マイクロコンピュータ用言語BASICAL

大場克彦，喜利元貞
（島津製作所 中央研究所）

Ⅰ はじめに

マイクロコンピュータシステムの中で，BASIC言語が広く使われているが，これは

(1) プログラム開発が手軽にできる
(2) 処理系が小さくてすむ

などの理由によるものである．しかし実行速度が遅く，かつ，ビット処理などの細かい処理が難かしいため，入出力処理ルーチン，モニタ，割込み処理ルーチン，言語処理系などシステムプログラムの記述には適さないという制約がある．この制約を取り除くため，アセンブリ言語なみに細かい記述ができ，かつ，BASICなみに手軽にプログラムを作成できるプログラミングシステムの一部として，コンパイラ型のマイクロコンピュータ用言語BASICALを開発した[1][2]．本報ではその言語仕様を中心にさらに詳しく報告する．

Ⅱ 設計の方針

マイクロコンピュータのプログラムでは，生成されるプログラムのオブジェクト効率のよいことが求められる．また，使用されるプログラム開発装置が安価であることが必要である．このため，同種のCPUからなるマイクロコンピュータシステムが使われることが多く，言語処理系が小型であることが必要になる．さらにプログラム開発システム（プログラミングシステム）の操作性が，プログラム作成の生産性を左右する．そこで，言語仕様の設計に当って，「システムプログラムの記述ができること」の他に，次の点を目標とした．

(1) 生成されるオブジェクト効率がよい．
(2) 言語処理系が小型である．
(3) プログラミングシステムの中に，組込まれた時の操作が簡単である．

ところで，一般の高級言語では，生成されるオブジェクト効率をよくするためには，複雑な最適化を行なわなければならない．これは処理系を小型にするという要請に反する．そこで，この相反する要請を満足させるために，言語仕様の中に，生成されるオブジェクトを短かくしうる命令文や組込み関数を設け，アセンブリ言語に近い細かい記述を高級言語の書式で書けるようにした．たとえば，整数やビット列の処理だけでなく，レジスタ演算や，レジスタ間接演算を可能にした．こうして言語処理系では簡単な最適化をするだけであるにもかかわらず，文や組込み関数の選び方によってオブジェクトの生成効率をよくすることができるようにした．

また，処理系を小型にし，かつ，その操作を簡単にするために，できるだけBASIC言語の文法を取り入れた．

Ⅲ 言語仕様の概要

言語仕様の概要を以下に述べる．

1 プログラム

プログラムは，BASIC言語と同様に行の集まりである．行は行番号と文とからなる．分岐によって実行順序が変えられない限り，プログラムの実行は行番号の小さい順に行なわれる．

2 データの型

次の4つの型がある．

整数　16ビット長の2の補表表現

符号なし整数　16ビットの正整数表現
バイト整数　8ビットの正整数表現
実数　32ビットの浮動小数点表現

3　データ参照

定数の参照，変数の参照，レジスタの参照，間接参照の4つの参照方法がある．

3.1　定　数

整定数，16進定数，実定数の3つがある．書式をFig.1に示す．

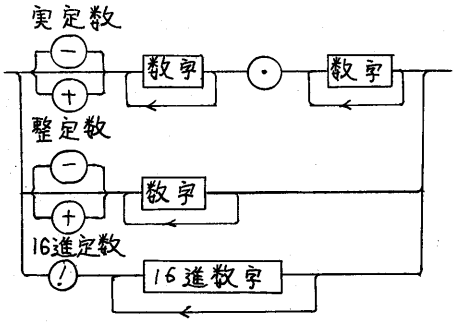

Fig.1　定　数

3.2　変　数

次の3つがあり，それぞれに単純変数と配列要素がある．

整変数　　単純整変数
　　整配列要素

実変数　　単純実変数
　　実配列要素

型なし変数　単純型なし変数
　　型なし配列要素

次に変数とデータの型の関係を示す．

整変数　　整数，符号なし整数
実変数　　実数
型なし変数　全ての型のデータ

変数の型は暗黙の宣言による．

整変数名＝変数名％
実変数名＝変数名
型なし変数名＝@変数名

変数名は英字で始まる2文字の英数字である．

配列は2次元まで可能である．添字は整定数，単純整変数，単純型なし変数で指定する．実数，式，配列要素などは許さない．

例

整変数　AI%，AX%，UX%(2)
　　VE%(5,3)

