第3章
HuBASICの内部構造

原則として，この章はディスク版NEW BASICに基づいて書かれています．各BASICで異なる場合は，その旨，表記してありますが，基本的には構造は同じです．

3-1 HuBASICの種類

いままでにX1に添付されてきたBASICには次の5種類があります．

CZ-8CB01　V1.0　最初のテープ用BASIC
CZ-8FB01　V1.0　最初のディスク用BASIC
CZ-8CB01　V2.0　テープ用NEW　BASIC
CZ-8FB01　V2.0　ディスク用NEW　BASIC
CZ-8FB02　V1.0　漢字HuBASIC(turboBASIC)

このうち，漢字HuBASICはX1turboシリーズでしか使えませんが，他のBASICはすべてのX1シリーズで起動できます．漢字HuBASICはturbo用の拡張機能をそなえているばかりでなく，高度な日本語処理を行えるようになっています．

NEW BASICは，フリーエリアの拡張をはかるために，旧BASIC(V1.0)の中で使用頻度の低かった命令や，他の命令で代用できる命令を削除しました．また，必要とする命令に応じて10段階にBASICの大きさを指定できるなどの特徴を持っています．

3-2 HuBASICメモリーマップ

HuBASIC(ver1.0)とturbo版HuBASICのメモリーマップを図3-1に示します．ただし，ディスクBASICとテープBASICでアドレスが異なる場合にはカッコ内にテープBASICの値を示しました．

図中の各エリアについて以下に説明します．

- **IOCS(Input　Output　Control　System)**
  その名称の通りハードウェアに直接関係するデータの出入力を管理します．
- **BASICテキストの先頭アドレス**
  LIMIT，CLEAR命令により変更することができます．また，NEW ONレベルに応じて変化します．終了アドレスはBASICテキストの長さによります．
- **変数エリア**
  数値変数の場合は種類，変数名と値そのものが入ります．文字変数の場合は種類，変数名，文字列の長さと文字列が格納されているアドレスを知るためのポインタが入っています．

●ファイル用ストリングバッファ
周辺装置とデータをやりとりするときの一時的なバッファ256バイトとデータの入出力先や処理アドレスの入った部分16バイトからなっています．このエリアの大きさは，MAXFILES 命令で指定した最大ファイル数により変化します．最大ファイル数がｎのときこのエリアの大きさは(ｎ＋1)×(256+16)バイトとなります．

