論　文

マイクロプログラム記述言語：MPGL*

馬場敬信** 萩原　宏*** 藤本裕司****

Abstract

A machine-independent high level microprogramming language, called MPGL, has been developed. This language consists of a machine description section (MDS), and an algorithm description section (ADS). The former describes a target machine, and the latter describes its microprograms.

The MRGL allows the microprogrammer to write microprograms sequentially and procedurally and provides the facility of generating horizontally-structured microinstructions. Furthermore, the MPGL can be applied to a wide range of machines by describing them in the MDS.

First, this paper briefly describes the design considerations for MPGL. Next, the syntax and semantics of MPGL are presented by using some examples. Finally, the characteristics of MPGL are summarized.

1. まえがき

近年，高速で比較的大容量の IC メモリが制御記憶として用いられるようになって，ダイナミック・マイクロプログラミングが実用の計算機で採用されるようになり，マイクロ診断，エミュレーション，高級言語計算機などを中心に使われている．また，ユーザ・マイクロプログラミングの有用性も論じられている[1),7)]．

これに伴って，マイクロプログラム（以下，μP と記す）の記述言語とその処理システムもいくつか考えられている．

MPL は，垂直型のマイクロ命令を持つ計算機に対して考えられた[3)]．CAS は，IBM 360 シリーズのマイクロプログラミングに利用された[2),8)]．また，1972年に，MDS が E. W. Dubbs らによって発表された[5)]．また，C. V. Ramamoorthy らによって開発された SIMPL は，水平型マイクロ命令を対象とした言語である[6)]．

日本で開発されたシステムには，シミュレーションと評価を主目的とする MPGS[4)]，並びに，汎用のマイクロプログラム・シミュレータとして開発された LFM[9)] と MIPS[10)] がある．

これらの諸システムに比し，特に，

（i） 高級言語でシーケンシャルに記述された μP を効率のよい水平型マイクロ命令のビット・パターン列に変換する．

（ii） 広汎な μP 制御方式計算機に対して適用できる．

などを，主な目標として μP 記述言語 MPGL (Micro-Program Generative Language) を設計した．

2. マイクロプログラム記述言語の記述方式

広汎な μP 制御方式計算機の μP を記述して，これを処理することを可能とするため，計算機の記述が必要である．更に，計算機の記述に対して，何らかの方式で μP を記述する必要がある．計算機の記述，及び μP の記述について検討した結果を次に要約する．

2.1 計算機の記述

計算機の記述は，制御記憶とマイクロ命令のデコーダを中心とする制御部の記述と，制御部からの制御信

---

* A High Level Microprogramming Language: MPGL by Takanobu BABA (University of Electro-communications), Hiroshi HAGIWARA (Faculty of Engineering, Kyoto University) and Yuji FUJIMOTO (Fujitsu Co., Ltd.)

** 電気通信大学電気通信学部
*** 京都大学工学部
**** 富士通(株)

号によって制御される被制御部の記述とを分けて考える．

（a） 制御部の記述要素

（i） マイクロ命令のフォーマットとデコーダ．

（ii） 制御記憶アドレス・レジスタへのアドレス生成法．

（iii） マイクロ操作の実行順序．

（b） 被制御部の記述要素

（i） 計算機の静的な構造（たとえば，レジスタのビット長など）の記述．

（ii） 制御部からの制御信号によって行われるマイクロ操作の記述．

MPGL では，これらの記述要素を記述するレベルとして，レジスタ・トランスファ・レベルを採用した．

2.2 マイクロプログラムの記述

マイクロ操作あるいは，マイクロ操作の組み合わせを記述したものを入力とし，記述されたマイクロ操作を指定するマイクロ命令を出力する翻訳プログラムを μP コンパイラと呼ぶ．μP コンパイラの入力となる μP の記述の方式について考えると，その汎用性から，μP の記述の対象が，（a）1つの計算機に限られるものと，（b）複数個の計算機について可能なもの，に分けられる．（b）については，更に，（i）μP の記述を計算機の記述に依存して行うものと，（ii）独立に行うもの，とに分けられる．

各方式において，オブジェクト μP の効率，コーディングの容易さ，ドキュメント性，処理システム開発の容易さ等について検討を行った．当初の方針（1の（ii））から，（b）の方式を選択し，更に，主としてオブジェクト・μP の効率を重視して（i）の方式を採用した．

