ミニコンピュータ向きシステム記述言語とその処理系

牧野圭二* 栃内香次* 永田邦一*

(昭和49年9月30日受理)

Minicomputer-Oriented System Description Language and its Implementation

Keiji MAKINO, Koji TOCHINAI and Kuniichi NAGATA

(Received September 30, 1974)

Abstract

In this report, a new minicomputer-oriented system description language and its implementation on a minicomputer are described.

Owing to inefficient tools for software implementation in the minicomputer field, usually, the system programs are implemented using assembly language or the cross-compiling technique on a large computer, and moreover their features are also constrained by hardware restrictions, namely, short word length, small memory storage, low speed I/O etc. Therefore, for the satisfactory utilization of a minicomputer, it is necessary to design a system description language and implement its language processor to use these features effectively.

From the above point of view, we developed an effective language for minicomputers and implemented its processor using the bootstrapping method from facets of

1) seeking central and basic concept of system description language,
2) designing a grammar of minicomputer-oriented system description language, and
3) implementing this language processor.

As the result, we have been able to obtain uniformity, simpleness, flexibility etc.

1. まえがき

最近ミニコンピュータ（以下ミニコンと呼ぶ）が著しく普及し，その使用方法は多様なものとなり種々の方面への応用が試みられている。しかし，ミニコンには有力なソフトウェアの作成手段がなく，アセンブリ言語によるか，中・大型計算機を利用できる場合にクロス・コンパイルを行うかしているのが現状である。ミニコンで用いられる高水準言語は FORTRAN あるいは BASIC が一般的になっており，他にもいくつか高水準言語を開発する試みが行われてはいるものの，ソフトウェア開発に有効で実用的な言語はない。

これまで，プログラミング言語はメモリの大容量化やハードウェア機能の多様化などに助けられて発展してきた面が強く，ミニコンのようにメモリが小容量でありまた基本的ではあるが限られたハードウェア機能しか持たない計算機では，そのまま処理系を作成しようとすると，どうしても実用という面から無理を生じる。したがって，ミニコンの実用性の向上を考えると，単に既存のプログラミング言語のサブセットを作成することに重点を置くよりも，ミニコンの持つ特徴

* 電子工学科 電子機器工学講座

をうまく生かしてその処理に適した高水準言語を作成すべきである。そして，そのような言語の処理系の作成を容易にするために，またそのような言語の実例としても，ミニコン向きシステム記述言語の開発が必要である。

以上の観点から，我々は，ミニコンの可能性や限界の探求をも念頭に置いて，基本構成程度のミニコンでも実用化しうるミニコン向きシステム記述言語とその処理系の開発を，次の三つの面から行っている。

1) システム記述言語の中心となる概念の追求
2) ミニコン向きシステム記述言語の言語仕様の設計
3) 設計した言語の処理系の作成

ソフトウェア・システムの開発の方法論やその表現法もいまだ確立されているとはいえず，又，システム記述に必要な機能も体系づけられるに至っておらず，作成者の経験によっている面が多い[1]。そこに，さらにミニコンという特殊性が加わるため，最初から各種機能を取り入れて一挙に全体系を作ろうという方向ではかえってその能力を生かしきれない恐れがある。そのため，1)において，言語仕様の中心となるべき概念を，統一性，簡潔性，柔軟性，拡張性，記述性といった面から検討・決定し，それを実現するために 2) と 3) をブーツ・ストラッピング法により行い，順次機能を追加・拡張していくことにした[2]。このような方法を取った結果，目標に向け，言語仕様—処理系—使用経験—言語仕様の修正というループが形成され，言語に対する機能の必要度に従い，ミニコンの能力を生かした拡張が比較的容易に可能となった。

2. 言語の基本概念

システム・プログラムの大きな特徴の一つは，計算機あるいは計算機システムの実際の動作を直接取り扱うプログラムであるということであり，これはミニコンにおいても大型計算機においても基本的には同じである。したがって，システム記述言語は，機械の状態やその変化，動作，制御の流れといったものが記述できなければならない。しかし，機械の具体的動作は各機種により異なるのが普通であり，さらに，システムやシステム・プログラムなどの設計に当り，その方法論や表現形式といったものが確立されていない現状である。このような状態から強引に処理系を作成していくことは，将来の拡張・修飾などに際し統一性を欠き，保守の困難さや記述性の低下などをもたらす。

