資 料

自己拡張言語プロセッサ SELP*

北原紀之** 池田克夫*** 清野 武***

Abstract

A self-extensible language processor called SELP is described. SELP is an extended version of GPM by Strachy, and is intended to facilitate to define the abbreviations of statements for which GPM is not so suitable in practical use. SELP takes full advantage of GPM's extensibility of language facilities, making it much easier to define and descride macros using REPEAT facility and the "keys" which specify statement format.

SELP works as a symbol string manipulator, which analyzes input character stream punctuated by the control symbols, and produces output character stream perhaps to be processed by some compiler or assembler.

SELP itself is programmed in PL/I.

1. まえがき

プログラムの簡単化，入力時の非本質的な労力の削減などを目的とし，単純な構文法から柔軟性のある表現能力をもつ言語として利用者定義の拡張可能言語が研究されてきた[2)-7)]．多くのアセンブラに見られるマクロ機能をもったマクロアセンブラシステムに基づく言語などがそれである．

本稿は，原言語を個々の計算機システムに個有のアセンブリ言語に限定せずに，あらゆるプログラミング言語に対して，その言語に関する構文単位から新しい構文を定義して，原言語を拡張する機能をもった自己拡張型の汎用目的マクロプロセッサ SELP (Self-Extensible Language Processor) について述べる．

SELP は，Strachy による GPM[4)] を拡張したものである．GPM の特徴は，言語機能の拡張性にある．すなわち，マクロの定義が，DEF マクロの引数として与えられた記号列によって行なわれ，マクロコールにおいては，どの位置からでも多重のマクロコールが許されている．この特徴により，プログラミング言語の拡張に大きな可能性を与える．しかし，省略形を定義するためのマクロという観点からすると，共通部分の多いステートメントに対する省略形としてマクロ定義を行なう場合には，定義が長くなったり，複雑な形式をとるために，プログラミングの点でも処理効率の点でも GPM は実用的でない部分がある．

SELP は，主に省略形としてマクロ定義を行なうための機能に関する拡張を行ない，GPM の機能にフォーマットを定めるキーと繰り返し機能をつけ加えることによって，マクロ定義の仕方と記述の方法を容易にし，処理能力の向上をも図った．

マクロ機能というものが，記述すべき多くのステートメントをいかに効率よく処理し，簡潔に書けるようにするかについて考慮されるべきものとすれば，この機能によって，SELP は，さらに実用的なシステムになったと考える．

2. 記号列と制御記号

SELP は，あらゆるプログラミング言語を対象としたプロセッサである．処理できる記号列は，標準の入力装置で使用するすべての文字（英数字 36 文字，特殊文字 27 文字）から成る．

SELP 言語は，原言語と区別して SELP に制御の情報を伝えるための特殊なデリミタを含んだ言語を使用する．この種のデリミタをここでは制御記号と呼

* Self-Extensible Language Processor SELP by Noriyuki KITAHARA (Maizuru Technical College), Katsuo IKEDA and Takeshi KIYONO (Department of Information Science, Kyoto University).
** 舞鶴工業高等専門学校
*** 京都大学工学部情報工学教室

ぶ.

PL/I のように 60 文字セットを使用する言語に対しては，記述の容易さを考慮して制御記号を選ぶと，制御記号の一部がその言語に含まれてしまう．したがって，SELP が制御記号を記号列から原言語と区別して完全に識別する保障はできない．完全を期すためには，特殊文字を組み合わせて制御記号を考える必要がある．

制御記号は，SELP において構文解析のために重要な記号である．これらは，外部表現において用いられるものと，内部表現のために用いられるものの2種類ある．外部表現形は，つぎに示す9種類である．

