したがって，自己双対関数全体の作る ν 次元表現では同一の型に属する関数がそれぞれ1個ずつの恒等表現を含むから全体で型の個数に等しいだけの恒等表現を含むその個数 P_n は単純指標の直交関係[7]から

$$P_n = \frac{1}{2^n n!} \sum_C n_C \chi_C$$

と求まる．ここで C は O_n の共役類，n_C は類 C 中の元の個数，χ_C は ν 次元表現の類 C に対する指標で，この場合，定義から C に属する O_n の元に対して不変に保たれる自己双対関数の個数に等しい．

参考文献

1) D. Slepian: On the number of symmetry types of Boolean function of n variables, Can. J. Math. **5**, No. 2, p. 185 (1953)
2) B. Elspas: Self-complementary symmetry types of Boolean functions, T.I.R.E. **EC-9**, No. 2, p. 264 (1960)
3) 高橋秀俊：計算機械，岩波講座，現代応用数学 B 14-aII，岩波書店，東京，(1958)
4) 喜安善市：ディジタル回路の数学，電子通信工学講座 10，3 E-1，共立出版，東京，(1960)
5) 室賀三郎，戸田巌：多数決素子の理論，信学誌 **43**，10，p. 1071 (1960)
6) F.D. Murnagham: The theory of group representations, p. 94, Baltimore (1938)
7) H. Boerner: Darstellungen von Gruppen, s. 71 (1955)

ALGOL Compiler の作成について*

竹　中　　靖**

まえがき

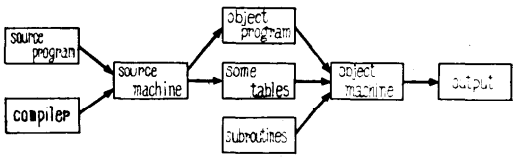
ここに述べるオートコーディング・システムはコンパイラー方式のものであって，ALGOL 60[1,2] で書かれたプログラムを処理してオブゼクト・プログラムを作成する働きを有し，コンパイラーは三つのルーチンから成り立っている[3]．

compier
- routine 1
- routine 2
- routine 3
  - phase 1
  - phase 2

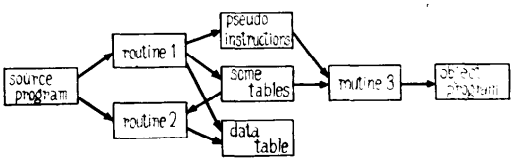
ルーチン1は H. Kanner の algebraic translator[4] をもとにして作成されたもので，Kanner の translator が算術演算のみを取り扱うのに対して，論理演算や計算の反復実行をも取り扱うことができる．また，演算子に対するレベル解析は，Kanner の 4-level hierarchy に対し，関係演算子や論理演算子を含めた 10-level hierarchy によって行なわれる．

ルーチン2は，ソース・プログラム中のシンボリック・パラメータに対して与えられたデータを，パラメータと正しく対応させる役割をする．

ルーチン3は，ルーチン1で作成された 3-アドレス方式の擬命令を必要な機械語に変換し，併わせて命令の節約を行なって所要のオブゼクト・プログラムを作成するルーチンであり，操作は二つの相に分離される．



第 1 図



第 2 図

なお，ルーチン1では，ルーチン2および3の操作のために，後述する各種のテーブルを作成する．

---

* An ALGOL Compiler, by Yasushi Takenaka (Joh Laboratory, Faculty of Engineering University of Osaka)

** 大阪大学工学部

このシステムを図示すれば**第1図**のようになり，また，コンパイラーの操作の流れは**第2図**に示すとおりである．

1. Source program に対する制限

初めにソース・プログラム作成についての制限を述べて，コンパイラーの処理能力を明らかにしておこう．

このコンパイラーは，機能上 ALGOL 60 の文法にしたがって書かれたプログラムのすべてを処理することはできない．すなわち，次に列記するような規約と制限が ALGOL の文法に付加される．

（1） identifier の長さは最大限3文字とする．

（2） identifier が連続して現われるときには，それらを分離記号␣で分離する．

（3） 関数指定子の助変数境界符号は comma だけを使用する．

（4） 型および配列の宣言は，一つの identifier に対して1回限りとし，型宣言のうち integer は配列の添字を示す identifier に対してのみ使用する．

