください。

```
CMP.B    #H'33:8,R0L
```

のように:8や:16や:24のオペランドサイズの指定を行ってください。

オペランドサイズはありません。

:8や:16や:24のようなオペランドサイズ指定は必要ありません。

オペランドサイズが正しくありません。

:8や:16や:24のようなオペランドサイズ指定が正しくありません。

オペランドの個数が違います。

オペランドの個数が正しくありません。

オペランドが正しくありません。
　　このオペコードは書かれているようなオペランドをとりません。

・実行時のエラーについて
実行時にエラーが発生したときは、変数 ERRにエラーコードが入ります。

＜続行可能なエラー＞
以下のエラーが発生したときはERRにエラーコードを代入し、実行を続行します。

0	エラーなし。通常はこの値が入っています。
1	オーバーフロー（四則演算の結果が-2147483648～ 2147483647を越えた、設定する時間が設定できる範囲を超えた等）
2	READでDATAのタイプが合わない（READ A$としているのにDATAの数値を書いた場所をREADしようとしている等）
3	SCI受信オーバーランエラー
4	SCI受信フレーミングエラー
5	SCI受信パリティエラー

＜続行不可能なエラー＞
以下のエラーが発生したときは、ERRに値を代入したあとでSLEEPします。

256	0で除算しました
258	メモリアロケーションでブロックIDが壊れている（POKE, POKEWやマシン語ルーチンでRAM上の使用している部分を破壊した可能性があります）
259	メモリが足りない
260	SCIが初期化されてない(通信フォーマットが未設定)(SERIALSETをせずに、PRINT #やINPUT#を実行しました）

予約語

以下の語はコンパイラによって予約されていますので、変数名等として使用することができません。

,

ABS	HEX$	POKE	STOP
ADINP	IF	POKEW	STR$
AND	INPUT$	PORTSET	STRING$
ASC	INSTR	PRINT	THEN
CALL	IRQ	RANDOMIZE	TIMER
CHR$	LEFT$	READ	TO
CONSOLE	LEN	REM	TIMERSET
CONTINUE	LET	RESTORE	VAL
DAOUT	MID$	RETURN	VARPTR
DATA	MOD	RETURNIRQ	WAIT
DIM	NEXT	RIGHT$	WHILE
ELSE	NOT	RND	WEND
END	ON	SERIALSET	XOR
FOR	OR	SPACE$	
GOSUB	PEEK	START	
GOTO	PEEKW	STEP	

☆☆☆

Q.『出力ファイルの作成に失敗しました』のエラーが出るのですが、ディスクはまだたくさん空き容量があります。
A. フロッピーディスクやハードディスクのルート(A:¥やC:¥)で作業をしていませんか？ルートに入るファイルの数は限りがあるので、必ずフォルダを作成して、そこで作業を行ってください。

Q. コンパイルしようとするとファイル保存のダイアログが出るのですが...
A. コンパイルするためにはソースが必ずファイルに保存されていることが必要となります。まだ、ファイル名がついていない場合はコンパイルする前に一度ファイル名を指定して保存することが必要となります。
ファイルから読み込んだときや一度セーブしたときはコンパイルのたびに自動的に保存されます。

Q. SCIで割込みを起こすようにプログラミングしたのですが、割込みが起きません。
A. PRINT #-1やINPUT #-1を使っていませんか？これらのBASICステートメントはSCIのレジスタの設定をいじるので、これらのBASICステートメント実行後は割込みビット等がクリアされている場合があります。

クロック周波数について

Runtime.RTLの3バイト目から「00003E80」という4バイトのデータが入っています。これを元にタイマーやシリアルコミニュケーションインターフェースのボーレート用に設定する値を計算しています。
これはクロック周波数のkHzを16進の整数で表した値です。初期の値は H'00003E80=16000 [kHz]になっています。H8のクロック周波数を変更したときはここの値も変更することによってBASICコンパイラも対応させることができます。
例えば、H8のクロック周波数を19.6608[MHz]に変更した場合は、H'00004CCC(=19660)に変更することになります。
変更をする場合は、必ずRuntime.RTLのバックアップをとってから、バイナリエディタまたは debug コマンドで行ってください。

ステートメント一覧

CALL

機能:マシン語ルーチンを呼出します。
使用法:CALL <マシン語ルーチンアドレス>
例:

```
CALL &H02000     'H`02000番地からのルーチンを呼ぶ
```

