資　　料

動的な表管理と COBOL コンパイラへのその応用*

首藤　勝** 小碇暉雄**

Abstract

Dynamic table control is a technique effectively applicable to compiler implementation. Size of each table or stack is not fixed, and a single common area is used for allocating many tables which grow through loading data elements. Removing explicit limitations on numbers and sizes of source program parameters yields effective increase of the "processing capability" of a compiler system.

Herein are described a representation of elementary functions of table handling and its application to the implementation of MELCOM 3100 Mk-II/III COBOL compiler.

1. まえがき

コンパイラでの処理過程で種々の情報を表，スタック，あるいは待ち行列の形（以下，単に表という）でもつことが多い．これらのおのおのに固定の領域を与えると，処理の過程でどれか1個の表の領域を使い切ったときそれ以上の処理の続行ができなくなり，システムの処理容量上の制限がきびしく現われる．

ここでは，個々の表の容量を固定せずデータの増加に従って表を拡大する方法をとり，1個の大きな領域を多くの表で共用することによって，メモリ領域の利用効率をあげ，システムの処理容量上の制限を緩和することを考える．この方法では表の大きさとメモリ割付けがプログラムの進行につれて変化するので，これを動的な表管理法と呼ぶ．ここで扱う個々の表としては索引型の表およびスタックとし，それらの要素は固定長または不定長であるとする．

動的な表管理法として大きさが増減する表に対して，増加と減少の双方に追随してメモリ割付けを変更する2方向性のものと，表の大きさの増大により必要となった場合にのみ割付けを変更する1方向性のものが考えられる．いずれも表割付け変動に伴ってデータ移動を要するため処理時間増大の欠点をもつが，両者の中では1方向性のほうがその程度は少ない．メモリ利用の有効さは2方向性のほうがよい．表位置を固定する方法では i 番目の表領域の大きさを A_i，i 番目の表の大きさを S_i として

$$S_i > A_i$$

が $i=1,2,\cdots,N$ のどれかについて成立したとき表領域溢れとなり処理不能となるのに対し，1方向性の動的管理では表領域全体の大きさを A，i 番目の表の最大の大きさを $S_{i\,max}$ として

$$\sum_{i=1}^{N}(S_{i\,max}) > A$$

のときに表領域溢れとなり，2方向性の場合は

$$\sum_{i=1}^{N} S_i > A$$

のときに溢れとなる．要素が減少する表が1個でもあれば，システムの処理容量上の制限は2方向性のほうがゆるくなる．ここでは1方向性の方法をとる．

2. 動的な表管理の体系

2.1 表および指標

表の個数を N とし，i 番目の表を T_i で表わす．$i=1,2,\cdots,N$ である．i 番目の表の j 番目の要素を x_j で表わす．i 番目の表の要素の個数を M_i とすると $j=1,2,\cdots,M_i$ である．i 番目の表に対し次の5個の指標をおく．

（1） 基点（表の先頭位置）：b_i

* Dynamic Table Control and its Application to COBOL Compilers, by Masaru Sudo and Teruo Koikari (Mitsubishi Electric Corporation)

** 三菱電機株式会社

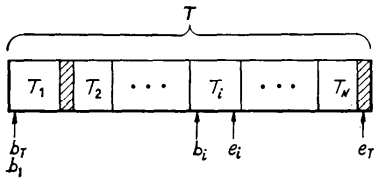


図 2.1　表領域中での表の割付け

(2)　端点（表の終端位置）: e_i

(3)　下点（不定長要素の表の最終要素の基点側に付加された区切り印の位置）: l_i

(4)　要素の大きさ: S_i（固定長要素の表で使用）

(5)　キ報の大きさ: k_i

2.2　表の割付け

(1)　表領域中での表の割付け

連続番地から成る表領域 $\boldsymbol{T}$ において表 T_i は i の順に低番地から高番地へ割り付けられる．$\boldsymbol{T}$ の基点を b_T，端点を e_T とする．基点 b_T は表 T_1 の基点 b_1 と一致するが，それぞれの表は必ずしも互に隣接せず，表の間あるいは T_N の上方に未使用領域を残すことがある（**図 2.1**）．

(2)　表中での表の要素の割付け

表 T_i の領域内で要素 x_j は j の順に低番地から高番地へ未使用領域を残すことなく割り付けられる．要素 x_1 の最下点は表 T_i の基点と一致し指標 b_i で示される．x_{M_i} の最上点は表 T_i の端点と一致し指標 e_i で示される．

要素 x_j の大きさが固定されている固定長要素の表は**図 2.2** のように構成される．

要素 x_j の大きさが固定されていない表を不定長要素の表と呼ぶ．これに2種あり，一方は各要素間に区切り情報を挿入したものである．この表は不定長要素から成るスタックとして使用され，その端点での要素取出し区切り情報を活用する．要素 x_j と x_{j+1} の間の区切り情報を m_j とすると，表 T_i 中で最上位置にある区切り情報は要素 x_{M_i} の下に付加された x_{M_i-1} であり，この位置が表 T_i の下点であって指標 l_i で示される〔**図 2.3 (a)**〕．もう一方は要素間に区切り