これを解決するために機械の動作を二つのレベル，すなわち

1) 機種（のアーキテクチャ）に依存する部分（下部レベル）
2) 機種に依存しない部分（上部レベル）

でとらえ，機種に依存しない部分の表現形のフォームを統一性を持たせて決め，依存する部分をそのフォームに埋め込む方法を取った。具体的に述べるならば，上のレベルである機種に依存しない部分というのは，主として機械の制御の流れを表わす部分であり，機械の状態から状態への遷移の図式（スキーム）の表現に対応し，一方，下のレベルである機種に依存する部分というのは，状態や状態の変化あるいは機械のあるまとまった動作を具体的に表わす部分である。したがって，このように分離することによって，上部レベルと下部レベルとを独立に拡張・修正することが可能となり，特に文法の構造の表現でもある上部レベルにおいて，システム記述に一般的である各種遷移図式の統一的記述法を決定することは，文法自体の表現の統一性・拡張性を増し，さらに，システムの記述性の向上や処理系の作成の容易さをもたらすことになる。

言語の基本形ともいうべき上部レベルの表現形は以下に述べるようになっている。システムプ

ログラムの記述あるいはモデル化の方法は未だ確立されているとはいえないが，オートマトンなどの抽象機械によって部分的には表現可能な場合が多く，しばしば用いられる。これらの表現法では，ある状態あるいはその状態への入力の性格により，取るべき動作と次の状態が決まるようになっている。従来のプログラミング言語では，この，状態とその遷移に関する部分と，動作や状態の変化に関する部分とを独立した表現で取り入れることが多かった。このことはいたずらにプログラム上の制御の流れを複雑にする可能性もあり，さらに，システム・プログラムの記述においては数値計算などの記述におけるより状態の取り扱いの比重が大きく，当然その表記手段であるプログラミング言語の仕様にその性格を反映させるべきである[3]。我々は，ステートメントを制御ステートメント（あるいは条件ステートメント）と命令ステートメント（あるいは非条件ステートメント）とに分離して考えることをせずに，ステートメントというものを，実行を行いうる条件とその条件のもとでの実行内容との対からなるものと考え，それをプログラムの基本的概念として取り扱うことにした。この概念は，モデルをマクロに見ることにより，実行方法・条件の類型化や実行内容の記述部分の階層化などを取り入れることができ，統一性を破らずに多様化・高度化が可能である。さらに文法（構文）表現を工夫することにより，現在の多くのプログラミング言語がそうであるように，条件文や繰り返し文などにより表現形式が異なるという点を改善でき，構文解析部分の簡単化を行える可能性が生じ，ミニコンに適していると思われる。

我々は記述の統一性を考慮に入れてステートメントの型式を次のように定めた。

〈ステートメント〉＝＝［〈名札〉：］［〈実行条件〉］（〈基本実行ステートメント〉｜〈ステートメントの並び〉）

〈実行条件〉　＝＝〈規制詞〉（〈条件〉）*

〈規制詞〉はフロー・チャートなどにおいてマクロにとらえたときの形……類型を示し，〈条件〉によって実行を行うための具体的な条件を示す。(ALGOL の条件文と繰り返し文に対応する例を図1に示す。) このように，各種類型に対しそれぞれ規制詞を導入することによって，多様な制御の流れを構文を複雑にすることなしにコンパクトに表現することが可能となり，表面上はサブルーチンや手続きと同一の記述型が得られ，規制詞導入に伴う処理系の拡張も容易に行うことができる。

ALGOL
例1（条件文）
```
1 : if  x>0  then  n : =n+1
```
例2（繰り返し文）
```
for q : =1  step s  until n  do
  A[q] : =B[q]
```
本言語
例1
```
L : ON (X>0) N=N+1 ;
```
例1
```
FOR (Q TO N BY S) A(Q) B(Q);
```
図1　規制詞の使用例

3. 言語仕様

3.1 設計方針

すでに述べたように，我々はミニコンを用いて，ミニコン向きシステム記述言語の可能性とその限界を求めることを意図した。現在までこのような試みはほとんどなされていないため，メモリ容量を始めとしてハードウェアなどの特殊性がどの程度作用し影響力を持つか予測することは困難である。このこともあってシステムの開発はブーツ・ストラッピング法を用いることにし，さらに今後の発展の基礎となるシステムとするために次のような点を考慮した。

1)　基本構成程度のミニコンでの使用を目的とする。

---

*　以下構文の記述には AN 記法を用いる。AN 記法については和田英一：ALGOL N (2-1)，情報処理，12 (1971)，9，p. 556-567 を参照。

最近では，カセット・テープや磁気ドラムなどミニコン用周辺装置も多様化し普及しつつあるが，語長が短く比較的小容量のメモリで使用されるといったミニコンのハードウェアから生じる制限と積極的に取り組むためにも，語長 16 bits/word，メモリ容量 4～8 kwords 程度で，入出力は紙テープ・ベース（テレタイプライタ，紙テープ・リーダ程度）の基本構成で使用できることを目標とする。

