論　文

ミニコンピュータ向きシステム・プログラム記述言語 SL/M*

牧野圭二** 栃内香次** 永田邦一**

Abstract

In this paper, a new minicomputer-oriented system description language, SL/M, and its implementation are described. There exist no efficient high level language for the description of system programs in the minicomputer field, and that the user must use the assembly language. SL/M was planed out to resolve these situations.

SL/M is a high level language designed for the system program implementation on minicomputers, and a state transition diagram, which is effective tool to develop a system program, is easily expressed. SL/M is now in use on a minicomputer, and its processor was implemented by the bootstrapping method. We believe that SL/M is convienient for the system and utility program production and more develops the minicomputer utilization.

1. まえがき

最近，多くの分野にミニコンが導入されてきているが，その多くは小規模のミニコンであり，しかも，いわゆる基本構成である場合がほとんどである．

このようなミニコンの普及につれ，そのユーザ層が拡大し，計算機に不慣れなユーザによる使用が増えており，一方，その使用目的を達成するためには，各自が用途に合ったアプリケーション・プログラムやユーティリティ・プログラムなどを作成しなければならない．しかし，ミニコンにはこのようなシステム・プログラムなどを有効に記述できる高水準言語が一般にはないため，ほとんどの場合，アセンブリ言語を使用せざるを得ないのが現状である．

ミニコン用プログラムはそれ自体小規模であるのが普通であるが，メモリの制約などからそれをさらにできるだけ小さくまとめる必要性が強く，その点ではアセンブリ言語が有効な面もある．しかし，アセンブリ言語はハードウェアと密接な関係にあり，その使用にはハードウェアの知識を必要とする．さらに，小さなミニコンでは直接アドレッシング可能領域が少ないなどのミニコン特有の問題もあって，効率の良いプログラムを作成するにはかなりの経験が必要である．また，記述対象それ自身を考える場合とそれを具体的にアセンブリ言語で記述しようとする場合とでは問題のとらえ方や処理の仕方における差が大きく，プログラミング時間の増大，エラーの頻発，デバッグの困難などの原因となっている．

我々は，このような点を改善し，計算機にあまり慣れていないユーザが基本構成の小さなミニコンで容易に使用できることを目指して，あまりにもハードウェアに近すぎるアセンブリ言語に代るミニコン向きシステム記述用高水準言語として SL/M を設計し，その，コンパイラを作成した．したがって，言語仕様の設計においては，ミニコンでの使用を強く意識し，さらに，学習のしやすさ，使いやすさ，読みやすさを念頭におき，文法の簡明化，機械独立性をはかっている．

コンパイラの作成にあたっては，ミニコンの能力の最大限の利用，さらに，設計した言語の記述性の確認並びに改良の容易さを考えて，ブーツストラッピング法を用いている．

2. SL/M 言語

2.1 システム記述についての考察

システム・プログラムなどの設計並びに作成は，次

* SL/M: A Minicomputer-Oriented System Description Language by Keiji MAKINO, Koji TOCHINAI and Kuniichi NAGATA (Department of Electronics, Faculty of Engineering, Hokkaido University)

** 北海道大学工学部電子工学科

のようなステップをふんで行われると考えられる.

1) 入力, 出力及びその間の対応関係を確定する.
2) 入力から出力へ至る処理過程の全体像を巨視的に把握しサブシステム (処理モジュール) へ分解する.
3) 各サブシステムを基本的機能を持つサブシステム単位 (基本処理ルーチン, サポート・ルーチン等) にまで分解する.
4) 基本的サブシステムの具体的動作を決定する.
5) プログラムとして実現する.

これら各ステップの処理過程に対する一般的な記述, 分析, 設計, 評価手法は確立されてはいず[1], 例えばステップ2),3) の分解過程は, システム全体からみたデータの加工度の考慮のもとに, 各サブシステムの入力及び出力となるデータの中間形態を仮定しながら, 適切と思われる分解が得られるまで繰り返されるのが普通であろう. この分解過程はデータの流れが中心であり, その流れが受ける作用を分析することが目的であるので, シグナル・フロー・グラフ的なデータ中心の記述法による分析が有効な手段である. この分析と並行して, このグラフの各ノードに対し, 使用するデータ領域 (変数など) の割り当てとそこに格納されるデータの表現形の決定も行われる. このような解析を行うことにより, ノード間で実現すべき基本サブシステムは, 各ノードに割り当てられたデータ領域から入力を受け取り変換を行って出力する抽象機械として認識される.