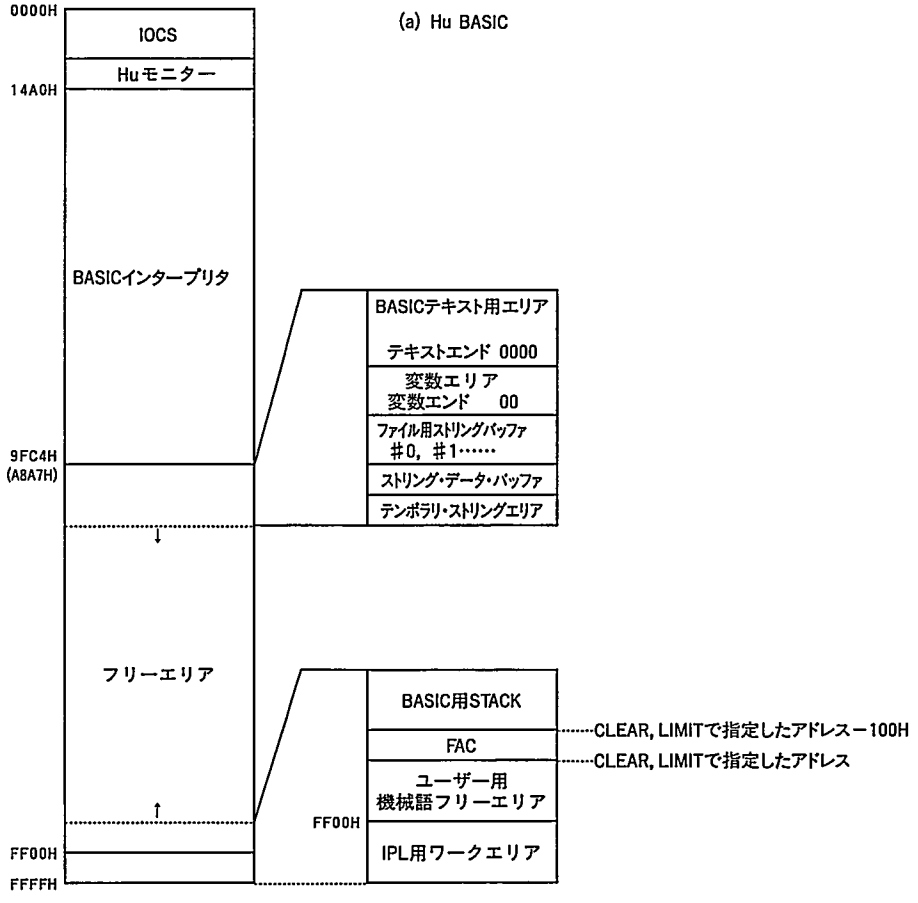
●ストリングデータエリア
文字変数の実際の文字列データが入るエリアです．

●テンポラリストリングバッファ
文字変数の多重処理，ファイルからの入力時に一時的に使われるエリアです．

● BASIC 用スタック
GOSUB 文の戻り番地などを保存するエリアです．

● FAC(Floating　Accumulator)
FAC は BASIC インタープリタのアキュムレータで，すべての計算はここを中心にして行われます．X1 では 100H(256バイト)と決められています．USR 関数で渡されたデータや浮動小数点の代入，演算は FAC に転送され，FAC を経由して処理されます．FAC の先頭番地は CLEAR 命令や LIMIT 命令により決定されます．

● FF00H～EFFFH は，汎用のワークエリアです．FF00H からはキー入力バッファやファイルのインフォメーションバッファとして，FFFFH からはスタックとして使われます．



