ALGOL Compiler の開発 (2)

稲田信幸

目　次

2.6.4. 宣言の解析

宣言には型の宣言，配列の宣言及び手続きの宣言がある。型の宣言と配列の宣言はオブジェクトコードは作られないが，手続きの宣言はオブジェクトコードが作られる。何の宣言であるかは第1シラブルと第2シラブルで判定することができる。

型の宣言

real〈名前〉，……，〈名前〉あるいは integer〈名前〉，……，〈名前〉の構文になっているか調べる。名前が同一ブロック内で二重定義になっていない

か調べる。ブロック解析表の Z, S, T, X 及び Y の各パートに 0, BLC の値, BGC の値, 名前表に登録されている番地, 及び宣言情報表へのポインタを入れる。また宣言情報表の D, K, M, A, B, C, ADD 各パートに 0, 0, 0, 0, (実数の時 2, 整数の時 3), 0 及び LC (ロケーションカウンタ) の値を入れる。このあと LC を 1 上げ, すべての名前について繰り返えす。

配列の宣言

real array 又は **integer array** のあとに〈名前〉又は〈名前〉, ……, 〈名前〉が来ているか調べる。更にそのあとに [a : b] 又は [a : b, c : d] と来ているか調べる。ここで a と c は配列の下限で b と d は上限である。a, b, c 及び d は,

$$0 \leqq a \leqq b,$$

$$0 \leqq c \leqq d$$

の両方を満足する整数でなければならない。

i, j, k の 3 つの値を 1 次元, 2 次元に応じて計算する。

$$i = d - c + 1$$

$$j = a \times i + c$$

$$k = (b - a + 1) \times i$$

を計算する。(1 次元のときは d=c=0 で計算する)

i の値を定数表に登録した番地を P とする。配列名はスタックされているので, pop up して配列名がなくなるまで次の動作を繰り返えす。配列名をスタックからとり出し, 同一ブロック内で二重定義か調べる。ブロック解析表の各パートには型の宣言と同様に入れ, 宣言情報表の D, K, M, A, B, C 及び ADD パートに 0, (1 次元の時 0, 2 次元の時 1), P の値, 2, (実数の時 2, 整数の時 3) 及び (LC−j) の値を入れる。このあと LC を k だけ上げる。この動作をスタックに入っている名前すべて繰り返えし, 次のシラブルが , (カンマ) の時は **array** のあとから上の動作を繰り返えすとよい。

手続きの宣言

procedure, real procedure 又は **integer procedure** のあとに名前 (手続名)

が来ているか調べる。この名前が同一ブロック内で二重定義か調べる。ブロック解析表の各パートには型の宣言と同様に入れ，宣言情報表の D, K, M, A, B, C 及び ADD パートに 0, 0, 0, 3,（0 又は 2 又は 3），0 及び 0 を入れる。手続名を左辺にもつ代入文が現われたら K パートに 1 を入れ，仮パラメータの解析後 M パートにパラメータの個数を入れる。ADD パートは PASS-II でオブジェクトコード出力中に入れられる。ここで手続き宣言の節 

を作り，この節のポインタをスタックする。

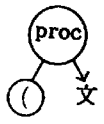
次のシラブルが ；（セミコロン）の時は仮パラメータ部が空であるので直ちに手続き本体の解析に入る。；ではなくて（ のときは仮パラメータ部の解析に入る。

（ の次の構文が〈名前〉, ……, 〈名前〉）; となっているか調べ，これらの名前をスタックする。この段階でブロックのレベルを 1 つ上げ，仮想のブロックを作る。すなわち BLC と BGC を共に 1 上げればよい。スタックから名前をとり出して，ブロック解析表を型の宣言のように作成し，宣言情報表の D, K, M, A, B, C 及び ADD パートに 0, 0, 0, 0, 0, 2 及び FLC（仮パラメータ用のロケーションカウンタ）の値を入れる。FLC と FPC（仮パラメータの個数を数えるカウンタ）を共に 1 上げる。A, B 及び C パートは規制の部の解析中に入れられる。スタックされた名前すべてについて以上の動作を繰り返えす。次は値の部があれば解析する。

value 〈名前〉, ……, 〈名前〉; の構文になっているか調べる。**value** 指定になっている名前が仮パラメータに列挙されているか調べ，二度以上 **value** 指定になっていないか調べた後で宣言情報表の C パートを 3 に変える。次の規制の部は仮パラメータ部があれば必ず書かなければならないものである。

