

《 論 文 》

ミニコンのためのコンパイラの作成*

小 野 隆 夫** 池 田 克 夫** 清 野 武**

Abstract

In this report, a new programming language, PL/R, and its implementation in a mini-computer, FACOM-R, are described.

PL/R is a compiler-level language designed for minicomputer programming, whose facilities range up to the assembly language level; it can handle all of the hardware features of the computer.

Accordingly, PL/R is suitable for writing every kind of program including system programs such as input/output control and interrupt processing.

The implementation of PL/R is performed by a bootstrapping method, programming in both PL/I and PL/R, utilizing a large scale computer system.

We believe that this method could be used in other computer systems to implement new computer languages efficiently.

1. まえがき

小型計算機, いわゆるミニコンのプログラミングは, 一部のコンパイラ言語 (FORTRAN, COBOL) を除いて, アセンブリ言語が主流である。ミニコンが数値計算やデータ処理の目的で用いられることはむしろ少なく, 実時間の制御の目的で使用されることが多いので, 入出力制御や割り込み処理プログラムの作成が利用者にまかされ, FORTRAN や COBOL では記述しきれない機能が存在するからである。

アセンブリ言語は機械語と1対1に対応しているので, 計算機のハードウェア機能の記述という観点からすれば最もすぐれているが, プログラマから見ればプログラムの見通しが悪いのでプログラミングやデバッグの能率が非常に悪く, 大変わずらわしい言語である。

そこで, ある程度実行効率や主記憶の使用効率を犠牲にしても, アセンブリ言語とほぼ同等のハードウェア機能の記述能力をもったコンパイラ・レベルの言語が, ミニコンのプログラミング言語として最もふさわしいと思われる。このような特徴をもった言語は, ミ

ニコンのシステム・プログラムやユーティリティなどを含むあらゆる目的のプログラムを, 簡単にむだなく行うことができ, ミニコン向き汎用プログラミング言語として適当であると思われる。

われわれは, PL/I をもとにしてミニコン用の言語 PL/R (A Programming Language for FACOM-R) を FACOM-R を対象に設計し, そのコンパイラを作成した。

PL/R のインプリメントは, 本研究の思想にしたがって, コンパイラ言語を用いて, 大型計算機でブートストラッピングにより行った。

以下, PL/R の仕様, インプリメンテーション, およびコンパイラの構造について述べる。

2. PL/R 言語仕様

PL/R はフリー・フォーマット記述の言語である。言語仕様の概要を以下に示す。

(1) プログラムの構造

プログラムは1つの外部手続きからなる。手続きはブロックを形成する。一般に手続きは, 手続き文, 条

表 3

```

<program>:: = {<procedure block>}
<procedure block>:: = <procedure head> END;
<procedure head>:: = <procedure statement>|
<procedure head> <declaration> | <procedure head> <statement> |
<procedure head> <procedure block>

```

* A Programming Language and its Implementation for a Mini-Computer. by Takao ONO, Katuo IKEDA and Takeshi KIYONO (Department of information Science, Kyoto university)

** 京都大学工学部情報工学教室

件文, 非条件文, 宣言文を含むことができる。

(2) 手続き文

手続きの先頭におかれる文であり, ラベルを持ち, 手続きの入口を指定する。仮引数を持つことができる。

Table 4

```
<procedure statement> ::= <label>: <procedure statement>|
<label>: PROC; | <label>: PROC (<formal parameter list>);
```

(3) 宣言文

宣言文は, その手続き内で使用する変数および配列を宣言する文であり, 次のオプション指定を行うことができる。DEF オプションは, 変数の割りつけ場所を絶対番地で指定する。INIT オプションは, 変数の初期値 (主記憶に割りつけられるとき) の設定を指定する。宣言は, 宣言の行われた手続きブロック内でのみ有効である。宣言は, ブロックの先頭で行われねばならない。

