SHARP

SHARP
COMPUTER
SOFTWARE

X1turbo・X1シリーズ用

ランゲージシリーズ

FORTRAN

CZ-115LF

ユーザーズマニュアル

LIFEBOAT

FORTRAN

ユーザーズマニュアル

CONTENTS

● FORTRAN

CONTENTS

CONTENTS

CONTENTS

CONTENTS

● APPENDIX

● ASSEMBLER

CONTENTS

はじめに

このマニュアルではこのシリーズFORTRANの使用法について説明してあります。このシリーズFORTRANは，8080／8085／Z80マイクロプロセッサの為のFORTRANです。FORTRANは，コンピュータ言語としては最も一般的に使用されているものですが，いくつかの異なった仕様のものがあります。このシリーズFORTRANは，FORT Ⅳ，或はANSI 66 FORTRAN，JIS6000レベルのサブセットですが，マイクロコンピュータの為に幾つかの拡張がなされています。

このマニュアルでは，ヴァージョン3.0以上のこのシリーズのFORTRANの使用法について説明してありますが，FORTRAN言語の文法については，以下に紹介する参考書を参照して下さい。

○FORTRAN入門　浦　昭二　編

培風館「電子計算機のプログラミング」シリーズ1巻

このマニュアルでも，文法をよく知らない方の為に，上記の本を〔参考書28ページ参照〕と引用します。もちろん，FORT Ⅳに精通なさってそる方は，参照する必要はありません。参考書のページ数は〔改訂版〕に於けるものです。

1・FORTRANの起動

●必要なハードウェア

(1) 8080／8085／Z80 マイクロプロセッサ
(2) 最低48Kバイトのフリーエリア（CP／M－OSを除く）
(3) 一台以上のフロッピーディスクドライブ

●必要なソフトウェア

(1) CP／M オペレーティング・システム
(2) テキスト・エディタ

●FORTRAN マスタ・ディスク

(1) FORT. COM …… FORTRANコンパイラです。
(2) FRUN. COM …… 実行時ルーチンです。
(3) CONFIG. COM …… 環境の設定の為に使用します。
(4) FRRORS …… エラーメッセージが入っています。
(5) READ. ME …… FORTRANに関する最新のニュースが入っています。
(6) ASSM. COM …… アセンブラですが，FORTRANコンパイル時にも必要です。

その他に，いくらかのサンプルプログラムが入っています。

●FORTRAN 起動の準備

もし，マスタ・ディスケットが書き込み禁止になっていなかったら，書き込み禁止にして下さい。

次にCP／MのPIPコマンドを使用してマスタ・ディスケットのコピーを作り，マスタ・ディスケットはしまってください。

1.1 FORTRANシステムの構築

1.1.1 FORT. ERRの作成

CONFIG. COMを起動して，ERRORSファイルから，FORT. ERRファイルを作成します。すなわち，CP／Mのコマンド待ち状態の時にCONFIG↵とします。ERRORSファイルの存在するドライブと，FORT. ERRファイルを作成するドライブを尋ねてきますから，ドライブ名を答えます。この作業はERRORSファイル内のテキスト（エラーメッセージ）を変更しないかぎりは，再び行なう必要はありません。CONFIGコマンドを実行すると，自動的に次の節の説明の様に，システムの構築がはじまります。

```
NEVADA FORTRAN CONFIGURATION PROGRAM (03MAR83)

DO YOU WANT TO CREATE THE ERROR FILE (Y/N)? Y↵
WHICH DRIVE CONTAINS THE FILE 'ERRORS'? B↵ A
TO WHICH DRIVE SHOULD 'FORT.ERR' BE WRITTEN? B↵ B
+++ CREATING 'FORT.ERR' +++

************************************************
+++ FILE 'FORT.ERR' DONE +++

+++ CONFIGURING NEVADA FORTRAN COMPILER +++

WHICH DRIVE CONTAINS THE FILE 'FORT.COM'?
SPECIFY DRIVE 'Z' TO SKIP THIS STEP: B↵ A
   (省略値を変更しない場合はZでよい)

THE NUMBER IN [] IS THE CURRENT DEFAULT FOR EACH PARAMETER.
TO USE THE DEFAULT, JUST HIT RETURN/ENTER, TO CHANGE. ENTER
THE NEW VALUE IN DECIMAL


SYMBOL TABLE SIZE [00050]↵          S
LABEL TABLE SIZE [00050]↵           L   3.7を参照して変更して下さい。
NUMBER OF ARRAYS [015]↵             A   オプションにそれぞれ相当します。
NUMBER OF NESTED DO LOOPS [005]↵    D
IF-THEN-ELSE NESTING DEPTH [005]↵   I

THE FOLLOWING 3 PARAMETERS HAVE TO DO WITH EXPRESSION EVAULATION
THE MORE COMPLEX THE STATEMENT TO BE EVALUATED, THE LARGER THE
FOLLOWING 3 PARAMETERS MUST BE.  THE DEFAULTS SHOWN SHOULD BE
STATISFACTORY FOR MOST EXPRESSIONS


NUMBER OF TEMPORARIES USED DURING EXPRESSION EVALUATION [015]↵
NUMBER OF OP-CODE STACK VARIABLES [040]↵
NUMBER OF OPERAND STACK VARIABLES [040]↵
   (T, O, P, オプションにそれぞれ相当します。3.7を参照して下さい)
CHARACTER TO BE USED TO SURROUND HEX CONSTANTS IN STRINGS [\]↵
   (ターミナルによっては¥, 本文中では\<バックスラッシュ>となっています。)
CAN YOUR CONSOLE OUTPUT DEVICE HANDLE LOWER CASE LETTERS? (Y/N) Y↵
WHICH DRIVE CONTAINS THE FILE 'FRUN.COM'? (コンソール出力に小文字が可能か)
SPECIFY DRIVE 'Z' TO SKIP THIS STEP: B↵
   (省略値を変更しない場合はZでよい)
+++ CONFIGURING NEVADA FORTRAN RUNTIME PACKAGE +++

YOU CAN SPECIFY ONE OF 3 WAYS THAT THE RUNTIME
PACKAGE CAN DO CONSOLE I/O.  EACH HAS ITS ADVANTAGES.
CONSULT YOUR CP/M MANUALS FOR THE DIFFERENCES OF EACH
ENTER:
0 TO USE DIRECT BIOS I/O
2 TO USE CP/M FUNCTIONS 1 & 2
3 TO USE CP/M FUNCTION 6

2↵   CP/M 1.4の場合は2を選んで下さい。

CAN YOUR CONSOLE OUTPUT DEVICE HANDLE LOWER CASE LETTERS
DURING PROGRAM EXECUTION? (Y/N) Y↵
   コンソールに小文字出力が可能か?

DO YOU HAVE A NORTH STAR FLOATING POINT BOARD IN YOUR
SYSTEM? (Y/N) N↵
   ノースター社の演算ボードを使用しているか?
+++ ALL DONE +++
```

1.1.2 FORTRANシステムの構築

FORT. ERRの作成に引続いて，コンパイル・オプションと実行時ルーチンのパラメタの省略値の設定を行ないます。オプション名とその省略値を表示して入力を促しますので，その省略値をそのまま認めるならば，↵を，変更したければ新しい値を入力して↵を続けて入力して下さい。16進数で答える場合とY／Nで答える場合がありますが，表示されている値をそのまま認める時は↵だけでよろしい。コンパイル・オプションについては，2．3節と3．7節を参照して下さい。小文字の使用を許すかどうかという問いがありますが，もしコンソールからの小文字入力が可能ならYを，不可能ならばNを入力して下さい。（例を参照して下さい。）

続けて　実行時ルーチンのパラメタの省略値を設定します。コンソールの入出力に，どのBDOSコールを使用するか，小文字が出力できるか，ノースター社の浮動小数点ボードを使用するかについて答えます。（例を参照して下さい。）

以上でシステムの構築作業は終了しました。ディスク・スペースに余裕がなければ，CONFIG. COMとERRORSファイルを消して下さい。

2・プログラムのコンパイルと実行

2.1 プログラムの作成

テキスト・エディタを使用してフォートランのソース・プログラムを作ります。ファイルの拡張子は，FORにして下さい。ソース・プログラムの詳細については３．２節か参考書P. 185を見て下さい。

2.2 プログラムのコンパイル

ソース・プログラムは，

FORT␣U：PGM. LAO␣$OPTIONS↵

とすればコンパイルできます。ここで，

- FORTはFORT. COMを示します。
- U：は，ソースプログラムの存在するドライブ名です。
- PGMは拡張子なしのソースプログラム名です。
- Lは，以下の様にリストの出力先を指定します。すなわち，

  A－P　ドライブ名
  X　システムコンソール
  Y　CP／Mのリストデバイス
  Z　リスト出力を省略する

ただし，A－Pを指定した時リスト出力ファイルはPGM. LST という名前になります。

- Aは中間ファイルの出力先を指定します。すなわち

A－P　ドライブ名
Z　中間ファイルの出力を省略する。

中間ファイルの拡張子は .ASMとなります。このファイルはフォートランコンパイラによって自動的に消去されます。

- Oはオブジェクト・ファイルの出力先を指定します。すなわち，

A－P　ドライブ名
Z　オブジェクト・ファイルを作らない。

オブジェクト・ファイルは，.OBJ という拡張子を持ちます。

＜注＞ もし，中間ファイルか，オブジェクト・ファイルかにZを指定するとオブジェクト・ファイルは生成されません。

もしドライブ名を指定しなければデフォルト・ドライブ上に作成します。

FORT. COMとASSM. COMファイルが，デフォルト・ドライブ上に存在しなければなりません。

もしOにZ以外を指定すると，中間ファイルは自動的に消去されます。Zを指定すると，Aに Z以外を指定した場合は，.ASM という拡張子を持つ中間ファイルを出力します。

2.3 コンパイル時のオプション

コンパイル時に指定できるオプションについて説明します。これらのオプションは，コンパイル時に，コマンド行の最後に$記号に続けて入力します。例えば

A＞FORT␣U：PGM. LAO␣$NP2↵

の様にします。

各オプションの説明

N 中間ファイルとオプジェクトファイルを作成しません。

P リスト出力をページ編成にします。即ち66行／1ページでフォートランのプログラム単位ごとに改ページします。

1 ソースプログラムの各行の66けた目まで空白を詰めます。

2 ソースプログラムの各行の72けた目まで空白を詰めます。

＜注＞通常ソース・プログラムの各行には空白を詰めません。この問題は継続行の処理の時に起こります。FORTⅣの標準的な文法では継続行の7けた目が前行の72けた目に続く規約になっています。文字定数が二行に分かれている時に72けたまで空白を詰めるかどうかが問題になります。例をあげます。

```
10      FORMAT ('THIS IS AN
     *␣EXAMPLE')
```

を空白を詰めないでコンパイルすると実行時には

```
THIS IS AN EXAMPLE
```

となりますが，2のオプションを付けると，

```
THIS IS AN－49個の空白－EXAMPLE
```

となります。これらのオプションは，昔のパンチカードに打たれたプログラムをフォートランでコンパイル，実行する時以外は使う必要はありません。

H　Pオプションと共に使用します。リスト出力にヘッダーをつけます。

C＝XXXX　コモン共通ブロックの数の最大値を指定します。10進数で指定します。省略値は15です。

B＝XXXX　ソースプログラムの論理行の最大字数を10進数で指定します。継続行も含めた文字数です。省略値は530です。(文字定数外の空白は数えません。)

M＝XXXX　無名コモン(無名共通ブロック)の最下位アドレスを指定します。すなわち，この値に0BFFF番地を指定すると，共通ブロックの大きさが512バイトならば，0BE00Hから0BFFFHまでが共通ブロックとなります。16進数で指定します。

このMオプションはCHAINを使ってプログラム間でデータを引渡す為に使います。

＜例＞

```
A>FORT MYPROG $C=20↵
```

デフォルトドライブ上のMYPROG.FORをコンパイルして同じドライブ上に，.ASM，.LST，.OBJファイルを作成します。コモンブロックは20個まで許します。

```
A>FORT TEST.YZZ $P↵
```

デフォルトドライブ上のTEST.FORをコンパイルします。リスト出力をCP/MのLST:デバイスに出力し，中間ファイル，オブジェクトファイルを作りません。プログラム単位ごとに改ページするオプションをつけました。

2.4 プログラムの実行

オブジェクトファイルができたら，

A＞FRUN␣U：ファイル名↵

とする事によってプログラムが実行できます。U：はファイル.OBJの存在するドライブの名前です。もしデフォルトドライブ上に存在する場合は指定する必要はありません。実行時ルーチンは，100H番地から3FFFH番地までの16KByteを占めます。オブジェクトファイルは4000H以降にロードされます。

プログラムが正常終了するか又はエラーによって異常終了するかによって，CP／Mのコマンド待ち状態に戻ります。

＜例＞

コンパイルと実行の例を示します。

A＞FORT␣GRAPH↵

A＞FRUN␣GRAPH↵

リスト出力とオブジェクトファイルをデフォルトドライブに作成します。そしてプログラムを実行します。

2.5 COMファイル(実行形式のファイル)の作成

CP／Mの実行形式のファイルを作成する事ができます。実行形式のファイルは，FRUN.COMとオブジェクトファイルを合わせたものになりますからオブジェクトファイルに比べて16Kバイト程大きくなります。実行形式のファイルを作成する為には，

A＞FRUN␣GRAPH.C↵

とします。　GRAPH. COMが作られます。このプログラムを実行する為には，

A>GRAPH↵

とします。

3・FORTRAN言語の文法

3.1 FORTRANで使える文字

英字 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

数字 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

特殊文字

␣ 空白 （␣で表わす）

＝ 等号

＋ プラス

－ マイナス

＊ アステリスク

／ スラッシュ（斜線）

（ 左かっこ （ left parenthesis ）

） 右かっこ （ right parenthesis ）

, コンマ

. 点 （小数点）

$ ダラー記号

＃　ナンバー記号

＆　アンド記号　（ ampesand ）

＼　バックスラッシュ

：　コロン

；　セミコロン

などのアスキー記号が使えます。

＜注＞ 小文字は，文字定数以外では大文字に変換されます。

以下の特殊文字はこのFORTRANでは特殊な意味があります。

＄　ダラー記号は以下に続く０～９，Ａ～Ｆと合わせて１６進数として扱う事を示します。４．３．１を参照して下さい。

＃　ナンバー記号は以下に続く０～９，Ａ～Ｆと合わせて１６進数として扱う事を示します。４．３．１を参照して下さい。

＆　アンド記号は，使われる場所によって２つの役目があります。

1）FORMAT文中で文字定数の中で＆を使用した場合は，以下に続くＡ～Ｚ，などと合わせてＡＳＣＩＩのコントロール記号を示します。
例えば　＆Ｊはラインフィードを示します。
（１０．５．１４を参照して下さい）

2）サブルーチン副プログラム中では，多重リターンの戻り番地を示します。
（９．５を参照して下さい）

＼　バックスラッシュは，アスキーコードを16進数で表わす為に用います。
（4．3．2を参照して下さい）

コンソールによっては¥記号の場合があります。JISコードでは¥です。

3.2 FORTRANの文と行とプログラム単位

FORTⅣの規約に従っています。ただし，オプションの選択によって，行の最後に詰められる空白の扱いが違います（2．3で説明しました）。各行はキャリッジリターンで区切られます。文は数行に渡る場合や，一行に複数個存在する場合も許されます。（3．4を参照して下さい）（参考書P．18参照）

ただしEND行はEND文となっています。（FORTⅣからの拡張）

プログラム単位とは，主プログラムか副プログラムの事を言います。主プログラムはSUBRUTINEとFUNCTIONとでない文から始まりEND文で終る一連の文の集まりで，実行可能プログラムには一つしか存在してはなりません。副プログラムはSUBROUTINE文かFUNCTION文で始まりEND文で終る一連の文の集まりです。実行可能プログラムは一つの主プログラムと，いくつかの副プログラムとから成っています。

3.3 FORTRANの文

FORTⅣの規約に従います。ただし複数行に渡る文は空白（文字定数の内の空白は除く）を含めないで530文字以内でなければなりません。（これはBオプションで変更できます）

3.4 複文

文は一行に複数個書く事ができます。各文はセミコロンで区切る事が必要で

す。その文がラベル（文番号）を持つ場合は，：で文番号と文を区切ります。例えば，

```
      A=1
3     CONTINUE
      A=A+1
      TYPE  A
      GO  TO  3
      END
```

は複文として書くと

```
      A=1；3：CONTINUE；A=A+1；TYPE␣A；
GOTO  3；  END
```

3.5 FORTRANプログラムの書き方

CP/Mのテキストエディタを使用して，FORTⅣの規約と前章までに説明した拡張された規約を守ってプログラムを書きます。プログラム単位ごとにその直前にOPTION文を付ける事ができます。

.FORファイルはすべての副プログラムを含んでいなければなりません。すなわち，各プログラム単位を別々にコンパイルする事はできません。（3．6を参照して下さい）

3.6 COPY文

文法：COPY␣ドライブ名：ファイル名

このFORTRANは各プログラム単位を別々にコンパイルする事はできませんが，COPY文を使用する事によってソースファイルを小さくする事ができます。例えば，あるプログラム単位を別のソースファイルとしてディスケット上に置いておき，コンパイル時にとり込む事ができます。例えば，Bドライブ上のADDIT. FORというファイルにADDITというサブルーチン副プログが用意されている時に，以下の様にする事ができます。

```
A=1
CALL ADDIT(A, T)
WRITE(1, *)T
STOP
END
COPY␣B:ADDIT
```

COPY文はファイル名の前に一文字以上の空白を入れなければなりません。COPY文はネストできません。COPY文で読み込まれるファイルの中にCOPY文が存在してはなりません。

3.7 OPTIONS文

文法：OPTIONS␣オプション並び

各オプションはコンマで区切ります。

OPTIONS文は以下に続くプログラム単位のコンパイル時のオプションを指定する為の文で，プログラム単位の先頭に存在しなければなりません(なくてもよい)。以下にそのパラメタの指定の仕方と意味を示します。各パラメタはコンマで区切り，最初のパラメタの前には空白を入れて下さい。 OPTIONS文の有効範囲は以下に続く一つのプログラム単位だけです。

S＝n　nは10進数でプログラム単位に存在し得る変数の数の最大値を指定します。省略値は50です。

L＝n　nは10進数でプログラム単位に存在し得る文番号の数の最大値を指定します。省略値は50です。

T＝n　nは10進数で一つの式の評価をする時の一時的な変数の数の最大値を指定します。省略値は15です。

D＝n　nは10進数でDO－ループの入れ子の数の最大値を指定します。省略値は5です。

A＝n　nは10進数で，配列の個数の最大値を指定します。省略値は15です。

O＝n　nは10進数で式の評価の際の演算子などの数の最大値を指定します。コンパイル時に3Fの番号のエラーが起ったらnを増して下さい。省略値は40です。

P＝n　nは10進数で式の評価の際の変数や定数の数の最大値を指定します。コンパイル時に3Fの番号のエラーが起ったらnを増して下さい。省略値は40です。

I＝n　nは10進数でIF－THEN－ELSE節の入れ子の数の最大値を指定します。省略値は5です。

E　リスト出力に変数や定数の参照表を作成する事を指示します。

G　リスト出力にエラーメッセージのかわりにエラー番号を出力する様に指示します。

X　各文にISN（内部文番号）を付けて，実行時エラーがどの文で発生したかわかる様にする事をコンパイラに指示します。このオプションをつけると各文に対して5バイトづつオブジェクトファイルが長くなりますが，デバッグの際には便利です。

N　中間ファイル，オブジェクトファイルを作成しない事を指示します。文法エラーを発見する為のコンパイルである事を意味します。

B　FORTRANの各文を中間ファイル（.ASMファイル）にアセンブラの注釈として挿入する事を指示します。

Q　　ユーザが実行時エラー処理を自分でしようと考えている場合にはこのオプションを付けなければなりません。（６．１０を参照して下さい）

Ｓ，Ｌオプション以外はｎは２５５以下です。Ｓ，Ｌ，Ｔ，Ｄ，Ａ，Ｏ，Ｐ，Ｉの各オプションはその省略値をＣＯＮＦＩＧ時に変更できます。

＜例＞

$OPTIONS　X, G, S＝２００, L＝１００

Ｘ，Ｇ，Ｓ，Ｌの各オプションを変更又は指定しました。

4・数、定数、変数型、共通ブロック

4.1 数の内部表現

このFORTRANでは数値の内部表現として以下の様な形式を採用しています。すなわち一つの数値は6バイトを占めます。6バイトのうちの4バイトに，8けたの仮数部がBCD形式（2進化10進形式）が，次の1バイトに正負の区別が，その次の1バイトに指数部がおさめられています。これによって，0.10000000E－127から，0.99999999E＋127までの正負の数と0が表現できます。このFORTRANの実数は，この範囲となります。指数部は，符号付き整数（2の補数表示）に80Hを加えたものとなっています。0は，指数部を0として表現しています。

9 9	9 9	9 9	9 9	符 号	指 数
上位バイト	仮数			下位バイト	

4.2 数値の範囲

数は実数も整数も4．1で説明した表現を使用しています。従って実数は4．1で説明した通りで整数は－99999999から99999999までです。

実数の範囲

- 0．0
- －．99999999E＋127≦r≦－0．1E－127
- 0．1E－127≦r≦0．99999999E＋127

整数の範囲

$$-99999999 \leqq i \leqq 99999999$$

4.3 定数

4.3.1 整定数と実定数

整定数と実定数はFORTⅣの規約に従ったものの他に，$記号，#記号を用いて16進数で与える事ができます。

$記号では－$FFFFから＋$FFFFまで，
#記号では#0000から#FFFFまでです。

$記号と#記号は全く異なるものなので注意して下さい。$記号は，内部表現に変換してから格納されますが，#記号はそのまま格納されます。即ち，

$805Fは内部では　32　86　30　00　00　85

となりますが，

#805Fは内部では　5F　80　00　00　00　00

となります。この違いを理解してから使用して下さい。805FHは10進数では32，863ですから，$記号の方は　0.32863E5　として格納されたのです。

4.3.2 文字定数

文字定数は引用符（'）で囲みます。文字定数の中に引用符を使用したい場合は，二つの引用符を続けて入力します。バックスラッシュで囲んで，アスキーコードを16進数で指定する事ができます。

＜例＞

```
'This is a string constant'
'Single quote is ( '' )'

'GOOD＼21＼"
```

'' は一つの引用符を，＼21＼はアスキーコードの21H(！)を表わします。

文字定数内に＼0＼を使用してはいけません。＼0＼は文字定数の終りを指示示します。／記号（バックスラッシュ）はCONFIG時に変更できます。

4.3.3 論理定数

論理定数には .TRUE. と .FALSE. があります。それぞれ真，偽を表わします。内部的には，.FALSE. が0で，.TRUE. が1であらわされてます。ただし，0以上の数はすべて．TRUE．として扱われます。

4.4 変数名

変数名はFORTⅣの規約に従っています。英字で始まる6文字以内の英数字です。

＜例＞

```
ABJ

A38

JGMJGM
```

4.5 型の指定（宣言）

型の指定は暗黙的な型宣言も含めてFORTⅣの規約に従っています。ただし型の種類は4種類です。

文法：

INTEGER v_1, v_2, ……, v_n

REAL v_1, v_2, v_3, ……v_n

LOGICAL v_1, v_2, ……, v_n

DOUBLE PRECISION v_1, v_2, ……, v_n

v_i は変数名，配列名，関数名または配列宣言子

複素数型は使用できません。また，DOUBLE PRECISION（倍精度実数型）はプログラムの内部ではREAL（実数型）と同じものとして扱われますから他機種からのプログラムの移植の際には注意が必要です。論理型は必ず宣言しなければなりません。各変数は一つの型しか持つ事ができません。型宣言文はIMPLICIT文に優先しますし，IMPLICIT文は暗黙的な型宣言に優先します。（暗黙的な型宣言とは，頭文字がA－H，O－Zの変数名を実数型として，I－Mの変数名を整数型として変数を使うというFORTⅣの規約の事でIMPLICITとは異なった意味に用いています。）

