デバッガ機能の紹介

渡部 善隆* 平尾 耕二**

目次

1992年11月25日受理

* 九州大学大型計算機センター 研究開発部
** 九州大学大型計算機センター システム管理掛

1 はじめに

本記事はＭＳＰ上でのデバッガ機能の紹介と，コマンドに即しての簡単な使い方を説明したものです．コマンド，注意等に際して使用する例は，ごく簡単なものを用いています．本記事で紹介する内容を元にデバッガをさらに使いこなすためには，参考文献［1］のマニュアルを手元におかれるか，第６節で紹介されるデバッガサブコマンド ＨＥＬＰを用いて機能をご確認下さるようにお願いします．また，デバッガのサブコマンドは，今後追加サポートされる予定のものもあります（1992年11月25日現在）．追加の分は随時ニュース・広報でお知らせいたします．なお参考文献，マニュアル等の参照方法は稿末を参照願います．

デバッガは FORTRAN言語と Ｃ言語のデバッグが可能です．本稿では主に FORTRAN言語に即してデバッガの使用法を説明しますが，Ｃ言語に関してもほぼ同様の手順でデバッグが可能です．また，デバッガは FORTRAN言語と Ｃ言語の混在したプログラム，あるいは，異種プログラムと結合した FORTRAN,Ｃ言語のデバッグも可能です．異種言語の混在・結合したプログラムのデバッグに関しては［1],p12-p13 をご覧下さい．

さらにデバッガは対話形式だけでなく，バッチ処理形式でも起動させることができますが，本稿ではデバッガ機能の紹介をＴＳＳ形式の例を用いて行います．バッチ処理でのデバッガの起動方法の詳細は［1］の第６章をご覧下さい．

２　デバッガとは

2.1. デバッガの概要

デバッガ（正式名：OS IV デバッガ・V10/L20用）は FORTRAN言語や Ｃ言語で作成したロードモジュール（プログラムを翻訳，結合編集して得られた実行可能なプログラム）を実行させながら，デバッグ（プログラムの誤りを見つけて修正，確認を行う作業）を行うことを目的にしています．ロードモジュールの作成方法は第３節で説明します．また，小規模なプログラムならば，わざわざロードモジュールを陽に作成することなく，エディタ（PFD/EDIT，EDIT）内からデバッガを起動する方法もあります．この方法に関しては第４節で述べます．

デバッグ用に作成したロードモジュールは，デバッガのための情報を含んでいます．しかし，これは運用上普通のロードモジュールと全く同じものです．従って，運用上のトラブルに対してただちに原因の追求が可能です．

デバッガは，FORTRANコンパイラの移行に伴って導入された新しいデバッグ関係のシステムで，従来のデバッグシステム TESTFORT77 の後継にあたります．しかし，機能・コマンドの種類等は変更点がありますので注意が必要です．

2.2. デバッガの利点

例えば FORTRANプログラムを計算機センターのエディタ（ＰＦＤなど）を用いて編集・実行させるとします．短くて簡単なプログラムならば，実行までさほどの苦労はありませんが，サブルーチンや手続き関数をたくさん用いて，しかも大規模なプログラムを作成・実行する場合は，実行段階でエラーが出たり，あるいは，予想と違う結果が得られたりの，いわゆる“バグ”が生じます．バグはプログラムおよびプログラマにとって宿命的につきまとうもので，コンパイラ自身の欠陥から，プログラム論理のミス，有限の打切りに伴う丸め誤差の累積，あるいは元々の理論上の見落としなど，レベルの異なった要因が複合的に絡み合って起きる現象です．この，原因の所在が最初から特定することが難しい“バグ”に直面したとき，たいていのプログラム作成者は先ず，冷静にプログラムを点検し，ソースファイルを書き換え，アルゴリズムを書き留めたノートを確認し，他の人に相談し，文法書・マニュアルを読み，それでもダメなときは怒り出します．現に筆者の周囲でも，十分に議論を詰めてできあがったアルゴリズムを土台に，何日もかかって

プログラムを組み，いざ実行させてみると，予想した数値と実際計算機で得られた結果の余りの違いに腹を立て暴れ出す人もいました．さらに，極めて専門的な分野に立ち入ったプログラムになると，作った本人以外（たまには，何日もほったらかしにしていると本人自身も）理解できないプログラムも珍しくはなく，デバッグはますます困難な作業となります．

エラーメッセージが出力される場合は，ある程度そのメッセージをもとにバグの発生原因を絞りこむことが可能です．しかし，実行が正常に終了しながら得られる結果が納得できないプログラムをデバッグする場合などは，プログラムリストを丹念に追いかけてバグをつぶしていくしかありません．

その場合，プリンターに打ちだしたプログラムリストを赤鉛筆片手に眺めているだけでは処理の流れがなかなかつかめません．従って，例えば変数 Q の値が予想される値かどうかを確かめるには，プログラムに WRITE文を書き込んで再コンパイル，実行させて値を調べるという手法をよく用います．しかしこの方法は何度もソースプログラムを書き換え，なおかつ翻訳させる手間がかかります．

デバッガでは，あらかじめ実行可能なロードモジュールを作成することで，適当な行で Q の値を出力させたり，別の値で置き換えてみたり，などの機能を備えているので，ソースプログラムをいじる手間が省けるなどの利点を持っています．

なお，機能の解説は第６節で紹介します．

2.3. TESTFORT との相違点

現在のデバッガの役割は従来 TESTFORT77 が担ってきましたが，FORTRAN77 EX コンパイラの導入に伴いデバッガに変更されました．TESTFORT77 は FORTRANプログラムのみのデバッグに対応していましたが，デバッガは FORTRAN，Ｃ言語のデバッグが可能になりました．デバッガはエディタ内から直接起動させることもでき，陽にロードモジュールを作成する手間がかからなくて済みます（第４節）．デバッグ用に陽に作成したロードモジュールは，そのまま実運用することもできます．サブコマンドの機能の比較は，下の表をご覧下さい．

表を見てわかるように，現在（1992年11月25日）において未サポートのコマンドがいくつかあります．サブコマンドは機能追加され次第，随時センターニュース，広報等でお知らせします．

	TESTFORT77	デバッガ
サブコマンドの機能		
中断	○	○
表示・設定	○	○
データモニター	○	× 今後サポート予定
実行開始点変更	○	× 今後サポート予定
引数・添字検査	○	△ DEBUGオプションとの組合せで可
ソースプログラム表示	○	△ X LIST コマンドで代行可
最適化コードサポート	×	○

サブコマンドの機能／最適化コードサポートの違い

３　デバッグ用ロードモジュールの作成方法

小規模なプログラムならば，エディタと連携させ，デバッガを起動させてバグを見つける方法が簡単で便利です．その方法は第４節で説明します．ここでは比較的大きめのプログラムについて，デバッグ用のロードモジュールをユーザが作成する手順について説明します．ただし，例題で用いるプログラムは原稿の都合上，小規模なものを用いています．また，大規模，小規模の基準は主観的なものです．ユーザは少なくともＴＳＳ上で実行可能なプログラムに対し，デバッガを起動させ，対話形式でデバッグを行う限り，第４節の方法に従っていきなりエディタ内からデバッガを起動させることもできます．

ＳＴＥＰ１：プログラムの準備

デバッガを起動させ，デバッグを行うためには，実行可能なロードモジュールが作成可能でなければなりません．従って翻訳可能なプログラムを用意する必要があります．

【例１】

```
PROGRAM EX1       ! FORTRAN77 EX では "!" の後に注釈が書けます
  X=                ! 代入し忘れ
  WRITE(6,*) X
 END
```

上の簡単なプログラム EX1 をエディタで編集して，ファイルにセーブしました．しかし，このままではデバッガを起動させてデバッグすることはできません．理由は，まだ翻訳が完了していないためです．例えば，例１をコンパイルしてみる（RUN,FORT コマンドを実行するなど）と，次のコンパイルエラーが出ます．

```
 JWD1352I-S   00000200  INVALID RIGHT SIDE EXPRESSION
 END OF COMPILATION, HIGHEST SEVERITY CODE=12
```

メッセージは日本語に切り換え可能です．切り換え方法は READY 状態で PROFILE TLANG(J) と打ちます．英語表示に戻すときは PROFILE TLANG(E) と打ちます．
ＰＦＤで編集中ならば，コマンド欄に X PROFILE TLANG(J)，X PROFILE TLANG(E) と打つことで切り換えができます．
日本語のメッセージは次の様になります．

```
 JWD1352I-S   00000200  代入文又は文関数定義文の右辺式が正しくありません
  【翻訳完了】，完了コード＝12
```

例１のプログラムは簡単に修正できますが，長いプログラムになると翻訳がなかなか完了しないことがあります．デバッガは翻訳段階でのデバッグには使えません．従って，翻訳が完了コード＝00 で終了するまで，ユーザ側ではメッセージを頼りにデバッグを行って実行可能なプログラムを用意する必要があります．

【例２】

```
PROGRAM EX2
 X=1.0/0.0          ! ゼロ割りが起きている
 WRITE(6,*) X
END
```

例２のプログラムは，変数 X の代入時にゼロ割り（分母がゼロになる現象）を起こしています．このため，もちろん実行時にはエラーとなりますが，翻訳は完了コード＝00 で正常終了します．従って，このプログラムからは実行可能なロードモジュールが作成でき，デバッガでのデバッグが可能となります．

ＳＴＥＰ２：翻訳

以下，デバッグを行う FORTRANプログラムを デバッグ対象プログラム または 被デバッグプログラム と呼びます．被デバッグプログラムは，コンパイルオプション DBGINF を指定して翻訳します．DBGINF の指定があるとき，FORTRAN77 EX コンパイラは，通常のオブジェクト命令列には何らの変更も加えずに，デバッガのための情報を含むオブジェクトモジュール（計算機が直接実行可能な機械語の集合）を作成します．DBGINF を指定して作成したオブジェクトモジュールは,DBGINF を指定しないときと全く同じに実行することができます.ただし,後述するように，デバッグ用の情報を付加する分だけ保存されるデータセットの容量は増加します．

最適化の注意

デバッガは，コンパイルオプション OPT=NO，OPT=B 等と，DBGINF で指定して翻訳した最適化プログラムをデバッグすることができます．しかし，最適化されたプログラムのデバッグはユーザが注意すべき事項がかなりあります．従って，テスト時はコンパイラオプションを OPT=NO と指定してテスト，デバッグを行い，テストを完了した段階で OPT=B 等を指定して翻訳・実行することをお勧めします．なお，最適化に関しては，[2],[3](p360-p393) をご覧下さい．

また，コンパイルオプション DBGINF を指定したときはコンパイルオプション間で優先順位によって無効になるコンパイルオプションがありますので注意が必要です．例えば，DBGINF を指定すると，コンパイルオプション GO，NOOBJECT は無効になります．詳しくは [3] の p49-p89 をご覧下さい．

被デバッグプログラム以外の翻訳

普通，FORTRAN プログラムは，複数個のプログラム単位で構成されています．また，プログラム全体が一つのデータセットに保存されているのではなく，普段よく使うサブルーチンや関数などは，別のデータセットにロードモジュールとして保存しておき，実行時に使うこともよく行われています．

FORTRAN プログラムは，一つのロードモジュールで構成してもいいし，複数個のロードモジュールで構成してもかまいません．ただし，それぞれのロードモジュールの結合の仕方によって被デバッグプログラムが含まれてよい部分に以下の制限があります．なお，リンク・結合に関しては [3],[4] を参照下さい．

