資　料

マイクロプログラムの自動作成について*

馬場敬信** 萩原　宏***

Abstract

A Microprogram Generator System (MPG) is a processor for Microprogram Generative Language (MPGL). MPG provides the facility of generating efficient object microprograms from higher level microprogramming language in a machine-independent fashion.

This paper evaluates MPG and MPGL from several aspects: efficiency of object microprograms, generality and naturalness of MPGL, correctness, effectiveness, and machine independence of MPG procedure.

For this purpose, experimentations are made for different types of machines and microprograms.

Two target machines, HITAC 8350 and HP 2100A, were chosen, in that HITAC 8350 has horizontal type microinstructions whereas HP 2100A has vertical ones.

For each machine, four microprograms were written: 1) Positive integer multiplication 2) Square root 3) Conversion of arithmetic expression to reverse polish notation 4) Elementary function evaluation algorithm, CORDIC.

Each microprogram was compiled and logically simulated by MPG. The experimental results show that MPG is applicable to different types of machines and that, if coded carefully, object microprograms are as efficient as hand-coded microprograms.

1. まえがき

近年，書換え可能な高速の IC 記憶が制御記憶として使用できるようになり，いわゆるダイナミック・マイクロプログラミングが実用化されている．それに伴い，マイクロプログラム（以下，μP）を効率良く作成するために，計算機による作成の自動化が試みられている[9]．

しかし，一般のソフトウェアにおいてコンパイラ言語が使用されているように，μP コンパイラがいまだ実用化されないのは，

（1） オブジェクト・μP の効率が低下する．

（2） μP の記述はハードウェアに依存する割合が大きいため，汎用性のある記述言語の設計，およびその処理システムの作成が困難である．

などの理由によるものと思われる．

これらは μP の自動作成を実現するための重要な課題であるが，実際に処理システムを作成し，これらの観点から評価を行うことは，ほとんどなされていない．このため，筆者らは先に報告した μP 記述言語－MPGL－[1] およびその処理システム，マイクロプログラム・ジェネレーター MPG－[2),3)] を使用して μP を作成し，その結果について検討を加えた．

実験に当っては，前述の（1），（2）の点について MPG の有効性を明らかにするため，次の各点に特に留意した．

（1） 記述対象とする計算機として，水平型マイクロ命令をもつ HITAC 8350 と垂直型マイクロ命令をもつ HP 2100A という，異なるタイプの実在の計算機を選んだ．

（2） 記述する μP は，特徴のあるものをいくつか選択した．算術演算として，乗算，開平算，お

* On Automatic Microprogramming by Takanobu BABA (University of Electro-communications) and Hiroshi HAGIWARA (Faculty of Engineering, Kyoto University)
** 電気通信大学電気通信学部
*** 京都大学工学部

よび初等関数を求める CORDIC[8] を選んだ．また，非算術演算として，算術式の逆ポーランド記法への変換を選んだ．

（3） HITAC 8350 には，μP の処理システムとして，RCM 処理システムがある[4]．このシステムは HITAC 8350 専用のマイクロアセンブラとシミュレータ（SIM 350 と呼ばれる）から構成されているが，これを用いて同じアルゴリズムに対して μP を作成し，MPG により作成した μP と効率の比較を行った．

以下，本稿ではまず，**2.** と **3.** で計算機と μP の記述結果を要約する．次に，**4.** と **5.** で処理結果について述べ，MPG マイクロプログラム・コンパイラによって手書きの μP に匹敵する効率をもつ μP が作成できること，および，MPG マイクロプログラム・シミュレータに実現された手法が μP 記述言語の高レベル化に有効であることを示す．

2. 計算機の記述

前述のように，計算機として HITAC 8350（以下，マシンAと記す）と HP 2100 A（以下，マシンBと記す）を選び，MPGL のマシン記述部（MDS）を用いて記述した．マシンAのマイクロ命令は，1語 60 ビットで 15 のフィールドから成り，水平型である．また，マシンBのマイクロ命令は，1語 24 ビットで6つのフィールドから成り，垂直型である．

次に，実際に記述とその処理を行うことにより明らかになった事柄について述べる．

2.1 マシン記述部の仕様について

マシン記述部の仕様で特にドキュメント性や記述能力の向上に効果があったと考えられる特徴を列挙する．

（1） 制御記述による記述能力の向上

制御記述は，1つのマイクロ操作の実行を制御するマイクロオーダ（一般に複数個）間の関係を，マイクロオーダ・ネモニックの論理式によって記述する．これによって，広範な計算機における間接符号化方式の記述やハードウェアに起因して生じるマイクロプログラミング上の制約*の記述が行えた．

（2） 間接機能制御の記述が可能

特定のレジスタ等の値によって，間接的に，分岐先の番地や，転送のビット位置，あるいは算術論理演算装置の動作が決定されるマイクロ操作を記述できる．これは，後述するように μP の特徴を生かした処理の高速化に役立った．