実変数　B，KC，YX(5)
　　XY(0,4)

型なし変数　@C，@LM(2)
　　@NW(6,8)

型なし変数について補足する．

実際にプログラムを書いて見ると，プログラム領域の任意の場所にデータ領域を取りたいことがある．また，同一データ領域に異なった型のデータを混在できると便利なことが多い．さらに，プログラム領域の外のデータを参照しなければならない場合がある．このような目的のために型なし変数を設けた．

型なし変数を参照する時，どの型のデータとして参照しているかを識別するため，次のような規則を設けた．

(1) 整数，符号なし整数として参照する場合は，識別子をつけない．
　(例　@A，@B(0,1))
(2) 実数として参照する場合は，前に識別子Fをつける．

（例　F@A，F@B(0,1)）
(3) バイト整数として参照する場合は前に識別子Bをつける．
（例　B@A，B@B(0,1)）

型なし変数は使用する前に，AREA文あるいはEQU文で宣言しておかなければならない．

3.3　レジスタ参照

0～15の16コの16ビットレジスタの参照を可能にした．ただし，プログラマが自由に参照できるのは，4から10のクコである．

レジスタの参照は，次の書式の組込み関数を利用して行なう．

書式　　.RX

ここでXは0～15の整数で，レジスタの番号を示す．

例　.R5，.R6

レジスタのデータは，整数か符号なし整数のどちらかである．

3.4　間接参照

ある番地のデータを直接参照したいことがある．このために，整変数やレジスタをポインタとして利用することを可能にする組込み関数を設けた．

間接参照できるデータの型は，整数，符号なし整数，バイト整数である．

4　式

BASICALにおける式の関係をFig.2に示す．各代入式あるいは関係式の中で，異なる型の混合演算は許さない．

整数代入式，実数代入式は，LET文の中で許される．符号なし整数代入式は，PLET文の中で許される．

整数関係式，実数関係式はIF文の中で許される．符号なし整数関係式はPIF文の中で許される．

実数代入式，実数関係式の中では，実数型のデータしか許さない（ただし配列の添字は整数）．

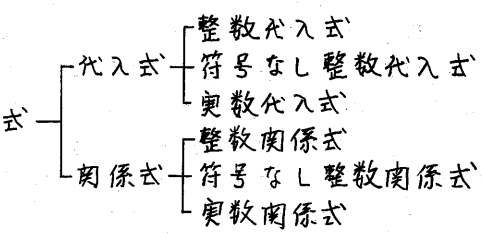

Fig.2　式の関係

整数代入式，整数関係式の中では，整変数，レジスタ，間接参照するデータの内容は，符号付きの整数として演算される．

符号なし整数代入式，符号なし整数関係式の中では，整変数，レジスタ，間接参照するデータの内容は，符号なし整数として演算される．

整数代入式，符号なし整数代入式の左辺には，整変数，符号なし変数だけでなく，レジスタ組込み関数，間接参照組込み関数を書くことができる．すなわち，次のような例が許される．

```
100 .R5 = .R6 + A%
110 .IW(A%) = .R5 + B%
```

演算の規則，算術演算子，関係演算子は，BASIC言語と同じである．

5　文

BASICALの文のうち，主要なものを表1に示す．このうち特徴的なものを説明する．

5.1　AREA文

型なし変数を宣言する文で，この文が書かれた場所に，宣言された構造の領域が確保される．書式をFig.3に示す．型なし変数は，変数あるいは配列要素の大きさを指定することができる．