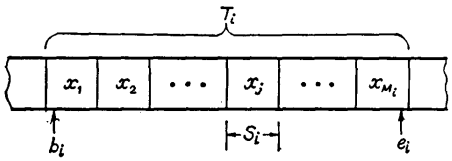
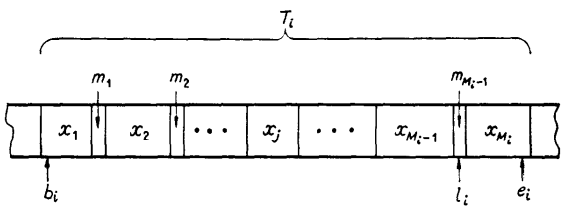
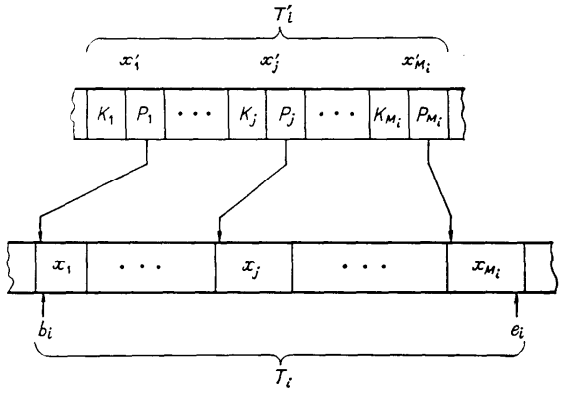


図 2.2　固定長要素の割付け



(a)　区切り情報を伴う不定長要素の割付け



(b)　補助表を用いる不定長要素の割付け

図 2.3

情報を持たず，要素の大きさは要素中の特定部分に存在する情報で示される．各要素の位置を知るために x_j の先頭位置を示す情報 p_j を含む補助表 T'_i を使用する．補助表の構成は種々に考えられるが，ここでは固定長のキイ K_j と指標 p_j から成る固定長要素をもつとする（キイを固定長としたのは適当な内部形に変えられていることを想定するからである）．補助表 T'_i は前述の固定長要素の表の一つであり，その割付けは上述の規定に従う〔**図 2.3(b)**〕．

(3)　指標の取扱い

指標 b_i，e_i，l_i，S_i および k_i は各表 T_i について1組あり，表領域中での各種の情報の位置を示すために使われる．これらの指標を表領域全体の分だけ集めて格納するために指標領域を設ける．

指標領域は $\boldsymbol{B}$, $\boldsymbol{E}$, $\boldsymbol{L}$, $\boldsymbol{S}$, および $\boldsymbol{K}$ から成り，それぞれ N 個の要素から構成される．それぞれの中では指標が対応する表 T_i の i 番号順に低番地から高番地へ割り付けられる．l_i，S_i，および k_i の中には空となるものもあるが，表番号 i から指標を引き出す過程を簡単にするために他の指標と同様の割付けを行なう．

(4)　表領域，指標領域，およびプログラム領域

表領域および指標領域は主メモリ中にあるとする．

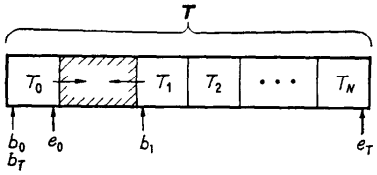


図 2.4 表 T_0 を設ける場合の割付け

コンパイラはいくつかのフェイズに分けて構成されているのが普通であり，表領域および指標領域に置かれる情報はいくつかのフェイズにまたがって保持されなければならないので，これらの領域はプログラム領域とは区別して割り付けられている必要がある．

2.3 表処理の初期条件および溢れの条件

(1) 初期条件

表領域の使用開始時点，たとえばあるプログラム単位のコンパイルの初めに，N 個の表に対して指標 b_i, e_i, l_i を全部 e_T に等しくする．これは，表への要素追加が進むにつれてメモリの高番地端から低番地の方向に表領域がひろがるような使い方を示している．指標 S_i および k_i は表の種類と構造に応じてあらかじめ定まっている．

(2) 溢れの条件

プログラム処理の進行に伴って表が拡大され，表領域の下端が低番地方向に伸びた結果，使用可能なメモリ領域を使い切ってしまったときに，そのプログラムの処理続行が不可能となる．これはいわゆる表のパンクであり，表領域と他の領域との交さの形で現われる．

表領域の下端の内側にもう1個の表 T_0 を置き，e_0 および l_0 の初期値を b_T とし，T_0 だけは要素追加によって高番地方向に伸びるような使い方をするならば，溢れの条件は T_0 と T_1 の交さ，すなわち $e_0 \geq b_1$ で生じる．この方法では表領域そのものは固定されている (図 2.4)．