2) 言語に含める機能は最も基本的なもののみとする。

システム・プログラム記述用の言語には，リスト処理機能などのかなり高級な機能が含まれていることが望ましい[4]。しかし，現時点では，先に述べた基本文型の記述性などの確認と，今後の拡張・修正の基礎レベルを形成することを目標とし，高級な機能は本試作システムの使用経験に基づいて今後拡張していくことにした。したがって，規制詞も最も基本的である ALGOL の条件文に対応するもののみとした。

3) トランスレータ作成の容易さ。

作成対象とするミニコン・システムは紙テープ・ベースであるため，入出力時間と紙テープの編集には十分な考慮を必要とする。さらに，メモリ容量の点から作業情報を必要最小限におさえることが必要で，又，情報が局所的となるような文法が望ましく，このことはプログラムの作成・修正や理解を容易にもする。したがって，同程度の記述性を持つならば，当然ながら，処理系の作成が容易な言語仕様にすべきである。

4) 統一性，拡張性，柔軟性。

この言語仕様は最終目標ではなく，今後何回にもわたって機能の拡張・修正を行うのであるから，その際，記述の統一性を欠くことなく拡張ができるような柔軟さを持った言語仕様にしておく必要がある。又言語の高水準化により，機械の動作を直接制御することが困難となる可能性があるので，これを解決するためにアセンブリ言語の埋め込みを許すこととする。

3.2 文法の概要

上述の目標にそって文法を定義した。以下にその概要を述べる（全文については Appendix を参照）。

1) 言語の構造

a) マクロに見るならば，キャリジ・リターンで区切られた可変長文字列からなるストリングである（ストリーム変換型の処理を念頭に置く）。

b) 書式は自由であるが，原則としてステートメントは1行であり，キャリジ・リターンで終端する。したがって規制詞の影響範囲は引き続くキャリジ・リターンまでである。

c) コンパイルの単位は一つのプログラムである。サブルーチンはいくつ含んでいてもよい。

d) キー・ワードは予約語とする。

2) 識別子，定数，データの型，宣言

a) 識別子は英字で始まり，任意長のものが書けるが，識別の対象となるのは先頭から4文字のみである。

b) 変数は単純変数と一次元配列である。

c) 各変数は使用に先だって DCL 文で宣言しなければならない。

・単純変数には初期値の指定をすることができる。

・一次元配列においては上限を8進数で指定する。下限は0である。

d) 定数は8進数と文字列である。8進数の場合2進数に変換し下位から1語長，文字列は先頭から1語長とし，パリティは除く。

e) データの型は1語長の2進ビット列であり，その解釈はユーザにまかせる。

3) 演算子，関係子

a) 演算子 ＋（加算），－（減算），＊（乗算），& (AND)

b) 関係子 ＝，＼＝，＞＜，＞＝，＜＝，＞，＜

複合関係子は記述順序によらず同一の効果を持つ。

c) 演算子には優先順位を定めず，演算の実行は左から右へ行う。

d) 条件において関係子の右辺に項の並びを許す。意味は次のようである。

X＝Y1, Y2, ..., YN ⟺ (X＝Y1) ∨ (X＝Y2) ∨ ... ∨ (X＝YN)

X△Y1, Y2, ..., YN ⟺ (X△Y1) ∧ (X△Y2) ∧ ... ∧ (X△YN)

（ここで“△”は“＝”以外の関係子である。）

4) ステートメント

a) 文

・規制詞は ON のみとする。

・基本的機械の動作を表現する基本実行文は次のもののみとする。

空文，代入文，GOTO 文，HALT 文，CALL 文，IN 文，OUT 文，PROC 文

b) PROC 文

アセンブリ言語の埋め込みのために用いる。直接埋め込みの他に CALL 文によりサブルーチンとして使用でき，その際，項をパラメータとして引き渡せるが，その使用はユーザの責任において行う。埋め込まれる部分は，“PROC”の次の行から“END”で始まる行の前の行までである。

c) SUB 文

サブルーチンにおいて識別子はすべて全域的である。サブルーチン・コールのネストは許されるが，必ず“END”を通って戻らなければならない。

d) 注釈文

“＊”で始まりキャリジ・リターンで終る任意の文字列であり，プログラムの実行に何ら影響を与えない。

e) STOP 文

コンパイル単位の終りを示し，プログラムの終りに必ず入れなければならない。“STOP”の後に名札を書くことにより実行開始位置を指定できる。

5) その他

a) 記憶域クラスはスタティックのみである。

b) IN 文は1文字ずつの読み込み，OUT 文は1語ずつの出力である。OUT 文に文字列が書かれた場合，ダブルクォーテイション・マーク (″) にはさまれた文字が，スペースを含めて，そのまま出力される。“／”は復帰改行を引き起こす。

c) デバッギング機能のためのステートメントは特に用意していない。エラー・チェックはできるだけきめ細かく行い，エラーの分類番号とソース・ステートメント程度は出力し，プログラムの終りまでチェックするものとする。