次に例を示す．

```
100 AREA @WP[:32],@A[3:2],@B[5,6:4],@C
```

この例では，@WPという名の32バイト変数，@Aという名の2バイト単位の4行配列，@Bという名の4バイト単位の6行7列配列，@Cという名の1バイト変数の領域を確保することを意味する．セミコロンの後の数字は，変数あるいは配列の要素の大きさを示す．この大きさが1の場合は，省略することができる．また変数名だけの場合は，1バイトの大きさの変数である．配列の次元は0から5始まる．

5.2 データテーブルの作成

AREA文で宣言した領域に，コンパイル時にデータを設定する文として，次のようなものがある．Fig.3に書式を示す．

INT文	整数データの設定
TEXT文	文字列データの設定
BYTE文	バイト整数データの設定
REAL文	実数データの設定

次に例を示す．

(1) INT文

```
100 AREA @A[3:2]
110 INT @A;1,2,4,5
```

2バイト単位の4行配列@Aに整数データが次のように設定される．

@A[0]=1　　@A[2]=4
@A[1]=2　　@A[3]=5

(2) TEXT文

```
200 AREA @B[3:4]
210 TEXT @B[1];AAAA,BBBB
```

4バイト単位の4行配列@Bに次のように文字列が設定される．

@B[1]="AAAA"
@B[2]="BBBB"

" "の中は文字列を示す．

表1 主要な文

AREA	型なし変数の宣言
DIM	実配列，整配列の宣言
EQU	外部型なし変数の宣言
LET	実数，整数の代入文
PLET	符号なし整数の代入文
GOTO	無条件分岐文
ON GOTO	ケースによる分岐文
IF THEN	条件付き分岐文
PIF THEN	符号なし整数の条件付き分岐文
FOR	FORループの開始
NEXT	FORループの終端
INT	型なし変数に整数データを設定
TEXT	型なし変数に文字列データを設定
BYTE	型なし変数にバイトデータを設定
REAL	型なし変数に実数データを設定
ESUB	外部サブルーチン宣言
ECALL	外部サブルーチン呼び出し
SUB	内部サブルーチン定義
SUBEXIT	内部サブルーチン出口
SUBEND	内部サブルーチンのテキストの終り
CALL	内部サブルーチン呼び出し
INC	オペランドの内容を増分する
DEC	オペランドの内容を減らす
AS.	インラインアセンブラ文

