論　文

マクロ機能によるあるシステム記述言語の試作*

矢野秀一郎** 奥井　順** 都倉信樹**

Abstract

A language for system programs ML-11 oriented to structured programming has been implemented on PDP-11/20, and is now used.

This language is an existing macro-assembly language augmented by several useful macros which introduce some of high level language features such as control statements, subroutine statements, and some statements with respect to data structures.

So, users are able to write well-structured system programs, and if necessary, to elaborate their programs utilizing the flexibility of assembly languages.

Furthermore, this language has been implemented by using the macro facility. This approach is much easier than preparing the compiler.

1. まえがき

最近，システムプログラムを作成する場合にも，デバッグ，メンテナンス，ドキュメンテーション等を含めたプログラミングにおける生産性が大きな問題となっている．この問題を改善する一つの方法として，Dijkstra 等により提案されている structured programming[1] が考えられる．しかし，そのためにシステムプログラムを一般の汎用高級言語で書くことは，個々の計算機の機能を完全に使用し，また，効率を重視した細い記述を行なうことを不可能にする．一方，システムプログラムをアセンブリ言語で書く場合，アセンブリ言語にプログラムの論理構造を表現する便利な機能がないため，structured programming の手法を生かすことが困難である．

そこで，ここではアセンブラに付属するマクロ機能を利用して，アセンブリ言語と論理構造を表わすマクロとの複合体を考えた．すなわち，if-then-else, do-while 等の機能を果すマクロを用意し，プログラムの論理構造をこのマクロで書き，他の部分はアセンブリ言語で書く．そして，ジャンプ命令およびブランチ命令を使用せずにプログラムを書けば，いわゆる GOTO-less なプログラムが作成でき GOTO-less プログラムの長所を生かすことができる．

ところで，システムプログラムにはデータ構造が不可欠である．データ構造としては，アレイ，ブロック，フィールド（**2.3** 参照）等が考えられる．この種のデータ構造を容易に扱えれば，システムプログラムを書く場合，プログラミングにおける生産性を高めるのに役立つ．たとえば，データ構造の宣言と参照およびフィールド処理を行なうマクロがあれば便利である．

なお，用意されているマクロを用いるか，アセンブリ言語で書くかはユーザの自由である．実際，システムプログラムのように効率を重視するプログラムでは，論理構造が見にくくならない範囲内で，GOTO-less を崩す必要も生ずる[2]．

さらに，マクロを用いる利点として，マクロによる言語の設計製作がコンパイラによる作成と比べ，かなり少ない費用で可能になることがある．そして，後日マクロの修正や追加が容易に行なえる．

このような，アセンブリ言語にない種々の機能を持ったマクロによる，アセンブリ言語への柔軟な援助が本文の目的である．

我々は，このねらいのもとにアセンブリ言語も含めたマクロ言語 ML-11 を，PDP 11/20 により試作し

* A language for system programs and its implementation using a macro processor by Shuichiro YANO, Jun OKUI and Nobuki TOKURA (Faculty of Engineering Science, Osaka University)

** 大阪大学基礎工学部情報工学科

たので，その結果をここに報告する．

2. ML-11 の概要

ML-11 は，アセンブリ言語 MACRO-11 と，それに付属するマクロ機能により作られたいくつかのマクロとの複合体である．プログラムは MACRO-11 の仕様に従って書かれるが，アセンブリ言語としての文（1行）のほかにマクロで作られた下記の文がある．

1) 実行制御文
2) サブルーチンに関する文
3) データ構造に関する文

これらの文には，その内部に他の文を含まない型の文と，含む型の文があり，前者は1行のマクロコール文になっているが，後者は複数のマクロコール文にはさまれたいくつかの文，すなわち，連続する複数行となる．なお，マクロコール文は MACRO-11 の仕様によりつぎの書式を持つ．