（3） テーブル形式仕様

言語仕様はテーブル形式を採用し，アルゴリズム記述部で使用される変数，定数，演算子を見出しとして，それぞれの属性を並べる．これによって，アルゴリズム記述部で μP を記述する際に参照しやすく，また，システム作成の立場からは言語仕様の変更に対処しやすいという利点を生じた．

（4） 幅広い ALU の記述が可能

ALU の記述は，シフト，論理演算，加減算など 18 種類の基本的な演算を記述要素として組合せることにより行う．これによって，計算機ごとに異なる演算機能を有する ALU を幅広く記述することが可能となった．

また，記述の結果，垂直型のマシンBのように間接符号化が複雑な場合には，これを制御記述の中に記述する際にフィールド間のマイクロオーダの関係に特に注意を要することが明らかとなった．

2.2 MDS トランスレータの処理について

マシン記述部は MDS トランスレータによって内部形式に変換される．この処理時間は **Table 1** に示す通りで，マシン記述部の仕様が内部テーブル形式に近いテーブル形式であるため，後述するコンパイラの処理時間と比較すると，かなり小さな値である．

Table 1 Sample translation.

MDS	HITAC 8350	HP 2100 A
ソースステートメント数	438	231
処理時間*	19 5	10 3
メモリ所要量（バイト）	15000	9300

* 処理時間の単位は秒である．また，上段は入出力を含む全所要時間，下段は CPU の動作時間である．

また，処理後の内部テーブルは，コンパイラとシミュレータによって共通に使用されるデータベースとなるが，記述対象とする計算機ごとに各部分テーブル[3]の大きさが異なるため，余分なメモリ・スペースを必要とした．これを改良するため，MDS トランスレータが不要な部分を削除して外部記憶に掃き出し，コンパイラとシミュレータがこれを読み込む時に部分テーブル間のポインタの修正処理を行う方法を採用した．

この結果，読み込み時に必要とされるメモリ・スペ

* 例えば「あるカウンタの内容の増加」と「そのカウンタへのデータ・パスからの値のセット」の2つのマイクロ操作が，同じマシンサイクルでは行えないとき，両方のマイクロ操作を制御するマイクロオーダは同一のマイクロ命令には指定できない，などの制約を生じる．

ースは Table 1 に示すように改善された*.

3. マイクロプログラムの記述

1. で述べたように，4種類のアルゴリズムを選び，MPGL のアルゴリズム記述部 (ADS) を用いて記述した．この結果，**3.1** から **3.3** に述べるように，計算機のハードウェア・レベルでの特徴を生かしたプログラムの作成が行えることが確かめられた．その主なものを列挙すると，

(a) 倍長演算用機能の利用 (**3.2**)

(b) カウンタとスクラッチパッド・メモリによるスタックの実現 (**3.3**)

(c) 符号のテスト用機能や論理演算用機能の利用 (**3.3**)

(d) 機能分岐を利用した多方向分岐 (**3.3**)

などである．次に，マシンAとBの乗算，およびマシンAの平方根と逆ポーランド記法への変換について具体的に述べ，上記の特徴を明らかにする**.

3.1 乗　算

乗数，被乗数ともに正整数と仮定して μP の作成を行った．

マシンAの μP を **Fig. 1** に示す．スクラッチパッド・メモリ SPMO (X″10″) と SPMO (X″11″) にそれぞれ乗数と被乗数があるものとして乗算を行う．カウンタGは桁数の計数に使用し，初期値 $(20)_{16}$ を μP によりセットする．積は SPMO (X″10″) と SPMO (X″11″) に置く．

```
ST-NO.      SOURCE   STATEMENT
            ADS MULT
     1       AVAIL SPM0(X"39") ;
     2       EXTRN ALG END (X"A20") ;
     3        ALG MULT (X"A50") ;
     4              G := XL8"20" ;
     5              U := XL32"0" ;
     6              L := XL32"0" ;
     7        L0 :  SPM0(X"10") := <SRLA> SPM0(X"10") ;
     8              IF SC'0' = BL1"0" THEN GOTO L1 ;
     9              U := U <+>  SPM0(X"11") ;
    10        L1 :  U := <SRL>  U ;
    11              L := <SRC> L ;
    12              > G ;
    13              IF G¬= XL8"0" THEN GOTO L0 ;
    14              SPM0(X"10") := U ;
    15              SPM0(X"11") := L ;
    16              GOTO END ;
    17       GLA;
           SDA;
```

Fig. 1 Multiplication microprogram for HITAC 8350.