① 動的リンクによって結合する場合

被デバッグプログラムは，どのロードモジュールに含まれてもかまいません．

② 動的プログラム構造によって結合する場合

被デバッグプログラムは，どのロードモジュールに含まれてもかまいません．ただし，それぞれのロードモジュールは，デバッグ対象プログラムの実行において２度以上続けてローディングされてはいけません．

③ オーバーレイ構造によって結合する場合

被デバッグプログラムは，ルートセグメントにだけ含むことができます．
また，マルチタスクプログラムでは，デバッガが最初に制御を渡したタスクのみがデバッグの対象となります．

デバッガの機能を利用しない FORTRANプログラムは，コンパイラオプション DBGINF を指定して翻訳する必要はありません．ただし，編集結合を行ってロードモジュールを作成する場合は，コンパイラオプションに OBJECT を指定してオブジェクトモジュールを作成しなければなりません．

ＳＴＥＰ３：結合編集

デバッグ対象プログラムは，ＳＴＥＰ２ で作成したオブジェクトモジュールを，リンケージエディタのオプション TEST を指定して編集結合します．リンケージエディタは，TEST オプションがあるとき，オブジェクトモジュールに含まれているデバッガのための情報をロードモジュールに出力します．なお，結合編集の詳細は［4](p118-p128)，［5］を参照下さい．

ＳＴＥＰ４：デバッガ起動の準備

デバッグ対象プログラムが使用するデータセットのうち，実行前に結合が必要なものは，デバッグを開始する前にユーザが割り当てておかなければなりません．

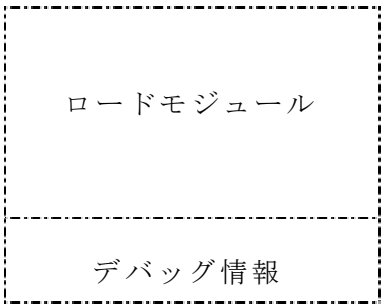
例えば，ALLOCATE コマンドを使って，装置参照番号６番に ’A79999A1.OUT.DATA’ を割り当てる例は次のようになります．

```
   READY
ALLOCATE FILE(FT06F001) DATASET('A79999A.OUT.DATA') OLD
```

なお点線を引いた部分は，ユーザが自分で打ち込むコマンドを意味します．
FORTRANプログラムが実行時に使用するデータセットについての詳細は［3］を参照下さい．

以上の手続きでデバッグ用の情報を持ったロードモジュールが作成されます．
しかし，デバッグ情報が付加されている分だけデータセットの容量は大きくなります．デバッグ用の情報を持ったロードモジュールの構成は次頁の表をご覧下さい．

以下，例３から例６にかけてロードモジュールを作成するための例題をいくつかあげます．
デバッグ用の情報を持ったロードモジュールを作成するためには，コンパイルオプションに DBGINF を，編集結合オプションに TEST を付けるのがポイントです．また，ＳＴＥＰ２で述べた最適化についても注意が必要です．



ロードモジュールの部分は通常のものが入っています．通常の実行時にはロードモジュールの部分のみが使用されます．

デバッガが読み込む情報が入っています

《デバッガ用情報の中身》

・プログラムの構成情報
・文とオブジェクト中のアドレスとの関係
・データの属性，アドレス情報　etc.

◆データセットの容量は通常の約1.5～2.0倍 必要になります．

デバッグ用情報を持ったロードモジュールの中身

【例３】

例２で使った例題をデータセット EX3.FORT に格納します．このプログラムは実行時エラーが出ますが，翻訳は可能ですので，デバッグ情報をもったロードモジュールが作成できます．

```
 READY
FORT EX3.FORT DBGINF OBJECT(TEST.OBJ(EX3))
 【FORTRAN77 EX 翻訳開始】
 【翻訳終了】，完了コード＝00
 READY
LINK TEST.OBJ(EX3) LOAD(TEST.LOAD(EX3)) FORTLIB TEST
 ** MEMBER NAME ** EX3        NOW REPLACED IN LIBRARY
 READY
```

ためしに実行してみます．

```
CALL TEST.LOAD(EX3)
JWE0013I-E 浮動小数点数の除算で除数が0です．PSWはOF8D0000810148B8です．
          :
   (エラーメッセージ)
          :
 TOTAL ERROR COUNT = 1
 READY
```

点線で表された下線部はユーザが自分で打ち込むところです．以下も同様です．ロードモジュール名（例では TEST.LOAD(EX3) ）はいくつか規則がありますが，だいたい勝手につけてかまいません．しかし例題の真似をして，どんどん ’TEST’ の名前のつくファイルを作っていくと，いつ

か何のファイルだったか判らなくなることがよくありますので，注意が必要です．また，コマンドは省略して打ち込むこともできます（OBJECT は OBJ でよいとか，.FORT .OBJ は書かなくてもよいなど）．また FORTLIB は結合の仕方によっては書いてはいけない場合もあります．プログラムの用途は様々ですので，ロードモジュールの作り方も色々あります．[3]-[5] を参照しながら，自分の用途にあったロードモジュールを作成します．

【例４】

例２で使ったプログラムを EX4.FORT としてデータセットに格納します．（従って EX3.FORT と全く同じです）．今度は ＴＳＳ処理ではなく，バッチ処理によってデバッガ用の情報を持ったロードモジュール TEST.LOAD(EX4) を作成します．

```
//A79999A1 JOB CLASS=A
// EXEC FORT,STEP=CL,
//  OPTION=DBGINF,PARM.LKED=TEST,OPT=NO
//FORT.SYSIN DD DSN=A79999A.EX4.FORT
//LKED.SYSLMOD DD DSN=A79999A.TEST.LOAD(EX4),DISP=SHR
//
```

上のジョブ制御文をエディタ（PFD/EDIT,EDIT,..）で作成して，SUBMITコマンド(cf.[6]) でジョブを依頼すれば TEST.LOAD(EX4) という名前のデバッガ用情報を持つロードモジュールが作成されます．なお，このジョブを依頼したユーザの ID は A79999A としています．

例４のジョブ制御文において，コンパイルオプションでは DBGINF を，また，編集結合オプションでは TEST を指定しています．また，ＳＴＥＰ２で述べた最適化の注意に従って OPT=NO を指定しています．実際，例題のプログラムは最適化する所など何もありませんが，大ざっぱに言って DO ループがたくさんあるプログラムなどは最適化についての注意が必要です(cf.[3] 第7章)．最適化レベルをユーザ自身で指定しないときは，OPT=B で最適化が行われます(cf.[2])．

例４は，プログラムが４行しかないのに，いかにも大げさなロードモジュールの作り方ですが，大規模なプログラムや，データをさかんに読み書きするプログラムを扱う場合はたいへん便利です．なお，例題に現れる各語意のより詳しい説明（CLASS=A, STEP=CL 等）や，そもそもバッチジョブとは何か，などは [6] を参照して下さい．

【例５】

サブルーチン，関数，メインプログラムが３つのデータセットに分割して保存あります．データセット名は，それぞれ SUB.FORT, FUNC.FORT, EX5.FORT とします．

《ファイルの中身》

```
SOBROUTINE SUB(X)
X=1.0+2.0
RETURN
END
```

SUB.FORT

```
REAL FUNCTION FUNC(X)
FUNC=X**2
RETURN
END
```

FUNC.FORT

```
PROGRAM EX5
 CALL SUB(X)
 Y=FUNC(X)
 WRITE(6,*) Y
END
```

EX5.FORT

被デバッグプログラム EX5.FORT にデバッガ用の情報を加味してロードモジュールを作成します．　例５では，各ソースプログラムからオブジェクトモジュールを作り，LINK コマンドでひとつのデバッグ情報付きのロードモジュールを作成します．

```
  READY
FORT SUB.FORT OBJECT(SUB.OBJ) NOOPTIMIZE
FORT FUNC.FORT OBJECT(FUNC.OBJ)
FORT EX5.FORT OBJECT(TEST.OBJ(EX5)) NOOPTIMIZE DBGINF
 【FORTRAN77 EX 翻訳開始】
 【翻訳終了】，完了コード＝00
```

オブジェクトファイルのメンバ名を指定しないと TEMPNAME という名前がつきます(cf.[4]).
次に結合編集を行います．

```
  READY
LINK (SUB,FUNC,TEST.OBJ(EX5)) FORTLIB TEST LOAD(TEST.LOAD(EX5))
  ** MEMBER NAME **  EX5 NOW ADDED TO LIBRARY.
  READY
```

SUB,FUNC は，正確には SUB.OBJ(TEMPNAME)，FUNC.OBJ(TEMPNAME) と書きますが，この場合省略が可能です（cf.[4]).

これでデバッガ用の情報を持った TEST.LOAD(EX5) が作成されました.ためしに実行させてみると普通のロードモジュールと変わりない結果が得られます．デバッガ用の情報を持ったロードモジュールは実運用上，普段のロードモジュールと同等に機能します．

```
CALL TEST(EX5)        .LOAD は省略可能
 9.0000000
 READY
```

【例６】

データセットは例５と全く同じものを用います．便宜上メインプログラムは EX6.FORT と改名します．例５で行った編集結合の代わりに，サブルーチンと関数は普段よく使われるということにして，私用のライブラリ MYLIB.LOAD のメンバーとして登録しておきます．私用のライブラリの定義・作成方法の詳細は [4],[5] を参照して下さい．ここでは，コマンドのみを示します．

```
  READY
  FORT SUB.FORT OBJECT(SUB.OBJ) NAME
  LINK SUB.OBJ LOAD(MYLIB.LOAD(SUB)) FORTLIB NCAL

  FORT FUNC.FORT OBJECT(FUNC.OBJ) NAME
 LINK FORT.OBJ LOAD(MYLIB.LOAD(FUNC)) FORTLIB NCAL
```

私用ライブラリに登録したサブルーチンを用いてメインプログラムのロードモジュールを作成します.

```
FORT EX5.FORT OBJECT(TEST.OBJ(EX6)) NOOPTIMIZE DBGINF
LINK TEST.OBJ(EX6) LIB(MYLIB) TEST LOAD(TEST.LOAD(EX6)) FORTLIB
** MENBER NAME ** EX6       NOW REPLACED IN LIBRARY.
```

最適化が問題なければ, TEST.LOAD(EX5) と TEST.LOAD(EX6) は全く同じものです.

４　デバッガの起動方法（１）ＧＯＤＢＧオプション

デバッガは第２節で述べたように, デバッガ用の情報を持ったロードモジュールを実行させることでデバッグを行います. そのためにユーザは第３節の例題などにならって, コンパイルオプション DBGINF および編集結合オプション TEST を指定してロードモジュールを作成する必要があります. 大規模なプログラムではこの方が効率的といえます. しかし, それほど大がかりでないプログラム（例えば, ＰＦＤで編集してそのまま RUN させるようなプログラム）のデバッグに対して, いちいちオブジェクトモジュールを作ってそれを編集結合させる作業をやるのはかなり面倒なことです. そこで FORTRAN77 EX では, コンパイラ起動コマンドに新オプション GODBG を指定することで,翻訳,結合編集およびデバッガの起動が一度にできる機能を装備しています.