〈規制詞〉〈名前〉, ……, 〈名前〉; の構文になっているか調べる。〈規制詞〉は **string, real, integer, real array** 又は **integer array** のいずれかである。各名前が仮パラメータに列挙されているか調べ，二度以上規制されてい

ないか調べた後で，宣言情報表の A と B パートを **stning** の時 B を 1 に，**real** の時 B を 2 に，**integer** の時 B を 3 に，**real array** の時 A を 2，B を 2 に，そして **integer array** の時は A を 2，B を 3 に変えてやる。規制の部は 1 つとは限らないので注意を要する。手続きの本体の解析に入る前すべての仮パラメータが規制されたか調べる必要がある。

手続きの本体の解析は文の解析ルーチンを実行するだけでよい。文の解析ルーチン実行後は仮想ブロック後処理のためブロック後処理ルーチンを実行し，文の節のポインタと節（ のポインタをスタックから pop up してとり出し，手続き宣言の節 

を作り，この節のポインタをスタックする。

2.6.5. 名札と飛越文の解析

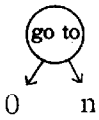
飛越文の行先式は JIS 水準 3000 の ALGÖL では名札だけであるから構文解析はとても簡単である。しかし，飛越文がその飛越文が属するブロックより内側のブロックにある名札を指してはならないことと，未定義の名札を指定できるため（名札が飛越文より後に宣言される場合と飛越文の属するブロックより外側のブロックで宣言される場合），この処理が多少複雑となっている。

飛越文の解析

go to 〈名前〉の構文を調べた後，この名前が同一ブロック内で既定義か調べる。

既定義の時は名前が名札として宣言されているか調べる。そして **go to** の節 

を作り，この節のポインタをスタックする。

未定義の時はひとまず未定義 **go to** の節 

を作り，この節のポインタをスタックする。未定義表の Z, S, T, X 及び L パートに 0，BLC の値，BGC の値，名前表に登録してある番地，未定義 **go to** 節のポインタを入れる。

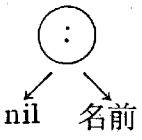
名札の解析

〈名前〉: の構文を調べた後，名前が同一ブロックで二重定義か調べる。ブロック解析表へは型の宣言と同様に入れ，宣言情報表の D, K, M, A, B, C 及び ADD パートに 0, 0, 0, 1, 0, 0, 0 を入れる。ADD パートは PASS-II でオブジェクトコード出力中に名札の番地が決定し，その時入れられる。

未定義のままでこの名札を使用していた飛越文があるかも知れないので，未定義表の S と X パートにマスクをかけ，現 BLC の値と名前表の登録番地とが等しいものを捜し，その未定義表の L パートは未定義 **go to** 節のポインタであるので，L パートの示す節 

を 

に変え，等しいものすべてについて繰り返えす。これ以外の未定義 **go to** 節の処理は前述のブロック後処理で扱われている。

最後に名札の宣言の節 

を作り，この節のポインタをスタックする。

2.6.6. 式の解析

式には算術式，比較式及び行先式がある。比較式は条件文の解析に，行先式は飛越文の解析に含まれる。ここでは算術式の解析が主になる。本 ALGŌL コンパイラは順位表を用いた bottom up 解析と top down 解析を併用している。それで変数と関数呼出は各々の解析ルーチンで行なっている。+，−，×，/，↑，(，) の算術作用素（演算記号）については左右の operator の順位を表にして，右の operator が左の operator より順位が低い時節を作っている。

節を作る時混合演算を禁止しているので，整数型と実数型を区別するフラグを節に設けている。変数名や手続名はこれらが属するブロックを含め，外側のブロックで既定義でなければならない。関数呼出の解析は手続文のそれと同じである。算術式を表わす節のコードは左右の足の型の違いによって表 5 のように異なった値を入れている。関数呼出のパラメータを表わす