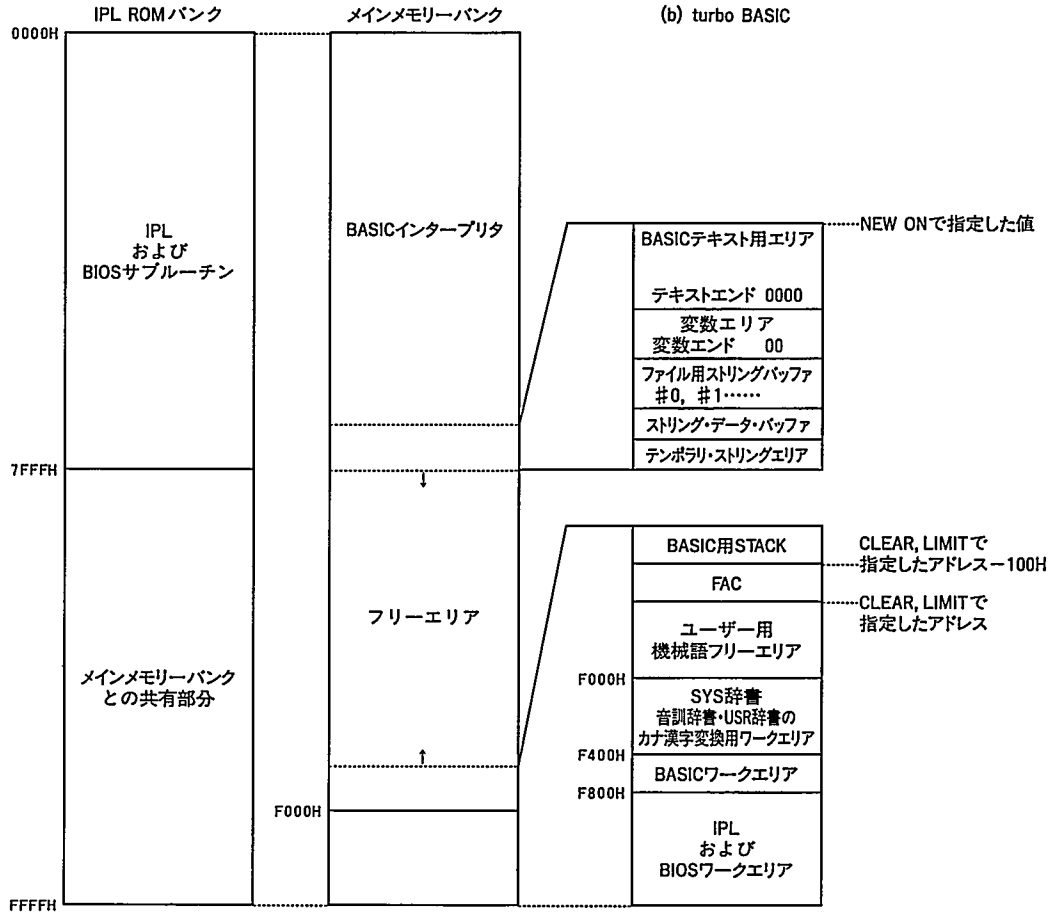


図3-1 Hu BASICメモリーマップ

内　　容	格納アドレス カッコ内は DISK BASIC	初　期　値		備　　考
		BASIC起動前	起 動 後	
BASICテキスト・トップ	9D52 (A635)	9FC5	9FC5 (A8A8)	NEW ON命令で変えられる
変数エリア・トップ	9D46 (A629)	A0EF	9FC7 (A8AA)	VARPTR命令はこのエリア内のアドレスを示す
ファイル用ストリング・バッファトップ	9D48 (A62B)	A0FC	9FC8 (A8AB)	STRPTR命令で値を見ることができる
ストリング・データ・バッファトップ	35F6 (3628)	A31C	A1E8 (ABDB)	
テンポラリ・ストリングエリア・トップ	9D4A (A62D)	A3D6	A1EA (ABDD)	
フリーエリア・トップ	35EF (3621)	A3D6	A1EA (ABDD)	
FACトップ	9D4C (A62F)	FD00	FE00 (FE00)	CLEAR, LIMITのアドレス−100Hである
ユーザー用機械語フリーエリアトップ	9D4E (A631)	FE00	FF00 (FF00)	CLEAR, LIMIT命令で変えることができる
IPL用ワークエリアトップ	9D50 (A633)	FF00	FF00 (FF00)	半固定

表3-1 HuBASICのワークエリアと初期値

IOCS, インタープリタ部分及びFF00H以降のIPLなどのワークエリアはメモリーマップに示される通りですが，BASICテキストエリアや変数エリアは状況によって変化します．そのための，各々の領域の場所を示すポインタがワークエリアの中にあります．このポインタのアドレスと各BASIC起動時における初期値を表3-1に示します．

また，NEW BASICにおける，各NEW ONレベルのBASICテキストスタートアドレスを次表に示します．

NEW ONレベル	0	1	2	3	4	5	6	7	8	9
TEXT先頭番地	7F96	919E	9449	9BB4	9F43	A3A8	A4CC	A587	A6C1	A914

表3-2　BASICテキストスタートアドレス

3-3　プログラムの格納状態

BASICプログラムは，行番号順に格納されています．BASICテキストの格納開始アドレスは，BASICのバージョンやNEW ONレベルにより変化します．それぞれの行は以下に示すような形式で格納されています．