4.1. ＧＯＤＢＧオプションの仕様

```
ＧＯＤＢＧ［（［ＡＲＧＣＨＫ　　｜　ＮＯＡＲＧＣＨＫ，］
　　　　［ＳＵＢＣＨＫ　　｜　ＮＯＳＵＢＣＨＫ，］
　　　　［ＵＮＤＥＦ　　｜　ＮＯＵＮＤＥＦ，　］
　　　　［ＩＯＶＥＲＦＬ　｜　ＮＯＩＯＶＥＲＦＬ，］
　　　　［ＯＶＥＲＬＡＰ　｜　ＮＯＯＶＥＲＬＡＰ］）］
```

《記号の意味》

［項目］は, この項目が省略できることをあらわします.

下線部は省略したときにとる項目を意味します.

【例】

GODBG : 全て下線部の省略値がとられる

GODBG(NOARGCHK) : ARGCHK を変更, あとは省略値のまま

GODBG(NOSUBCHK,NOIOVERFL) : SUBCHK,IOVERFL を変更, あとは省略値のまま

GODBGオプションのサブオプション（オプションのオプション）の意味は, DEBUG オプションのサブオプション（cf.[3],p269-p274）と等価です. 以下に簡単な説明をします.

ＡＲＧＣＨＫ	引数の妥当性の検査
ＳＵＢＣＨＫ	添字式及び部分列式の値の検査
ＵＮＤＥＦ	未定義データの引用の検査
ＩＯＶＥＲＦＬ	整数型オーバーフローの検出
ＯＶＥＲＬＡＰ	重複した実引数の副プログラム内での定義検査

″ＮＯ″が頭につくと，これらの検査をしないという意味になります． GODBG のみで全部の検査を行います．

4.2. ＧＯＤＢＧオプション指定時の動作

GODBG オプションを指定することで，原始プログラムに対して，以下の順序で翻訳，結合編集およびデバッガの起動が行われます．

(1) コンパイラオプション DBGINF と DBGINF(list) を指定したオブジェクトモジュールが一時データセット（作業のために一時的に作成され，作業の終了とともに消去されるデータセット）に作成されます．
　list ： GODBG オプションで有効になっているサブオプション（cf.4.1.）

(2) (1) のオブジェクトモジュールを入力として，オプション TEST が有効な状態で，結合編集が行われ，一時データセットにロードモジュールが作成されます．

(3) (2) のロードモジュールに対して，デバッガが起動されます．

デバッガが終了すると，ロードモジュールは消去されます．従って第３節で作成したようなデバッガ用の情報をもつロードモジュールはデータセットに保存されません．大規模なプログラムをデバッグする場合，もう一度デバッガを起動するためには，翻訳からやり直さなければなりませんのであまり効率がよくありません．なんども翻訳を行うのが煩わしいときは，第３節の手順に従って直ちに実行可能なロードモジュールをデータセット内に作成しておく方が有効です．

4.3. 注意事項

コンパイラオプション GODBG を指定した場合の注意事項を以下に示します．

《注意 1》 コンパイルオプション OPTIMIZE を指定しても，コンパイラオプション NOOPTIMIZE が有効になります．

【例７】
データセット EX3.FORT にコンパイルオプション GODBG と OPTIMIZE(B) をつけてデバッガを起動させます．

```
FORT EX3.FORT GODBG OPT(B)
```

このとき，OPT(B) の指定は無視され，NOOPT が設定されます．
OPT(B) は OPTIMIZE(B) の，NOOPT は NOOPTIMIZE のそれぞれ省略形です．

《注意 2》 翻訳指示行（原始プログラムの中に特別な意味を持つ翻訳指示のために書き込まれた文・行 cf.[3],p290-293 ）に DEBUG オプションの指定がある場合，翻訳指示行の DEBUG オプションのサブオペランド（オプションに関して演算を行わせる対象を指定するもの）が有効になります．

【例８】
データセット EX8.FORT の翻訳指示行には DEBUG オプションの指定があります.

```
@OPTIONS DEBUG(SUBCHK)
X=1.0/0.0
WRITE(6,*) X
END
```

EX8.FORT の中身

@OPTIONS の後に続けてコンパイラオプションを指定すると，翻訳時に指定された翻訳の手続きが行われます.
上のデータセット EX8.FORT に GODBG オプションをサブオプション付きで添加し，実行させます.

```
FORT EX8.FORT GODBG(NOSUBCHK,NOUNDEF)
```

DEBUG(SUBCHK) 指定の意味するところは,4.1. で挙げた機能のうち,SUBCHK のみを有効にして,あとは無効にする，ということです（cf.[3],p87）. @OPTIONS文によって指定されたオプションはFORT コマンドの GODBG オプションよりも優先されます. 従ってこの例では，翻訳単位としてSUBCHK 機能だけが有効となって，他の ARGCHK, UNDEF, OVERLAP, IOVERFL 機能は無効になります.

《注意 3》 結合編集時に，以下のメッセージがＤＤ名（データ定義名 data definition name：プログラム中で使用するデータセットを定義するとき,定義文につけられる名前 cf.[6]）ＳＹＳＴＥＲＭ に出力されます．ＤＤ名ＳＹＳＴＥＲＭで定義したデータセットを割り当てていないときは，端末に出力されます.

［メッセージ］
** MEMBER NAME ** TEMPNAME NOW ADDED TO LIBRARY.

【例９】
例２のデータセット EX3.FORT に対してＰＦＤの中からデバッガを起動させます.

```
EDIT --- A79999A.EX3.FORT ---------------------------------- 表示欄  001 072
コマンド ===> FORT GODBG                                      移動量 ===> HALF
****** **************************** データの先頭 **************V10L30********
000100        PROGRAM EX1
000200        X=1.0/0.0
000300        WRITE(6,*) X
000400        END
****** **************************** データの末尾 **************************
```

このとき，次のメッセージが表示され，デバッガが起動されます.

```
  【FORTRAN77 EX 翻訳開始】
  【翻訳終了】, 完了コード＝00
** MEMBER NAME ** TEMPNAME NOW ADDED TO LIBRARY.
DEBUG/I
```

最後の DEBUG/I は,モードメッセージ(現在の端末の制御状態を規定したメッセージであり,コマンドが入力できる状態を示す cf.6.3.）であり，デバッガが起動したことを意味します.
《注意 4》 コンパイラの復帰コード（プログラムやルーチンの終了状態を表すコード）は，翻訳，結合編集，デバッガの最大の復帰コードになります．実行における復帰コードは反映されません.

【例10】
例９の続きとして，デバッグ中に Ｅレベルのエラー（cf.[3] p135）を検出した場合.
なお，打ち込むサブコマンドの詳細は第６節を参照下さい.

```
 DEBUG/I
C
 浮動小数点の除算で除数が0です. PSWはOF8D1000810847CAです.
 実行時エラーが発生しました.
     :
 プログラムの実行が復帰コード8で終了しました.
 DEBUG/T
Q
  【実行終了】, 完了コード＝00
```

デバッグ中に発生した復帰コード８のエラーは，中度のエラーをあらわします.
一方，デッバガ自身は正常に終了したため，完了コード＝00 を出します.

５　デバッガの起動方法（２） ＤＥＢＵＧＧＥＲコマンド

第３節の手順に従って作成されたデバッガ用の情報を持つロードモジュールは，DEBUGGER コマンドによってデバッガを起動して解析します．このコマンドを入力した後，デバッグを行うサブコマンドが入力可能な状態になります.

5.1. 入力形式（ＤＥＢＵＧＧＥＲ）

コマンド	オペランド
ＤＥＢＵＧＧＥＲ または ＤＢＧ	データセット名 ［ＰＡＲＭ（’実行可能なプログラムオプション文字列’）］ ［ＣＰ］ ［ＣＯＮＶ｜ＮＯＣＯＮＶ］

5.2. オペランドの説明（ＤＥＢＵＧＧＥＲ）

・データセット名

デバッグ対象プログラムが格納されているロードモジュールのデータセット名を指定します. このオペランドは必須であり，最初に指定しなければなりません.

【例11】

例３で作成したデバッガ用の情報をもつロードモジュール TEST.LOAD(EX3) に対してデバッガを起動させます.

```
READY
DBG TEST.LOAD(EX3)
DEBUG/I
```

・ＰＡＲＭ（’実行可能なプログラムオプション文字列’）

実行時パラメタの文字列には，FORTRANプログラムに渡す実行可能なプログラムオプションを指定します(cf.[3],p93-p103). また，Ｃプログラムに渡す実行時パラメータを指定する場合にも使用します. 実行可能なプログラムオプションが不要な場合は，ＰＡＲＭオペランドは省略できます.

【例】

例11と同じロードモジュール TEST.LOAD(EX3) に対してデバッガを起動させます. ここで，実行可能プログラムオプションで，４バイトの整数型または８バイトの整数型になる式の計算や型変換でオーバーフローによるプログラム割込みが発生した場合，それを検出して，実行時のエラー処理を行うことを指定します([3],p99).

```
READY
DBG TEST.LOAD(EX3) PARM('FLIB(IOVERFL)')
DEBUG/I
```

・ＣＰ

デバッグ対象プログラムがコマンドプロセッサ(利用者が利用者端末から入力したコマンドの 処理をするプログラム）であることを示します. デバッグ対象プログラムへは，コマンドプロ セッサが通常起動されるときのパラメタリスト（ＣＰＰＬ）が渡されます. このオペランドを 指定すると，デバッグ対象プログラムへ渡すコマンドの入力を端末に促します. またオペラン の指定によってＰＡＲＭオプションは無視されます.

コマンドプロセッサについての詳細は「FACOM OS IV/F4 MSP TSS/E コマンドプロセッサ作成手引書(78SP-3371-1) 富士通,346ﾍﾟｰｼﾞ/6,700円」を参照下さい.

・ＣＯＮＶ ｜ ＮＯＣＯＮＶ

サブコマンドをＣ代替コード系で入力する場合にＣＯＮＶを指定し，EBCDIC(ASCII) コード系で入力する場合はＮＯＣＯＮＶと指定します.

省略値はＮＯＣＯＮＶです. 詳しくは [1] の p63 をご覧下さい.

６　デバッガの機能

6.1.デバッガの機能概略

デバッガの機能は大きく次の３つに分類できます.

① プログラムの実行を中断・再開する機能
② プログラム中のデータを表示・変更する機能
③ 補助的な機能

以下にそれらの機能を簡単に説明します.

① プログラムの実行の中断・再開

プログラムの実行を特定の位置で中断させるには，ＢＲＥＡＫサブコマンドを使用します.ＢＲＥＡＫサブコマンドによりプログラムに設けられた中断する位置のことを中断点と呼びます.中断点は，文単位だけでなく，入口や出口の機能単位でも指定できます．また，中断周期を指定することにより,中断点への到達回数によって,中断を行うかどうかを指定することもできます.

中断点を設けるプログラムは，ＢＲＥＡＫサブコマンドを使用したときに必ずしも仮想記憶域上に存在していなくてもかまいません．仮想記憶域上に存在しないプログラムに対する中断点の設定は，そのプログラムが仮想記憶域上に存在するかを意識せずに中断点を設定することができます.

また中断点に到達したときに,各種のサブコマンドを実行するように指定することも可能です.一連のサブコマンドは，ＢＲＥＡＫサブコマンドのオペランドにサブコマンド群で指定します.この機能を利用すると，データの値を変更するなどの，プログラムの一時的な修正ができます.ただし，ファイルの書き換えはできません．ファイルの書き換えはユーザがエディタ等によって別途行う必要があります.