〔ラベル：〕マクロ名〔，引数$_1$〕…〔，引数$_n$〕〔；注釈〕*

以下に 1)～3) の説明を与える．

2.1 実行制御文

ML-11 には，下記の実行制御文がある．

1) if
2) do-while, do-until
3) case, select (BLISS[3] 参照)
4) FORTRAN 型 do
5) exit

1)～4) は論理判定を行ない，プログラムの論理構造を決定する．判定条件と，判定結果により定まる実行範囲（例えば if 文では then clause や else clause）はつぎのようにして指定する．すなわち，マクロ引数としてオペランドや大小関係等を示す文字列を並べることで判定条件を表わし，複数の対応するマクロではさむことにより実行範囲を表わす．

Fig. 1 の if 文では，A番地の内容が 10 に等しいかどうかが調べられ，等しいならそれ以後対応する ELSE が現われるまでの文 (then clause) を実行し，等しくなければ ELSE 以後対応する IFEND が現われるまでの文 (else clause) を実行する．どちらも clause の実行後は IFEND のつぎの実行文にコントロールが移される．また，ELSE がなければ，then clause のみの if 文になる．

```
IF A EQ #10          WHILE B GT #1
   .                    .
   .                    .
   .                    .
ELSE                 WHEND
   .
   .
   .
IFEND
  (a) if               (b) do-while
```

Fig. 1 Examples of control statements

if, do-while, do-until, select における判定条件はつぎの4種類の単位で表わすことができる．ユーザは，同じ機能を持つマクロ (if 文では IF, IFB, IFC) の中から希望する単位に合わせて適当なものを選択する．

1) word 単位
〔例〕 IF A EQ B ; A 〈0,16〉=B 〈0,16〉?**
2) byte 単位
〔例〕 IFB A NE B ; A 〈0,8〉 ≠B 〈0,8〉?
3) データ構造単位 (2.3 参照)
〔例〕 IF A 〈0,3〉 GE B 〈8,3〉
; A 〈0,3〉≧B 〈8,3〉 ?
4) PS (Program Status Word) 単位
〔例〕 IFC VS ; previous instruction
; causes overflow?

1)～3) は論理判定のため比較命令を実行する．4) は比較命令を実行せずそのときの PS の内容により直接論理判定を行なう．つまり，先行する命令の演算結果についてオーバーフロ，キャリィ，正負，零非零を PS により判定できる．なお，1)～4) を適当に使えばアセンブリ言語で可能な論理判定はすべて行なえる．また，判定後も PS の内容は不変である．

exit 文はつぎの書式をもつ．

EXIT n　　(ただし n=1, 2, 3, …,)

この文は，これを含む実行制御文の中で，内側から n 番目の文のつぎに来る実行文へコントロールを移す．(Fig. 8 では if と do-while の2つの文から出る．)

do-while は，論理判定をループの先頭で行ない，do-until は，論理判定をループの最後で行なうが，exit 文と if 文を組み合わせれば，ループの中の任意の場所で任意回の論理判定を行なえる．これは，異常状態によるループの実行中断等に便利である．しかし，この場合もジャンプ命令やブランチ命令を無制限に使用して書かれたプログラムと比べれば，いわゆる望ましい構造を持ったプログラムを作ることができ

---

* 〔 〕は省略される場合もあることを示す．マクロ名と引数あるいは引数間の区切りはコンマでなくスペースでもよい．

** PDP-11 の1語は，1語＝2バイト＝16 ビット．また，A〈i, j〉とはA番地の第 i ビットから連続する j ビットを意味する．

```
CALL MAX, ⟨A,B⟩, C
     .
     .
     .
SUBROUTINE MAX, ⟨R0,R1⟩, ⟨R0,R1⟩
   IF R0 GE R1; R0≥R1?
      RETURN ⟨R0⟩; YES
   ELSE
      RETURN ⟨R1⟩; NO
   IFEND
SUBEND
```

Fig. 2 Example of subroutine call and definition

る.

2.2 サブルーチン

ここでは，サブルーチン宣言文，コール文の説明を与える.