(1)	左角括弧とコロン	〈:	マクロコールの開始
(2)	コロンと右角括弧	:〉	マクロコールの終了
(3)	左丸括弧とコロン	(:	閉じた記号列の開始
(4)	コロンと右丸括弧	:)	閉じた記号列の終了
(5)	2重引用符	"	実引数リストの区切
(6)	シャープ	#	仮引数の表示
(7)	アットマーク	@	タブセット
(8)	セントマーク	¢	復改
(9)	感嘆符	!	プログラムの停止

内部表現形は，4.で述べる．

3. マクロ言語

SELP には，2種数のマクロ命令がある．1つはシステムマクロ命令であり，もう1つはプログラママクロ命令である．これらは，通常のマクロシステムに見られる命令とは多少意味が異なる．前者は，マクロコールによって，システムに用意されている記号列を展開するための命令ではなく，マクロ名に与えられたある機能を実行する命令である．これらの機能は，定義の登録，変更，取り出し，多重展開，数値変換，演算などを行なう命令である．

3.1 マクロステートメント

マクロステートメントは，原言語の入力テキストに含まれるステートメントであり，その定義を形成している記号列をコールするために使用される．このステートメントは，そのマクロ定義に従って展開されたステートメントで置き換えられる．

マクロステートメントは，マクロ名と実引数リストから成る．このステートメントは，SELP が原言語とマクロステートメントを容易に識別できるように，開始を示す制御記号 〈: と終りを示す制御記号 :〉 によって囲まれ，このステートメントの要素は2重引用符で区切られる．

〈: $F(X, Y, Z)$"X"Y"0 :〉

は，マクロステートメントの1例である．このマクロ名は $F(X, Y, Z)$ であり，3つの引数をもつ．この例において実引数は，X, Y および 0 である．実引数を必要としない場合は，対応する仮引数の個数だけ2重引用符で区切り，実引数を省略する．また，引数のないマクロをコールする場合は，単にマクロ名を 〈: と :〉 の対で囲う．

3.2 マクロ定義

DEF は，マクロ定義を示す命令である．マクロ定義は，DEF が2つの引数—利用者の定義するマクロの名前とマクロの本体を示す任意の記号列—を持つマクロステートメントの1種である．ここではマクロの本体を構成する記号列を単にマクロ値と呼ぶ．

マクロ定義の一般的な形は

〈:DEF"macro name"(:macro value:):〉

である．マクロ値を囲む制御記号 (: および :) の対は，マクロコールに対してマクロ値の参照を限定する．あらゆる状態からのマクロコールが認められるので，マクロコールが完了しない時点で別のマクロコールが行なわれ，それに対して記号列が作成されることがある．展開されて出力される記号列には任意のものを含み得る．このことは，マクロコールの実引数やマクロ定義のマクロ値にマクロの定義を含んでもよいことをも意味する．定義過程においてマクロ値が，別のマクロコールによって変更され得ることは，マクロの本来の定義から別の意味を表現することになるので，より柔軟な機能を与えることとなる．この制御記号の対で囲まれた記号列を閉じた記号列と呼び，多重に入れ子とすることができる．この対に囲まれない開いた記号列となるのは，マクロの展開が完了したときであって，一つの展開に対して1組の対が外側から除かれる．

定義を含むことができる原言語は，その言語の拡張が可能であるという特徴をもつ．SELP は，拡張された言語から原言語へリダクションを実行するためのプロセッサであるとも見なせる．リダクションのための規則は，原言語とは無関係であるので，すべてのプログラミング言語を対象とすることができる．しかし，これを実行するためには，マクロ値として様々の言語を組み込む場合に，言語の拡張を利用者が自由に表現できるような形式を選ぶことが肝要である．

マクロには全体的な定義と局部的な定義がある．全体的な定義は，マクロステートメントにおいて一番外

側の命令が DEF であるマクロ定義である．局部的な定義は，マクロステートメントの実引数部分にあらわれるマクロ定義である．一般に，マクロステートメントの実引数リストは，展開が完了したとき失われる．このことは，実引数部分において定義されたマクロについてもいえる．すなわち，この種の定義は，一時的な効力しかもたない．