(3) BYTE文

```
300 AREA @C[3]
310 BYTE @C;3,2,0
```

1バイト単位の3行配列に，次のようにバイト整数が設定される．

@C[0]=3
@C[1]=2
@C[1]=0

ただし設定できるデータは，0～255の整数である．

(4) REAL文

```
400 AREA @E[3:4]
410 REAL @E;2.0,3.5,6.2,7.8
```

4バイト単位の4行配列@Eに，次のように実数データが設定される．

@E[0]=2.0　　@E[1]=3.5

@E[2]= 6.2　　@E[3]= 7.8

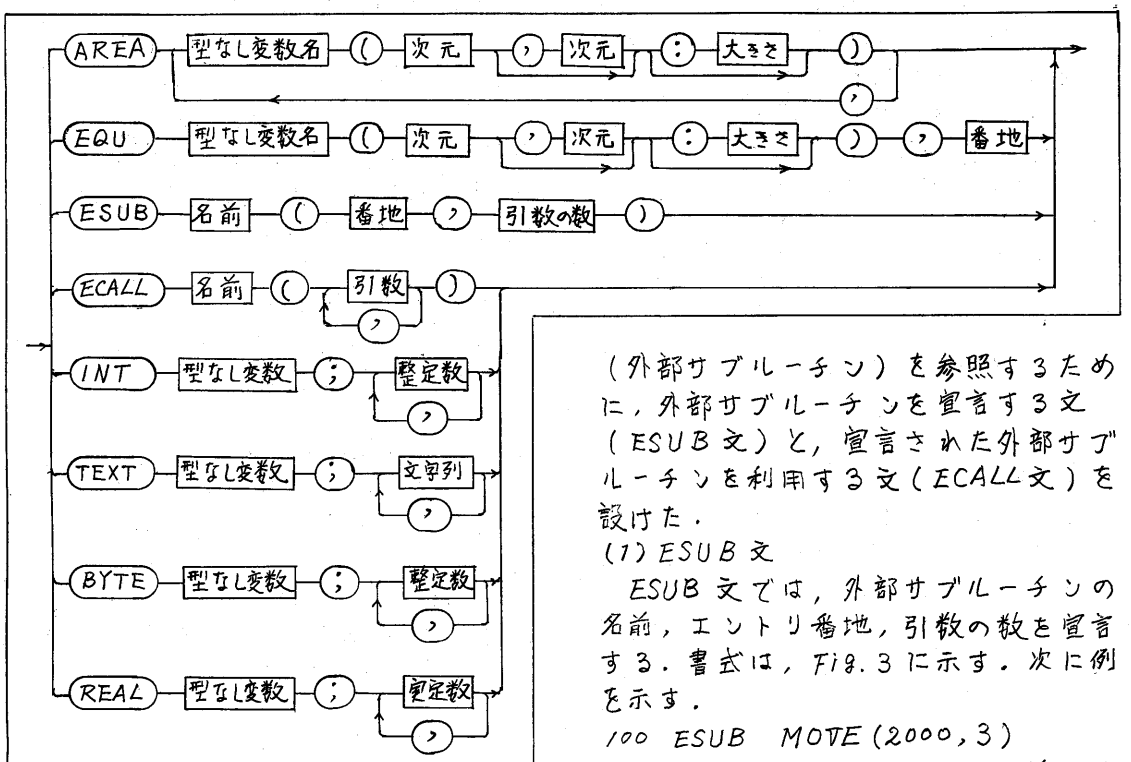

Fig.3 いくつかの文の書式

5.3　外部データの参照

プログラムの外部にあるデータを型なし変数として宣言しておき，これをプログラムの中で参照することができる．この外部データを宣言するためにEQU文を用いる．

書式はFig.3に示す．例を示す．

```
50 EQU @BZ(19:2),200
```

この例は，@BZという名前で2バイト単位の20行配列が，16進表示の200番地から割付けられていることを示す．この宣言の後，この配列へのデータ参照，代入が可能になる．

5.4　外部サブルーチン参照

プログラムの外にあるサブルーチン（外部サブルーチン）を参照するために，外部サブルーチンを宣言する文（ESUB文）と，宣言された外部サブルーチンを利用する文（ECALL文）を設けた．

(1) ESUB文

ESUB文では，外部サブルーチンの名前，エントリ番地，引数の数を宣言する．書式は，Fig.3に示す．次に例を示す．

```
100 ESUB  MOVE(2000,3)
```

この例では，MOVEという名前のサブルーチンのエントリ番地が，16進の2000で，引数の数が3個であることを宣言している．名前は英字で始まる6文字以内の英数字である．

(2) ECALL文

ESUB文で宣言された外部サブルーチンを利用する文である．書式はFig.2に示す．例を示す．

```
200 ECALL MOVE(A%,B%,C%)
```

MOVEという名前の外部サブルーチンに，A%，B%，C%という3つの引数を渡して呼んでいる．

5.5　INC文，DEC文

整変数，レジスタの内容を符号なし整数として，1あるいは2増減する文である．

(1) INC文

INC1文とINC2文がある．INC1文は，オペランド欄に書かれた整変数およびレジスタの内容を1増分する．

INC2文は同様に2増分する．書式は次の通りである．

INC1 OP,-----,OP
INC2 OP,----,OP

ここでOPは，整変数あるいはレジスタを示す．例を示す．

```
200 INC1 A%,.R5
210 INC2 .R6
```

(2) DEC文

DEC1文とDEC2文がある．DEC1文は，オペランド欄に書かれた整変数およびレジスタの内容を1へらす．

DEC2文は同様に2へらす．書式は次の通りである．

DEC1 OP,---- OP
DEC2 OP,---- OP

OPの意味はINC文と同じである．

5.6 インライン アセンブラ文

ハードウエアに密着した入出力命令や，特に実行速度を速くしたり，メモリ容量を小さくする必要がある場合のために，BASICALのプログラムの中に部分的にアセンブリ言語のプログラムを書けるようにした．これがインラインアセンブラ文である．書式は次の通りである．