4. 処　理　系

処理系の作成は TACC 1200 ミニコンピュータで行ったが，同レベルの他機種ミニコンとの汎用性を考慮に入れて作成した。

4.1 ミニコンの特殊性

処理系の作成に際し，対象がミニコンであるということから考慮しなければならない重要な点がいくつか存在する。

1) 語長が短く（16 bits/word 程度），命令長が 1 語長固定の機種が多い。

a) ディスプレイスメントが半語長程度しかとれないため，直接アドレッシング可能領域が 8 進表示で 0～377 番地程度と少ない。

b) オペレーション・フィールドが短いため，命令数が少なく，基本的な命令から構成されてはいるものの多様性に乏しい。

2) レジスタ数が比較的少ない。

3) 基本構成では記憶容量が少ない（4～8 kwords）。

4) 紙テープ・ベースでは入出力に時間がかかり，さらに入力テープの作成や修正・編集も容易ではない。

これらのことから処理系の作成には次のようなことを考える必要がある。

1) 変数や定数，作業域などの割り当てに細心の注意が必要である。

2) オブジェクト・コードの生成において冗長にならないようパターンを定める必要がある。

3) レジスタの割り当て方に注意する。

4) 単にメモリを増設しても，レジスタ数などから，インデクス・レジスタをベース・レジスタとして使用することなどが有効に行えるとは限らない。

5) 紙テープ・ベースでは，入出力に要する時間を考慮してできる限り入出力回数を減らし，入出力情報の冗長さを取り除く必要がある。

6) 入出力フォームは，後でテープの編集・修正などが行いやすいような考慮が必要である。

7) 基本構成程度の規模では，アルゴリズムの選定に当り，実行速度よりオブジェクト・コードの長さに重点を置く必要がある。

4.2 処理系の構造

コンパイラは，一般に，語彙解析部，構文解析部，コード生成部を中心に構成される。我々はこれら三つの部分を完全に独立させて相互に影響しないようにし，さらに語彙解析部と構文解析部とからなる解析フェーズと，コード生成部からなるコード生成フェーズの二つの独立したプログラム・モジュールとして作成し，二つのフェーズ間で受け渡されるデータ・ストリームを中間言語ストリームとして陽に取り扱っている（図 2）。

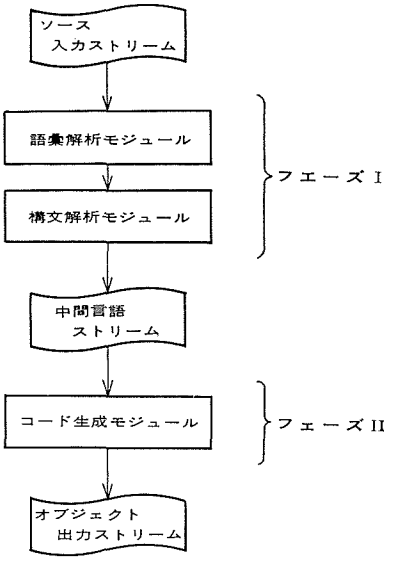

図 2 処理系の構造

現在の言語仕様は小規模であるので，メモリ容量 8 kwords のシステム構成では中間言語ストリームをコア・メモリ 4 kwords を用いて受け渡しするようにしている。又，メモリ容量 4 kwords のシステムでは中間言語ストリームを紙テープに出力する。このようにして，メモリ容量 4 kwords と 8 kwords の両システム構成ではほぼ完全にプログラムを共用することができる。

ストリーム変換を 2 フェーズ型にすることにより，上記の利点の他に，さらに次のような利点がある。まず，

第1フェーズである解析フェーズによりシンタクティカルな文法エラーのチェックをコード生成と独立して行うことができる。さらに中間言語を機械独立にすることにより，第2フェーズであるコード生成フェーズのみを修正して各種のオブジェクト・フォームを得ることができ，他機種のコードを生成することも可能である。システム・プログラムの作成においては出力コードの最適化が重要な問題であるが，これをコード生成フェーズにおいて統一的に行うことができ，ミニコン向き最適化手法の開発に役立つと思われる。

オブジェクト・プログラムの出力コードは，本処理系の生成コードの検討を始めとして，出力テープの結合，修正，編集などにより，種々の試験やチェックを行いやすくするためアセンブリ言語としたが，バイナリ・フォームでの出力に変更することも比較的容易に行える。