（5） 配列は3次元までを許す．その添字は初期値が 0，公差が1の有限の数列を示すものでなければならず，また，添字が identifier のときは，その長さは1文字で必要に応じて1～4の数値の加減算のみ可能である．また，同じ添字部の中で，数と identifier を同時に使用することはできない．

（6） for clause につづく statement は，必らず statement 括弧でくくる．

（7） 条件付 statement は，必らず statement 括弧でくくる．

（8） ソース・プログラム中の演算開始場所は，必らず特別のラベル・BOP で明示せねばならない．

（9） while 要素を除いて，for list の中には算術表現を作用しない．また，step-until 要素，while 要素などを一つの for clause の中で同時に作用してはいけない．

以上の他に，ALGOL の文法から少しはずれるものとして，次の規約を設ける．

(10) procedure statement における name replacement と value assignment は区別して取り扱わない．

2. Routine 1 の操作

2・1 レベルの解析

標準関数，算術演算子，関係演算子，論理演算子に対して，優先規則に従って**第1表**に示すとおりの hierarchy number を定める．また，括弧によるレベルの変化を示すために括弧に対するレベルとして parenthesis level を定義する．

第 1 表

operator	hierarchy number
standard function	10
↑	9
×, ／	8
+, −	7
<, ≦, =, ≧, >, ≠	6
¬	5
∧	4
∨	3
⊃	2
≡	1

parenthesis level は，左括弧の直前にある演算子の実レベルで表わされ．したがって初期値は0である，

演算子の実レベルは，以上の二つの和で与えられる．すなわち

(actual operator level)＝

(hierarchy number)＋(parenthesis level)

parenthesis level は，対応する右括弧がくれば除去されて，それ以前の状態にもどる．

このように定められた演算子の実レベル，L_n を，相隣れる二つの演算子について比較し

$L_n \geqq L_{n+1}$ ($n=1, 2, 3, \cdots\cdots$)

なる関係が満たされれば，L_n に対する演算が可能であると判定し後述する3アドレス方式の擬命令を作成する．ただし，等号が成立する場合で演算子がいずれも関数のときは，L_{n+1} を演算可能と判定する．

2・2 procedure heading の処理

まず，関数指定子を処理して procedure table を作成する．その形は

××××××	×	××××
identifier	n	0 0 0 0

$n=1$ のときは，identifier が procedure identifier であり，形式助変数に対しては $n=0$ である．下4桁はルーチン3の操作のために0のままで保存される．procedure identifier に対するものは，そのままの形で後述する label table の中にも入れられる．

型宣言および配列宣言は，処理されて identifier table 1となる．その形は

××××××	××	××××
parameter	△	D_p

△は，助変数に対して宣言された宣言子のコードの和であって助変数の型および性質を示す．Dp は，助変数に対するデータの貯えられるべき番地である．宣言子のコードは**第2表**に示されているとおりである．

第 2 表

	code
Boolean	1
integer	2
real	3
own	4
value	8
array	50

配列宣言に対しては，この他に次の型の array table 1 が作成されるが，これは配列の境界対表を変形したものと考えればよい．その形は

$$\underbrace{\times\times\times\times}_{D_i}\ \underbrace{\times}_{d}\ \underbrace{\times\times}_{C_1}\ \underbrace{\times\times}_{C_2}\ \underbrace{\times\times}_{C_3}$$

D_i は，この table を添字表として有する identifier の identifier table 1 で占める番地を示し，d はその次元，C_1, C_2, C_3 はそれぞれ第 1, 2, 3 次元の成分の数を示す．この場合，identifier table 1 の Dp は，配列の第 1 成分に対する番地を示す．

2・3 Procedure body の処理

ここでは，各種の演算を規定する statement の処理について述べる．このコンパイラーでは，statement はすべて 3 アドレス方式のプログラムに変換され，それをもとにしてオブゼクト・プログラムが作成される．この段階では，処理のための補助テーブルとして次の 3 種類のテーブルを作成する．すなわち，statement 内で使用されているすべての助変数に対して

$$\underbrace{\times\times\times\times\times\times}_{\text{parameter}}\quad\underbrace{\times\times\times\times\times}_{D_A}$$