3. 表処理の機能

3.1 端点での要素追加

表の端点側を拡大して新しく要素を追加する．表の端点の上方に充分な空隙がなければ，表を下方に移動して空隙を作り出さなければならない．この表移動は動的な表管理の特徴的な処理であるが，これがプログラム処理時間増大の欠点となる。小きざみにひんぴんと表移動を行なうと時間増大が著しくなるので，移動幅を1回分の必要分より大きくとり表移動の頻度を減らして時間増大を軽減する．端点での要素追加の機能は次のように表現できる．

(除去前)

(除去後)

(1) $e_i+S_i<b_{i+1}$ であれば (3) を行なう．

(2) $e_i+S_i \geq b_{i+1}$ であれば，T_1 から T_i までの表の全要素を下方に νS_i だけ移動し，それに伴い

$$\left.\begin{aligned} b_\lambda &\leftarrow b_\lambda - \nu S_i \\ e_\lambda &\leftarrow e_\lambda - \nu S_i \\ l_\lambda &\leftarrow l_\lambda - \nu S_i \end{aligned}\right\} \lambda = 1, 2, \cdots, i$$

によって指標を更新する．ν は移動幅を大きくとるための量で，正整数である．この過程で溢れの条件が成立すれば処理続行不能を指示する．

(3) e_i+1 から上の領域に大きさ S_i の新しい要素を入れ，それに伴って $e_i \leftarrow e_i+S_i$ によって指標を更新する．

不定長要素のスタックでの要素追加には **3.4** で述べる区切り設定をしたのち上記の方法を用いる．

3.2 表中での書込み，読出し

表の構造を変えることなくその中から情報を読み出し，また所定の位置に情報を書き込む操作は，動的な表管理とは本来無関係に必要な機能であるが，これらはこの体系では次のように行なわれる．

(1) 指標から実効番地を算出する．表中の要素位置は $b_i+(j-1)\times S_i$ で決まる．特に要素の中の語の相対位置 r を指定するときは $b_i+(j-1)\times S_i+r$ とする．

(2) 指定された位置の情報の出し入れを行なう．

3.3 端点での要素読出しおよび除去

スタックとして用いる表では端点にある要素の読出し，および除去が基本的な機能として必要である．

(1) 固定長要素の表の場合

端点要素の位置は e_i-S_i+1 により指定できる．また要素除去に伴う指標更新は $e_i \leftarrow e_i-S_i$ により行なう．

(2) 不定長要素の表の場合

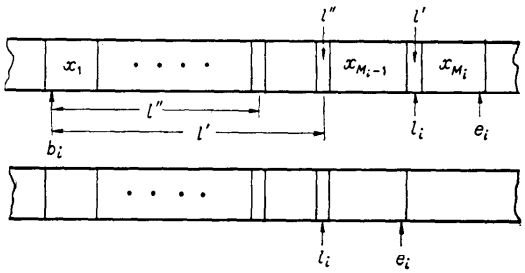


図 3.1 不定長要素の表の端点での要素除去

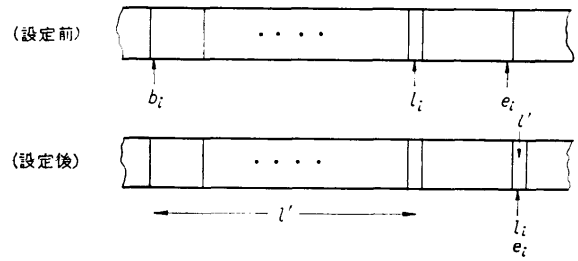


図 3.2　不定長要素の表の端点での区切り設定

スタックとして用いる不定長要素の表は区切り情報をもったものとし，指標 l_i をもとに区切り情報を見出して端点要素の位置を決める．要素除去に伴う指標更新は次のように行なう．

$e_i \leftarrow l_i - 1$

$l_i \leftarrow b_i + l'$

l' は端点要素下方の区切り情報の内容として保たれているもので，基点から端点の一つ手前の区切り情報までの距離を示している（**図 3.1**）．

3.4　端点での区切り設定

これは不定長要素のスタックにおける要素追加に先立って必要となる機能である．端点要素の上に区切り情報を積み，その内容として一つ下方の区切り情報の基点からの距離を入れる．この処理は次の操作で行なわれる．

$e_i \leftarrow e_i + 1$

$l_i' \leftarrow l_i - b_i$

l_i' は新しく設定された区切り情報の内容として入れる指標である（**図 3.2**）．この過程で表 T_i の大きさが増すから，これに先立って空隙の存在を確認し必要に応じて表を移動させる操作を挿入しなければならない．これは **3.1** で述べたのと同様の方法で行なう．

この区切り設定の後で区切りの上に要素を積み上げる操作が伴うのが通例であるが，それには最初に述べた要素追加機能を使う．コンパイラではソースプログラムの解析を進行させる過程で固定長要素の処理を基本としてスタックが作られ，ある段階に達するとそれまでに積み上げられた1群の要素を1個の不定長要素とみなす形の処理が多く，不定長の要素を直接に積上げ操作の対象とする必要はない．