,（カンマ）もパラメータの種類に応じて異なったコードが入れられる。

算術式を解析した後，トリーの根の節を指すポインタには型を表わす情報を入れておく。

2.6.7. 代入文の解析

:= の左辺が変数か手続名であるか調べ，型を記憶しておく。右辺は算術式解析ルーチンを実行する。左辺と右辺の型が異なる時は表5に従って := のコード22を54又は55に変えてから代入文の節 

を作り，この節のポインタをスタックする。

2.6.8. 条件文の解析

条件文は

1. **if**〈比較式〉**then**〈無条件文〉**else**〈文〉
2. **if**〈比較式〉**then**〈繰返文〉
3. **if**〈比較式〉**then**〈無条件文〉

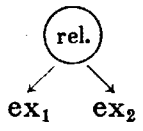
のいずれかの構文をしているので，FŌR-SWITCH を設けて，**then** のあとに繰返文が来た時 on にする。FŌR-SWITCH が on なのに **else** 以下が続く時はエラー扱いにする。条件文の解析の中に文の解析があるので回帰的サブルーチンとなっている。

比較式の解析手順

step 1. 算術式解析ルーチンを実行し，型を記憶する。

step 2. 次に来ているシラブルが比較作用素か調べてからスタックする。

step 3. 算術式解析ルーチンを実行し，**step 1.** の算術式の型と一致しているか調べる。

step 4. スタックから2つの算術式の節のポインタ及び比較作用素を pop up し，比較式の節 

を作り，この節のポインタをスタックする。

条件文の解析手順

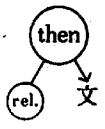
step 1. データをスタックし，FŌR-SWITCH を off にする。

step 2. 最初のシラブルが **if** か調べてスタックする。

step 3. **if** と **then** の間の比較式を解析し，**then** をスタックする。

step 4. **then** の次の文が **if** 文ならエラーの status を上げ，**for** 文なら FŌR-SWITCH を on にする。

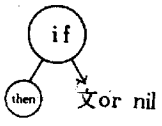
step 5. 文の解析ルーチンを実行する。

step 6. スタックから文のポインタ，比較式の節のポインタ及び **then** を pop up して，**then** の節 

を作り，この節のポインタをスタックする。

step 7. 次のシラブルが **else** でなかったら，nil をスタックして **step 9.** へ行く。**else** の時 FŌR-SWITCH が on か調べ，on の時エラーの status を上げる。

step 8. 文の解析ルーチンを実行する。

step 9. スタックから文のポインタ又は nil，**then** の節のポインタ及び **if** を pop up して，**if** の節 

を作り，この節のポインタをスタックする。

step 10. スタックしたデータを元に戻す。

2.6.9. 繰返文の解析

繰返文は

for 〈整数型単純変数〉:= 〈算術式〉 **step** 1 **until** 〈算術式〉 **do** 〈文〉

の構文をしている。JIS 3000 の ALGŌL では **step** と **until** の間の値は必ず 1 でなければならなく，各算術式も整数型でなければならない。繰返文の構文の中に文があるので回帰的サブルーチンとなっており，次の手順で解析される。

step 1. データをスタックする。

step 2. 最初のシラブルが **for** か調べて，スタックする。

step 3. **for** と := の間に整数型の単純変数が来ているか調べる。単純変

数と := のコード22を44に変えて，これらをスタックする。

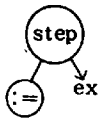
step 4. 算術式解析ルーチンを実行し，整数型であるか調べる。

step 5. スタックから算術式のポインタ，:= 及び単純変数を pop up して，:= の節 

を作り，この節のポインタをスタックする。

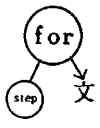
step 6. 次に来るシラブルが **step**，1 及び **until** であるか調べて，**step** と **until** をスタックする。

step 7. 算術式解析ルーチンを実行し，整数型か調べる。

step 8. スタックから算術式のポインタ，**until, step** 及び := の節のポインタを pop up して，**step** の節 

を作り，この節のポインタをスタックする。

step 9. 次のシラブルが **do** か調べ，文の解析ルーチンを実行する。

step 10. スタックから文のポインタ，**step** の節のポインタ及び **for** を pop up して，**for** の節 

を作り，この節のポインタをスタックする。

step 11. スタックしたデータを元に戻す。

2.6.10. 手続文の解析

手続文は

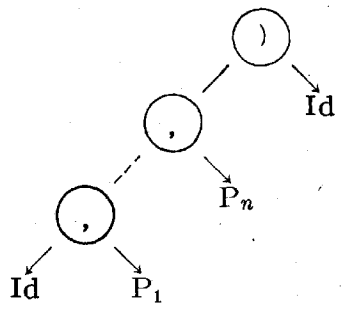
1. 〈手続名〉

2. 〈手続名〉(〈実パラメータ〉, ……, 〈実パラメータ〉)