マシンBの μP を **Fig. 2** に示す．レジスタBとSP1にそれぞれ乗数と被乗数があるものとして乗算を行い，積をレジスタFとQに残す．

```
ST-NO.      SOURCE   STATEMENT
            ADS MULT
     1       EXTRN ALG END (X"300") ;
            ***  MULTIPLICATION
     2       ALG MULT (X"200") ;
     3              F := XL16"0" ;
     4              Q := XL16"0" ;
     5              C'1:4' := DL4"0" ;
     6        LOOP :  B := <RRF> B ;
     7              IF FLG = BL1"0" THEN GOTO L0 ;
     8              F := SP1 <+> F ;
     9        L0 :  FLG <- 0 ;
    10              F := <RRF> F ;
    11              Q := <RRF> Q ;
    12              < C ;
    13              IF C'1:4' ¬= DL4"15" THEN GOTO L1 ;
    14              GOTO L2 ;
    15        L1 :  GOTO LOOP;
    16        L2 :  FLG <- 0 ;
    17              F := <RRF> F ;
    18              Q := <RRF> Q ;
    19              GOTO END ;
    20       GLA ;
           SDA;
```

Fig. 2 Multiplication microprogram for HP 2100A.

3.2 平 方 根

正整数を入力として，その平方根を引き戻し法で求める μP を作成した．

マシンAに対しては，倍長語の平方根を求める μP を作成した (**Fig. 3**(次頁参照))．入力の正整数 x の記憶番地が SPMO (X″10″) に入っているとする．主記憶 MM は4バイトで1語のため，最初の4バイトに x の前半を，次の4バイトに後半を置く．桁数はカウンタGに入れ，結果は SPMO (X″11″) の内容を番地として MM に書込む (ST. NO. 38)．また，剰余は SPMO (X″50″) と SPMO (X″51″) に残る．

ST. NO. 42 と 43 では，演算子 〈SR 2 UA〉 の2ビットシフトにより右側に落ちる2ビットを次の 〈SR 2 LA〉 のシフトで左からの入力とすることにより，2ビットずつの倍長シフトを行っている．また，倍長の加減算は，下位のデータの加減算による MSB からの桁上げを，上位のデータの加減算のイニシャルキャリとすることにより行う．(ST. NO. 24 と 26 が倍長の減算，ST. NO. 33 と 35 が倍長の加算である.)

3.3 算術式の逆ポーランド記法への変換

アルファベット1文字からなる変数と，区切り記号として +, −, *, ／[加減乗除], #[べき], (,)[括弧], および, ;[終了記号] を用いた算術式を逆ポーランド記法に変換する．区切り記号の優先順位は，# が 4, * と ／ は 3, + と − は 2, (は1とし,) によってスタックの内容を (まで出力する．また, ; によってスタックの内容を全部出力する．

マシンAの μP を **Fig. 4**(65頁参照) に示した．1語4バイトの先頭に，EBCDIK コードの区切り記号か変数を置く．入力式の先頭番地は SPMO (X″30″) にあるものとし，逆ポーランド形式に変換後の式の格納開始番地が SPMO (X″31″) にあるものとする．

---

* 改良する前は，内部テーブル作成時のメモリ・スペース 19000 バイトをそのまま必要とした．

** μP のアルゴリズムに関しては，京都大学 渡辺勝正助教授，小柳滋氏 (現在 東芝総合研究所) の協力を得た．

```
ST-NO.    SOURCE   STATEMENT
          ADS ROOT
     1     EXTRN ALG EI (X"953") ;
     2      ALG ROOT (X"800") ;
          ** FETCH OPERAND
     3             SPM0(X"50") := MM(SPM0(X"10")'8:30') ;
     4             SPM0(X"10") := SPM0(X"10") <+(C)> XL32"4" ;
     5             SPM0(X"51") := <SLL> MM(SPM0(X"10")'8:30') ;
          ** INITIAL SET
     6             SPM0(X"18") := XL32"0" ;
     7             SPM0(X"19") := XL32"0" ;
     8             SPM0(X"30") := XL32"0" ;
     9             SPM0(X"30")'0:7' := XL8"20" ;
    10             SPM0(X"31") := XL32"0" ;
    11             SPM0(X"38") := XL32"0" ;
    12             SPM0(X"39") := XL32"0" ;
    13             SPM0(X"58") := XL32"0" ;
    14              G := XL8"1F" ;
    15      LOOP : L := SPM0(X"31") ;
    16             L := SPM0(X"39") <+>  L ;
    17             SPM0(X"19") := L ;
    18             U := SPM0(X"30") ;
    19             U := SPM0(X"38") <+(C)> U ;
    20             SPM0(X"18") := U ;
    21             SPM0(X"38") := <SRLA> SPM0(X"38") ;
    22             SPM0(X"39") := <SRCA> SPM0(X"39") ;
    23             L := SPM0(X"19") ;
    24             L := SPM0(X"51") <-C> L ;
    25             U := SPM0(X"18") ;
    26             U := SPM0(X"50") <-(C)> U ;
    27             S1 <- U'0' =BL1"1" ;
    28             IF S1=BL1"1" THEN GOTO TEST ;
    29             SPM0(X"50") := U ;
    30             SPM0(X"51") := L ;
    31             SPM0(X"58") := SPM0(X"58") <+>  XL32"1" ;
    32             U := SPM0(X"31") ;
    33             SPM0(X"39") := SPM0(X"39") <+>  U ;
    34             U := SPM0(X"30") ;
    35             SPM0(X"38") := SPM0(X"38") <+(C)> U ;
    36      TEST : > G ;
    37             IF G¬=XL8"0" THEN GOTO NEXT ;
    38             MM(SPM0(X"11")'8:30') := SPM0(X"58") ;
    39             SPM0(X"51") := <SRLA> SPM0(X"51") ;
    40             GOTO EI ;
    41      NEXT : SPM0(X"58") := <SLLA> SPM0(X"58") ;
    42             SPM0(X"30") := <SR2UA> SPM0(X"30") ;
    43             SPM0(X"31") := <SR2LA> SPM0(X"31") ;
    44             GOTO LOOP ;
    45       GLA;
          SDA;
```