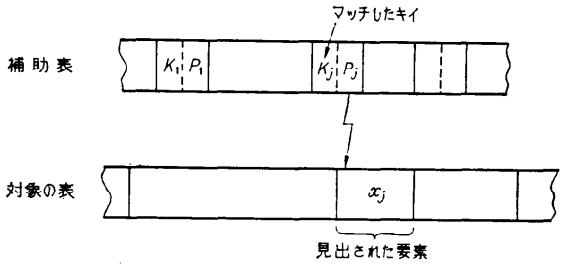


図 3.3　補助表を用いた表引き

3.5　表　引　き

(1)　固定長要素の表の場合

表引き処理は与えられたキイ情報（大きさ k_i）と一致するキイをもつ要素（大きさ $S_i, S_i > k_i$）の位置 j を見出すことである．要素中のキイの位置はあらかじめ決められている．表引き処理は複合的な処理であり，この処理系を実現する命令系を活用して組み立てる．表引き実施範囲を指定するために指標 b_i, l_i, e_i を使う．

(2)　不定長要素の表の場合

不定長要素の表の場合には固定長要素の補助表を使用する．まず上述の方法で補助表に対する表引きを行ない，そこから所要の指標 p_j を見出したうえ，不定長要素の表の要素を処理する（**図 3.3**）．

3.6　スタックを利用したサブルーチン・ジャンプ

サブルーチン・ジャンプの際に復帰番地をスタックに積み上げ，復帰に際しスタックを縮める方法をとるとサブルーチン・ネストの深さの制限を固定する必要がなくなる．サブルーチン・ジャンプに伴うパラメータの受渡しにもスタックを利用し，復帰に際して必要なだけ縮めることを可能にしておく．これらの操作のために復帰番地用と作業領域用の2種のスタックを使用するが，これらをあらかじめ特定番号の表の機能としておいて，サブルーチン・ジャンプ系のルーチン使用で自動的に働くようシステム設計をすると用法が簡単になる．この形のサブルーチン・ジャンプを使用すると回帰的な呼出しの処理が可能となる．

3.7　スタックと待ち行列の変換

コンパイラで中間情報をスタックに積みながら処理を進め，ある区分まで構成できたときにそれをオブジェクト・プログラム列の原形としてスタックから逆順で取り出す操作が有効になる．この操作はスタックの待ち行列への変換としてとらえられるが，これを表操作の一つの処理機能としてまとめておく．スタックとして使う表の中で指定された部分の要素の並び方を反転させる操作を設けることでこれが行なわれ，表からその部分を取り出すには通常の要素除去の機能を用いる．

表 4.1 MELCOM 3100 Mk-II/III COBOL コンパイラで使われる表

表の名前	*Si*	*ki*	*S/T*	*F/U*	機能・用途
MADLBL	1	1	*T*	*F*	前方参照 (Forward Reference) のアドレス生成のために用いる. READ 文の AT END 処理などで使用する.
AUX 1	1	1	*S*	*F* & *U*	condition の解析に使用. IF, PERFORM で現われる条件としてソース・プログラムで調べた情報を記憶する.
AUX 2	1	1			
AUX 3	1	1			
TEMP 1	1	1			算術演算の一時記憶に使用.
TEMP 2	1	1			複合条件の一時記憶に使用.
STCK 1	1	1			解析途上の一時記憶に使用. PERFORM～UNTIL～ PERFORM～AFTER～ IF 文 のように条件を伴う複雑な文のオブジェクト列の生成のために AUX 1～AUX 3 と組み合わせて使う.
STCK 2	1	1			
STCK 3	1	1			
MNT	1	1	*T*	*F*	DATA DIV. のファイル定義でレコード数チェックのための一時記憶として使用.
CORR 1	2	2	*S*	*F*	MOVE CORRESPONDING の送り出し側走査でレベル番号とデータ名を対にして記憶する. MOVE CORRESPONDIG の受入れ側走査で用いる.
CORR 2	2	2			
CTNT	1	1	*T*	*F*	紙テープコード変換表の ID 記憶用.
FNT	2	1	*T*	*F*	ENVIRONMENT DIV. の SELECT 句解析過程でファイル名と FDT ポインタが記憶され, DATA DIV. の FILE SECTION 解析過程で後者を NT ポインタに置き換える.
USET	3	2	*T*	*F*	USE 文の情報を記憶し, ラベル・ルーチン/エラー・ルーチンを作るファイルと出力時点をきめる.
SYM	2	2	*T*	*F*	SYMBOLIC KEY 項目名の記憶に使用.
ACT	2	2	*T*	*F*	ACTUAL KEY 項目名の記憶に使用.
SYSMAC	2	2	*T*	*F*	入出力処理で使用したシステム・マクロのシンボル名を登録し, オブジェクト列出力のときにマクロライブラリの ID 名に変換する.
REC	2	2	*T*	*F*	RECORD KEY 項目名の記憶に使用.
EXTNAM	2	2	*T*	*F*	プログラム・リンクのために使う外部名を記憶.
EXT	1	1	*S*	*F*	JSB ルーチンで使用, 復帰アドレスを記憶する. EXT ルーチンで 1 要素が解放される.
WOT	1	1	*S*	*F&U*	表管理系内での一時記憶として種々の処理のために使用.
RST	2	1	*T*	*F*	手続き名の表. 手続き名とその相対アドレスを積むフィールドとが対で記憶される.
GEST	4	1	*T*	*F*	データ階層の解析に使用.
GENROL	1	1	*S**	*F*	PROCEDURE DIV. のオブジェクト列を作って, 随時出力する. *逆転操作の対象となる.
COT	1	1	*T*	*U*	ラベル情報, 編集用 PICTURE 指定. 条件名の VALUE, PROCEDURE LITERAL を入れる.
NT	2	1	*T*	*F*	呼び名, ファイル名, レコード名, データ名, 条件名などの記憶. 要素は 2 語長で, 前半に名前の内部形, 後半に詳細情報または他の表へのポインタがはいる.
RDT	1	1	*T*	*U*	レコード名, データ名, プロセデュア・リテラルの詳細情報の記憶. NT を補助表として参照する.
FDT	13(Mk-II) /21(Mk-III)	1	*T*	*F*	ファイルに関する情報の記憶. 実行時用 FDT, およびマクロ命令生成のために使う.