Table 5

```
<declaration> ::= <declare statement>|
<declaration> <declare statement>
<declare statement> ::= <declare head>; | DCL;
<declare head> ::= DCL <declare term>|
<declare head>, <declare term>
<declare term> ::= <declare factor>|
<declare factor> <declare option>
<declare factor> ::= <identifier>|<identifier> (<constant>)
<declare option> ::= INIT (<constant list>)| DEF (<constant>)|...
```

(4) 条件文

条件文は IF 文だけである。IF 文は, IF と THEN にはさまれた条件項が真のとき THEN に続く文を実行し, 偽のとき THEN に続く文を飛ばして実行することを指定する。条件項は論理式もしくは CAR (CARRY Register), OVF (OVERFlow register) である。

Table 6

```
<conditional statement> ::= <if clause> <unconditional statement>
<if clause> ::= IF <logical expression> THEN
```

(5) 非条件文

非条件文は, 代入文, GO TO 文, CALL 文, RETURN 文, I/O 文, ON 文, AKI 文, CM 文および DO ユニットである。

DO ユニットは, 内部に条件文, 非条件文を含むことができる。

DO 文によって作られる DO ユニットは, いくつかの文をまとめて1つの非条件文にする役割をし, DO WHILE 文によって作られる DO ユニットは,

Table 1 Operators of PL/R.

オペレータ	種 類	レベル	意 味
OR	二項	1	論理 OR
AND	二項	2	論理 AND
NOT	単項	3	論理否定
>	二項	4	大小比較
<	二項	4	大小比較
=	二項	4	大小比較
BIT	二項	4	XBIT n : 変数 X の bit n を 1 と比較
@	二項	5	1 語にわたって bit ごとに排他 OR
/	二項	5	1 語にわたって bit ごとに OR
&	二項	6	1 語にわたって bit ごとに AND
†	単項	7	1 語にわたって bit ごとに 0,1 の反転
-	単項	8	2 の補数を作る
+	二項	9	加算
-	二項	9	減算
SLL	二項	10	Shift Left Logical
SRL	二項	10	Shift Right Logical
SLC	二項	10	Shift Left Circular
SRA	二項	10	Shift Right Arithmetic

注 レベルが高いほど優先的に実行される。

WHILE と; の間におかれる条件項が真である間 DO ユニット内の文をくりかえし実行し, 偽になれば DO ユニットの次の文へ飛ぶ役割をする。

Table 7

```
<do unit> ::= <do head> END;
<do head> ::= DO; | DO WHILE <logical expression>; |
<do head> <statement>
```

代入文は, 変数および配列要素の演算結果を変数または配列要素に代入するための文である。後で述べるように, 変数および配列には属性が規定されていないので, 演算子と変数または配列要素との間に条件はなく, Table 1 に示すレベル 5 以上の演算子を式の中で自由に使用することができる。アキュムレータも特に ACC と書くことにより直接演算の対象とすることができる。

Table 8

```
<assignment statement> ::= <primary> = <expression>;
<primary> ::= <identifier>|<identifier> (<expression>)| ACC
<expression> ::= <expression> <operator-1> <term-0>|<term-0>
<term-0> ::= <term-0> & <term-1>|<term-1>
<term-1> ::= # <term-2>|<term-2>
<term-2> ::= <term-2> <operator-2> <term-3>| - <term-3>|<term-3>
<term-3> ::= <term-3> <operator-3> <factor>|<factor>
<factor> ::= <variable>|<variable> (<expression>)| (<expression>)|
<constant>
```

CALL 文は手続きをよび出すときに用いる文であり, 実引き数をもつことができる。引き数の受け渡しは, 値を受け渡す方式 (Call by Value) である。

Table 9

```
<call statement> ::= CALL <label>; |
```

```
CALL <label> (<actual parameter list>);
<actual parameter list> ::= <expression> |
    <actual parameter list>, <expression>
<return statement> ::= RETURN;
```