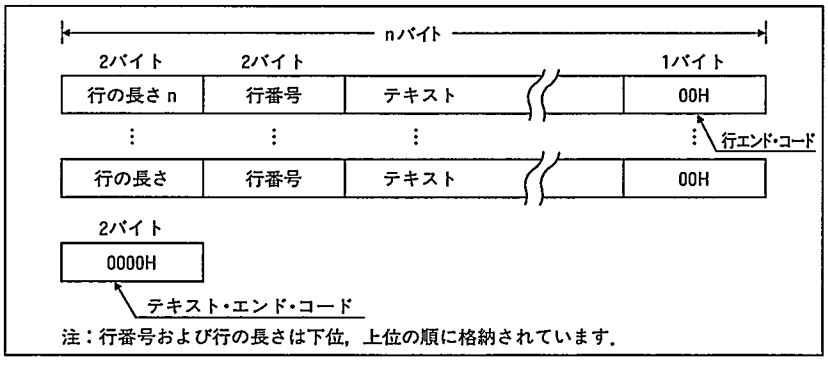


図3-2　テキストの格納形式

プログラムの終りにはテキストエンドコードとして0000Hが書き込まれていて，後に続く文がないことを示します．

次に，プログラム／テキストがどのように格納されているか見てみましょう．例として次のプログラムを使います．

これを，モニタのDコマンドでダンプしたリストを続いて示します．

リスト3-1

```
10 PRINT "ABCDEF"
20 END
```

```
:A8A8=0F 00 0A 00 8F 20 22 41 /.... / "A
:A8B0=42 43 44 45 46 22 00 06 /BCDEF"..
:A8B8=00 14 00 98 00 00 00 00 /........
:A8C0=00 4F 3A 69 00 00 00 00 /.O:i....
:A8C8=00 00 00 00 00 00 00 00 /........
:A8D0=00 00 00 00 00 00 00 00 /........
:A8D8=00 00 00 00 00 00 00 00 /........
:A8E0=00 00 00 00 00 00 00 00 /........
:A8E8=00 00 00 00 00 00 00 00 /........
:A8F0=00 00 00 00 00 00 00 00 /........
:A8F8=00 00 00 00 00 00 00 00 /........
:A900=00 00 00 00 00 00 00 00 /........
:A908=00 00 00 00 00 00 00 00 /........
:A910=00 00 00 00 00 00 00 00 /........
:A918=00 00 00 00 00 00 00 00 /........
:A920=00 00 00 00 00 00 00 00 /........
```

これを見るとわかるように，プログラムはLISTのままの形でメモリーに格納されているわけではなく，できるだけメモリーをつかわないように工夫して格納されています．

3-4 中間言語

前の節で，BASICプログラムのテキストが圧縮された形で格納されていることがわかりました．これはBASICのキーワード(PRINT, ENDなど)を1バイトか2バイトの中間言語で表しているからです．

このPRINT, ENDといったキーワードは予約語と呼ばれ，BASIC言語に，あらかじめ登録されています．その一覧は中間言語の順に並べられています．各BASICにおける予約語テーブルアドレスを次に示します．

NEW BASIC(DISK)	267BH～2A02H
旧BASIC(TAPE)	28F6H～2CDDH
旧BASIC(DISK)	2921H～2D0BH
turbo版	5741H～5B80H

表の構造は，予約語を羅列したものとなっており，各予約語の終りは，次の予約語と区別するために，最後の文字の最上位ビットがONになっています．また，中間言語テーブルは3つに分かれていて，最初が基本的なBASICの命令，次に拡張されたBASICの命令，最後に関数，システム変数となっています．それぞれの区切りには0FFHがおかれています．

基本的なBASIC命令の中間言語は080Hからはじまる1バイト，拡張命令は0FEH+080Hから始まる1バイトの計2バイト，関数は0FFH+080Hからの1バイトの計2バイトという構成になっています．

3-5 変数テーブル

テキストエリアの後に変数テーブルと呼ばれる，変数の値を格納しておくエリアが用意されています．ただし，プログラムを実行して実際にその変数が使われないと，変数領域に登録されません．

3-5-1　単純変数テーブル

変数の種類，変数名の長さによって必要なバイト数は異なりますが，各変数とも同じような形式で格納されます．文字型変数を除いて，1バイト目で示される変数の種類の値はそのまま変数データの格納バイト数も表しています．