4. COBOL コンパイラでの動的な表管理の応用

動的な表管理を活用して組立てられた実用コンパイラの例として, MELCOM 3100 Mk-II/III COBOL コンパイラをとり, そのあらましを述べる. このコンパイラはマイクロ・プログラム計算機のために作られたものであり, 機械の方式を活用するためにコンパイラ作成上いくつかの特色ある技術を使っている. 動的な表管理のためにマイクロ・プログラム・ルーチンを一揃い整えて, それらを呼ぶ機能をマクロ・アセンブラの命令として作りつけたこともその一つである.

4.1 COBOL コンパイラで用いる表の構造と用法

MELCOM 3100 Mk-II/III COBOL コンパイラでは合計 30 個の表を動的な表管理の対象としている．これらの表の構造と種類および用法を**表 4.1** に示す．

MELCOM 3100 はキャラクタ単位にデータを処理する機能を基礎としているが，実際にひんぴんと用いられる語長（18 ビット）単位の処理に対してマイクロ命令を持っているので，以下の説明では情報の長さの単位としてこの語長を用いる．

表管理の基礎となる指標 b_i, e_i および l_i は語単位で示されたメモリ番地に対応する．要素の大きさ S_i およびキイ情報の大きさ k_i も語を単位とする．S_i および k_i はそれぞれ5ビットおよび2ビットで表わしうるためこれらの指標を共に1語の中に入れ，制御語 C_i として用いる．全部の表についての C_i をたくわえるために領域 $\boldsymbol{C}$ を設けて $\boldsymbol{S}$ と $\boldsymbol{K}$ の代わりに使用する．したがって，指標領域は $\boldsymbol{B}$, $\boldsymbol{E}$, $\boldsymbol{L}$, および $\boldsymbol{C}$ で構成される．

COBOL で使われる名前，予約語，その他プログラムに現われる語は，リテラル，編集用 PICTURE 指定，ラベル情報を除いてコンパイラの字句解析段階で1語長の内部形に変換されている．そのためコンパイラの処理の対象となる情報はほとんど1語長以下の長さのものとして扱える．リテラルなど原形を保存する必要のあるものは不定長要素として扱い，補助表を伴う方法で処理を進める．

表 4.1 で F/U の欄は固定長要素か不定長要素かを示している．F & U は両方の扱いを受けることを示している．

WOT は表処理系における作業領域として種々の用途で使用される．これを用いる処理を簡潔にプログラムできるように，表指定を暗黙に行なう作業領域表専用のマイクロ・プログラム・ルーチンを設けてある．

EXT は復帰アドレス格納専用のスタックである．サブルーチン・ジャンプの機能をもつルーチンを使用すると自動的にこの表が処理の対象として選ばれる．

表処理機能には分岐，判断を伴うものがある．これらのための論理情報（真・偽）を記憶する1ビットの領域を論理アキュムレータとして特定の所に設置し，表処理の過程でセット，リセットし，またこの内容によってプログラム分岐を起こさせる．

4.2 表管理マイクロ・プログラム・ルーチン

MELCOM 3100 Mk-II/III COBOL コンパイラのために作られたマイクロ・プログラム・ルーチンの機能および操作内容，それを呼び出す命令形（これがマクロ・アセンブラ言語機能として使われる）を**表 4.2** に示す．ここではまったく補助的なルーチンいくつかを省略した．使用した記号は次のとおりである．