3. MPGL 言語

MPGL は，前述の如く，計算機に独立な仕様を有し，μP の記述は計算機の記述に依存して行う方式を採用した．

MPGL は，μP 制御方式計算機の記述を行うマシン記述部 (Machine Description Section, MDS) と，これに対して μP のアルゴリズム記述を行うアルゴリズム記述部 (Algorithm Description Section, ADS) とから成る．各部の言語仕様をバッカス記法で定義したものを要約して付録に示した．

3.1 マシン記述部 (MDS)

マシン記述部は処理の対象となる計算機の記述を行う．記述は制御部と被制御部の記述から成る．

3.1.1 制御部の記述

次に述べる5つのブロックから構成される．

（1） field definition table (FDT)

FDT は，マイクロ命令の各フィールドの名称とそのビット長を記述する．

```
(例) BB  6  T1
     P1  1  P
     SP  5  #
```

BB フィールドは，6ビットで，BB フィールドに属するマイクロオーダはタイミング T1 で実行される．P1フィールドはパリティ・ビットである．SP フィールドは，5ビットで，タイミングには関係のないフィールドである．

（2） timing table (TIM)

FDT 中のタイミング名と対応して，各フィールドに属するマイクロオーダの実行されるタイミングの指定を行う．

```
(例) T1   1
     TD  13
```

タイミング T1 と TD は，それぞれ値1及び13を持ち，T1 に対応するフィールドは，TD に対応するフィールドより先に実行される．

（3） parity specification table (PST)

パリティ・チェック・ビットとなるフィールドについて記述する．

```
(例) P1  O  BB, AB, AA
```

P1 は，フィールド BB, AB, AA の奇数パリティ (O) である．

（4） mnemonic table (MT)

フィールドごとに，マイクロオーダのネモニック名と対応するビット・パターンを記述する．

```
(例) BB  NBB  000000
          U   001000
          C   111×××
```

BB フィールドには，マイクロオーダ・ネモニック NBB, U, C が含まれる．フィールド名の頭に N を付けた NBB は，BB フィールドの無操作 (no operation) を表わす．また，C の 111××× は，後の3ビットが0でも1でもよい冗長なビットであることを表わす．

（5） address generation scheme (AGS)

次に実行するマイクロ命令のアドレスの生成法を記述する．

```
(例)
  U  "0 : 3"  H  1001B
     "4 : 11" V  FR "0 : 7"  /*ES@NTS@¬MRR*/
```

無条件分岐（U）に対するアドレス生成法の記述である．制御記憶アドレス・レジスタ（CMARと記す）の'0：3'* ビットには，結線論理により，$(1001)_2$ がセットされる．また，'4：11'には，レジスタ FR'0：7'の値がセットされる．これらは /* と */ で包まれたマイクロオーダ（**3.1.2** で後述）によって実行される．例では，実行時にならないと分岐先の番地が決まらず，これを機能分岐と呼ぶ．

CMAR へのアドレス生成法として，以下の方法が記述できる．

（i） 前の CMAR の内容を残す．
（ii） フィールドの値をセットする．
（iii） CMAR の一部あるいは全体を，ある値だけ増加する．
（iv） 結線論理によって直接固定された値をセットする．
（v） レジスタ，カウンタ等の値をセットする．
（vi） テスト条件の成立，不成立に応じて1あるいは0をセットする．
（vii） CMAR のアドレス・スタックの先頭を CMAR 全体にセットする．

分岐の仕方に応じて記述できるアドレス生成法は制限されている．

3.1.2 被制御部の記述

被制御部で記述される部分は，ADS から見れば，使用できる変数，定数，演算子の記述である．

a．制御記述

マイクロ操作を記述する場合に，これを制御するマイクロオーダを記述する部分を制御記述と呼び，/* と */ で包んで記述する．

制御記述はマイクロオーダの論理式を基本的な構成要素とする．また，マイクロ操作実行のタイミングは，対応する制御記述の左端のマイクロオーダの属するフィールドの実行のタイミング**とする．ただし，メモリの動作に関してはタイミングをイクスプリシットに記述する．

(例 1) /*C 1@(¬C 2♯C 3)*/***

マイクロ操作実行の条件は，C 1 があって，C 2 がないか C 3 があることである．マイクロ操作実行のタイミングは C 1 の属するフィールドより得る．

(例 2) /*⟨BB⟩, 'C 1♯C 2'(−1)*/

BB フィールドに，NBB 以外のマイクロオーダがあり，かつ1つ前のマイクロ命令 (−1) に C 1 か C 2 があることがマイクロ操作実行の条件である．実行のタイミングは BB フィールドから得る．

b．variable table (VT)