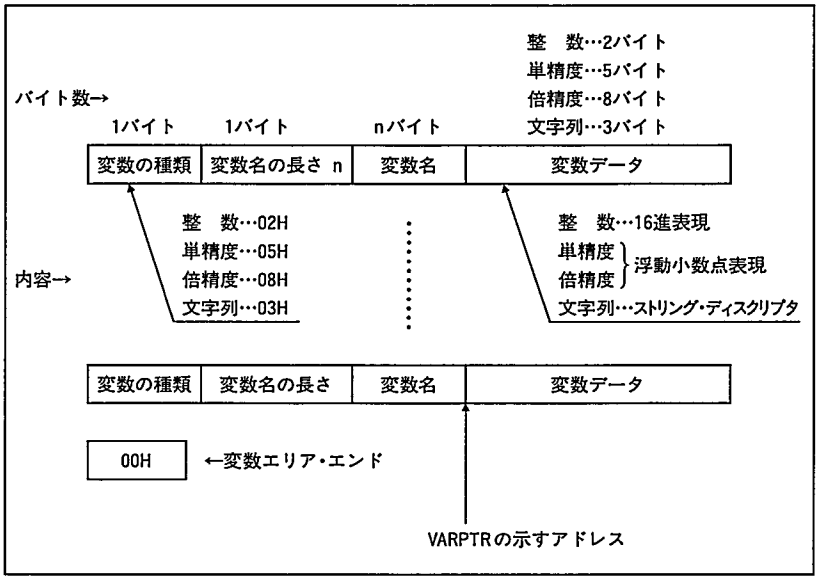


図3-3　変数の格納形式

変数データは，整数型変数の場合は数値がそのまま16進数2バイトで入っており，単精度，倍精度型変数のときには次節で説明する浮動小数点表記で格納されています．また，文字変数の場合は，他の変数型と異なり，直接このエリアにデータは格納されていません．他の変数型のデータ格納部分にあたる部分にはストリングディスクリプタといわれるものが格納されていて，このデータを用いて，実際のストリングデータエリアのアドレスを計算します．ストリングディスクリプタとは文字変数の値(文字列)の格納されているストリングデータエリアの先頭アドレスを0としたとき，何バイト離れたところに文字列の先頭があるかを示す値です．ストリングデータエリアの先頭アドレスはSTRPTR変数に入っているので，実際に文字列が格納されたアドレスは，STRPTR変数の値にストリングディスクリプタを加えることによって求められます．

NEW BASICの場合はSTRPTR変数が削除されていますので，直接BASICの内部ポインタである，7D2AHを参照することによって，ストリングデータエリアの先頭番地を求めて下さい．

では，実際にどのように変数が格納されているかを見てみましょう．次の図は，NEW BASIC DISK版で，NEW ON 9の状態で実験したものです．

リスト3-2

```
10 a%=10
20 a!=10
30 a#=10
40 a$="10"
```

```
:A940=00 00 00 02 01 41 0A 00
:A948=05 01 41 84 20 00 00 00
:A950=08 01 41 84 20 00 00 00
:A958=00 00 00 03 01 41 02 32
:A960=03 00 01 4F 2C 42 B7 28
:A968=27 2A 80 30 E5 1A 13 06
:A970=08 1F CB 16 10 FB 23 0D
:A978=20 F3 22 80 30 EB CD 2E
:A980=56 E1 ED 5B 2A 7D B7 ED
:A988=52 EB E1 E5 23 73 23 72
:A990=E1 C3 65 50 CD 87 A5 F6
:A998=80 FE 80 CA D6 A4 3E 0D
:A9A0=C3 D6 A4 CD 5B 4C 3D FE
:A9A8=0C 30 15 C6 10 C3 D6 A4
:A9B0=3E 03 CA D6 A4 CD 70 4C
:A9B8=1D 7A B7 28 06 3C 28 1D
```

アドレス	データ	意味
A943	02	変数の種類(整数型単純変数)
A944	01	変数名の長さ＝1バイト
A945	41	変数名＝「A」
A946	0A 00	変数データ(0AH＝10)
A948	05	変数の種類(単精度型単純変数)
A949	01	変数名の長さ
A94A	41	変数名
A94B	84 20 00 00 00	変数データ(浮動小数点表示)
A950	08	変数の種類(倍精度型単純変数)
A951	01	変数名の長さ
A952	41	変数名
A953	84 20 00 00 00 00 00 00	変数データ
A95B	03	変数の種類(文字型単純変数)
A95C	01	変数名の長さ
A95D	41	変数名
A95E	02	変数データの長さ
A95F	03 32	ストリング・ディスクリプタ
A961	00	変数領域の終り

表3-3

3-5-2 配列変数テーブル

配列変数の格納形式は次図のようになっています.