2.2.1 サブルーチン宣言文

サブルーチン宣言文は，SUBROUTINE なるマクロコールに始まりSUBENDに終る複数行の文で，これらの間に一つ以上の RETURN 文と任意個の実行文を含む．SUBROUTINE のマクロ引数として，サブルーチン名，入力仮引数，値を退避させたいオペランドをこの順に宣言する．出力仮引数は RETURN のマクロ引数として宣言する．なお，引数などが複数個ある場合はそれらを“⟨”と“⟩”でくくる.

2.2.2 コール文

コール文は，CALL の後にマクロ引数としてサブルーチン名，入力実引数，出力実引数，退避させたいオペランドの順に指定する．また，サブルーチンは値で呼ばれる．**Fig. 2** では，コール文が実行されるとA,Bの値を持ってサブルーチンMAXに飛ぶ．MAXでは，R0,R1 の内容をスタック（退避）し，A,B の値を R0, R1 に代入して if 文以下を実行する．すなわち，R0 か R1 の小さくない方の値が C に返される．退避していた R0,R1 の値がもとに戻され，コントロールはコール文の次へ移る.

2.3 データ構造

ML-11 にはつぎの3種類のデータ構造がある.

1) ブロック

主記憶領域上の連続した n 語 (n≧1) の集まりを大きさnのブロックと呼び，最初の語の番地をこのブロックの先頭番地という.

2) フィールド

1語の中で連続したビットの集まりをフィールドと呼ぶ．語もフィールドに属すものとする.

3) アドレスマッピング

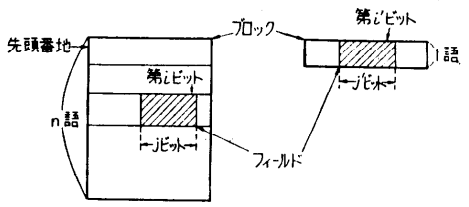

Fig. 3 Concept of block and field

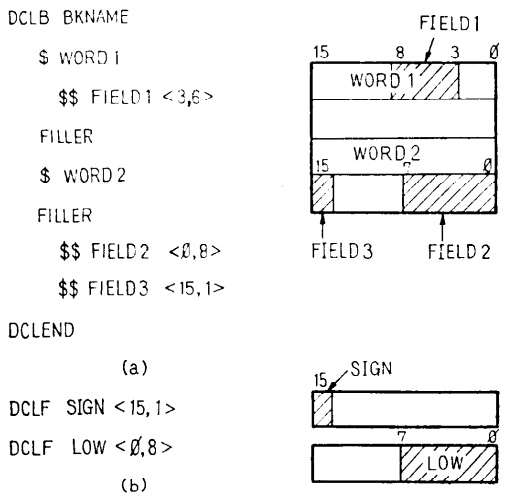

Fig. 4 Deblarations of block and field

任意個のパラメータの値に対し，主記憶上の特定の番地を一意に対応づけることをアドレスマッピングという．配列やハッシング等がこれに相当する.

Fig. 3 にブロックとフィールドの概念図を示す．ユーザは，**Fig. 4**(a) のようにあらかじめいくつかのフィールドを含むブロックの型を宣言すれば，各フィールドを名前で参照することができる．ここではDCLB～DCLEND のマクロにより右側に示されたブロックの型が宣言され，これに BKNAME なるブロック名が与えられる．$ なるマクロでブロック内の語に先頭より順に名前を与える．名前が不用のときはFILLER を用いる．フィールドに名前を与えるには，そのフィールドが属す語に対応する $ あるいは FILLER のつぎの行に $$ なるマクロでフィールド名と位置を書けばよい．フィールドが互いに重なるように宣言してもよい．この宣言は，いわばシンボルに先頭番地からの相対位置に関する情報を割り付けるもので，特定の領域をブロックとして確保するものではない.

ブロックの参照は，つぎの書式に従って行なう.

ブロック名⟨先頭番地*，フィールド名⟩

* 先頭番地は，その値を持つオペランドを書いて指定する.