＜例＞

```
INTEGER  A, ZOT, ZAP(10)

REAL  R1, RJ, INT(100)

LOGICAL  INIT, FLAG1, LOG1
```

4.6 DATA文

文法：DATA␣v／d_1, d_2, ……d_m／

d_i は定数。ただしvが配列名の場合に限って複数個並べる事ができます。

DATA文はFORTⅣの規約に従っています。ただし以下の形のものに限って使えません。

DATA A, B, C /1, 2, 3/ ×

DATA A/1/, B/2/, C/3/ ○

2番目の書き方に従って下さい。

無名共通ブロック内の変数にはDATA文で初期値を与える事はできません。配列に初期値を与える場合は要素の数とデータの数が異なっていてもかまいません。DATA文は，プログラム単位をすべてコンパイルし終ってから評価されますからDATA文にエラーが存在した場合にはEND文の後に表示されます。DATA文で初期値を与えられた変数がプログラム単位の中のどこにも現われていない場合にもエラーが起こります。

4.7 共通ブロック（COMMON文）

COMMON文はFORTⅣの規約に従っています。無名共通ブロック（ブランクコモン）は，CP/MのTPAの最終番地におかれます。（2．3Mオプションを参照）ブランクコモンは初期化されませんから同じサイズのコモンブロックを使用すればCHAIN（11．2を参照して下さい）時にデータの受け渡しができます。

文法：COMMON/ブロック名1/v_1, v_2, ……, v_n/ブロック名2/v_{n+1}, v_{n+2}, ……v_{n+m}

ブロック名1はなくてもよい。その場合2本の斜線もなくてもよい。

名前付コモンの名前は関数名やサブルーチン副プログラム名とは違う名前を付けて下さい。名前付きコモンの名前は5文字以内です。

コモンの名前として，A, B, C, D, E, H, L, M, SP, PSWは許されていません。

4.8 IMPLICIT文

IMPLICIT文はFORTⅣには存在しません。暗黙の型宣言を変更する為に使用します。即ち，FORTⅣでは　型宣言をしなかった場合にはA－H，O－Zを頭文字とする変数を実数型，I，J，K，L，M，Nを頭文字とする変数を整数型とします。IMPLICIT文は以下の様にこれらの暗黙の型宣言を変更します。

文法：IMPLICIT␣型（範囲），型（範囲）

＜例＞

```
IMPLICIT  INTEGER(Z, A-E)
```

頭文字がZ，A，B，C，D，Eではじまる変数を整数型として宣言します。ハイフン（マイナス）は，その間の文字すべてを意味します。

```
IMPLICIT  INTEGER(A-Z)
```

すべての変数を整数型とします。

IMPLICIT文は，プログラム単位の中ではSUBROUTINE文あるいはFUNCTION文の直後になくてはなりません（なくてもよい）。主プログラムでは一番始め（$OPTIONS文の次）に存在しなくてはなりません（なくてもよい）。

5・式

式には三種類あります。算術式，関係式，そして論理式です。算術式は，定数，変数，配列要素などを算術演算子で結合したものでその構成要素によって整数型，実数型のどちらかの型を持ちます。関係式は二つの算術式を関係演算子で結んだもので，真か偽かの値を持ちます。論理式は，論理型の変数や関係式を論理演算子で結んだもので真か偽かの値を持ちます。

算術演算子	＊＊ 又は ∧	べき乗
	＊	乗算
	／	除算
	＋	加算
	－	減算
関係演算子	.EQ.	等しい
	.NE.	等しくない
	.GT.	大きい
	.LT.	小さい
	.GE.	大きいか等しい
	.LE.	小さいか等しい
論理演算子	.NOT.	否定
	.AND.	論理積
	.OR.	論理和
	.XOR.	排他的論理和

∧ と.XOR. はFORTⅣ からの拡張です。
.NOT. と －（負符号の場合）は単項演算子です。

5.1 演算子の結合の強さ

算術演算子と関係演算子，論理演算子が同一式中にあり，その演算順序がか

っこによって完全には規定されていない時には，つぎの順序で評価されます。
（強～弱）

強　関数の評価
　　べき乗　　＊＊　又は　＾

　　乗除算
　　加減算　（単項演算子の負符号を含む）

　　関係演算

　　否定
　　論理積
　　論理和又は排他的論理和

弱　代入

5.2 式の評価

式の評価は以下の順序で行なわれます。

1)　かっこで囲われた式の評価

2)　かっこの中は前節の結合の強さの順序で評価されます。

3)　同じ強さの演算子は左から右に評価されます。

5.3 整数の演算

整数と実数の違いはけたあふれと除算の際の小数点以下の切り捨てとにあります。

除数と被除数が共に整数である場合には，結果の小数点以下は切り捨てられます。例えば４／３は１となります。

乗算の場合に整数ならば99999999を越えるとけたあふれが起ります。例えば　10000×12000はエラーになります。（ INT RANG ERROR ）

＜注＞

・通常のFORTRAN では整数の桁あふれはおこらない事になっています。移植の際には注意して下さい。

5.4 実数の演算

実数の演算は，結果が実数の範囲を越えたら　OVERFLOW ERROR になります。

0．0 での除算は，エラーとなります。

5.5 論理型の演算

論理型の演算は実数型，整数型の演算とは異なっています。結果は真（１）か偽（０）のどちらかになります。関係演算子は，２つの算術式の関係が真であれば．TRUE．，偽であれば．FALSE．となります。真と１と．TRUE．，偽と０と．FALSE．は同じ事を意味します。

＜例＞

```
A = 1.GT.2      Aは0となります
A = 1.EQ.1      Aは1となります
A = .TRUE..OR..FALSE.  Aは1となります。
A = P.AND.Q     PとQの値の真偽に応じてAの値が決まります。
```

．AND．，．OR．，．XOR．，．NOT．は通常の真理値表に従います。

排他的論理和（．XOR．）はFORTⅣからの拡張です。

5.6 演算に於ける型の混合

FORT Ⅳ では演算や代入に於て型は一致していなければなりませんが，以下の例外があります。

実数　演算子　整数 整数　演算子　実数	これらの場合は，演算の前に整数は実数型に変更され，演算が行なわれます。結果は実数型となります。
整数　＝　実数	実数の，小数点以下が切り捨てられて整数になります。切り捨てられた結果が，整数の範囲を越えていたら INTRANG ERROR になります。
実数　＝　整数	整数を実数型に変更して代入します。

＜注＞ この場合の演算子は　＋，－，／，＊　です。

べき乗の場合は，整数の実数乗は許されていません。整数の整数乗は整数となります。

6・制御文

制御文はプログラムの流れを制御する為の文です。FORTⅣの規約に従うものの他に構造化をサポートする為にIF－THEN－ELSE－ENDIF節が用意されています。

6.1 単純GO TO 文

FORT Ⅳ の規約に従っていますが，拡張としてFORMAT文をGO TO 文の 飛び先に指定する事ができます。この場合FORMAT 文はCONTINUE文と同じ役目を果します。

文法：GOTO 文番号

6.2 計算型GO TO 文

FORTⅣの規約に従っています。ただし整変数の数が1より小さかった場合と，文番号の数を越えた場合とにはCOM GOTO ERRORが実行時に起ります。

文法：GOTO(n_1, n_2, n_3, ……n_m), i

nは文番号，iは整変数

iが$1 \leqq i \leqq m$以外の場合，他のFORTRANと解釈が異なる場合があります。

<例>

```
GOTO( 10, 20, 30, 40 ), I

GOTO( 1, 1, 1, 999, 1, 1, 999 ), J
```

6.3 割り当てGO TO 文

FORTⅣの規約に従っています。ASSIGN文で割り当てられていない場合や，定義されていない文番号に割り当てられている場合は ASN GOTO ERROR が実行時に起ります。

文法：GOTO k,（n_1, n_2, ……n_m）

kは整変数。nは文番号

kに与えられる可能性のある文番号はあらかじめかっこの内にすべて登録しておかなければなりません。

6.4 ASSIGN文

FORTⅣの規約に従っています。

文法：ASSIGN n TO k

nは文番号，kは割り当てGOTO文の中で用いる整変数

この文で定義した整変数は再定義されない限り割り当てGOTO 文以外では引用する事ができません。

＜例＞

```
ASSIGN 20 TO LAB
  ⋮
IF(KN.GT.3)ASSIGN 10 TO LAB
  ⋮
GO TO LAB,(10,20)
```

6.5 算術IF文

FORTⅣの規約に従っています。3つの飛び先のうちのいくつかを省略する事はできません。

文法：IF （算術式）n_1，n_2，n_3

nは文番号。

nは必ず3つ必要です。他のFORTRANでは2つの場合が許されている事があります。

算術式の値が負ならn_1に，0ならn_2に，正ならn_3に飛びます。

＜例＞

```
IF(DATA*GAMMA)100, 200, 300
IF(I-1)1, 1, 99
```

6.6 論理IF文

FORTⅣの規約に従っています。ただし論理式の後の実行文としてEND文と，IF文，DO文を用いることはできません。

文法：IF(論理式) 文

論理式の値が真ならば文を実行します。

＜例＞

```
IF(DEG. EQ. 28. 1)WRITE(1, *)QVA
IF(I. GE. 2. AND. I. LE. 4)RETURN
```

6.7 IF-THEN-ELSE-ENDIF節

FORTⅣからの拡張として構造化をサポートする為に用意されています。文法は以下の通りです。

文法：ＩＦ（論理式）ＴＨＥＮ

　　文
　　文
　　……
　ＥＬＳＥ
　　文
　　文
　　……

　ＥＮＤＩＦ

説明：論理式の値が真ならば，ＴＨＥＮ以下，ＥＬＳＥ文の直前までの文を実行しその後にＥＮＤＩＦ文の直後の文に制御が移ります。

論理式の値が偽ならばＥＬＳＥ文の直後の文からＥＮＤＩＦの直前の文までを実行し，その後はＥＮＤＩＦ文の直後の文に制御が移ります。

ＥＬＳＥ文からＥＮＤＩＦ文の直前までの文を省略する事ができます。この場合，論理式の条件が偽ならば直ちにＥＮＤＩＦ文の次の実行文に制御が移ります。

＜注＞ ＩＦ－ＴＨＥＮ－ＥＬＳＥ－ＥＮＤＩＦ節は入れ子にする事ができます。ただし，完全な入れ子構造になっていなければなりません。

ＩＦ－ＴＨＥＮ－ＥＬＳＥ－ＥＮＤＩＦ節に於て，ＴＨＥＮの後，ＥＬＳＥの後，ＥＮＤＩＦの後は↵か；で区切らなければなりません。これは，

それぞれが独立した文だからです。また，ELSE文，ENDIF文には文番号を付ける事はできません。

この節の中から外へ制御を移す事はできますが，外からこの節の中へ制御を移す事はできません。

6.8 DO文

FORTⅣの規約から拡張されています。DOの端末文（DO␣文番号……の，文番号をもつ文）は，他のDO文，GO TO文，STOP文，IF－THEN文，ELSE文，ENDIF文，END文，RETURN文であってはいけません。これらのものになってしまう場合には，CONTINUE文を用いて下さい。

＜注＞ FORTⅣの規約では以上の他に算術IF文と，以上のいずれかを含む論理IF文，PAUSE文も端末文とはなり得ない事になっています。

＜注＞ DOの範囲（DO文の次の文から端末文まで）でDOの制御変数，初期値パラメタ，終値パラメタ，増分パラメタを変更してはいけません。

＜注＞ FORTⅣでは，DOの範囲を必ず一度は実行します。FORT 77で書かれたプログラムをα－FORTRANでコンパイルする場合には注意して下さい。

文法：DO　n　i＝m1，m2，m3

nはDOの端末文の文番号，iは　実または整変数，m1，m2，m3は実変数，整変数，実定数，整定数。i，mには配列要素を用いる事はできません。また，m3は省略する事ができます。その場合にm3は1となります。iをDOの制御変数，m1，m2，m3をそれぞれ初期値，終値，増分パラメタといいます。

＜例＞

```
DO  10  I=1, 100
DO  2  J=I, END, 0.5
DO  72  A=START, END, AINCR

DO  10  I=1, -1
```

最後の例でもDOの範囲を一度は実行してしすいます。

6.9 CONTINUE文

FORTⅣの規約に従っています。

文法：文番号　　CONTINUE

6.10 ERRORトラップ

通常実行時にエラーが起るとこのFORTRANでは実行時処理ルーチンがエラーメッセージを出力します。このFORTRANではERRSET文とERRCLR文を用意する事によって利用者がエラー処理を行う事を許しています。

文法：ERRSET　文番号，変数名

ERRCLR

説明：ERRSET文はその文の存在するプログラム単位内でエラー発生時に制御を移す実行文の文番号と，エラー番号を返す変数名を設定します。
ERRSET文は，OPTIONS文でQを指定した時に限って有効となります。
ERRSET文が有効となった場合は実行時処理ルーチンはエラーメッセージを表示しません。
返されたエラー番号とその意味は11．5を参照して下さい。

ERRSET文が同じプログラム単位内で複数個実行した場合は最後に実行したERRSET文が有効です。

ERRCLR文は，ERRSETをクリヤしてエラー処理を実行時処理ルーチンに委ねます。

＜例＞

```
OPTIONS Q
        ⋮
        ERRSET 10, CODE
        ⋮
        ERRCLR
        ⋮
        STOP
10      TYPE 'ERROR CODE =', CODE
        ⋮
        END
```

ERRSET文実行後，ERRCLR文を実行するまでにエラーが起った場合は，TYPE文を実行します。

6.11 コントロールC制御

このFORTRANではCP/Mのアボートであるコントロール C に対して注意を払っています。すなわち，

（1）REAC文又はWRITE文を実行する前に，コントロールC－アボートフラグを調べます。

（2）もし，コントロールC－アボートフラグがセットされていたら，コンソールのステイタスを調べ，コントロールCキーが押されているかどうか調べます。もしコントロールCキーが押されていたら，ERRSETが有効かどうかによって以下のどちらかが起ります。

（a）　ERRSET文が有効な場合は，コントロールCエラーコードを返して制御がERRSET文で指定された文番号へ移ります。

（b）　ERRSET文が有効でない場合は，実行時処理ルーチンが，コントロールC－エラーメッセージを表示します。

利用者は，この過程でコントロールCアボートフラグをセットしたりリセットしたりする事によって，コントロールCに対する応答を変更する事ができます。

CTRL␣ENABLE文はこのフラグをセットします。

CTRL␣DISABLE文はこのフラグをリセットします。

CTRL␣DISABLE文が実行されるとコンソールからのコントロールCは無視されます。

＜注＞ CIN関数（11. 3参照）はコントロールCも読み込んでしまいます。これはアボートフラグには依存しません。

6.12 プログラムの実行のトレース

TRACE ON

TRACE OFF

文でこのFORTRANのトレース機能を利用できます。プログラムの実行開始の時点ではトレース機能はOFFになっています。もしトレースを開始したいならば，TRACE ON文を実行しなければなりません。トレース機能は大

域的なものです。すなわち，プログラム単位を越えてトレースします。従って一旦トレース機能をＯＮにしたら，他のサブルーチンや関数に制御が移ってもトレースは続けられます。あるプログラム単位で，ＯＰＴＩＯＮＳ文でＸＯＰＴＩＯＮを指定しておけば，トレース機能によって，その文を実行する直前に，その文のＩＳＮを表示します。Ｘオプションが指定されていない場合はサブルーチンやファンクションに制御が移った時点でその旨が表示されるだけです。

6.13 DUMP文

DUMP文は実行時のエラーを発見する為の，デバッグ用の文です。ERRSET文によって捕えられない様な実行時のエラーを表示する事ができます。

文法：ＤＵＭＰ　／識別子／　出力並び

説明：識別子は１０文字以下の文字列で，どのＤＵＭＰ文による出力かを識別します。出力並びはＷＲＩＴＥ文の出力並びと同じです。

ＤＵＭＰ文はプログラム単位ごとにそれぞれのＤＵＭＰ文を用意する事ができます。一つのプログラム単位の中に複数個のＤＵＭＰ文を実行させる事ができますが，そのプログラム単位中で一番最後に実行したＤＵＭＰ文が有効となります。

＜例＞

```
DUMP  /AFTER-DIV/'K  =', K
  ・
  ・
  ・
A=K/I
  ・
  ・
  ・
END
```

この例は，もしＩが０で，Ａ＝Ｋ／Ｉ文が実行されるとＤＵＭＰ文の効果によってコンソールに

Ｋ　＝　ｋの値

と表示されます。

7・プログラムを終了させる者の制御文

7.1 PAUSE文

PAUSE文はFORTⅣから拡張されています。
ただしPAUSE␣nは許されていません。

文法： PAUSE　　または

　　PAUSE　'文字列'

説明：

任意の長さの文字列が許されています。PAUSE文を実行するとコンソールに文字列を出力してプログラムの実行を中断します。コンソールから任意のキーを押す事によってプログラムを再開する事ができます。

7.2 STOP文

STOP文はFORTⅣから拡張されています。

文法： STOP　'文字列'　　または

　　STOP　n　　または

　　STOP

STOP文は，プログラムを終了させます。コンソールに，文字列か，または5けた以内の整数か，またはSTOPだけかを表示してプログラムは終了します。（CP/Mのコマンド待ち状態に戻ります。）

STOP文のかわりにCALL␣EXITとする事によってもプログラムを終了させる事ができます。この場合はコンソールに何も表示せず，CP/Mの

コマンド待ち状態に戻ります。

7.3 END文

FORTⅣではEND行となっていますが，このFORTRANではEND文となっています。この違いは，複文にしてもよいし，2行にまたがってもよいという意味です。

プログラム単位をコンパイラに指示するのがEND文です。一つのプログラム単位はその最終文がEND文でなければなりませんし1つのプログラム単位には2つ以上のEND文は存在できません。

また，END文は実行文として実際にプログラムを終了させる事ができます。この場合，このFORTRANは

```
STOP  END  IN  -  XXXXX
```

と表示してプログラムを終了させます。XXXXXは，END文の存在したプログラム単位の名前です。

8・配列

配列に関してはFORTⅣの規約に従っています。このFORTRANに於ける配列に対する制限は以下の通りです。

（1）次元は7次元までです。

（2）一つの配列の要素の数が5461を越えてはいけません。

配列の宣言に関する注意を8．1に述べますが，一般的な事はFORTⅣの規約に従っていますから参考書などを参照して下さい。

8.1 配列の宣言（DIMENSION文）

DIMENSION文はFORTⅣの規約に従っています。配列の寸法は主プログラムでは必ず整定数でなければなりませんが，サブルーチン副プログラム，関数副プログラムの中では整変数が許されています。整合配列に関しては参考書などを参照して下さい。

文法：DIMENSION v(n_1, n_2, ……, n_m), ……

vは配列名，n_1 は整変数名あるいは整定数　$m \leqq 7$，$n_1 * n_2 * \cdots\cdots n_m \leqq 5461$

＜例＞

```
REAL MOMENT
DIMENSION MOMENT(1000)

DIMENSION A(2, 4, 6), B(3, 2, 4, 6)
```

プログラム実行中には，配列の添字が寸法を越えているかどうかを調べる事

はしません。従って以下の様な事が可能になります。

```
COMMON／DUMMY／A( 1000 ), B( 1000 )
```

としておく事により，A(1320)としてB(320)をアクセスする事ができます。

配列の占めるメモリ上の容量は，要素数＊6バイトです。これは整数型，実数型，論理型，倍精度実数型を通して共通です。

配列の宣言は各配列に対して一度しかしてはいけません。

8.2 添字

配列の添字は，FORTⅣからの拡張として，定数と変数（配列要素と関数は使えない）の算術式が許されています。実数型の算術式も許されていますが，式が評価された後に小数点以下を切り捨てます。式については5．2を参照して下さい。

＜添字の使用例＞

```
ZI( 8 )
A( I+3 )

A( K+J )=A( K )+A( J )

A( K )=A( K )+B( J, K )*C( K, I )

A( 2. 3 )はA( 2 )を指します。
```

9・副プログラム

副プログラムはFORTⅣの規約に従っています。副プログラムは3種類に分けられます。すなわち

1） サブルーチン副プログラム

2） 関数副プログラム

3） 組込み関数

FORTⅣの文関数はサポートされていません。関数副プログラムに書き直して下さい。

定数を副プログラムへ実引数として引き渡す場合は副プログラム中で書き変えない様にして下さい。その場合，正常な動作は保証されません。副プログラム名は5文字以内でA，B，C，D，E，H，L，PSWという名前は使えません。

すべての副プログラムは主プログラムと同時にコンパイルしなくてはなりません。

9.1 サブルーチン文

SUBROUTINE文はFORTⅣの規約に従っています。サブルーチン名は5文字以内でなければなりません。

文法： SUBROUTINE サブルーチン名(v_1 , v_2 , v_3 , …… v_n)

v_i は 変数名，配列名

v_i として外部手続きの仮の名前は許されていません。配列名の場合は，サ

ブルーチン副プログラム内で宣言されなければなりません。整合配列は許されていますが，配列の宣言子添字は v_1～v_nの中に含まれていなければなりません。

（　）内は仮引数を使用しない場合は省略できます。

＜整合配列の例＞

```
SUBROUTINE(N, DIM, J)
DIMENSION DIM(N)
   :
```

9.2 FUNCTION文

FUNCTION文はFORTⅣの規約に従っています。関数名は５文字以内でなければなりません。関数の型は暗黙的な約束に従って副プログラム名で指定するかFUNCTION文のFUNCTIONという語の前に，INTEGER, REAL, DOUBLE PRECISION, LOGICALをつける事によって宣言しなければいけません。FUNCTION文では必ず一つ以上の仮引数を持たねばなりません。関数副プログラム名はその副プログラム中で変数名として副プログラムが呼ばれる度に代入されなければなりません。

文法：関数の型　FUNCTION　ファンクション名（v_1……v_n）

実数の型はINTER, REAL, DOUBLE PRECISION, LOGICALのいずれか，またはなくてもよい。

v_1～v_m についてはサブルーチン文を参照（９．１）ただし関数の場合は省略する事はできない。

9.3 CALL文

CALL文はFORTⅣの規約に従っています。サブルーチンの呼び出しの際には参照する側と参照される側の実引数の数と型が一致していなければなりません。この規約を破ると実行時にARG　CNTエラーが起こります。

文法： CALL　サブルーチン名（実引数並び）

＜例＞

```
CALL  SUB( J, K, N)

CALL  NOAUG
```

9.4 RETURN文

FORTⅣのRETURN文と，その拡張としての多重リターン文が用意されています。副プログラム中でRETURN文を実行する事によって，副プログラムを呼び出していたプログラム単位に戻ります。主プログラム中にRETURN文が存在してはいけません。

9.5 多重リターン文

多重リターン文はFORTⅣのRETURN文の拡張として用意されています。一部のFORTRANでサポートされていますが，文法的に異なっている場合がありますのでプログラムを移植する際には注意して下さい。

文法： RETURN　仮引数名

説明： 例によって説明します。

あるプログラム単位で

```
      ⋮
CALL  XR(&1, Y, 2, &2)
実行文a
      ⋮
```

```
1   実行文ｂ
    ……
2   実行文ｃ
    ……
    END
    SUBROUTINE  XR(I, A, ID, J)
    ……
    RETURN
    ……
    RETURN  I
    ……
    RETURN  J
    END
```

この例では，サブルーチン副プログラムＸＲの１番と４番目の仮引数が多重リターンの為の仮引数となっています。ＸＲの中で３ケ所のＲＥＴＵＲＮ文がありますが，そのなかで ＲＥＴＵＲＮ Ｉ の様に仮引数名を文末に付けているＲＥＴＵＲＮ文があり，また，その仮引数に対応する実引数には，＆つきの文番号が書かれています。この場合に，ＸＲ中で例えばＲＥＴＵＲＮ Ｉを実行すると，次はＩに対応する文番号１を持つ実行文ｂから実行が始まります。ＸＲ中でＲＥＴＵＲＮ Ｊが実行されると，ＣＡＬＬした側のルーチンの実行文Ｃから実行が再開されます。単なるＲＥＴＵＲＮ文を実行すると，ＣＡＬＬ文の次の文ａから実行が再開されます。この様に呼ばれた方のルーチンから，呼んだ方のルーチンのプログラムの流れを制御する事ができます。