4.3 処理系の作成

本処理系の作成は TACC 1200 ミニコンピュータにおいて次のような外部条件のもとに行った。

記憶容量　8 kwords（語長 16 bits/word）

入出力装置　テレタイプライタ，紙テープ・リーダ

4.3.1 ブーツ・ストラッピング

本言語仕様による処理系の一応の完成までに4回のブーツ・ストラッピングを行った。

1) 最初のトランスレータはアセンブリ言語で記述しなければならないので，言語の機能は最小限におさえ，処理系の構造も目標にこだわらないことにし，コーディング・エラーを減らすことにのみつとめた。このため，トランスレータは翻訳を1行ずつ行って直接出力コードを生成する1フェーズ型であり，出力コードの効率は無視した。このバージョンには次のような制限がある。

・変数の宣言はすべてプログラムの先頭で行う。
・文において文の並びを禁止する。
・IN/OUT 文，条件において項の並びを禁止する。
・SUB 文，文字定数を禁止する。
・乗算と複合関係子は禁止する。

2) 第2回目は，2フェーズ型の構造の処理系を実現し，その基礎アルゴリズムの構成に主力をおいた。制限は次のようである。

・変数の宣言はすべてプログラムの先頭で行う。
・文字定数，IN/OUT 文における項の並び，サブルーチン・コールのネスト，注釈文の禁止。

3) 第3回目は，処理アルゴリズムと出力フォームの整備，エラー・チェック機能とその表示の充実を目ざし，出力コードの冗長度を減らすよう工夫した。文字定数だけを禁止している。

4) 第4回目は全機能を実現し，各モジュール内でも処理機能のサブルーチン化など，後の修正・拡張が容易にできるようにつとめた。

4.3.2 処理の流れ

ここでは各モジュールの処理の概略を述べる（図3)。

第1フェーズは語彙解析モジュールと構文解析モジュールとからなり，語彙解析モジュールの処理結果は，ソース入力ストリーム1行毎に構文解析モジュールに渡される。

語彙解析モジュールは読み込みルーチンにより1行読み込み，その先頭が“*”ならば次の行の読み込みを要求し，“DCL”ならば識別子表への登録の処理を行った後次の行の読み込みを要求する。行の先頭が他の要素からなっているならば，語彙解析を行い構文解析モジュールに制御

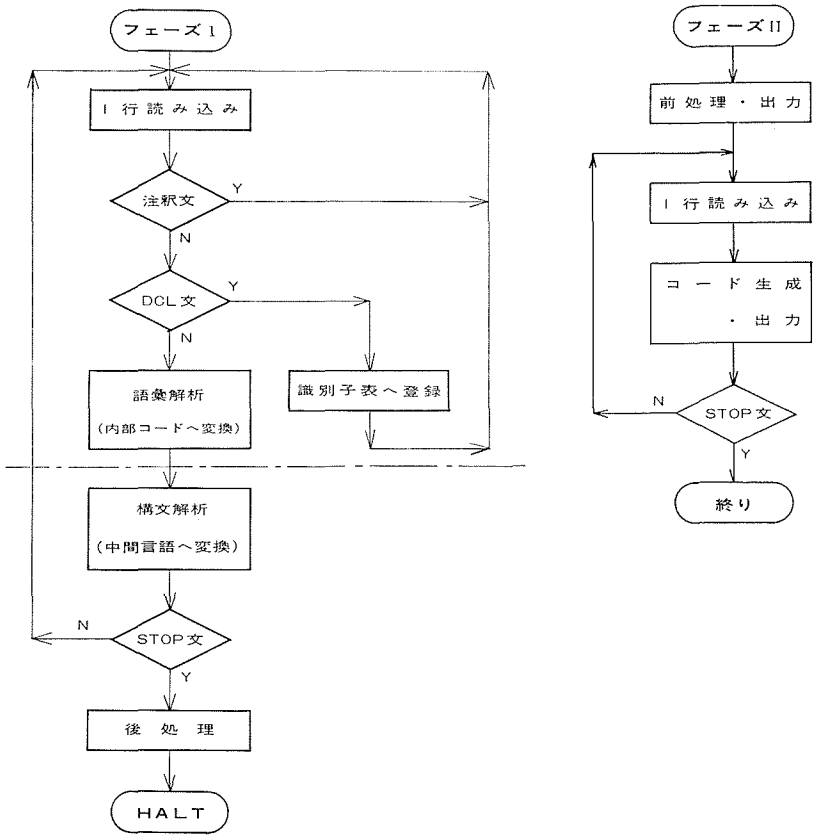

図3 処理の流れ

を渡す。語彙解析の結果は作業用バッファにより引き渡される。語彙解析モジュールでは，文字列と PROC 文の本体を除いて，シラブル・リード・ルーチンにより区切り記号表を用いそれぞれ対応する1語長の内部コードに変換される。文字列と PROC 文の本体は外部コードのまま2文字ずつ1語にパックされる。