中断点以外にも，プログラムの実行中にアテンションキー（STOPキーであったり PA1キーであったり端末によって異なります．）を押すことによりプログラムの実行を中断させることができます．アテンションキーによる中断は，アテンションキーが押し下げられた位置で実行が中断します．それ以外にも，異常終了や実行時エラーなどの異常状態が発生した場合にも，実行が中断します．異常状態による中断には，次のものがあります.

（１）異常終了の発生
（２）実行時エラーの発生
（３）再帰呼出しエラーの発生

実行を中断させたプログラムを中断位置から再び実行させるには，ＣＯＮＴＩＮＵＥサブコマンドを使用します．また，ＢＲＥＡＫサブコマンドで設定した中断点を解除したい時には，ＤＥＬＥＴＥサブコマンド使用します.

② プログラム中のデータの表示・変更

プログラム中のデータを表示するには，ＬＩＳＴサブコマンドを使用します．表示するデータの指定には，原始プログラムに記述できる変数名，配列要素名を記述します．データの表示形式は，ＬＩＳＴサブコマンドのオペランドで,次の３通りの書式を指定することができます.また,書式のオペランドを指定しなかった場合は，指定されたデータの型に対応した書式で表示されます.

（１）１６進表示形式は，値を１６進の数字列で表示します．
（２）文字表示形式は，値を文字データとして表示します．
（３）位置表示形式は，値が表す位置を原始プログラムの文番号に変換して表示します．

各形式の詳細は [1],p27-p32 をご覧下さい．

プログラム中のデータを変更するには，ＳＥＴサブコマンドを使用します．変更の指定には，右辺に定数，変数名，配列要素名，文番号を指定した算術代入文を記述します．ＳＥＴサブコマンドによる代入は，FORTRANの演算規則に従って代入します．

③ その他の補助的な機能

a）条件付きサブコマンドの実行

中断位置において，指定した条件が満たされた場合に一連のサブコマンドを実行するには，ＩＦサブコマンドを使用します．条件の指定には，２つの定数，識別名の値を比較する関係式，または論理型の変数を記述します．一連のサブコマンドは，ＩＦサブコマンドのオペランドにサブコマンド群で指定します．この機能を利用すると，条件によってサブコマンドのオペランドの実行を制御できるので，データの値が特定の値になったときだけ中断するなど，条件とサブコマンドを上手く組み合わせることによって，効率的なデバッグが可能になります．

b）中断位置の表示

現在，実行の中断している位置をＷＨＥＲＥサブコマンドを使用して表示することができます．

c）呼出し経路の表示

現在，実行の中断している位置へのプログラムの呼出し経路をＢＡＣＫＴＲサブコマンドを使用して表示することができます．

d）プログラムによる修飾

FORTRANプログラムが複数個ある場合，中断点や識別名とプログラムの対応をとる機能としてプログラム修飾(cf.6.4.)があります．プログラム修飾は，プログラムの実行中断時に中断位置のプログラム名が自動的に設定されますが，サブコマンドＩＮオペランドで明示的に指定したり，ＳＣＯＰＥサブコマンドにより，明示修飾を省略した場合に使用されるプログラム修飾を変更することができます．

e）文識別番号の指定形式の変更

中断点の位置は，原始プログラム上のエディタ行番号，相対行番号，またはＳＳＮ番号(cf.6.5.）のどれで指定してもかまいません．各番号は ＭＯＤＥサブコマンドで指定　します．

f）デバッグ環境の表示

プログラムに設定された中断点の位置，明示修飾を省略した場合に使用されるプログラム修飾，文識別番号の指定形式といったデバッグ環境を，ＳＴＡＴＵＳサブコマンドを使用して表示することができます．

g）端末表示の打切り

ＬＩＳＴ や ＳＴＡＴＵＳサブコマンドなどによる多量な情報を表示中に，アテンションキーを押し下げることにより表示を打ち切ることができます．サブコマンドによる表示をアテンションキーの押し下げによって打ち切ると，表示を実施しているサブコマンドに続くサブコマンドも合わせて打ち切られてしまいます．この場合，ＰＵＲＧＥサブコマンドを使用すると，表示を実施しているサブコマンドだけを打ち切り，後続のサブコマンドは実行させることができます．

その他にも，コメントを表示させるＮＯＴＥサブコマンド，サブコマンドの機能や使用方法を表示させるＨＥＬＰサブコマンド，デバッグを終了させるＱＵＩＴサブコマンドなどがあります．
ただし，TESTFORT77 にサポートされていた実行開始点の変更や，データをモニターする機能は今後サポートされる予定です（1992年11月25日現在）．

次頁 6.2. にサブコマンドの一覧を示します．サブコマンド欄で太字で表された箇所は，サブコマンドの指定で太字の部分だけ打ち込めばいいことを意味します．例えばＢＲＥＡＫサブコマンドは
ＢＲＥＡＫの代わりに Ｂ と打ち込んでも指定されたことになります．

6.2. サブコマンドの一覧

サブコマンド	主な機能
ＢＲＥＡＫ	デバッグプログラムの実行を中断，再開する位置を指定する
ＤＥＬＥＴＥ	ＢＲＥＡＫで設定した中断点を解除する
ＣＯＮＴＩＮＵＥ	中断している位置から実行を再開する
ＬＩＳＴ	プログラム中のデータの値を表示する
ＳＥＴ	データの値を任意の値に変更する
ＩＦ	条件を評価し，結果が真の場合にサブコマンド群の実行を行う
ＤＯＥＮＤ	ＤＯグループの終了を指示する
ＭＯＤＥ	文識別番号モードを変更する
ＳＣＯＰＥ	ＩＮオペランドを省略した場合にプログラム修飾に使用する 被デバッグプログラムを指定する
ＷＨＥＲＥ	実行が中断している位置を表示する

ＢＡＣＫＴＲ	実行が中断しているプログラムの呼出し経路を表示する
ＳＴＡＴＵＳ	文識別番号モード，暗黙プログラム修飾，設定されている中断点及び実行を延期しているＢＲＥＡＫサブコマンドとＤＥＬＥＴＥサブコマンドの情報を表示する
ＰＵＲＧＥ	サブコマンドによる表示を打ち切り，後続のサブコマンドを実行
Ｘ	ＴＳＳコマンドを実行する
ＤＵＭＰ	対象プログラムのタスクのスナップショットダンプを出力
ＨＥＬＰ	サブコマンドの機能や使用方法を表示する
ＮＯＴＥ	コメントを表示する
ＱＵＩＴ	デバッグを終了する

6.3. サブコマンドの入力

サブコマンドは，モードメッセージが表示されたときに入力できます．入力したサブコマンドに誤りがある場合は，エラーメッセージが表示されて，再びモードメッセージが表示されます．

サブコマンドが入力できる状態の一覧は以下の通りです．

中断の種類	中断の内容	メッセージ	入力不可サブコマンド
実行開始前	デバッグ対象プログラムの実行が開始する前の状態	DEBUG/I	ＷＨＥＲＥ ＢＡＣＫＴＲ
実行終了後	デバッグ対象プログラムの実行が終了した後の状態	DEBUG/T	ＣＯＮＴＩＮＵＥ
中断点	ＢＲＥＡＫサブコマンドで設定した中断点による中断	DEBUG	
実行時エラー 異常終了	実行時エラーまたは異常終了が発生した場合の中断	DEBUG/E	ＳＥＴ
再帰呼出しエラー	再帰呼出しエラーによる中断	DEBUG/C	ＣＯＮＴＩＮＵＥ

アテンション	デバッグ対象プログラムの実行中にアテンションキーを押すことによる中断	DEBUG/A	ＳＥＴ

6.4. ＨＥＬＰサブコマンド

このコマンドさえ知っておけば，とりあえずマニュアルや本稿をいちいち調べなくても，一応の構文がわかります.

1. 入力形式（ＨＥＬＰ）

サブコマンド	オペランド
ＨＥＬＰ Ｈ	[　サブコマンド名 ［ＦＵＮＣＴＩＯＮ］［ＳＹＮＴＡＸ］ [［ＯＰＥＲＡＮＤＳ［（キーワード名…）］］　］] ＡＬＬ

《入力形式の意味》

以下の記号｛　｝や［　］などは繰返し登場します．一見何を意味するのかよく判らないと思いますが，例題を参考にぜひ意味を理解して下さい.

なお，省略形は太文字の部分です.

｛　｝：選択記号
　この記号で囲まれた項目の中から，どれか一つを選択することを示します.

［　］：省略可能記号
　この記号で囲まれた項目を省略してもよいことを示します.

…　：反復記号
　この記号の直前の項目を繰り返して指定できることを示します.

＿　：省略値記号
　省略可能記号内の項目をすべて省略したときの省略値を示します.

2. オペランドの説明（ＨＥＬＰ）

・サブコマンド名

情報を得たいサブコマンド名を指定します．サブコマンド名はその省略形でも指定できます.

・ＦＵＮＣＴＩＯＮ

指定したサブコマンドの機能を表示します.

・ＳＹＮＴＡＸ

指定したサブコマンドの構文を表示します.

・ＯＰＥＲＡＮＤＳ［（キーワード名）］

指定したサブコマンドのオペランドについての情報を表示します．キーワードを省略した場合は，全てのオペランドの情報を表示します．特定のキーワードオペランドについての情報を得たい場合は，ＯＰＥＲＡＮＤＳの後ろにそのキーワード名を括弧でくくって指定します．キーワード名は，その省略形であっても構いません.

・ＡＬＬ

指定したサブコマンドの機能，構文，および全てのオペランドの情報を表示します.

3.注意事項（ＨＥＬＰ）

a) オペランドを省略した場合は，サブコマンド名の一覧表と，各サブコマンドの簡単な機能の説明を表示します.

b) サブコマンド名を省略した場合は，他のオペランドを指定することはできません.

c) ＦＵＮＣＴＩＯＮ，ＳＹＮＴＡＸ，ＯＰＥＲＡＮＤＳを指定すると，特定のサブコマンドの機能,構文,オペランドの説明文が出力されます.サブコマンド名だけを指定して，これらのオペランドを何も指定しない場合は，ＡＬＬを指定したものとみなし，指定したサブコマンドの機能，構文およびオペランドの情報がすべて表示されます.

d) ＨＥＬＰサブコマンドで表示されるサブコマンドの構文は，本文とは少し異なった形式で表現されます．構文の説明は [1]の p46 を参照下さい.

e) ＨＥＬＰサブコマンドは，バッチ処理や問題プログラムからの呼出しでデバッグする場合やＣＡＬＬコマンドで起動した場合は，使用できません.

【例12】

ＦＯＲＴコマンドにＧＯＤＢＧオプションをつけるか，ＤＥＢＵＧＧＥＲコマンドを入力するかして，先ずデバッガを起動させます.次に，デバッガのモードメッセージが出ている状態で，ＨＥＬＰサブコマンドを入力します.

```
 DEBUG/I
H
 サブコマンド -
   BACKTR,BREAK,CONTINUE,DELETE,DOEND,DUMP,HELP,IF,LIST,MODE,NOTE,
    PURGE,QUIT,SCOPE,SET,STATUS,WHERE,X
 FOR MORE INFORMATION, ENTER 'HELP SUBCOMMAND-NAME' OR 'HELP HELP'
 DEBUG/I
```

HELP または H のみを入力した場合はサブコマンドの一覧が表示されます.

モードメッセージが DEBUG/I 以外でも(cf.6.3) ＨＥＬＰサブコマンドは入力可能です.