多重リターンはサブルーチン副プログラムでのみ使用する事ができます。従って関数副プログラムでは使用する事ができません。

＜注＞ 多重リターン文はプログラムの流れをわかりにくくするので，ＩＦ文，

計算型ＧＯＴＯ文などを使用した方がよいと思います。

9.6 BLOCK DATA文

BLOCK DATA文はFORTⅣの規約に従っています。名前付共通ブロックを初期化する為の副プログラムです。実行文を含んではいけません。

＜注＞ 名前付き共通ブロックの初期化は，どのプログラム単位でも可能です。従ってBLOCK DATA文はFORTⅣとの互換性の為にだけ用意されています。

10・入出力

10.1 FORTRANの入出力文

10.1.1 一般的な説明

このFORTRANの入出力文には以下の様なものがあります。すなわち

（1） FORMAT付き入出力文

（2） 自由形式の入出力文

（3） FORMATなしの入出力文

FORMAT付き入出力文は，入出力並びに指定されたものを指定された書式仕様に従ってファイルやコンソールやリストデバイスとデータをやりとりします。出力の形式は書式仕様に従いますが基本的には右ぞろえです。

自由形式の入出力文はBASICのPRINT文，INPUT文に似ています。入力時は各変数をコンマで区切って入力すればよく，出力時は自動的に見やすい形に編集してくれます。

FORMATなしの入出力文は，高速に大量のデータをファイルに書いたり読んだりする為に使います。ディスクの使用量もFORMAT付き入力文に比べて格段に少なくなりますが，アスキーファイルではありませんから，CP／MのTYPEコマンドで見る事はできません。

このFORTRANは論理装置番号0と1をコンソールの入力と出力として割り当てています。これらの機番はOPENしたりCLOSEしたりする事ができません。FORMATなし入力も許されておらず，これらの操作を行なうと実行時エラーが発生します。

TYPE文とACCEPT文が，用意されています。コンソールとの入出力に使う事ができます。

配列内の書式仕様も用意されています。変数一つ当り6バイトを占めますので，A6を使用して下さい。詳しくは参考書P.253を参照して下さい。参考書のA8はA6と読み直して下さい。

＜配列内の書式仕様の例＞

```
 DIMENSION  FORM(10)
 READ(0, 10)'DATA  FORMAT?', FORM
10  FORMAT(10A6)
  ⋮
 READ(4, FORM)A, B, C
  ⋮
 WRITE(5, FORM)P, Q, R
  ⋮
```

コンソールから

（F10.3，F10.5，F10.7／）

と打ち込みます。左右のかっこも含めて打ち込みます。すると，この様なFORMAT文をプログラム中に書いておいたのと同じ様に入出力が行なわれます。コンパイルをし直さなくても，ダイナミックに書式を変更できるので便利です。

10.1.2 入出力並び

入出力並びはFORTⅣから拡張されています。入力と出力に共通です。即ち

（1）変数名
（2）配列要素名
（3）配列名
（4）DO形並び

更にFORTⅣの拡張として

（5）文字列
（6）定数　（出力の時のみです）

またはこれらをコンマで区切って並べたものです。

配列名は，配列名を書く事によってその配列の要素すべてを表示する事ができます。

DO形並びは，入出力並びのあとにコンマで区切って，

$i = m_1, m_2, m_3$

と書き（DO形仕様）全体をかっこでくくったものです。i，m_1，m_2，m_3 はDO文で定義したものと同じです。DO形並びは任意回入れ子にする事ができます。

＜例＞

```
WRITE(1, *)(F(I), I=1, 7, 2)
```

F(1)，F(3)，F(5)，F(7)を表示します。

```
WRITE(1, *)(J, F(I, J), I=1, 4), J=1, 30, 2)
```

```
1, F(1, 1), F(2, 1), F(3, 1), F(4, 1),
2, F(2, 1)·····
```
と表示します。

文字定数は入力文でも出力文でも使用する事ができます。入力文では，入力促進記号として，出力文では出力の識別として使用できます。

＜例＞

```
WRITE(1, *)'A=  ', A
TYPE  'The answer is  ', ANS

READ(0, *)'A  =', A
ACCEPT  'Enter quantity', QUANT
```

コンソール以外の入力文で文字定数を使用すると，実行時エラーとなります。

10.2 READ文

READ文はプログラム実行中にコンソールやファイルからデータなどを入力する為の文です。論理装置番号0はコンソールの為に予約されています。論理装置番号1はコンソール出力の為のものでREAD文では使用できません。それ以外の論理装置番号は入力する前にOPEN文を使用してOPENしなければなりません。FORMAT付きの入力文の場合は対応する文番号をもつ実行文はFORMAT文でなければなりません。文番号のかわりに書式を含む配列名を書いてもよろしい。更に＊を書いて自由形式のREAD文としてもよろしい。書式なしREAD文も使用する事ができます。

文法：READ(論理装置番号，FORMAT文の文番号)入力並び

READ(論理装置番号，＊)入力並び

READ(論理装置番号)入力並び

更に，どの形式のREAD文にもERR時，END時の飛び先を指定する事ができます。（省略する事はもちろんできます。）

文法：READ(u, f, ERR=文番号1, END=文番号2)k

uは論理装置番号，fは書式仕様です。文番号1は，FEAD時にエラーが起った時に制御を移すべき文号条，文番号2はEND OF FILE時に制御を移すべき文番号です。ERRは，bad sector などのエラーに対応するもので，書式と実際に存在するデータの形式が異なっていた場合(input format error)などには，ERRSET文で指定していない限り，プログラムは終了してしまいます。

＜例＞

```
      S=0
1     READ(4, 100, ERR=900, END=200)A
100␣FORMAT(F10.0)
      S=S+A
      GOTO1
200   TYPE  'SUM  ', S
      CALL  EXIT
900   TYPE  'INPUT  FILE  ERROR'
      CALL  EXIT
      END
```

入出力並びについては10. 1. 2で説明しました。

10.3 WRITE文

WRITE文は，プログラム実行の結果などをコンソールやファイルに出力する為の文です。装置番号1がコンソールへの出力の為に予約されています。0は使用できません。それ以外の装置番号はOPEN文又はLOPEN文でWRITE 文を実行する以前にOPENしておかねばなりません。

ＥＮＤ＝は使用する事ができますが何の機能も意味も持ちません。

文法：ＷＲＩＴＥ（装置番号，書式，ＥＮＤ＝文番号，ＥＲＲ＝文番号）出力並び

ＥＮＤ＝文番号，ＥＲＲ＝文番号は省略する事ができます。

＜例＞

```
ＷＲＩＴＥ（１，２）ＩＪ，ＰＡ，ＤＪ
ＷＲＩＴＥ（４）（Ｉ，Ｉ＝１，１０）
ＷＲＩＴＥ（６，１２，ＥＲＲ＝９９）ＬＯＯＰ，ＮＵＭ
```

10.4 MEMORYからMEMORYへの入出力文

ＦＯＲＴ Ⅳからの拡張としてＥＮＣＯＤＥ文，ＤＥＣＯＤＥ文が用意されています。ＥＮＣＯＤＥ文，ＤＥＣＯＤＥ文は，ＦＯＲＴ７７に於ける内部ファイルの前身に当るもので，変数間で形式の違う形のデータのやりとりを行う事ができます。ＢＡＳＩＣではＭＫＩ＄関数，ＣＶＩ関数に対応します。単純変数，配列要素などは６バイトを占める事に注意して下さい。

10.4.1 DECODE文

ＤＥＣＯＤＥ文はＲＥＡＤ文に似ています。メモリ内（変数内）にアスキーコードで書かれた数字から，ＦＯＲＴＲＡＮの数値の内部形式に変換します。長さはバイト単位で指定します。

文法：ＤＥＣＯＤＥ（変数名，長さ，書式）入力並び

変数名（配列名か単純変数名に限ります）

長さ，読みだすべきバイト数，

書式，FORMAT文の文番号

＜例＞

```
 INTEGER  YY, MM, DD
 DIMENSION  A(2)
 READ(4, 10)A
10  FORMAT(2A6)
    DECODE(A, 80, 11)YY, MM, DD
11  FORMAT(I2, X, I2, X, I2)
```

この例では，装置番号4のファイルから，A変換で年月日（84−02−14）を読み込み整数に変換しています。

10.4.2 ENCODE文

ENCODE文は，DECODE文の反対の動作を行ないます。

文法：ENCODE（変数名，長さ，書式）出力並び

＜例＞

```
 DIMENSION  A(2), E(2), J(2)
 INTEGER  YY, MM, DD
 YY=84
 MM=10
 DD=21
 ENCODE(A, 8, 10)mm, DD, YY
 ENCODE(E, 8, 10)DD, mm, YY
 ENCODE(J, 8, 10)YY, mm, DD
10  FORMAT(I2, '1', I2, '1', I2)
 WRITE(1, 11)'AMERICAN', A
 WRITE(1, 11)'EUROPIAN', E
 WRITE(1, 11)'JAPANESE', J)
```

11 FORMAT(X, A20, 2A6)

10.5 FORMAT文と書式記述子

FORMAT文はFORTⅣの規約に従う他に，いくつかの拡張が用意されています。FORMAT文は入出力の形式を制御する為の文で必ず文番号を付けて使用します。FORMAT文に関する細かな規約は参考書 P.255 ～を参照して下さい。ここではこのFORTRANに特徴的な欄記述子についてのみ詳しく説明します。

このFORTRANは実数値0をF変換，E変換，D変換にかかわらず0．0と表示します。0. 0000 …… と表示した場合は，表示けたより小さい値であった場合です。

また，F変換などで指定した範囲を越えた場合には＊＊＊＊で欄を埋めます。

＜例＞ F3．0で 1200．0を表示しようとした場合

＊＊＊ と表示されます。

F7．3で －123．456を表示しようとした場合

＊＊＊＊＊＊＊ と表示されます。

10.5.1 X変換 (wX)

wは1から255までの整定数です。空白をw個出力します。入力時はwけた読みとばします。

以下wは同じ意味です。

10.5.2 I 変換 (Iw)

入力時空白は0と同じ扱いとなります。整数は右ぞろえで読み込まれます。

␣␣␣␣␣␣␣2,7␣ をＩ１０で読み込んだ場合は２７０となります。

10.5.3 A変換 (Aw)

単純変数または配列要素には６文字入力，出力する事ができます。A変換は左ぞろえです。

10.5.4 ／区切り

／は記録（レコード）の区切りを示します。

10.5.5 Z区切り

Zは，出力にのみ用いる事ができます。出力後にキャリッジリターン，ラインフィードを行ないません。

＜例＞

```
   WRITE(1,10)
10 FORMAT('INPUT X ',Z)
   READ(0,*)X
```

WRITE文が実行された後，カーソル位置は

INPUT␣X␣■

となります。BASICのPRINT文に於けるセミコロンに対応します。

10.5.6 L変換 (Lw)

論理型変数はL変換を用いて入出力します。FORTⅣと異なる点はすべて空白ならば．FALSE．となります。一番左の英字がFでもLでもない場合はエラーとなります。

10.5.7 T変換 (Tw)

T変換はFORTⅣにはありません。T変換は左端からのけた位置を指定します。T変換はX変換の絶対位置版だと思えばよいと思います。X変換は直前の欄記述子の次のけたからけたを動かしますが，T変換は必ず左端からけたを動かします。

<例>

```
      READ(1, 56)I, LOT
56  FORMAT(I10, T20, I4)
```

データが，

␣␣␣␣1,2,4,0,0␣2,7,4␣␣␣␣4,7,2,8␣

となっていた場合にはLOTに4728が代入されます。

```
    J=1234
    WRITE(1, 34)J
34  FORMAT("##########", T5, I4)
```

を実行した場合

$$$$$1234$

と出力されます。

10.5.8 K変換 (Kw)

K変換もFORTⅣからの拡張です。変数の内容6バイトを12けたの16進数で表示します。wは一つの変数当り6バイト／変数×2けた／バイト＝12けたまで可能です。もし12より少ない場合はメモリの下位番地から表示されます。

<例>

```
    J=1
    WRITE(1, 99)J
```

99 FORMAT(K12)
を実行すると
100000000081
● と表示されます。この意味は，4．1を参照して下さい。

10.5.9 F変換 (Fw・d)

FORTⅣの規約に従っています。wとdは充分に注意して設定して下さい。w，dは符号なし整定数です。wは欄の長さ，dは小数点以下の桁数です。w，dはG変換以外は同じ意味です。

wけた読み込んだ後に，小数点を含まなかった場合には，右からdけた目に小数点を挿入します。もちろんwけたの中に小数点を含んでいる場合はそちらが優先します。これらの理由から，入力時にはFw．0を使用する事をすすめます。

10.5.10 E変換 (Ew・d)

E変換はFORTⅣの規約に従っています。注意すべき事を以下にあげます。

（1） 指数部は3けたまで許されています。

（2） 入力時，小数点を省略した場合は，右からdけた目に小数点を挿入します。

（3） 入力時には右づめになります。右に空白がある場合は0となります。

（4） Eという文字が入力欄にない場合はF交換と同じです。

入力欄	欄記述子	内部の値
12345E00	E8．2	0.12345×10^{3}
	E10．0	不可能です。
12．34E1␣	E8．2	0.1234×10^{12}

10.5.11 D変換 (Dw・d)

D変換はFORTⅣのコンパチビリティの者にのみ用意されています。実質的にはE変換と同じです。

10.5.12 G変換 (Gw・d)

G変換は入力にも出力にも使えます。FORTⅣの規約に従っています。

＜入力の場合＞

入力並びが整数型の場合にはIwと同じです。

入力並びが実数型の場合はEw．dと同じです。

＜出力の場合＞

出力並びが整数型の場合にはIwと同じです。

出力並びが実数型の場合は以下の様になります。rをその実数の絶対値とすれば，

```
.1≦r<1          F(w-5).d, 5X
 1≦r<10         F(w-5).(d-1), 5X
   ⋮

10**(d-2)≦ r < 10**(d-1)F(w-5).1, 5X
10**d-1   ≦ r < 10d          F(w-5).0, 5X
```

一般的には

F(w-5).(d-(指数)), 5X

となります。

10.5.13 欄記述子のくり返し

欄記述子はくり返し指定が可能です。FORTⅣの規約に従っています。いくつかの欄記述子や欄区切りの集まりをくり返して使用する場合にはその集まり全体をかっこでくくり，その直前にくり返しの回数を指定します。

一般に転送されるデータの数よりも欄記述子の数のほうが少ない場合には最後から2番目の右かっこに対応する欄記述子の集まりに戻ってそこからくりかえして使用されます。

くり返しの深さは2重以下のもののみ許されています。

＜例＞

```
      READ(2, 10)KNT, (Z(I), I=1, KNT)
10    FORMAT(I5/(F10.5))
```

Z(I)はすべてF10.5で読み込まれます。

```
10    FORMAT(2(I5, 3(I10, 2I2)))
```

は使用できません。

10.5.14 文字列の出力

書式つき出力文では文字列の出力が許されています。文字列は引用符でかこみます。引用符を出力する場合は2つの連続した引用符を使用します。＼(バックスラッシュ)は使用できません。

H変換を使用する事ができます。

0以外の16進数で表記されたアスキーコードをバックスラッシュで囲む事によって出力する事ができます。

&記号に引続いた英字によってコントロールコードを出力する事ができます。

```
WRITE(1,40)

40  FORMAT('TEST OF ASC \41\&J')
```

は,
TEST OF ASC␣A(改行)と出力されます。

10.6 自由形式の入出力

10.6.1 入力

自由形式の入力が用意されています。コンマかレコード区切り(キャリッジリターン)によって入力を区切ります。

文法： READ(論理装置番号, *)入力並び
　　ACCEPT␣入力並び

＜例＞

```
READ(0,*)A,B,C
```

コンソールから
12.3,24.6,36.9↵
と入力すると,それぞれがA, B, Cに代入されます。

すべての入力並びに対する入力が行なわれるまで,続々とレコードを読み進みます。BASICのInput文と同じです。

10.6.2 出力

自由形式の出力が用意されています。G変換で出力されます。

自由形式の入出力はファイルに対しても行う事ができます。

文法：ＷＲＩＴＥ（論理装置番号，＊）出力並び

ＴＹＰＥ　出力並び

10.7 書式なし入出力文

書式なし入出力文が用意されています。ＦＯＲＴⅣの規約に従っていますが，他のＦＯＲＴＲＡＮ（例えばマイクロソフト・ＦＯＲＴＲＡＮ）の書式なし入出力文で出力したファイルを読み込む事はできません。数値の内部形式が異なっているからです。このＦＯＲＴＲＡＮの書式なし出力文で出力したファイルから読み込む事が一般的です。この様な制限がありますが，書式なし入出力文は大量のデータを高速に入出力する為に使用します。単純変数又は配列要素一つ当り６バイトしか使いませんから，大量のデータを小さなファイルにしまう事ができます。

＜注＞

書式なし入力文は，ファイルの内容を調べずに変数に代入します。従ってその変数に，正しい表現の実数あるいは整数が代入されているかどうかは使用者の責任です。正しくない形式で代入された変数に対して算術演算を行うとエラーが起ります。

書式なしのファイルのＥｎｄ ｏｆ ｆｉｌｅ記号は６バイトの連続した０ＦＦＨです。

文法：ＷＲＩＴＥ（論理装置番号）出力並び

ＲＥＡＤ（論理装置番号）入力並び

＜例＞

```
WRITE(1)(I, I=1, 10)

WRITE(4, , ERR=66)ARRAY

READ(4)N, (ARRAY(I), I=1, N)
```

10.8 RFWIND文

リワインド文はFORTⅣの規約に従っています。ファイルポインタをファイルの最初に位置付けます。

文法： REWIND 論理装置番号

10.9 BACKSPACE文

BACKSPACE文はサポートされていません。コンパチビリティの為に，BACKSPACE文が実行されるとその旨をコンソールに出力します。

文法： BACKSPACE 論理装置番号

10.10 ENDFILE文

ENDFILE文はFORTⅣとのコンパチビリティの為に用意されています。論理装置番号のファイルを強制的にクローズします。現在のファイルポインタ以降にデータが存在したとしても，それはすべて失なわれます。現在のファイルポインタの位置にEOFを書き込みます。

文法： ENDFILE 論理装置番号

＜注＞ ENDFILE文はCLOSE文とは異なります。

ファイルに対して一度でも書式なし入出力を行なった場合には６バイトの０ＦＦＨを，一度も書式なし入出力を行なわなかった場合にはコントロールＺ（０１ＡＨ）を書き込みます。

ＥＮＤＦＩＬＥ文を実行後はその論理装置番号は解放されます。再びOPENするまでは使えません。

10.11 FORTRANの入出力とCP/Mのファイル

ＣＰ／Ｍのファイルと入出力とに関する注意を以下に個条書きします。充分に注意して使用して下さい。

（１）論理装置番号とＣＰ／Ｍ上のファイルとの結びつけは，ＯＰＥＮ文またはＬＯＰＥＮ文（11.12，11.10）を用いて行ないます。

（２）０から７までの８個の論理装置を使用する事ができます。０はコンソールからの入力，１はコンソールへの出力として，予約されていますから，ＯＰＥＮする必要はありません。それ以外の２～７を使用する時は必ずＯＰＥＮしなければなりません。

（３）ＣＯＮ：という装置名を使用して２～７の論理装置番号をコンソールに割り付けられますが，これらに対しては書式なし入力文，ＲＥＷＩＮＤ文，ＥＮＤＦＩＬＥ文，ＳＥＥＫ文は使用できません。

（４）既に存在するファイルに出力しようとする場合には，そのファイルをＤＥＬＥＴＥサブルーチンを使用して消去してから行なって下さい。ＯＰＥＮ文では入力か出力かの指定は行ないませんから，ファイルは消去されません。

（５）ＣＬＯＳＥ文は，コントロールＺなどのＥＯＦ記号を書き込みません。ＥＯＦ記号を書き込みたい場合にはＥＮＤＦＩＬＥ文を使用して下さい。（１０．１０を参照して下さい。）

（6） SEEK時にはバイト単位でポインタを移動します。書式付出力文で書かれたファイルには各レコードの区切りとしてCRとLF（0DH，0AH）が存在しますが，そのバイト数も使用者の責任で数えて下さい。

10.12 コンソール入出力に於る特殊文字

（1） CP／Mと同じくコントロールXは，行入力のキャンセルです。CR，LFに続いて感嘆符が表示されます。

（2） コンソール入力時，コントロールZを入力するとEnd of fileとなります。

（3） DELETE（07FH），または，コントロールHは一文字消去です。

11・システムが定義したサブルーチン及び関数

次に示すのはこのFORTRANが定義したサブプログラムと関数でCP／Mのファイルやメモリーの操作に役立ちます。

サブプログラム

BIT	CHAIN	CIN
CLOSE	CTEST	DELAY
DELETE	EXIT	LOAD
LOPEN	MOVE	OPEN
OUT	POKE	PUT
RENAME	RESET	SEEK
SETIO	SETUNT	

関数

CALL	CBTOF	CHAR
COMP	INP	PEEK
RAND		

以下に詳しい説明を示します。

ファイル操作を行うサブプログラムは，オプションの引数を用いてファイル操作時のエラーを検出することができます。このオプションの変数は各ルーチンの説明では{，error}と書いてあります。このパラメータを指定しない時，ファイル操作中にエラーがおこるとプログラムの実行は中断しCP／Mのシステムにもどります。またこのパラメータを指定しておくと，サブプログラムが終った時にエラーの有無，種類を知ることができます。

パラメータの値	意　味
0	エラーはありません。
1	指定したファイルがありません。
2	ディスクの容量が不足しました。
3	エンドオブファイル。

4	サブプログラムRENAMEで，変更しようとした新しい名のファイルが既にあります。
5，6	サブプログラムSEEKのエラー。
7	サブプログラムLOAD，CHAINで指定したファイル名が正しくありません。

11.1 BIT

```
CALL BIT(D, P, { 'S'      })
               { 'R'      }
               { 'F'      }
               { 'T', V   })
```

変数Dの左からP番目のビットに関するサブプログラムで，3番目のパラメータで機能を指定します。

'S' 指定したビットを 1 にします。
'R' 指定したビットを 0 にします。
'F' 指定したビットを 反転します。
'T' 指定したビットを調べて，その値（0か1）をVで指定した変数に入れます。

11.2 CHAIN

```
CALL CHAIN( 'program name' {, error } )
```

'program name' は，拡張子 '.obj' を持つこのFORTRANのオブジェクトで，実行は新しいプログラムにうつります。

現在オープンされているファイルのCLOSEはしないので，そのファイルを新しいプログラムでそのまま用いることができます。

オーバーレイではないので，メモリー上のデーターは保存しません。

ただし，無名共通ブロック(ブランクコモン)の内容は保存されるので，新しいプログラムにわたしたいデータは，ブランクコモンに入れておきます(4.7参照)。

指定したファイルが見つからない場合はエラーとなり，引数 error がない

時はプログラムを中断します。引数errorがある場合はエラーコードをセットしてもどります。

error	意　味
1	ファイルが見つかりません。
7	ファイルのフォーマットが正しくありません。

11.3 CIN

```
CALL CIN(V)
```

コンソールから1文字を読みこんで，Vの最も左のバイトに入れます。文字は7 bit ASCIIとして扱われ，MSBは0となっています。

＜例＞

```
C WAIT FOR A CR (0DH) FROM THE CONSOLE
  30    CALL CIN(CHARA)
        IF ( COMP(CHARA, #0D00, 1) .NE. 0 )
        GO TO 80
```

COMPで比較すべき文字を#を用いてあらわすにはこのように指定します。メモリー上では，2バイトごとに区切られ，

0D00　0000　0000

となっているためです。

11.4 CLOSE

```
CALL CLOSE(unit)
```

OPENまたはLOPENで開いた論理装置番号を持つファイルを閉じます。

11.5 CTEST

CALL CTEST(status)