CONSOLE

機能:PRINT命令で出力される表示方法を指定します。
使用法: CONSOLE モード,[表示行]
モード: 0…PRINT命令における表示でLCDを二行使います。
　　1…PRINT命令における表示でLCDを一行ずつ使います
表示行: 0…モード1においてPRINT命令で一行目を使います
　　1…モード1においてPRINT命令で二行目を使います

例1:0から100までの数字がLCD 2行にスクロールさせて表示します。

```
CONSOLE 0                          '2行モードに
FOR KAZU=0 TO 100
        MOJI$ = STR$(KAZU) + " "
        DISP$ = DISP$ + MOJI$    '表示していた文字列の最後にくっつける
        DISP$ = RIGHT$(DISP$, 32)
        PRINT DISP$
NEXT
```

例2:LCD一行目に"H8BASIC"の文字が二行目には0から100の数が順に表示します。

```
CONSOLE 1,0      ' 一行目に表示
PRINT "H8BASIC"
CONSOLE 1,1      ' 二行目に表示
FOR KAZU = 0 TO 100
        PRINT KAZU
        WAIT 100
NEXT
```

★この表示はＡＥ－Ｈ８ＭＢの１６文字×２行液晶に対応しています。
２０文字４行液晶の場合は、２行のみの表示になります。

DAOUT

機能:D/A出力に信号を出力します
使用法: DAOUT <チャンネル>,<出力値>
チャンネル:D/A変換チャンネル。0または1
出力値:チャンネルに出力するレベル。(0～255, -1で出力禁止)
注:チャンネルに0,1以外の数値を指定した場合でもエラーにはなりません。(チャンネル1として動作します。)

例:

```
FOR LEVEL= 0 TO 255
        DAOUT 0, LEVEL
NEXT
DAOUT 0, -1
```

DATA

機能:READ命令で読み込む数値、文字定数の並びを指定します。

例:

```
FOR COUNT = 1 TO 2
        READ NAME$, TEL$, AGE
        PRINT NAME$, TEL$, AGE
NEXT

DATA "YAMADA Tarou", "000-111-2222", 20
DATA "IWAKI Masami", "000-110-4444", 21
```

DIM

機能:配列変数を定義します。最大で4次元までの配列変数を確保できます。

例:2次元配列にインデックス同士の積を入れます。

```
DIM A(10,10)

FOR X=0 TO 9
        FOR Y=0 TO 9
                A(X,Y) = X*Y
        NEXT
NEXT
```

END

機能:プログラムを終了します。

例:

```
SEED=1
GOSUB !SQUARE
SEED = 32
GOSUB !SQUARE

END     ' ここに"END"を入れないとサブルーチンに入ってしまう。

:!SQUARE
PRINT SEED * SEED
RETURN
```

FOR ... TO ... STEP, NEXT

機能:指定回数だけ処理を繰り返して実行します。
STEPは省略するとSTEP 1と同じ意味を持ちます。また、NEXTの次の変数名は省略することができます。

例:

```
FOR X=0 TO 90 STEP 9
        FOR Y=9 TO 1 STEP -1
                PRINT X,Y
        NEXT Y
NEXT X
```

GOSUB

使用法:GOSUB <ラベル名>
機能:プログラム中のサブルーチンを呼び出します。

例:

```
FOR A=1 TO 10
        GOSUB !DISP      '表示サブルーチンを呼び出します。
NEXT
END


:!DISP
        MOJI$ = SPACE$(16) + STR$(A)

        FOR I=1 TO 17
                ' I文字目から表示
                DSP$ = MID$( MOJI$, I, 16 )
                PRINT DSP$
                WAIT 100
        NEXT I
        RETURN
```

GOTO

使用法:GOTO <ラベル名>
機能:指定されたラベル名の行に流れを分岐します。

例:

```
        I$="1"
:!LOOP
        PRINT I$
        I$ = I$ + "0"
GOTO !LOOP
```

IF ... THEN ... ELSE

使用法: IF <論理式> THEN <文> [ELSE <文>]
機能:論理式の条件によって結果が真ならばTHEN以下の文を、偽ならばELSE以下の文を実行します。ELSE以下は省略することもできます。

例:

```
SW = PEEK(&HFFFFC3)
DISP$=""
FOR I=0 TO 7
        ' SWのビットIが立っていたら"0"そうでなければ"1"
        IF SW AND 2^I THEN MES$="0" ELSE MES$="1"
        DISP$=DISP$+MES$
NEXT
PRINT DISP$
```

ON ... GOSUB

使用法:ON <式> GOSUB <ラベル名>,[ラベル名].....