構文解析モジュールは，作業用バッファを通して語彙解析モジュールの出力を受け取り，中間言語ストリームへ変換し，変換したステートメントが STOP 文でなければ語彙解析モジュールへ制御を戻し，もし STOP 文ならば後処理の後第2フェーズをロードするためローダへ制御を渡す。構文解析モジュールはキー・ワードを鍵にしてそれぞれの構文解析ルーチンを呼び，エラー・チェックと中間言語への変換を行う。

第2フェーズはコード生成モジュールからなり，第1フェーズから引き渡された中間言語からアセンブリ言語出力を生成する。コード生成モジュールの処理単位は，冗長出力コードの除去も考慮して，行の先頭から次の行の先頭の名札までとし，キー・ワードに従ってそれぞれ対応するコード生成ルーチンを呼び出力コードの生成と出力を行う。アセンブリ言語出力において，識別子のシステム名への付け替えは行わず，ソース入力名を保存することによってコード生成過程の検討を行いやすくしている。

4.3.3 内部コード

語彙解析モジュールにより，識別子，8進数，キー・ワード，区切り記号はすべてそれぞれ対応する内部コードに変換される。内部コードは1語長 (16 bits) からなるが，その上部半語 (上位

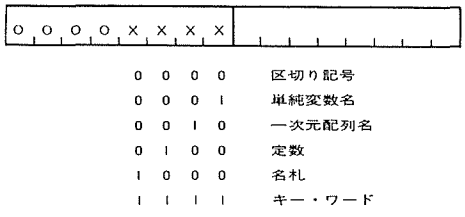

図4　内部コード

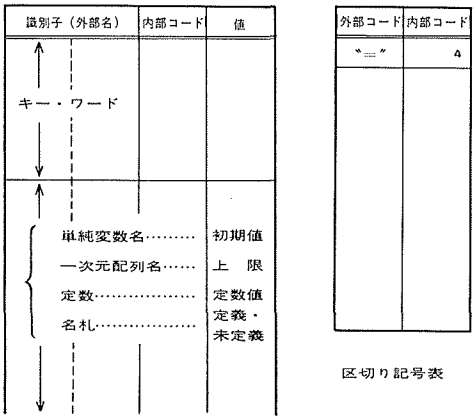

図5　表の構成

8 bits）によりその対象のクラスを表わしている（図4）。現在，クラスとしては，区切り記号，単純変数，一次元配列，定数，名札，キー・ワードの6種類がある。

内部コードへの変換は区切り記号表と識別子表（図5）を用いて行われるが，区切り記号は区切り記号表に，キー・ワードは識別子表の先頭に，それぞれ処理系固有のものとして登録されている。残りのクラスに属する対象は出現順に識別子表に登録され必要に応じ参照される。識別子表は1欄4語からなり，ソース入力の識別子をパックした形で2語，内部コードに1語，値やその他の情報に1語，それぞれ用いている。区切り記号表は外部コード1語，内部コード1語の計2語からなる固定した表となっている。

4.3.4 中間言語

中間言語は語彙解析モジュールで内部コードに変換されたソース入力を構文解析モジュールで処理して得られる。その仕様は，第2フェーズでのコード生成の容易さ，メモリあるいは入出力時間の節約のため冗長度の減少，さらに汎用性を持たせることなどを念頭において定めた。概略は次のとうりである。

・キー・ワードの内部コードを先頭に置き，そのキー・ワードに対応する処理本体の中間言語表現を続けて置く。その際，代入文に対しては代入文であることを指示する内部コードを導入する。これにより，ステートメントの区切りのセミコロンが不要となり，実行条件や IN/OUT 文などのカッコも不要となる。

・配列に対しては配列名を添字に対する単項演算子と考え逆ポーランド記法を用いる。これにより添字のカッコは不要となり，項の並びにおけるコンマも不要となる。さらに，今後の言語仕様の拡張において添字式を導入するようなときにも統一性を保つことができる。