GO TO 文は、プログラムの流れを変えるための文であり、ラベルにより行先きを指定する。

Table 10

```
<go to statement> ::= GO TO <label>;
```

I/O 文は、プログラム・モードおよびインターレース・モードの入出力を行うための文である。プログラム・モード入出力命令には、WRA (入力), RDA (出力) があり 1 バイト (8 ビット) の転送を行う。インターレース・モードの入出力命令には、SIO (入出力とも) があり、サイクル・スチールにより、ブロック転送を行う。このほかに入出力に関連した命令で、入出力機器起動のための CTL, 状態を調べるための SNS, SNI がある。それぞれパラメータで、入出力装置の機番、コマンド、処理対象の変数名などを指定する。

Table 11

```
<I/O statement> ::= <I/O label-1> (<expression>, <primary>); |
    <I/O label-2> (<expression>, <expression>);
<I/O label-1> ::= SNS | RDA | SNI
<I/O label-2> ::= CTL | WRA | SIO
```

ON 文, AKI 文, CM 文は割り込み処理ルーチンを記述するための文である。ON TRAP 文は割り込み処理ルーチンの入口を指定する文であり, ON RESET 文は割り込み処理終了後, 通常モードでの実行が開始される位置を指定する文である。AKI 文は割り込みをおこした機器の機番を調べる文であり, CM 文は通常モードと割り込み禁止モードとのモード変更を行うための文である。

Table 12

```
<ON statement> ::= ON TRAP <label>; | ON RESET <label>;
<AKI statement> ::= AKI (<primary>);
<CM statement> ::= CM;
```

(6) データ

処理の対象となるデータとして、定数、変数、配列 (1 次元) がゆるされている。定数はつぎの 4 種類, 10 進定数, 16 進定数, BIT 定数, 文字定数がゆるされているが、すべて 1 語 (16 ビット) に納まる範囲内の大きさでなければならない。変数および配列要素も 1 語単位に割りつけられ、属性をもたない。

(7) 予約語

プログラムの中で特別な意味をもつキー・ワードは

予約語として定義されており、変数やラベルの記号 (Identifier) として用いることはできない。

3. PL/R のインプリメント

RL/R の FACOM-R へのインプリメントは、大型計算機 FACOM 230-75 (以後 F. 230-75 と記す。)を用いて、ブートストラッピングにより行っている。まず核となる PL/R コンパイラを PL/I で記述して F. 230-75 上に作り、以後は PL/R により PL/R コンパイラを記述してインプリメントを完成させる。この方法は、アセンブリ言語を用いずにコンパイラ言語のみを用いて行うので能率がよく、また途中使用する計算機の機種や言語の影響をうけない。

ブートストラッピングを形式的に表記するために記号とその演算の定義を行う。

【定義 1】 言語 L で動作する計算機を M(L) とする。

【定義 2】 処理 X を行うプロセッサ*を P(X) とする。

【定義 3】 言語 L₁ で記述されたプログラムを言語 L₂ で記述されたそれと等価なプログラムに変換するコンパイラで、それ自身は言語 L₀ で記述されているものを COMP (L₀|L₁→L₂) とする。

【定義 4】 M, P, COMP の間の演算を次のように定義する。

- i) $M(L_1) * COMP(L_1 | L_2 \rightarrow L_3) = P(L_2 \rightarrow L_3)$
- ii) $P(L_2 \rightarrow L_3) * COMP(L_2 | L_4 \rightarrow L_5)$
 $= COMP(L_3 | L_4 \rightarrow L_5)$

以上の定義を用いれば、PL/R のインプリメントは次のように表現できる。F. 230-75 および FACOM-R の機械語を ML-75, ML-R とすれば、PL/I で記述された核となる PL/R コンパイラおよび F. 230-60 システムに用意されている PL/I コンパイラは、

$COMP(PL/I | PL/R \rightarrow ML/R)$

$COMP(ML-75 | PL/I \rightarrow ML-75)$

と表わすことができる。ここで i), ii) 式を用いると、

$$M(ML-75) * COMP(ML-75 | PL/I \rightarrow ML-75) \\ = P(PL/I \rightarrow ML-75)$$

$$P(PL/I \rightarrow ML-75) * COMP(PL/I | PL/R \rightarrow ML-R) \\ = COMP(ML-75 | PL/R \rightarrow ML-R)$$