のいずれかの構文をしている。

1の場合は名前が手続きとして宣言されているか調べて，スタックするだけでよく，2の場合は1の場合と同じようにしてスタックしたあと，実パラメータの解析をして，実パラメータの節を作ればよい。実パラメータと仮パラメータの対応によって表5のように45から52までの値を，（カンマ）に与える。実パラメータは記号列，算術式及び配列名のいずれかである。第1パラメータの節の左足には手続文の節 ）の右足と重複して手続名を入れて

いる。すなわち手続文が Id(P_1, P_2, ..., P_n) の時，これらの節は



となる。

3. PASS-II

3.1. 概　　要

PASS-II の仕事は PASS-I の出力結果である中間語のトリーを機械語のオブジェクトプログラムに変換することである。節の左右の足のいずれかが名前の時は宣言情報表を使用する。本 ALGŌL コンパイラはオブジェクトコードの最適化はしていない。またオブジェクトコードを二次記憶装置に出力しないで直接主記憶上に出力している。

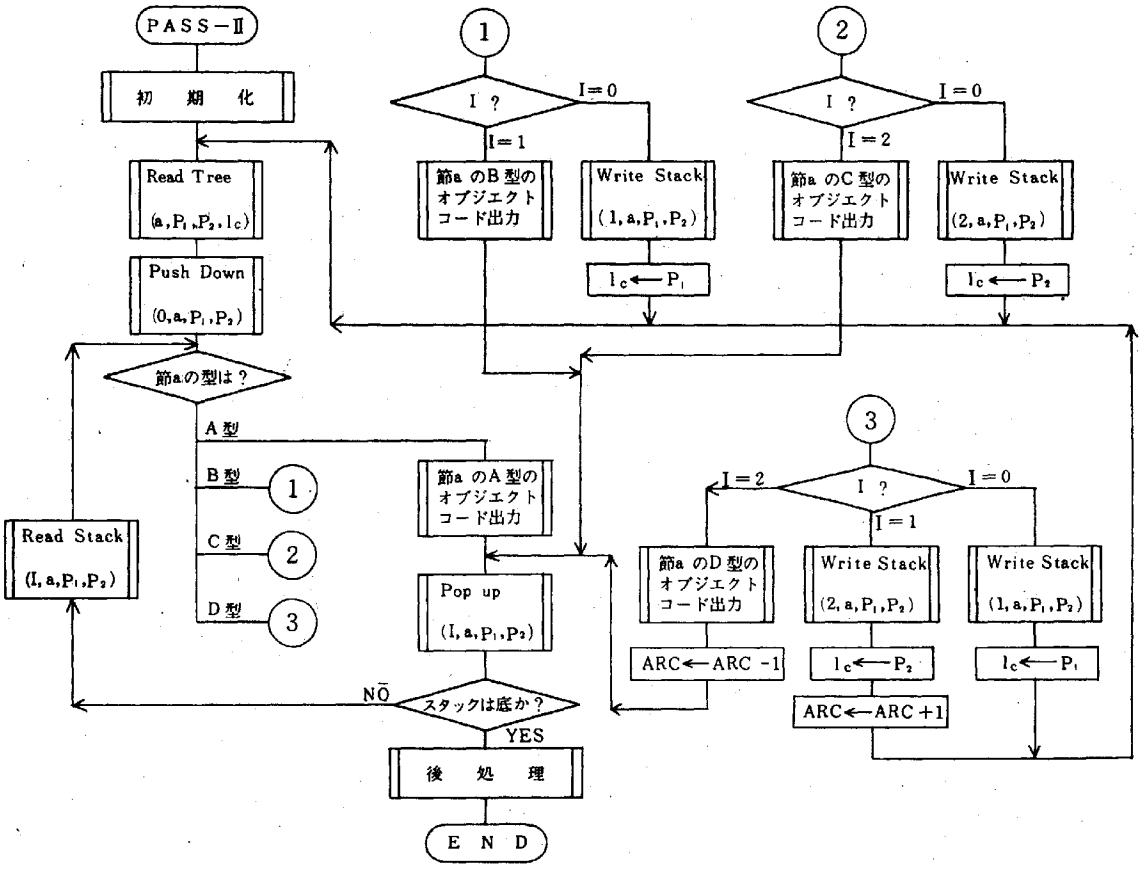


図6　PASS-II 処理

PASS-II の概略は図6のようになる。翻訳の際，オブジェクトコード主記憶格納位置を示すロケーションカウンタを設け，H には64個の AR（アリスメティックレジスタ）があるので演算レジスタ指定カウンタである ARC を設けて，オブジェクトコード作成の際命令の AR パートに ARC の値を入れてオブジェクトプログラムがいちいち中間結果を退避させることなく1つ上の AR で計算できるように考慮されている。

中間語のトリーの根の節のポインタ（l_c）は PASS-I からモニタエリアを介して引き渡される。

3.2. コードジェネレーション

〈名札と飛越文〉

飛越文と名札の節は A 型だけである。飛越文の指している名札が既定義かどうかを示すために宣言情報表の D パートに1を入れ，D パートが0でない時，名札の番地が決っているとする。既定義な名札をもつ **go to** 節は名札の宣言情報表の ADD パートの値を L とすると

J , , L

と翻訳される。しかし，名札の番地が決っていない時はひとまず

L1 : J , , 0

と翻訳しておき，芋ずる領域の空いている番地を L2 とする時，L2 番地に L1 と名札の宣言情報表の ADD パートの値を入れ，宣言情報表の ADD パートには L2 を入れる。