VT は，ADS で使用する変数に関する記述を行うテーブルである．特に，レジスタ，ターミナル，カウンタについて共通の属性として記述するテスト条件と転送について先に述べる．

（A） テスト条件の記述

テストには，関係式の成立，不成立をテストするリレーション・テストと，1次元の変数のブール式の値が0か1かをテストするイクスプレッション・テストがある．

(例) |U'8：11'=0|/*U17@¬UNC*/

マイクロオーダ U 17 があって，UNC がないとき U'8：11' が0か否かをテストする．

（B） 転送に関する記述

転送には，ある変数の値によって転送先の変る間接指定転送と，無条件に行われる直接指定転送がある．さらに，直接指定転送，間接指定転送共に，同時に複数個の変数へ転送を行う同時転送がある．

以下の記述例はレジスタRについての記述とする．

(例 1) 直接指定転送（同時転送）

'0：31'⇒B'0：31', '0：15'⇒C'0：15'/*CN*/

R'0：31'⇒B'0：31' と R'0：15'⇒C'0：15' の2つの転送がマイクロオーダ CN により，同時に行われる．

(例 2) 間接指定転送

⌙BAB'0：1'(0)'0：7'⇒G'0：7'(1)'8：15'⇒G'0：7'
(2)'16：23'⇒G'0：7'(3)'24：31'⇒G'：7'/*RG*/

制御記述 RG により，カウンタ BAB の内容が( ) 内に示された値のとき，その右側にある転送が行われる．たとえば，BAB'0：1' の値が1のとき R'8：15'⇒G'0：7' の転送が行われる．

（1） レジスタとターミナルの記述

変数名，変数の属性 (REG/TER)，長さ，テスト条件，及び，転送が，レジスタとターミナルについての記述の属性である．レジスタについて記述例を示す．

(例) U REG 32 |U'8：11'=0|/*U17@¬UNC*/
'0：31'⇒BUSB'0：31'/*U*/

U は 32 ビットのレジスタ (REG) である．テスト

---

* レジスタ等のビット位置の指定は，左端を0とし 'i：j' によって i から j ビットまでの指定を行う．これを以後レジスタの次元と呼ぶ．

** 3.1.1（2）に述べたように，マイクロオーダの実行されるタイミングは各フィールドごとに指定される．これをそのフィールドの実行のタイミングと呼ぶ．

*** ¬, @, ♯, % はそれぞれ否定，論理積，論理和，排他的論理和を表わす．

可能な条件は U'8：11'=0であり，BUSB'0：31'へ転送できる．ターミナルについても同様である．

（2） カウンタの記述

カウンタの記述がレジスタと異なるのは，カウンタの値の増減のための記述が付け加わる事だけである．

（例） G COU 8 |G'0：7'¬=0|/*GNZ@¬UNC*/
'0：7'⇒BUSB'24：31'/*G*/
>'0：7'/*DG 1*/, 2<'0：3'/*IG 2*/

カウンタ (COU) Gは8ビットであり，G'0：7'¬=0 がテスト可能な条件で，BUSB'24：31'へ転送できる．G'0：7'を1だけデクリメントすることと，G'0：3'を2だけインクリメントすることが可能である．

（3） フリップ・フロップの記述

フリップ・フロップの記述がレジスタと異なるのは，転送の記述がなくフリップ・フロップの値のセットの記述があることである．セット法の記述には，テストの記述と同様，関係式の成立，不成立によって1, 0 をセットするリレーション・セットと，ブール式の値をセットするイクスプレッション・セットの2通りがある．

（例） F 1 FF 2 |F 1'0'=1|/*F 12@¬UNC*/
'0' ¬F 2/*TF 1*/, F 2@F 3/*F 23*/
'1' BUSA'0：31'¬=0/*NZ*/

フリップ・フロップ (FF) F 1は2ビットで，F 1'0'=1がテスト可能な条件である．F 1'0'には，¬F 2とF 2ⓐF 3の値がセットでき，F 1'1'には，BUSA'0：31'が0でないとき1，0のとき0がセットされる．

（4） 主記憶の記述

主記憶に関しては，メモリの名称，1語の長さ，メモリの容量，アドレス・レジスタ (mar と記す)，データ・レジスタ (mdr と記す)，読出しと書込みのタイミング及び制御記述が，記述要素である．