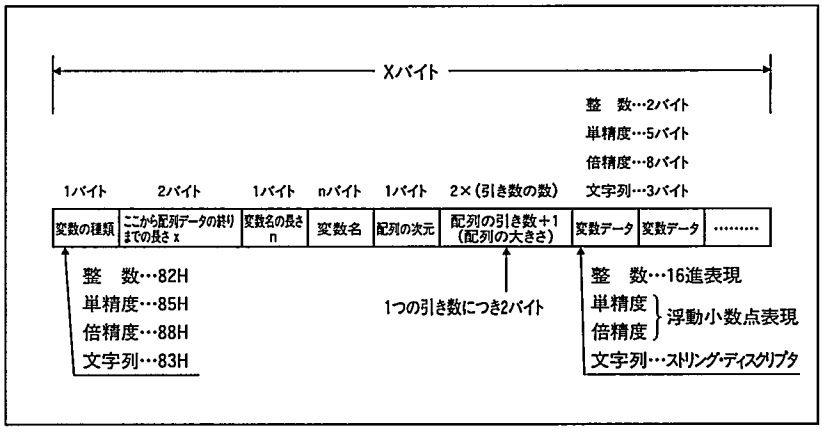


図3-4 配列の格納形式

n次元配列の場合は，「配列の大きさ」の部分がn個になります．また，「配列の大きさ」に格納される順番は，後の引数ほど先になり，変数のデータの格納順序は逆に引数の出てきた順になります．例えば，DIM A(2，5)と宣言した場合は「配列の大きさ」には引数5が先に格納され，次に引数2が格納されます．変数データ域には，A(0，0)，A(1，0)，A(2，0)，A(0，1)……A(2，5)の順に格納されていきます．

変数の種類を示すデータは，単純変数と区別するために，最上位ビットを1にしています．

では，実際にどのように変数が格納されているかを見てみましょう．

(NEW BASIC DISK版／NEW ON9)

リスト3-3

```
100 DIM a%(1,2)
110 '
120 a%(0,1)=10
130 a%(1,2)=12
140 a%(1,1)=11
```

```
:A958=00 00 00 82 15 00 01 41
:A960=02 03 00 02 00 00 00 00
:A968=00 0A 00 0B 00 00 00 0C
:A970=00 00 01 4F 2C 42 12 0A
:A978=00 00 0B 00 14 00 61 21
:A980=F4 12 0A 00 00 0B 00 1E
:A988=00 61 23 F4 12 0A 00 00
:A990=0C 00 28 00 61 24 F4 22
:A998=31 30 22 00 00 00 02 01
:A9A0=41 0A 00 05 01 41 84 20
:A9A8=00 00 00 08 01 41 84 20
:A9B0=00 00 00 00 00 00 03 01
:A9B8=41 02 32 03 00 00 4F A9
:A9C0=42 B7 28 27 2A 80 30 E5
:A9C8=1A 13 06 08 1F CB 16 10
:A9D0=FB 23 0D 20 F3 22 80 30
:A9D8=EB CD 2E 56 E1 ED 5B 2A
```

アドレス	データ	意　味
A95B	82	変数の種類(整数型配列変数)
A95C	00 15	配列データの終りまでの長さ＝15Hバイト
A95E	01	変数名の長さ　＝1バイト
A95F	41	変数名　＝「A」
A960	02	配列の次元　＝2次元
A961	00 03	二つ目の引き数+1　＝3
A963	00 02	一つ目の引き数+1　＝2
A965	省略	A(0, 0)のデータ
		A(1, 0)
		A(0, 1)
		A(1, 1)
		A(0, 2)
		A(1, 2)
A971	00	変数領域の終り

表3-4

3-6 数値の内部表現

3-6-1 浮動小数点

HuBASIC内部における演算と数値の変数領域への格納は，浮動小数点表記法で行われています．浮動小数点表記では，数値は仮数部＋指数部で表現されます．

浮動小数点表記とは，数値を有効数値と位どりの部分にわけて表現する方法で，例えば5430000は5.43×10^6と表せます．この方法をとれば，コンピュータ内部に数値を格納する場合にも，少ないバイト数で表現できます．