名札の節の場合は，オブジェクトコードは出力されないが宣言情報表の ADD パートが0の時はロケーションカウンタの値を ADD パートに入れ，D パートを1にして終る。

宣言情報表の ADD パートが0でなく，D パートが0の時は芋ずるで次のように処理する。

step 1. 宣言情報表の ADD パートをとり出して，この値を R とする。

step 2. R 番地の M 及び N パートをそれぞれと P, Q する。

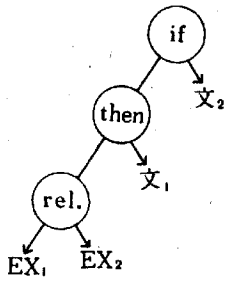
step 3. P 番地の番地部にロケーションカウンタの値を入れる。

step 4. Q の値を R に入れ，R が 0 でない時 **step 2.** へ行って繰り返えす。

芋ずる処理後，宣言情報表の ADD パートにロケーションカウンタの値を入れ，D パートを 1 にして終る。

〈条件文〉

条件文のトリーは一般に



の形をしている。本 ALGŌL コンパイラの条件文の翻訳は

	EX_1 及び EX_2 の翻訳等	…… ①
L1 :	JZ, ar, L3	…… ②
	文1 の 翻 訳	
L2 :	J , , L4	…… ③
L3 :	文2 の 翻 訳	
L4 :		…… ④

となる。翻訳される順は **2.4.** でも述べたように EX_1, EX_2, (rel.), 文1, (then), 文2, (if) の順に行なわれる。

① の翻訳は，(rel.) が A 型で整数型の時

L, ar, EX_1

S, ar, EX_2

で，実数型の時は第 2 命令コード S を SF と変えるだけでよく，以下 B, C, D 型も同様であるので実数型は省略する。(rel.) が B 型の時は

ar で EX_1 の 翻 訳

S, ar, EX_2

となる。(rel.) が C 型の時は

ar で EX_2 の翻訳

S, ar, EX_1

CHG, ar, 0

となる。(rel.) が D 型の時は

ar で EX_1 の翻訳

ar+1 で EX_2 の翻訳

S, ar, ar+1

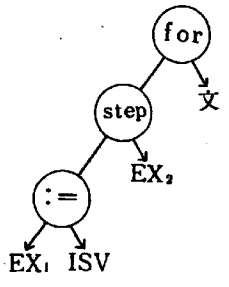
となる。以上のいずれかに (rel.) の翻訳では ② が加わる。L1 番地の命令コードは比較作用素に応じたものを入れ，番地部の L3 はこの段階では未定なので，L1 を push down しておく。

