Pre-Compile 方式による FORTRAN Debugger の開発

林　　雄　二*
中　津　正　志**

FORTRAN Debugger by the pre-compiling method

Yuji HAYASHI
Masashi NAKATSU

要　旨

プログラムの実行過程を追跡するデバッグルーチンを開発した。既成のコンパイラやオペレーティングシステムに手を加えることなしに作り上げることを意図したもので，原始プログラムに前処理をほどこして，FORTRAN コンパイラの入力として送り込むものである。前処理では，変数（配列要素），文番号出力のための呼出し命令を，原始プログラムの必要な個所に挿入する方式をとっている。

Synopsis

The debugging system that is useful for tracing process of running program was developed. This system was made without revising FORTRAN compiler now used, for it is always not easy and not desirable to revise a compiler already made. This system, by the pre-compiling method, insert CALL statements into the places where it is expected to put out the values of variables (or subscripted variables) and the statement numbers. Then the source programs made by this system are the inputs for FORTAN compiler.

1．は　じ　め　に

プログラムを完成させる過程で，実行時に生ずる種々の誤りを検出することは，プログラマーにとって一般に容易ではない。そのために，デバック補助のユーティリティプログラムが数多く開発されているが，実行時の変数（配列要素）値，文番号の追跡ルーチンは，有効なユーティリティプログラムの１つである。

本校の電子計算機で使用している FORTRAN では，このユーティリティプログラムが使用できずプログラムデバッグに支障をきたしていたが，今回，原始プログラムを書き換える手法によって働く追跡ルーチンを開発した。このような新システムを作る場合に，既存のソフトウェアシステムに手を加えることは，必らずしめ容易とはいえず，また，そのための充分な情報が得られるとは限らない。我々は，メーカーから提供された FORTRAN コンパイラや，オペレーティングシステムに修正を加えることなしに作り上げることを意図してこのシステムを完成した。

本処理プログラムは，指定された Debug 文に従って，前処理の段階で変数（配列要素）値や文番号を出力するルーチンの呼び出し命令（CALL 文）を原始プログラムに挿入し，作り出された原始プログラムを FORTRAN コンパイラの入力とする方式をとっている。機能及び用法は，一般に用いられている追跡ルーチンとほぼ同じ仕様にしてある。