なる形の identifier table 2 が作成される．D_A は通常は 0 であるが，助変数が添字つきの場合には，対応する array table 2 の番地が挿入される．array table 2 は，助変数の添字を貯えるためのもので，添字が数値であるか表現であるかによって次のいずれかの形をとる．

sign	$\underbrace{\times\times\times}$	$\underbrace{\times\times\times\times}$	$\underbrace{\times\times\times\times}$
数値：＋	0 ζ_3	0 0 ζ_2	0 0 ζ_1
表現：－	ζ λ_3	0 ζ λ_2	0 ζ λ_1
	第3次元	第2次元	第1次元

ζ は 1～4 の加減算を示す数であり，ζ が 0,1,2,3,4 のいずれかのときは添字は $\lambda_i+\zeta$ ($i=1, 2, 3$)．ζ が 6,7,8,9 のいずれかのときは $\lambda_i-(10-\zeta)$ である．

ラベルに対して作成されるテーブルは次に示す形をとり，一部分または大部分が procedure heading の処理と共に作成される．

$$\underbrace{\times\times\times\times\times\times}_{\text{identifier}}\ \underbrace{\times}_{n}\ \underbrace{\times\times\times\times}_{0}$$

$n=0$ のときは identifier が label，$n=1$ のときは procedure identifier，$n=2$ のときは switch 変数である．procedure table と同様，下 4 桁はルーチン 3 の処理のときに使用される．

以上のテーブルを作成すると同時に，statement の中の助変数およびラベルは，すべてテーブルの番地で置き換えられ，これらの番地によって 3 アドレス方式の擬命令が作成されるが，命令の形は処理すべき演算の種類によって種々の様式を取る．

2・3・1 assignment statement

assignment statement はレベルの解析によって判定された演算順序に従って擬命令に変換されるが，命令の形は

被演算子が 2 個のときは

(operand, operator, operand, result)

被演算子が 1 個のときは

(ϕ, operator, operand, result)

:＝に対しては

(ϕ, ϕ, right part, left part)

の 3 種類がある．たとえば

$a:=b+(-C)+\sin(d/(b\uparrow e))+f$;

に対しては，

1. (ϕ, −, C*, T_1)
2. (b*, +, T_1, T_1)
3. (b*, ↑, e*, T_2)
4. (d*, ／, T_2, T_2)
5. (ϕ, sin, T_2, T_2)
6. (T_1, +, T_2, T_1)
7. (T_1, +, f*, T_1)
8. (ϕ, ϕ, T_1, a*)

の八つの擬命令が作成される．ここに，＊印は，その identifier に対応する identifier table 2 の番地を，また T_n は，演算の中間結果を記憶する一時記憶番地を示している．

2・3・2 go to statement

go to D;

に対しては

$(\phi,$ go to, $\phi,$ $D^*)$

ただし，D^* はDに対する label table の番地である．この場合，D が未出ラベルのときは lable table の追加が行なわれる．

2・3・3 label

(ϕ, ϕ, ϕ, L^*)

の形を取る．記号その他については go to statement と同様である．

2・3・4 statement bracket

begin は，読み込みの直後に

$(\phi,$ begin, $\phi, \phi)$

end は，その前の statement が処理された後

$(\phi,$ end, $\phi, \phi)$

に，それぞれ変換される．

2・3・5 for statement

このコンパイラーでは，次の三つの型の for statement を処理することができる．

（1） for $V:=a_0, a_1, \cdots\cdots, a_n$ do begin $\Sigma; \Sigma;$ $\cdots\cdots\Sigma$ end

（2） for $V:=a$ step b until c do begin $\Sigma;$ $\Sigma;\cdots\cdots\Sigma$ end

（3） for $V:=E$ while B do begin $\Sigma; \Sigma;\cdots\cdots$ Σ end

ここで，E, B, Σ はそれぞれ算術表現，論理表現，および statement を表わす．

第1型の for clause は，処理されて

$(\phi,$ for, $\phi, \phi)$

(ϕ, ϕ, a_0, V)

(ϕ, ϕ, a_1, V)

.....

(ϕ, ϕ, a_n, V)

$(\phi,$ do, $\phi, \phi)$

第2型は

$(\phi,$ for, $\phi, \phi)$

(ϕ, ϕ, a, V)

$(\phi,$ step, $b, \phi)$

$(\phi,$ until, $c, \phi)$

$(\phi,$ do, $\phi, \phi)$

第3型は，

$(\phi,$ for, $\phi, \phi)$

.....