また, ステップ 4) ではこのような抽象機械の具体的動作の解析が行われるが, そこでは, 入出力応答システムなどの解析に従来から用いられている状態遷移図による方法が有効に利用できる. したがって, 全システムは, ステップ2),3) におけるデータ遷移を表すグラフとステップ 4) における状態遷移を表すグラフとが合成された遷移グラフ (状態遷移図) として解析される.

ついで, ステップ 5) において, 実際にプログラミングが行われるが, 通常は上記のようにして得られた対象の解析結果を, フローチャート表現に変換してからプログラミングするのが一般的である. しかし, 本言語は, フローチャートを経由することなく, 解析の結果得られるデータ並びに状態遷移図から直接プログラムの記述が行えることを目指した.

このような遷移図による表現法にはモデル化の方法により種々のバリエーションがあるが, ここでは一般性を考え, モデルとして次のような抽象機械を想定する. すなわち, この抽象機械は記憶部と制御部を持ち (**Fig. 1**), その基本遷移動作は, 制御部の状態が S_i のときに記憶部の状態が M_{il} であれば動作系列 ACTIONS$_{il}$ が行われ, 制御部の状態は S_k となる (**Fig. 2**) ものと定義される. ここで, 動作系列における個々の ACTION は記憶部に対して行われるものとする. したがって, 上記基本遷移動作 (S_i : M_{il} {ACTIONS$_{il}$} S_k) の集合を指定することによって, ある機能を実現する特定の抽象機械が規定される.

2.2 言語の構造, 特徴

この言語は, 前述のように, 基本構成程度の小さなミニコン (具体的には 16 bits/word, 記憶容量 4～8 k words, テレタイプライタ, 紙テープ読取り装置程度) で使用することを目的としている. このようなミニコンに特有の問題点として, 直接アドレッシング可能領域が狭いこと, レジスタ数が少ないこと, 記憶容量が少ないことなどがあり, 複雑な解析処理を必要とする文法では, ソース・プログラムの大きさに実用上問題となる制限が付加される恐れがあるなど, 言語仕様の決定に際して処理系作成時のことに対しても十分考慮を払う必要がある. 本言語では, プログラムにおける記述単位を1行に制限し, 継続行を許していないのはその一つの現れである.

想定した抽象機械 (その遷移図) とプログラムとの

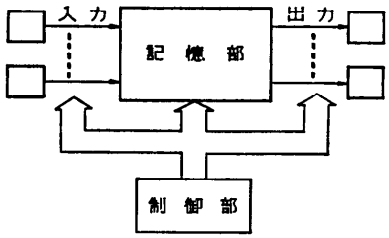


Fig. 1 The structure of an abstract machine

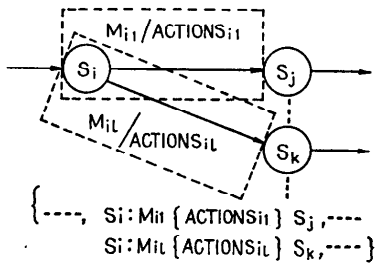


Fig. 2 The state diagram representation

対応関係は，記憶部がその変数域に，制御部は命令域にそれぞれ対応する．これを反映して，本言語は記憶部の記述部分と制御部の記憶部分とからなる．

○ 記 憶 部

記憶部は“宣言文”により規定される．宣言文では，使用される記憶セルに付けられた名前とその構造を宣言する．許されるデータ構造は単純変数と1次元配列であり，単純変数には初期値（初期状態）を指定することができ，1次元配列には上限を指定する必要がある．宣言文の一般形を次に示す．

DCL␣$v_1, v_2, \cdots, v_n$;　v_i は単純変数又は一次元配列．

データ・タイプは，システム記述という点や使用における柔軟性などを考慮し，1語長のビット・パターンのみとし，個々のデータが具体的に何を表現するかは（例えば，2進数値，2進ビット列，文字コード等）ユーザの解釈にまかされる．しかし，プログラム上に記述する場合は，8進数あるいは文字列として表現する．その際，OUT 文（後述）中の文字列を除いては1語に収まる部分だけが意味を持つ．