（例） MM MEM 8 131000 MAR'0：21' MR'0：31'
T 4(−1) TD(−1)
/*'RW@IH'(−1)*/ /*'CW@IH'(−1)*/

MM はメモリ (MEM) で，1語8ビット，容量131000 語である．MAR'0：21'によって番地が指定され，MR'0：31'が mdr である．

T 4 (−1) は，読出しに対するタイミングの記述である．1つ前のマイクロ命令 (−1) で mar に番地をセットする．読出しは，タイミング T 4 で行う．TD (−1) は，書込みに対するタイミングの記述である．1つ前のマイクロ命令 (−1) で mar に番地をセットする．書込みは，タイミング TD で行う．'RW@IH' (−1) と 'CW@IH' (−1) は，それぞれ読出しと書込みのマイクロ操作に対する制御記述である．

（5） スクラッチパッド・メモリの記述

スクラッチパッド・メモリ (spm と記す) としては，高速の IC メモリが使用されるのが一般的である．このため「1マイクロ命令で，番地のセット，及び読出しと書込みが行えるメモリ」を spm として記述する．読出しは spm からの転送として記述し，書込みは spm への転送として記述する．また，spm には，一般的にマイクロオーダによって指定できる定数の番地が用意されている場合が多い．これを，アドレス・ソースとして記述する．

（例） SPM 0 SPM 32 128 SPAR'1：7' T 2 TB
/*〈SP〉*/ /*〈SP〉@〈BA〉*/
10X/*U 0*/, 11X/*U 1*/,
18X/*U 2*/
'0：7'⇒ALUA'24：31'/*AA 0*/

SPM 0 は spm (SPM) で，1語 32 ビット，容量 128 語で，番地の指定は SPAR '1：7' で行う．T 2, TB は各々読出しと書込みのタイミングである．〈SP〉と〈SP〉ⓐ〈BA〉は，それぞれ読出しと書込みの制御記述である．$(10)_{16}$, $(11)_{16}$, $(18)_{16}$ がアドレス・ソースとして使える．読出されたデータの '0：7' が ALUA '24：31' に転送できる．

c. constant table (CT)

CT は ADS で使用できる定数を記述する．定数の生成法としては，（1） 結線論理によって定まった値が生成される場合（ハードウェア定数と呼ぶ）と，（2） マイクロ命令中の特定のフィールドのビット・パターンが定数のビット・パターンの全部もしくは一部に使用される場合（フィールド定数と呼ぶ）とがある．

（例 1） ハードウェア定数
X "4" ALUB'0：31' /*C 4*/

$(4)_{16}$ が，制御記述 C 4 によって ALUB'0：31' に生成される．

（例 2） フィールド定数
R "10××××××" FA'3：5', FB'0：2' VAR'0：7'
/*C*/

フィールド FA'3：5' が 101，FB'0：2' が 001 であったとすると，VAR'0：7' には $(10101001)_2$ が，制御記述Cによって生成される．

d. operator table (OPT)

OPT は ALU の動作を記述することにより，ADS で使用できるオペレータを定義する．また，通常の計算機の ALU を記述できることが必要である．このため，MPGL では基本的な演算の機能を選びこれを

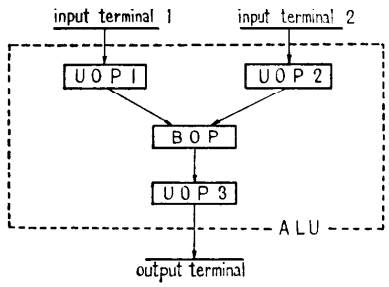

Fig. 1 Definition of operator

Fig. 1 のように組み合わせて記述する．図中，UOP 1，UOP 2，UOP 3 は単項演算，BOP は 2 項演算である．

基本的な演算としては，データの連結，シフト（ロジカル，アリスメティック，サーキュラ），論理演算（論理積，論理和，排他的論理和，論理否定），算術演算（2 進と 10 進の加減算，2 進の乗算）がある．更に，入力をそのまま出力とする場合（→と記述する）と，入力の如何に拘らず出力を常に 0 とする場合（||と記述する）とがある．

(例 1)

```
〈SRL〉 /*SRL*/ # ALUB'0 : 31' BUSA'0 : 31'
          ||    →  →   .SRL.(0 : SC'0')|
```

入力端子は ALUB 1 個で，出力端子は BUSA である．基本演算シフト・ライト・ロジカル (.SRL.) を行うが，この際，左端より 0 が入力され，右端にこぼれ落ちた値が SC'0' にセットされる．

ALU の動作の指定をレジスタ等の値により間接的に行う場合がある．これを間接指定演算と呼ぶ．

(例 2)

```
〈FR〉 /*IND*/
  ALUA'0 : 31' ALUB'0 : 31' BUSA'0 : 31' FR'4 : 7'
      →            →      @      →          4
      →            →      %      →          7
      →            →   —(SC'1' : SC'1')  →  15
```