コンピュータで数値を扱う場合には基本的に2進数ですから，位どり等も2進数で表現します．2進数の浮動小数点表記には，いくつかの方法がありますが，ここではHuBASIC内部で使用してる表現を用いて10進の20.5を浮動小数点表記する過程を示してみましょう．

$$
\begin{aligned}
20.5&=16+4+0.5\\
&=2^4+2^2+2^{-1}\\
&=10100.1B\\
&=1.01001B\times2^4
\end{aligned}
$$

0以外の実数はすべてこの形式で表現することができ，数式的に表すと次のようになります．2進数は1か0しかないため，いちばん上の桁(整数部)は常に1となります．

$$\pm1.********B\times2^{\pm n}\quad(*は0か1)$$

指数部に1バイト割り当て，0の時は指数部を00Hとすれば基本的にすべての数値が表現できます．また，指数部は正負両方の値をとるので，1バイトで表現するために，実際の数値でなく，81Hを0とする指数を用います．この方法では，もとの指数の数値を1とすると82H，2なら83H，同様に－1なら80H，－2なら7FHとなります．

仮数部の表現方法は，その整数部が常に1であるため，このビットを格納する必要はありません．ですから，このビットは仮数部の符号として使用します．仮数部が正の場合は最上位ビットを0，負の時には最上位ビットを1にします．

変数型により仮数部に使用されるバイト数は異なり，単精度数値を格納するためには4バイト，倍精度数値には7バイト使用します．

実際に，先ほどの値20.5($1.01001B\times2^4$)を単精度数値としてHuBASICの格納形式で表せば，

```
指数部    4＋81H＝85H
仮数部    00100100   00000000   00000000   0000000
          ＝24H  00H  00H  00H
```

と，なります．

さて，10進数を2進数に変換すると，きれいに割り切れない場合がでてきます．たとえば，0.1は$1.001100110011\cdots\times2^4$となって無限循環小数になってしまいます．このような場合は最後の桁を0捨1入します．

```
指数部    －4＋81H＝7DH
仮数部    01001100  11001100  11001100  11001100  1100…
                                          (↑桁上げ)
          ＝4CH  CCH  CCH  CDH
```

3-6-2　浮動小数点の演算誤差

浮動小数点表記で，10進数を2進数に変換すると，誤差がでることがあります．前節で，0.1が，無限循環小数になってしまうことを示しましたが，例えば，次のプログラムをHuBASICで実行してみると，

```
A#=0.1
PRINT A#

0.1000000000058208
```

と表示され，A#は正確に0.1ではないことが確認されます．これは，この場合，仮数部の57ビット目が0捨1入されたためで，このまま2進数で浮動小数点表記を使っている限り，避けることができません．

このような誤差を避けるための方法としては，10^n倍して，できるだけ整数の状態で計算をして，表示，登録の段階で10^nで除して実際の数値に戻す方法などが挙げられます．

3-7　機械語サブルーチンとのリンク

HuBASICには，機械語のサブルーチンを呼び出したい場合のためにCALL命令とUSR関数が用意されています．CALL命令は手軽に使える反面，パラメータの受け渡しが困難という短所があり，逆にUSR関数は関数なのでパラメータの受け渡しは簡単なのですが，少々理解しづらい面があります．

3-7-1　CALL命令

CALL命令は，直後に書かれたアドレスから始まる機械語サブルーチンを呼び出します．機械語サブルーチンからBASICに復帰するためには機械語プログラムの中でRET命令(0C9H)を実行します．

HuBASICのCALL命令には機械語サブルーチンとのパラメータ受け渡しの機能はサポートされていませんので，PEEK命令，POKE命令を使ってメモリーを介してパラメータを受け渡します．また，マニュアルには書かれていませんが，CALL　アドレス　(データ)という形式でHLレジスタにデータを渡すこともできます．ただし，機械語サブルーチンからBASICにデータを返す機能はサポートされていないので，データの読みとりはPEEK命令を使います．