変数名を始めとして識別子の通用範囲はプログラム全域であり，変数の宣言は使用に先だって行う必要がある．

○ 制 御 部

制御部は“実行文”と“SUB 文”により記述される．SUB 文は遷移図の一連の遷移系列（抽象機械の遷移動作系列）をひとまとまりのものとして記述するために用いられ，その本体の記述法はプログラム本体の記述法と同様である．

実行文は抽象機械の基本遷移動作

S_i : M_{ii} {ACTIONS$_{ii}$} S_k

を記述する文（ステートメント）であって，

制御部の状態（S）　↔　名札

記憶部の状態（M）　↔　“実行条件”

動作系列（ACTIONS）　↔　“基本実行文”の並び

制御部の状態の遷移　↔　GOTO 文

という対応関係を持つ四つのプログラム要素から構成され，その一般形は

S_i : ON (condition) actions; GOTO S_j 〈CR〉

である．

ここで，基本遷移動作の要素とプログラム要素との関係は次のように考えられている．

- 制御部の状態を表す名前 S は制御部がどの動作段階にあるかの識別に用いられており，プログラム上では名札に対応すると考えられる．
- 記憶部の状態（M）とは，記憶部の全記憶セルの状態の集合のことである．しかし，現実には，記憶部の状態変化は局所的におこり，これを反映して，どの基本遷移動作を選択するかの判断には通常ごく少数の記憶セルの状態しか関係しない．そのためこの記述は，注目する記憶セルに対し，関係式によっていわゆる条件判定の形で表される．
- 抽象機械の基本動作（ACTION）は，記憶部に対する入出力，書き換え，記憶セル間のデータの移動などである．したがって，これらはいわゆる入出力文，代入文などとして表現される．
- 制御部の状態を名札に対応づけたことにより，その状態の遷移は GOTO 文として表現される．

このように本言語の“実行文”は遷移図表現と非常に良い対応関係を持っている．しかも従来の一般的手続き向き言語の慣行とも一致し，例えば，冗長な名札や GOTO 文は除去できる．また実行文の実行の仕方は次のようになる．すなわち，条件が成り立つ（真である）ときは actions 以下の実行が行われ，条件が成り立たない（偽である）ときは実行されず次の行の実行文の実行に移る．また，もし条件が恒真である（すなわち常にその実行文全体が実行される必要がある）場合には，実行条件自体を省略できる．なお，実行条件は ON (condition$_1$) ON (condition$_2$)… のように複数個置くことができるが，その場合は各条件の論理積がとられる．

本言語の実行文と同等の効果は，既存の言語，例えば ALGOL では，

S_i : **if** condition **then begin** actions; **goto** S_i **end**

のように書くことにより得られる．しかし，遷移図とプログラムとの対応が視覚的にもつけやすいこと，一つの基本遷移動作の記述を1行で行うという制限から，各行内の記述量を増すためにも冗長なキー・ワードはできるだけ除去し，コンパクトな表現形としたいこと，また，後で述べる拡張された関係式など，条件の記述に有効と思われる表現法を従来の言語と概念の混同を起こすことなく導入したいことなどから，上述のような表現形式を定めた．

見方を変えるならば，“実行条件”は“規制詞”と“条件”という二つの概念から構成されており，“条件”が“規制詞”に従って解釈され，その解釈に従ってその行内の残りの部分の実行の仕方が決定されるという

ことである．このことにより，新たに規制詞を導入するだけで，遷移図に現れる繰り返しループなども基本遷移動作として，現在と同様な表現形式で1行で記述することが可能となる．したがって本言語においては，代入文・入出力文（あるいは無条件文）と制御文（あるいは条件節や繰り返し節）という二つの異なった概念から構成されているという考え方ではなく，いわゆる実行される文は，それが実行されるべき条件や実行の仕方を常に付随して持っており実際の実行動作の記述部分と一体となった不可分のものであるという考え方に基づいている．これを記述するのがこの言語でいう“実行文”であり，したがって，いわゆる制御文に対応する独立した概念はない．