コンソールに入力があるかどうかをしらべます。

status	意味
0	コンソールの入力はまだありません。
1	コンソールの入力がありました。

入力された文字はサブプログラムCINで入力できます。

CINや，READでコンソールの内容を読むと，次の入力があるまでstatusは0となります。

11.6 DELAY

CALL DELAY(wait time)

wait time×10ミリ秒だけプログラムを遅らせます。wait timeは1～65535で指定します。0は65536に解釈されるので注意が必要です。

CPUとしては2MHzの8080を仮定しているので，他のシステムでは実際に計って見ることが必要です。

11.7 DELETE

CALL DELETE('file'{,error})

'file' で指定したCP/Mファイルを消去します。

指定したファイルが存在しなくても，プログラムは続行します。

引数errorがある時には次のような意味をもちます。

error	意味
0	指定したファイルを消去しました。
1	指定したファイルは，はじめからありません。

消去したファイルは，回復できませんから注意が必要です。

11.8 EXIT

```
CALL  EXIT
```

プログラムを正常に終了させるために用います。STOPステートメントを用いる時と異なり，コンソールに何も表示せずそのままCP／Mのコマンドモードになります。

11.9 LOAD

```
CALL  LOAD ( 'file name' , type {, error } )
```

file name で指定したCP／Mファイルをメモリーにロードします。拡張子は，type で指定します。

type	拡張子
0	'.HEX'
0以外	'.OBJ'

主に関数CALLで呼ぶ機械語ルーチンのロードに用いますが，　このFORTRANは100H～4000Hにランタイム，4000H以後にオブジェクトがロードされるので，機械語ルーチンは，それを避けるようにロードする必要があります。

指定したファイルが存在しない，あるいは，ファイルの内容が正しくないときは，実行時エラーが起ります。

error を指定しないと，プログラムは中断され，CP／Mシステムに実行がもどります。error を指定した時には次のようにセットされます。

error	意　味
0	正常にロードしました。
1	ファイルがありません。
7	ファイルの内容が正しくありません。

11.10 LOPEN

CALL LOPEN(unit, 'file name' {, error })

ほとんどの動作は，サブプログラムOPENと同じです。異なるのは，出力時にフォートランの行送り制御が仮定されることです。各行のはじめの文字で行送りの制御が行なわれます。

最初の文字	代わりに出力されるコード	行なわれる動作
+	CR	前の行に重ねて書きます。
空白	CR, LF	前の行の次の行に書きます。
0	CR, LF, LF	前の行から1行あけて書きます。
−	CR, LF, LF, LF	前の行から2行あけて書きます。
1	CR, FF	次のページの先頭から書きます。

ここでCR, LF, FFはそれぞれ次のことを示します。

コード	16進	動作
CR	0DH	今の行の1カラムにもどります。
LF	0AH	カラム位置を変えずに次の行に進みます。
FF	0CH	次のページの先頭に送ります。

表示される装置はこの動作を実行できるものである必要があります。

11.11 MOVE

CALL MOVE (count, from, disp 1, to, disp 2)

メモリーの内容を他の場所に複写します。countは複写するバイト数を示します。fromとtoは複写のもとと行く先を示します。disp 1，disp 2は，fromとtoの使い方を示します。

disp	fromとtoの意味
−1	引数が示す値を用います。
0，1，2，……	引数があるアドレスにこの値を加えた値を用います。

＜例＞ CALL MOVE(2, A, -1, $CC00, -1)

Aの示すアドレスから2バイトを$CC00に複写します。

CALL MOVE(5, 'STRING', 1, $CC00, -1)

'STRING'の2番目以後5文字つまり'TRING'を, $CC00以後に複写します。

CALL MOVE(1024, $CC00, -1, A, 0)

CC00 以後 1024バイトを変数Aのあるアドレス以後に複写します。

11.12 OPEN

CALL OPEN.(unit, 'file name'{, error})

'file name' で指定したファイルをオープンして unit で指定する論理装置番号に対応させます。

指定したファイルがすでに存在している場合はそのファイルをオープンします。そのファイルがない時には, 2つの対応があります。error という引数をつけずにOPENを実行した場合は, 指定した名前のファイルを新たに, 作ります。error という引数を用いた場合は, OPENの実行後に error は次のような値を持ちます。

error	意　味
0	目的のファイルは存在していてオープンされました。
1	目的のファイルは存在せず何も行なわれません。

そこで, 次のようにしてあるファイルが存在するかを調べることができます。

```
CALL OPEN(3, 'INPUT', IERROR)
IF(IERROR.NE.0)TYPE 'NOT FOUND'
```

OPENでは, CP/Mファイル以外にも, CP/Mのシステムコンソール, プリンターのオープンができます。

'CON:' コンソールをオープンします。

'LST:' プリンターをオープンします。

＜例＞

```
CALL OPEN(2, 'DISKFILE')
CALL OPEN(3, 'C:INPUT.DAT')
CALL OPEN(5, 'CON:')
```

11.13 OUT

```
CALL OUT(P, V)
```

8080／8085／Z80 の出力ポートを直接アクセスします。ポートPにVを8ビットの値になおして出力します。

11.14 POKE

```
CALL POKE( location, value )
```

location で指定したメモリーに，value で指定した値を代入します。

＜例＞

```
CALL POKE(ADD, PEEK(ADD)+1)
```

は，ADDで示すメモリーの内容を1つ増します。

11.15 PUT

```
CALL PUT( value )
```

コンソールに value に対応するASCII文字を出力します。

＜例＞

```
CALL PUT(65)
```

65に対応するASCII文字'A'を出力します。

```
CALL PUT (CHAR('1', 0))
```

CHAR('1', 0)は49なので, '1' を出力します。

11.16 RENAME

CALL RENAME('old', 'new' {, error })

ファイルの名をつけかえます。もし'old'という名のファイルがなかったり'new'という名のファイルが既にある時にはエラーとなります。引数errorがない場合はプログラムCP/Mにもどります。引数 error を指定した時は次のような意味をもちます。

error	意　味
0	正常にRENAMEしました。
1	RENAMEすべきファイルがありません。
4	新しい名前をもつファイルが既にありません。

＜例＞

```
    DIMENSION   OLD( 2 ), NEW( 2 )
    READ( 0, 1 )OLD, NEW
1   FORMAT( 2A6/2A6 )
    CALL  RENAME  (OLD, NEW, ERROR)
```

11.17 RESET

CALL RESET

ディスケットを交換したことをシステムに教えるサブプログラムです。交換されるディスケットでオープンしていたファイルは交換に先だってCLOSEする必要があります。RESETの中でディスケットを交換するように指示を出します。

＜例＞

```
CALL CLOSE(3)
CALL CLOSE(4)
CALL RESET
```

11.18 SEEK

CALL SEEK(unit, position {, error})

unitで指定されたファイルの先頭からpositon番目のバイトで次のread／writeが行なわれるよう用意します。データーのない所にSEEKを試みると実行時エラーとなります。引数errorがない場合は実行を中断してCP／Mのシステムにもどります。引数errorを指定した場合は，errorを次のような値にしてもどります。

error	意　味
0	正常にＳＥＥＫしました。
5	その部分は存在しないセクターです。
6	その部分は存在しないイクステントです。

5の場合はファイルはCLOSEされないので，そのポジションに新しいデータを書き込むことができます。しかし，6の場合はファイルはCLOSEされます。CP／M 2．0以後のバージョンでは6のエラーは出ません。

11.19 SETIO

CALL SETIO(V)

コンソールのアクセスの方法を変更します。

V	意　味
0	CP／M BIOSのコンソール入出力ルーチンを用います。
2	CP／MのBDOSの1，2でコンソール入出力をします。コントロールPでプリンターをON／OFFできるのは，この場合のみです。

0，2以外　　CP／MのBDOSの6でコンソール入力をします。これが可能なのはCP／M 2.0以後のバージョンのみです。

すべてのCP／Mで用いることができるのは，V＝2のみです。

11.20 CALL

A＝CALL（address，argument）

address からはじまる，ユーザが定義する機械語ルーチンを呼びます。argument の内容を16ビットの数になおして，BC，DEレジスタペアに入れます。機械語ルーチンは得られた結果をHLレジスタペアに入れることで値をプログラムに返すことができます。機械語サブプログラムは普通に8080／Z80のRET命令を実行することで，もとのルーチンにもどります。

機械語ルーチンは，先に述べたLOADサブプログラムでメモリー上にロードします。

11.21 CBTOF

A＝CBTOF（from，disp　{，8－bit}）

Convert binary to floating の略で，16bit あるいは（,8－bit とつけ加えることで）8 bit の binary の数を，このFORTRANの内部表現に変換します。

変換するデータは次のように指定します。

disp が0以上の時は，変数fromが存在するアドレスに dispを加えたアドレスの内容を変換します。

disp が0未満の時は，変数fromが示すアドレスの内容を変換します。

＜例＞

V＝CBTOF（A，0，8－bit）

により変数Aの一番下のバイトの内容をVに代入します。

```
BIOS=CBTOF($0001,-1)-3
```
によりCP/MのBIOSの存在するアドレスを変数BIOSに代入します。

11.22 CHAR

A=CHAR(variable, disp)

変数variableの存在するアドレスにdispを加えたアドレスの文字が持つASCIIコードを得ます。

＜例＞
```
A='ABCDEF'
B=CHAR(A, 0)
C=CHAR(A, 5)
```
ではB, Cにはそれぞれ'A'と'F'に対応する65と70が代入されます。

11.23 COMP

A=COMP(str 1, str 2, leng)

2つの文字例str 1, str 2のはじめからlengの長さを比較します。比較は2つの文字例のはじめから1バイトづつとりだし，ASCIIコードで比較して行きます。結果は

−1	str 1	＜str 2
0	str 1	＝str 2
1	str 1	＞str 2

＜例＞
```
A=COMP('ABCDEF', 'ABCDEG', 5)
```
は'ABCDE'と'ABCDE'の比較でAは0となります。
```
A=COMP('ABCDEF', 'ABCDEG', 6)
```
は'ABCDEF'と'ABCDEG'を比べて，Aは−1となります。

11.24 INP

A＝INP（port）

8080／8085／Z80 の入力ポートを直接アクセスします。
port で指定した入力ポートの内容を変数Aに代入します。

11.25 PEEK

A＝PEEK（location）

location で示すメモリーの内容を変数Aに代入します。

＜例＞

BIOS＝PEEK（1）＋PEEK（2）＊256－3

によりBIOSの存在するアドレスをBIOSに代入します。これは

BIOS＝CBTOF（$0001，－1）－3

と同じ値を得ます。

11.26 RAND

A＝RAND（X）

0と1の間の擬似乱数を与える関数です。引数Xの値によって機能が異なります。

X＞0の場合は，Xを種（SEED）とする新しい系列の乱数のはじめの乱数を与えます。Xの値が同じならば得られる系列も同じものです。

X＝0の場合は，X＞0の値で定めた系列の次の乱数を与えます。

X＜0の場合は，X＞0の場合と同じように新しい系列の乱数を作りますが，Xの絶対値によらず同じ系列となります。

とにかく，X＝0としておけば，RAND関数は，次々と新たな乱数を発生して行きます。

Summary of System Function

Name	Function	Arg	result	argument
SIN	Sine(x)	1	real	real
COS	Cosine(x)	1	real	real
TAN	Tangent(x)	1	real	real
ATAN	Arctangent(x)	1	real	real
ATAN2	Arctangent(y/x)	2	real	real
ALOG	Log base e (x)	1	real	real
ALOG10	Log base 10 (x)	1	real	real
MOD	Remainder (x/y)	2	integer	integer
AMOD	Remainder (x/y)	2	real	real
SQRT	Square Root (x)	1	real	real
FLOAT	Make real (x)	1	real	integer
IFIX	Truncate (x)	1	integer	real
ABS	Absolute (x)	1	real	real
IABS	Absolute (x)	1	integer	integer
RAND	Random Number (x)	1	real	real 0.0<R<1.0
EXP	e**(x)	1	real	real
COMP	Compare strings	3	either	real
CALL	CALL assembly pgm	2	either	real
AMAX0	Maximum	<255	either	either
AMAX1	Maximum	<255	either	either
MAX0	Maximum	<255	either	either
MAX1	Maximum	<255	either	either
AMIN0	Minimum	<255	either	either
AMIN0	Minimum	<255	either	either
MIN0	Minimum	<255	either	either
MIN1	Minimum	<255	either	either
BIT	Bit handling	3/4	either	either
SIGN	Transfer of sign	2	real	real
ISIGN	Transfer of sign	2	integer	integer
DIM	Positive difference	2	real	real
IDIM	Positive difference	2	real	real
CHAR	character value	1	either	character
CALL	execute asm pgm	2	either	either
INP	input from a port	1	either	either
CBTOF	convert to real	2/3	real	both
COMP	compare strings	3	either	either
PEEK	examine mem loc.	1	either	either

● APPENDIX

1 実行時のエラーメッセージ

プログラムの実行中に続行不可能なエラーが起ると，次のようなメッセージをコンソールに出力してCP／Mシステムにもどります。

```
Rutime error: XXXX, called from loc. YYYYH
Pgm was executing line LLLL in routine NNNN
```

XXXX は以後に示すエラーメッセージです。

YYYY はエラーを起したルーチンを呼んだアドレスです。

このアドレスは，NNNNというルーチンのLLLL行目にあったことを示します。注意すべきことはLLLLはソースプログラムの行ではなく，コンパイル時のリストを見なければわからないということです。またコンパイル時にXオプションを用いないと行番号LLLLは表示することはできず????が表示されます。

Pgm was…… が2行以上表示された場合ははじめの行が正しいメッセージです。

ARG CNT	サブプログラムや関数を呼ぶがわと呼ばれるがわの引数の数が異なる。
ASN GOTO	割当てGOTOを実行しようとしたが，正しい文番号が変数にASSIGNされていなかった。
CALL POP	対応する呼び出しがないのにRET命令を実行した。主に機械語ルーチンの中でおこる。
CALL PSH	サブルーチン呼び出しのネストが深すぎる。 サブルーチンや関数を再帰的に用いた時などにおこる。

CHAIN FL	CHAINまたはLOADで指定したファイルが存在しなかった。
COM GOTO	計算型GOTOを実行しようとした所，指定した文番号がなかった。
CON BIN	システムコンソールへは，書式なし入出力はできないのにそれを試みた。
CONTRL／C	CTRL ENABLEの状態でコントロールCを受けつけたが，ERRSET文が実行されていなかった。
CONVERT	変数の内部表現を16ビットの整数に変換しようとしたところオーバーフロウをおこした。
DIV ZERO	0で割り算をしようとした。
DSK FULL	ディスク容量またはディレクトリーが不足した。
FILE OPR	ファイル操作時のエラー。
FORMAT	フォーマット入出力時のフォーマット文が正しくなかった。
ILL CHAR	READを実行しているときに正しくない字を読んだ。
ILL UNIT	READ, WRITE, OPEN, LOPEN, REWIND, CLOSE, SEEKにおいて，装置番号に 2，3，4，5，6，7以外を用いた。
INPT ERR	READで数を入力する時に正しくないデータがあった。例えば，整数を入力する所に 4．などと入力したり，

小数点を誤って2つ入力した時におこる。

INT RANG	計算結果を整数型変数に代入しようと思ったら，8桁の範囲を越えた。
I／O ERR	READ, WRITEの時にエラーが起った。そしてERR＝……　というエラー処理が指定されていなかった。
I／O LIST	フォーマット入出力の時に，指定した入出力リストにエラーがあった。
LINE LEN	入出力で1度に処理しようとしたレコードの長さが250バイトを越えた。この中には行の終りのCR（0DH）を含む。
LOG NEG	ALOG, LOG10で，負の数の対数の値を求めようとした。
OVERFLOW	計算結果が，実数の表現の範囲を越えた。
SEEK ERR	サブプログラムSEEKでエラーが起った。
SQRT NEG	SQRTで負の数の平方根の値を求めようとした。
UNIT CLO	サブプログラムCLOSEでクローズしようとしたファイルは，オープンされていなかった。
UNIT OPN	すでにオープンしている装置番号を再びLOPEN, OPENでオープンしようとした。装置番号0と1をOPEN, LOPENでオープンしようとした。

2 コンパイル時のエラーメッセージ

コンパイル時にエラーを発見すると，どのようなエラーかを2桁の16進コードで知らせます。さらにコンパイルのオプションにGを指定すると，エラーに対するメッセージも表示します。その中で＊ＦＡＴＡＬ＊は，特に重大なエラーです。

ØØ *FATAL* compiler error
００ （＊FATAL＊ コンパイラーのエラー。）

Ø1 Syntax error, 2 operators in a row
０１ （文法エラー，１行に２つの演算子がある。）

Ø2 unexpected continuation (column 6 not blank or Ø)
０２ （不正な継続文。）

Ø3 input buffer overflow (increase B= compiler option)
０３ （入力バッファ不足(コンパイルオプションＢ＝で増やすことができる)）

Ø4 invalid character for FORTRAN statement
０４ （ソースファイルに不正な文字がある。）

Ø5 unmatched parenthesis
０５ （'('と')'の数が一致していない。）

Ø6 statement label > 99999
０６ （文番号で99999より大きいものがある。）

Ø7 invalid character encountered in statement label
０７ （文番号に不正な字がある。）

Ø8 invalid HEX digit encountered in constant
０８ （16進数で不正な字がある。）

Ø9 expected constant or variable not found
０９ （変数または定数が不足している。）

ØA 8 bit overflow in constant
０Ａ （定数の値が８ビットを越えている。）

ØB unidentifable statement
０Ｂ （意味不明の文。）

ØC statement not implemented
0C （このFORTRANにはない機能を用いている。）

ØD quote missing
0D （quoteがない。）

ØE SUBROUTINE/FUNCTION/BLOCK DATA not first statement in routine
0E （SUBROUTINE／FUNCTION／BLOCKDATA文がルーチンの先頭にない。）

ØF columns 1-5 of continuation statement are not blank
0F （継続行なのに，1～5カラムが空白でない。）

1Ø cannot initialize BLANK COMMON
10 （無名コモンをBLOCKDATAで初期化しようとしている。）

11 RETURN is not valid in main program
11 （メインプログラムにRETURN文がある。）

12 syntax error on unit specification
12 （論理装置番号の指定がおかしい。）

13 missing comma after) in COMPUTED GO TO
13 （計算型GOTO文で）の後に，がない。）

14 missing variable in COMPUTED GO TO
14 （計算型GOTO文で変数がない。）

15 invalid variable in ASSIGNED/COMPUTED GO TO
15 （計算型GOTO文，割当てGOTO文で変数の値が不正。）

16 invalid LITERAL, no beginning quote
16 （不正な文字列。）

17 number of subscripts exceeds maximum of 7
17 （配列の次元が7を越えている。）

18 invalid SUBROUTINE or FUNCTION name
18 （不正なサブプログラム，関数名。）

19 subscript not POSITIVE INTEGER CONSTANT
19 （配列の添え字に使われた定数が正の整数でない。）

1A FUNCTION requires at least one argument
1A （関数に引数が1つもない。）

1B syntax error
1B （構文エラー。）

1C invalid argument in SUBROUTINE/FUNCTION call
1C （サブプログラム，関数を呼ぶための引数に不正がある。）

1D first character of variable not alphabetic
1D （変数名のはじめの字が，アルファベットでない。）

1E ASSIGNED/COMPUTED GOTO variable not integer
1E （計算型ＧＯＴＯ文，割当てＧＯＴＯ文の変数が整数ではない。）

1F label has already defined
1F （同じ値の文番号が２回使われている。）

20 specification of array must be integer
20 （配列の添字が整数でない。）

21 invalid variable name
21 （不正な変数名。）

22 invalid DIMENSION specification
22 （ＤＩＭＥＮＳＩＯＮ文で不正な指定が行なわれている。）

23 dimension specification is invalid
23 （配列宣言に不正がある。）

24 variable has already appeared in type statement
24 （型の宣言が２重に行なわれている。）

25 invalid subroutine name in CALL
25 （ＣＡＬＬ文で呼ばれるサブルーチン名の不正。）

26 SUBPROGRAM argument cannot be initialized
26 （サブルーチンで引数を初期化しようとしている。）

27 improperly nested DO loops
27 （ＤＯループの入れ子構造の不正。）

28 unit not integer constant or variable
28 （論理装置番号が整数型でない。）

29 Array size exceeds 32K
29 （配列の大きさが３２Ｋバイトを越えている。）

2A invalid use of unary operator
2A （単項演算子の不正使用。）

2B variable DIMENSION not valid in MAIN program
2Ｂ （メインプログラムで，配列の大きさが変数で与えられている。）

2C variable dimensioned array must be argument
2Ｃ （引数でない変数で配列の大きさの宣言をしている。）

2D DO/END/LOGICAL IF cannot follow LOGICAL IF
2Ｄ （論理ＩＦ文の後に，DO，END，論理ＩＦ文がある。）

2E undefined label
2Ｅ （未定義文番号の参照。）

2F unreferenced label
2Ｆ （参照されなかった行番号。）

3Ø FUNCTION or ARRAY missing left parenthesis
３０ （関数や配列で，）がない。）

31 invalid argument of FUNCTION or ARRAY
３１ （関数や配列の引数が不正。）

32 DIMENSION specification must precede first executable statement
３２ （ＤＩＭＥＮＳＩＯＮ文が実行文の後にある。）

33 unexpected character in expression
３３ （式の中に不正な字がある。）

34 unrecognized logical opcode
３４ （論理演算子の不正。）

35 argument count for FUNCTION or ARRAY wrong
３５ （関数または配列の引数，次元の不正。）

36 *COMPILER ERROR* poped off bottom of operand stack
３６ （＊コンパイラーのエラー＊　演算子スタックが空になった。）

37 expecting end of statement, not found
３７ （ＥＮＤ文がない。）

38 statement too complex; increase P and/or O table
３８ （複雑すぎる文（ＰまたはＯの量をコンパイルオプションで増やす））

39 invalid delimiter in ARITHMETIC IF
３９ （算術ＩＦ文での不正な区切り。）

3A invalid statement number in IF
3A （ＩＦ文内の文番号の不正。）

3B HEX constant > FFFF (HEX)
3B （16進数がＦＦＦＦＨを越えている。）

3C replacement not allowed within IF
3C （ＩＦ文内に代入文がある。）

3D multiple assignment statement not implemented
3D （多重代入文を用いている。）

3E subscripted-subscripts not allowed
3E （配列の添字に配列を用いている。）

3F subscript stack overflow; increase P= or O=
3F （配列の添字の計算のためのバファ不足（ＰまたはＯの量をコンパイル
オプションで増す）。）

40 missing left (in READ/WRITE
40 （READ, WRITE文の（がない。）

41 invalid unit specified
41 （不正な装置番号の指定。）

42 invalid FORMAT, END= or ERR= label
42 （READ, WRITE文のEND＝, ERR＝の行番号がない。）

43 invalid element in I/O list
43 （入出力リストのエラー。）

44 built-in function invalid in I/O list
44 （WRITE文の入出力リストに関数を参照している。）

45 cannot subscript a constant
45 （定数に添え字をつけている。）

46 variable not dimensioned
46 （宣言していない変数に添え字をつけている。）

47 invalid subscript
47 （不正な添え字。）

48 missing comma
48 （カンマ ' , 'がない。）

49 index in IMPLIED DO must be a variable
49 （DO文に変数が用いられていない。）

4A invalid starting value for IMPLIED DO
4A （DO文の初期値の不正。）

4B invalid ending value of IMPLIED DO
4B （DO文の終値の不正。）

4C invalid increment of IMPLIED DO
4C （DO文の増分の不正。）

4D illegal use of built-in function
4D （システムで定義した関数使用法の不正。）

4E variable cannot be dimensioned in this context
4E （配列名しか使えない所に添字をつけている。）

4F invalid or multiple END= or ERR=
4F （READ文，WRITE文のEND＝，ERR＝の重複や不正。）

50 invalid constant
50 （不正な定数。）

51 exponent overflow in constant
51 （定数の指数のオーバーフロウ。）

52 invalid exponent
52 （不正な指数。）

53 character after . invalid
53 （ピリオド'.'の後の文の不正。）

54 integer overflow
54 （整数のオーバーフロウ（大きすぎる）。）

55 integer underflow (too small)
55 （整数のアンダーフロウ（小さすぎる）。）

56 missing = in DO
56 （DO文に＝がない。）

57 string constant not allowed
57 （文字列定数を用いた。）

58 invalid variable in DATA list
58 （DATA並びの変数の不正。）

59 DATA symbol not used in program, line
59

5A invalid constant in DATA list
5A （DATA文中の定数の不正。）

5B error in DATA list specification
5B （DATA並びの指定の方法の不正。）

5C FUNCTION invalid in DATA list
5C （DATA並びに関数名がある。）

5D no filename specified on COPY
5D （COPY文のファイル名がない。）

5E runtime format not array name
5E （FORMATが入っている変数が配列でない。）

5F DUMP label invalid or more than 10 characters
5F （DUMP文のラベルの不正または名前が10字を越えている。）