- i 表番号，$i=w$ は表 WOT を，$i=R$ は表 EXT を示し，E_i の代わりに E_w, E_R と書く．
- j 表中の要素番号．
- P_{ij} 表要素 x_{ij} を指す指標．
- $L.A.$ 論理アキュムレータ．
- (A) 番地 A の内容．
- $((A))$ 番地 A の内容が示す番地の内容．
- $(A)\rightarrow(B)$ 番地 A の内容を番地 B へ転送．
- $(A_1\sim A_2)\rightarrow(B_1\sim B_2)$ 番地 A_1 から A_2 までの連続領域の内容を番地 B_1 から B_2 までの連続領域へ転送．
- $(C_i)\Rightarrow i, j$ 制御語 C_i から i と j とを抽出．
- $P_{ij}\Leftarrow\langle i, j\rangle$ i と j とから指標 P_{ij} を合成．
- Search $T_i[A_1\sim A_2, k_i, S_i]$ 表 T_i を表引き操作．範囲は番地 A_1 から A_2 まで．
- Flip $T_i[A_1\sim A_2]$ 表 T_i の番地 A_1 から A_2 までの内容を順序逆転．
- Perform α ラベル α のついた手続きを行なう．

4.3 COBOL コンパイラでの表処理の例

上記の COBOL コンパイラでは随所に動的な表処理を用いているが，その効果的な使用例として PROCEDURE DIVISION の解析とオブジェクト・プログラム列の生成を受けもつフェイズ5での GENROL 表の操作をとりあげる．その概要は**図 4.1** に示すとおりである．

フェイズ5への入力はタイプ3と呼ばれる中間情報で，これはソース・プログラムに字句解析を施したのち1語長単位のコード化した内部形に変え，さらにそれ以後の解析に適するように整理したものである．

入力バッファにはいったタイプ3の情報は語単位で走査される．これはコンパイラが持っている文法原形表とのつき合わせをもとにした方法で行なわれる[1]．ここでは PROCEDURE DIVISION を構成する文の動詞を識別し，その文の処理ルーチンで解析を行なったのち再び次の動詞の識別に進む．この走査はタイプ3のエンド・マークが現われるまで続く．それぞれの文の解析ではタイプ3を1語ずつ読み進めながら構文の適正さを調べ，オブジェクト・プログラム原形の列を生成し GENROL 表に積み上げて行く．IF 文，READ