・代入文は，右辺の中間言語表現，代入記号の内部コード，左辺の中間言語表現の順序で表わされ，右辺と左辺はそれぞれ逆ポーランド記法を用いる。

・実行条件は，関係子の内部コード，左辺の中間言語表現，右辺の中間言語表現の順とする。

・名札はコロンを除く。

なお，図6にソース・プログラムから中間言語，アセンブリ言語出力への変換例を，又，図7にオブジェクト・プログラムのメモリ割り当てを示す。

```
* SAMPLE PROGRAM;
* 1-LINE READ;
* IBUF ... INPUT BUFFER;
* PI   ... POINTER OF INPUT BUFFER;

DCL PI, IBUF(120);
LCL NUL:0, LF:12, CR:15, DEL:177;

     PI=1;
L:   IN(12,IBUF(PI)); IBUF(PI)=IBUF(PI)&177;
     ON(IBUF(PI)=NUL,DEL,LF) GOTO L;
     ON(IBUF(PI)\=CR) PI=PI+1; GOTO L;
     * NEXT STATEMENT;

     STOP;
```

```
007401      <A.ST.>
002022      <1>
000004      <=>
000414      <PI>
000200      <.CR.>
004023      <L>
007405      <IN>
002024      <12>
000414      <PI>
001015      <IBUF>
007401      <A.ST.>
000414      <PI>
001015      <IBUF>
002025      <177>
000023      <&>
000004      <=>
000414      <PI>
001015      <IBUF>
000200      <.CR.>
007402      <ON>
000004      <=>
000414      <PI>
001015      <IBUF>
000416      <NUL>
000421      <DEL>
000417      <LF>
007403      <GOTO>
004023      <L>
000200      <.CR.>
007402      <ON>
000013      <\=>
000414      <PI>
001015      <IBUF>
000420      <CR>
007401      <A.ST.>
000414      <PI>
002022      <1>
000021      <+>
000004      <=>
000414      <PI>
007403      <GOTO>
004023      <L>
000200      <.CR.>
007413      <STOP>
000200      <.CR.>
```

```
.LOC 2
3

.LOC 40
S.IN
S.OUT
S.MPY

IBUF:.IBUF

PI  :0
NUL :0
LF  :12
CR  :15
DEL :177

C.000:1
C.001:12
C.002:177

L   :.L

.LOC 400
.IBUF:.BLK 121


L.000:LDA 0,C.000
      STA 0,PI
.L    :JSR @40
       12
       2
       PI
       .IBUF
       0
       LDA 3,IBUF
       LDA 2,PI
       ADD 3,2
       LDA 0,0,2
       LDA 1,C.002
       AND 1,0
       LDA 2,PI
       ADD 3,2
       STA 0,0,2
       LDA 2,PI
       ADD 3,2
       LDA 0,0,2
       LDA 1,NUL
       SUBZ# 0,1 SNR
       JMP L.002
       LDA 1,DEL
       SUBZ# 0,1 SNR
       JMP L.002
       LDA 1,LF
       SUBZ# 0,1 SZR
       JMP L.001
L.002:JMP @L
L.001:LDA 2,PI
       ADD 3,2
       LDA 0,0,2
       LDA 1,CR
       SUBZ# 0,1 SNR
       JMP L.003
       LDA 0,PI
       INC 0,0
       STA 0,PI
       JMP @L
L.003:


.END L.000
```

図6　ソース・プログラムから中間言語，アセンブリ言語への変換例

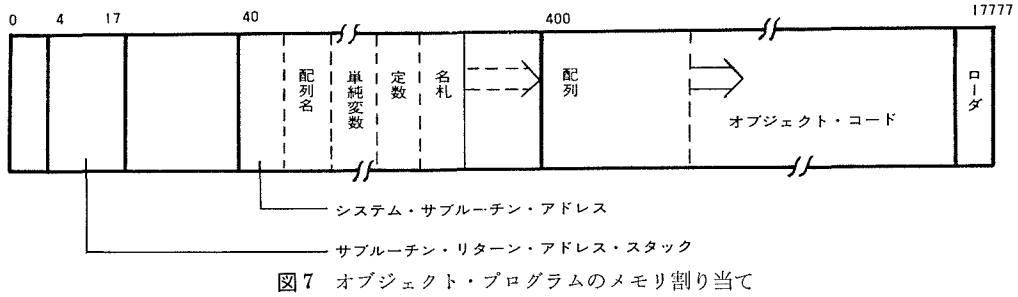

図7　オブジェクト・プログラムのメモリ割り当て

5. おわりに

以上，ミニコン向きシステム記述言語の実用化を目ざして，その基礎となる実験システムの開発について述べた。本処理系の作成及び使用経験から次のようなことがいえる。

1) ブーツ・ストラッピング法の利点・不利点については種々論じられているが，本システム程度の小規模なシステムでも，新しい言語仕様による処理系の作成に際して，自己の作成・使用経験による修正やその検証をすぐ行うことができ，有効な方法である。

2) 言語の水準がそう高くなくても，レジスタ類や命令コードの選択などに気を使う必要がないということは，思考をアルゴリズムに集中でき，エラーも減少し作成効率が上昇する。特にミニコンではシステムなどの記述に用いることのできる言語は，従来，アセンブリ言語以外にはなかったので，使用経験の浅いユーザにとってはこの言語が便利である。

3) ブーツ・ストラッピング法を用いる場合には，初期の段階で，プログラムの構造や処理の流れ，さらに基礎アルゴリズムを十分検討しておくことが，後の修正や拡張を容易に行うために不可欠である。