【例13】

BREAKサブコマンドの機能についてのすべての情報を表示させます.

```
 DEBUG/E
H BREAK
 機能　－
 BREAKサブコマンドは，被デバッグプログラムに中断点を設定するのに用い
   る.

 構文　－
      BREAK/B　’中断点指定’ IN(’プログラム修飾’)/
                          :
                          :
```

スラッシュ(/) は選択記号を表します. 従って BREAK/B とは, BREAK または B のどちらか一方を入力することを示しています.

【例14】

MODEサブコマンドのオペランド NUM の情報を表示させます.

```
 DEBUG/T
HELP MODE 0(NUM)
     NUM      - 文識別番号モードを行番号モードに変更する.
 DEBUG/T
```

6.5. ＢＲＥＡＫサブコマンド

ＢＲＥＡＫサブコマンドは，被デバッグプログラムの実行を中断する位置を示す中断点を指定します.

1. 入力形式（ＢＲＥＡＫ）

サブコマンド	オペランド
ＢＲＥＡＫ Ｂ	中断点指定　［ＩＮ（ﾌﾟﾛｸﾞﾗﾑ修飾）］ ＥＮＴＲＹ　　ﾌﾟﾛｸﾞﾗﾑ名［ﾌﾟﾛｸﾞﾗﾑ名］… ［ＰＲＯＣ（　＊　　）］ ＥＸＩＴ　　＆ ＮＯＴＩＦＹ　　ｻﾌﾞｺﾏﾝﾄﾞ群 ［ＣＹＣＬＥ（整定数）］［　　］［　　］ ＮＯＮＯＴＩＦＹ　　ＤＯ

2．オペランドの説明（ＢＲＥＡＫ）

・中断点指定

実行を中断する位置を次の形式で指定します．

文識別番号＝BUN とします．

```
BUN
[BUN]:[BUN]
    BUN              BUN
(                [                ] …)
   [BUN]:[BUN]      [BUN]:[BUN]
```

《文識別番号とは》

FORTRANプログラムの文を引用するための指定であり，以下のどれかで指定します．指定された文識別番号がプログラム単位内で一意でない場合は，プログラム単位上で最初に定義された文識別番号となります．なお，サブコマンドのオペランドが，エディタ行番号，相対行番号またはＳＳＮ番号のどれかであるかは，文識別番号モード（cf.ＭＯＤＥサブコマンド）に従います．デバッグ開始時の文識別番号モードは，エディタ行番号モードです．文識別番号モードは，ＭＯＤＥ指定のＳＴＡＴＵＳサブコマンドによって表示することができます．

```
文番号
ＳＳＮ番号
                          エディタ行番号        [．Ｔ]
 [インクルード指定子]
                          相対行番号
```

上の語意を簡単に説明します．

a）文番号

FORTRANプログラム中に記述された文番号を使用して文を指定します．文番号はFORTRANプログラムに記述した文番号の前に，アスタリスク（＊）を付加して指定します．

【例15】

文番号 10 をもつソースプログラムに対する設定

```
000100        PROGRAM EX15
000200         GOTO 10
000300     10  I=1+2
000400        END
```

第５節の手順に従ってデバッガを起動させ，文番号 10 で実行を中断させるように指定します．

```
 DEBUG/I
BREAK *10
 DEBUG/I
C
'EX15'のNUM : 300 *10で中断しました.
 DEBUG
```

b) ＳＳＮ番号

原始プログラムの各文に対して，FORTRAN77 EX コンパイラが翻訳時に割り当てた文通番を使用して文を指定します．ＳＳＮ番号は，コンパイラが割り当てた文通番をそのまま指定します．

c) インクルード指定子

ＩＮＣＬＵＤＥ文によって，組み込んだFORTRANプログラム中の文を引用する場合に指定します．インクルード指定子は，引用したい文の組込みを指示するＩＮＣＬＵＤＥ文のエディタ行番号または相対行番号の前に英字の ″I″，後ろにピリオド（．）を指定します．

```
        エディタ行番号                  エディタ行番号
I                       . [ I                       . ] …
        相対行番号                      相対行番号
```

・インクルードされる原始プログラムがネストされている（ＩＮＣＬＵＤＥで組み込まれるプログラム中にもＩＮＣＬＵＤＥ文があること）場合は，インクルード指定子を複数個指定しなければなりません．
・インクルードライブラリ中の文を，文番号やＳＳＮ番号で引用する場合は，インクルード指定子は指定できません．

インクルード文の詳細は [3](p290-p292) を参照下さい．

【例16】

ＩＮＣＬＵＤＥ文によって組み込んだデータセット FORT.INC(NATCH) の 100行目で中断させます．

```
000100       PROGRAM EX16    ! 原始プログラム
000200         INCLUDE(NATCH)
000300         I=I+2
000400         WRITE(6,*) I
000500       END
```

ＤＤ名ＳＹＳＩＮＣに割り当てられたデータセット FORT.INC のメンバ NATCH の中身

```
000100          I=1
```

中断位置を組み込まれるインクルードライブラリに設定します.

```
    DEBUG/I
  BREAK I200.100
```

d) エディタ行番号

エディタ行番号を使用して文を引用します．エディタ行番号は，エディタの行番号を指定します．ただし，エディタの行番号が数字以外の場合は，エディタ行番号は 0 とします.

e) 相対行番号

ファイル内の相対行番号を使用して文を引用します．相対行番号は，
ＦＯＲＴＲＡＮの原始プログラムが格納されているデータセット内の相対行番号を指定します.

f) .T

論理ＩＦ文の論理式の結果が真の場合に実行される文（トレーラ文）を指定する場合に設定します．″．Ｔ″を指定した場合は，文番号ＳＳＮ番号，エディタ行番号または相対行番号で指定した論理ＩＦ文のトレーラ文を示します.

【例17】

論理ＩＦ文のトレーラ文を指定します.

```
000100        PROGRAM EX17     ! 原始プログラム
000200          I=0.1
000300          IF(I.NE.10E-1) I=0.0
000400          WRITE(6,*) I
000500        END
```

中断位置を設定します.

```
    DEBUG/I
  BREAK 300.T
```

・ＩＮ（プログラム修飾）

文識別番号を指定する場合に，文識別番号の有効範囲を指定します.

《プログラム修飾とは》

プログラム修飾は，サブコマンドに指定する文識別番号および識別名の有効範囲を指示するものです．プログラム修飾は，サブコマンドのＩＮオペランドで明示的に指定する方法と，ＩＮオペランドを指定せずに，現在有効なプログラム修飾（暗黙プログラム修飾）を使用する方法とがあります．

暗黙プログラム修飾は，プログラムの実行が中断したときに自動的に設定されます．被デバッグプログラム以外で中断した場合には，呼出し経路の中でもっとも近いプログラムを暗黙プログラム修飾にします．動作中の被デバッグプログラムが一つもない場合は，暗黙プログラム修飾は決まらず，ＩＮオペランドは省略できません．実行の開始前の中断では，システムは主プログラムを暗黙プログラム修飾とみなします．暗黙プログラム修飾は，ＳＣＯＰＥサブコマンドによって変更することができます．また，暗黙プログラム修飾は，ＳＣＯＰＥ指定のＳＴＡＴＵＳサブコマンドによって表示することもできます．

ＩＮオペランドおよびＳＣＯＰＥサブコマンドに指定するプログラム修飾は，以下のように指定します．また，指定されたプログラムに含まれる識別名，あるいは指定されたプログラムに定義された文識別名がオペランドの対象となります．

主プログラム
関数副プログラム
サブルーチン副プログラム

【例18】

例５で用いた３つのデータセットの中で，サブルーチン副プログラム SUB の 200 行に中断点を設定します．ただし，例５では，サブルーチン副プログラム SUB を格納してある SUB.FORT にはデバッガ用の情報を付加していませんでした．第３節の手順に従って，デバッグ用の情報を持ったロードモジュール TEST.LOAD(EX18) が作成されているとします．

デバッガを TEST.LOAD(EX18) に対して起動させ，中断点をＢＲＥＡＫサブコマンドによって設定します．

```
DBG TEST.LOAD(EX18)
 DEBUG/I
B 200 IN(SUB)
 DEBUG/I
C
'SUB'のNUM : 200で中断しました.
 DEBUG
```

（オペランド説明の続き）

・ＥＮＴＲＹ

プログラムの入口に中断点を設定する場合に指定します．

・ＥＸＩＴ

プログラムの出口に中断点を設定する場合に指定します.

・ＰＲＯＣ（｛プログラム名［プログラム名］…｜＊｜＆｝）

ＥＮＴＲＹまたはＥＸＩＴを指定する場合に，中断点を設定するプログラムを指定します．プログラム名を指定した場合，指定したプログラム名の入口または出口に中断点を設定します．アスタリスク（＊）を指定した場合は，すべての被デバッグプログラムの入口または出口に中断点を設定します．アンパサンド（＆）を指定した場合は，実行が中断しているプログラムの入口または出口に中断点を設定します.

・ＣＹＣＬＥ（整定数）

中断点にＣＹＣＬＥで指定した回数の制御が渡るたびに実行を中断する中断周期を指定します．省略した場合は，中断点に制御が到達するたびに中断します.

・｛ＮＯＴＩＦＹ｜ ＮＯＮＯＴＩＦＹ｝

実行を中断する場合に，実行の中断を示すメッセージを表示する（ＮＯＴＩＦＹ），表示しない（ＮＯＮＯＴＩＦＹ）を指定します.

・｛サブコマンドリスト ｜ ＤＯ｝

実行を中断した場合に，実行するサブコマンド群を指定します.

《サブコマンド群とは》

サブコマンド群とは，サブコマンドの集合のことをいいます．サブコマンド群には，サブコマンドリストで指定する方法と，ＤＯグループで指定する方法の２通りがあります．サブコマンド群はＢＲＥＡＫサブコマンドとＩＦサブコマンドのオペランドとして指定します．ＢＲＥＡＫサブコマンドに付加されたサブコマンド群は，中断点に到達し 中断する条件が満たされた時に実行されます．ＩＦサブコマンドに付加されるサブコマ ンド群は，ＩＦサブコマンドが実行され，ＩＦサブコマンドに記述された条件が満たさ れた時に実行されます.

サブコマンドリストの記述方法

サブコマンドリストは，サブコマンドの集合を以下の形で，オペランドとして指定します.

（［サブコマンド［；サブコマンド］…］）

サブコマンドを構成するサブコマンドについてもサブコマンド群を付加することができます．ただし，サブコマンドだけが記述可能であって，ＤＯグループでの指定はできません.

【例19】

例２で用いたプログラム EX2 を使い，ＢＲＥＡＫサブコマンドにコマンドリストを付加して中断点を設定します.

設定は，300 行で QUIT の内容を表示させ続行し，更に 400 行で変数 X を 5 に置き換えています.

```
 DEBUG/I
BREAK 300 (HELP Q;C)
 DEBUG/I
BREAK 400 (SET X=5)
 DEBUG/I
```

この設定で実行させてみます.

```
CONTINUE
 'EX2'のNUM : 300 で中断しました.
 機能 -
  デバッグセッションを終了する. このとき, 被デバッグプロ‥
                :
     (ＱＵＩＴサブコマンドの説明)
                :
浮動小数点の除算で除数が0です. PSWは0F8D1000810837CAです.
 実行時エラーが発生しました.
 'EX2'のNUM : 300 #INSIDEで中断中です.
 DEBUG/E
CONTINUE
   :
 DEBUG
CONTINUE
 5.0000000
   :
```

ＤＯグループの記述方法

ＤＯグループは, サブコマンドの集合をＤＯモードと呼ばれるモードで入力する方法です. ＤＯモードにするためには, ＤＯオペランドを指定します. ＤＯモードになると, モードメッセージ ″ＤＯ n″ が出力され, サブコマンドが入力可能となります.