(then) の翻訳は ③ の 1 語である。L3 番地の値は L2+1 で確定するので，スタックから L1 を pop up してとり出し，L1 番地の番地部に L3 を入れる。L2 番地の番地部の L4 は未定なので，L2 を push down しておく。

(if) の翻訳ではオブジェクトコードはないが未定だった L4 が確定するので，スタックから L2 を pop up してとり出し，L2 番地の番地部に L4 を入れて条件文のトリーは翻訳し終る。

〈繰返文〉

繰返文のトリーは一般に



の形をしている。本 ALGŌL コンパイラの繰返文の翻訳は

```
        L, ar, EX1      }
        T, ar, ISV      }  ..... ①
   L1 : L, ar, ISV      }
        S, ar, EX2      }  ..... ②
   L2 : JP, ar, L3      }
        [文 の 翻 訳]
        L, ar, ISV      }
        A, ar, DCŌNE    }  ..... ③
        T, ar, ISV      }
        J , , L1        }
   L3 :
```

となる。翻訳される順は **2.4.** でも述べたように EX_1, (:=), EX_2, (step), 文, (for) の順に翻訳される。

① の翻訳は (:=) が A 型の場合で，(:=) が C 型の場合は

```
        [ar で EX1 の翻訳]
        T, ar, ISV
```

となり，格納命令だけとなる。③ で L1 番地を引用するので L1 を push down する。また ② と ③ で ISV を使用するので，ISV も push down してスタックに入れておく。

② の翻訳は (step) が B 型の場合で，(step) が D 型の場合は EX_2 が (step) に入る前に翻訳される。この時 (step) が D 型であるので EX_2 は ar+1 の AR で演算されている。ar+1 を使用したオブジェクトコードを出力する。すなわち

```
   L1 : [ar+1 で EX2 の翻訳]
        S, ar+1, ISV
   L2 : JNP, ar+1, L3
```

となる。スタックの一番上を read すれば ISV をとり出せる。③の処理が終らないと L3 番地が確定しないので，L2 を push down しておく。

③の翻訳ではスタックから L2, ISV, L1 を pop up してとり出し，ISV, L1 を使用してオブジェクトコードを出力し，L2 番地の番地部に L3 を入れて繰返文のトリーは翻訳し終る。