FR'4 : 7' の値によって演算が指定される．値が 4，7，15 のとき，それぞれ論理積 (@)，排他的論理和 (%)，2 進減算 (—) が行われる．

例 1，例 2 の先頭の〈SRL〉，〈FR〉が，ADS で演算子として使用される．

3.2 アルゴリズム記述部 (ADS)

ADS では，MDS に記述された計算機に対して，作成する μP の記述を行う．

ADS は，宣言ブロック，アルゴリズム・ブロック，及び，プロセジュア・ブロックの 3 つのブロックから成る．宣言ブロックは，コンパイラに対して種々の情報を与える宣言文から成り，ADS の先頭に記述する．アルゴリズム・ブロックとプロセジュア・ブロックは，それぞれ主プログラムとサブルーチンに相当する．記述例を次に示す．

```
(例)  ALG EX1 (X"800");
          .
          .
          .
      GLA;
```

例では，EX 1 という名称のプログラムを，制御記憶の $(800)_{16}$ 番地をエントリ・ポイントとして作成することを表わす．アルゴリズム・ブロックとプロセジュア・ブロックの中には，オブジェクトのマイクロ命令に変換される実行文の記述を行う．

3.2.1 宣 言 文

宣言文について述べ，記述例を示す．

(1) external statement

別に作成されたアルゴリズム・ブロック，プロセジュア・ブロックとの結合を可能にするため，そのブロックの名称等を宣言する．

(例) EXTRN PRO EX3 (X"820");

外部プロセジュア・ブロックの EX 3 のエントリ・アドレスが $(820)_{16}$ 番地である．

(2) available statement

実行時の一時記憶領域としてコンパイラが割り当ててよいレジスタ，あるいは spm を宣言する．

(例) AVAIL SPM 0 (X"39");

spm SPM 0 の $(39)_{16}$ 番地が割り当て可能である．

(3) address statement

ステートメントを特定の番地に割り付けることを指定する．

(例) ADDR LAB 1 (X"910");

LAB 1 というラベルを持つステートメントを $(910)_{16}$ 番地に割り付ける．

(4) equate statement

ADS での変数は，そのアドレスや次元の記述のためその表記が煩雑になることがあり，これを簡単に表記できるようにする．

(例)

```
EQU MM(SPM 0(X"10")'8 : 29')'0 : 31', MMU 0
```

spm SPM 0 の $(10)_{16}$ 番地の '8 : 29' までを番地とするメモリ MM の内容を MMU 0 と表記する．

3.2.2 実行文

実行文には，次の8種類のものがある．

（1） assignment statement

メモリの読出し，書込み，及び変数間の転送や演算を記述する．

記述例を次に示す．

(例 1) MM (U '8 : 29'), L:=
MM (SPM 0 (X "10") '8 : 29');

spm SPM 0 (X "10") '8 : 29' で指定される MM の内容を読出して，レジスタ L へ転送し，同時に，レジスタ U '8 : 29' で指定される MM へ書込む．

(例 2) G '0 : 7' := R(♯BAB);

レジスタ R の，カウンタ BAB の値によって指定される次元の内容を，カウンタ G に転送する．これは，MDS の間接指定転送の記述に対応する．

（2） flip-flop set statement

フリップ・フロップにブール値0または1をセットするもので，右辺にはブール式または関係式を記述する．

(例) F 1←F 2@F 3;

F 2 と F 3 の論理積を F 1 にセットする．

（3） counter control statement

MDS のカウンタの記述に対応して，カウンタの値の増減を記述する．

(例) 2<G '0 : 3';

G '0 : 3' の内容を，2だけインクリメントする．

（4） procedure call

プロセジュア・ブロック名を指定してコールする．

(例) SUB;

μP サブルーチン SUB をコールする．

（5） return statement

プロセジュア・ブロック内で使用され，このプロセジュア・ブロックを呼び出したプログラムへ戻る．RETURN; と記す．

（6） go to statement

無条件分岐を表わし，指定したラベルを持つステートメント，またはアルゴリズム・ブロックへの分岐を行う．

(例) GOTO L 0;