記述性の向上をはかるため，条件の記述には，通常一般に用いられる関係式の他に，(1)に示されるような拡張された関係式を許している．

T0△T1, T2, …, TN （△は関係子） (1)

ここで，(1)は(2), (3)に示すように解釈される．

関係子が ＝ のとき

(T0＝T1)∨(T0＝T2)∨…∨(T0＝TN) (2)

関係子が ＝ 以外のとき

(T0△T1)∧(T0△T2)∧…∧(T0△TN) (3)

この表現法は関係子が＝と\＝の場合に特に有用であり，ある集合の要素であるかどうかの関係，すなわち，

$t_0 \in \{t_1, t_2, \cdots, t_n\}$ （＝の場合） (2′)

$t_0 \notin \{t_1, t_2, \cdots, t_n\}$ （\＝の場合） (3′)

を表現している．

抽象機械の基本動作（ACTION）は基本実行文により記述される．基本実行文は，大きく次の三つのクラスに分類される．

- 入力（外部から記憶部へ）
- 出力（記憶部から外部へ）
- 記憶部の状態の変更すなわち書き換え

入力を表すものとしてIN文とHALT文があり，出力を表すものとしてOUT文がある．IN文，OUT文はその第1パラメータで入出力機器の機番（8進数で表示する）を指定し，第2パラメータ以降に対象となる記憶セル（変数）を列挙する．OUT文では，対象記憶セルを必要としない定数（文字列も含まれる）の出力が許され，機能文字／によって復帰改行が指示できる．HALTは記憶セルを必要としない特殊な入力形式と考えられ，コンソールからの実行継続信号の入力を要求する．

記憶部の書き換えは，いわゆる代入文の形式で記述される．また，空文，CALL文，PROC文も広い意味で記憶部の書き換えに含めて考える．データ・タイプはビット・パターン・タイプだけであるが，代入文における演算は，便宜上，その両オペランドを2進整数（加・減・乗算）あるいはビット列（AND演算）と解釈して行われる．演算子の優先順位はなく左から右へ実行され，カッコの使用も許されない．

CALL文は，一連の遷移行列を記述するサブルーチンを，それに付けられた名前により呼び出し実行させるものであり，複合動作を表す．識別子は全域的であるからCALL文は（サブルーチンも）パラメータを持たない．

プログラム上の記述単位を1行と制限したことにより，一つの遷移に伴って起きる動作系列を1行内に書けない場合も生じうる．この場合にCALL文とサブルーチンが重要な役割りをはたす．それに伴い，サブルーチンの乱造，オブジェクト・コードの増加といった問題が心配される．しかし，記述対象を十分分析し構造を明確にするならば，逆に，むしろ機能単位に構造化された見やすいプログラムとなる．また，CALL文，サブルーチンはパラメータを持たないためオブジェクト・コードの増加はあまり問題にならず，むしろ処理系を作成する際，処理が1行単位で行えること，ジャンプやブランチ命令のアドレス指定が統一的かつ容易に行えることなど，利点が多い．

PROC文はアセンブリ言語の埋め込みを行うためのものである．通常はPROC文なしでも十分記述できるが，SL/Mによる記述では効率が悪くなるような特殊動作に対して，実用上の観点から，この機能を許すことにした．アセンブリ言語埋め込み機能は便利ではあるが，この機能を用いることはプログラムの機械独立性をそこなうことにもなり，使用に際して慎重を期す必要がある．

PROC文によって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．この埋め込まれたアセンブリ言語の部分とSL/Mで記述された部分との情報の受け渡しは，識別子を通して行われ，したがって，SL/Mで用いている識別子をアセンブリ言語のオペランドとして直接記述すればよく，アセンブリ言語の簡単な知識で使用可能である．

埋め込まれる本体はその書き方によってはサブルーチンの形で記述することができ，機械独立性を確保するためには，識別子を通してSL/Mプログラムと直接

相互作用を行わないサブルーチンとして使用することが望ましい．この点を考慮して，PROC 文で記述されたサブルーチンを呼び出す CALL 文に限りパラメータを持つことを許している．なお，パラメータとの情報の受け渡し部分は，標準的な形が定まっており，それにしたがって容易に記述することができる．