〈代人文〉

代入文のトリーは

の形をしている。本 ALGÖL コンパイラの代入文の翻訳は，(:=) が A 型の時は

L, ar, EX

T, ar, V

となる。(:=) が B 型の時は

ar で EX の翻訳

T, ar, V

となる。(:=) が C 型の時は

ar で V の添字の翻訳

L, ar+1, EX

T, ar+1, ar*

となる。(:=) が D 型の時は

ar で EX の翻訳

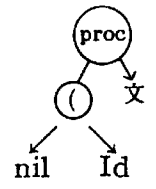
ar+1 で V の添字の翻訳

T, ar, ar+1*

となる。また型変換が必要な時は，T 命令の前に実数化の時 FL 命令を，整数化の時 FIX 命令をつけ加える。

〈手続きの宣言〉

手続きの宣言のトリーは



の形をしている。本 ALGŌL コンパイラの手続きの宣言の翻訳は

L1 : TPSC, , L2 …… ①

文 の 翻 訳

L2 : J , , 0 …… ②

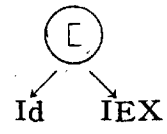
となる。翻訳される順は (, 文, proc である。

① の翻訳では L2 が未定のため L1 を push down する。そして L1 の値を手続名の宣言情報表の ADD パートに入れる。

② ではスタックから L1 を pop up してとり出し，L1 番地の番地部に L2 を入れる。L2 番地の番地部は TPSC 命令が実行されたら入る。

〈添字付き変数〉

1 次元の添字付き変数のトリーは



の形をしている。この 1 次元の添字付き変数に対して本 ALGŌL コンパイラのオブジェクトは

L, ar, IEX

AL, ar, AI

L, ar, ar*

である。また IEX（添字式）が単純変数，数以外の時，すなわち [が C 型のときは

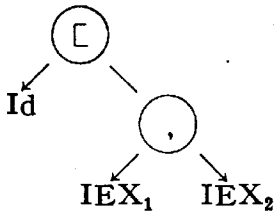
ar で IEX の 翻 訳

AL, ar, AI

L, ar, ar*

となる。AI は配列名の基準番地である。添字計算を AR で行なって，その AR の内容に基準番地を加えている。AR の絶対番地に間接番地指定をして AR に値をとり出している。

2次元の添字付き変数のトリーは



の形をしている。(,) が A 型の場合のオブジェクトコードは

L, ar, IEX_1
L1 : M, ar, MU
A, ar, IEX_2
} …… ①

AL, ar, AI
L, ar, ar*
} …… ②

である。①，②はそれぞれ(,)，([) の節の翻訳である。

(,) が B 型の場合は

ar で IEX_1 の翻訳

L1 : M, ar, MU

A, ar, IEX_2

となる。(,) が C 型の場合は

ar で IEX_2 の翻訳

L, ar+1, IEX_1

L1 : M, ar+1, MU

A, ar, ar+1

となる。(,) が D 型の場合は

ar で IEX_1 の翻訳

ar+1 で IEX_2 の翻訳

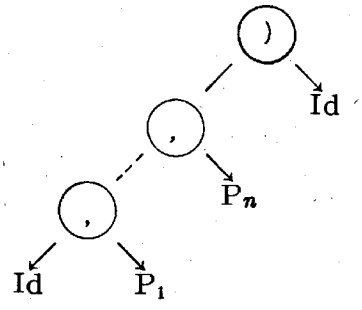
L1 : M, ar, MU

A, ar, ar+1

となる。① では MU が未定なので，L1 を push down しておく。② ではスタックから L1 を pop up してとり出し，L1 番地の番地部に宣言情報表の M パートの値を入れる。AI は配列名の宣言情報表の ADD パートの値（基準番地）である。

〈手続文と関数呼出〉

一般にパラメータがある場合の手続文と関数呼出のトリーは



の形をしている。本 ALGŌL コンパイラのオブジェクトコードは

P_1 の翻訳
.....
.....
P_n の翻訳

} ..... ①

J , , Id ..... ②

となり，はじめに P_1～P_n のパラメータに対する翻訳 ① があり，次に) の翻訳 ② がある。

パラメータの翻訳では節 , の型と種類に応じたオブジェクトコードが出力される。P_1～P_n が葉のとき , は A 型か B 型である。このとき名前替えのオブジェクトコードは

LL, ar, P_i

T, ar, FP_i

となり，値とりの時は

L, ar, P_i

T, ar, FP_i

となる。P_1～P_n がポインタの時は実パラメータは算術式か添字付き変数である。このときのオブジェクトコードは

ar で式又は添字付き変数の翻訳

T, ar, FP_i

となる。このとき添字付き変数で名前替えの時は，ロケーションカウンタを1戻して L 命令の上に T 命令を重ねる必要がある。

パラメータが配列名のみのときは

LL, ar, P_i

T, ar, FP_i

L, ar, PM

T, ar, FM

と翻訳する。P_i は実パラメータの番地，FP_i は仮パラメータの番地，PM は実パラメータ配列名の宣言情報表の M パート，FM は仮パラメータの宣言情報表の M パートの値である。

② は手続文の場合で，関数呼出の時は J 命令のあとに

L, ar, $37700_{(8)}$

の命令がつけ加えられる。

パラメータのない手続文と関数呼出の翻訳は ② のみとなる。

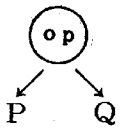
〈ブロックと文〉

ブロックで作られる節 (blk) と (文) が D 型以外のときは non effect 処理とし，D 型の時は ARC が増減しないように control するだけである。