3.3 マクロ展開

SELP は，入力テキストを読み込み，このテキストをある規則によって変換し，出力テキストを作成する．この出力テキストは，普通，コンパイラかアセンブラのソースプログラムとなる．

テキストの変換は，マクロ値によって規定された記号列を用いて実行される．これを展開という．マクロ値には仮引数を含むことがある．仮引数を明示するために制御記号 # と引数記号とを使用する．仮引数は，マクロステートメントの実引数リストにおける引数の位置的な順序によって #1, #2, …と対応づけられる．ただし，マクロ名の位置に対応する仮引数は #0 である．仮引数の番号は，1バイトで示され 10 以上の表示は，ISO コード系において数字に続く文字を用いる．たとえば，文字 I は数字の 19 を意味する．

いま，つぎのようなマクロ ADD を考える．

〈:DEF"ADD" (:@LOAD@#1¢@#2¢
@STORE@#1¢:):〉

このマクロステートメントは，加算変数と被加算変数を引数として，演算結果を被加算変数に格納するための1アドレスアセンブリ言語の命令群を定義している．マクロ値に含まれる制御記号@と¢によって，出力形式が決まる．これは，GPM の拡張である．タブセットは，適当なカラムに行なわれているものとする．マクロステートメント

〈:ADD"*A*"*B*:〉

により，つぎの展開結果を得る．

▼LOAD	▼*A*	
ADD	*B*	
STORE	*A*	(▼ はタブセットの位置)

あらゆるプログラムは，SELP の対象として見ると単なる記号列であるから，特別の言語を指示して展開方法を示すことはしない．拡張した言語の翻訳を利用者が意図したとおりに実行させる方法が重要である．

SELP は，一般的なマクロコールを処理する機能を備えているので，様々の記号列を作り出すことができる．例を用いてマクロの展開がいかに行なわれるかを示す．

例 1

マクロ定義

〈:DEF"*X*" (:〈:#1 IS #2.¢:):〉
〈:DEF* "Y" (:〈:RPT"*X*"*A*"*B*"*B*"*C*"2:〉
#1〈:*X*"*A*"*C*:〉:):〉

マクロコール

〈:*Y*"HENCE, :〉

マクロ展開

A IS *B*.
B IS *C*.
HENCE, *A* IS *C*.

3.4 システムマクロの形式と機能

システムマクロ命令は，SELP の処理能率とプログラムの記述性を高める目的で提供される．システムマクロ命令は，定義に関係のある命令 DEF, ALT, VAL, コールに関する命令 RPT, 数値変数に関する命令 BTD, DTB, BTH, HTB, 演算に関する命令 ARI の9種類である．代表的な命令のマクロコールの形式と機能を以下に述べる．

(1) 〈:DEF"macro name" (:macro value:):〉
マクロの定義，マクロ名 (macro name) とマクロ値 (macro value) を SELP の MDC (Macro Definition Chain) に登録する．

(2) 〈:RPT"macro name"a_{11}"…"a_{1j}"…"a_{nk}"
n:〉
マクロコールの繰り返し，これは，GPM の拡張機能である．このマクロステートメントによってマクロコールの繰り返しをさけることができ，長い記号列でも簡潔にマクロ定義することを可能にする．n は，繰り返し回数であり，a_{ij} は，i 回目の繰り返しにおける k 番目の実引数である．ただし，k は，仮引数番号の最大値であり，i および j は，$1 \leqq i \leqq n$, $0 \leqq j \leqq k$ である．$k=0$ のときは，実引数を必要としない．