3. 記 述 例

以下に，SL/M による記述例を示す．

この例は，**Fig. 3** a）の構文で記述された算術代入文をテレタイプライタから読み込み，スタックを用いて逆ポーランド記法へ変換し，変換結果を再びテレタイプライタから出力するものである．式で用いられる演算子の優先順位は↑が最も高く，ついで＊と／，最も低いのが＋と－であり，同レベルの演算子の間では左の演算子から実行される．

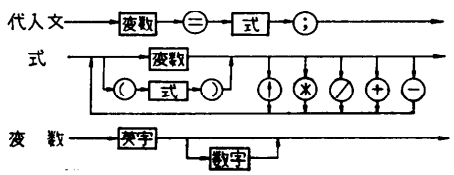
Fig. 3 a) の構文は，構文図式を用い Fig. 3 b) としても表され，Fig. 3 c) のような状態遷移図が得られる(但し，エラーに伴う状態遷移は除いてある)．この状態遷移図において，条件記述部分の一は無条件遷移を表し（したがって，状態 A_2, E_3 は Fig. 3 b) と

<代入文> ＝ <変数>＝<式>；

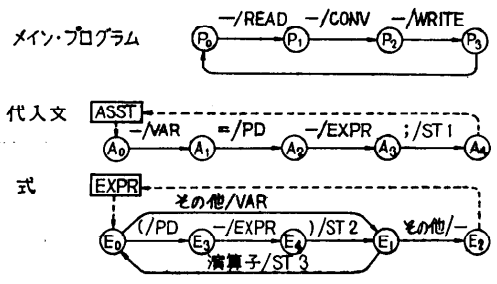
<式> ＝ (<変数>|(<式>)) {↑|(＊|／)|(＋|－)}

<変数>＝<英字>[<数字>]

a) The AN notation representation



b) The graphical vepresentation



c) The transition diagram representation

Fig. 3 Representation of assignment statement

の対応を付けるため入れられた状態であって省略できる)，動作記述部分の一は何も動作を行わないことを示す．この後，スタックなどに関する操作の分析・分解を行って得られたのが **Fig. 4** (次頁参照) のプログラムである．

このプログラムにおいて，拡張された関係式が READ, EXPR, ST3 の各サブルーチンで使われており，特に ST3 では実行条件が複合されて使われており，例えば ST3 の

ON(BI(PI)＝"＋","－")ON(STACK(PS)\
＝"(","＝")

は，次の論理的記述と同等のものとして解釈される．

((BI(PI)＝"＋")∨(BI(PI)＝"－"))
∧((STACK(PS)≠"(")∧(STACK(PS)≠"＝"))

このように，実行の条件がコンパクトに理解しやすい形で表現できる．

また，PROC 文が変数処理の VAR サブルーチンで使われている．この PROC 文は変数 I1 と I2 に入っている文字コードをパックしその結果を I1 に格納している．この例では，乗算でシフト演算を代用し，PROC 文を除き，この PROC 文に続く代入文を BO(PO)＝I2＊400－I1 のように書き換えても同じ結果が得られ，入出力速度で処理速度が制限されるような場合には実行速度上も問題はない．また，メモリに余裕がある場合には，パック操作自体行わずにすませてもよい．

EXPR サブルーチンに見られるように，サブルーチンの再帰的使用が許されているが，そこで使用する変数の値の待避などの処理はユーザが行う必要がある．

なお，この例の文字コードは ASCII コードである．

4. SL/M 処理系

SL/M 処理系の作成は，基本構成の TACC 1200 ミニコンピュータ(語長16 bits/word, 記憶容量8 kwords, テレタイプライタ，紙テープ読取り装置）で，ブーツストラッピング法を用いて行った．