60 more than 1 IMPLICIT is not allowed
60 （IMPLICIT文が2つ以上ある。）

61 IMPLICIT not first statement in MAIN, 2nd statement in SUBPROGRAM
61 （IMPLICIT文が，メインプログラムの先頭にない。あるいはサブプログラム，関数，ブロックデータの2番目にない。）

62 data type not REAL, INTEGER or LOGICAL
62 （REAL, INTEGER, LOGICAL以外のデータータイプで宣言している。）

63 illegal IMPLICIT specification
63 （不正なIMPLICIT文の使用。）

64 improper character sequence in IMPLICIT
64 （IMPLICIT文の文字の順序の不正。）

65 variable already DIMENSIONED
65 （配列の2重定義。）

66 Q option must be specified for ERRSET/ERRCLR
66 （Qオプションのない ERRSET文またはERRCLR文。）

67 Hex constant of zero (Ø) invalid in I/O stmnt
67 （入出力文中での16進定数0の使用。）

68 Argument cannot also be in COMMON
68 （引き数がコモンブロック中にも用いられている。）

69 Illegal COMMON block name
69 （コモンブロック名の不正。）

6A Variable already in COMMON
6A （コモンブロックにすでにあらわれた変数。）

6B Array specification must precede COMMON
6B （コモンブロックより配列の宣言が後にある。）

6C Executable statement invalid in BLOCK DATA
6C （ブロックデータ副プログラム中に実行文がある。）

6D Hex constant of 27H (') invalid in FORMAT
6D （フォーマット文中に＼を用いて27H（ ' ）を混ぜている。）

6E Invalid number following STOP or PAUSE
6E （STOP文, PAUSE文 で使う数の不正。）

6F invalid TRACE statement (operand not ON/OFF)
6F （TRACE ON文でもTRACE OFF文でもない。）

7Ø invalid IOSTAT= variable
70 （ERR=, END= で, 不正な変数がある。）

71 missing , in ENCODE/DECODE
71 （ENCODE, DECODE文中で, ' , 'がない。）

72 invalid label in ASSIGNED GOTO
72 （割当てGOTO文の文番号の不正。）

73 invalid variable in ASSIGNED GOTO
73 （割当てGOTO文の変数の不正。）

74 label not allowed on this statement
74 （文番号を用いてはいけない所に文番号がついている。）

75 multiple RETURN not valid in FUNCTION
75 （関数内での多重リターン。）

76 UNUSED
76 （未定義。）

77 no matching IF-THEN for ELSE or ENDIF
77 （IF－THEN－ELSE－ENDIFの入れ子構造の不正。）

78 invalid ELSE or ENDIF
78 （ELSE, ENDIFの不正。）

79 missing ENDIF
79 （ENDIFがない。）

7A initialization of non-COMMON variable
7A （BLOCKDATAの中でコモンに入っていない変数の初期化をしている。）

7B "DOUBLE PRECISION" not supportted, treated as "REAL"
7B （DOUBLE PRECISIONはREALとして扱う。）

7C UNUSED
7C （未定義）

7D UNUSED
7D （未定義）

7E UNUSED
7E （未定義）

7F UNUSED
7F （未定義）

80 *FATAL* no program to compile
80 （*FATAL* コンパイルするプログラムがない。）

81 *FATAL* missing $OPTIONS statement
81 （*FATAL* $OPTIONS文がない。）

82 *FATAL* missing = in $OPTIONS statement
82 （*FATAL* $OPTIONS文に＝がない。）

83 *FATAL* invalid digit in number in $OPTIONS
83 （*FATAL* $OPTIONS文に不正な数字がある。）

84 *FATAL* value exceeds 255 in $OPTIONS
84 （*FATAL* $OPTIONS文に出る数値が255を越えた。）

85 *FATAL* COMMON table overflow, increase C=
85 （*FATAL* コモンブロックが大きすぎる（C＝を増やすのがよい）。）

86 *FATAL* unknown option (letter before =)
86 （*FATAL* 不正なオプションが指定された。）

87 *FATAL* missing END statement
87 （*FATAL* END文がなかった。）

88 *FATAL* LABEL TABLE overflow, increase L=
88 （*FATAL* 文番号テーブルの不足（L＝を増やすのがよい）。）

89 *FATAL* SYMBOL TABLE overflow, increase S=
89 （*FATAL* 名前テーブルの不足（S＝を増やすのがよい）。）

8A *FATAL* ARRAY STACK overflow, increase A=
8A （*FATAL* 配列の添え字スタックの不足（A＝を増やすのがよい）。）

8B *FATAL* DO LOOP STACK overflow, increase D=
8B （*FATAL* DOループスタックの不足（D＝を増やすのがよい）。）

8C *FATAL* stack overflow (compiler error)
8C （*FATAL* スタック不足（コンパイラのエラー）。）

8D *FATAL* stack overflow (compiler error)
8D （*FATAL* スタック不足（コンパイラのエラー）。）

8E *FATAL* internal tables exceed user memory
8E （*FATAL* メモリーが不足。）

8F *FATAL* MEMORY ERROR
8F （*FATAL* メモリーのエラー。）

90 *FATAL* OPEN error on COPY file
90 （*FATAL* COPY文で指定したファイルがOPENできない。）

91 *FATAL* too many routines to compile (> 62)
91 （*FATAL* サブプログラム，関数の数が62を越えた。）

92 *FATAL* no more room to store DATA statements
92 （*FATAL* DATA文で定数の数が変数の数より多い。）

93 *FATAL* IF-THEN stack overflow, increase I=
93 （*FATAL* IF－THEN スタックの不足（I＝増やすのがよい）。）

94 *FATAL* Nested "COPY" statements not permitted
94 （*FATAL* COPY文で見ているＦＩＬＥに再びCOPY文がある。）

95 *FATAL* Disk write error (disk probably full)
95 （*FATAL* ディスクライトエラー（たぶん容量不足）。）

96 *FATAL* Cannot close file (disk probably full)
96 （*FATAL* ファイルのクローズができない（たぶん容量不足）。）

97 *FATAL* Input file not found
97 （*FATAL* 入力ファイルがない。）

98 *FATAL* Invalid drive specifier
98 （*FATAL* ドライブ指定の不正。）

99 *FATAL* No filename found on COPY statement
99 （*FATAL* COPY文にファイル名がない。）

9A *FATAL* File specified on COPY not found
9A （*FATAL* COPY文で指定したファイルがない。）

3 アセンブリ言語とのインターフェース

このFORTRANはコンパイルの際．ASMという一時的なファイルを作り，それをASSM. COMでアセンブルして．OBJファイルを得ています。フォートランのソースプログラム中で，第1カラムがアステリスク（'＊'）のものは，．ASMファイルにそのまま出力されます。これにより，ユーザーは，フォートランのソースプログラムに機械語を混ぜることができます。

この機能を用いる時の注意。このFORTRANはコンパイルの時必ず1行のさき読みを行うので，アセンブリ語のブロックがはじまる度に，直前にCONTINUE文を入れなければなりません。また，どのレジスタを変えてもかまいませんが，スタックポインタは保存する必要があります。

＜例＞

```
      CONTINUE
*  MVI  A, '*'
*  STA  STRING
```

4 その他の注意

1. 今まで文字列を引数とするいくつかのルーチンを示しましたが，この文字列のかわりに，文字列を含む変数や配列名を用いることができます。

変数は，代入文で文字列を含ませることができます。

```
A='STRING'
```

一度に代入できるのは6文字までで，文字列が6字に満たないと変数の下のバイトから0を埋めて行きます。

OPENルーチンのように引数がファイル名の場合は，

(1) 0ではない，(2) はじめの15バイト

をファイル名と考えます。

2. 変数，定数のかわりに16進定数が使用できます。16進定数は，$ではじまり0～9，A～Fを用いて0～FFFFの間でなくてはなりません。実際には整定数と扱われます。

＜例＞ A=$2000
B=-$FF

$を用いた16進数は，整定数として扱われるので，

```
A=$41
```

としても，Aは，'A'をあらわしません。このようなことをするためには，#を用います。#のあとにくる16進数は，そのまま変数に代入されます。ただし8080の約束で低位のバイトが高位のバイトより前になります。

16進数をバックスラッシュ(＼)が囲むことでリテラルの中に任意の値のバイトを含ませることができます。例えば

```
A='THIS \32\ IS AN EXAMPLE'
10 FORMAT ('JUMP\1B\Y!! TO 2,2')
```

'＼'のかわりに別の字を使いたい時は，CONFIG.COMの実行時に変更します。バックスラッシュは，コンソールによっては，¥で表示されます。

3. 一度にオープンできるファィルは，8ケ(0～7)です。ただし0と1はいつでもオープンされています。

4.1 NORTH STAR浮動小数点ボードの使い方

このFORTRAN では NORTH STAR の浮動小数点ボードが使用できます。CONFIG. COM でそのことを指定します。するとランタイム ルーチンは浮動小数点演算をそのボードで行ないます。実行時にはボードの有無をしらべ，もしない場合は，ソフトウェアで行うように切りかえます。注意すべきことは，NORTH STAR のボードを用いると，浮動小数点範囲が，10＊＊±63に制限されることです。

4.2 FORTRANとANSI 66 FORTRANの違い

改善点

1. 自由形式の入出力ができます。
2. IMPLICIT 文による型の宣言ができます。
3. READ／WRITE 文でEND＝，ERR＝による制御ができます。
4. COPY文で他のファイルからプログラムの結合ができます。
5. インラインでアセンブリ言語を含めることができます。
6. CP／Mのファイルアクセスができます。
7. 1バイトまでのファイルのランダムアクセスができます。
8. 実アドレスによるメモリー操作ができます。
9. プログラムで実行を遅らせることができます。（DELAY文）
10. 擬似乱数発生ができます。（RAND関数）
11. 実行時のエラートラップをプログラムで制御できます。
12. プログラムの CHAIN ができます。
13. プログラム実行中にオブジェクトプログラムが LOAD できます。
14. 13. でロードしたプログラム実行のための CALL 関数があります。
15. プログラムのトレース機能があります。
16. IF－THEN－ELSE 文があります。
17. プログラムによりコンソールからのプログラムの実行の中断をすることを許可，拒否することができます。

18. ENCODE, DECODE 文によりメモリー上でのデータ変換ができます。
19. 多重リターンができます。
20. 入出力文でK交換を用いることができます。

FORTRANにない機能

1. 倍精度実数は単精度実数として扱われます。
2. 複素数型がありません。
3. EQUIVALENCE 文がありません。
4. 次のようなDATA文は使えません。

   DATA I1, I2, I3／1, 2, 3／
5. 入出力におけるP変換（桁移動子）は使えません。
6. 文関数の使用ができません。
7. 関数，サブプログラム，共通ブロックの名として次のものを用いることができません。

   A, B, C, D, E, H, L, M, SP, PSW
8. EXTERNAL 文は使えません。
9. 添え字として配列の要素は使えません。
10. ハイパボリックなどいくつかの数学関数がありません。

SAMPLE PROGRAMS

On the following pages you will find listings of the sample programs that may have been included on your diskette.

```
C
C "CHAIN.FOR"
C
C THIS ROUTINE DEMONSTRATED THE 'CHAIN' FUNCTION, ALL IT
C DOES IS REQUEST THE NAME OF THE PROGRAM TO CHAIN TO
C AND THEN CHAIN.
C
      DIMENSION IF(3)
      TYPE 'FILE?'
C
C GET THE FILENAME TO CHAIN TO
C
      READ (0,1) IF
1     FORMAT (3A6)
C
C CHAIN TO IT
C
      CALL CHAIN (IF,IER)
C
C ONLY GETS HERE IF AN ERROR HAPPENS
C
      TYPE 'ERROR FROM CHAIN = ',IER
      CALL EXIT
      END
```

```
OPTIONS X
C
C "DUMP.FOR"
C
C THIS PROGRAM DEMONSTRATED THE USE OF THE DUMP STATEMENT.
C
C CALL 'X' FOR TRACEBACK PRINTOUT (JUST FOR SHOW)
C
        CALL X
        END
OPTIONS X
        SUBROUTINE X
C
C DEFINE THE DUMP STATEMENT TO BE USED IN CASE OF AN
C ERROR, WITH DUMP ID OF 'ROUTINE-X'
C

        DUMP /ROUTINE-X/ I,J,K
        I=1
        J=2
        K=I+J
C
C CREATE AN ERROR TO CAUSE DUMP STATEMENT TO BE ACTIVE
C
        Z=1/Ø
        END
```

```
OPTIONS X
C
C "GRAPH.FOR"
C
C GRAPH SINE FUNCTION FROM -PI TO PI IN INCREMENT OF .12
C
      DIMENSION LINE(70)
      INTEGER WHERE
C
C OPEN UNIT 6 TO WRITE TO CONSOLE
C
      CALL OPEN (6,'CON:')
C
C WRITE TITLE
C
      WRITE (6,2)
2     FORMAT (28X,'GRAPH OF SIN')
      TYPE
      TYPE
C
C SET PI AND -PI
C
      PI=3.1415926
      MPI=-PI
C
C MAIN LOOP
C
      DO 100 ANGLE=MPI,PI,.12
C
C FIGURE OUT WHICH ELEMENT IN ARRAY SHOULD BE SET TO *,
C SIN RETURNS -1 TO 1 WHICH IS CONVERTED TO -35 TO 35
C AND THEN OFFSET SO FINAL RANGE IS 1 TO 70
C
      WHERE=SIN(ANGLE)*35+35
C
C FIGURE OUT HOW MUCH TO BLANK IN THE OUTPUT ARRAY
C
      IBLANK=MAX0(35,WHERE)
C
C AND BLANK IT
C
      DO 15 I=1,IBLANK
15    LINE(I)=' '
C
C HMM... WHICH SIDE OF ZERO ARE WE ON?
C
```

```
        IF (WHERE .GT. 35)  THEN
C
C RIGHT SIDE
C
                            DO 20 I=36,WHERE
20                          LINE(I)='*'
                                ELSE
C
C LEFT SIDE
C
                            DO 30 I=WHERE,35
30                          LINE(I)='*'
                                ENDIF
C
C SET "ZERO"
C
        LINE(35)='+'
C
C AND THE SIN VALUE
C
        LINE(WHERE)='*'
C
C IF THIS VALUE IS < 35, SET SO WE OUTPUT TO ZERO LINE
C
        IF (WHERE .LE. 35)WHERE=35
C
C AND FINALLY OUTPUT THE LINE
C
        WRITE (6,21) (LINE(I),I=1,WHERE)
21      FORMAT (70A1)
100     CONTINUE
        CALL EXIT
        END
```

```
C
C "LOAD.FOR"
C
C THIS ROUTINE DEMONSTRATED THE USE OF THE 'LOAD' ROUTINE
C TO LOAD AN ASSEMBLY LANGUAGE FILE INTO MEMORY AND
C THEN CALL IT FORM FORTRAN
C
      INTEGER A
C
C FIND OUT WHICH ONE TO LOAD
C
      TYPE 'Enter Ø to "LOAD" LD.HEX'
      TYPE 'Enter 1 to "LOAD" LD.OBJ'
      ACCEPT 'Which one: ',LTYPE
C
      IF (LTYPE .EQ. Ø) THEN
                  TYPE '"LOAD"ing "LD.HEX"'
                   ELSE
                  TYPE '"LOAD"ing "LD.OBJ"'
                   ENDIF
C
C MUST LOAD "LD.HEX" OR "LD.OBJ" INTO MEMORY
C BEFORE WE CAN CALL IT
C
      CALL LOAD ('LD',LTYPE,IER)
C
      TYPE 'ERROR FOR LOAD=',IER
C
C CHECK THE RETURNED ERROR CODE FROM LOAD
C
      IF (IER .NE. Ø)STOP 'LOAD ERROR'
C
C "CALL" THE ROUTINE
C
      A=CALL ($8ØØØ,1)
C
C RESULT SHOULD BE 2
C
      TYPE 'THE RESULT OF THE ASSEMBLY ROUTINE IS: ',A
      CALL EXIT
      END
```

```
;
; "LD.ASM"
;
; THIS ROUTINE IS USED BY "LOAD.FOR", ALL IT DOES IS TO
; DOUBLE THE NUMBER SENT TO IT
; NUMBER IS PASSED IN DE FROM FORTRAN PROGRAM AND RESULT
; IS PASSED BACK IN HL TO FORTRAN PROGRAM
;
        ORG     8000H
        PUSH    D       ;NUMBER FROM FORTRAN PROGRAM
        POP     H       ;GET IT TO HL
        DAD     H       ;HL*2
        RET             ;RETURN IT

OPTIONS X,Q
C
C "RAND.FOR"
C
C THIS PROGRAM GENERATES A SEQUENCE OF RANDOM NUMBERS,
C DIVIDES THEM INTO 10 INTERVALS AND COUNTS HOW MANY
C RANDOM NUMBER FALL INTO EACH INTERVAL.  FINALLY IT
C PRINTS OUT THE COUNTS OF EACH INTERVAL.
C
        DIMENSION NUM(10)
        DATA NUM/10*0/
        INTEGER T,A,D,FLAG,TIME(6),DATE(6),START,END
99      DO 50 I=1,10
50      NUM(I)=0
        ACCEPT 'How many? ',K
        DO 1 I=1,K
        L=RAND(0)*10+1
1       NUM(L)=NUM(L)+1
        TYPE NUM
        GO TO 99
        END
```

```
OPTIONS X,Q
C
C "SEEK.FOR"
C
C THIS PROGRAM DEMONSTRATES RANDOM ACCESS I/O
C
C IT FIRST WRITES A FILE OF NUMBERS, THEN REQUESTS
C A RECORD, READ NUMBER THAT THE RECORD CONTAINS,
C ADDS 1 TO THE NUMBER READ AND WRITES IT BACK INTO
C THE SAME RECORD
C
        ERRSET 500,I
        CALL DELETE('TEST')
C
C OPEN THE TEST FILE
C
        CALL OPEN (2,'TEST')
C
C READ HOW MANY RECORDS TO CREATE
C
        ACCEPT 'HOW MANY RECORDS? ',K
C
C WRITE THE FILE
C
        DO 1 I=0,K
1       WRITE (2,2) I
2       FORMAT (I5)
        TYPE 'FILE WRITTEN'
        TYPE
        REWIND 2
        GO TO 10
        CALL OPEN(2,'TEST')
C
C REQUEST RECORD TO DISPLAY
C
10      ACCEPT 'WHICH RECORD? ',K
C
C POSITION THE FILE (EACH RECORD IS 7 CHARACTERS,
C 5 FOR NUMBER, 1 FOR CARRIAGE RETURN AND 1 FOR LINE FEED
C
        CALL SEEK (2,7*K,IER)
C
C CHECK THE ERROR CODE
C
        IF (IER .NE. 0) THEN
                        TYPE 'SEEK ERROR, CODE= ',IER
                        CALL CLOSE (2)
                        CALL DELETE ('TEST')
                        STOP
                        ENDIF
```

```
C
C READ THE CURRENT VALUE
C
        READ (2,2) I
        TYPE 'CURRENT VALUE OF RECORD ',K,' IS ',I
C
C POSITION BACK TO THE SAME RECORD
C
        CALL SEEK(2,7*K)
        I=I+1
C
C WRITE THE UPDATED VALUE
C
        WRITE (2,2) I
        GO TO 10
C
C TRAP ERROR
C
500     TYPE '*** ERROR TRAPPED ***'
        TYPE 'ERROR CODE = ',I
        CALL CLOSE (2)
        CALL DELETE ('TEST')
        STOP 'ERROR EXIT'
        END
```

```
C
C "SORT.FOR"
C
C THIS ROUTINE IS A DEMONSTRATION OF A SHELL SORT
C
        INTEGER T,A,D,FLAG,TIME(6),DATE(6),START,END
        DIMENSION A(2000)
        TYPE 'Shell sort'
        TYPE
C
C GET HOW MANY NUMBERS TO SORT
C
88      ACCEPT 'How many numbers (2-2000) ',NN
        IF (NN .LT. 2.OR.NN .GT. 2000)STOP
C
C GENERATE ARRAY OF NUMBERS TO SORT
C
        DO 10 I=1,NN
10      A(I)=(RAND(0)*NN)+1
        TYPE 'Starting sort'
        D=NN
        FLAG=0
C
100     D=IFIX((D+1)/2)
C
C TYPE OUT INTERMEDIATE STUFF
C
        TYPE 'D=',D
C
110     ND=NN-D
        DO 150 N=1,ND
        IF (A(N) .LE. A(N+D))GO TO 150
        NPD=N+D
        T=A(N)
        A(N)=A(NPD)
        A(NPD)=T
        FLAG=1
C
150     CONTINUE
        IF (FLAG .EQ. 1)THEN
                        FLAG=0
                        GO TO 110
                        ENDIF
        IF (D .GT. 1)GO TO 100
        TYPE 'All done'
        TYPE
C
C TYPE OUT SORTED ARRAY
C
        TYPE (A(I),I=1,NN)
        GO TO 88
        END
```

```
OPTIONS X,Q
C
C "TRACE.FOR"
C
C THIS ROUTINE DEMONSTRATES THE USE OF THE 'TRACE' AND
C 'ERROR' TRAPPING FUNCTIONS
C
        TYPE 'STARTING EXECUTION'
C
C SET ERROR TRAPPING: ON ERROR GO TO STATEMENT 500 WITH
C ERROR CODE IN VARIABLE I
C
        ERRSET 500,I
10      CONTINUE
C
C TURN TRACING OFF
C
        TRACE OFF
C
C GET AN INPUT # FROM THE USER
C
        ACCEPT '#: ',K
C
C IF <0, TERMINATE
C
        IF (K .LE. 0)GO TO 99
C
C IF INPUT # > 100, THEN TURN TRACING ON
C
        IF (K .GT. 100)TRACE ON
C
C AND OUTPUT THE NUMBERS, TO SEE EFFECT OF THE
C ERROR TRAPPING, HIT CONTROL-C
C
        DO 20 I=1,K
20      TYPE I
        GO TO 10
C
99      TYPE 'DONE'
        STOP
C
C ERROR TRAPPING HAPPENS HERE
C
500     TYPE 'ERROR TRAPPED, IER= ',I
        END
```

```
C
C THIS SAMPLE PROGRAM SHOWS HOW TO HANDLE THE
C CURSOR AND CLEAR SCREEN FUNCTIONS OF A CRT
C
        PAUSE 'CLEAR SCREEN'
        CALL SCREEN (1,0,0)
C
C WRITE THE LINE NUMBERS OUT FROM THE BOTTOM
C LINE TO THE TOP
C
        DO 10 I=23,0,-1
        CALL SCREEN (2,0,I)
        WRITE (1,2) I
2       FORMAT ('LINE ',I2,Z)
10      CONTINUE
C
C WAIT 5 SECONDS
C
        CALL DELAY (500)
C
C CLEAR THE SCREEN AGAIN
C
        CALL SCREEN(1,0,0)
        ACCEPT 'ENTER ANY NUMBER ',A
        I=AMOD(A,300)+1
        DO 99 J=1,I
99      Z=RAND(0)
C
C AGAIN CLEAR IT
C
        CALL SCREEN(1,0,0)
C
C NOW PUT RANDOM PLUS SIGNS ALL OVER THE SCREEN
C
        DO 40 I=1,400
C
C GET NEXT SCREEN POSITION AND GO THERE
C
        CALL SCREEN(2,IFIX(RAND(0)*80),IFIX(RAND(0)*23))
C
C PUT THE + ON THE SCREEN
C
40      CALL PUT (CHAR('+',0))
C
        CALL DELAY(500)
C
C OUTPUT A BELL
C
        CALL SCREEN(5,0,0)
C
C CLEAR IT NOW
C
        CALL SCREEN(1,0,0)
        CALL EXIT
        END
```

```
        SUBROUTINE SCREEN(FUNCT,X,Y)
C
C THIS IS A SAMPLE SCREEN DRIVER FOR AN ADM-3A TERMINAL
C
        INTEGER FUNCT,X,Y
        IF (FUNCT .LE. Ø.OR.FUNCT .GT. 5)RETURN
C
C FUNCTION:
C
C       1=CLEAR SCREEN
C       2=POSITION CURSOR
C       3=SET REVERSE VIDEO
C       4=SET NORMAL VIDEO
C       5=BELL
C
        GO TO (1ØØ,2ØØ,3ØØ,4ØØ,5ØØ),FUNCT
C
C CLEAR SCREEN
C
1ØØ     CALL PUT (12)
        RETURN
C
C SET CURSOR TO X,Y
C
2ØØ     CALL PUT(27)
        CALL PUT(1Ø2)
        CALL PUT(X+32)
        CALL PUT(Y+32)
        RETURN
C
C SET REVERSE VIDEO
C
3ØØ     RETURN
C
C SET NORMAL VIDEO
C
4ØØ     RETURN
C
C OUTPUT A BELL
C
5ØØ     CALL PUT(7)
        RETURN
        END
```

```
C
C THIS PROGRAM DEMOSTRATES THE USE OF THE SET FUNCTION OF
C THE "BIT" ROUTINE.  YOU ENTER THE BIT TO BE SET AND
C CAN SEE EXACTLY WHICH BIT IS CHANGED.
C
1       A=Ø
        ACCEPT 'BIT? ',B
        IF (B .GT. 47) THEN
                TYPE 'INVALID BIT NUMBER, ONLY Ø-47'
                GO TO 1
                ENDIF
        IF (B .LT. Ø)STOP
C
C SET THE BIT
C
        CALL BIT (A,B,'S')
C
C OUTPUT THE WORD IN HEX FORMAT
C
        WRITE (1,2) A
2       FORMAT (K12)
        GOTO 1
        END
```

SAMPLE PROGRAM COMPILATIONS AND EXECUTION

On the following pages you will find examples of the compilation and execution of several of the sample programs listed above. The following notes refer to the next few pages:

1) input is underlined.