Fig. 3 Square root microprogram for HITAC 8350.

スタックは SPMO 内の X''00'' 番地からを使用し，カウンタGをそのポインタとする．

まず，MM より1語読んで，それが変数か区切り記号かをテストする．EBCDIK コードの場合これは MSB の値によって決まるので，符号のテスト用フリップ・フロップ S1 を用いて行える (ST. NO. 15, 16)．変数の場合はそのまま出力する (ST. NO. 17)．区切り記号の場合は各区切り記号の値により機能分岐を行い (ST. NO. 21)，優先順位をセットした後，スタックの処理を行う．スタックのプッシュダウンはGのデクリメントによって行う(ST. NO. 33, 35など)．

4. コンパイラの処理

3. で述べた μP を MPG によりコンパイルし，オブジェクト・μP のシミュレーションを行った．また，マシンAについては RCM 処理システムを用いてできるだけ効率の良い μP を作成し，比較の対象とした．

* アベイラブル・レジスタ，およびセグメントの概念については文献3) に述べた．

** 例えば，マシンAの逆ポーランドでは6回，マシンBの逆ポーランドでは12回の主記憶の読み書きが記述されている．

これらの結果を **Table 2, 3**(66頁参照) に示す．

以下，この実験結果に基づき，各フェーズの処理時間，処理アルゴリズムの有効性，計算機の相違による影響などについて述べる．

4.1 フェーズごとの評価[3)]

4.1.1 構文解析と最適化前処理（フェーズ I, II)

処理時間は，ほぼソース文の数に依存する．また，フェーズI，IIでは計算機の記述に依存する処理が少ないため，処理時間にもターゲット・マシンの相違による影響は見られない．

最適化前処理では，代入文の左辺のレジスタあるいは，スクラッチパッド・メモリがアベイラブル・レジスタ*となるため，その数は比較的多い．しかし，セグメント*内で見出す現在の方法では利用するのに十分な範囲が得られない事が多く，セグメント間の動的な関係をも考慮した方法によってこの範囲を広げることが有効と考えられる．

4.1.2 マイクロ命令の構成（フェーズIII)

マイクロ命令の構成は構文解析の結果出力される四項系列ごとに行われるため，処理時間はほぼ四項系列数に依存する．

また，本フェーズの目標はよりコンパクトなマイクロ命令を構成することであったが，Table 2 において，実行文の数とフェーズIIIの出力マイクロ命令数を比較することにより，最適化の効果が出力マイクロ命令数の減少となって表われていることが判る．ただ，マシン A, B とも主記憶の読み書きには2つのマイクロ命令を要すること，さらにマシンBでは，後述するように一時記憶のためにレジスタへの転送が補われること，などから最適な構成を行っても出力マイクロ命令数が増加する場合もある**．

また，マシン A, B の相違については，水平型のマシンAでは並列化の効果が大きく，例えば Fig. 4 の ST. NO. 32, 33, 34 などは1つのマイクロ命令に構成される．これに対して，垂直型のマシンBでは出力マイクロ命令数の減少の割合は少ない．これは，マイクロ命令の符号化の程度を進めた場合，並列化の効果が少なくなることを裏付けている．