記憶容量 4 kwords の最小記憶容量のシステムでも使用でき，さらに，ソース・プログラムの大きさに対する制限を除くため，語彙解析と構文解析を中心とした解析フェーズとコード生成フェーズとの二つのフェーズからなる2段階のストリーム・データ変換型の処理方式[1]を採用した (**Fig. 5**(次頁参照))．その際，フェーズ間の情報の受け渡しは機械独立な中間言語で行

```
* SAMPLE PROGRAM (SYNTAX ANALYSIS OF ASSIGNMENT STATEMENT
* AND CONVERSION TO REVERSED POLISH NOTATION);

* INPUT BUFFER & ITS POINTER;
DCL BI(120), PI;

* OUTPUT BUFFER & ITS POINTER;
DCL BO(120), PO;

* STACK & ITS POINTER;
DCL STACK(120), PS;

* WORKING VARIABLE;
DCL I1, I2,  WK;

* MAIN PROGRAM;
M01: CALL READ;  CALL CONV;  CALL WRITE;  GOTO M01;


READ: SUB;
      PI=1;
 RD1: IN(1,WK); WK=WK&177; OUT(11,WK);
      ON(WK\=0,12,46,177)  BI(PI)=WK; PI=PI+1;
      ON(WK\="!")  GOTO RD1;
      OUT(11,/);
      END;


WRITE: SUB;
       PO=1;
  WT1: ON(BO(PO)\="!")  OUT(11,BO(PO)); PO=PO+1; GOTO WT1;
       OUT(11,"!",,,,,);
       END;


ERROR: SUB;
       OUT(11,/,"ERROR",,/,/);
       END;


* CONVERSION;
CONV: SUB;
      PI=1; PO=1; PS=0;  CALL ASST;
      END;

* ASSIGNMENT STATEMENT;
ASST: SUB;
      CALL VAR;
      ON(BI(PI)\="=")  CALL ERROR;
      CALL PD;
      CALL EXPR;
      ON(BI(PI)\="!")  CALL ERROR;
      CALL ST1;
      END;


* EXPRESSION;
EXPR: SUB;
 EX1: ON(BI(PI)\="(")  CALL VAR; GOTO EX2;
      CALL PD;  CALL EXPR;
      ON(BI(PI)\=")")  CALL ERROR;
      CALL ST2;
 EX2: ON(BI(PI)="!","*","/","+","-")  CALL ST3; GOTO EX1;
      END;

* VARIABLE;
VAR: SUB;
     ON(BI(PI)<=100)  CALL ERROR;
     ON(BI(PI)>=133)  CALL ERROR;
     I1=BI(PI); PI=PI+1; I2="";
     ON(BI(PI)>=60)ON(BI(PI)<=71)  I2=BI(PI); PI=PI+1;
PROC;
LDA 0,I1
LDA 1,I2
MOVS 1,1
ADD 1,0
STA 0,I1
END;
     BO(PO)=I1; PO=PO+1;
     END;
         .

* PROCESSING ON STACK (ST1,ST2,ST3);
ST1: SUB;
 S11: ON(STACK(PS)="(")  CALL ERROR;
      ON(STACK(PS)\="=")  CALL POP; GOTO S11;
      CALL POP; BO(PO)="!";
      END;

ST2: SUB;
 S21: ON(STACK(PS)="=")  CALL ERROR;
      ON(STACK(PS)\="(")  CALL POP; GOTO S21;
      PI=PI+1; PS=PS-1;
      END;

ST3: SUB;
 S31: ON(BI(PI)="!")ON(STACK(PS)="!")  CALL POP; GOTO S31;
      ON(BI(PI)="*","/")ON(STACK(PS)="!","*","/")  CALL POP; GOTO S31;
      ON(BI(PI)="+","-")ON(STACK(PS)\="(","=")  CALL POP; GOTO S31;
      CALL PD;
      END;

* PUSH-DOWN;
PD:  SUB;
     PS=PS+1; STACK(PS)=BI(PI); PI=PI+1;
     END;


* POP-UP;
POP: SUB;
     BO(PO)=STACK(PS); PS=PS-1; PO=PO+1;
     END;


* END OF SAMPLE PROGRAM;

STOP;
```

Fig. 4 An example of SL/M program

っている．本システムではメモリに余裕が生じるため，中間言語の受け渡しはその部分を用いてすることにし処理速度の向上をはかっている．

出力命令コードなどの機種に依存する情報を含むコード生成処理を他の処理機能から完全に分離独立させるとともに，各フェーズにおいて，論理機能単位に積極的にモジュール化を行い，さらに，主モジュール～処理モジュール～処理ルーチン～サポート・ルーチンなる階層性を持たせて，コンパイラの内部構造の明瞭化とモジュール間の制御関係の明確化をはかっており，拡張・修正などが容易なものとなっている．