〈算術式〉

(↑) だけは関数呼出扱いとし，他の節は節に応じた命令が出力される。整数型のときは固定小数点命令が，実数型のときは浮動小数点命令が出力され

る。

A 型の節 

のオブジェクトは

L, ar, P

op, ar, Q

となる。B 型のときは

op, ar, Q

とするだけでよく，C 型の ⊗ と ⊕ については

op, ar, P

とすればよい。C 型の ⊖ は

op, ar, P

CHG, ar, 0

とする。C 型の ⊘ は 1 つ上の AR で演算してから AR に入れる命令を出力する。すなわち

L, ar+1, P

op, ar+1, ar

L, ar, ar+1

とする。

D 型のときは番地部が ar+1 であるオブジェクトになる。それで

op, ar, ar+1

とすればよい。

4. 実行ルーチン

4.1. 巾乗・関数

巾乗は関数と同様に組み込みサブルーチンである。演算は精度の問題もあるので倍精度で行なっている。標準関数のうち平方根だけはニュートン法で求め，他は多項式近似で求める。多項式の係数は主に参考文献[7]によっている。関数・巾乗の演算結果は AR0 に入っている。

4.2. 入出力変換

入出力変換は JIS の水準30で用意すべきもののほかに，inoctal と outoctal を用意している。inoctal と outoctal は本 ALGŌL コンパイラで8進数の入出力のために特別に設けたものである。また，outlinefeed, inlinefeed, outstring は1.3.の入出力管理のサブルーチンをそのまま使用している。

ininteger は入力用の data buffer から数を1つ読みとり，整数のときはそのまま実パラメータに入れ，実数のときは整数化して実パラメータに入れる。

inreal は入力用の data buffer から数を1つ読みとり，実数のときはそのまま入れ，整数のときは実数化して実パラメータに入れる。

outinteger は実パラメータの値を整数型のフォーマットに編集し，出力 buffer に詰める。

outreal は実パラメータの値を実数型のフォーマットに編集し，出力 buffer に詰める。

実際に出力 buffer の内容が LP に印字されるのは，outlinefeed のサブルーチンを実行したときであるが，出力 buffer が120字になった時は変換ルーチンの中で自動的に outlinefeed のサブルーチンを実行している。

(1975. 5. 31)

参 考 文 献

[1] 日本規格協会；情報処理 (JIS ハンドブック)，1971.

[2] 森口繁一；ALGŌL 入門，日科技連出版社，1970.

[3] 富士通 K.K.；FACŌM 230-60, ALGŌL 文法編 (SP-051-3-4)，1970.

[4] 沖電気工業 K.K.；ŌKITAC-5090H インストラクションマニュアル，1964.

[5] 岸田孝一；システム・プログラム入門，日本経営出版会，1970.

[6] 田中 一；コンパイラ，森北出版，1971.

[7] 一松 信；近似式，竹内書店，1963.

[8] 広瀬，中田，筧，佐久間，島内；「コンパイラのうちとそと」，*bit*, Vol. 3,

No. 1～No. 12, 共立出版 (1971).

[9] 穂鷹良介；「Automatic Coding について Ⅲ (1) および (2) ——CŌBŌL Compiler——」, 商学討究, 第 18 巻 1, 2 号, 1967.

[10] 穂鷹良介；「Automatic Coding について Ⅳ (1), (2), (3) および (4) ——FŌRTRAN Compiler——」, 商学討究, 第 19 巻 1, 2, 4 号, 第 20 巻 1 号, 1968～1969.

[11] 稲田信幸；「逆アセンブラ」, 商学討究, 第 25 巻第 1, 2 合併号, 1974. 10.

附　　録

本 ALGŌL コンパイラの symbolic code

一般形　[label:] operation, [register], [operand] [*]

例　　L1 : L, ar, M　M の内容を ar にもってくる.

　　T, ar, ar'*　ar' の表わす番地に ar の内容を格納する.

L	: load	J	: unconditional jump
T	: store	JZ	: jump if zero
A	: add	JNZ	: jump if non-zero
S	: subtract	JP	: jump if positive
M	: multiply	JNP	: jump if non-positive
D	: divide	JN	: jump if negative
AL	: add literal	JNN	: jump if non-negative
LL	: load literal	AF	: add floating
CHG	: change sign	SF	: subtract floating
FIX	: fix	MF	: multiply floating
FL	: float	DF	: divide floating
TPSC	: store previous SCC		