2) CP/M output is printed in bold.

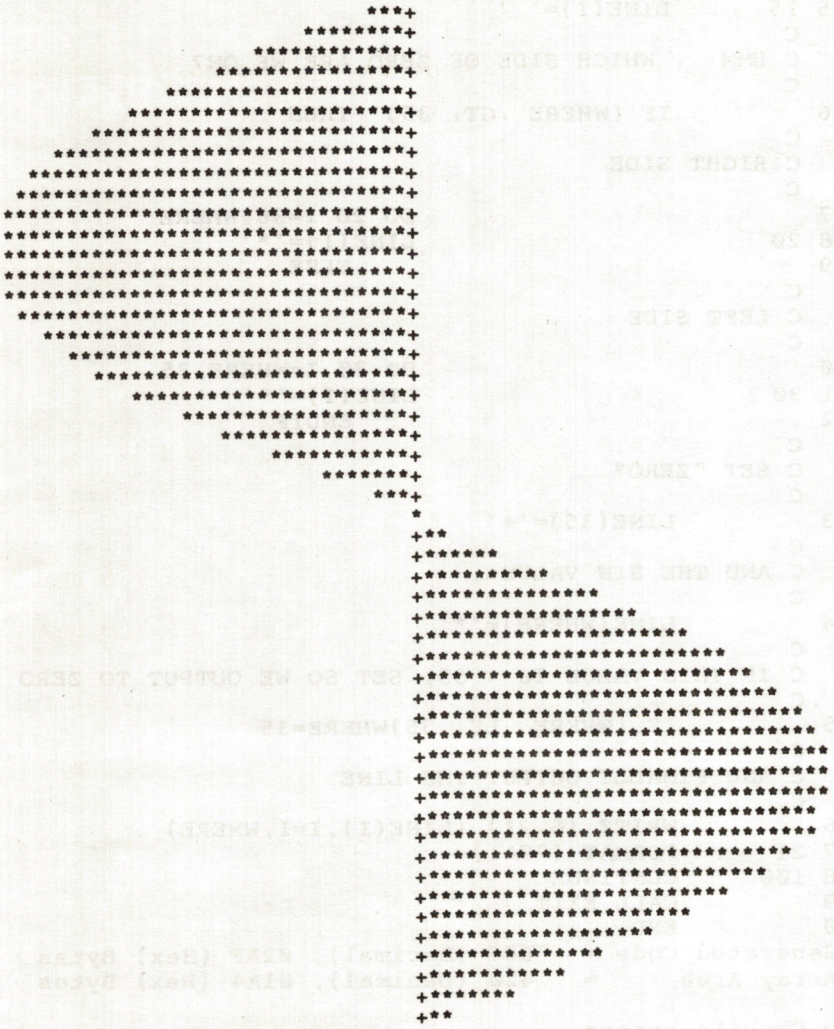
3) FORTRAN output (either compiler or execution) is neither underlined or bold.

4) notes are in { }.

```
B>FORT GRAPH.XBB    {compile with listing to console,
                     .ASM and .OBJ to drive B}

NEVADA FORTRAN 3.0 (MOD 0)
Copyright (C) 1979, 1980, 1981, 1982, 1983 Ian Kettleborough

*****  NEVADA  Fortran  3.0  (Mod  0)  **  Compiling  File:
GRAPH.FOR *****

0001 OPTIONS X
     C
     C  GRAPH SINE FUNCTION FROM -PI TO PI IN
     C  INCREMENT OF .12
     C
0002          DIMENSION LINE(70)
0003          INTEGER WHERE
     C
     C OPEN UNIT 6 TO WRITE TO CONSOLE
     C
0004          CALL OPEN (6,'CON:')
     C
     C WRITE TITLE
     C
0005          WRITE (6,2)
0006 2        FORMAT (28X,'GRAPH OF SIN')
0007          TYPE
0008          TYPE
     C
     C SET PI AND -PI
     C
0009          PI=3.1415926
0010          MPI=-PI
     C
     C MAIN LOOP
     C
0011          DO 100 ANGLE=MPI,PI,.12
     C
     C FIGURE OUT WHICH ELEMENT IN ARRAY SHOULD BE SET TO *,
     C SIN RETURNS -1 TO 1 WHICH IS CONVERTED TO -35 TO 35
     C AND THEN OFFSET SO FINAL RANGE IS 1 TO 70
     C
0012          WHERE=SIN(ANGLE)*35+35
     C
     C FIGURE OUT HOW MUCH TO BLANK IN THE OUTPUT ARRAY
     C
```

```
0013            IBLANK=MAX0(35,WHERE)
      C
      C AND BLANK IT
      C
0014            DO 15 I=1,IBLANK
0015 15         LINE(I)=' '
      C
      C HMM... WHICH SIDE OF ZERO ARE WE ON?
      C
0016            IF (WHERE .GT. 35)  THEN
      C
      C RIGHT SIDE
      C
0017                                DO 20 I=36,WHERE
0018 20                             LINE(I)='*'
0019                                    ELSE
      C
      C LEFT SIDE
      C
0020                                DO 30 I=WHERE,35
0021 30                             LINE(I)='*'
0022                                    ENDIF
      C
      C SET "ZERO"
      C
0023            LINE(35)='+'
      C
      C AND THE SIN VALUE
      C
0024            LINE(WHERE)='*'
      C
      C IF THIS VALUE IS < 35, SET SO WE OUTPUT TO ZERO LINE
      C
0025            IF (WHERE .LE. 35)WHERE=35
      C
      C AND FINALLY OUTPUT THE LINE
      C
0026            WRITE (6,21) (LINE(I),I=1,WHERE)
0027 21         FORMAT (70A1)
0028 100        CONTINUE
0029            CALL EXIT
0030            END
** Generated Code =   687 (Decimal), 02AF (Hex) Bytes
** Array Area      =   420 (Decimal), 01A4 (Hex) Bytes

  No Compile errors


 NO ASSEMBLY ERRORS.  175 LABELS WERE DEFINED.
```

```
B>FRUN GRAPH          {execute the program}
                      GRAPH OF SIN
```



```
B>FORT LOAD.X          {compile, listing to console, .ASM
                        and .OBJ to default drive}

NEVADA FORTRAN 3.Ø (MOD Ø)
Copyright (C) 1979, 1980, 1981, 1982, 1983 Ian Kettleborough

***** NEVADA Fortran 3.Ø (Mod Ø) ** Compiling File: LOAD.FOR
*****

0001 C
     C "LOAD.FOR"
     C
     C THIS  ROUTINE  DEMONSTRATED THE USE
     C OF THE 'LOAD' ROUTINE
     C TO LOAD AN ASSEMBLY LANGUAGE FILE INTO MEMORY AND
     C THEN CALL IT FORM FORTRAN
     C
0002          INTEGER A
     C
     C FIND OUT WHICH ONE TO LOAD
     C
0003          TYPE 'Enter Ø to "LOAD" LD.HEX'
0004          TYPE 'Enter 1 to "LOAD" LD.OBJ'
0005          ACCEPT 'Which one: ',LTYPE
     C
0006          IF (LTYPE .EQ. Ø) THEN
0007                           TYPE '"LOAD"ing "LD.HEX"'
0008                            ELSE
0009                           TYPE '"LOAD"ing "LD.OBJ"'
0010                            ENDIF
     C
     C MUST LOAD "LD.HEX" OR "LD.OBJ" INTO MEMORY
     C BEFORE WE CAN CALL IT
     C
0011          CALL LOAD ('LD',LTYPE,IER)
     C
0012          TYPE 'ERROR FOR LOAD=',IER
     C
     C CHECK THE RETURNED ERROR CODE FROM LOAD
     C
0013          IF (IER .NE. Ø)STOP 'LOAD ERROR'
     C
     C "CALL" THE ROUTINE
     C
0014          A=CALL ($8000,1)
     C
     C RESULT SHOULD BE 2
     C
```

```
0015            TYPE 'THE RESULT OF THE ASSEMBLY ROUTINE ',A
0016            CALL EXIT
0017            END
** Generated Code =    405 (Decimal), 0195 (Hex) Bytes

 NO ASSEMBLY ERRORS.  135 LABELS WERE DEFINED.



B>FRUN LOAD             {execute the program}

Enter 0 to "LOAD" LD.HEX
Enter 1 to "LOAD" LD.OBJ
Which one: 0
"LOAD"ing "LD.HEX"
ERROR FOR LOAD=             0
THE RESULT OF THE ASSEMBLY ROUTINE IS:          2
```

```
B>FORT TRACE.ZCC    {compile with no listing, .ASM and
                     .OBJ to drive C}


B>FRUN TRACE        {execute the program}
STARTING EXECUTION
#: 4
          1
          2
          3
          4
#: 140
Pgm  is executing line 0009 in routine MAIN
Pgm  is executing line 0010 in routine MAIN
          1
Pgm  is executing line 0010 in routine MAIN
Pgm  is executing line 0014 in routine MAIN
ERROR TRAPPED, IER=          23
STOP  END IN - MAIN
```

```
B>FORT SORT.BBB     {compile with listing, .ASM an
                    .OBJ files to drive B}

B>FRUN SORT         {execute the program}
Shell sort

How many numbers (2-2000) 100
Starting sort
D=          50
D=          25
D=          13
D=           7
D=           4
D=           2
D=           1
All done

           2           6           6           6           7           8
           8           9           9          12          12          14
          14          18          19          21          21          21
          21          22          25          27          28          28
          29          29          30          32          33          33
          34          34          37          38          39          40
          40          41          46          46          48          48
          48          49          50          51          52          54
          58          58          59          60          60          61
          62          62          64          64          65          65
          66          67          68          70          70          71
          73          74          75          76          76          77
          80          80          82          82          83          84
          85          86          88          89          89          91
          91          91          93          93          93          93
          94          96          96          96          97          98
          99          99         100         100
How many numbers (2-2000) 0
STOP
B>
```

●ASSEMBLER

1・概　要

ASSEMBLERは，8080のアセンブリ言語のアセンブラで，およそ次のような使い方をします。

まずED．COMなどを用いてソースファイルを作ります。ファイル名の拡張子は．ASMを用います。

次にASSM. COMでアセンブルします。ASSMは2パスではじめのパスでラベルの値を定め，次のパスでオブジェクトを作ります。構文上のエラーはこの段階で発見します。

こうしてできたオブジェクトは，RUNA. COMでメモリー上にロードして実行します。また，FORTRANのサブプログラムとして用いる時は，LOAD文でメモリーにロードします。

＜例＞

```
A>ED  TEST. ASM↵   ソースファイルのエディト
A>ASSM  TEST↵      アセンブル
A>RUNA  TEST↵      オブジェクトの実行
```

2・操作法

ハードウエアとしては32K以上のCP／Mが動く8080，8085，Z80によるマイクロコーンピュータシステムが必要です。

ディスケット上には，このFORTRANディスクにある，ASSM. COMとRUNA. COMが必要です。

使用するのは，これらをコピーしたディスクで，マスタディスクは大事にしまっておきます。

以下にアセンブリ言語のアセンブルと実行の基本的な方法を示します。

2.1 ソースファイルのアセンブル

次のようなコマンドでアセンブルをしてオブジェクトを作ります。

ASSM␣file〔. loe␣s $LPO〕↵

〔　〕の中は，オプションなので省略ができます。

fileはソースファイル名で，. ASMが仮定されます。先頭にA：～P：を付けてドライブ名も指定できます。

l　はリスティングファイル（. LST）を作るドライブ名で，A～Pでディスクファイル，Xでコンソールに出力します。不要の場合はZを指定します。

o　はオブジェクトファイル（. OBJ）を作るドライブ名で，A～Pでディスクファイルを指定します。デバッグでオブジェクトを作らない時はZとします。

e　はエラーファイル（. ERR）の指定で，lと同様に指定します。

s　はシンボルファイル（. SYM）の指定で，ドライブ名をA～Pで指定します。省略すると，シンボルファイルは作りません。

$以後には次のようなオプションをつけます。複数のオプションを用いる時は続けて入力してゆきます。

＋LまたはL　ソースファイルの1～4カラムは行番号として扱います。

－L　　ソースファイルに行番号はありません。

＋Ｌ，－Ｌの指定がない時には，ＡＳＳＭ自身が判断します。

＃　　ソースファイルに行番号があっても，ＡＳＳＭ自身が順に行番号をつけなおして．ＬＳＴファイルを作ります。

Ｐ　　．ＬＳＴファイルをページごとに清書し各ページに見出しをつけます。

０　　．ＬＳＴファイルはできるだけ短かく出力します。
．ＬＳＴファイルは１行７２桁で出力することを仮定して出力します。

２　　８０文字を仮定して．ＬＳＴファイルを作ります。

３　　１３２文字を仮定して．ＬＳＴファイルを作ります。

０，１，２，３のどれも指定しないと，２が仮定されます。

Ｓ　　．ＳＹＭファイルが，ＳＩＤ，ＺＳＩＤとコンパチブルになるように作ります。

＜例＞

```
A>ASSM␣TEST
A>ASSM␣B:TEST.AAX␣A$-L#P0
```

長いファイルのアセンブルの途中でエラーを発見した時などには，コントロールＣによりアセンブルを終了させることができます。

2.2 オブジェクトのロードと実行

こうしてできた．ＯＢＪファイルは次のようにして実行します。

```
RUNA␣file〔.ZLC〕↵
```

〔　〕内はオプションです。

Ｚ　　．ＯＢＪをロードする前にメモリーをクリアします。

Ｌ　　．ＯＢＪをロードするだけで実行しないで終ります。

Ｃ　　．ＯＢＪをロードし，実行せずに．ＣＯＭファイルを作ります。．ＣＯＭファイルは１００Ｈ からはじまるものに限るので，α－ＦＯＲＴＲＡＮ や α－ＣＯＢＯＬ の出力のオブジェク

トに対してこのコマンドを用いることは無意味です。

RUNAは実行がはじまると，まず自分自身を使用できるメモリの最も高い所に移し，次に．OBJファイルをロードしてオプションに従い実行をはじめます。

3・ソースプログラムの形式

ASSM．COMが理解する入力ファイルはED．COMなどで作る最も普通のテキストファイルです。1つの行には1つの文が入ります。1つの文は最大4つのフィールドに分割されます。それはラベル，オペレーション，オペランド，コメントです。また1行の長さは80文字までです。

3.1 ラベルフィールドとラベル

ラベルフィールドは1行の先頭（カラム1）からはじまります。ラベルはその行に対するシンボリックな名でプログラム内から参照が可能になります。ラベルは，アルファベットではじまり任意数の英数字を用いて作ります。ただしASSMはラベルのはじめの5文字しか識別しないので注意が必要です。

ラベルは，空白（ブランク文字）で区切られますが，他のアセンブラのようにコロン（：）で区切ることもできます。

8080のレジスタ名であるA，B，C，D，E，H，L，M，PSW，SPはラベルとしては使えません。

3.2 オペレーションフィールドとオペレーション

オペレーションフィールドはラベルに少なくとも1つの空白をおいたカラムからはじまります。ラベルがない行では，1カラム目を空白として2カラム目からはじまります。

オペレーションとして用いるのは，8080アセンブリ言語で定められたニーモニックと，ASSM．COMで使用可能な擬似オペレーションです。これらは付表や4章に述べられています。

3.3 オペランドフィールドとオペランド

オペランドフィールドはオペレーションに少なくとも1つの空白をおいたカラムからはじまります。

オペランドは各オペレーションの対象となるもので，1つの場合と2つの場

合があります。オペランドが2つある場合はカンマ ',' で区切ります。間に空白を入れていはけません。空白はフィールドの区切りなので，その空白以後を3.4で述べるコメントフィールドとして扱うからです。

オペランドとして使用するのは，1. レジスタ名，2. ラベル名，3. 定数，4. 式などです。

3.3.1 レジスタ名

レジスタ名としては，8080の A，B，C，D，E，H，L，M，SP，PSWがあります。BC，DE，HL といったレジスタペア名は，B，D，H で表わします。

3.3.2 ラベル名

ラベルは，同じプログラムの内部で定義された値を持っています。ASSMはその値をオペランドとして用います。

特に8 bitの値が必要な時は，ラベルの値が－256～255の間でないとエラーとなります。

特別なラベルの1つに　$　があります。$はその行現在の命令カウンタの内容でありその値をオペランドとして使うことができます。

3.3.3 定数

定数として使えるのは，2進数，8進数，10進数，16進数といった数定数とASCII文字列です。数定数は0～9，A～Fから作り基数をあらわす文字をつけてあらわします。

文字	基数	使える数字
B	2	0，1
Q，O	8	0～7
ナシ，D	10	0～9
H	16	0～9，A～F

＜注＞(1)　16進でA～Fではじまる数は，ラベルとの区別のためその前に0をつけます。

(2)　8進のOは0と混同するので，Qを用いるべきです。

ASCII文字列は，'で囲った1文字または2文字です。定数に'自身を含ませる場合は''と2つ並べて'一字とします。2バイトのASCII文字列は，8080の約束に従い，はじめの文字を低位バイト，2番目を高位バイトに代入します。

8ビットの数として扱う場合，定数の値として－256～255以外の数を用いるとエラーとなります。

3.3.4 式

ラベル，定数などを演算子で結合したものが式で，オペランドとして式を用いることもできます。演算子としては以下のものを用い，計算は2の補数で行います。

演算子

＋	和	または正数を表わす。
－	差	または負数を表わす。
＊	積	を表わす。
／	商	を表わす。

3.3.5 上位バイト、下位バイト

オペランドの先頭に'＜'をつけることで，以後で得た数の上位バイトの値を抽出できます。'＞'は下位バイトを抽出します。

＜と＞は単項演算子ではないので式の内部で使っても無効です。

3.4 コメントフィールドとコメント

コメント（注釈）フィールドは，オペランド（オペランドのないオペレーシ

ョンではオペレーション）フィールドから少なくとも１つの空白をあけたフィールドです。プログラムの注はここに書いておきます。

特に第１カラムが'；'または'＊'ではじまる行は，コメントとして扱われます。

3.5 行番号

ＡＳＳＭのソースファイルは，次のような場合，行番号付きファイルとして扱われます。それは１～４カラムに，０～９の数字による番号があり，５カラム目が空白である場合です。この場合，今まで述べた各フィールドはすべて５カラムずつ右へずれることになります。しかし行番付きのファイルはＥＤ．ＣＯＭとはコンパチブルではなく，ソースファイルが大きくなるのであまり勧めることはできません。

3.6 レジスタ名とその値

レジスタ名のＡ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ，Ｍ，ＰＳＷ，ＳＰは，実はそれぞれ　７，０，１，２，３，４，５，６，６，６という値を持っています。これらをその値をもつラベルとして扱うこと，逆にレジスタ名のかわりに０～７の値をもつラベルや定数を用いることができます。しかし混乱を防ぐためにもこれらは勧められません。

4・ 擬似オペレーション

擬似オペレーションはアセンブラを機能的に用いるためのもので次のようなものがあります。ここでいう＜式＞は，定数，ラベルまたは，それらを＋，－，＊，／，などの演算で結んだものを示します。

4.1 EQU

＜label＞ EQU ＜式＞

＜label＞に＜式＞の値を定義します。＜式＞の値は，その時点で確定している必要があります。また同一のラベルを２ケ所で定義することはできません。

4.2 ORG

〔＜label＞〕 ORG ＜式＞

＜式＞の値を命令カウンタに代入します。以後の命令は＜式＞のメモリ番地以後に作られて行きます。＜label＞はオプションです。
＜注＞ ORG 命令がない場合，オブジェクトはCP／Mで使用する 100 H から作られて行きます。

4.3 XEQ

XEQ ＜式＞

＜式＞で与えられたメモリ番地からプログラムの実行をはじめることを指定します。ORG と異なり，命令カウンタの値は変わりません。この指示は RUNA を用いる時に有効です。

4.4 DS, RES

〔＜label＞〕 DS ＜式＞

〔＜label＞〕 RES ＜式＞

＜式＞で指定したバイトだけ命令カウンタを進めて，その領域をプログラムで用いるために確保しておきます。

4.5 DB

〔＜label＞〕 DB ＜式＞〔，＜式＞，．．．〕

＜式＞の値の下位８ビットを（複数の＜式＞がある時には次々と）命令カウンタの示すメモリにストアして，メモリ内容の初期化を行います。必要な値は１バイトなので＜式＞の値は－２５６～２５５の間である必要があります。

4.6 DW

〔＜label＞〕 DW ＜式＞

＜式＞の値２バイトを命令カウンタの示すメモリにストアして，メモリの初期化をします。8080 の約束に従い下位バイト，上位バイトの順にストアして行きます。

4.7 DDB

〔＜label＞〕 DDB ＜式＞

＜式＞の値２バイトを命令カウンタの示すメモリにストアしてメモリの初期化をします。DWと異り上位バイト，下位バイトの順にストアして行きます。

4.8 ASC ASCZ

〔＜label＞〕 ASC #ASCII STRING#

〔＜label＞〕 ASCZ #ASCII STRING#

命令カウンタ以後にオペランドに指定した文字列をストアして行きます。#はストリングの区切り文字で，ストリングに含まれていない任意の文字が使えます。ASCZ では文字列の後に更に 00 をストアします。それを用いて文字列の区切りを示すことができます。