$(\phi,$ while, $\phi, \phi)$

.....

$(\phi,$ do, $\phi, \phi)$

ただし，$V:=E$ および B は，assignment statement におけると同じ処理が行なわれる．

for clause につづく statement の end は，for clause に対して定められた型数が同時に処理されて

$(\phi, \zeta$ end, $\phi, \phi)$

の形を取る．

型数 ζ: $\zeta = \begin{cases} 3 & (\text{第1型}) \\ 2 & (\text{第2型}) \\ 1 & (\text{第3型}) \end{cases}$

2・3・6 conditional statement

if clause の中の論理表現は assignment statement と同様の処理を受けるが，その最終形は，:=におけると同様

(ϕ, ϕ, T_t, B)

の形を取る．ここにBは最終の論理値を貯えるべき番地を示す．また，if, then, else の三つの演算子は，それぞれ

$(\phi,$ operator, $\phi, \phi)$

の形に変換される．したがって，たとえば

begin if B then Σ else Σ' end

は，変換を受けると

$(\phi,$ begin, $\phi, \phi)$

$(\phi,$ if, $\phi, \phi)$

B

$(\phi,$ then, $\phi, \phi)$

Σ

$(\phi,$ else, $\phi, \phi)$

Σ'

$(\phi,$ end, $\phi, \phi)$

の形になる．

2・3・7 procedur statement には，次の二つの形式があると考える．

（1） procedure heading において，形式的助変数が明示されているもの．

（2） procedure heading においては，その名前だけが与えられているもの．

前者の効用は ALGOL の文法から明らかである．後者は，ライブラリーとして貯えられているサブルーチンをできるだけ活用しようとの意図に基づいて設定したもので，ソース・プログラム中には演算を規定する statement が全く存在しないようなサブルーチンへのスイッチにも使用されるものである．

いずれの場合も，その名前は同じ形で procedure table と label table の中に存在するから，それらの番地を用いて次の形の擬命令を作成する．

$(I_p\ \phi,\ \phi,\ I_l)$

ただし，I_p……procedure table の番地

I_l……label table の番地

助変数表の処理は次のとおりである．

第1型式のときは，対応する形式助変数が procedure table の中に貯えられているから，その番地と実助変数の identifier table 2 の中での番地から

$(\phi,\ \phi,\ P_a,\ P_f)$

の形の擬命令を作成する．この場合，:= の処理の結果と区別するために，P_f の最高位に1を置く．

第2型式のときは，形式助変数は与えられないから

$(\phi,\ \phi,\ P_a,\ \phi)$

の形の擬命令を作成する．

ただし，P_a……実助変数の番地

P_f……形式助変数の番地

3. Routine 2 の操作

ルーチン2は，実助変数に対して与えられたデータを処理して，オブゼクト・プログラムでの演算に備える．データは，ソース・プログラムの最後に一括して与えられ，その形式は，statement:

parameter:=number;

からなる一種の複合 statement である．

単純変数に対して数値を対応させるときの操作は簡単である．すなわち

（1） 変数を identifier table 1 で探して，対応するデータ番地と，変数の型を知る．

（2） 数を変数の型に従って適当な形に変換する．

（3） データ番地に変換された数を貯える．

添字付変数のときは，操作1で得たデータ番地を添字と array table 1 によって適当に修正しなければならないが，その修正は，次に述べる配列に対するデータの処理に準ずる．

配列に対しては

begin $a[0:n_1,\ 0;n_2,\ 0;n_3]:=N_0,\ N_1,\cdots\cdots,\ N_k$ end

の形でデータを与えることができるが，各成分に対する番地の割り当ては，次のように定義する．

3・1 1次元配列のとき

$a[0],\ a[1],\ a[2],\cdots\cdots a[n_1]$

に対して

$D,\ D+1,\ D+2,\cdots\cdots D+n_1,$

3・2 2次元の配列のとき

$a[0,\ 0],\ a[0,\ 1]\cdots\cdots a[0,\ n_2]$

$a[1,\ 0],\ a[1,\ 1]\cdots\cdots a[1,\ n_2]$

……

$a[n_1,\ 0],\ a[n_1,\ 1]\cdots\cdots a[n_1,\ n_2]$

に対して

D　　$D+n_1+1\cdots\cdots D+(n_1+1)n_2$

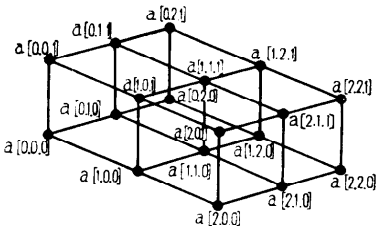
$D+1$　$D+n_1+2\cdots\cdots D+(n_1+1)n_2+1$