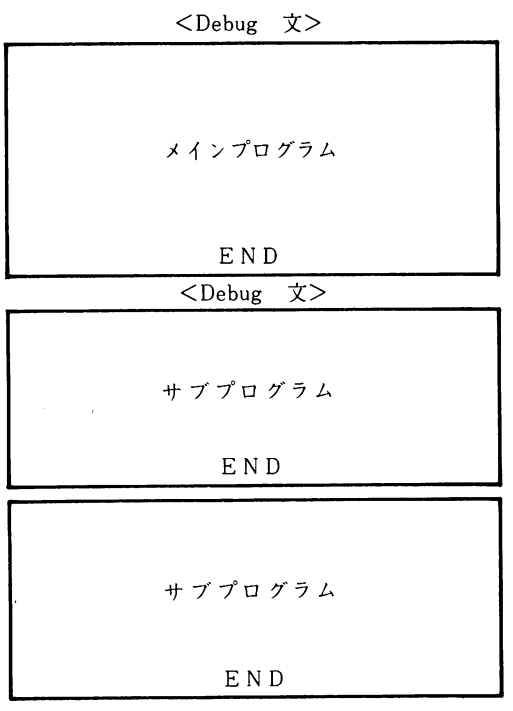
2．Debug 文

本システムは，Debug 文を指定することによって，プログラム実行時に任意の個所で，変数（配列要素）値及び通過した実行文の文番号を追跡することが可能である。

Debug 文の指定位置は，各プログラムエレメントの先頭であることが必要である。すなわち，追

＊助教授　機械工学科
＊＊助　手　機械工学科

跡の対象がメインプログラムであれば，メインプログラムの先頭に，サブプログラムであれば，サブプログラムの先頭に与える。



図－1

Debug 文の文法は以下の様に規定し，記述は，FORTRAN の文と同様に，第7欄以後とする。

〈Debug 文〉 : :=TRACE 〈List〉
〈List〉 : := 〈Element〉 | 〈List〉, 〈Element〉
〈Element〉 : := 〈名前〉 | 〈文番号〉 |
　　＊ | ＊＊ | 〈Range〉
〈Range〉 : := ＊ (〈文番号〉, 〈文番号〉) |
　　＊＊ (〈文番号〉, 〈文番号〉) |
　　〈名前〉(〈文番号〉, 〈文番号〉)

ここに，〈名前〉は FORTRAN で許されている変数名，配列名であり，〈文番号〉は FORTRAN で許されている文番号である。

〈名前〉指定は，プログラム実行時に，この変数（配列）が，代入文，DO 文によって値が変化した時点でその値を出力することを意味する。

＊指定は，プログラムエレメントのすべての変数（配列）に対して〈名前〉の指定が行なわれたものとする。

〈名前〉(〈文番号〉, 〈文番号〉) 指定は，プログラムエレメントのこの2つの文番号の間においてのみ〈名前〉の指定がなされたものとする。

＊ (〈文番号〉, 〈文番号〉) 指定は，プログラムエレメントのこの2つの文番号の間においてのみ＊指定がなされたものとする。

〈文番号〉指定は，プログラムエレメントで，この文番号の文が実行される時点で，この文番号を出力する。対象となる文番号は，実行文の文番号である。

＊＊指定は，プログラムエレメントのすべての実行文の文番号が指定されているものとする。

＊＊ (〈文番号〉, 〈文番号〉) は，プログラムエレメントのこの2つの文番号の間においてのみ＊＊指定がなされたものとする。

3．処　理　方　法

3.1　処理概要

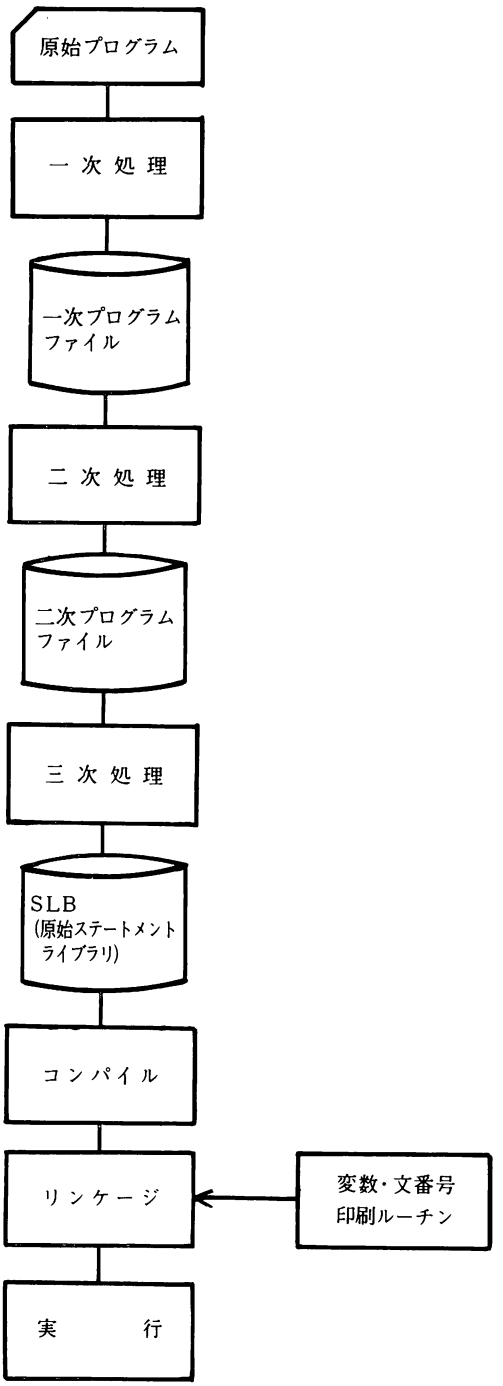
本 Debug 文処理ルーチンは，原始プログラムを書き換え，追跡処理が行なえる原始プログラムに修正した後，FORTRAN コンパイラにその原始プログラムを渡す方式をとっている。すなわち，Debug 文を解析し，名前や文番号の指定を調べ，名前の場合には，その変数（配列）が変化する位置に，値を出力するルーチンの呼出し命令を挿入し，文番号の場合には，その実行文が実行される個所に，文番号出力ルーチンの呼出し命令を挿入する。