解析フェーズは，宣言文処理モジュール，語彙解析モジュール，構文解析モジュールの三つの処理モジュールからなる．語彙解析モジュールでは，各シラブルの分類，1語長の内部コードへの変換を行い，構文解析モジュールでは，文法エラーのチェック．中間言語

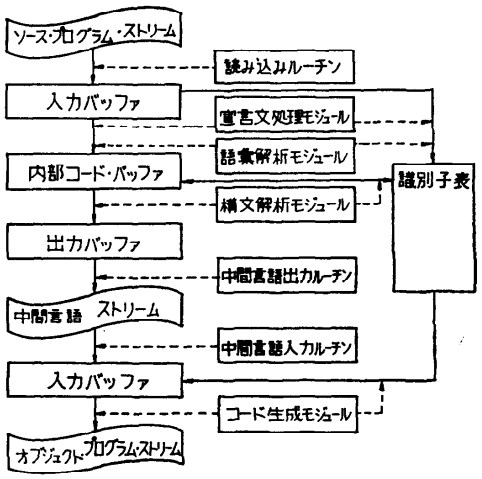


Fig. 5 Data flow of SL/M compiler

への変換を行う．中間言語は，逆ポーランド記法をもとに，できる限り冗長さを除きかつコード生成の容易さをも考慮した機械独立な表現形式である．

コード生成フェーズでは，コード生成モジュールを中心に，記憶域の割り当て，最適化，コード生成を行っている．本ミニコンのような小さいミニコンでは，直接アドレッシング可能領域が $0\sim377_8$ 程度であることが多く，しかもレジスタ数が少ないこともあって，記憶域の割り当てや効率の良い出力コードの生成を難しいものにしている．本処理系では，直接アドレッシング可能領域に，変数名，定数，名札の全域情報のみを静的に割り当て，他方，添字付き変数を中心に冗長コードの除去や最適コードの生成を行っている．

この処理系の作成には，ミニコンというハードウェアの制約内でその機能を生かした処理を行うことや，SL/M 言語の記述能力の確認などを行うためブーツストラッピング法[2),3)]を用いた．現在稼動中の処理系は，5回のブーツストラッピングにより得られたものである．

5. むすび

SL/M は，遷移図におけるある一つの遷移動作に関係することはすべて，一つの“実行文”によりコンパクトな理解しやすい形で記述できる．この結果，記述対象の分析により得られる遷移図から，フローチャートを経ることなく，直接しかも容易にプログラミングでき，計算機に不慣れなユーザでも手軽にミニコンのシステム・プログラムの作成を行うことができる．システム・プログラムという記述対象を具体的な遷移図として分析・分解していく際の，記述法あるいは詳細化の方法などは今後一層の検討を必要とするが，SL/M における考え方はミニコン用に限らず広くシステム記述言語一般に適用できるものである．

SL/M の設計にあたっては，例えば代入文に見られるように必要性が少ないと思われる機能は思いきって省き，できるだけ基本的な機能のみで構成した[4)]が，ブーツストラッピング過程における SL/M の使用経験からみても，ミニコンのシステム記述用として問題はなかった．処理系の増大や複雑化を避けるためにも，機能を吟味・選択して少数におさえることが，ミニコン用高水準言語の採るべき一つの方向であると思われる．

本処理系では，識別子などの全域情報はすべて直接アドレッシング領域に割り当て，出力コードの効率化をはかっている．しかし，システムの柔軟性という面からは，出力コードの効率が多少落ちても，レジスタをベース・レジスタのように使用して識別子を直接アドレッシング可能領域から排除し，この領域を他のプログラムに開放することも，本処理系の方式との比較において検討する必要があると思われる．

Table 1 Size of SL/M compiler

	解析フェーズ	コード生成フェーズ
ソース・プログラムの行数（ほぼ全状態数）	約 280	約 230
オブジェクト・プログラムの命令部分のステップ数	約 2100	約 1300
明示された状態数（名札の数）	76	50