……

$D+n_1,\ D+2n_1+1\cdots\cdots D+(n_1+1)(n_2+1)-1$

3・3 3次元配列のとき

たとえば**第3図**の配列に対する番地は次のとおりである．



第 3 図

$a[0,0,0],\ a[0,1,0],\ a[0,2,0]$	D	$D+3$	$D+6$
$a[1,0,0],\ a[1,1,0],\ a[1,2,0]$	$D+1$	$D+4$	$D+7$
$a[2,0,0],\ a[2,1,0],\ a[2,2,0]$	$D+2$	$D+5$	$D+8$
$a[0,0,1],\ a[0,1,1],\ a[0,2,1]$	$D+9$	$D+12$	$D+15$
$a[1,0,1],\ a[1,1,1],\ a[1,2,1]$	$D+10$	$D+13$	$D+16$
$a[2,0,1],\ a[2,1,1],\ a[2,2,1]$	$D+11$	$D+14$	$D+17$

4. Routine 3 の操作[5]

ルーチン3は，ルーチン1で作成された擬命令を所要の機械語に変換する一種のコンバージョン・プログラムであるから，論義を進めるためにオブゼクト・マシンとして NEAC-2203[6] を採用する．

オブゼクト・プログラム作成の手順は次のとおりである．

（1） 擬命令の中では，助変数は identifier table 2 の番地で示されているから，その内容を identifier table 1 で探して対応するデータ番地を取り出す．このとき，添字付変数に対しては，取り出したデータ番地を添字によって修正するか，または，修正のための命令の列を作成する．

（2） 取り出されたデータ番地をもとにして，演算子に応じて適当な命令を作成する．

（3） 擬命令全体についての命令変換が終われば，ジャンプ命令の宛名部を決定し，必要な場所に no effect 10 0～Y を挿入する．

（4） 命令の節約を行なったあと，プログラムを整頓し，所要の形式でタイプ・アウトする．

1，2の手順は，ルーチン3の第1相で行なわれ 3,4 は第2相で行なわれる．タイプ・アウトの形は相対番地か絶対番地のいずれか適当な形式を指定することができるが，その指定は第1相の入口で行なわれる．

添字付変数に対するデータ番地の修正は，ルーチン2における場合に準ずる．すなわち，添字が数値で示されているときは，添字と array table 1によりルーチン2における番地の割り当ての規則に従って修正を行なう．添字が算術表現のときは，命令を作成する段階では修正を施すことができないから，修正のための命令の列を適当な場所に挿入する．挿入される命令は次のとおりである．

1次元の配列成分のとき

$V[a\pm\zeta]$　　(ζ=1, 2, 3, 4)

に対して，V, a のデータ番地をそれぞれ D_V, D_a とすれば

floating off	58	0	～	Y
clear add	30	0	D_a	Y
raise (lower)	28 (29)	0	ζ	Y
store	11	0	D_a	Y
	30	0	D_a	Y
memory to index	72	2	～	20
floating on	57	0	～	Y

2次元のときは

$V[a\pm\zeta, b\pm\zeta']$　　(ζ, ζ'=1, 2, 3, 4)

に対して D_V, D_a, D_b を1次元のときと同様に定義し，対応する array table 1 を

×　×　×　×　2　n_1　n_2　0

とすれば，第3表（a）に示す命令の列が挿入されるものである．

同様に，3次元のときは

$V[a\pm\zeta, b\pm\zeta', c\pm\zeta'']$　　(ζ, ζ', ζ''=1,2,3,4)

に対して D_V, D_a, D_b, D_c を定義し，対応する array table 1 を

×　×　×　×　3　n_1　n_2　n_3

とすれば，挿入される命令の列は第3表（b）になる．

いずれの場合も，ζ, ζ' または ζ'' が0ならば，対

第 3 表

（a）

	58	0	～ Y
	30	0	D_a Y
	28	0	ζ Y
	(29)		D_a Y
	11	0	D_b Y
uncond. jump	30	0	D_b Y
	28	0	ζ' Y
	(29)		
	11	0	D_b Y
擬命令数値	43	0	(××××)+1
			(××××)
			n_2 ZI
load MDR	30	0	D_a Y
plus mult.	14	0	D_b Y
left shipt	22	0	(××××)
	50	0	11 Y
	72	0	～ZO
	57	0	～Y