ブロック名でブロックの型を示し，かつ先頭番地を指定すればこのブロック内の各フィールドは主記憶領域上に一意に定まる．つまり，同じ型を持つブロック内の対応する位置にあるフィールドは，同じフィールド名で参照でき，目的のブロックは先頭番地の指定により参照時に定まる．

また，Fig. 4(b)のようにDCLFを用いれば，レジスタを含むすべての語に対してその中のフィールドに名前をつけることもできる．参照はつぎの書式による．

ワード名* 〈フィールド名〉

語の中のフィールドは，つぎの書式に従えば名前によらず直接参照することもできる．

ワード名 〈i, j〉**

アドレスマッピングは，**Fig. 5** のようにユーザがアドレス計算をする一つのルーチンを書くことで定義され，マッピング名がつけられる．参照の書式を示す．

1) マッピング名 〈〈実引数〉〉
2) 〃 〈〈 〃 〉, フィールド名〉
3) 〃 〈〈 〃 〉, i, j〉

2), 3)を用いれば計算されたアドレス（語）の中のフィールドを参照することができる．ただし，フィールド名はDCLFで宣言された名前である．

なお，これらのデータ構造はつぎの個所で参照できる．

1) 実行制御文の判定条件内
2) マクロで用意されている転送，クリア，比較，ビット演算におけるオペランド
3) サブルーチンの引数指定部

参照例を **Fig. 6** に示す．ダブルオペランド命令では両方のフィールドの位置や大きさは任意である．大きさが異なると標準解釈により自動的に整えられる．

```
MAP ARRAY 〈R0, R1〉; SIZE=ARRAY SIZE
    WHILE R0 GT #1; c (Register 0)>1 ?
        ADD #SIZE, R1; R1←SIZE+c (R1)
        DEC R0; R0←c (R0)−1
    WHEND
    ADD #START−1, R1; START=START ADDRESS OF THIS ARRAY
    RETURN 〈R1〉; R1←START+SIZE×(c (R0)−1)+c (R1)−1
MAPEND
```

Fig. 5 Example of address mapping definition

```
a) IF R0 NE 〈BKNAME 〈#1000, FIELD1〉〉
b) MOVE 〈BKNAME 〈R0, WORD2〉〉, 〈R1 〈0, 10〉〉
c) AND #PATTERN, 〈ARRAY 〈〈R2, #1〉, LOW〉〉
```

Fig. 6 Some references to data structures

3. 翻訳処理

この言語は，先にも述べたようにアセンブリ言語とマクロの複合体であり，アセンブラによって機械語に変換される．システムにはあらかじめマクロ定義を集めたファイルが用意されており，アセンブラはこのファイルの存在を知っている．先に示したマクロはすべてマクロ名とそれに続く実引数という形になっており，これがマクロコールになる．アセンブラはマクロ名を手掛かりにしてシステムのマクロ定義を参照し，それに従ってマクロ展開を行なう．マクロ定義では，実引数を受け取ると適当なコードを発生するように定義されており，このようにして各マクロコールはその機能を果すためのコード群に置き換えられて行く．

このマクロによる翻訳には，マクロアセンブラがいくつかの機能を備えていることが必要である．以下に今回 ML-11 の製作に必要とした機能を示す．

1) 変数をマクロ展開時に作れ，かつのちに行なう別のマクロ展開時にその変数を参照できる．
2) 名以外に値でマクロコールできる．
3) マクロ展開がつぎに示す諸条件で制御できる．
   a) 変数に代入されている値の大小．
   b) ある記号列が変数として定義済みかどうか．
   c) 仮引数に対応する実引数が与えられたか．
   d) 二つの記号列が同一かどうか．
4) 文字列の連結が可能．
5) 複数の実引数や文字列をまとめて一つの引数として扱い，また逆に，この引数からもとの実引数や文字列を個別に取り出すことができる．
6) マクロ定義中に，再帰も許してマクロコールがあってよい．
7) マクロ定義中に他のマクロ定義があってよい．
8) マクロ展開時に文法のエラーをユーザに知らせる手段がある．