4.9 ASCF

ASCF 0　　または ASCF 1

DBまたはASC命令のリストの時，ASCF 0 としておくとはじめの4バイトの表示をしてあとは表示しません。ASCF 1 としておけば，すべてのバイトが表示されます。何も指定しないと ASCF 1 が仮定されます。

4.10 IF 〈式〉

条件アセンブル機能で，＜式＞の値が0でない時には IF と ENDF の間の文のアセンブルします。＜式＞が0の時には，IF と ENDF の間の文はアセンブルしません。

4.11 IFLS

条件アセンブルで，＜式＞が0の場合，IF と ENDF の間はアセンブルしません。そればかりではなく，リストにも出力されません。IFLS命令は，＜式＞が0でもリストにソースプログラムを出力させるオプションです。

4.12 COPY

COPY　＜ファイル名＞

他のディスクファイルを読んで，あたかも1つのファイルのアセンブルのようにする命令です。ファイル名の拡張子は．ASMを仮定しています。

注意すべきことは，COPYで読み込むファイルにCOPY命令があってはいけないことです。

ファイル名には：で区切ってドライブ名を付けることができます。

4.13 NLSTとLST

NLST命令以後のリストへの出力を停止します。LST命令でリストへの出力を再開します。

4.14 TITL ＜1行目＞"＜2行目＞

ASSMの実行時にPオプションをつけると，リストはページごとに行なわれますが，この命令で各ページの1，2行目に出てくるタイトルをセットすることができます。

4.15 PAGE

ASSMの実行時にPオプションをつけると，リストはページごとに行なわれます。PAGE命令は強制的に改ページをさせるために用います。

4.16 END

END命令は，ソースプログラムの終りを示しASSMはこれ以後のアセンブルを行いません。しかし，ファイルそのもののエンドオブファイルでもアセンブルを終えるので，END命令はあまり用いられません。

5・エラーとメッセージ

5.1 アセンブラ 起動時のエラー

アセンブラを実行する段階やオプションを解釈する段階でエラーを発見すると，次のようなメッセージをコンソールに出力してCP／Mシステムにもどります。

EXPECTED NAME

ASSMで指定したファイルがない。

ILLEGAL OPTION

ASSMで指定した$オプションがおかしい。

91 ファイルの大きさが大きくならない。→ディスクスペースの不足。
92 ファイルが書けない。→ディスクスペースの不足。
93 ファイルのオープンができない。
94 ディレクトリーが作れない。→ディスクスペースの不足。
95 ファイルが見つからない。
96 ファイルはすでにオープンされている。
97 書かれていないデータを読もうとしている。

5.2 構文上のエラー

実際にアセンブルしている間に発見したエラーはリストとエラーファイルの両方に出力されます。パス1実行中で発見したエラーは再びパス2でも出力されるので注意が必要です。

オペランドにエラーがある場合は，そのオペランドに対して0が与えられます。オペレーションにエラーがある場合は，命令のバイト数が不定なので，3バイトの0が出力されます。

エラーの種類は1文字のエラーコードで知ることができます。

エラーコード	意味
A	オペランドのエラー。多くは，数定数に正しくない字を使ったり，数字ではじまるラベルを使った時で，時には，ASC '''A''' とストリングのデリミタの誤りなどがあります。
D	ラベルの二重定義。ASSMではラベルの識別を最初の5文字で行っていることに気をつけなくてはなりません。
L	ラベルフィールドにエラーがあることを示します。例えば，1カラム目に数字があるものなどです。
M	EQU命令にラベルがない時に起ります。
O	オペレーションコードの誤まりです。
R	レジスタ名が必要なのに，それが正しく与えられていません。
S	ASSMには全く理解できない構文です。
U	ラベルが未定義です。EQU, ORG, DS, RES, IFの<式>は，必ず値が確定していることが必要です。
V	オペランドで使う数の範囲と<式>の値が正しくありません。例えば，8ビットのレジスタには－256～255の数しか入れることはできません。

JUMP

C3	JMP	Adr
C2	JNZ	
CA	JZ	
D2	JNC	
DA	JC	
E2	JPO	
EA	JPE	
F2	JP	
FA	JM	
E9	PCHL	

CALL

CD	CALL	Adr
C4	CNZ	
CC	CZ	
D4	CNC	
DC	CC	
E4	CPO	
EC	CPE	
F4	CP	
FC	CM	

RETURN

C9	RET
C0	RNZ
C8	RZ
D0	RNC
D8	RC
E0	RPO
E8	RPE
F0	RP
F8	RM

RESTART

C7	RST	0
CF	RST	1
D7	RST	2
DF	RST	3
E7	RST	4
EF	RST	5
F7	RST	6
FF	RST	7

ROTATE†

07	RLC
0F	RRC
17	RAL
1F	RAR

CONTROL

00	NOP
76	HLT
F3	DI
FB	EI

MOVE (cont)

58	MOV	E,B
59	MOV	E,C
5A	MOV	E,D
5B	MOV	E,E
5C	MOV	E,H
5D	MOV	E,L
5E	MOV	E,M
5F	MOV	E,A
60	MOV	H,B
61	MOV	H,C
62	MOV	H,D
63	MOV	H,E
64	MOV	H,H
65	MOV	H,L
66	MOV	H,M
67	MOV	H,A
68	MOV	L,B
69	MOV	L,C
6A	MOV	L,D
6B	MOV	L,E
6C	MOV	L,H
6D	MOV	L,L
6E	MOV	L,M
6F	MOV	L,A
70	MOV	M,B
71	MOV	M,C
72	MOV	M,D
73	MOV	M,E
74	MOV	M,H
75	MOV	M,L
77	MOV	M,A
78	MOV	A,B
79	MOV	A,C
7A	MOV	A,D
7B	MOV	A,E
7C	MOV	A,H
7D	MOV	A,L
7E	MOV	A,M
7F	MOV	A,A

ACCUMULATOR*

80	ADD	B
81	ADD	C
82	ADD	D
83	ADD	E
84	ADD	H
85	ADD	L
86	ADD	M
87	ADD	A
88	ADC	B
89	ADC	C
8A	ADC	D
8B	ADC	E
8C	ADC	H
8D	ADC	L
8E	ADC	M
8F	ADC	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
96	SUB	M
97	SUB	A
98	SBB	B
99	SBB	C
9A	SBB	D
9B	SBB	E
9C	SBB	H
9D	SBB	L
9E	SBB	M
9F	SBB	A
A0	ANA	B
A1	ANA	C
A2	ANA	D
A3	ANA	E
A4	ANA	H
A5	ANA	L
A6	ANA	M
A7	ANA	A
A8	XRA	B
A9	XRA	C
AA	XRA	D
AB	XRA	E
AC	XRA	H
AD	XRA	L
AE	XRA	M
AF	XRA	A
B0	ORA	B
B1	ORA	C
B2	ORA	D
B3	ORA	E
B4	ORA	H
B5	ORA	L
B6	ORA	M
B7	ORA	A
B8	CMP	B
B9	CMP	C
BA	CMP	D
BB	CMP	E
BC	CMP	H
BD	CMP	L
BE	CMP	M
BF	CMP	A

MOVE IMMEDIATE

06	MVI	B,
0E	MVI	C,
16	MVI	D,
1E	MVI	E,
26	MVI	H,
2E	MVI	L,
36	MVI	M,
3E	MVI	A,

D8

Acc IMMEDIATE*

C6	ADI	D8
CE	ACI	
D6	SUI	
DE	SBI	
E6	ANI	
EE	XRI	
F6	ORI	
FE	CPI	

LOAD IMMEDIATE

01	LXI	B,
11	LXI	D,
21	LXI	H,
31	LXI	SP,

D16

STACK OPS

C5	PUSH	B
D5	PUSH	D
E5	PUSH	H
F5	PUSH	PSW
C1	POP	B
D1	POP	D
E1	POP	H
F1	POP	PSW*
E3	XTHL	
F9	SPHL	

SPECIALS

EB	XCHG
27	DAA*
2F	CMA
37	STC†
3F	CMC†

INPUT/OUTPUT

D3	OUT	D8
DB	IN	D8

MOVE

40	MOV	B,B
41	MOV	B,C
42	MOV	B,D
43	MOV	B,E
44	MOV	B,H
45	MOV	B,L
46	MOV	B,M
47	MOV	B,A
48	MOV	C,B
49	MOV	C,C
4A	MOV	C,D
4B	MOV	C,E
4C	MOV	C,H
4D	MOV	C,L
4E	MOV	C,M
4F	MOV	C,A
50	MOV	D,B
51	MOV	D,C
52	MOV	D,D
53	MOV	D,E
54	MOV	D,H
55	MOV	D,L
56	MOV	D,M
57	MOV	D,A

INCREMENT**

04	INR	B
0C	INR	C
14	INR	D
1C	INR	E
24	INR	H
2C	INR	L
34	INR	M
3C	INR	A
03	INX	B
13	INX	D
23	INX	H
33	INX	SP

DECREMENT**

05	DCR	B
0D	DCR	C
15	DCR	D
1D	DCR	E
25	DCR	H
2D	DCR	L
35	DCR	M
3D	DCR	A
0B	DCX	B
1B	DCX	D
2B	DCX	H
3B	DCX	SP

DOUBLE ADD†

09	DAD	B
19	DAD	D
29	DAD	H
39	DAD	SP

LOAD/STORE

0A	LDAX	B
1A	LDAX	D
2A	LHLD	Adr
3A	LDA	Adr
02	STAX	B
12	STAX	D
22	SHLD	Adr
32	STA	Adr

PSEUDO INSTRUCTION

ORG	Adr
END	
EQU	D16
DS	D16
DB	D8 [,]
DW	D16 [,]

D8 constant, or logical arithmetic expression that evaluates to an 8 bit data quantity

all Flags (C,Z,S,P) affected

D16 - constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity

† = only CARRY affected

Adr = 16 bit address

** - all Flags except CARRY affected. (exception INX & DCX affect no Flags)

00	NOP		28	---		50	MOV	D,B	78	MOV	A,B	A0	ANA	B	C8	RZ		F0	RP	
01	LXI	B,D16	29	DAD	H	51	MOV	D,C	79	MOV	A,C	A1	ANA	C	C9	RET		F1	POP	PSW
02	STAX	B	2A	LHLD	Adr	52	MOV	D,D	7A	MOV	A,D	A2	ANA	D	CA	JZ		F2	JP	Adr
03	INX	B	2B	DCX	H	53	MOV	D,E	7B	MOV	A,E	A3	ANA	E	CB	---		F3	DI	
04	INR	B	2C	INR	L	54	MOV	D,H	7C	MOV	A,H	A4	ANA	H	CC	CZ	Adr	F4	CP	Adr
05	DCR	B	2D	DCR	L	55	MOV	D,L	7D	MOV	A,L	A5	ANA	L	CD	CALL	Adr	F5	PUSH	PSW
06	MVI	B,D8	2E	MVI	L,D8	56	MOV	D,M	7E	MOV	A,M	A6	ANA	M	CE	ACI	D8	F6	ORI	D8
07	RLC		2F	CMA		57	MOV	D,A	7F	MOV	A,A	A7	ANA	A	CF	RST	1	F7	RST	6
08	---		30	---		58	MOV	E,B	80	ADD	B	A8	XRA	B	D0	RNC		F8	RM	
09	DAD	B	31	LXI	SP,D16	59	MOV	E,C	81	ADD	C	A9	XRA	C	D1	POP	D	F9	SPHL	
0A	LDAX	B	32	STA	Adr	5A	MOV	E,D	82	ADD	D	AA	XRA	D	D2	JNC	Adr	FA	JM	Adr
0B	DCX	B	33	INX	SP	5B	MOV	E,E	83	ADD	E	AB	XRA	E	D3	OUT	D8	FB	EI	
0C	INR	C	34	INR	M	5C	MOV	E,H	84	ADD	H	AC	XRA	H	D4	CNC	Adr	FC	CM	Adr
0D	DCR	C	35	DCR	M	5D	MOV	E,L	85	ADD	L	AD	XRA	L	D5	PUSH	D	FD	---	
0E	MVI	C,D8	36	MVI	M,D8	5E	MOV	E,M	86	ADD	M	AE	XRA	M	D6	SUI	D8	FE	CPI	D8
0F	RRC		37	STC		5F	MOV	E,A	87	ADD	A	AF	XRA	A	D7	RST	2	FF	RST	7
10	---		38	---		60	MOV	H,B	88	ADC	B	B0	ORA	B	D8	RC				
11	LXI	D,D16	39	DAD	SP	61	MOV	H,C	89	ADC	C	B1	ORA	C	D9	---				
12	STAX	D	3A	LDA	Adr	62	MOV	H,D	8A	ADC	D	B2	ORA	D	DA	JC	Adr			
13	INX	D	3B	DCX	SP	63	MOV	H,E	8B	ADC	E	B3	ORA	E	DB	IN	D8			
14	INR	D	3C	INR	A	64	MOV	H,H	8C	ADC	H	B4	ORA	H	DC	CC	Adr			
15	DCR	D	3D	DCR	A	65	MOV	H,L	8D	ADC	L	B5	ORA	L	DD	---				
16	MVI	D,D8	3E	MVI	A,D8	66	MOV	H,M	8E	ADC	M	B6	ORA	M	DE	SBI	D8	**HEX-ASCII TABLE**		
17	RAL		3F	CMC		67	MOV	H,A	8F	ADC	A	B7	ORA	A	DF	RST	3			
18	---		40	MOV	B,B	68	MOV	L,B	90	SUB	B	B8	CMP	B	E0	RPO		Non-Printing		
19	DAD	D	41	MOV	B,C	69	MOV	L,C	91	SUB	C	B9	CMP	C	E1	POP	H			
1A	LDAX	D	42	MOV	B,D	6A	MOV	L,D	92	SUB	D	BA	CMP	D	E2	JPO	Adr	00	NULL	
1B	DCX	D	43	MOV	B,E	6B	MOV	L,E	93	SUB	E	BB	CMP	E	E3	XTHL		07	BELL	
1C	INR	E	44	MOV	B,H	6C	MOV	L,H	94	SUB	H	BC	CMP	H	E4	CPO	Adr	09	TAB	
1D	DCR	E	45	MOV	B,L	6D	MOV	L,L	95	SUB	L	BD	CMP	L	E5	PUSH	H	0A	LF	
1E	MVI	E,D8	46	MOV	B,M	6E	MOV	L,M	96	SUB	M	BE	CMP	M	E6	ANI	D8	0B	VT	
1F	RAR		47	MOV	B,A	6F	MOV	L,A	97	SUB	A	BF	CMP	A	E7	RST	4	0C	FORM	
20	---		48	MOV	C,B	70	MOV	M,B	98	SBB	B	C0	RNZ		E8	RPE		0D	CR	
21	LXI	H,D16	49	MOV	C,C	71	MOV	M,C	99	SBB	C	C1	POP	B	E9	PCHL		11	X-ON	
22	SHLD	Adr	4A	MOV	C,D	72	MOV	M,D	9A	SBB	D	C2	JNZ	Adr	EA	JPE	Adr	12	TAPE	
23	INX	H	4B	MOV	C,E	73	MOV	M,E	9B	SBB	E	C3	JMP	Adr	EB	XCHG		13	X-OFF	
24	INR	H	4C	MOV	C,H	74	MOV	M,H	9C	SBB	H	C4	CNZ	Adr	EC	CPE	Adr	14		
25	DCR	H	4D	MOV	C,L	75	MOV	M,L	9D	SBB	L	C5	PUSH	B	ED	---		1B	ESC	
26	MVI	H,D8	4E	MOV	C,M	76	HLT		9E	SBB	M	C6	ADI	D8	EE	XRI	D8	7D	ALT MODE	
27	DAA		4F	MOV	C,A	77	MOV	M,A	9F	SBB	A	C7	RST	0	EF	RST	5	7F	RUB OUT	

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity

Adr = 16 bit address

APPENDIX 2
TABLE OF ASCII CODES (Zero Parity)

Paper tape 123.4567P	Upper Octal	Octal	Decimal	Hex	Character		
.	0000	000	0	00	ctrl @	NUL	
•.	0004	001	1	01	ctrl A	SOH	Start of Heading
•.	0010	002	2	02	ctrl B	STX	Start of Text
•• .	0014	003	3	03	ctrl C	ETX	End of Text
•.	0020	004	4	04	ctrl D	EOT	End of Xmit
• •.	0024	005	5	05	ctrl E	ENQ	Enquiry
••.	0030	006	6	06	ctrl F	ACK	Acknowledge
•••.	0034	007	7	07	ctrl G	BEL	Audible Signal
. •.	0040	010	8	08	ctrl H	BS	Back Space
• .•	0044	011	9	09	ctrl I	HT	Horizontal Tab
• .•	0050	012	10	0A	ctrl J	LF	Line Feed
•• .•	0054	013	11	0B	ctrl K	VT	Vertical Tab
•. •	0060	014	12	0C	ctrl L	FF	Form Feed
• •. •	0064	015	13	0D	ctrl M	CR	Carriage Return
••. •	0070	016	14	0E	ctrl N	SO	Shift Out
•••. •	0074	017	15	0F	ctrl O	SI	Shift In
. •	0100	020	16	10	ctrl P	DLE	Data Line Escape
• . •	0104	021	17	11	ctrl Q	DC1	X On
• . •	0110	022	18	12	ctrl R	DC2	Aux On
•• . •	0114	023	19	13	ctrl S	DC3	X Off
•. •	0120	024	20	14	ctrl T	DC4	Aux Off
• •. •	0124	025	21	15	ctrl U	NAK	Negative Acknowledge
••. •	0130	026	22	16	ctrl V	SYN	Synchronous File
•••. •	0134	027	23	17	ctrl W	ETB	End of Xmit Block
.••	0140	030	24	18	ctrl X	CAN	Cancel
• .••	0144	031	25	19	ctrl Y	EM	End of Medium
• .••	0150	032	26	1A	ctrl Z	SUB	Substitute
•• .••	0154	033	27	1B	ctrl [	ESC	Escape
•. ••	0160	034	28	1C	ctrl \	FS	File Separator
• •. ••	0164	035	29	1D	ctrl]	GS	Group Separator
••. ••	0170	036	30	1E	ctrl ^	RS	Record Separator
•••. ••	0174	037	31	1F	ctrl _	US	Unit Separator
. •	0200	040	32	20	Space		
• . •	0204	041	33	21	!		
• . •	0210	042	34	22	"		
•• . •	0214	043	35	23	#		
•. •	0220	044	36	24	$		
• •. •	0224	045	37	25	%		
••. •	0230	046	38	26	&		
•••. •	0234	047	39	27	'		
.• •	0240	050	40	28	(		
• .• •	0244	051	41	29	)		
• .• •	0250	052	42	2A	*		
•• .• •	0254	053	43	2B	+		
•. • •	0260	054	44	2C	,		
• •. • •	0264	055	45	2D	-		
••. • •	0270	056	46	2E	.		
•••. • •	0274	057	47	2F	/		
. ••	0300	060	48	30	0		
• . ••	0304	061	49	31	1		
• . ••	0310	062	50	32	2		
•• . ••	0314	063	51	33	3		
•. ••	0320	064	52	34	4		
• •. ••	0324	065	53	35	5		
••. ••	0330	066	54	36	6		
•••. ••	0334	067	55	37	7		
.•••	0340	070	56	38	8		
• .•••	0344	071	57	39	9		
• .•••	0350	072	58	3A	:		
•• .•••	0354	073	59	3B	;		
•. •••	0360	074	60	3C	<		
• •. •••	0364	075	61	3D	=		
••. •••	0370	076	62	3E	>		
•••. •••	0374	077	63	3F	?		

TABLE OF ASCII CODES *(Cont'd)* (Zero Parity)

Paper tape 1 2 3 . 4 5 6 7 P	Upper Octal	Octal	Decimal	Hex	Character	
. . . . ●	0400	100	64	40	@	
● . . . . . . ●	0404	101	65	41	A	
. ● . . . . . ●	0410	102	66	42	B	
● ● . . . . . ●	0414	103	67	43	C	
. . ● . . . . ●	0420	104	68	44	D	
● . ● . . . . ●	0424	105	69	45	E	
. ● ● . . . . ●	0430	106	70	46	F	
● ● ● . . . . ●	0434	107	71	47	G	
. . . ● . . . ●	0440	110	72	48	H	
● . . ● . . . ●	0444	111	73	49	I	
. ● . ● . . . ●	0450	112	74	4A	J	
● ● . ● . . . ●	0454	113	75	4B	K	
. . ● ● . . . ●	0460	114	76	4C	L	
● . ● ● . . . ●	0464	115	77	4D	M	
. ● ● ● . . . ●	0470	116	78	4E	N	
● ● ● ● . . . ●	0474	117	79	4F	O	
. . . . ● . . ●	0500	120	80	50	P	
● . . . ● . . ●	0504	121	81	51	Q	
. ● . . ● . . ●	0510	122	82	52	R	
● ● . . ● . . ●	0514	123	83	53	S	
. . ● . ● . . ●	0520	124	84	54	T	
● . ● . ● . . ●	0524	125	85	55	U	
. ● ● . ● . . ●	0530	126	86	56	V	
● ● ● . ● . . ●	0534	127	87	57	W	
. . . ● ● . . ●	0540	130	88	58	X	
● . . ● ● . . ●	0544	131	89	59	Y	
. ● . ● ● . . ●	0550	132	90	5A	Z	
● ● . ● ● . . ●	0554	133	91	5B	[	
. . ● ● ● . . ●	0560	134	92	5C	\	
● . ● ● ● . . ●	0564	135	93	5D	]	
. ● ● ● ● . . ●	0570	136	94	5E	^	
● ● ● ● ● . . ●	0574	137	95	5F	_	
. . . . . ● ●	0600	140	96	60	`	
● . . . . ● ●	0604	141	97	61	a	
. ● . . . ● ●	0610	142	98	62	b	
● ● . . . ● ●	0614	143	99	63	c	
. . ● . . ● ●	0620	144	100	64	d	
● . ● . . ● ●	0624	145	101	65	e	
. ● ● . . ● ●	0630	146	102	66	f	
● ● ● . . ● ●	0634	147	103	67	g	
. . . ● . ● ●	0640	150	104	68	h	
● . . ● . ● ●	0644	151	105	69	i	
. ● . ● . ● ●	0650	152	106	6A	j	
● ● . ● . ● ●	0654	153	107	6B	k	
. . ● ● . ● ●	0660	154	108	6C	l	
● . ● ● . ● ●	0664	155	109	6D	m	
. ● ● ● . ● ●	0670	156	110	6E	n	
● ● ● ● . ● ●	0674	157	111	6F	o	
. . . . ● ● ●	0700	160	112	70	p	
● . . . ● ● ●	0704	161	113	71	q	
. ● . . ● ● ●	0710	162	114	72	r	
● ● . . ● ● ●	0714	163	115	73	s	
. . ● . ● ● ●	0720	164	116	74	t	
● . ● . ● ● ●	0724	165	117	75	u	
. ● ● . ● ● ●	0730	166	118	76	v	
● ● ● . ● ● ●	0734	167	119	77	w	
. . . ● ● ● ●	0740	170	120	78	x	
● . . ● ● ● ●	0744	171	121	79	y	
. ● . ● ● ● ●	0750	172	122	7A	z	
● ● . ● ● ● ●	0754	173	123	7B	{	
. . ● ● ● ● ●	0760	174	124	7C	|	
● . ● ● ● ● ●	0764	175	125	7D	}	
. ● ● ● ● ● ●	0770	176	126	7E	~	Prefix
● ● ● ● ● ● ●	0774	177	127	7F	DEL	Rubout