(3) 〈:BTD"binary number:〉
2進数から 10 進数への変換．

(4) 〈:ARI"operator"a_1"a_2:〉
代数演算，演算子 (operator) は，＋，－，＊，／のいずれかである．

---

* RPT については 3.4 節参照

4. 処理プログラムの構成と動作

SELP は，PL/I で記述されている．その大きさは，約 600 ステップであり，2割程度がエラー処理とデバッグ機能に使われている．

データ領域の構成は，回帰的処理を可能とするように，プッシュダウンスタックを基本としている．スタックの1要素は，長さが1バイトである．処理する対象は記号列であり，記号列は，1文字ごと入力時に内部コード（ISO コード）に変換され処理される．この要素は，文字のほかに，外部に現われることのない制御記号やポインタの格納にも使用される．

入力テキストに現われる記号列は，制御記号によって単語に分割され，内部表現形に変換される．内部表現形で表わされる単語は，その先頭に単語の長さが付加されており，論理的には長さの制限がない．

Fig. 1 に SELP の基本的な処理過程を示す．

4.1 構文解析

構文の解析は，記号列を1文字単位に分割して行なわれる．走査される文字は，入力ストリームかスタックのどちらかのものである．どちらから入力するかの選択は，入力ポインタ p によって決定される．$p=0$ のとき入力ストリームから，$p\neq 0$ のとき p が指定するスタックの要素が使用される．走査された文字は，

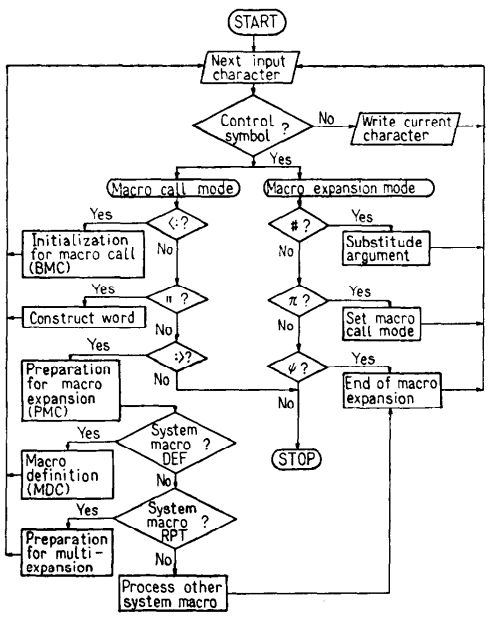

Fig. 1 Main loop of SELP

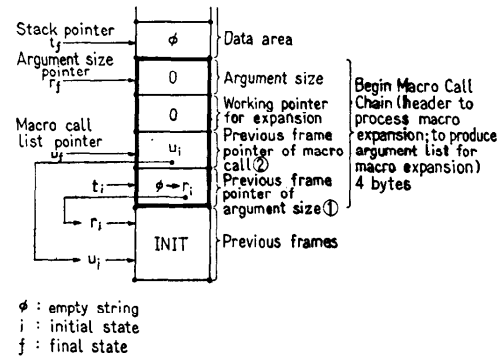

Fig. 2 Diagram of stack for control symbol 〈:

制御記号か否か判定され，制御記号以外は単語の長さを管理するポインタ r の値によって複写すべき行き先を決定する．$r=0$ のとき出力ストリームにおかれ，$r\neq 0$ のときスタックポインタ t の示す位置に記入される．制御記号が認識（2文字から成る制御記号は，2度の走査によって認識）されると，その記号のもつ意味に従って SELP に定められた規則が適用される．

4.2 制御記号の処理

（1） マクロコールの開始

制御記号 〈: によってスタックには BMC (Begin Macro call Chain) が形成されてプッシュされる．処理後のスタックの状態を **Fig. 2** に示す．

（2） 実引数リストの区切り

制御記号 "は，マクロステートメント以外で使用される場合は，何ら処理を受けず単に出力される．マクロステートメントの中で用いられた場合は，単語 WORD の長さを計算してスタックの要素の内容が変更され，つぎの単語に対する準備をする．スタック状態を **Fig. 3** に示す．