（7） if statement

条件の判定による分岐で，条件式は，VT 中のテストの記述に対応して記述する．

(例 1)

```
IF SPM 0(X "10")〈@〉L =XL 32 "0" THEN
    GOTO L 0 ELSE GOTO L 1;
```

SPM 0 (X "10") と L との論理積の結果が，0か否かに応じて L 0 か L 1 へ分岐する．

(例 2)

```
IF F 1=BL 1 "0",  F 2=BL 1 "1"
                  GOTO L 0, L 1, L 2, L 3;
```

2つの条件の成立，不成立に応じて，4方向分岐を行う．

（8） case statement

AGS の機能分岐 (**3.1.1.** (5)) の記述に対応して，CMAR に値をセットする変数名，及び，その値と分岐先との対応関係を記述する．

(例)

```
CASE FR '0 : 7' (X "60") ENT 5, (X "61") ENT 6
     (X "62") ENT 7;
```

変数 FR '0 : 7' の値が，$(60)_{16}$, $(61)_{16}$, $(62)_{16}$ の各場合に，ENT 5, ENT 6, ENT 7 へ分岐する．

4. むすび

μP の記述言語の記述方式について述べ，更に，これらの検討に基づいて開発した言語——MPGL——について述べた．MPGL の特徴を要約する．

（i） μP 制御方式計算機の記述を行うマシン記述部 (MDS) と，これに対して μP のアルゴリズム記述を行うアルゴリズム記述部 (ADS) とから成る．

（ii） 計算機の記述はマイクロ操作単位であり，これを制御するマイクロオーダを制御記述としてイクスプリシットに記述する．マイクロ操作を単位とするレジスタ・トランスファ・レベルでの記述方式は，μP 制御方式計算機の記述言語として自然であり，かつ，回路レベル以下の細部の相違の影響を受けない計算機の記述が行える．これによって，広汎な μP 制御方式計算機の記述が可能となった．

（iii） 制御記述の考え方により，マイクロオーダとマイクロ操作の対応が柔軟に記述できる．また，これによって間接符号化方式*にも対処できる．たとえば，**3.1.2 b.** (A) の記述例は，レジスタ U の値のテストを制御するマイクロオーダ U 17 の意味が，マイクロオーダ UNC が同一マイクロ命令内にあるか否かによって変ること

---

* マイクロ命令のあるフィールドの意味が，そのマイクロ命令の他のフィールドの値によって変るような方式を間接符号化方式 (indirect encoding) と呼ぶ．この方式はまた，2レベル符号化，あるいは，ビット・ステアリングと呼ばれることもある．

を表わす.

(iv) MDS の仕様は，ADS で μP を記述し易いように表形式である．即ち，各ブロックは，ADS で使用する変数，定数，オペレータを見出しとして，一定の記述要素を配列する.

(v) 水平型マイクロ命令の μP をシーケンシャルに記述できる．即ち，1つの水平型マイクロ命令は，複数個のマイクロ操作の実行を制御するのが普通であり，マイクロ操作相互の関係を考慮しつつプログラミングを行っていくことが，マイクロアセンブラによるプログラミングを煩雑なものとする一因であった．これを，**3.2.2** で述べたように，マイクロ操作単位に，シーケンシャルに記述できるようにした.

(vi) プログラマはマイクロ命令の番地付けを行わなくてもよい．即ち，マイクロアセンブラでは各マイクロ命令を制御記憶に配置するのは，プログラマの仕事であった．しかし，(v)からの必然的な結果として，プログラマがオブジェクト・マイクロ命令に手を加えることは考えられない．このため，μP の順序制御は ADS に制御文 (**3.2.2** の(4)～(8)) を用いて記述し，これをもとに処理システムが MDS の AGS に記述された次マイクロ命令アドレス生成法を参照して行うこととした.

(vii) ADS の仕様は ALGOL-like である.

(viii) 間接機能制御方式*の記述が可能である．機能分岐は，MDS の AGS で CMAR へのアドレス生成法を記述し (**3.1.1** (5)の例)，これを使用する μP を ADS の case statement (**3.2.2** (8) の (例)) で記述する．間接指定転送，及び，間接指定演算は，それぞれ MDS の VT と OPT で，機能レジスタの値と，機能レジスタがある値のときに行われる転送，及び，演算との対応を記述し，これを ADS の assignment statement で記述して使用する．**3.1.2.b.** (B) (例2) が VT での間接指定転送の記述例であり，これを使用する μP の記述が，**3.2.2** (1) (例2) である．OPT での間接指定演算の記述例である **3.1.2 d.** (例2) に対し，ADS では記述されたオペレータ 〈FR〉を SPM 0 (X "10"): =SPM 0(X "10")〈FR〉U; のように使用する.