ＤＯモードを終了するためには, ＤＯＥＮＤサブコマンドを入力します. ＤＯオペランドとＤＯＥＮＤサブコマンドの対でひとつのＤＯグループを構成します. ＤＯグループを構成するサブコマンドについてもサブコマンド群を付加することができ, サブコマンドリスト, ＤＯグループのどちらの形式で指定してもかまいません. モードメッセージ ″ＤＯ n″ の n は, ＤＯグループのネストレベルを示します. 最も外側のＤＯグループを構成するサブコマンド入力時には, ネストレベルが１であり, ネストが深くなるに従って１ずつ大きい値となります.

【例20】

例19と同じプログラム EX2 に今度はＢＲＥＡＫサブコマンドにＤＯオペランドを付加して中断点を設定します.

順にＨＥＬＰサブコマンドの説明を表示させ，変数 X を４に設定し，処理を続行させるという手続をとります.

```
 DEBUG/I
BREAK 400 DO
 DO 1
HELP HELP
 DO 1
SET X=4
 DO 1
C
 DO 1
DOEND
 DEBUG/I
```

続いて処理を実行させます.

```
 DEBUG/I
C
浮動小数点の除算で除数が0です. PSWは0F8D1000810837CAです.
 実行時エラーが発生しました.
 'EX2'のNUM : 300 #INSIDEで中断中です.
 DEBUG/E
  :
 'EX2'のNUM : 400で中断しました.
 機能  -
  HELPサブコマンドは，サブコマンドのオペランドまたは‥‥
     （ＨＥＬＰサブコマンドの解説）
 4.0000000
```

サブコマンド群の解析

サブコマンド群内のサブコマンドは，ＤＥＢＵＧモードでのサブコマンドと異なり，入力と同時には実行されません．サブコマンドは，記憶されるだけで，中断点への到達や条件の評価結果が真の場合に実行されます．サブコマンドを記憶する際に誤りが検出されれば，エラーメッセージが出力されます．誤りを検出した後の処置は，サブコマンドリストならば，サブコマンドリストを付加したサブコマンドを無効にします．ＤＯグループならば，記憶させようとしたサブコマンドだけを無効にします．サブコマンドの誤りには，サブコマンド群を実行する際に検出されるものもあります．

サブコマンド群の実行

サブコマンド群の実行は，サブコマンド群を構成するサブコマンドの集合を順次実行します．実行順序は，サブコマンドリストでは左から右に，ＤＯグループでは入力

された順です．サブコマンド群の実行中，ＣＯＮＴＩＮＵＥサブコマンドまたは
ＱＵＩＴサブコマンドが現れると，それより後のサブコマンドは実行されません．
最後までこれらのサブコマンドが現れないと，ＤＥＢＵＧモードとなります．サブコマンド群実行中に，誤りを含んだサブコマンドが検出されれば，エラーメッセージが出力されますが，サブコマンドの実行はそのまま続けられます．サブコマンド群が実行される時の文識別番号はモードは，サブコマンド群を記憶した時の文識別番号モードと関係なく，実際に実行される時点での文識別番号モードになります．また，プログラム修飾についても同様に，実際に実行される時点でのプログラム修飾です．

3．注意事項（ＢＲＥＡＫ）

a）中断点はすべての実行文に設定することができます．ただし，ＥＬＳＥ文，ＥＮＤＩＦ文およびオブジェクト命令列が展開されていない文は除きます．

b）ＥＮＴＲＹおよびＥＸＩＴは，構文標記上は，キーワードオペランドの形式ですが，ここでは位置オペランドとして扱うので，オペランドの最初に指定しなければなりません．

c）ＥＮＴＲＹを指定した場合に中断点を設定する入口とは，ＰＲＯＧＲＡＭ文，
ＦＵＮＣＴＩＯＮ文，ＳＵＢＲＯＵＴＩＮＥ文，およびＥＮＴＲＹ文の後の最初の実行文です．

【例21】

例18のプログラムにＥＮＴＲＹオペランドで中断点を設定します．

```
  DEBUG/I
 BREAK ENTRY
  DEBUG/I
 C
 ENTRY指定により，'EX3'のNUM ： 200で中断しました．
  DEBUG
 C
 ENTRY指定により，'SUB'のNUM ： 200で中断しました．
```

d）ＥＸＩＴ文を指定した場合に，中断点を設定する出口とは，ＲＥＴＵＲＮ文，ＥＮＤ文，およびＳＴＯＰ文です．

【例22】

例18のプログラムにＥＸＩＴオペランドで中断点を設定します．

```
  DEBUG/I
 BREAK EXIT
  DEBUG/I
```

```
C
 EXIT指定により，'EX3'のNUM ： 300で中断しました.
  DEBUG
C
 EXIT指定により，'FUNC'のNUM ： 300で中断しました.
```

e) コロン（:）で範囲を指定した場合は，左端に指定した文識別番号から，右端に指定した文識別番号までの間にあるすべての実行文に中断点を設定します．左端の文識別番号は，右端の文識別番号より原始プログラム上前になれらばなりません．左端を省略した場合は，プログラム単位内に記述された最初の実行文の文識別番号が設定されたとみなされます．右端を省略した場合は，プログラム単位内に記述された最後の実行文の文識別番号が指定されたとみなされます．

【例23】

行番号 200 から 500 までに中断点を設定します．

```
  DEBUG/I
BREAK 200:500
  DEBUG/I
```

f) ＥＮＴＲＹおよびＥＸＩＴを指定し，ＰＲＯＣオペランドを省略した場合は，
ＰＲＯＣ（＊）が指定されたとみなされます．

g) 中断点を設定した文は，ＢＲＥＡＫ指定のＳＴＡＴＵＳサブコマンドで表示することができます．

【例24】

例21で設定した中断点をＳＴＡＴＵＳサブコマンドで見てみます．

```
  DEBUG/I
STATUS BREAK
 BREAK POINTS
   SUB
     NUM ： 200 ENTRY
   FUNC
     NUM ： 200 ENTRY
   EX3
     NUM ： 200 ENTRY
  <OTHERS>
    BREAK ENTRY
```

h) 次の場合，ＰＲＯＣオペランドにアンパサンド（&）を指定できません．
(1) 実行の開始前の中断
(2) 動作中の被デバッグプログラムがひとつもない場合
(3) 実行の終了後の中断

i) 中断点を指定し，ＩＮオペランドを省略した場合は，暗黙プログラム修飾が指定されたとみなされます．ただし，暗黙プログラム修飾が設定されていないとき，ＩＮオペランドは省略できません．

j) ＣＹＣＬＥオペランドの整定数は 1～65535 の値でなければなりません．

k) 仮想記憶領域上に存在していないプログラムに対する中断点の設定は，入力と同時には実行されません．中断点の設定指示は記憶されるだけで，該当するプログラムが仮想記憶領域上へローディングされた時点で実行されます．中断点の設定指示を記憶する際に誤りが検出されれば，エラーメッセージが出力されます．プログラムがローディングされ，中断点を設定する際に検出される誤りもあります．

l) 設定した中断点は，次の場合無効になります．
(1) ＤＥＬＥＴＥサブコマンドの実行
(2) 同じ位置に対する中断点の設定
(3) 仮想記憶領域上からのプログラムのデリート
(4) デバッグの終了

6.6. ＤＥＬＥＴＥサブコマンド

ＤＥＬＥＴＥサブコマンドは，設定した中断点を解除します．

1. 入力形式（ＤＥＬＥＴＥ）

サブコマンド	オペランド
ＤＥＬＥＴＥ ＤＥＬ Ｄ	中断点指定　［ＩＮ（プログラム修飾）］ ＥＮＴＲＹ　　ﾌﾟﾛｸﾞﾗﾑ名［ﾌﾟﾛｸﾞﾗﾑ名］… ＥＸＩＴ　［ＰＲＯＣ（　＊　　）］ ＊　　&

2. オペランドの説明（ＤＥＬＥＴＥ）

・中断点指定

中断点を解除する位置を指定します．形式はＢＲＥＡＫサブコマンドと同じです．

・ＩＮ（プログラム修飾）

文識別番号を指定する場合に，文識別番号の有効範囲を指定します．

・ＥＮＴＲＹ
プログラムの入口に設定されている中断点を解除する場合に指定します.

・ＥＸＩＴ
プログラムの出口に設定されている中断点を解除する場合に指定します.

・＊
プログラムに設定されているすべての中断点を解除する場合に指定します.

・ＰＲＯＣ（｛プログラム［プログラム名］…｜＊｜＆｝）
ＥＮＴＲＹ，ＥＸＩＴまたはアスタリスク（＊）を指定する場合に，中断点を解除するプログラムを指定します．プログラム名を指定した場合には，指定したプログラム名に設定されている中断点を解除します．アスタリスク（＊）を指定した場合には，すべての被デバッグプログラムの設定されている中断点を解除します．アンパサンド（&）を指定した場合は，実行が中断しているプログラムに設定されている中断点を解除します.

3. 注意事項（ＤＥＬＥＴＥ）
ＤＥＬＥＴＥサブコマンドに関する注意は，ほぼＢＲＥＡＫサブコマンドに関する注意事項に対応しています．詳しくは［1](p26-p27）を参照下さい.

【例25】
ＢＲＥＡＫサブコマンドで中断点を設定したが，気が変わって中断点を解除しました.

```
 DEBUG/I
BREAK *10
 DEBUG/I
DEL *10
```

6.7. ＣＯＮＴＩＮＵＥサブコマンド
ＣＯＮＴＩＮＵＥサブコマンドは，中断している位置から実行を再開します.

1. 入力形式（ＣＯＮＴＩＮＵＥ）

サブコマンド	オペランド
ＣＯＮＴＩＮＵＥ ＣＯＮＴ Ｃ	なし

2. 注意事項（ＣＯＮＴＩＮＵＥ）

a) 実行を再開すると，プログラムの実行が終了するか，中断が成立するまで実行は中断しません.

b) 以下の場合は，ＣＯＮＴＩＮＵＥサブコマンドを入力することはできません.（cf.6.3.）

(1) 実行の終了後の中断

(2) 再帰呼出しエラーによる中断

【例26】

例６で作成したロードモジュールに対しデバッガを起動させ，ＣＯＮＴＩＮＵＥサブコマンドを押し続けます.

```
 READY
DBG TEST.LOAD(EX6)
 DEBUG/I
CONTINUE
 9.0000000
 プログラムの実行が復帰コード0で終了しました.
 DEBUG/T
CONT
 現状態ではCONTINUEサブコマンドは使用できません.
 DEBUG/T
```

6.8. ＬＩＳＴサブコマンド

ＬＩＳＴサブコマンドは，FORTRANプログラム中のデータの値を表示します.

1. 入力形式（ＬＩＳＴ）

サブコマンド	オペランド
ＬＩＳＴ Ｌ	識別名 全配列指定　　Ｈ 識別名　　識別名　　［ＦＯＲＭ（　Ｃ　）］ （　　［　　］…）　　Ｐ 全配列指定　　全配列指定 ［ＩＮ（プログラム修飾）］

2. オペランドの説明（ＬＩＳＴ）

・｛識別名｜全配列指定｜（｛識別名｜全配列指定｝［｛識別名｜全配列指定｝］…）｝

値を表示するデータを指定します.複数個指定された場合は,左から順に表示を行います.