処理は，以下に示されるとおり，一次処理，二次処理，三次処理から成っており，三次処理の完了時点で，新たな原始プログラムが作り出され，コンパイラの入力ファイルとして許されている SLB（利用原始ステートメントライブラリ）に格納される。

3.2　一次処理

Debug 文及び FORTRAN 原始プログラムを入力し，Debug 文テーブルを作成する一方，宣言文を解析することにより変数テーブルを作成する。作られたテーブル類と入力した原始プログラムとから，ディスクパックに，一次原始プログラムファイルを作成する。

Debug 文テーブルは，Debug 文に指定された名前，＊，文番号，＊＊について，その指定範囲を記憶しておくものである。変数テーブルは，宣言が与えられたすべての名前についての型を，名前と型の組として記憶しておくものである。一次原始プログラムファイルは，Debug を除いた原始プログラムを入力順に記憶し，各プログラムエレメ



図－2

ントの最後尾に Debug 文テーブル及び変数テーブルを記憶している。

名前	文番号1	文番号2
*		

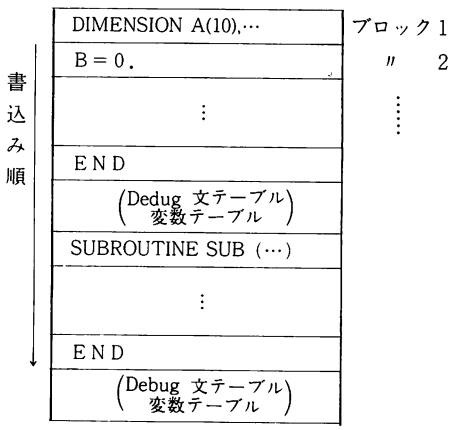
文番号	文番号1	文番号2
* *		

Debug 文テーブル

名　前	型

変数テーブル

図－3

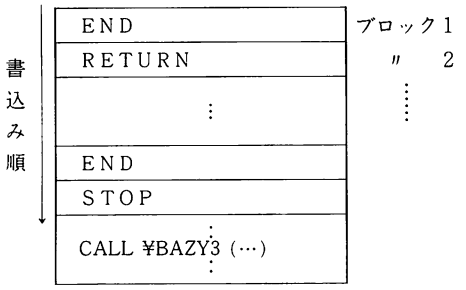


図－4　一次原始プログラムファイル

3.3　二次処理

一次原始プログラムファイルより，逆順に Debug 文テーブル，変数テーブル及び原始プログラムを入力し，上記テーブルに従って必要な位置に CALL 文(変数値, 文番号出力ルーチン呼出し)を挿入し，新たな FORTRAN プログラムを作成する。作られたプログラムは，二次原始プログラムファイルとして記憶されるが，これは，END 文から始まる逆順に直されたプログラムファイルである。