となり、ML-75 に変換された、PL/R コンパイラを得る。ここで $COMP(PL/R | PL/R \rightarrow ML-R)$ を記述し i), ii) 式を適用すると、

* ここでいうプロセッサは一種の Black Box である。

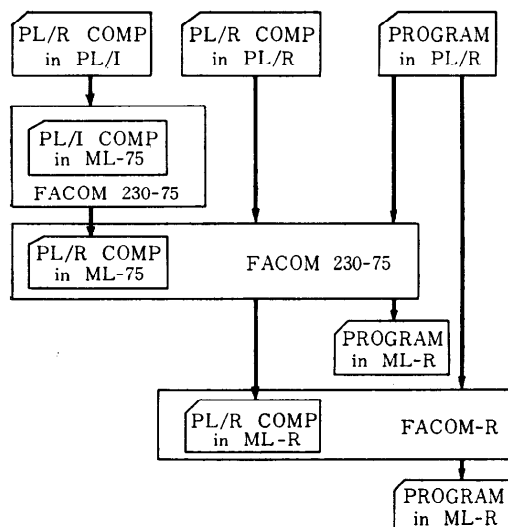


Fig. 1 Bootstrapping method.

$$M(ML-75) * COMP(ML-75 | PL/R \rightarrow ML-R)$$

$$= P(PL/R \rightarrow ML-R)$$

$$P(PL/R \rightarrow ML-R) * COMP(PL/R | PL/R \rightarrow ML-R)$$

$$= COMP(ML-R | PL/R \rightarrow ML-R)$$

となり、FACOM-R の機械語で記述された PL/R コンパイラを得ることができる。(Fig. 1 参照)

4. PL/R コパイラ

本 PL/R コンパイラは4パス方式である。各パスは次の処理を行っている。

4.1 パス 1

パス1は単語解析 (Lexical Analysis) パスであり、PL/R ソース・プログラムを区切り記号 (Delimiter) によって単語に分割し、個々の単語を内部表現形でおきかえる。

4.2 パス 2

パス2は構文解析パスである。このパスを構成するためにプロダクション言語 PL を用いてアルゴリズムを記述し、それに基づいて構文解析ルーチンを作成した。より一般的な方法としては、プロダクション言語のためのインタプリタとプロダクション言語によるアルゴリズムの記述によってこのパスを作成する方法も考えられる。

(1) プロダクション言語

プロダクション言語 PL は文法の生成規則、生成規則の適用順序、意味処理、エラー処理などの表現を含む言語であり、ボトム・アップ構文解析の手続き記

述のための実用的な言語である。

PL の書式は、ラベル部、プロダクション部、実行部、スキャン部、行き先部の5つのフィールドをもつ固定形式である。

PL は次のように解釈されて構文解析が行われる。(Table 2 参照) 構文解析のためのスタックが用意されていて、シンボルが積まれている。プロダクション部の矢印の両側のシンボル列は、右端がスタックの最上部と対応している。スタックに積まれているシンボル列がプロダクション部の矢印の左側のシンボル列と一致したとき、そのシンボル列をスタックからポップ・アップし、矢印の右側のシンボル列をスタックにプッシュする。同時に実行部に指定された意味処理ルーチンを実行し、スキャン部を実行して、行き先部で指定されたラベルをもつ文へ飛ぶ。プロダクション部の矢印の左側のシンボル列とスタックのシンボル列が一致しない時は処理を行わずに次の文に行く。どれとも一致しなければエラーであり、エラー処理ルーチンが呼ばれる。

(2) 構文解析のためのデータ・ベース

PL のアルゴリズムに従って構文解析を行うために

(1)

	(SYN)	(SEM)
TOS	10	—
	9	<term-0> CONST, 5 ₍₅₎
	8	<operator-1> /
	7	<expression> VAR _(Y) y
	6	(
	5	<operator-2> +
	4	<term-2> VAR _(X) x
	3	=
	2	<primary> VAR _(A) a
	1	<procedure head> —
	0	⊢ —

(2)

	(SYN)	(SEM)
TOS	8	—
	7	<expression> TEMP, 1 (temporary variable)
	6	(
	5	<operator-2> +
	4	<term-2> VAR _(X) x
	3	=
	2	<primary> VAR _(A) a
	1	<procedure head> —
	0	⊢ —

Fig. 2 Syntax-analysis Stack.