また，アベイラブル・レジスタの割付けは特にマシンBで有効に行われた．例えば，マシンBの逆ポーランドの μP に表われる代入文，

SP 3 ≔ SP 3 〈+〉 SP 4;

```
ST-NO.    SOURCE   STATEMENT
          ADS REVPOL
          ** SPM0(X"30") : START ADDRESS OF EXPRESSION
          ** SPM0(X"31") : START ADDRESS OF REVERSE POLISH FORM EXPRESSION
          ** SPM0(X"38") : ONE WORD FETCHED FROM MAIN MEMORY
          ** SPM0(X"00"),(X"01"), . . , (X"0F") : STACK
          ** G : STACK POINTER (COUNTER)
          ** SPM0(X"50") : WORK
          ** U : WORK REG.
          ** L : WORK REG.
     1     ADDR LFTP (X"94D") ;
     2     ADDR ADD (X"94E") ;
     3     ADDR MULT (X"95C") ;
     4     ADDR RITP (X"950") ;
     5     ADDR SUB (X"960") ;
     6     ADDR DIV (X"961") ;
     7     ADDR EXP (X"97B") ;
     8     ADDR SEMC (X"95E") ;
     9     EXTRN ALG END (X"853") ;
    10     AVAIL U ;
    11     AVAIL SPM0(X"50") ;
    12      ALG REVP (X"94C") ;
          ** READ ONE WORD
    13            SPM0(X"38") := MM(SPM0(X"30")'8:29') ;
          ** TEST THE MOST SIGNIFICANT BIT
    14            L := SPM0(X"38") ;
    15            S1 <- L'0' = BL1"1" ;
    16            IF S1 = BL1"0" THEN GOTO DLMT ;
          ** OUTPUT VARIABLE
    17            MM(SPM0(X"31")'8:29') := SPM0(X"38") ;
    18            SPM0(X"31") := SPM0(X"31") <+>    XL32"4" ;
    19            GOTO INC ;
          ** BRANCH ACCORDING TO DELIMITER
    20      DLMT : FR'0:7' := SPM0(X"38")'0:7' ;
    21            CASE FR'0:7'      (X"4D") LFTP,(X"4E") ADD,(X"5C") MULT,
                  (X"5D") RITP,(X"5E") SEMC, (X"60") SUB,(X"61") DIV,
                  (X"7B") EXP ;
          ** '(' **
    22      LFTP : SPM0(X"38") := SPM0(X"38") <+>    XL32"1" ;
    23            GOTO PUSH ;
          ** '+' **
    24      ADD :  SPM0(X"38") := SPM0(X"38") <+>    XL32"2" ;
    25            GOTO STAC ;
          ** '*' **
    26      MULT : SPM0(X"38") := SPM0(X"38") <+>    XL32"3" ;
    27            GOTO STAC ;
          ** ')' **
    28      RITP : FR'8:15':=SPM0(G'1:7')'24:31' ;
    29            > FR'8:15' ;
    30            IF FR'8:15' =XL8"0" THEN GOTO DECG ;
    31           /* FR'8:15' =XL8"1" ? */
                  MM(SPM0(X"31")'8:29') := SPM0(G'1:7') ;
    32            SPM0(X"31") := SPM0(X"31")<+>    XL32"4" ;
    33            > G ;
    34            GOTO RITP ;
    35      DECG : > G ;
    36            GOTO INC ;
          ** '-' **
    37      SUB :  SPM0(X"38") := SPM0(X"38") <+>    XL32"2" ;
    38            GOTO STAC ;
          ** '/' **
    39      DIV :  SPM0(X"38") := SPM0(X"38") <+>    XL32"3" ;
    40            GOTO STAC ;
          ** '↑' **
    41      EXP :  SPM0(X"38") := SPM0(X"38") <+>    XL32"4" ;
    42      STAC : IF G ¬= XL8"0" THEN GOTO COMP ;
          ** PUSH DOWN
    43      PUSH:  SPM0(X"51")'24:31' := G ;  /* G+1*/
                  SPM0(X"51") := SPM0(X"51") <+>    XL32"1" ;
    45            G := SPM0(X"51")'24:31' ;
    46            SPM0(G'1:7') := SPM0(X"38") ;
    47            GOTO INC ;

          ** COMPARE TOS PRIORITY WITH INPUT PRIORITY
    48      COMP : L := SPM0(G'1:7') <&> XL32"FF" ;
    49            U := SPM0(X"38") <&> XL32"FF" ;
    50            L := L<-C> U ;
    51            S1 <- L'0' =BL1"1" ;
    52            IF S1 = BL1"1" THEN GOTO PUSH ;
          ** POP UP
    53            MM(SPM0(X"31")'8:29') := SPM0(G'1:7') ;
    54            SPM0(X"31") := SPM0(X"31") <+>    XL32"4" ;
    55            > G ;
    56            GOTO STAC ;
          ** INCREMENT EXPRESSION POINTER
    57      INC :  SPM0(X"30") := SPM0(X"30") <+>    XL32"4" ;
    58            GOTO REVP ;
          ** ';' **
    59      SEMC : MM(SPM0(X"31")'8:29') := SPM0(G'1:7') ;
    60            SPM0(X"31") := SPM0(X"31") <+>    XL32"4" ;
    61            > G ;
    62            IF G ¬= XL8"0" THEN GOTO SEMC ;
    63            MM(SPM0(X"31")'8:29') := SPM0(X"38") ;
    64            GOTO END ;
    65      GLA;
          SDA;
```

Fig. 4 Reverse polish translation microprogram for HITAC 8350.