APPENDIX 3

ASSEMBLER LISTING

ADDRESS	ASSEMBLED CODE	ERROR FLAG	LINE NO.	LABEL	OPERATION	OPERAND	COMMENT
			0000	*			
			0001	*SEARCH TABLE FOR MATCH TO STRING			
			0002	*EACH TABLE ENTRY IS FOLLOWED BY A TWO-BYTE DISPATCH ADDRESS.			
			0003	*TABLE MUST HAVE AT LEAST ONE ENTRY AND IS TERMINATED BY A			
			0004	*ZERO BYTE.			
			0005	*ON ENTRY:	HL POINTS TO STRING		
			0006	*	DE POINTS TO TABLE		
			0007	*	C IS NUMBER OF CHARACTERS IN TABLE ENTRIES		
			0008	*ON RETURN:	ZERO FLAG SET IF NO MATCH, ELSE DE POINTS TO		
			0009	*	DISPATCH ADDRESS		
			0010	*			
0100	E5		0011	TSRCH	PUSH	H	SAVE STRING ADDRESS
0101	41		0012		MOV	B,C	INITIALIZE CHARACTER COUNT
0102	1A		0013	TS1	LDAX	D	COMPARE CHARACTERS
0103	BE		0014		CMP	M	
0104	C2 11 01		0015		JNZ	TS3	
0107	23		0016		INX	H	CHARACTERS MATCH, GO ON TO NEXT
0108	13		0017		INX	D	
0109	05		0018		DCR	B	
010A	C2 02 01		0019		JNZ	TS1	
010D	F6 01		0020		ORI	1	MATCHING ENTRY FOUND
010F	E1		0021	TS2	POP	H	
0110	C9		0022		RET		
0111	B7		0023	TS3	ORA	A	TEST FOR END OF TABLE
0112	CA 0F 01		0024		JZ	TS2	
0115	13		0025	TS4	INX	D	SKIP TO NEXT ENTRY
0116	05		0026		DCR	B	
0117	C2 15 01		0027		JNZ	TS4	
011A	13		0028		INX	D	
011B	13		0029		INX	D	
011C	E1		0030		POP	H	
011D	C3 00 01		0031		JMP	TSRCH	
			0032	*			
			0033	*EXAMPLE OF TSRCH USE:			
			0034	*			
			0035	*(ASSUME HL POINTS TO A FOUR-CHARACTER COMMAND STRING)			
0120	11 35 01		0036		LXI	D,CTABL	DE POINTS TO COMMAND TABLE
0123	0E 04		0037		MVI	C,4	TABLE ENTRIES ARE FOUR CHARACTERS LONG
0125	CD 00 01		0038		CALL	TSRCH	
0128	CA 00 00	U	0039		JZ	ERROR	COMMAND NOT IN TABLE
012B	EB		0040		XCHG	.	SET UP STACK FOR RETURN TO MAIN ROUTINE
012C	11 00 00	U	0041		LXI	D,COMMAND	
012F	D5		0042		PUSH	D	
0130	7E		0043		MOV	A,M	DISPATCH TO APPROPRIATE COMMAND ROUTINE
0131	23		0044		INX	H	
0132	66		0045		MOV	H,M	
0133	6F		0046		MOV	L,A	
0134	E9		0047		PCHL		
			0048	*			
			0049	*COMMAND TABLE			
			0050	*			
0135	43 4F 4D 31		0051	CTABL	ASC	'COM1'	FIRST ENTRY
0139	00 00	U	0052		DW	SUB1	ADDRESS OF SUB1
013B	43 4F 4D 32		0053		ASC	'COM2'	SECOND ENTRY
013F	00 00	U	0054		DW	SUB2	ADDRESS OF SUB2
0141	00		0055		DB	0	END OF TABLE MARK

SYMBOL TABLE LISTING

Label	Addr.	Label	Addr.	Label	Addr.	Label	Addr.
CTABL	0135	TS1	0102	TS2	010F	TS3	0111
TS4	0115	TSRCH	0100				

This is a sample program. The loader source code.

```
0001 ************************************************
0002 *
0003 *                  RUNA file-name[.ZCL]
0004 *
0005 *    An .OBJ file consists of one or more segments tha
0006 *    have the format:
0007 *       #BYTES          DESCRIPTION
0008 *         2          Number of code and data bytes in
0009 *                    segment
0010 *         2          Load address of code and data
0011 *                    belonging to the segment.
0012 *       Variable     Code and/or data.
0013 *
0014 *    The run time package will load each segment at th
0015 *    specified address until a starting address is
0016 *    encountered.  A starting address is represented a
0017 *    load address with a zero byte count.
0018 *
0019 * ************************************************
0020 *
0021 RELOC EQU 0     ;4200H FOR TRS-80 MOD 1
0022 BDOS EQU 5+RELOC  ;CP/M
0023 BLKSIZ EQU 128
0024 OFCB EQU 5CH+RELOC  ;IN CP/M
0025 OEX EQU OFCB+12
0026 OCR EQU OFCB+32
0027 OBUF EQU 80H+RELOC  ;IN CP/M
0028 *
0029 CSTART EQU $
0030  LXI SP,STK
0031  MVI C,0CH  ;RETURN VERSION #
0032  CALL BDOS
0033  MOV A,L
0034  ORA A
0035  JNZ VER2X
0036  LDA 4+RELOC  ;CPM 1.4 DEFAULT DRIVE
0037  CPI 5
0038  JC SETDF
0039  XRA A
0040 SETDF EQU $  ;11-30-81 FOR MP/M II
0041  STA ODRIVE  ;DEFAULT DRIVE
0042 *  GET OPTIONS FROM TYPE FIELD
0043  LXI H,5CH+8
0044  MVI C,4
0045 NEXT EQU $
0046  INX H
```

```
0047  DCR C
0048  JZ NOOPTIONS
0049  MOV A,M
0050  CPI ' '
0051  JZ NOOPTIONS
0052  CPI 'Z'
0053  JZ ZEROFIL
0054  CPI 'C'
0055  JZ COMFILE
0056  CPI 'L'
0057  JZ NOEXEC
0058 * ERROR ILLEGAL OPTION
0059  LXI H,MESGA
0060  CALL DISPLAY
0061  JMP 0+RELOC
0062 *  GET SIZE OF INSTRUCTION
0063 GETSZ LXI H,TBL-1
0064 AGAIN MOV A,C
0065  INX H
0066  MOV B,M
0067  ANA B
0068  JZ BYTE1
0069  INX H
0070  MOV B,M
0071  XRA B
0072  INX H
0073  JNZ AGAIN
0074  MOV A,M
0075  RET .  EXIT
0076 BYTE1 MVI A,1
0077  RET .  EXIT
0078 *
0079 REL EQU $  RELOCATION
0080  PUSH H
0081  PUSH D
0082  PUSH PSW
0083  INX H
0084  MOV E,M
0085  INX H
0086  MOV A,M
0087  ORA A  WE DON'T RELOCATE BELOW 100+RELOC
0088  JZ NOREL
0089  MOV D,A
0090  PUSH H
0091  LHLD BASE
0092  DAD D  ADDRESS IS NOW ADJUSTED
0093  XCHG
0094  POP H
0095  MOV M,D  PUT IT BACK
0096  DCX H
0097  MOV M,E
0098 NOREL EQU $
0099  POP PSW
0100  POP D
```

```
0101  POP H
0102  RET
0103 *
0104 VER2X EQU $
0105  MVI C,19H  ;GET CPM 2.X DEFAULT DRIVE
0106  CALL BDOS
0107  JMP SETDF
0108 *
0109 MESGA ASC 'ILLEGAL OPTION'
0110  DB 0DH,0AH
0111  ASC 'RUN D:FILE.ZCL<CR>'
0112  DB 0DH,0AH,0
0113 *
0114 TBL DB -1,11101001B,1
0115  DB -1,11001101B,3
0116  DB 11000111B,11000100B,3
0117  DB -1,11000011B,3
0118  DB 11000111B,11000010B,3
0119  DB 11000111B,11000111B,1
0120  DB -1,11001001B,1
0121  DB 11000111B,11000000B,1
0122  DB 11001111B,1,3
0123  DB 11100111B,00100010B,3
0124  DB 11110111B,11010011B,2
0125  DB 11000111B,6,2
0126  DB 11000111B,11000110B,2
0127  DB 0              END OF TABLE
0128 *
0129 BASE DW 0  BASE ADJ TO ADD TO ADDRESS TO BE RELOCATED
0130 START DW 0  STARTING ADDR OF RELOCATED CODE
0131 *
0132 ZEROFILL EQU $
0133  STA ZX
0134  JMP NEXT
0135 *
0136 COMFILE EQU $
0137  STA CX
0138  JMP NEXT
0139 *
0140 NOEXEC EQU $
0141  STA LX
0142  JMP NEXT
0143 *
0144 OSET EQU $
0145  LXI D,OBUF
0146  MVI C,26  ;SET DMA
0147  CALL BDOS
0148  LDA ODRIVE
0149  MVI D,0
0150  MOV E,A
0151  MVI C,14  ;SET DRIVE
0152  CALL BDOS
0153  LXI D,OFCB
0154  RET
```

```
0155
0156 NOOPTIONS EQU $
0157  LXI H,080H+3+RELOC
0158  MOV A,M
0159  CPI ':'  ;WAS DRIVE REQUESTED?
0160  JNZ DEFDRIVE  ;DEFAULT IS SET
0161  DCX H
0162  MOV A,M
0163  CPI 'A'
0164  JC DEFDRIVE
0165  SUI 'A'
0166  STA ODRIVE
0167 DEFDRIVE EQU $
0168  CALL SETFCB
0169  MVI M,'O'
0170  INX H
0171  MVI M,'B'
0172  INX H
0173  MVI M,'J'
0174  CALL OSET
0175  MVI C,15  ;OPEN
0176  CALL BDOS
0177  CPI -1
0178  JZ OERR  ;OPEN ERROR
0179  XRA A
0180  STA OCR
0181 *  RELOCATE CODE TO JUST BELOW CP/M
0182  LHLD 6+RELOC
0183  DCX H  HIGHEST ADDR
0184  LXI B,LAST-LOADFILE  SIZE OF CODE TO BE RELOCATED
0185  MOV A,L
0186  SUB C
0187  MOV L,A
0188  MOV A,H
0189  SBB B
0190  MOV H,A
0191 ; H&L= STARTING ADDRESS
0192  SHLD START
0193  PUSH H
0194  LXI D,LOADFILE
0195  MOV A,L
0196  SUB E
0197  MOV L,A
0198  MOV A,H
0199  SBB D
0200  MOV H,A
0201  SHLD BASE
0202  POP H
0203  LXI B,CONSTANTS-LOADFILE  SIZE OF INSTRUCTION MOVE
0204  XCHG
0205 NXTI EQU $
0206  PUSH H
0207  PUSH D
0208  PUSH B
```

```
0209  MOV C,M  GET OPCODE
0210  CALL GETSZ  GET SIZE OF INSTRUCTION
0211  POP B
0212  POP D
0213  POP H
0214  CPI 3
0215  JC SKPREL
0216  CALL REL  RELOCATE ADDR IN THIS 3 BYTE INST
0217 SKPREL EQU $
0218  PUSH B
0219  PUSH PSW
0220  MOV C,A  SIZE
0221 NXTM EQU $
0222  MOV A,M
0223  STAX D
0224  INX H
0225  INX D
0226  DCR C
0227  JNZ NXTM
0228  POP PSW
0229  POP B
0230 NXTD EQU $
0231  DCX B
0232  DCR A
0233  JNZ NXTD
0234  MOV A,C
0235  ORA B
0236  JNZ NXTI
0237 *  RELOCATE CONSTANTS
0238  LXI B,LAST-CONSTANTS  SIZE OF CONSTANTS
0239 NXTC EQU $
0240  MOV A,M
0241  STAX D
0242  INX H
0243  INX D
0244  DCX B
0245  MOV A,C
0246  ORA B
0247  JNZ NXTC
0248  LHLD START
0249  PCHL .  CODE HAS BEEN RELOCATED NOW GO TO IT
0250 *
0251 ********************************************************
0252 *  RUNA  A:FILE.OBJ<CR>
0253 *
0254 *  MOVE PARAMETERS AND CHECK
0255 ********************************************************
0256 *
0257 LOADFILE EQU $
0258  LXI SP,STK  SET STACK AFTER RELOCATION
0259  LDA ZX  ZERO FILL MEMORY?
0260  ORA A
0261  JZ SKPCLR
0262  LXI D,LOADFILE-1
```

```
0263  MVI H,1      STARTING ADDR  +RELOC
0264  MVI L,0
0265 CLEAR EQU $
0266  XRA A
0267  MOV M,A
0268  INX H
0269  MOV A,L
0270  SUB E
0271  MOV A,H
0272  SBB D
0273  JC CLEAR
0274 SKPCLR EQU $
0275  CALL ORD   ;GET 1ST RECORD OF .OBJ FILE
0276 OLOAD EQU $
0277  CALL GETOP
0278 MAO MOV A,M   ;MOVE 4 BYTES FROM BUF TO WORK
0279  STAX D
0280  INX H
0281  INX D
0282  DCR C
0283  CZ ORD
0284  DCR B
0285  JNZ MAO
0286 ; H&L = BUFFER  C=COUNT
0287  XCHG
0288  LHLD OWRK   ;SIZE OF NEXT READ
0289  MOV A,L
0290  ORA H
0291  JZ CLOSE
0292  SHLD OSIZE
0293  LHLD OWRK+2
0294  XCHG
0295 MAOA MOV A,M   ;MOVE FROM BUF TO OBJ ADDR
0296  STAX D
0297  INX H
0298  INX D
0299  DCR C
0300  CZ ORD
0301  PUSH H
0302  LHLD OSIZE
0303  DCX H
0304  SHLD OSIZE
0305  MOV A,L
0306  ORA H
0307  POP H
0308  JNZ MAOA
0309  CALL SAVOP
0310  JMP OLOAD
0311 *
0312 GETOP EQU $   ;GET O POINTERS
0313  LXI D,OWRK
0314  LHLD OCBA   ;BUF ADDR
0315  MVI B,4   ;LENGTH OF WRK
0316  LDA OCBC   ;BUF CNT
```

```
0317  MOV C,A
0318  RET
0319 *
0320 ORD EQU $
0321  PUSH B
0322  PUSH D  ;OPNT
0323  LXI D,OFCB
0324  MVI C,20  ;READ
0325  CALL BDOS
0326  POP D
0327  POP B
0328  ORA A
0329  JNZ RERR
0330  LXI H,OBUF
0331  MVI C,BLKSIZ
0332  RET
0333 *
0334 SAVOP EQU $
0335  SHLD OCBA  ;BUFF ADDR
0336  MOV A,C
0337  STA OCBC  ;BUF CNT
0338  LDA HIGH
0339  CMP D
0340  RNC
0341  MOV A,D
0342  STA HIGH
0343  RET
0344 *
0345 CLOSE EQU $
0346  LXI D,OFCB
0347  MVI C,16  ;CLOSE
0348  CALL BDOS
0349  LDA CX
0350  ORA A
0351  JNZ GENCOM
0352  LDA LX
0353  ORA A
0354  JNZ 0+RELOC  LOAD BUT DON'T EXECUTE
0355  LHLD OWRK+2  ;STARTING ADDRESS
0356  PCHL
0357 *
0358 SETFCB EQU $
0359  XRA A
0360  STA OFCB
0361  STA OCR
0362  LXI H,OEX
0363  MVI C,4
0364 EXLUP EQU $  ;10-2-81 ZERO CPM EXT AREA
0365  MOV M,A
0366  INX H
0367  DCR C
0368  JNZ EXLUP
0369  LXI H,OBUF
0370  SHLD OCBA
```

```
0371  MVI A,BLKSIZ
0372  STA OCBC
0373  LXI H,5CH+9+RELOC  ;CP/M FILE TYPE
0374  RET
0375 *
0376 CREATE EQU $
0377  LXI D,OFCB
0378  MVI C,22  CREATE
0379  CALL BDOS
0380  CPI -1
0381  RNZ
0382 OERR EQU $
0383  LXI H,MESGO  OPEN ERROR
0384  CALL DISPLAY
0385  JMP OXT1
0386 *
0387 GENCOM EQU $  GENERATE .COM FILE
0388  CALL SETFCB
0389  MVI M,'C'
0390  INX H
0391  MVI M,'O'
0392  INX H
0393  MVI M,'M'  .COM IN FCB
0394 *OPEN
0395  LXI D,OFCB
0396  MVI C,15  OPEN .COM FILE
0397  CALL BDOS
0398  CPI -1
0399  CZ CREATE
0400  XRA A
0401  STA OCR
0402 *WRITE
0403  LDA HIGH
0404  DCR A
0405  MOV H,A
0406  MVI L,0FFH
0407  SHLD SIZ  OF THIS WRITE
0408  MVI D,1  STARTING ADDRESS   +RELOC
0409  MVI E,0
0410  LXI H,OBUF  BUFFER ADDRESS
0411  MVI C,BLKSIZ  BUFFER SIZE
0412 NXTW EQU $
0413  LDAX D
0414  MOV M,A
0415  INX H
0416  INX D
0417  DCR C  BUFF COUNT
0418  CZ WRITE
0419  PUSH H
0420  LHLD SIZ
0421  DCX H
0422  SHLD SIZ
0423  MOV A,L
0424  ORA H
```

```
0425  POP H
0426  JNZ NXTW
0427  CALL WRITE  LAST BLOCK
0428 * CLOSE
0429  LXI D,OFCB
0430  MVI C,16  CLOSE
0431  CALL BDOS
0432  JMP 0+RELOC
0433 *
0434 WRITE EQU $
0435  PUSH D
0436  LXI D,OFCB
0437  MVI C,21  WRITE
0438  CALL BDOS
0439  POP D
0440  ORA A
0441  JNZ ERRW
0442  LXI H,OBUF
0443  MVI C,BLKSIZ
0444  RET
0445 *
0446 *++*************************************************
0447 *$$  DISPLAY A MESSAGE TO THE CONSOLE
0448 *     ENTRY H&L CONTAIN STARTING ADDRESS OF THE MESSAGE
0449 *           THE MESSAGE TEXT IS TERMINATED BY 0 HEX
0450 *           CALL DISPLAY
0451 *--*************************************************
0452 *
0453 DISPLAY EQU $
0454  MOV A,M
0455  ORA A
0456  RZ .  EXIT TO CALLING ROUTINE **
0457  MOV E,A
0458  MVI C,2
0459  PUSH H
0460  CALL BDOS  ;PUT THE CHAR TO THE CONSOLE
0461  POP H
0462  INX H
0463  JMP DISPLAY
0464 *
0465 ERRW EQU $
0466  LXI H,MESGW  WRITE ERROR
0467  CALL DISPLAY
0468  JMP OXT1
0469 *
0470 RERR EQU $
0471  LXI H,MESGR  READ ERROR
0472  CALL DISPLAY
0473 OXT1 EQU $
0474  LXI H,OFCB+1  ;FILE NAME
0475  CALL DISPLAY
0476  JMP 0+RELOC  RETURN TO CP/M
0477 *************************************************
0478 CONSTANTS EQU $
```

```
0479 HIGH DB 0  HIGHEST PAGE USED FOR .COM
0480 ZX DB 0  DEFAULT NO CLEAR ;Z=ZERO FILL BEFORE LOADING
0481 CX DB 0  DEFAULT NO .COM ;C=.COM FILE
0482 LX DB 0  DEFAULT EXECUTE ;L=LOAD BUT NO EXECUTION
0483 ODRIVE DB 0
0484 OWRK DB 0,0,0,0
0485 OCBA DW OBUF  ;CURRENT BUFFER ADDRESS
0486 OCBC DB BLKSIZ  ;CURRENT BUFFER COUNTER
0487 OSIZE DW 0  ;SIZE OF NEXT OBJ BLOCK
0488 SIZ DW 0  SIZE OF COM FILE CODE
0489 MESGO ASC 'OPEN ERROR '
0490  DB 0
0491 MESGR ASC 'READ ERROR '
0492  DB 0
0493 MESGW ASC 'WRITE ERROR '
0494  DB 0
0495  DS 30
0496 STK DB 'S'
0497 LAST DB 0
```

FORTRAN ユーザーズマニュアル

1986. 7. 1　初版発行

著　作　　Ellis Computing

訳　者　　片山伸彦

　　坪山　透

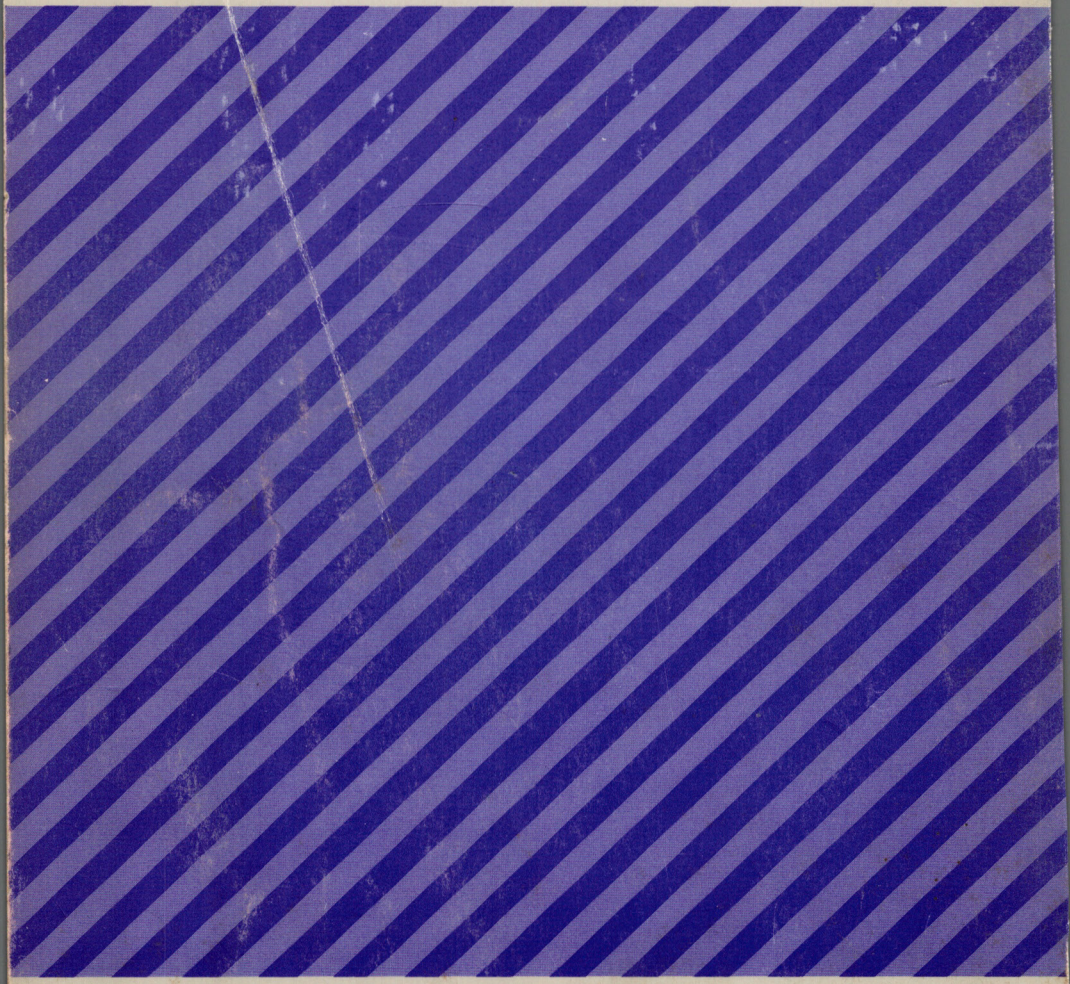
発行人　　田先政秀

(非売品)

LIFEBOAT 株式会社ライフボート

〒101 東京都千代田区神田錦町3−6　PHONE 03-293-4711

FAX 03-293-4710　TELEX 242-3296



発売元
シャープ株式会社
本　社　〒545 大阪市阿倍野区長池町22番22号　電話 (06)621-1221(大代表)
東京支社　〒162 東京都新宿区市谷八幡町8番地　電話 (03)260-1161(大代表)
テレビ事業部　ソフト開発部

供給元
株式会社ライフボート
〒101 東京都千代田区神田錦町3-6
電話 (03)293-4711