Table 2 Production syntax of PL/R (Expression part).

***** PRODUCTION SYNTAX OF EXPRESSION PART *****					
LABEL	PRODUCTION RULE		ACTION	SCAN	NEXT
ID1	$\langle ID \rangle : - \rightarrow$		EXEC 54	*	ID 0
	$\langle ID \rangle = - \rightarrow \langle PRIM \rangle =$		EXEC 51	*	EXP 0
	$\langle ID \rangle ($		EXEC 52	*	EXP 0
	$ACC = - \rightarrow \langle PRIM \rangle =$		EXEC 53	*	EXP 0
			ERROR (301)		
EXP 0	$($			*	EXP 0
	$\#$			*	EXP 0
	$+ - \rightarrow$			*	EXP 0
	$-$			*	EXP 0
	$\langle ID \rangle - \rightarrow \langle VAR \rangle$		EXEC 70	*	P 1
	$\langle CONST \rangle - \rightarrow \langle FACT \rangle$		EXEC 72	*	TM 3
	$ACC - \rightarrow \langle FACT \rangle$		EXEC 72	*	TM 3
			ERROR (701)		
P1	$($			*	EXP 0
	$\langle VAR \rangle \langle ANY \rangle - \rightarrow \langle FACT \rangle \langle ANY \rangle$		EXEC 73		TM 3
TM 3	$\langle TERM3 \rangle \langle OP-3 \rangle \langle FACT \rangle \langle ANY \rangle - \rightarrow \langle TERM3 \rangle \langle ANY \rangle$		EXEC 74		TM 2
	$\langle FACT \rangle \langle ANY \rangle - \rightarrow \langle TERM3 \rangle \langle ANY \rangle$		EXEC 75		
TM 2	$\langle TERM3 \rangle \langle OP-3 \rangle$			*	EXP 0
(#2)	$\langle TERM2 \rangle \langle OP-2 \rangle \langle TERM3 \rangle \langle ANY \rangle - \rightarrow \langle TERM2 \rangle \langle ANY \rangle$		EXEC 74		TM 1
	$- \langle TERM3 \rangle \rangle ANY - \rightarrow \langle TERM2 \rangle \langle ANY \rangle$		EXEC 77		TM 1
	$\langle TERM3 \rangle \langle ANY \rangle - \rightarrow \langle TERM2 \rangle \langle ANY \rangle$		EXEC 75		
TM 1	$\langle TERM2 \rangle \langle OP-2 \rangle$			*	EXP 0
	$\# \langle TERM2 \rangle \langle ANY \rangle - \rightarrow \langle TERM1 \rangle \langle ANY \rangle$		EXEC 77		TM 0
	$\langle TERM2 \rangle \langle ANY \rangle - \rightarrow \langle TERM1 \rangle \langle ANY \rangle$		EXEC 75		
TM 0	$\langle TERM 0 \rangle \& \langle TERM1 \rangle \langle ANY \rangle - \rightarrow \langle TERM0 \rangle \langle ANY \rangle$		EXEC 74		EXP 1
	$\langle TERM1 \rangle \langle ANY \rangle - \rightarrow \langle TERM0 \rangle \langle ANY \rangle$		EXEC 75		
EXP 1	$\langle TERM0 \rangle \&$			*	EXP 0
(#1)	$\langle EXP \rangle \langle OP-1 \rangle \langle TERM0 \rangle \langle ANY \rangle - \rightarrow \langle EXP \rangle \langle ANY \rangle$		EXEC 74		EXP 2
	$\langle TERM0 \rangle \langle ANY \rangle - \rightarrow \langle EXP \rangle \langle ANY \rangle$		EXEC 75		
EXP 2	$\langle EXP \rangle \langle OP-1 \rangle$			*	EXP 0
	$\langle EXP \rangle \langle OP-4 \rangle$			*	EXP 0
(#3)	$\langle PRIM \rangle = \langle EXP \rangle ; - \rightarrow \langle ASSIGN \rangle$		EXEC 86		UC 0
	$\langle VAR \rangle \langle EXP \rangle - \rightarrow \langle FACT \rangle$		EXEC 94	*	TM 3
	$\langle EXP \rangle - \rightarrow \langle FACT \rangle$		EXEC 95	*	TM 3
	$\langle ID \rangle \langle EXP \rangle - \rightarrow \langle PRIM \rangle$		EXEC 93	*	PRM 0
$\langle ID \rangle : \langle identifier \rangle$		$\langle VAR \rangle : \langle variable \rangle$	$\langle FACT \rangle : \langle factor \rangle$		
$\langle CONST \rangle : \langle constant \rangle$		$\langle OP-1 \rangle : /, @$	$\langle OP-2 \rangle : +, -$		
$\langle OP-3 \rangle : SLL, SRL, SLC, SRA$		$\langle OP-4 \rangle : >, <, =, BIT$	$\langle EXP \rangle : \langle expression \rangle$		
$\langle PRIM \rangle : \langle primary \rangle$		$\langle ANY \rangle : \text{Any symbol}$	EXEC.: Semantic routine name		
* : Scan next symbol					