（b）

	58	0	～ Y
	30	0	D_a Y
	28	0	ζ Y
	(29)		
	11	0	D_a Y
	30	0	D_b Y
	28	0	ζ' Y
	(29)		
	11	0	D_b Y
	30	0	D_c Y
	28	0	ζ'' Y
	(29)		
	11	0	D_c Y
	43	0	(××××)+2
			(××××)
			n_2 ZI
			n_1 ZI
	30	0	D_b Y
	14	0	D_c Y
	22	0	(××××)+1
	50	0	11 Y
	14	0	～ZO
	30	0	D_a Y
	22	0	(××××)
	50	0	11 Y
	72	0	～ZO
	57	0	～Y

第 4 表

（a）

(ϕ, if, ϕ, ϕ)……{77 3 ～$Z\alpha$
B ……{ n ZI (B)
(ϕ, then, ϕ, ϕ)……{19 0 B^* Y
42 0 (end*)
Σ ……{適当な命令の列
(ϕ, end, ϕ, ϕ)……{(end*)

（b）

(ϕ, if, ϕ, ϕ)……{77 3 ～$Z\alpha$
B ……{ n ZI (B)
(ϕ, then, ϕ, ϕ)……{19 0 B^* Y
42 0 (else*)
Σ ……{適当な命令の列
(ϕ, end, ϕ, ϕ)……{43 0 (end*) (else*)
Σ' ……{適当な命令の列
(ϕ, end, ϕ, ϕ)……{(end*)

（c）

(ϕ, for, ϕ, ϕ)……{77 3 ～$Z\alpha$
10…0 V ZI
n ZI
(ϕ, ϕ, a_0, V)……{ a_0 ZI
(ϕ, ϕ, a_1, V)……{ a_1 ZI
……
(ϕ, ϕ, a_{n-1}, V)……{ a_{n-1} ZI
(ϕ, do, ϕ, ϕ)……{43 0 (endζ*)
(ϕ, begin, ϕ, ϕ)
Σ ……{適当な命令の列
(ϕ, ζend, ϕ, ϕ)……{43 0 ～$Z\alpha$ (end 1*)

（d）

(ϕ, for, ϕ, ϕ)……{77 3 ～$Z\alpha$
20…0 V ZI
(ϕ, ϕ, a, V)……{ a ZI
(ϕ, step, b, ϕ)……{ b ZI
(ϕ, until, c, ϕ)……{ c ZI
(ϕ, do, ϕ, ϕ)
(ϕ, begin, ϕ, ϕ)
Σ (c) と同じ
(ϕ, ζend, ϕ, ϕ)

$(\phi,\ \text{for},\ \phi,\ \phi)\cdots\cdots\begin{cases}77\ \ 3\ \ {}_{n}\sim Z\alpha\\ ZI\end{cases}$

$(V:=E)\quad\cdots\cdots\{\quad(V:=E)$

$(\phi,\ \text{while},\ \phi,\ \phi)\cdots\cdots\{\quad 99\cdots9\ \ ZI$

$B\quad\cdots\cdots\{\quad(B)$

$(\phi,\ \text{do},\ \phi,\ \phi)$
$(\phi,\ \text{begin},\ \phi,\ \phi)$
Σ
$(\phi,\ \zeta,\ \text{end},\ \phi,\ \phi)$　(c) と同じ

(e)

おのおのの soure statement:
(a) if B then Σ end
(b) if B then Σ else Σ' end
(c) for $V:=a_0, a_1,\cdots\cdots, a_n$ do begin Σ end
(d) for $V:=a$ step b until c do begin Σ end
(e) for $V:=E$ while B do begin Σ end
(a), (b), (e) の n は (B), $(V:=E)$ などの占める語数である.

応する〔 〕内の命令は作成されない.

3アドレス方式の擬命令を機械語に変換する操作は RUNCIBLE[7] と同様に逆処理の方法を採用している. 算術演算を表わす擬命令から作成される命令群は次のとおりである.