（3） マクロコールの終了

制御記号 :〉によって1つのマクロコールが認識さ

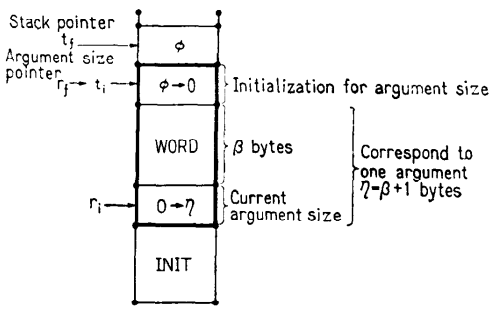

Fig. 3 Diagram of stack for control symbol "

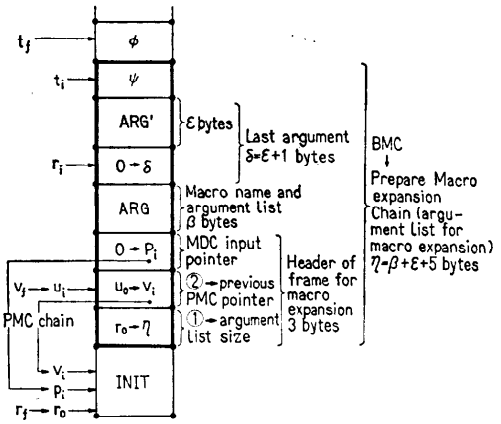

Fig. 4 Diagram of stack for control symbol :〉

れたことになり，BMC が変更されマクロ展開の準備としての実引数リスト PMC (Prepare Macro expansion Chain) が形成される．また，コールが終了した印として内部終端記号 ψ がスタックポインタ t の指す要素に格納される．スタックの状態を **Fig. 4** に示す．v は PMC の鎖となるポインタである．

マクロ展開に対する準備の後，β 個の要素 APG (マクロ名と実引数リストから成る) のマクロ名と MDC に定義されたマクロ名との照合が行なわれる．一致がとれなければエラーである．一致のとれたときそれがシステムマクロか否か判定される．システムマクロであれば制御はその処理に移る．システムマクロでなければ MDC のマクロ値を含む要素の先頭を指すように p を設定する．

(4) 仮引数の処理

制御記号は，制御記号 :〉によってマクロ展開のため実引数リスト PMC が形成された後，入力ポインタ p の指定によって MDC の内容を走査する段階で認識される．制御記号で指定された仮引数とスタック上の実引数との対応を取り実引数を出力する．スタックの状態は，Fig. 4 のままである．実引数が省略された場合は，対応を取る処理を行なわない．

(5) 内部終端記号

制御記号 ψ は，制御記号 :〉によってマクロ展開のための実引数リスト PMC が形成されるとき，その終りを示すために挿入された内部終端記号である．この記号を見つけたことは，MDC に定義されているあるマクロの展開が完了したことを意味する．この展開を要求したマクロステートメントは，展開が完了すると必要がなくなるので PMC がスタックから除かれる．マクロの展開が中断されている場合には，コールの復帰を行ない，一般的なマクロ定義のための MDC が PMC の上に積まれている場合には，MDC のリンクを矛盾なく実行するための準備が行なわれる．その準備段階のスタックの状態と展開が完了したときの状態は，**Fig. 5** のようになる．展開結果 ANY は，$r=0$ のとき 0 である．