はスタックが必要である。

構文解析スタックは Fig. 2 に示すように、ターミナル・シンボルまたはノンターミナル・シンボルをスタックする欄 (SYN) と、SYN のシンボルに対応する内容 (変数のアドレスや定数値など) をスタックする欄 (SEM) とから構成されている。SYN は生成規則とシンボル列の比較のために用いられ、SEM は中間言語などを生成する意味処理ルーチンにおける処理に用いられる。このようなスタックを用いることにより、回帰的な構文の解析を行うことができる。

(3) パス 2 における処理の例

$A = X + (Y/5);$

は Fig. 3 に示す構文木の構造をもっている。内部表

現形で与えられたシンボル列を左から順に読み込んでボトム・アップ方式で構文解析を行う。Fig. 3 に示した #1 では構文解析スタックは Fig. 2 の (1) の状態にある。Table 2 の PL に従って、

$\langle expression \rangle \langle operator \rangle \langle term-0 \rangle \rightarrow \langle expression \rangle$ の生成規則が適用されてスタックは Fig. 2 の (2) の状態に変更され、同時に、意味処理ルーチン EXEC 74 によって次の四つ組*が生成される。

(OR, (VAR, y), (CONST, 5), (TEMP, 1)) (1)
(VAR, y), はベースが VAR でオフセットが y の変数を意味し、(CONST, 5) は値が 5 の定数を意味し、