・ＦＯＲＭ（｛Ｈ｜Ｃ｜Ｐ｝）

値の表示形式を指定します.

Ｈ：１６進表示形式

Ｃ：文字表示形式

Ｐ：位置表示形式

オペランドを省略した場合は，標準表示形式で出力されます.

値の表示形式についての詳細は [1], p29-p32 をご覧下さい.

・ＩＮ（プログラム修飾）

識別名，全配列指定の有効範囲を指定します.

《識別名とは》

被デバッグプログラムで宣言されたデータを識別するための名前です.

原始プログラムの変数名，または配列要素名のいずれかを指定します．ただし，擬寸法仮配列は配列名で指定することはできません.

配列要素名を指定する場合の注意点は [1],p21 を参照下さい.

《全配列指定とは》

全配列指定は,配列の全要素の引用を示します.全配列指定は,配列要素の参照の形式で,添字にアスタリスク（＊）を指定します．多次元配列の全要素を引用する場合でも，添字は一つだけ指定します.

【例27】

いろいろな配列を表示させます.

```
EDIT --- A79999A.EX27.FORT ---------------------------------- 表示欄  001 072
コマンド ===> FORT GODBG                                        移動量 ===> HALF
****** **************************  データの先頭  **************V10L30********
000100        PROGRAM EX27
000200         INTEGER A(3)/1,2,3/                ！'/' で値が設定できます
000300         REAL B(2,2)/4*1.0/                  ！ 全要素に１が入ります
000400         REAL*8 C/4.0/
000500         WRITE(6,*) C
000600        END
****** **************************  データの末尾  **************************
```

中断点を 500行に設定し，ここで各値をＬＩＳＴサブコマンドを用いて出力させます.

```
  DEBUG
 LIST C
  C = 0.4000000000000000D+01
  DEBUG
```

```
LIST A(2)
 A(2) = 2
 DEBUG
LIST B(*)
 B(1,1) =  0.1000000E+01  0.10000000E+01  0.10000000E+01  0.10000000E+01
```

3. 注意事項（ＬＩＳＴ）

a) 文の途中で実行が中断している場合は，実際の値と異なる値が表示されることがあります．

b) 識別名として，以下のものは指定できません．

(1) 仮手続き名
(2) 動作中でないプログラムの仮引数
(3) 動作中でないプログラムの関数結果
(4) 動作中でないプログラムのＡＥオプションの対象となった変数や配列
(5) 領域が割り当てられていない変数

c) ＩＮオペランドを省略した場合は，暗黙プログラム修飾が指定されたとみなされます．ただし，暗黙プログラム修飾が指定されていないとき，ＩＮオペランドは省略できません．

d) 引数によってその位置が渡されるデータなど，複数のプログラムが共用したデータを参照する場合には以下の点で注意が必要です．

《注意点》

中断位置が被デバッグプログラム内であり，参照するデータが中断位置のプログラム内で使用しているデータの時，参照するデータが中断位置においてレジスタ割付けされている可能性があります．よって，中断位置のプログラムをプログラム修飾に指定して参照して下さい．他のプログラムや外部プログラムをプログラム修飾に指定して参照すると，実際の値と異なる値が表示されることがあります．

6.9. ＳＥＴサブコマンド

ＳＥＴサブコマンドは，データの値を任意の値に変更します．

1. 出力形式（ＳＥＴ）

サブコマンド	オペランド
ＳＥＴ Ｓ	識別名② 識別名①＝　定数　　［ＩＮ（プログラム修飾）］ ＊文番号

2. オペランドの説明（ＳＥＴ）

・識別名①　　値を代入する識別名を指定します．

・識別名②　　代入したい値を持つ識別名を指定します.

・定数　　代入したい値を指定します.

・文番号　　代入したい位置を表す文番号の前にアスタリスク（＊）をつけて指定します.

・ＩＮ（プログラム修飾）　識別名や文番号の有効範囲を指定します.

3. 注意事項（ＳＥＴ）

a) 文の途中で実行が中断している場合は，ＳＥＴサブコマンドは入力できません.
b) 最適化されたプログラムのデータに対して，ＳＥＴサブコマンドは入力できません.
c) 値の代入は FORTRANの規則に従います.
d) 右辺に文番号を指定した場合，左辺は４バイトの整数型でなければなりません，このときの代入は，FORTRAN の ＡＳＳＩＧＮ文の代入規則に従います.

その他の注意事項は 6.8. の注意事項 b),c),d) と同じです.

【例28】

例27のデータセット EX27.FORT を用います. ＳＥＴサブコマンドによって配列の値を変更　させます.

```
 DEBUG
SET B(1,2)=2.0
 DEBUG
L B(*)
 B(1,1) =  0.1000000E+01  0.10000000E+01  0.20000000E+01  0.10000000E+01
```

配列 B の１行２列の成分が変更されました.

6.10. ＩＦサブコマンド

ＩＦサブコマンドは，条件を評価し結果が真の場合に，サブコマンド群の実行を行います.

1. 出力形式（ＩＦ）

サブコマンド	オペランド
ＩＦ	識別名　関係　識別名　　ｻﾌﾞｺﾏﾝﾄﾞﾘｽﾄ 定数　演算子　定数　［ＩＮ（ﾌﾟﾛｸﾞﾗﾑ修飾）］ 論理変数　論理変数　　ＤＯ

2. オペランドの説明（ＩＦ）

・｛識別名｜定数｝関係演算子｛識別名｜定数｝

評価する条件式を指定します．関係演算子には以下のものが指定できます．

関係演算子	意味
．ＥＱ．　＝＝	等しい
．ＮＥ．　＜＞	等しくない
．ＬＴ．　＜	小さい
．ＬＥ．　＜＝	小さいか等しい
．ＧＴ．　＞	大きい
．ＧＥ．　＜＝	大きいか等しい

・ＩＮ（ﾌﾟﾛｸﾞﾗﾑ修飾）
　条件に指定した識別名の有効範囲を指定します．

・｛サブコマンドリスト｜ＤＯ｝
　条件式の評価結果が真の場合に実行するサブコマンド群を指定します．

3．注意事項（ＩＦ）

a）文の途中で実行が中断している場合は，条件の評価結果が異なることがあります．
b）条件式の評価は，FORTRAN の演算子の優先順位と結合規約に従います．
c）全配列指定は指定できません．
d）条件に指定されたデータの値が参照できない等,条件を評価できない場合は，評価結果は偽となります．

その他の注意事項は 6.8. の注意事項 b),c),d) と同じです．

【例29】 次のプログラムを考えます．

```
000100       PROGRAM EX29
000200         INTEGER A(2)/1,2/,X/0/
000300         X=X+A(1)+A(2)
000400         WRITE(6,*) X
000500       END
```

Xの値が 0ならば（そうに決まっていますが）配列要素 A(2) と X の値を設定し直して実行させます．

```
 DEBUG/I
IF X==0 (SET A(2)=3;SET X=100)
 DEBUG/I
C
 104
 プログラムの実行が復帰コード0で終了しました.
 DEBUG/T
```

6.11. ＤＯＥＮＤサブコマンド

ＤＯＥＮＤサブコマンドは，ＤＯグループの終了を指示します.

入力形式（ＤＯＥＮＤ）

サブコマンド	オペランド
ＤＯＥＮＤ	なし

6.12. ＭＯＤＥサブコマンド

ＭＯＤＥサブコマンドは，分識別番号モードを変更します.

1. 入力形式（ＭＯＤＥ）

サブコマンド	オペランド
ＭＯＤＥ Ｍ	ＮＵＭ ［　ＳＥＱ　］［ＦＯＲＴ］ ＳＳＮ

2. オペランドの説明（ＭＯＤＥ）

・ＮＵＭ

文識別番号モードをエディタ行番号モードにします.

・ＳＥＱ

文識別番号モードを相対行番号モードにします.

・ＳＳＮ

文識別番号モードをＳＳＮ番号モードにします. 言語種類がＦＯＲＴの場合のみ指定できます.

・ＦＯＲＴ

変更する文識別番号の言語種類を指定します.

3. 注意事項（ＭＯＤＥ）

a) 文識別番号モードは，サブコマンドのオペランドに指定する文識別番号の指定形式を示します．ただし，文番号は文番号モードに依存しないで指定することができます.

b) デバッグ開始時の文識別番号モードは，エディタ行番号モードです.

c) 文識別番号モードは，ＭＯＤＥ指定のＳＴＡＴＵＳサブコマンドで表示することができます.

d) ＳＴＡＴＵＳサブコマンドによる中断点の指示，中断点による実行の中断を示すメッセージを除いて，文識別番号は，現在の文識別番号モードに従って表示されます.

e) 言語種別を省略した場合は，暗黙プログラム修飾の言語種別が指定されたとみなされます．ただし，言語種別を確定できない場合は，言語種別を省略することはできません.

f) 文識別番号モードは言語種類ごとに設定されているので，一方を省略しても，他方には影響しません.

g) 言語種類がＣの場合については，[1] 第２部 11.10 を参照下さい.

6.13. ＳＣＯＰＥサブコマンド

ＳＣＯＰＥサブコマンドは，ＩＮオペランドを省略した場合に，プログラム修飾に使用する被デバッグプログラムを定義します.

1. 入力形式（ＳＣＯＰＥ）

サブコマンド	オペランド
ＳＣＯＰＥ ＳＣ	［プログラム修飾］

2. オペランドの説明（ＳＣＯＰＥ）

・プログラム修飾

ＩＮオペランドを省略した場合にプログラム修飾に使用する被デバッグプログラムを指定します.

3. 注意事項（ＳＣＯＰＥ）

a) オペランドを省略した場合は，現在実行が中断している被デバッグプログラムを修飾に使用します．ただし，動作中の被デバッグプログラムが一つもない場合は，オペランドを省略することはできません.

b) ＩＮオペランドを省略した場合に修飾に使用する被デバッグプログラムは，ＳＣＯＰＥ指定のＳＴＡＴＵＳサブコマンドで表示できます.

c) ＳＣＯＰＥサブコマンドで定義した修飾に使用する被デバッグプログラムは，新しい

ＳＣＯＰＥサブコマンドの入力か，ＣＯＮＴＩＮＵＥサブコマンドの入力によって無効となります．

【例30】

例18で用いたプログラムを使って中断点の設定をします．

```
DBG TEST.LOAD(EX18)
 DEBUG/I
SCOPE SUB
 DEBUG/I
BREAK 200
 DEBUG/I
 C
  'SUB'のNUM ： 200で中断しました
 DEBUG
```

ＳＣＯＰＥサブコマンドで被デバッグプログラムを SUB.FORT に割り当てました．従って次のＢＲＥＡＫサブコマンドではＩＮオペランドを省略しても中断点は SUB.FORT に設定されます．

6.14. ＷＨＥＲＥサブコマンド

ＷＨＥＲＥサブコマンドは，現在実行が中断している位置を表示します．

1. 入力形式（ＷＨＥＲＥ）

サブコマンド	オペランド
ＷＨＥＲＥ Ｗ	なし

ＷＨＥＲＥサブコマンドの出力形式は [1],p38 をご覧下さい．また ＨＥＬＰサブコマンドのＷＨＥＲＥオペランドを指定することでも参照が可能です．

2. 注意事項（ＷＨＥＲＥ）

a) 実行の開始前の中断の場合は，ＷＨＥＲＥサブコマンドは入力できません．

b) 現在の中断位置が被デバッグプログラムの場合は，ＦＯＲＴＲＡＮプログラム上の位置を，現在の文識別番号モードに従って，エディタ行番号，相対行番号またはＳＳＮ番号のどれかで表示します．

その他注意事項の詳細は [1],p39 をご覧下さい．

【例31】

例30の続きでＷＨＥＲＥサブコマンドを入力してみます.

```
 DEBUG
WHERE_
 'SUB'のNUM : 200で中断中です
 DEBUG
```

6.15. ＢＡＣＫＴＲサブコマンド

ＢＡＣＫＴＲサブコマンドは，現在実行の中断しているプログラムの呼出し経路を表示します.