$(a^*,\ \pm,\ b^*,\ T)$
- clear add　30 0 D_aY
- add (subtract)　20 0 D_bY　(21)
- store　11 0 TY

$(a^*,\ \times,\ b^*,\ T)$
- load MDR　14 0 D_aY
- clear plus mult.　32 0 D_bY
- 11 0 TY

$(a,^*\ /,\ b,^*\ T)$
- 30 0 D_aY
- divide　17 0 D_bY
- store MQR　12 0 TY

$(\phi,\ \text{function},\ a,^*\ T)$
- set index　30 0 D_aY
- jump　77 3 $\sim Z\alpha$
- 11 0 TY

$(a^*,\ \uparrow,\ b^*,\ T_n)$
- 77 3 $\sim Z\alpha$
- D_a ZI
- D_b ZI
- 11 0 TY

$(\phi,\ \pm,\ a,^*\ T_n)$
- 30 0 D_aY
- (clear subtract)　(31)
- 11 0 T_nY

$(\phi,\ \phi,\ a^*,\ b,^*)$
- 30 0 D_aY
- 11 0 D_bY

番地の修正のための命令群を伴なったものに対してはインデックス指定部に2が入れられ, インデックス2の重複使用を避けるための若干個の命令が追加される.

$(\phi,\ \text{go to},\ \phi,\ D^*)$
および $(\phi,\ \phi,\ \phi,\ L^*)$
は, それぞれ

unconditionally jump　43 0 (D^*)
および　擬 命 令　00 0 (L^*)
に変換される.

if statement および for statement に関しては, このコンパイラーでは, 論理演算や反復演算のためのサブルーチンを用意し, そこへのスイッチを行なって所要の演算を行なう方式のプログラムが作成されるようになっているから, 変換の結果得られる命令群はサブルーチンへのスイッチ命令である[5]. ここでは, 擬命令と変換によって得られる命令群の対応を列記するにとどめる* (**第4表**参照).

procedure statement は, ルーチン1で

$(I_p,\ \phi,\ \phi,\ I_l)$
$(\phi,\ \phi,\ P_a{}^1,\ P_f{}')$
$(\phi,\ \phi,\ P_a{}^2,\ P_f{}^2)$
.....
$(\phi,\ \phi,\ P_a{}^n,\ P_f{}^n)$

あるいは

$(I_p,\ \phi,\ \phi,\ I_l)$
$(\phi,\ \phi,\ P_a{}^1,\ \phi)$
$(\phi,\ \phi,\ P_a{}^2,\ \phi)$
.....
$(\phi,\ \phi,\ P_a{}^n,\ \phi)$

の形に変換されているが, これらは次のようにして命令語に変換される.

前者は

(1) $(\phi,\ \phi,\ P_a{}^n,\ P_f{}^n)$ を
30 0 $D_{p_a{}^n}Y$
11 0 $\sim Z\alpha_n$
77 3 $\sim Z\alpha\cdots\cdots\alpha$ は (3) で決定に変換する.

(2) 残りの $(\phi,\ \phi,\ P_a{}^i,\ P_f{}^i)$ をすべて
30 0 $D_{p_a{}^i}Y$
11 0 $\sim Z\alpha_i$
に変換する. 1, 2において助変数が配列の名前のときは, 上に述べた命令の代りに

72 2 (××××)
(△△△△)　　……擬命令
71 2 $\sim Y$
30 2 $D_{p_a{}^i}Y$
11 2 $\sim Z\alpha_i$
76 2 (△△△△)
43 0 (××××)+1
10 0 $\sim Y$
(××××)　　……擬命令
n　　ZI　　……n は配列成分の数

* 操作の手順は, 文献 5) に詳しく述べられている.