* 四つ組 (Quadruple): このコンパイラでは中間言語として最適化につごうのよい四つ組を用いている。

		LOC.	CTL	ML	ASSEMBLY LIST	
FACOM-R PL/R COMPILER (KIYONO LAB)						
		0000	0080	0000	ORG	0000
		0000	0004	1406	L	N **006
		0001	0004	7002	ST	D 0002
1	TAPUNC : PROC ;	0002	0004	1405	L	N **005
2	/* READ FROM KEY BOARD * PUNCH OUT TO PAPER TAPE */	0003	0004	7003	ST	D 0003
2	DCL BUFFER (120) , BUFPT , CR INIT('A'X) ,	0004	0004	9601	B	R **001
2	BSP INIT('B'X) , CODE ;	0005	0008	0080	DC	0080
3	BEGIN :CALL FEED(300) ;	0006	0008	0008	DC	0008
4	BUFPT=0 ;	0007	0008	0153	DC	0153
5	START :CALL GETKYB ;	0008	0002	008B	DA	0083
6	NEXT :IF CODE=CR THEN GO TO PUNCH ;	008B	STM NO. 1	0004	9601	B R **001
7	IF CODE=BSP THEN	008C	0008	0153	DC	0153
7	GO TO	008D	0004	1006	L	D 0006
8	/* BACK SPACE PROCESSING */	008E	0004	7404	ST	N **004
8	BUFPT=BUFPT-1 ;	008F	0004	C403	THI	N **003
9	IF BUFPT<0 THEN BUFPT=0 ;	0090	0004	9403	B	N **003
10	GO TO START ;	0091	0004	9601	B	R **001
11	END ;	0092	0004	0000	DC	0000
12	BUFFER(BUFPT)=CODE ;	0093	STM NO. 3	0004	8601	BL R **001
13	BUFPT=BUFPT+1 ;	0094	0008	00EE	DC	00EE
14	GO TO START ;	0095	0004	1A0B	L	1 000B
15	PUNCH :BUFFER(BUFPT)=CR ;	0096	STM NO. 4	0004	1A0C	L 1 000C
16	CALL PUTHSP(BUFFER,BUFPT) ;	0097	STM NO. 5	0004	7801	ST 0 0001
17	CALL FEED(5) ;	0098	STM NO. 6	0004	8601	BL R **001
18	BUFPT=0 ;	0099	0008	0112	DC	0112
19	CALL GETKYB ;	009A	0004	1804	L	0 0004
20	IF CODE='*' THEN GO TO BEGIN ;	009B	STM NO. 6	0004	3802	S 0 0002
21	GO TO NEXT ;	009C	0004	E900	TAZ	0000
22	FEED :PROC (NO) ;	009D	0004	9403	B	N **003
23	DCL I ;	009E	0004	1A00	L	1 0000
24	I=0 ;	009F	0004	9402	B	N **002
25	DO WHILE I<NO ;	00A0	0004	F100	SLL	0000
26	WRA(3,0) ;	00A1	0004	E900	TAZ	0000
27	I=I+1 ;	00A2	0004	9403	B	N **003
28	END ;	00A3	0004	9601	B	R **001
29	END /*FEED */ ;	00A4	0008	00A7	DC	00A7
30	GETKYB:PROC ;	00A5	0004	9601	B	R **001
31	/* GET ONE CHARACTER FROM KEY BOARD */	00A6	0004	00CD	DC	00CD
31	DCL ONECHR INIT('8000'X) ;	00A7	STM NO. 7	0004	00CD	DC 0 0004
32	CTL(2,ONECHR) ;	00A8	0004	3803	S	0 0003
33	RDA(2,CODE) ;	00A9	0004	E900	TAZ	0000
34	CODE=CODE & '7F'X ;	00AA	0004	9403	B	N **003
35	END /* GETKYB */ ;	00AB	0004	1A00	L	1 0000
36	PUTHSP:PROC (BUFFER(*),BUFPT) ;	00AC	0004	9402	B	N **002
37	/* PUNCH OUT DATA FROM BUFFER TO PAPER TAPE */	00AD	0004	F100	SLL	0000
38	I=0 ;	00AE	0004	E900	TAZ	0000
39	DO WHILE I<BUFPT+1 ;	00AF	0004	9403	B	N **003
40	WRA(3,BUFFER(I)) ;	00B0	0004	9601	B	R **001
41	I=I+1 ;	00B1	0008	00C3	DC	00C3
42	END ;	00B2	STM NO. 8	0004	1801	L 0 0001
43	END /* PUTHSP */ ;	00B3	0004	3A0D	S	1 000D
44	END /* MAIN PROGRAM */ ;	00B4	0004	7801	ST	0 0001
		00B5	STM NO. 9	0004	3A0C	S 1 000C
		00B6	0004	E800	TAP	0000
		00B7	0004	9403	B	N **003
		00B8	0004	F100	SLL	0000
		00B9	0004	9402	B	N **002
		00BA	0004	1A00	L	1 0000
		00BB	0004	E900	TAZ	0000
		00BC	0004	9403	B	N **003
		00BD	0004	9601	B	R **001
		00BE	0008	00C1	DC	00C1

Fig. 4 Example of PL/R Source List and Partial List of object.