MM (Q'1 : 15') := MM (F'1 : 15');

などでは，アーキテクチャ上の制約から右辺の演算結果，あるいは主記憶から読み出されたデータを一時記憶に貯える必要がある．この一時記憶としてアベイラブル・レジスタが割付けられている．

4.1.3 制御記憶への割付け（フェーズIV）

コンパイラの処理時間の大半は制御記憶への割付けに費やされる．これはアドレス生成法を参照して割付けを行うための計算機に独立な手法が煩雑であることによると考えられる．そこで，筆者らは計算機に独立とするためのオーバヘッドを知るため，『MPGの処理プログラム中で次マイクロ命令アドレス生成法を参照して分岐可能範囲を出力する部分』だけをマシンA専用のものに置き換える実験を行った*．この結果，乗算，平方根，逆ポーランドの順に，4, 10, 14秒ずつのCPU時間の短縮という結果を得た．

割付けのアルゴリズムの有効性については，(1)マイクロ命令が可能な限り連続した領域に割付けられること (Table 2**) (2)種々の次マイクロ命令アドレス生成法に対して割付けが正しく行われること，などから有効であることが確かめられた．

マシンAとBを比較すると，マシンBの場合にダミー・マイクロ命令[3]が補われていることが特徴的である (Table 2)．これはマシンBのマイクロ命令が垂直型で，演算を指定するマイクロオーダと順序制御を行うマイクロオーダが同じフィールドに属するため，演算を行って分岐する場合には次の番地にしか分岐できないなど，分岐可能な番地が限定されていることによる．

割付け処理の例として，**Fig. 5** (67頁参照) に Fig. 4 の μP の割付けの結果を示す．図中，1〜2B の各節点は1つのマイクロ命令に対応し，矢印と B, BCT,…*** が分岐のし方を表す．各節点の左側にラベ

---

* 即ち，次マイクロ命令アドレス生成法に関する情報を予めプログラム中に組み込み，マシンAの記述を参照せずに番地付けを行う．

** マシン A, B 共に，逆ポーランド記法への変換 μP ではオブジェクト・マイクロ命令数以上の制御記憶領域を必要としている．これはマシンAの場合には EBCDIK コード，マシンBの場合には ASC II コードの値に応じて，予め固定された番地へ機能分岐するためである (Fig. 4 参照)．

*** B: 無条件分岐．BCT: テスト分岐 ((T) はテスト成立，(F) は不成立を表す)．BR: 機能分岐 ((4D), (4E),……は機能レジスタFR '0:7' の値を示す)．

Table 2 Compilation statistics.

MDS		HITAC 8350				HP 2100 A			
ADS		乗 算	平方根	逆ポーランド	CORDIC	乗 算	平方根	逆ポーランド	CORDIC
MPGマイクロプログラム・コンパイラ	ソース・ステートメント数	17	45	65	127	20	25	74	147
	実行文の数	13	42	52	99	17	21	62	123
	四項系列数	16	56	66	129	22	29	84	172
	セグメント数	5	6	19	40	7	7	34	62
	フェーズIIIの出力マイクロ命令数	10	42	43	66	14	20	79	119
	ダミー・マイクロ命令数	0	0	0	0	1	1	9	10
	オブジェクト・マイクロ命令数	10	42	43	66	15	21	88	129
	割付けに要した制御記憶の容量	10	42	48	66	15	21	96	129
	処理時間* フェーズ I, II	3 1	5 1	6 1	12 3	3 1	3 1	6 2	28 3
	フェーズ III	25 4	132 32	108 31	185 40	31 5	46 9	166 37	274 50
	フェーズ IV	41 13	95 35	111 43	472 239	58 19	77 26	483 274	797 447
	合 計	69 18	232 68	225 75	669 282	92 25	126 36	655 313	1099 500
RCMシステム処理	マイクロ命令数	10	40	39	55				
	処理時間*	71 4	84 4	90 4	99 5				

* 処理時間の単位は秒である．また，上段は入出力を含む全所要時間，下段は CPU の動作時間である．

Table 3 Simulation statistics.

MDS		HITAC 8350				HP 2100 A			
ADS		乗算	平方根	逆ポーランド	CORDIC	乗算	平方根	逆ポーランド	CORDIC
MPGマイクロプログラム・シミュレータ	実行マイクロ命令ステップ数	136	637	237	1154	112	84	352	419
	マイクロインストラクション・トレースモード 処理時間*	24 19	124 108	46 39	190 168	28 19	21 14	75 65	83 70
	実行セグメント数	69	100	85	523	48	30	133	104
	セグメント・トレースモード 処理時間*	27 21	130 114	64 45	236 187	33 21	22 15	85 68	88 72
S3 5M0	実行マイクロ命令ステップ数	136	575	187	941				
	処理時間*	50 6	167 13	70 7	258 17				

* 処理時間の単位は秒である．また，上段は入出力を含む全所要時間，下段は CPU の動作時間である．

ルを記し，右側に割付けられた番地を示す．番地の右肩の ＊ は ADS の宣言ブロックで予め宣言された番地であることを表す．

4.2 オブジェクト・マイクロプログラムの効率

Table 2 に示した結果は，MPG によるオブジェクト・マイクロ命令数が RCM 処理システムによるマイクロ命令数に十分近いことを示している．

さらに，両者のオブジェクト・リストを細かく比較検討することにより，効率低下の原因の1つがセグメント内でのローカルな最適化にあることが判った．

具体例をあげると，Fig. 3 で ST. NO. 36 の >G はこのままでは1つのマイクロ命令を構成する．これをセグメントを越えて ST. NO. 27 の次に置くと，>G は ST. NO. 27 に対するマイクロ命令の空いているフィールドに挿入されることになり，マイクロ命令数が1つ減少する．