機能:式の値によって GOSUB命令で呼び出すサブルーチンを変えます。<ラベル名>は<式>の値が1に対応するものを順に並べます。<式>の値が0だったり、対応するラベル名がない場合にはサブルーチンは呼び出されません。

例:

```
!LOOP
READ KOUMOKU
'DATAの最初の値により、サブルーチンを呼び分ける。
ON KOUMOKU GOSUB !PITCHER, !BATTER
IF KOUMOKU THEN GOTO !LOOP
END

:!PITCHER
READ NAME$, WIN, LOSE, SAVE
PRINT NAME$;" ";WIN;" ";LOSE;" S";SAVE
RETURN

:!BATTER
READ NAME$, AVR, HR, SB
PRINT NAME$;" .";AVR;" HR:";HR;" SB";SB
RETURN

****
DATA 1, "NOGUSI", 14, 9, 0,
DATA 2, "MOMURA", 280, 13, 26
DATA 2, "SUSUKI", 337, 16, 3
DATA 0
```

ON ... GOTO

使用法:ON <式> GOTO <ラベル> [, ラベル.....]

機能:式の値によって GOTO命令で飛ぶ分岐先を変えます。<ラベル>は<式>の値が1に対応するものを順に並べます。<式>の値が0だったり、対応するラベル名がない場合には分岐しません。

例:

```
'  Nの値により、座標が動く。
ON N GOTO !UE, !SHITA, !MIGI, !HIDARI

:!UE
Y = Y - 1
GOTO !MOVENEXT

:!SHITA
Y = Y + 1
GOTO !MOVENEXT

:!MIGI
X = X + 1
GOTO !MOVENEXT

:!HIDARI
X = X - 1
GOTO !MOVENEXT

:!MOVENEXT
        :
END
```

ON IRQ(...) GOTO

使用法:ON IRQ() GOTO <ラベル>

機能:割込みが発生したときにジャンプするルーチン(Interrupt Service Routine)を指定します。

例:

```
ON IRQ(8) GOTO !WARIKOMI

        :

:!WARIKOMI
        ......
        割込みルーチンの処理
        ......
        '割込みルーチンから復帰
        RETURNIRQ
```

POKE, POKEW

使用法:POKE <アドレス>, <書き込むデータ>
機能:アドレスを指定して、直接そこにデータを書き込みます。
POKEは 8bitsデータを指定したアドレスに書き込みます。
POKEWは 16bitsデータを指定したアドレスと指定したアドレス+1に書き込みます。

例:

```
POKE &HFFFFC0, &HFF      ' ポート1を出力に設定
POKE &HFFFFC1, &HFF      ' ポート2を出力に設定
POKEW &HFFFFC2, &H8632   ' ポート1と2に一度に出力データ設定
```

PRINT

機能:LCDに数値、文字列を表示します

例:

```
REM くっつけて出力
PRINT "VALUE=";VALUE
REM 間隔をとって出力（タブ）
PRINT "VALUE=", VALUE,
```

★このＰＲＩＮＴ文はＡＥ－Ｈ８ＭＢの１６文字×２行液晶に対応しています。２０文字４行液晶の場合は、２行のみの表示になります。

PRINT

使用法:PRINT <チャンネル指定>, <数式・文字列等>...
チャンネル指定には #-0 でチャンネル0、#-1でチャンネル1を指定します。
数式・文字列をシリアルコミニュケーションインターフェースに出力します。
例:

```
PRINT #-1, "1から100までの数を当ててください。";CHR$(13);CHR$(10)
```

RANDOMIZE

機能:乱数系列の初期値を設定します。

例:

```
FOR ADR = 0 TO &h100
        SEED = SEED + PEEKW(ADR)
NEXT
FOR ADR = &HFFEF00 TO &HFFFF00
        SEED = SEED + PEEKW(ADR)
NEXT
RANDOMIZE SEED
```

READ

使用法:READ <変数>
機能:データ命令で記述したデータを変数に読み込みます。
　　ＲＥＡＤ命令で読めるデータは１度に１つです。
例:

```
FOR COUNT = 1 TO 2
        READ NAME$
        PRINT NAME$
NEXT

DATA "YAMADA Taro"
DATA "IWAKI Masami"
```