る情報を目的コードの所定の位置に代入し、ローダに対する制御情報をつけて相対2進形式の目的プログラムとして出力する。

目的プログラム中では、変数および定数の参照がどの位置からでも行えるように変数および定数の領域を1カ所にまとめ、インデックス・レジスタをベース・レジスタとしたインデックス修飾方式で参照している。この方法はベース・レジスタの値の変更によって記憶域を主記憶上の任意の位置に割り付けることができるので、相対形式の目的プログラムを作成するのにつづがよい。

配列および絶対番地指定の変数の参照は、インデックス修飾と間接番地指定を同時に行うことができないので、絶対番地指定領域(0~511)にレジスタを設定して、実行中に番地計算を行い、間接番地指定でアクセスする方式を採用している。

シフト命令および入出力命令はオペランドが定数である場合には命令に組み込まれるが、変数および演算式の場合には実行中に命令が合成される。

分岐命令は近傍への分岐を除いてすべて自己相対間接番地指定で行われ、番地定数を分岐命令の次の語に持っている。

現在のレベルのコンパイラは変数および配列をすべて静的に割り付ける。

PL/R のプログラム例と、そのコンパイルの結果のアセンブリ・リストを Fig. 4 に掲げる。

5. むすび

PL/R で記述されたプログラムの能率は、IF 文、DO WHILE 文を除いてはパス 3 の最適化処理でアセンブリ言語で記述された場合とほとんどかわらないものになる。配列の処理の場合に実行時にソフトで番地の計算を行い、また 1 文中に同じ配列が二度以上現れた場合などに番地計算を重複して行うので多少能率が低下するが、最適化処理の能力をあげることによってある程度解決できる。

IF 文、DO WHILE 文は、PL/R 言語仕様で条件項の記述に自由な論理式が許されているので、論理計算部と条件分岐部を完全に分離して目的コードを生成しており、アセンブリ言語で記述する場合に比べて能率はかなり低下する（アセンブリ言語で記述した場合の約 2 倍の長さになる）。しかし、全体的にみた能率は最悪の場合でもアセンブリ言語で記述した場合の約 1/2 であり、さらに対象がミニコンであり、1 人で専有して対話的に用いることが多いことを考えあわせれば、実行能率の低下よりプログラミング能率の向上の方により多くの利益があると思われる。

われわれは実際に PL/R を用いてマクロ・プロセッサを作成した。PL/R で約 600 文、目的プログラム

の命令部分だけで約 3600 語であった。アセンブリ言語で作成するのに比べ、はるかに短期間で完成させることができた。

現在のレベルの RL/R の問題点は、リンク機能がないこと、入出力命令が単純なのでもう少し高級なマクロ命令も望まれること、デバッグ機能が不足していることなどである。今後はこれらの機能を付加し、さらにポインタ変数や、ラベル変数を導入する予定である。

6. 謝辞

この研究を進めるにあたり有益な助言をいただいた京都大学の林恒俊氏に感謝します。また、日ごろ協力いただいた京都大学・情報工学科・清野研究室の諸氏、ならびに舞鶴高専の北原紀之氏に感謝します。

参 考 文 献

- 1) N. Wirth, PL 360, Programming Language for the 360 Computers, J. ACM, Vol. 15, No. 1 (1968).
- 2) D. Gries, Compiler Construction for Digital Computers, John Wiley & Sons (1971).
- 3) J. A. Feldman, A Formal Semantics for Computer Languages and its Application In a Compiler-Compiler, C. ACM, Vol. 9, No. 1 (1966).
- 4) J. J. Donovan, Systems Programming, McGraw-Hill (1972), pp. 227~348.

(昭和 49 年 4 月 2 日受付)