4) 出力オブジェクトの最適化は，システム・プログラムにとって必要であるが，現時点ではまだまだ不充分である。特にミニコン向きの最適化手法の開発が望まれる。本処理系では，アセンブリ言語出力に，直接手作業で不要命令の除去などの修正を行うことができる。

6) アセンブリ言語の埋め込みは便利であるが，高水準化の本来の目的から言っても乱用を避け，言語仕様をアセンブリ言語の埋め込みの不用な方向に改善・工夫すべきである。本処理系においては，IN/OUT ルーチン・乗算ルーチンと，シフト命令やレジスタ類の待避・回復部分において使用しているが，できるだけ使用を減らすようにしている。

6) 現在はストリング処理やポインタの取り扱いなど，高度な機能は入っていない。本システム程度の記述においてはそう不自由を感じないが，今後各種補助記憶装置を持った処理系やモニタ・システムの記述を考えるならば，これらの機能を導入することを考慮する必要があろう。しかし，ミニコンの特徴を考えるとき，言語仕様の簡潔性に重点を置き，いたずらに機能の高度化や細分化ばかりをはかるべきではないと思われる。

7) 本言語仕様で取り入れたステートメントに関する考え方は成功であったと思われる。しかし，規制詞は現在のところ“ON”のみであり，多様なパターンが表現可能であるとはいうものの，今後処理の型に応じた規制詞の導入が検討されるべきである。

以上のように一応初期の目的を達成し，今後の基礎が得られた。この結果をもとに，ミニコン向きモニタの構造の検討を含め，次の段階の言語仕様の検討を行っている。

参考文献

1) 電気試験所彙報，ESDL 特集号，34 (1970), 5-6.
2) 小久保靖世，佐谷鉄夫：コンパイラ記述用言語 BPL，情報処理，11 (1970), 6, p. 342-349.
3) Floyd, R. W.: A descriptive language for symbol manipulation, J. ACM, 8 (1961), 4, p. 579-484.
4) 山田他：FORTRAN によるコンパイラの記述に関する研究，北海道大学工学部研究報告，73 (昭 49).

Appendix

〈英字〉 ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

〈8 進数字〉 ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

〈英数字〉 ::= 〈英字〉 | 〈8 進数字〉 | 8 | 9

〈文字〉 ::= 〈'" ' 以外の使用可能な character set の要素〉

〈識別子〉 ::= 〈英字〉 [〈英数字〉]...

〈8 進数〉 ::= 〈8 進数字〉...

〈文字列〉 ::= " [〈文字〉]..."

〈定数〉 ::= 〈8 進数〉 | 〈文字列〉

〈変数〉 ::= 〈識別子〉 [((〈定数〉 | 〈識別子〉))]

〈項〉 ::= 〈定数〉 | 〈変数〉

〈演算子〉 ::= + | − | * | &

〈関係子〉 ::= = | > | < | ¬= | >< | >= | <=

〈規制詞〉 ::= ON

〈条件〉 ::= 〈項〉〈関係子〉〈項〉 { , }...

〈実行条件〉 ::= 〈規制詞〉 (〈条件〉)

〈空文〉 ::=

〈代入文〉 ::= 〈変数〉 = 〈項〉 {〈演算子〉}...

〈GOTO 文〉 ::= GOTO ␣ 〈識別子〉

〈HALT 文〉 ::= HALT

〈CALL 文〉 ::= CALL ␣ 〈識別子〉 [(〈項〉 { , }...)]

〈IN 文〉 ::= IN (〈8 進数〉, 〈変数〉 { , }...)

〈OUT 文〉 ::= OUT (〈8 進数〉, (〈項〉 /) { , }...)

〈基本実行文〉 ::= 〈空文〉 | 〈代入文〉 | 〈GOTO 文〉 | 〈HALT 文〉 | 〈CALL 文〉 | 〈IN 文〉 | 〈OUT 文〉 | 〈PROC 文〉

〈文〉 ::= [〈識別子〉:] [〈実行条件〉] (〈基本実行文〉; | 〈文〉...)

〈PROC 文〉 ::= [〈識別子〉:] PROC; 〈.CR.〉 〈.CR. を加えた任意の文字列〉 〈.CR.〉 END;

〈SUB 文〉 ::= 〈識別子〉: SUB; 〈文〉 {〈.CR.〉}... END;

〈DCL 文〉 ::= DCL ␣ (〈識別子〉 [: 〈定数〉 | (〈8 進数〉)]) { , }...;

〈STOP 文〉 ::= STOP [␣〈識別子〉];

〈注釈文〉 ::= * 〈任意の文字列〉

〈プログラム〉 ::= [(〈注釈文〉 | 〈DCL 文〉 | 〈文〉 | 〈PROC 文〉 | 〈SUB 文〉) 〈.CR.〉]... 〈STOP 文〉 〈.CR.〉