筆者等は，この言語によって実在するいくつかの μP 制御方式計算機の記述と，その μP の記述を試み，スペックが不明な部分等を除き，十分記述できることが判った．また，MPGL の処理システムを作成することにより，ADS でシーケンシャルに記述された μP を，水平型のオブジェクトマイクロ命令に変換できることを確かめているが，これについては機会を改めて報告したい.

謝辞 言語仕様について助言戴いた京都大学渡辺勝正助教授に感謝する.

* マイクロ操作の制御情報の一部を特別のレジスタ (機能レジスタ (function register) と呼ぶ)，あるいはフリップ・フロップにもたせ，その内容によって，マイクロ操作の内容が変る方式である.

参 考 文 献

1) C. V. Ramamoorthy and M. Tsuchiya: A study of user-microprogrammable computers, Proc. SJCC, 1970, pp. 165～181 (1970).
2) S. S. Husson: Microprogramming: Principles and Practices, Prentice-Hall (1970).
3) R. H. Eckhouse: A high level microprogramming language (MPL), Proc. SJCC, 1971, pp. 169～177 (1971).
4) M. Hattori, M. Yano and K. Fujino: MPGS-A high level language for microprogram generating system, Proc. ACM Annual Conf., 1972, pp. 572～581 (1972).
5) E. W. Dubbs, R. L. Parsons and J. E. Petersen: A microprogram design system translater, Compcon 72, Sixth Annual IEEE Computer Society International Conference, Sept. 1972, pp. 95～97 (1972).
6) C. V. Ramamoorthy, M. Tsuchiya: A High-Level Language for Horizontal Microprogramming, IEEE Trans. on Computers, Vol. C-23, No. 8, pp. 791～801 (1974).
7) 萩原他：マイクロプログラミングの現状と将来の展望，情報処理，Vol. 14, No. 6, pp. 458～465 (1973).
8) 日本電子工業振興会：論理設計の自動化システムに関する研究――第1報――，pp. 78～88 (1973).
9) 日本電子工業振興会：論理設計の自動化システムに関する研究――第2報――，pp. 93～152 (1974).
10) 森田他：マイクロプログラムシミュレータ MIPS について，情報処理学会第15回大会，pp. 501～502 (1974).
11) 馬場，萩原：マイクロプログラム制御方式計算機の記述について，信学会電子計算機研究会資料 EC 73-23 (1973-07).
12) 馬場：マイクロプログム・ジェネレータ，情報処理学会ダイナミックマイクロプログラミングシンポジウム資料 (1973).

13) 馬場，萩原：マイクロプログラム・ジェネレータ，情報処理学会計算機設計自動化研究会資料 73-8 (1973).