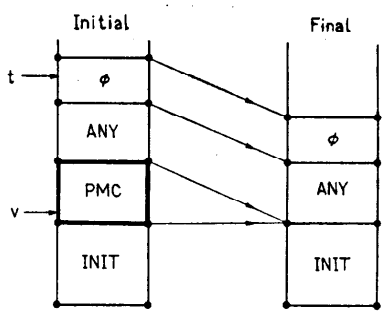

Fig. 5 Diagram of stack for internal termination symbol

(6) 内部繰り返し記号

制御記号 π は，制御記号 :〉によってマクロ展開のための実引数リスト PMC を形成する過程で，現在のマクロステートメントを何回か繰り返し実行させるために，PMC の最後に積まれた記号である．この処理は，PMC の上に積まれている MDC の先頭に入力ポインタ p を指定し，繰り返しカウント n を 1 だけ減ずる．その結果，$n=0$ のとき処理は (5) に移り，$n\neq 0$ のとき入力ポインタ p で指定されたマクロ定義の MDC のマクロ値に仮引数があるか否かを判定する．仮引数がなければ制御を復帰する．あれば，すでに使用済みとなった実引数リストを取り除き，つぎの展開のために残りの実引数を空となった要素へ順次移動させる．移動した実引数リストの最後には内部終端記号を挿入する．スタックの変化を **Fig. 6** (次頁参照) に示す．

(7) 閉じた記号列

制御記号 (: と :) は，括弧で囲まれた記号列を入力ポインタ p の指定に従って無条件に取り出せる機能をもつ．

4.3 システムマクロの処理

システムマクロは，マクロ定義の MDC にあらかじめ登録されていて変更や抹消のできない命令である．これらは，MDC にマクロ値を関連づけて登録しているわけではないので，コールによってマクロ名が照合されても記号列を展開する働きはない．この命令は，

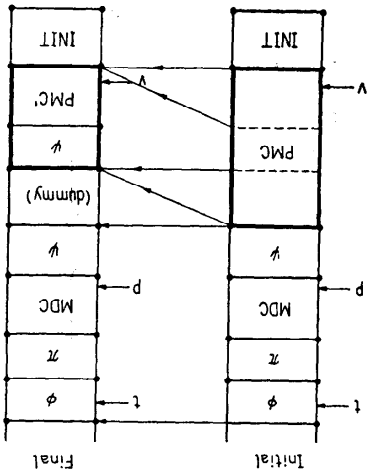

Fig. 6 Diagram of stack for control symbol π

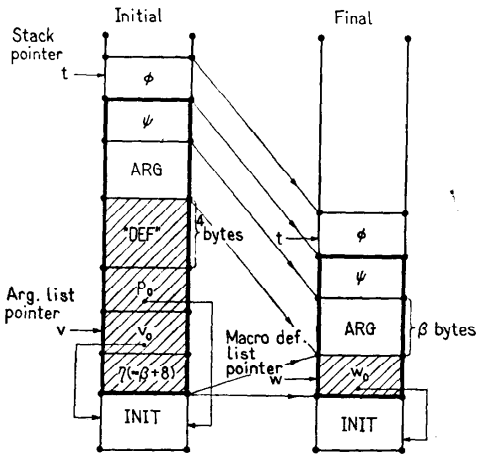

Fig. 7 Processing of system macro DEF

マクロ展開のための実引数リスト PMC が形成された後，前節の (5) へ行くべき前処理のために SELP のシステムマクロ処理に対する入口を示す．

システムマクロの主な処理として，しばしば使用される DEF と RPT について説明する．

(1) DEF

Fig. 7 にスタックの変化を示す．これは，コールのリストポインタ v をもつマクロ展開のための引数リスト PMC がマクロ定義のリストポインタ w をもつマクロ定義の MDC に変換されたことを示す．

(2) RPT

この処理は，マクロ展開のための引数リスト PMC から命令 RPT を取り除いてスタックを再編成した

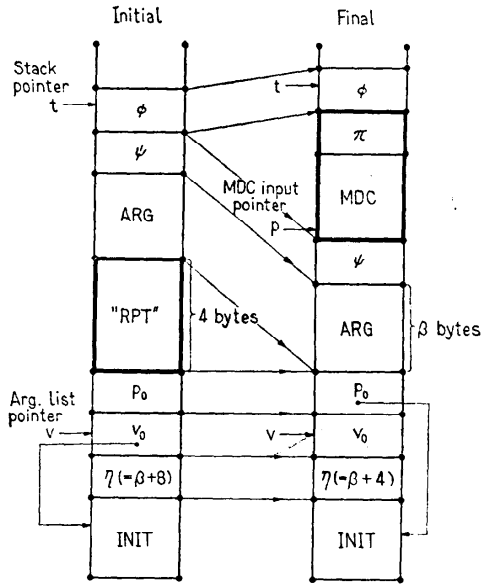

Fig. 8 Processing of system macro RPT

後，スタックの要素 ARG に含まれるマクロ名を定義している MDC を，スタックポインタの指定する要素に積んでいく．そして，最後の要素に内部繰り返し記号を挿入する．スタックの変化を **Fig. 8** に示す．