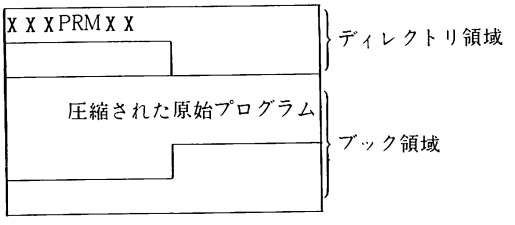
一次処理においては，DO 文の解析を行なっていないため，二次処理では DO の最終文から入力し必要な CALL 文を挿入しなければならない。これは後述の DO 文に対する CALL 文挿入方法からくる制約である。このため，二次処理では，原始プログラムを逆順に入力している。



図－5　二次原始プログラムファイル

3.4 三次処理

二次原始プログラムファイルを最後尾より入力し（元の原始プログラムの順序となる），独自に確保したSLBに登録する。SLBに登録するのは，FORTRANの入力ファイルとして，ディスクパックファイルはSLBでかければならないという制約からくるものである。この処理ルーチンでは，原始プログラムの圧縮を行ないSLBの形式に整えると共に，SLBのディレクトリの内容も独自に変更を加えている。原始プログラムは，常にSLBの同一位置に書き込み，プログラム名（ブック名）も同一（ＸＸＸPRMＸＸ）である。従って一定のコントロールカードで制御が行なえる。



図－6　SLBの形式

4．CALL文の組込み

二次処理では，原始プログラムにDebug文指定に従って変数（配列），文番号の出力ルーチン呼出し命令（CALL文）を挿入する。挿入するCALL文は以下のとおりである。

CALL ¥BAZYn（15H 名前、 名前）

ただし　n＝0　は　論理型
　　n＝1　は　倍精度実数型
　　n＝2　は　複素数型
　　n＝3　は　整数型
　　n＝4　は　実数型

CALL ¥AZYXW（4H 文番号）

サブルーチン¥BAZYnは，それぞれの型に従って，変数（配列要素）の値をその名前と共に出力するサブルーチンである。¥AZYXWは，引渡された文番号を出力するサブルーチンである。これらはいずれもFORTRAN言語で作られ，相対形式にコンパイルされており，結合編集の段階で組み込まれる。以下，名前，文番号，それぞれの場合についてCALL文の挿入方法を述べる。

4.1 変数（配列要素）の出力

① DOの最終文でない代入文（代入文を含む論理IF文も含む）及びDO文の場合には，その文の直後にCALL ¥BAZYn（）を挿入する。

例）

```
〔    A＝B＊C                          〕
⇨
⇨ ⎡  A＝B＊C                          ⎤
  ⎣  CALL ¥BAZYn (15HA     ,A      )  ⎦
```

② DOの最終文が代入文の場合には，DO文の直後に，制御変数が初期値の時を除いて値を出力させ，同時にDOの最終文の直後でも出力させるべくCALL文を挿入する。

例）

```
⎡    DO  100  I＝1, 10                ⎤
⎢          ⋮                          ⎥
⎣100 A＝B＊C                          ⎦
⇨
⇨ ⎡    DO  100  I＝1, 10               ⎤
  ⎢    IF (I.NE.1)  CALL ¥BAZYn (15HA    ,A    )
  ⎢                ⋮                   ⎥
  ⎣100 A＝B＊C                         ⎦
       CALL ¥BAZYn (15HA       ,A      )
```

4.2 文番号の出力

① DO文，分岐文後外の実行文で，DOの最終文でない場合は，その文の直後に，CALL ¥AZYXW（　）を挿入する。

例）

```
〔10  A＝B＊C                          〕
⇨
⇨ ⎡10  A＝B＊C                        ⎤
  ⎣    CALL ¥AZYXW (4H10    )         ⎦
```

② DO文及び分岐文の場合には，その直前にCALL ¥AZYXW（　）を挿入し，同時に文番号をこのCALL文に移す。

例）

```
〔10  GO  TO  100                      〕
⇨
⇨ ⎡10  CALL ¥AZYXW (4H10    )         ⎤
  ⎣    GO  TO  100                     ⎦
```