を作成して，成分のすべてを形式助変数に対応させうるようにする．

（3）（I_p, ϕ, ϕ, I_l）によって（1）におけるαを決定する．

後者は次のように処理される．

（1）（ϕ, ϕ, $P_a{}^i$, ϕ）から

$D_{p_a}{}^i$ ZI

を作成し，その数nを知る．このとき助変数が配列を表わすならば，その成分の数を array table 1 で計算して

k $D_{p_a}{}^i$ ZI

を作成する（k は成分の数）．

（2） n ZI

を作成し，つづいて（I_p, ϕ, ϕ, I_l）によって

77 3 $\sim Z\alpha$

を作成する．

以上いずれの場合にも，決定されたαは procedure table の下4桁に格納してαの重複使用を避け，あるいは，同一の procedure body を指定する他の procedure statement の処理のために備える．

最後に，ルーチン3の第2相で行なわれるプログラムの整備と命令の節約について述べることにする．

第1相の操作によって，3アドレス擬命令はすべて機械語に変換されるが，その中にはまだ若干の擬命令が残存しており，残った擬命令に関連のある命令の宛名部も未決定のままであるから，擬命令の除去を行ない，同時に，関連する命令の宛名部を決定してやらなければならない．

まず，ラベルに対する擬命令とそれに対応するジャンプ命令の処理について述べる．この操作には label table の下4桁が使用され，擬命令とジャンプ命令のいずれが先に現われるかで操作が多少異なってくる．擬命令がはじめに現われるときは，その直前の命令が左右いずれであるかを判定して，左命令であれば

no effect 10 0 $\sim Y$

を命令の列に追加[6]した後，擬命令がオブゼクト・プログラムの中で占めるべき相対番地をテーブルの下4桁に記憶する．関連するジャンプ命令の宛名部には，記憶された相対番地をあてはめればよい．ジャンプ命令の出現の方が早いときは，宛名部を0にしたまま適当な場所に貯え，その番地を，テーブルの下4桁が0ならばそこへ，0でない（すなわち，関連するジャンプ命令がすでに1個以上出現している）ときは，新らしくテーブルの追加を行なってその下4桁に記憶する．このようにして放置された命令の宛名部は，擬命令が出現したときにその相対番地で満たされ，同時に，追加されたテーブルはすべて除去される．擬命令の処理とその後に出現するジャンプ命令の宛名部の決定は擬命令の出現が早い場合と全く同じである．

他の擬命令も同様の方法で除去されるが，関連する命令の出現が規則的なので操作は簡単である．

なお，この操作と同時に，命令の左右が判定されて必要な箇所に 10 0～Y が挿入され，併わせて命令の節約が行なわれるが，コンパイラーの機能上，次の三つの場合についてだけ節約が可能である．

（1） 11 0 ×××× Y
の次に 30 0 ×××× Y
がくるときは，これらを消去する（1語節約）．

（2） 11 0 ×××× Y
の次に 14 0 ×××× Y
がくるときは，これらを消去して
14 0 $\sim ZO$
を挿入する（1/2 語節約）．

（3） 12 0 ×××× Y
の次に 30 0 ×××× Y
がくるときは，これらを消去して
30 0 $\sim ZO$
を挿入する（1/2 語節約）．

以上の操作によって作成された命令は，絶対番地方式のオブゼクト・プログラムのためのものであって，相対番地方式のプログラムが指定されているときは，タイプ・アウトの直前に宛名部の変換が行なわれる．

あとがき

このコンパイラーは，NEAC-2203 基礎計算装置を標準にして作成されたものであるが，操作は，できるだけ一般性を持たせてあり，また，オブゼクト・マシンに従属する操作をルーチン3に集中してあるので，前記機種と同規模の計算機に適用することも容易である．

なお，紙面の都合上，操作の流れ図，前記機種に対して作成されたコンパイラー，およびオブゼクト・プログラムの例は掲載できないので省略することにする．

最後に，コンパイラー作成に際していろいろと御助言をいただいた大阪大学工学部城研究室の方々，コンパイラーのテストに御助力を下さった日本電気株式会社の皆様に深謝の意を表する．

参考文献

1) J.W. Backus, et al.: Report on the Algorithmic Language ALGOL 60., Comm. ACM 3, No. 5 (1960)
2) J.W. Backus, et al.: Report on the Algorithmic Language ALGOL 60., Numer. Math. Bd. 2, S. 106-136 (1960)
3) 竹中: 昭35年度情報処理学会予稿集, p. 15.
4) H. Kanner: An Algebraic Translator, Comm. ACM 2, No. 10 (1959)
5) 竹中: 第2回プログラミング・シンポジウム報告集, B-57 (1961)
6) 日本電気(株): NEAC-2203, 電子計算機説明書.
7) D.E. Knuth: RUNCIBLE, Algebraic Translation on a Limited Computer., Comm. ACM 2, No. 11 (1959)