なお実用上の点から，マクロ展開された結果をリスティングファイルに出すか否かを，アセンブル時に制御できることも重要である．

つぎにマクロにより翻訳処理を行なうときの具体的な問題を考えてみよう．本来マクロ展開はそれぞれ独立に行なわれるものであるが，完全に独立であればML-11 の翻訳はできない．すなわち，それぞれのマ

* ワード名は，アセンブリ言語で許された1語を示すシンボル．
** i, j はフィールドのビット位置とビット長を示す整数．

```
IF A EQ B              CMP A,B; compare
   IF C GT D           BNE .1; jump to ELSE if A≠B
     ·
     ·〔α〕                CMP C,D
     ·                   BLE .2; skip 〔α〕 if C≤D
   IFEND                   ·
     ·                     ·〔α〕
     ·〔β〕                   ·
     ·                 .2:
ELSE                       ·
  ·                        ·〔β〕
  ·〔γ〕                     ·
  ·                    BR .3; skip else clause
IFEND
                  .1:    ·
                         ·〔γ〕
                         ·
                    .3:
(a) before expansion    (b) after expansion
```

Fig. 7 Examples of macro expansion

クロ展開同志がいくつかの情報をやり取りする必要がある．そこで，ML-11 に関してどのような情報がどのようにして異なるマクロ展開の間で受け渡しされるかについて述べる．ただし，マクロ機能でこのような情報の受け渡しをするには，送る側のマクロ展開時にこの情報を貯えたマクロを定義し，受ける側がこのマクロをコールするか，あるいは送る側がある変数を用意しそれに値を代入して受ける側が参照するか，この二つの方法しかないことを注意しておく．つまり情報は“前に”しか送れず，しかも制限した形でしか送れない．

3.1 実行制御文のネストの処理

判定条件を持つ実行制御文を翻訳するには，論理判定とその結果によりコントロールを変えるためのブランチ命令を発生する必要がある．論理判定は，比較命令のオペランドに実引数を組み込めばよい（**Fig. 7**参照）．ブランチ命令では，コントロールの飛び先を示すラベルがいるが，対応するマクロ同志が同じラベルを発生しなければならない．Fig. 7 において，最初の IF で発生されるブランチ命令の飛び先は ELSE でラベルとして発生されていなければならない．また，同種のマクロ（Fig. 7 では IF）でも発生すべきラベルは異なる．

そこで，まずラベルとして **.n**（n は5桁までの数で．と $ はアルファベットに属す）なる形を用いることにし，ユーザにはこの形の変数を使用禁止にする．あとは，各マクロ展開時に n をいくらにすればよいか知ることができればよい．そのため，いわゆるスタックの働きをするマクロが作られた．それは，**$n** なる名前をもつマクロで，ラベルが必要となるごとに新しい n を割り当てその値を記憶する（プッシュダウン）．Fig. 7 では，最初の IF に対し n=1 を与えこの値をプッシュダウンする．つぎの IF では n=2 とし，同じことを行なう．そのあとの IFEND では，ポップアップにより n=2 を得て **.2** なるラベルを発生し，ELSE では，ポップアップにより **.1** なるラベルを発生できる．なお n を記憶する際実行制御文の種類も同時に記憶すれば，構文の誤りも検出できる．

3.2 データ構造の処理

データ構造の宣言を行なうマクロ（DCLB など）はコールされると **$$n** なるマクロを作るだけで，命令コードは発生しない．一方，データ構造の参照を行なうマクロでは参照に使われたシンボルを実引数として，n=1 より順に **$$n** をコールする．**$$n** は条件展開により仮引数が宣言されたシンボルと同じとき以下の値をある変数に返すが，異なるときはなにもしない．