なお，機械語サブルーチン内でレジスタを保存する必要はありません．

3-7-2　USR関数

USR関数は，関数の形で機械語のサブルーチンを呼び出します．機械語サブルーチンから復帰したあとは，その結果を持つ関数になります．USR関数は，実行前に呼び出す機械語サブルーチンの実行開始アドレスを定義する必要がありそれにはDEF USR文を使って次のように書きます．

```
DEF USRn=a
```

nは関数識別番号で，0～9の最大10個の機械語サブルーチンを定義する事ができます．nを省略すると0が指定されます．実行開始アドレスaは，機械語サブルーチンの実行が開始される

アドレスであり，メインメモリー64KB中のどこにでも指定できます．

DEF USR文でいったんアドレスを定義すると，再定義しなおすまで，そのアドレスが保持され，NEWやCLEAR命令を実行しても，消去されません．

DEF USRで，nとaを定義したら次のようにしてUSR関数を使うことができます．

① y＝USRn(x)

① y：数値変数
n：USR関数識別番号．0～9の整数．
x：数式(パラメータ)

② y$＝USRn(x$)

② y$：文字変数
n：USR関数識別番号．0～9の整数．
x$：文字式(パラメータ)

USRの後ろに付ける番号nは，DEF USR文で設定した番号に対応します．

USR関数は，パラメータxやx$の値を持って，n番の機械語サブルーチンを呼び出し，復帰時に，その値をyやy$に代入します．機械語サブルーチンの中で，パラメータの値を書き換えると，その結果がyやy$の値となります．USR$のパラメータx，x$は省略する事ができません．

xは，数式で，単独の定数もしくは数値変数でもよく，精度も整数型，単精度型，倍精度型のいずれでも構いません．x$は，文字列を値に持つ文字式です．

USR関数で機械語サブルーチンを呼び出すときCPUの各レジスタには次の表で示される値が入っています．

パラメータ / レジスタ	数値型	文字型
A	データの種類・バイト数 02H…整数　05H…単精度　08H…倍精度	03H…文字型のみ
HL	データ格納アドレス	ストリングディスクプリタ格納アドレス
B	無意味	文字列の長さ
DE	無意味	文字列データ格納先アドレス
IX	エラー処理ルーチン・エントリーアドレス	

表3-5

● Aレジスタ(アキュムレータ)

Aレジスタ(アキュムレータ)には，パラメータの型を示す値が入っています．これらの値は，文字変数を除き，メモリー内において何バイトで表現されているかを表しています．

● HLレジスタ

HLレジスタの値は，数値変数と文字変数の場合で異なっています．数値変数の場合はパラメータの入っているメインメモリーのアドレスを示しており，パラメータが文字変数の場合は，直接文字列の入っているアドレスを示さず，ストリングディスクリプタと呼ばれているテーブルのアドレスを示しています．パラメータの値は，HLレジスタが指しているアドレスから格納されています．

● DEおよびBレジスタ

DEとBレジスタは，Aレジスタが3のとき，すなわちUSR関数のパラメータが文字変数の時のみ意味を持ちます．この時DEレジスタにはその文字データが格納されているアドレス(絶対アドレス)，Bレジスタには文字数が入っています．

※ USR関数の引数に文字列を使う場合の注意

USR関数の引数に文字列を使った場合，機械語ルーチンからBASICに制御が戻ったときに，引数に使用した文字列が破壊される場合があります．例えば，A$を引数に使い，A$の先頭1文字を削除した文字列を値として返す関数を作った場合，BASICに制御が戻った時点でA$は先頭1文字が削除されてしまっています．これを避けるためには次のようにします．

DM$＝USR(A$＋″″)

この方法をとれば，機械語ルーチンで参照するのは，テンポラリストリングバッファにコピーされたA$＋″″ ですから，破壊されるのはそちらであり，A$は破壊されません．

● USR関数内でのエラー処理

USR関数内でエラーが発生した場合，BASICにエラーの発生を知らせることができます．このとき，BASIC内であらかじめON　ERROR　GOTOの定義をしていれば，エラー処理ルーチンへジャンプすることもできます．

エラーの通知はエラー番号をAレジスタに入れIXレジスタの示すアドレスにジャンプすることにより行います．