REM

機能:この命令以降に注釈を書けます。'(クォーテーション)を代わりに使用することもできます。

例:

```
REM このように注釈を書くことができます。
' シングルクォーテーションでもOKです。
```

RESTORE

使用法:RESTORE <ラベル>
機能:READ命令で次に読むべきDATA命令の位置を指定します。

例:

```
RESTORE !NAME_DATA
FOR COUNT = 1 TO 2
        READ NAME$
        PRINT NAME$
NEXT

DATA "他のデータ", 30, 300, 30
:!NAME_DATA
DATA "YAMADA Taro"
DATA "IWAKI Masami"
```

RETURN

機能:サブルーチンから復帰します。

例:

```
REM LCD表示サブルーチン
:!DISP
        MOJI$ = SPACE$(16) + STR$(A)

        FOR I=1 TO 17
                ' I文字目から表示
                DSP$ = MID$( MOJI$, I, 16 )
                PRINT DSP$
                WAIT 100
        NEXT I
        RETURN
```

RETURNIRQ

機能:ON IRQ(...) GOTOで飛んだルーチン(割込み処理ルーチン)から復帰します。

例:

```
:!WARIKOMI
        ......
        割込みルーチンの処理
        ......
        '割込みルーチンから復帰
        RETURNIRQ
```

SERIALSET

使用法:SERIALSET <チャンネル>, <ボーレート>, <キャラクタ長>, <ストップビット>, <パリティ>
チャンネル:0,1
ボーレート:300, 600, 1200, 2400, 4800, 9600, 19200, 31250, 38400
キャラクタ長:7, 8
ストップビット:1, 2
パリティ:0…パリティなし、1…偶数パリティ、2…奇数パリティ
機能:シリアルコミニュケーションインターフェースの初期設定をします。
パラメータが設定できる値でなかった場合にはコンパイル時にエラーになります。パラメータに変数は使えません。

例:

```
' シリアルコミニュケーションインターフェース・チャンネル1の設定
' 9600ボー、8ビット、1ストップビット、パリティなし
SERIALSET 1, 9600, 8, 1, 0
```

TIMER CONTINUE

使用法:TIMER <チャンネル> CONTINUE
機能:タイマーを引き続き利用するには、タイマー割り込みルーチンの中には必ずこのステートメントを書きます。

例:

```
REM タイマー割込みルーチン
:!TIMERWARIKOMI

        :
'タイマ割込みルーチン内では必ずこのステートメントを書く。
TIMER 0 CONTINUE

COUNT = COUNT + 1
IF COUNT >= 50 THEN SEC = SEC + 1:COUNT = 0
IF SEC >= 60 THEN MINUTES = MINUTES + 1:SEC = 0
IF MINUTES >= 60 THEN HOUR = HOUR + 1:MINUTES = 0
IF HOUR >= 24 THEN HOUR = 0

:

RETURNIRQ
```

TIMER START

使用法:TIMER <チャンネル> START
機能:タイマー割込みを開始します。

例:

```
'タイマーの設定
TIMERSET 0, 20
IF ERR <> 0 THEN PRINT "ERROR!";ERR:END

'タイマースタート
TIMER 0 START
```

TIMER STOP

使用法:TIMER <チャンネル> STOP
機能:タイマー割込みを停止します。

例:

```
'タイマー終了
TIMER 0 STOP
END
```

TIMERSET

使用法 TIMERSET <チャンネル>, <設定間隔[ms]>

機能:タイマーの時間の設定を行います。チャンネルは0～4を指定します。

チャンネルは0～4の数を直接指定することしかできません。それ以外の場合はエラーになります、また、設定間隔が設定できる範囲を超えたときは実行時にエラーになります。(変数ERRにエラーコードが設定されます)

例:

```
'タイマーの設定
TIMERSET 0, 20
'エラーが発生していなければ
IF ERR <> 0 THEN PRINT "ERROR!";ERR:END

'タイマースタート
TIMER 0 START
```

WAIT

使用法:WAIT <停止する時間>

機能:一定時間処理を停止させる。時間の指定はミリ秒単位。0[ミリ秒]～2147483647[ミリ秒](約24日)の設定が可能です。停止する時間が設定できる範囲を超えたときは実行時にエラーになります。(変数ERRにエラーコード1が設定されます)

例:

```
:!LOOP
PRINT HOUR;":";RIGHT$("0" + STR$(MINUTES), 2);":";RIGHT$("0" + STR$(SE
    C), 2)
WAIT 100

GOTO !LOOP
```

WHILE ... WEND

使用法:WHILE <論理式>

[繰り返される処理]