本処理系はブーツストラッピング法を用いて作成されているが，この作成過程において記述性の良さが確かめられ，またSL/Mサブセットで記述したステップは，アセンブリ言語で記述する必要のあった最初のステップと比較してエラーの減少や作成時間の短縮が顕著であった．例えば，SL/M サブセットで記述したステップのエラーは平均約1個/100行で，しかもほとんどが識別子の書きまちがいなどのケアレスミスであった．なお現在稼動中の処理系の大きさは **Table 1** のようである．

参考文献

1) ESDL 特集号，電気試験所彙報，Vol. 34, No. 5-6 (1970).
2) 小久保，佐谷：コンパイラ記述用言語 BPL，情報処理，Vol. 11, No. 6, pp. 342～349 (1970).
3) J. Earley & H. Sturgis: A Formalism for Translator Interactions, Comm. ACM, Vol. 13, No. 10, pp. 607～617 (1970).
4) 山田，他：FORTRAN によるコンパイラの記述に関する研究，北大工学部研究報告，Vol. 73, pp. 1～3 (1974).
5) 和田英一：ALGOL N，情報処理，Vol. 12, No. 9, pp. 556～567 (1971).

(昭和 49 年 11 月 22 日受付)
(昭和 50 年 9 月 9 日再受付)

◇

付　録　SL/M の構文（AN 記法[5)]による）

〈プログラム〉=[〈宣言文〉|〈実行文〉|〈SUB 文〉|〈注釈文〉〈CR〉]$_{\infty}$STOP[␣〈名札〉];〈CR〉

〈SUB 文〉=〈名札〉: SUB; 〈CR〉[〈宣言文〉|〈実行文〉|〈SUB 文〉|〈注釈文〉〈CR〉]$_{\infty}$ END;

〈注釈文〉=＊[〈文字〉]$_{\infty}$

〈宣言文〉=DCL␣(〈識別子〉[: 〈定数〉|(〈8進数〉)])
{,}$_{\infty}$;

〈実行文〉 =[〈名札〉:][〈実行条件〉]$_{\infty}$
[〈基本実行文〉;]$_{\infty}$[〈GOTO 文〉;]
〈実行条件〉 =〈規制詞〉(〈条件〉)
〈規制詞〉 =ON
〈条 件〉 =〈項〉〈関係子〉〈項〉{,}$_{\infty}$
〈関係子〉 ==|<|>|(<=|=<)|(>=|=>)|
(\=|=\|><|<>)
〈基本実行文〉=〈空文〉|〈代入文〉|〈CALL 文〉|〈PROC 文〉|
〈HALT 文〉|〈IN 文〉|〈OUT 文〉
〈空 文〉 =
〈代入文〉 =〈変数〉=〈項〉{+|−|*|&}$_{\infty}$
〈CALL 文〉*=CALL␣〈名札〉[(〈項〉{,}$_{\infty}$)]
〈PROC文〉**=[〈識別子〉:]PROC; 〈CR〉[[〈文字〉]$_{\infty}$〈CR〉]$_{\infty}$
END
〈HALT 文〉=HALT
〈IN 文〉 =IN(〈8進数〉,〈変数〉{,}$_{\infty}$)
〈OUT 文〉 =OUT(〈8進数〉,(〈項〉|／){,}$_{\infty}$)
〈GOTO 文〉=GOTO␣〈名札〉

〈名 札〉 =〈識別子〉
〈項〉 =〈変数〉|〈定数〉
〈変 数〉 =〈識別子〉[((〈識別子〉|〈定数〉))]
〈定 数〉 =〈8進数〉|〈文字列〉
〈識別子〉*** =〈英字〉[〈英数字〉]$_{\infty}$
〈8進数〉 =〈8進数字〉$_{\infty}$
〈文字列〉 ="[〈"を除く任意の〈文字〉〉]$_{\infty}$"
〈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|6|6|7
〈英数字〉 =〈英字〉|〈8進数字〉|8|9
〈文 字〉 =〈空白を含むキー・ボード上の任意の文字〉
〈CR〉 =〈復帰改行〉

* CALL 文において，パラメータは PROC 文によって記述されたサブルーチンを呼び出すときに限り許される．
** PROC 文の本体の各行の先頭シラブルは "END" 以外でなければならない．
*** 識別子の適用範囲はプログラム全域である．
識別子の認識は4文字までで行い，キー・ワードは予約語である．