1. 入力形式（ＢＡＣＫＴＲ）

サブコマンド	オペランド
ＢＡＣＫＴＲ ＢＴＲＷ	なし

ＢＡＣＫＴＲサブコマンドの出力形式は [1],p40 をご覧下さい. また ＨＥＬＰサブコマンドのＢＡＣＫＴＲオペランドを指定することでも参照が可能です.

2. 注意事項（ＢＡＣＫＴＲ）

呼出し経路の表示で，被デバッグプログラム以外が存在する場合は，”？？？”を表示します. ただし，被デバッグプログラム以外の呼出しが複数回連続していても，一回しか表示されません.

【例32】

例31の続きでＢＡＣＫＴＲサブコマンドを入力してみます.

```
 DEBUG
BACKTR_
 SUB CALLED FROM NUM : 200 IN(EX3)
 EX3 CALLED FROM DEBUGGER
 DEBUG
```

6.16. ＳＴＡＴＵＳサブコマンド

ＳＴＡＴＵＳサブコマンドは次の情報を出力します.

① 文識別モード
② 暗黙プログラム修飾
③ 設定されている中断点および実行を中断しているＢＲＥＡＫサブコマンドとＤＥＬＥＴＥサブコマンド

1. 入力形式（ＳＴＡＴＵＳ）

サブコマンド	オペランド
ＳＴＡＴＵＳ ＳＴ	［ＭＯＤＥ］［ＳＣＯＰＥ］［ＢＲＥＡＫ］ ［　　］ ＡＬＬ プログラム名［プログラム名］… ［ＰＲＯＣ（　＊　）］ ＆

ＳＴＡＴＵＳサブコマンドの出力形式は［1],p42 をご覧下さい．また ＨＥＬＰサブコマンドのＳＴＡＴＵＳオペランドを指定することでも参照が可能です．

2. オペランドの説明（ＳＴＡＴＵＳ）

・ＭＯＤＥ

文識別モードを表示することを示します．

・ＳＣＯＰＥ

暗黙プログラム修飾を表示することを示します．

・ＢＲＥＡＫ

設定されている中断点および実行を延期しているＢＲＥＡＫサブコマンドとＤＥＬＥＴＥサブコマンドを表示することを示します．

・ＡＬＬ

ＭＯＤＥ，ＳＣＯＰＥ．ＢＲＥＡＫのすべての情報を表示します．

・ＰＲＯＣ（｛［プログラム名［プログラム名］…｜＊｜＆｝）

ＢＲＥＡＫを指定した場合に表示の対象とするプログラムを指定します．特定のプログラムだけを表示する場合，そのプログラム名を指定します．現在実行が中断しているプログラムを表示する場合はアンパサンド（＆）を指定します．すべてのプログラムを表示する場合は，アスタリスク（＊）を指定します．オペランドを省略した場合は，アスタリスク（＊）が指定されたとみなされます．

3. 注意事項（ＳＴＡＴＵＳ）

次の場合は，ＰＲＯＣオペランドにアンパサンド（＆）を指定できません．

（１）実行時の開始前の中断

（２）動作中の被デバッグプログラムが一つもない場合

（３）実行の終了後の中断

【例33】

例32の続きでＳＴＡＴＵＳサブコマンドを入力してみます.

```
 DEBUG
STATUS_
 DEBUGGER MODE : FORT - NUM
                 C    - SEQ
 SCOPE         : SUB
 BREAK POINTS
   SUB
     NUM : 200
 DEBUG
```

6.17. ＰＵＲＧＥサブコマンド

ＰＵＲＧＥサブコマンドは，サブコマンド群内のサブコマンドによる表示を打切り，後続のサブコマンドを実行します.

1. 入力形式（ＰＵＲＧＥ）

サブコマンド	オペランド
ＰＵＲＧＥ Ｐ	なし

2. 注意事項（ＰＵＲＧＥ）

a) ＰＵＲＧＥサブコマンドは，サブコマンド群に指定されたサブコマンドによる表示を打ち切るために，アテンションキーを押し下げて端末割込みを発生させた後に使用します.

b) サブコマンド名は，入力の先頭になければなりません.

c) 端末割込みで表示を打ち切ることができるサブコマンドは，次の通りです．その他のサブコマンドの表示は打ち切ることはできません.

- ・ＬＩＳＴサブコマンド
- ・ＳＴＡＴＵＳサブコマンド
- ・ＢＡＣＫＴＲサブコマンド
- ・ＨＥＬＰサブコマンド

d) アテンションキーを押し下げたとき，サブコマンド群のＸサブコマンドを実行している時は，ＰＵＲＧＥサブコマンドの入力の有無に関係なく，サブコマンド群の後続するサブコマンドを実行します.

6.18. Ｘサブコマンド

Ｘサブコマンドは，ＴＳＳコマンドを実行します．

1. 入力形式（Ｘ）

サブコマンド	オペランド
Ｘ	ＴＳＳコマンド

2. オペランドの説明（Ｘ）

・ＴＳＳコマンド

実行したいＴＳＳコマンドを指定します．

3. 注意事項（Ｘ）

a) ＰＲＯＦＩＬＥコマンドやＴＥＲＭＩＮＡＬコマンドを利用して，ユーザ属性や端末属性を変更しても，実行中のデバッグ処理に反映することはできません．

b) 次のＴＳＳコマンドは実行することができません．

・ＬＩＢＲＡＲＹコマンド

・ＴＥＳＴコマンド

・ＤＥＢＵＧＧＥＲコマンド

c) Ｘサブコマンドは，バッチ処理や問題プログラムからの呼出しでデバッグする場合やＣＡＬＬコマンドで起動した場合には使用できません．

d) デバッグ対象プログラムと同じ名前を持つロードモジュールを利用するプログラムや，コマンドをＸサブコマンドで呼び出す場合は，デバッガにより仮想記憶領域上にローディングされているロードモジュールが利用される可能性がありますので注意が必要です．

【例34】

デバッガ起動中に，バッチジョブの状況を調べます．

```
 DEBUG/E
X_ST_
 JOB A79999A#(TSU0634) IS EXECUTING
 JOB A79999A1(JOB05450) IS WAITING FOR OUTPUT
 DBG
 DEBUG
```

6.19. ＤＵＭＰサブコマンド

ＤＵＭＰサブコマンドは，デバッグ対象プログラムが動作しているタスクのスナップショットダンプを出力します．

1. 入力形式（ＤＵＭＰ）

サブコマンド	オペランド
ＤＵＭＰ	ＰＲＩＮＴＤＤ（ＤＤ名）

2. オペランドの説明（ＤＵＭＰ）

・ＰＲＩＮＴＤＤ（ＤＤ名）

スナップショットダンプを出力するデータセットを割り当てたＤＤ名を指定します.

3. 注意事項（ＤＵＭＰ）

a) デバッグ対象プログラムが，複数のタスクで動作する場合は，デバッグ対象プログラムが最初に動作したタスクのスナップショットダンプを出力します.

b) ＤＤ名として，ＳＹＳＡＢＥＮＤ，ＳＹＳＵＤＵＭＰおよびＳＹＳＭＤＵＭＰは指定できません.

c) データセットへの出力方法については，ＤＤ名の割り当て状態に依存します．スナップショットダンプをデータセットに追加する場合は，ＡＬＬＯＣＡＴＥコマンドのオペランドでＭＯＤを指定します.

d) データセットの属性は以下の属性でなければなりません.

ＤＳＯＲＧ＝ＰＳ または ＰＯ(ＰＯの場合はメンバ名指定が必要です)

ＲＥＣＦＭ＝ＶＢＡ

ＬＲＥＣＬ＝１２５

ＢＬＫＳＩＺＥ＝８８２ または １６３２

スナップショットダンプについては，各システムのデバッグ手引書を参照下さい.

6.20. ＮＯＴＥサブコマンド

ＮＯＴＥサブコマンドは，指定された文字列を表示します.

1. 入力形式（ＮＯＴＥ）

サブコマンド	オペランド
ＮＯＴＥ	’　　’ 文字［文字］… ”　　”

2. オペランドの説明（ＮＯＴＥ）

・文字

表示したい文字の列を指定します．文字の左辺の右辺に指定する引用符（’）および二重引用符（”）は同じものでなければなりません．文字には，英字，数字，各国用文字および母国語文字が指定できます.

・注意事項

引用符（’）で区切られた文字の中の引用符（’）は，二つの連続した引用符（’）を一つの文字として数えます．同様に，二重引用符（”）で区切られた文字の中の二重引用符（”）は，二つの連続した二重引用符（”）を一つの文字として数えます．

【例35】

```
 DEBUG/I
NOTE ’これはびっくりタケノベルベットだ！’
 これはびっくりタケノベルベットだ！
 DEBUG/I
```

6.21. ＱＵＩＴサブコマンド

ＱＵＩＴサブコマンドは，デバッグを終了します．

入力形式

サブコマンド	オペランド
ＱＵＩＴ Ｑ	なし

【参考文献】

参考文献は1992年11月25日現在入手できる最新のものです．ただし，マニュアルや利用の手引は常に書き換えられてます．従って本文で参照したページは，参考文献の更新に伴い異なることもありますのでご注意下さい．

［1］「OS IV/MSP デバッガ使用手引書 FORTRAN, C言語用 (70SP-6430)」 富士通株式会社．

［2］「FORTRAN77 EX コンパイラの最適化オプションについて」
九州大学大型計算機センターニュース，No.463, 1992.

［3］「OS IV/MSP FORTRAN77 EX 使用手引書 V12 用 (79SP-5031)」 富士通株式会社．

［4］「OS IV/MSP リンケージエディタ／ローダ使用手引書 AF2 V10 用 (79SP-3410)」
富士通株式会社．

［5］「利用の手引・ＴＳＳ編（第３版）」 九州大学大型計算機センター・ネットワーク室．

［6］「利用の手引・バッチジョブ編（第２版）」 九州大学大型計算機センター・ライ

ブラリ室.

参考文献の入手・参照方法

1.マニュアル

例えば富士通のマニュアル [1],[3],[4] などは，九州大学大型計算機センター内にあるプログラム相談室（２階）あるいは図書室（４階）で閲覧することができます．また，センターではコピーサービスは行っておりませんのでご注意下さい．

ユーザが個人，あるいは研究室で購入されるときは，例えば九州大学生協書籍部を通じて購入されるなどの方法があります．ちなみに値段は税別で次の通りかなりお高くなっています．

[1]: 185ページ　4,300円
[3]: 555ページ　13,000円
[4]: 245ページ　5,100円（新版が出る予定です）

2.利用の手引き，講習会資料

入手方法の詳細は，九州大学大型計算機センター・共同利用掛にお尋ね下さい．