1) どの種類のデータ構造かを表わすコード．
2) そのシンボルが表わすフィールドの位置や，それが含まれる語の先頭番地からの距離．

もし，上記の値が返されないときは次の **$$n** をコールする．値が返されればそれを見て適当なコードを発生する．

たとえば，Fig. 4 の“$ WORD1”ではブロック名が“BKNAME”でフィールド名が“WORD1”のとき，ある変数に 1),2)の値を代入するアセンブラへの擬似命令を行なうマクロ **$$1** が作られる．“$ WORD-2”も同様なマクロ **$$3** を作る．その後で Fig. 6(b) が現われると，MOVE なるマクロはコード発生に先だち“BKNAME”と“WORD2”を持って **$$1** から順にコールする．**$$3** だけが前述の値を返す．このようにして“WORD2”が R0 の値を先頭番地とするブロックの第3番目の語であることが分かる．第2オペランドは該当する **$$n** がないので直接フィールドが指定されたことが分かる．この後はじめて実際のコードが発生される．

各シンボルは一意でなければならないが，n の決め方を変更すればブロック内のフィールド名の有効範囲をそのブロックに限ること，すなわち異なるブロック内で同じフィールド名を使うこともできる．また，未定義や二重定義の検出もマクロ機能を用いて行なえる．

```
DCLB BK; STRUCTUE OF THE LISTS
   $ V; DATA STORED
   $ P; LINK POINTER
   FILLER
      $$ F <0,3>; FLAG
DCLEND
   .
   .
   .
MOVE #START, R1; INITIALIZE
WHILE R1 NE #0; END OF LIST?
   IF <BK <R1,F>> EQ <STATUS <3,3>>
      MOVE <BK <R1,V>>, R0
      EXIT 2; EXIT WHILE LOOP
   ELSE
      MOVE <BK <R1,P>>, R1; NEXT BLOCK
   IFEND
WHEND
```

Fig. 8 Example of ML-11 program

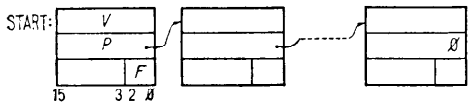

Fig. 9 Structure of linked lists

4. 例　　題

実際のプログラム例を **Fig. 8** に示す．ただし重要でない部分は省略する．このプログラムは，**Fig. 9** の構造を持つ主記憶領域上のリンクつきリストに対し，Fフィールドに STATUS 番地の〈3,3〉フィールドの値と同じ値をもつリストがあるか否かを先頭のリストから調べて行く，もし該当するものがあれば同じリストのVフィールドの値を R0 に転送し，最後まで調べても求めるリストがない場合は R0 に 0 を代入して終る．

5. 結果と問題点

ML-11 の設計製作と使用経験について述べる．

5.1 オブジェクトプログラムの効率

効率についてとくに細かい配慮を要する部分はアセンブリ言語を主に用いればよいが，ここで導入されたマクロもできるだけ効率の良いコードを発生するように作られている．PDP-11 では，前後 128 語の範囲内で飛ぶブランチ命令と任意のアドレスに飛べるジャンプ命令がある．このときつぎの問題が起こる．Fig. 7 の最初の IF に対する翻訳は ELSE までの命令数がつかめないので，**Fig. 10**(a) の形にせざるをえない．しかし，飛び先が上記の範囲なら Fig. 10(b) の形（2語コードが短かく計算時間も有利）にできる．この問題の一つの解決として ML-11 では，pseudo operation 用のマクロを用意しユーザが，Fig. 10 (a), (b) のどちらにするか指定できるようにした．他にもこの種の配慮がいくつかなされている．

```
CMP A, B              CMP A, B
BEQ .+6; skip next    BNE .1
JMP .1
   (a)                   (b)
```

Fig. 10 Macro expansion of "IF A EQ B"

5.2 マクロによる ML-11 の設計製作

マクロでは数式処理が不可能に近いため，ML-11 には数式がない．しかし，数式の少ないシステムプログラムでは実用上大きな欠点にはならないと思われる．また，ソースプログラム中アセンブリ言語で書かれた部分をマクロ側から参照できないため，文法のミスの検出が完全でない．