また，マシン記述部に記述されたターゲット・マシンの機能を十分に活用しないで μP の記述を行った場合，あるいはレジスタ間の不必要な転送を行うなどの冗長な記述を行った場合には，やはり効率低下の原因となる．

しかしながら，このようなセグメント間にわたるグローバルなレベルでの最適化や記述上の問題は，ADS のステートメント・レベルで注意深くコーディングを

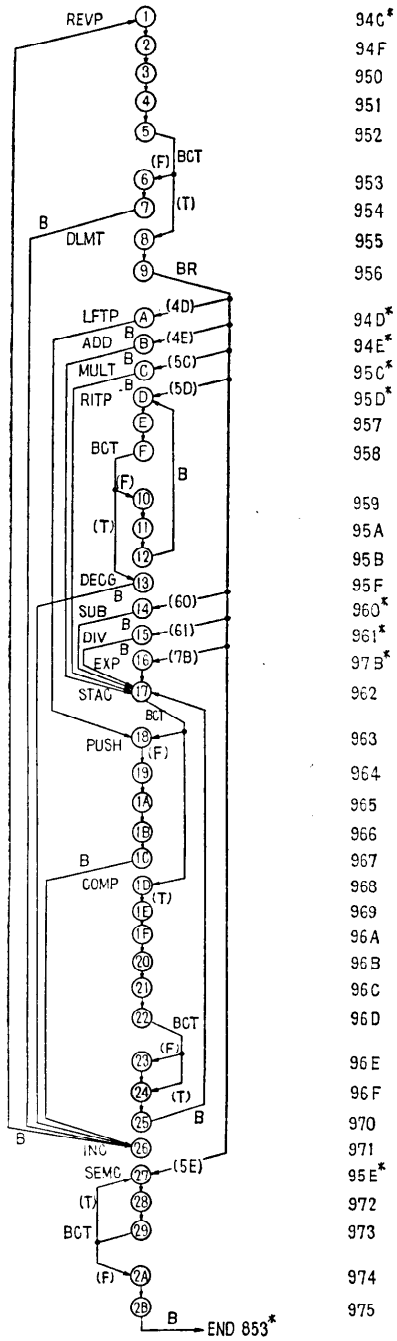

Fig. 5 Graph representation of addressed microinstructions.

行うことにより解決されるものである．

したがって，以上述べた実験結果は，十分注意して計算機の記述と μP の記述を行うことにより，手書きの μP に近い効率をもった μP が作成できることを実証している．

5. シミュレータの処理

MPG マイクロプログラム・シミュレータ[10]を用いてコンパイラのオブジェクト・μP をシミュレートし，その論理チェックを行うと共に，Table 3 に示す結果を得た．また，マシンAのマイクロアセンブラの出力を SIM 350 によってシミュレートした結果を比較のために示した．

MPG マイクロプログラム・シミュレータには2つのトレースモードが用意されている．1つはマイクロインストラクション・トレースと呼ばれ，SIM 350 と同様に各マイクロ命令実行ごとにユーザにより指定された変数の値を出力する．これに対して，セグメント・トレースでは各セグメントごとに「値を使用される変数」と「更新される変数」を出力する（例えば，Fig. 4 に対する **Fig. 6**(a)（次頁参照））．

セグメント・トレースは，コンパイラの出力するオブジェクト・μP のシミュレータであることから特に設けたものであるが，実際にコンパイラと共に使用した場合，マイクロインストラクション・トレースと比較して次のような利点のあることが明らかになった．

（1） ADS ステートメントとセグメントごとの出力との対応が付けやすい．

（2） プログラムの流れが変る事がセグメントごとに明らかにされるため追跡が行いやすい．

（3） トレースを行うべき変数を指定する必要がなく，また，トレース情報の冗長度が少ない．

特に MPG の場合には，コンパイラによってローカルな最適化と番地付けが行われることにより，ADS のステートメントとオブジェクト・マイクロ命令との対応付けが困難であるため，(1)と(2)の利点は極めて有効であった．また，(3)のトレース情報の冗長度については，各セグメント当りの実行マイクロ命令ステップ数が多いほど冗長度は減少すると予想されたが，マシンAの Fig. 3, 4 の μP に対する Fig. 6 の出力リストを比較すると，ステップ数の多い平方根の方がより有効であることが判る*．