WEND

機能:論理式の値が真である間、処理を繰り返し実行します。

例:

```
WHILE A < B*B
        PRINT B
        B = B + 1
WEND
```

関数一覧

ABS

書式:ABS(数式)
機能:引き数の絶対値を返します。

例:
```
Z = ABS(-5)
```

ADINP

書式:ADINP(数式)
機能:引数のチャンネルのA/D変換を行い、その結果を返します。数式は0～7
単一モードで動作します。返値は上位10ビットが有効です(下位6ビットは常に0)。
また、引数は下位3ビットのみを見ています。つまり、0～7の範囲を超えた場合、例えば
8なら0を指定したのと同じになります。

例:配列変数に100msごとにA/D変換したデータを入れます。
```
DIM     ADATA(100)

FOR INDEX = 0 TO 99
        ADATA(INDEX) = ADINP(0)
        WAIT 100
NEXT
```

ASC

書式:ASC(文字式)
機能:文字のASCIIコードを返します。

例:
```
MOJI$="A"
CODE = ASC(MOJI$)
```

CHR$

書式:CHR$(数式)
機能:引き数のASCIIコードの文字を返します。

例:改行コード(13)をPRINTします。
```
PRINT CHR$(13)
```

HEX$

書式:HEX$(数式)
機能:数値を16進表記とした文字列を返します。

例:
```
ADRS=1000
PRINT HEX$(ADRS)
```

INPUT$

書式:INPUT$(チャンネル指定, 数式)
チャンネル指定には #-0 でチャンネル0、#-1でチャンネル1を指定します。
機能:数式で指定した文字数分シリアルコミニュケーションインターフェースから文字を読んでその文字列を返します。

例:

```
:!LOOP_LINEINPUT
        A$=INPUT$(#-1, 1)
        IF A$=CHR$(13) THEN GOTO !END_LINEINPUT
        PRINT #-1, A$
        LINEIN$=LINEIN$+A$
        GOTO !LOOP_LINEINPUT
:END_LINEINPUT
```

INSTR

書式:INSTR(数式, 文字式1, 文字式2)
機能:文字式1の中から文字式2を探してその開始位置を先頭の文字を1として返します。数式は文字列1の中での検索開始文字位置を指定します。

例:"AKI"の文字のある位置をTARGET$の先頭から探します。

```
C = INSTR(1, TARGET$, "AKI");
```

LEFT$

書式:LEFT$(文字式, 数式)
機能:文字式の左端から数式の長さの部分文字列を取り出します。

例:"00A0"の左から2文字をPRINTします。

```
PRINT LEFT$("00A0", 2)
```

LEN

書式:LEN(文字式)
機能:文字列の長さを返します。

例:

```
S$ = "Length of this sentence."
L = LEN(S$)
```

MID$

書式:MID$(文字式, 数式1, 数式2)
機能:文字式の数式1文字目から数式2文字分だけ部分文字列を取り出します。

例:

```
S$ = "Ende good ist alles good"
PRINT MID$(S$, 4, 2)
```

PEEK

書式:PEEK(数式)
機能:数式で指定したアドレスの1バイトのデータを返します。

例:
```
D = PEEK(&HFFFFC2)
```

PEEKW

書式:PEEKW(数式)
機能:数式で指定したアドレスからの2バイトのデータを返します。

例:
```
DW = PEEKW(&HFFFFC2)
```

RIGHT$

書式:RIGHT$(文字式, 数式)
機能:文字式の右端から数式の長さの部分文字列を取り出します

例:
```
PRINT RIGHT$("00A0", 2)
```

RND

書式:RND(数式)
機能:0～数式-1の間の一様乱数を返します。

例:1～6の間で乱数を発生させます。
```
SAIKORO = RND(6) + 1
```

SPACE$

書式:SPACE$(数式)
機能:数式個の空白文字列を返します。

例:
```
COUNTRY$ = "Japan"
COUNTRY = 47
PRINT COUNTRY$;SPACE$(10 - LEN(COUNTRY$));COUNTRY
```

STR$

書式:STR$(数式)
機能:数式の値を文字式にして返します。

例:
```
S=10
PRINT "00" + STR$(S)
```

STRING$

書式:STRING$(数式, 文字式)
機能:文字式を数式会だけ繰り返した文字式を返します。

例:

```
COUNTRY$ = "Japan"
COUNTRY = 47
PRINT COUNTRY$;STRING$(10 - LEN(COUNTRY$), ".");COUNTRY
```

VAL

書式:VAL(文字式)
機能:文字式を数値になおして返します。

例:

```
A$ = "6"
SUM = VAL(A$) + 12
```

VARPTR

書式:VARPTR(変数)
機能:変数の値が格納されているアドレスを返します。