なお，ML-11 の製作中，MACRO-11 につぎのような機能がないことに不便を感じた．

1) 文字列を任意に分解する．
2) 文字列をシンボルで扱う．
3) 先に行なったマクロ展開を後で修正する．

5.3 使用経験と実用上の問題

ML-11 を用いて実際にプログラムを行なった2，3の例（データ・ベース管理プログラムの一部のモジュール，SOAP：ソースリスティングの形式を整えて出力するプログラム）の経験では，論理構造がみやすく論理的なミスは非常に少なく，デバッグも容易で比較的短時間でプログラムが完成した．しかしつぎのような問題も明らかとなった．マクロによる翻訳は作業用記憶領域をかなり要し，コアが少ないミニコンでは少し大きなプログラムに対しアセンブル時に支障をきたす．また処理速度も落ちる．重大な文法ミスが検出されたときは，マクロによるエラーメッセージが与えられるが，その他の多くはアセンブラに対する矛盾となりアセンブラによるエラーメッセージになる．また，このミスが他のマクロに影響して，付随的な文法ミスとなり，ユーザがエラーメッセージの本意をつかみにくいことがある．

6. む　す　び

ML-11 は設計，コーディング，デバッグを含めて途中何回か仕様の変更があったにもかかわらず約 60 人日で一応の完成を得た．その後も，修正や改良を加えているが，マクロ機能の利用はこの作業を容易なものにしている．しかし，前章に述べたいくつかの問題の中には，マクロ機能を利用する限りは解決不可能な

ものもある．そこで，仕様が安定した段階でコンパイラを作ることも検討している．

ML-11 ではマクロの多重使用が許されているので，bottom up/top down プログラミングもある程度実行できる．structured programming については多くの議論がなされているが，ML-11 を用いて実際に実験することも一つの課題となろう．

実用上の機能として，報告機能，デバッグ機能の向上なども検討中である．

なお，ML-11 の作成後に，この言語と共通したねらいを持って作られた言語 AL[7] の存在を知ったが，両者の相異点はつぎの通りである．

1) AL はフリーフォーマットであり，プリプロセッサが必要であるが，ML-11 はマクロ仕様に完全に従っているため，現存のマクロアセンブラのみで翻訳処理できる．
2) AL にもビット単位の操作はあるが，ML-11 はさらにブロックやアドレスマッピングなどのデータ構造を表現できる．
3) ML-11 では，効率と見やすさに対する種々の配慮がなされている．

また，現存のアセンブリ言語がそのまま使えること，マクロ定義を変更するだけでユーザが言語仕様を修正できることが PL 360[8] などの言語と異なっている．

最後に，日頃御指導いただく嵩忠雄教授，御討論いただいた嵩研究室諸氏に厚く感謝いたします．

参 考 文 献

1) O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare: Structured Programming, Academic press, London (1972).
2) D. E. Knuth: Structured Programming with Go To Statements, Stanford CS report, 74-416 (1974).
3) W. A. Walf, D. B. Russell, and A. N. Habermann: BLISS: A Language for Systems Programming, Comm. ACM, Vol. 14, No. 12, pp. 780～790 (1971).
4) P. J. Brown: Using a Macro Processor to Aid Software Implementation, Comp. J., Vol. 12, No. 4, pp. 327～331 (1969).
5) P. J. Brown: A Survey of Macro Processors, Annual Review in Automatic Programming, Vol. 6, pt. 2, pp. 37～87 (1969).
6) 林達也：システム設計言語 DEAPLAN について，情報処理，Vol. 14, No. 9, pp. 652～660 (1973).
7) E. C. Haines: A Structured Assembly Language, SIGPLAN Notices, Vol. 8, No. 1, pp. 15～20 (1973).
8) N. Wirth: PL 360, A Programming Language for the 360 Computers, J. Association for Computing Machinery, Vol. 15, No. 1, pp. 37～74 (1968).
9) BLISS-11 Programmer's Manual, DEC (1972).
10) BATCH-11/DOS-11 Assembler(MACRO-11), DEC (1973).

(昭和 49 年 10 月 14 日受付)
(昭和 50 年 1 月 28 日再受付)