③ DO の最終文の場合には，DO 文の直後に，制御変数が初期値でない場合に文番号を出力させ，DO の最終文の直後でも出力させるべく CALL 文を挿入する。

例）

```
      DO  10  I=1, 100
            :
10    A=B*C
⇨
      DO  10  I=1, 100
      IF (I.NE.1)  CALL  ¥AZYXW (4H10    )
            :
10    A=B*C
      CALL  ¥AZYXW (4H10    )
```

文番号，名前の両方が同一ステートメントに指定された場合でも，上記処理が両方共に行なわれる。

5．使　用　例

実際に本 Debug ルーチンを使用した例を示す。

Debug 文指定なしの原始プログラム

```
        FORTRAN                          SOURCE LISTING

ISN    SOURCE STATEMENT
0001          A=1.
0002          DO 100 I=1,5
0003          A=A*(A-0.5)+1.
0004      100 CONTINUE
0005          STOP
0006          END
```

Debug ルーチンで作られた原始プログラム。
ただし，Debug 文は，

TRACE *，* *

```
        FORTRAN                          SOURCE LISTING

ISN    SOURCE STATEMENT
0001          A=1.
0002          CALL ¥BAZY4(15HA          ,A          )
0003          DO 100 I=1,5
0004          IF(I      .NE.1      ) CALL ¥AZYXW(4H100 )
0005          CALL ¥BAZY3(15HI          ,I          )
0006          A=A*(A-0.5)+1.
0007          CALL ¥BAZY4(15HA          ,A          )
0008      100 CONTINUE
0009          CALL ¥AZYXW(4H100 )
0010          STOP
0011          END
```

実行結果

```
      *** DEBUG *** A              =   0.1000000E 01
      *** DEBUG *** I              =           1
      *** DEBUG *** A              =   0.1500000E 01
-TRACE-100
      *** DEBUG *** I              =           2
      *** DEBUG *** A              =   0.2500000E 01
-TRACE-100
      *** DEBUG *** I              =           3
      *** DEBUG *** A              =   0.6000000E 01
-TRACE-100
      *** DEBUG *** I              =           4
      *** DEBUG *** A              =   0.3400000E 02
-TRACE-100
      *** DEBUG *** I              =           5
      *** DEBUG *** A              =   0.1140000E 04
-TRACE-100
```

6．あ　と　が　き

追跡ルーチンを，原始プログラムの Pre-Compile によって実行させる我々の方法は，既存のシステムに手を加えなくともよいという利点を持つ反面，機能的にかなりの制限をうける。例えば，代入文を含んだ論理 IF 文に対して，代入文の結果を出力する場合に，論理式が真の場合のみについて出力することは，容易とはいえない。また，DO の最終文で制御変数を添字として持つ変数に対する代入が行なわれる場合には，4.1 ②の方法では，最終値の出力が必らずしも容易に得られない。しかし，これらについては，原始プログラムの大巾な修正まで考えるとすれば，解決は不可能なことではない。

現在の段階では，本処理ルーチンは，変数（配列）の値及び文番号の出力を対象としているが，添字範囲のチェックが行なえること，出力条件を論理式で与えられ，条件を充す範囲でのみ追跡を行なえること等が今後の課題である。

また，この処理プログラムは，コンパイラと同様な構文解析をかなり行なっており，処理の効率は，プログラム側から見ると，必らずしもよいとはいえない。このようなことから，作成時にあらかじめ機能の拡張が考慮されたコンパイラが期待される。

本処理ルーチンの作成言語はアセンブラ（約 1200 ステップ）である。

本研究は，文部省教育改善経費による計画の一環として行なった電子計算機室室員グループの活動の一部である。

参　考　文　献

(1) Program Test Methods, William C. Hetzel, Prentice-Hall

(2) HITAC8250 NDOS ユーティリティプログラムシステム編集，日立

（昭和 52 年 11 月 30 日受理）