AS. 文字列

文字列の部分にアセンブリ言語の命令を書く．

6 組込み関数

アセンブリ言語なみの細かい処理を可能にするために，レジスタ参照，ビット処理，直接データ参照，変数アドレスの参照，データの型変換を可能にする組込み関数を設けた．レジスタ参照については既に述べたので，それ以外のものについて述べる．

6.1 ビット処理

16ビット（1語）単位のビット列の処理を可能にするため，Fig.4に示す組込み関数を用意した．次に例をもってその内容を説明する．

```
100 B%=.SR(A%,2)
```

A%の内容を2ビット右シフトしてB%に代入する．

```
110 .R5=.SL(.R4,1)
```

レジスタ4の内容を1ビット左シフトしてレジスタ5に代入する．

```
120 S%=.SB(.R10,15)
```

レジスタ10の15ビット目をセットビットした結果をS%に代入する．

```
130 .R5=.RB(A%,3)
```

A%の3ビット目をリセットビットした結果をレジスタ5に代入する．

```
140 C%=.TB(.R3,0)
```

レジスタ3の0ビット目が1なら1を，0なら0をC%に代入する．

```
150 C1%=.AND(A%,B%)
```

A%とB%のビットごとの論理積をとった結果をC1%に代入する．

```
160 CX%=.OR(.R4,X%)
```

レジスタ4とX%のビットごとの論理和をとった結果をCX%に代入する．

```
170 CY%=.XOR(.R4,.R5)
```

レジスタ4とレジスタ5のビットごとの排他的論理和をとった結果を，CY%に代入する

```
180 M%=.NOT(A%)
```

A% の各ビットの値を反転した結果をM%に代入する．

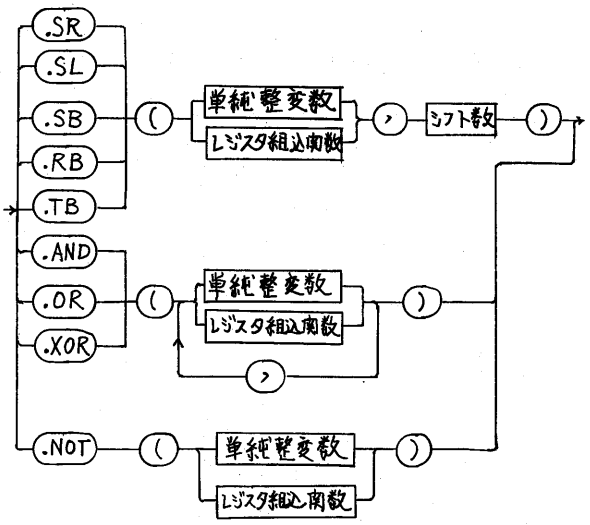

Fig.4 ビット処理組込み関数

6.2 間接データ参照

間接データ参照組込み関数を用いると，ある番地のデータを参照することができる．6種類の関数を用意したが，その書式をFig.5に示す．これらの関数では，引数の値が参照番地となる．

.IW，.IWI，.IWDの3つは語データを参照する．データ参照後，.IWIは引数の値を2増加させ，.IWDは引数の値を2へらす．

.IB，.IBI，.IBDの3つは，バイトデータを参照する．データ参照後，.IBIは引数の値を1増加させ，.IBDは引数の値を1へらす．

番地付けはバイト単位でされていることを想定しているので，語データを参照する時は，引数の値の番地とその次の番地が参照される．

次に書式例を示す．

```
100  A%=.IW(B%)
110  C%=.IWI(.R4)
120  .R4=.IWD(.R6)
130  .IB(C%)=.IB(.R4)
```