4.4 構文解析の例

3.3 の例 1 を用いて構文解析の処理過程を考える．**Fig. 9** は，例 1 のマクロ Y に関する展開過程のスタックの状態を示している．スタックにはインデックス 0 から 53 までの要素 INIT にシステムマクロを定義した MDC があらかじめ形成されている．したがって，マクロ DEF をコールして X と Y のマクロの定義を行なう MDC が **4.2** の処理によってスタックに付け加えられた．マクロ定義リストポインタの現在の値は $w=69$ である．つぎに，Y のマクロコールが行なわれると，マクロ展開のための引数リスト PMC が形成されてインデックス 104 から 116 まで積まれ，117 に終端記号 ϕ が積まれる．104 の内容 14 は引数リストの大きさを示している．要素 ARG 1 に含まれるマクロ名を照合するためには，マクロ定義リストポインタ w を介して MDC の鎖をサーチする．一致するマクロ名が，要素 MACRO Y にあることが認識されるとマクロ Y の展開が始まる．しかし，このマクロ値の最初の部分にはマクロステートメント RPT が含まれているので展開を一時中断し再び繰り返し展開を行なうための PMC が ARG 1 の上に形成され

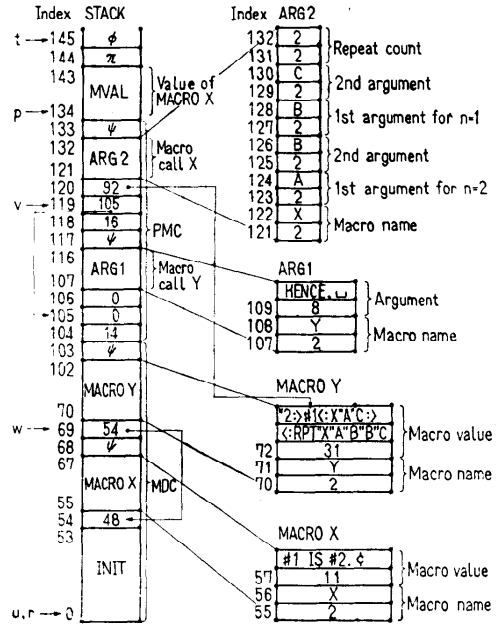

Fig. 9 Stack state associated with processing of example 1

る．引数リストの鎖を示すポインタの現在の値は $v=119$ である．この場合 RPT は，マクロ X を2度展開することを意味する．Fig. 8 に示されたようにスタックの再編成が行なわれ，その展開のための引数リスト PMC の上に，RPT によって，今展開されるべきマクロ X の定義からそのマクロ値が取り出されてスタックに積まれ，繰り返し記号 π がその上に置かれる．Fig. 9 のスタックの状態はこの段階までが示されている．インデックス 120 の要素には内側のマクロコールの展開終了後に展開すべきスタックの位置を示す入力ポインタ p の値が入っている．この内容 92 は，その定義におけるマクロ値の制御記号 # の入った要素を示す．要素 ARG 2 には，第1回目（繰り返しカウンタ $n=2$）の展開のための実引数 A および B と第2回目の実引数 B および C が含まれている．この展開によって，A IS $B.\phi B$ IS $C.$ が出力されると，ARG 2 に関する情報が消え，スタックポインタはインデックス 118 に設定される．この間に引数リストポインタ v はインデックス 105 を指すように変更され，マクロコール Y のコールから復帰する．