6. むすび

マイクロプログラム記述言語（MPGL）を用いて実

* Tables 3 から，1つの実行セグメント当りの平均実行マイクロ命令ステップ数は，平方根 (Fig. 3) で約 6.4，逆ポーランド記法への変換 (Fig. 4) で約 2.8 である．

```
HITAC8350                    MICRO-PROGRAM  SIMULATION
                          REVERSE POLISH

SEGMENT TRACE MODE
        *** USED VARIABLE ***                    *              *** DEFINED VARIABLE ***
                                                 *
                                  *** SEGMENT.NO: 0001 ***
 SPM0(30) = 00000000    MM  (000000) = C1000000  *     SPM0(38) = C1000000      L    = C1000000
                                                 *
                                                 *          S1   = 01
                                                 *
                                  *** SEGMENT.NO: 0002 ***
 SPM0(31) = 00000100      SPM0(38) = C1000000     *          SC   = 00        SPM0(31) = 00000104
                                                 *
                                                 *  MM  (000100) = C1000000
                                                 *
                                  *** SEGMENT.NO: 0017 ***
 SPM0(30) = 00000000                             *          SC   = 00        SPM0(30) = 00000004
                                                 *
```

(a) Reverse polish translation microprogram for HITAC 8350.

```
HITAC8350                    MICRO-PROGRAM  SIMULATION
                          SQUARE ROOT

SEGMENT TRACE MODE
        *** USED VARIABLE ***                    *              *** DEFINED VARIABLE ***
                                                 *
                                  *** SEGMENT.NO: 0001 ***
  SPM0(10) = 00000500          SC   = 00         *     SC   = 00          SPM0(10) = 00000504
MM  (000500) = 00000000    MM  (000504) = 00320000 *  SPM0(50) = 00000000   SPM0(51) = 00640000
                                                 *
                                                 *  SPM0(18) = 00000000   SPM0(19) = 00000000
                                                 *
                                                 *  SPM0(30) = 20000000   SPM0(31) = 00000000
                                                 *
                                                 *  SPM0(38) = 00000000   SPM0(39) = 00000000
                                                 *
                                                 *  SPM0(58) = 00000000      G    = 1F
                                                 *
                                  *** SEGMENT.NO: 0002 ***
  SPM0(31) = 00000000      SPM0(39) = 00000000    *     L    = 00640000       SC   = 00
                                                 *
  SPM0(30) = 20000000      SPM0(38) = 00000000    *  SPM0(19) = 00000000      U    = E0000000
                                                 *
  SPM0(51) = 00640000      SPM0(50) = 00000000    *  SPM0(18) = 20000000   SPM0(38) = 00000000
                                                 *
                                                 *  SPM0(39) = 00000000      S1   = 01
                                                 *
```

(b) Square root microprogram for HITAC 8350.

Fig. 6 Simulation listing in the segment trace mode.

在の計算機 HITAC 8350 と HP 2100A の記述を行い，さらに，それぞれの計算機に対して4種類のマイクロプログラムを記述した．これをマイクロプログラム・ジェネレータ (MPG) を使用してコンパイルし，オブジェクト・マイクロプログラムのシミュレーションを行った．

これらの結果を検討することにより，筆者らの目標としたオブジェクト・マイクロプログラムの効率向上とシステムの汎用性について，作成したシステムが有効であることを示した．

なお，MPG は HITAC 8350 のアセンブラ言語を使用して作成されており，特に京大情報工学教室の HITAC 8350 ではオプションとして付けられた書換え可能制御記憶まで含めて使用可能である．

謝辞 有益な御意見を戴いた(株)富士通 藤本裕司氏に感謝する．

参 考 文 献

1) 馬場，萩原，藤本：マイクロプログラム記述言語：MPGL，情報処理，Vol. 18, No. 6, pp. 558～565 (1977)
2) 馬場，藤本，萩原，高橋，碇谷：マイクロプログラム・ジェネレータ(3)—MPG によるマイクロプログラムの作成—，情報処理学会第 17 回大会，No. 287 (1976)
3) 馬場，藤本，萩原：MPG マイクロプログラム・コンパイラ，情報処理，Vol. 19, No. 1, pp. 16～25 (1978)
4) 日立製作所：HITAC 8350 RCM 処理システム・マニュアル（マイクロ命令解説書，RCM 機構仕様書，RCM 用マイクロプログラム・シミュレータ・マニュアル）

5) Hewlett Packard Company: Model 2100 A Computer Mannual (Microprogramming Guide, Microprogramming Software, A Pocket Guide to the Hewlett-Packard 2100 A Computer)
6) S. S. Husson: Microprogramming: Principles and Practices, Prentice-Hall (1970)
7) マイクロプログラム・ジェネレータ仕様書（マイクロプログラム・ジェネレータ概説書，マシン記述部仕様書，アルゴリズム記述部仕様書，シミュレータ仕様書）
8) J. S. Walther: A Unified Algorithm for Elementary Functions, AFIPS Conf. Proc., SJCC, Vol. 38, pp. 378～385 (1971)
9) 萩原：マイクロプログラミングの発展，情報処理，Vol. 14, No. 6, pp. 374～378 (1973)
10) 馬場，萩原，藤本，高橋，碇谷：MPG マイクロプログラム・シミュレータ，情報処理(投稿中)

(昭和51年9月6日受付)
(昭和52年5月13日再受付)