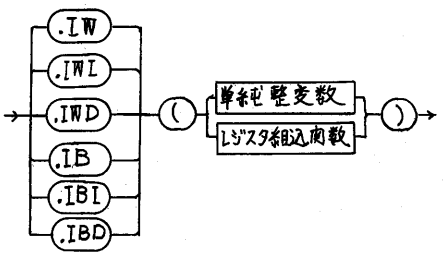

Fig.5 間接データ参照組込み関数

```
140  .IBI(.R5)=.IBI(.R7)
150  .IBD(C%)=.IBD(A%)
```

6.3 変数の番地の参照

変数の番地を参照する組込み関数を用意した．書式を次に示す．

.AD(変数)

引数として書かれた変数の番地を，値として与える．

次に例を示す．

```
200  .R4=.AD(A%[0])
```

配列要素A%[0]の番地をレジスタ4に代入する．

6.4 型変換

型変換の組込み関数により，実数から整数，整数から実数への変換が可能になる．

(1) 実数から整数への変換

書式は次の通りである．

.FX(実変数)

引数として書かれた実変数の値を，整数に変換した結果が値として与えられる．次に例を示す．

```
200 B%=.FX(A)
```

実変数Aの値を整数に変換した結果をB%に代入する．

(2) 整数から実数への変換

書式を示す．

.XF(整変数)

引数として書かれた整変数の値を実変数に変換した結果が，値として与えられる．次に例を示す．

210 C=.XF(X%)

整変数X%の内容を実数に変換した結果を，実変数Cに代入する．

IV プログラム例

2バイト単位の10行配列@A，@B，@Cがある．各配列はそれぞれ，2000番地，2100番地，2200番地から割付けられている．この配列に対して，次のような整数演算をするプログラムを下に示す．

@C[I]=@A[I]+@B[I]
(I=0～9)

```
10 REM  EXAMPL
20 EQU  @A[9:2];2000
30 EQU  @B[9:2];2100
40 EQU  @C[9:2];2200
50 .R4=2000,.R5=2100,.R6=2200
60 FOR  I%=0 TO 9
70  .IWI[.R6]=.IWI[.R4]+.IWI[.R5]
80 NEXT I%
90 STOP
```

V おわりに

BASICALのインプリメンテーションは，TMS9900という16ビットCPUチップを用いたフロッピーディスク(512Kバイト)ベースのマイクロコンピュータSHIP-9(RAM 60Kバイト，ROM 4Kバイト)上で行なった．

コンパイラは，プログラム約20Kバイト，データテーブル約20Kバイトの約40Kバイトである．

BASICALの記述性を確かめるために入出力処理プログラムやインタプリタの一部をBASICALで記述してみた．インタプリタは完全に記述することができ，入出力処理プログラムも，入出力処理装置を駆動する部分をインラインアセンブラ文で書けば，完全に記述することができた．

オブジェクト生成効率を見るため，コンパイラの中の1つのモジュールをBASICALで記述した．アセンブリ言語で書いたプログラムを，そのまま移したもの(プログラムA)とプログラム容量を小さくするように工夫したもの(プログラムB)を作って比較した．その結果は次の通りである，

プログラム容量は，アセンブリ言語で548バイト，プログラムAで1080バイト(アセンブリ言語の約2倍)，プログラムBで824バイト(1.5倍)であった．これは，最適化をほとんど行なっていないことを考えれば，満足すべき結果であると言える．また，プログラムステップ数は，アセンブリ言語で310，プログラムAで125，プログラムBでめであった．これは，アセンブリ言語のそれぞれ約40%，28%である．単純に，プログラムの開発費がステップ数に比例するとすれば，それぞれ，アセンブリ言語の40%，28%の費用ですむことになる．

以上の結果から，当初，目標とした内容は達成されたと考えられる．

今後の課題として，生成されるオブジェクト効率を，アセンブリ言語の110%以下にすることも可能であるようにしたいと，考えている．

[1] 大場，喜利　マイクロコンピュータ用言語BASICAL(言語仕様について)
情報処理学会第22回全国大会予稿集
[2] 藤田，大場，喜利　マイクロコンピュータ用言語BASICAL(インプリメンテーション)
情報処理学会第22回全国大会予稿集