5. 実　　例

Fig. 10 は，FACOM-R のアセンブリプログラム

```
<:DEF"PUT"(:#1@L@N@@2 @.RA@@#3 @B@N@@-1 :>
<:DEF"DC"(:#1@DC@@#2 :):>


*****   TEST   *****
@ORG@@b800
 START@L@N@N10
@ST@N@COUNT
@TMD@N@COUNT
@B@N@@-1
@B@N@FEED
<:PUT""1"CHAR:>
@B@N@START+2
<:PUT""FEED-1"CR:>
@HLT
<:RPT"DC"COUNT""N10"10"CHAR"@@1"CR"@A"4:>
@END@@START   1
```

Source program

```
*****   TEST   *****
        ORG    @b00
START   L      N N10
        ST     N COUNT
        TMD    N COUNT
        B      N @@1
        B      N FEED
        L      N CHAR
        WRA    1
        B      N @-1
        B      N START+2
FEED    L      N CR
        WRA    1
        B      N @-1
        HLT
        B      N @-1
COUNT   DC
N10     DC     10
CHAR    DC     @@1
CR      DC     @A
        END    START
```

Object program

Fig. 10 An example of source and object program

を出力するための簡単な例題である．

6. あ と が き

SELP は，柔軟性のある表現力を有し，単純な構文法以外にはマクロ定義やマクロコールの記述方法に制限がない．加えて，大きな特徴は条件文を必要とするようなマクロの定義においても分岐先を指定するラベルを必要とせずにマクロ定義から条件に合う命令系列を生成することである．すなわち，マクロの定義を行なうときに IF 文や GOTO 文に類するものがなく，実引数に空を含めた記号列を用いてもよいようにすることによってなされる．したがって，通常のマクロプロセッサに発生しがちな定義内でのラベルと原言語プログラムで用いられているラベルとの混同の問題を避けることができる．

このような方式は，言語の拡張とかコンパイラに対する処理効率の向上などに関して強力な手段となり得，有効な応用面として言語間の変換なども考えられる．しかし，変換されるべき言語を SELP 言語で書き直す必要があることが1つの欠点である．これは，マクロステートメントの書式における区切り符号としての制御記号に起因する．したがって，SELP の適用性をより強力にするため LIMP[5] におけるテンプレート木の概念を考慮したマクロ名の認識過程を工夫する必要があろう．

最後に，本研究は昭和 48 年度文部省情報関係内地研究員として著者の一人である北原が，京都大学情報工学科清野研究室に留学中に行なわれた．

有益な御助言をいただいた京都大学大学院生小野隆夫，杉山守の両氏ならびに御討論いただいた清野研究室の皆様に感謝の意を表する．

なお，本システムの作成にあたっては，京都大学大型計算機センターを利用した．

参 考 文 献

1) J. J. Donovan; Systems Programming, McGraw Hill, pp. 111～148 (1972).
2) M. D. Mcilroy; Macro Instruction Extensions of Compiler Languages, CACM, Vol. 3, pp. 214～220 (1960).
3) M. I. Halpen; XPOP; A Meta-Language without Metaphysics. Proc. AFIPS, Vol. 26, pp. 57～68 (1964).
4) C. Strachy; A General Purpose Macrogenerator, Comput. J., Vol. 8, pp. 225～241 (1965).
5) W. M. Waite; A Language Independent Macro Processor, CACM, Vol. 10, No. 7, pp. 433～440 (1967).
6) P. J. Brown; The ML/I Macro Processor, CACM, Vol. 10, No. 10, pp. 618～623 (1967).
7) I. A. Macleod; MP/I-A FORTRAN Macroprocessor, Comput. J., Vol. 14, pp. 229～231 (1971).

(昭和 49 年 6 月 8 日受付)
(昭和 49 年 10 月 17 日再受付)