表 4.2 MELCOM 3100 Mk-II/III COBOL コンパイラの表管理マイクロ・プログラム・ルーチン機能一覧

命令形	機能	操作内容	備考
STRL w_1/w_2	端点に1要素 (S_i) 追加	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(E_i)\rightarrow(E)$; $S_i\rightarrow(S)$; $(B_{i+1})\rightarrow(B)$; perform α; $(w_2\sim w_2+S_i)\rightarrow((E_i)+1\sim E_i+1+S_i)$; $(E_i)+S_i\rightarrow(E_i)$; α: if$(E)+(S)<(B)$ then return else if $(B_1)-\nu\times(S)\leq(E_0)$ then O'FLOW EXIT else $((B_1)\sim(E))\rightarrow((B_1)-\nu\times(S)\sim(E)-\nu\times(S))$	*
STRR w_1/w_2	表中に1要素 (S_i) 書込み	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i\rightarrow(X)$; $(w_2\sim w_2+S_i)\rightarrow((X)\sim(X)+S_i)$;	*
STO $w_1/w_2/n_3$	表中に1語書込み	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow Si$; $(B_i)+j\times S_i+n_3\rightarrow(X)$; $(w_2)\rightarrow((X))$;	
LDR w_1/w_2	表中から1要素(S_i)読出し	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i\rightarrow(X)$; $((X)\sim(X)+S_i)\rightarrow(w_2\sim w_2+S_i)$;	*
LDO $w_1/w_2/w_3$	表中から1語読出し	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i+n_3\rightarrow(X)$; $((X))\rightarrow(w_2)$	
PNG w_1/w_2	ポインタ合成	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow S_i$; $j=\left[\frac{(E_i)+1-(B_i)}{S_i}\right]$; $P_{ij}\Leftarrow\langle i, j\rangle$; $P_ij\rightarrow(w_2)$;	
CPE w_1/w_2	ポインタからアドレス抽出	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i\rightarrow(w_2)$	
SRA $w_1/w_2/w_3/w_4$	表引き $(w_1)=i$ $(w_2)=$キイ $(w_4)=$ジャンプ先	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow k_i, S_i$; Search T_i $[(L_i)+1\sim(E_i); k_i, S_i]$ if match at j then $P_ij\Leftarrow\langle i, j\rangle$; $P_ij\rightarrow(w_3)$; goto w_4; else goto next;	
SRB $w_1/w_2/w_3/w_4$	表引き $(w_1)=P_{ij}$ $(w_2)=$キイ $(w_4)=$ジャンプ先	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow k_i, S_i$; $(B_i)+j\times S_i\rightarrow(X)$; Search $T_i[(X)\sim(E_i); k_i, S_i]$ if match at j_1 then $P_ij\Leftarrow\langle i, j_1\rangle$; $Pij\rightarrow(w_3)$; goto w_4; else goto next;	
RSV w_1	端点に区切り情報を設定	$(w_1)\Rightarrow i$; $(E_i)\rightarrow(E)$; $1\rightarrow(S)$; $(B_{i+1})\rightarrow(B)$; perform α; $(E_i)+1\rightarrow(E_i)$; $(L_i)-(B_i)\rightarrow((E_i))$; $(E_i)\rightarrow(L_i)$;	α は STRL 参照
REL w_1	端点から1要素と1区切り情報を除去	$(w_1)\Rightarrow i$; if $(L_i)=(B_i)$ then $(B_i)\rightarrow(E_i)$; else if $(L_i)>(B_i)$ then $(L_i)-1\rightarrow(E_i)$; $(B_i)+((L_i))\rightarrow(L_i)$;	
MOA w_1/w_2	端点から1語取出し	$(w_1)\Rightarrow i$; $((E_i))\rightarrow(w_2)$; $(E_i)-1\rightarrow(E_i)$; if $(E_i)=(L_i)$ then $0\rightarrow(L.A.)$ else $1\rightarrow(L.A.)$	
CAR w_1/w_2	端点1要素を他に移転 $(w_1)=i$ $(w_2)=i_1$	$(w_1)\Rightarrow i$; $(E_{i_1})\rightarrow(E)$; $(E_i)-(L_i)\rightarrow(S)$; $(B_{i_1+1})\rightarrow(B)$; perform α; $((L_i)+1\sim(E_i))\rightarrow((E_{i_1})+1\sim(E_i)-(Li)+1)$; perform REL;	α は STRL 参照
REF w_1	端点から1要素除去	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow S_i$; $(E_i)-S_i\rightarrow(E_i)$;	
REP w_1/w_2	端点から1要素除去	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow S_i$; $(E_i)-S_i\rightarrow(E_i)$; if $(E_i)=(B_i)$ then goto w_2 else goto next;	
FET w_1	端点に1語追加	$(E_w)\rightarrow(E)$; $1\rightarrow(S)$; $(B_{w+1})\rightarrow(B)$; perform α; $(E_w)+1\rightarrow(E_w)$; $(w_1)\rightarrow((E_w))$;	α は STRL 参照
ASP w_1	端点から1語取出し	$((E_w))\rightarrow(w_1)$; $(E_w)-1\rightarrow(E_w)$;	
ASK w_1	端点で1語読出し	$((E_w))\rightarrow(w_1)$;	
POW n_1	端点から n_1 語除去	$(E_w)-n_1\rightarrow(E_w)$;	
JSB	サブルーチン・ジャンプ	$(E_R)\rightarrow(E)$; $1\rightarrow(S)$; $(B_{R+1})\rightarrow(B)$; perform α; $(E_R)+1\rightarrow(E_R)$; 復帰アドレス$\rightarrow((E_R))$; goto w_1;	α は STRL 参照
EXN	復帰 $((E_R))=$復帰アドレス	$(E_R)\rightarrow(X)$; $(E_R)-1\rightarrow(E_R)$; goto (X);	
EXF	復帰 $((E_R))=$復帰アドレス	$0\rightarrow(L.A.)$; $(E_R)\rightarrow(X)$; $(E_R)-1\rightarrow(E_R)$; goto (X);	
EXT	復帰 $((E_R))=$復帰アドレス	$1\rightarrow(L.A.)$; $(E_R)\rightarrow(X)$; $(E_R)-1\rightarrow(E_R)$; goto (X);	
JAF w_1	論理分岐	if $(L.A.)=0$ then goto w_1;	
JAT w_1	論理分岐	if $(L.A.)=1$ then goto w_1;	
FLP	端点要素を反転	$(w_1)\Rightarrow i$; Flip $T_i[(L_i)+1\sim(E_i)]$	

* 特定の表 (RDT) については要素転送の過程で，パッキングまたはアンパッキングが行なわれる．

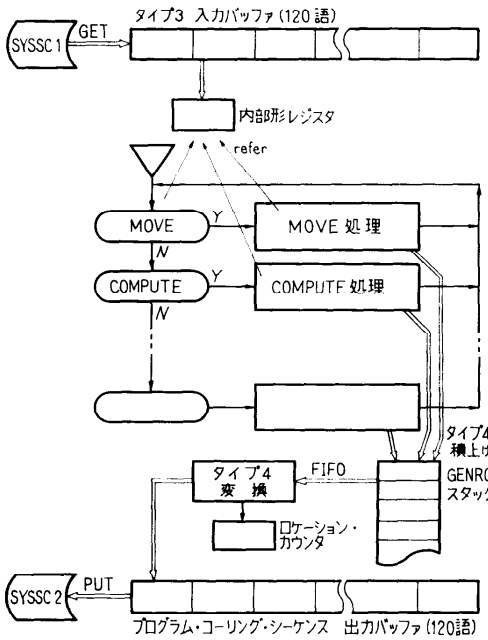


図 4.1 COBOL フェイズ5での PROCEDURE DIVISION の処理

文の AT END 指定，および算術文の SIZE ERROR 指定のように文の中に文を含む場合には，動詞を識別してその処理ルーチンへ飛ぶためのルーチンを回帰的に呼び出す．これは前述の JSB 命令の復帰番地をスタックする機能により実現される．またそれぞれの文の処理ルーチン中での複雑なサブルーチンのネストに対しても，JSB 命令と EXT 命令が有効に使われている．

GENROL 表にスタックされたオブジェクト原形列は，タイプ3の1語読出しルーチンでソース・プログラムの行(カード1枚)の区切りのマークを検出した時点に，FLP 命令による要素の逆転を施したのち取り出される．FLP 命令は GENROL の内容を FIFO (first-in first-out) で処理するために使用される．GENROL の内容は1語ずつ取り出され，オブジェクト・プログラムを構成する語が現われると相対番地を数えるロケーション・カウンタを1加算し，前方参照のフラグが現われると MADLBL に，パラグラフかセクションのコードが現われると，PST にそのときのロケーション・カウンタの値を入れる．こうして，次のフェイズで機械的に指標をそれが示す情報で置き換えればオブジェクト・プログラム列になるようにしたのち，タイプ4として出力される．

表 4.3 COBOL コンパイラにおける表の制限事項

制限項目	Mk-I COBOL コンパイラ	Mk-II/III COBOL コンパイラ
異なるデータ名の文字総数	24,000	24,000
異なるリテラルの文字総数	1,500	1,500
セクション名，パラグラフ名総数	200	300
データ名，FILLER，PROCEDURE DIV. 中の異なる値のコンスタントの個数，ENVIRONMENT DIV. の SPECIAL NAMES 中の名前の総和	700	—
ファイル数	10	—
ファイル名の2倍と FILE SECTION 中のレコード名との和	50	—
レベル番号チェックレベルの深さ	70	—
修飾語	6	—
ADD, SUBTRACT における加数，減数の個数	9	—
IF 命令の個数	200	—
GO TO P_1 P_2…P_n DEPENDING 命令の P_i の個数 n	30	—

4.4 動的な表管理の効果

動的な表管理のねらいは初めに述べたように，表領域の有効利用による処理容量制限の緩和にある．これの具体的効果として，MELCOM 3100 Mk-II/III COBOL コンパイラではソース・プログラムを書くうえで課せられる制限事項の多くを表面に固定的に出さなくすることができた．このコンパイラの前の版である Mk-I COBOL コンパイラからの制限事項の緩和の状況は**表 4.3** に示すとおりで，表中11項目のうち3項目を除いて動的管理の対象として固定的な制限がはずされた．

また，固定的な制限の場合は，それぞれの表ごとに余裕をとるが，動的管理の場合はその必要がなく，表領域をきりつめうるのでプログラム領域を大きくでき，その結果コンパイラを構成するフェイズ数の減少も可能となる．これはコンパイル時間の短縮につながり，動的な表管理の副産物的効果の一つとなる．

その他の効果としては，表の管理を統一的に行なえるため，表処理のプログラミングに伴う指標処理のわずらわしさを除けること，サブルーチンの回帰呼出しが可能となることがあげられる．

動的表管理の欠点としては処理時間の増大がある．これについては動的表管理の有無だけを取り出した比較は行なっていないが，中間言語としてアセンブラ語をとる間接コンパイル方式でかつ動的な表管理を用いていない Mk-I COBOL コンパイラから，直接コンパイル方式でかつ動的な表管理を採用した Mk-II/III COBOL コンパイラになってコンパイル速度が約4倍に改善された．これは動的な表管理のマイナス効果を含み，さらに Mk-II/III コンパイラではリンク・

エディト時間をも含めての比較であるので，実用的には動的な表管理の採用によって性能がおちたとはいえず，全体としての向上の方が著しい．Mk-III COBOL コンパイラによるコンパイル所要時間は，リンク・エディトを含めて平均毎分約150行となっている．

5. むすび

コンパイラ作成上の技術として有効な表の動的管理についてその体系的な説明を行ない，表処理の諸機能を実用にとおしたまとめ方で整理した．それの応用例として，中型計算機 MELCOM 3100 Mk-II/III COBOL コンパイラでの表の概要と，特にマイクロ・プログラム・ルーチンとして作成した表処理機能について述べた．このコンパイラは初版完成以来基本的な改造なく現在も第一線で稼動しているものである．

終りにあたり，このコンパイラ開発過程で特に本稿で触れた部分に関して作業を担当された三菱電機(株)鎌倉製作所の井上高志，阪田勇夫の両氏，および討論していただいた同，関本彰次氏に謝意を表したい．

参考文献

1) “MELCOM 1530 COBOL (コンパイラ編)”，日本電子工業振興協会，東京 (1966)．
2) “MELCOM COBOL 説明書 G1-TL01-00A 〈7004〉”，三菱電機 (株)，東京(1970)．
3) “MELCOM 3100 MARK-III COBOL 説明書 G4-TL01-00A 〈7001〉”，三菱電機(株)，東京 (1970)．
4) 情報処理学会COBOL研究会：“国産のCOBOL コンパイラ”，情報処理，Vol. 8, No. 3, pp. 140～144(1967)．
5) 三井大三郎，他：“MELCOM-3100ディスクオペレーティングシステム(1)”，三菱電機技報，Vol. 43, No. 11, pp. 1539～1545 (1969)．

(昭和46年10月1日受付)