付 録 バッカス記法による MPGL の定義（要約形）

```
** MPG Language **
  <MPG LANGUAGE> ::= <MACHINE DESCRIPTION SECTION>
        <ALGORITHM DESCRIPTION SECTION>

** Machine Description Section **
  <MACHINE DESCRIPTION SECTION> ::= MDS <MDS IDENTIFIER>
        <MDS BODY> SDM;
  <MDS IDENTIFIER> ::= <IDENTIFIER>
  <MDS BODY> ::= <FIELD DEFINITION TABLE> <TIMING TABLE>
        <PARITY SPECIFICATION TABLE> <MNEMONIC TABLE>
        <ADDRESS GENERATION SCHEMA> <VARIABLE TABLE>
        <CONSTANT TABLE> <OPERATOR TABLE>

** Control Description **
  <CONTROL DESCRIPTION> ::= /* <CONTROL CONTENT> */
  <CONTROL CONTENT> ::= # | <POINTED MNEMONICS>
  <POINTED MNEMONICS> ::= <POINTED MNEMONIC> |
        <POINTED MNEMONICS> , <POINTED MNEMONIC>
  <POINTED MNEMONIC> ::= ' <MNEMONIC EXPRESSION> '
        ( <MNEMONIC POINTER> ) | <MNEMONIC EXPRESSION>
  <MNEMONIC POINTER> ::= +1 | -1 | +2 | -2
  <MNEMONIC EXPRESSION> ::= <MNEMONIC SECONDARY> |
        <MNEMONIC EXPRESSION> # <MNEMONIC SECONDARY>
  <MNEMONIC SECONDARY> ::= <MNEMONIC PRIMARY> |
        <MNEMONIC SECONDARY> @ <MNEMONIC PRIMARY>
  <MNEMONIC PRIMARY> ::= <MNEMONIC IDENTIFIER> |
        ¬<MNEMONIC IDENTIFIER> | << <FIELD IDENTIFIER> >> |
        (<MNEMONIC EXPRESSION>) | ¬(<MNEMONIC EXPRESSION>)
  <MNEMONIC IDENTIFIER> ::= <IDENTIFIER>

** Address Generation Schema **
  <ADDRESS GENERATION SCHEMA> ::= AGS <AGS BODY> SGA;
  <AGS BODY> ::= <AGS BLOCK> | <AGS BODY> <AGS BLOCK>
  <AGS BLOCK> ::= <ADDRESS ATTRIBUTE> <ADDRESS GENERATION>
        <AGS MNEMONIC>
  <ADDRESS ATTRIBUTE> ::= U | T | P | R
  <ADDRESS GENERATION> ::= <AGS ROW> |
        <ADDRESS GENERATION> <AGS ROW>
  <AGS ROW> ::= <DIMENSION><GENERATION SOURCE>
  <GENERATION SOURCE> ::= F <FIELD SOURCE> |
        I <INCREMENT SOURCE> | H <HARDWARE SOURCE> |
        V <VARIABLE SOURCE> | T <RESULT OF TEST> | R |
        S <TOS DIMENSION>,<DEPTH OF STACK>
  <AGS MNEMONIC> ::= <CONTROL DESCRIPTION>

** Variable Table **
  <VARIABLE TABLE> ::= VT <VT BODY> TV;
  <VT BODY> ::= <VT ROW> | <VT BODY> <VT ROW>
  <VT ROW> ::= <REG ROW> | <RET ROW> | <TER ROW> |
        <COU ROW> | <FF ROW> | <MEM ROW> | <SPM ROW>

** Test **
  <TEST> ::= <TESTABLE CONDITION> | <EMPTY>
  <TESTABLE CONDITION> ::= <CONDITION> |
        <TESTABLE CONDITION>,<CONDITION>
  <CONDITION> ::= ||<TEST CONTENT>|| <CONTROL DESCRIPTION>
  <TEST CONTENT> ::= <RELATION TEST> |
        <EXPRESSION TEST>
  <RELATION TEST> ::= <RELATION>
  <RELATION> ::= <ID DIM> <RELATIONAL OPERATOR>
        <RELATION RIGHT PART>
  <RELATION RIGHT PART> ::= <ID DIM> | <X-VALUE>X |
        <INTEGER>
  <EXPRESSION TEST> ::= <BOOLEAN EXPRESSION><EQUAL OR NOT>
        <BOOLEAN VALUE>
  <BOOLEAN EXPRESSION> ::= <BOOLEAN FACTOR> |
        <BOOLEAN EXPRESSION> % <BOOLEAN FACTOR> |
        <BOOLEAN EXPRESSION> # <BOOLEAN FACTOR>
  <BOOLEAN FACTOR> ::= <BOOLEAN SECONDARY> |
        <BOOLEAN FACTOR> @ <BOOLEAN SECONDARY> |
  <BOOLEAN SECONDARY> ::= <BOOLEAN PRIMARY> |
        ¬<BOOLEAN PRIMARY>
  <BOOLEAN PRIMARY> ::= <ONE DIMENSIONAL VARIABLE> |
        (<BOOLEAN EXPRESSION>)
  <ONE DIMENSIONAL VARIABLE> ::= <IDENTIFIER>
        <IDENTIFIER>'<INTEGER>'
  <EQUAL OR NOT> ::= = | ¬=
  <BOOLEAN VALUE> ::= <8-DIGIT>

** Transfer **
  <TRANSFER> ::= <EMPTY> |
        <TRANSFER BLOCK>
  <TRANSFER BLOCK> ::= <TRANSFER ROW> |
        <TRANSFER BLOCK> , <TRANSFER ROW>
  <TRANSFER ROW> ::= <TWO KIND OF TRANSFER>
        <CONTROL DESCRIPTION>
  <TWO KIND OF TRANSFER> ::= <UNCONDITIONAL TRANSFER> |
        <CONDITIONAL TRANSFER>
  <UNCONDITIONAL TRANSFER> ::= <ELEMENTARY TRANSFER> |
        <UNCONDITIONAL TRANSFER> , <ELEMENTARY TRANSFER>
  <ELEMENTARY TRANSFER> ::= <SOURCE>=> <DESTINATION>
  <SOURCE> ::= <DIMENSION>
  <DESTINATION> ::= <REG TER RET COU ID> <DIMENSION> |
        <SPM IDENTIFIER> <DIMENSION>
  <REG TER RET COU ID> ::= <REG IDENTIFIER> |
        <TER IDENTIFIER> | <RET IDENTIFIER> |
        <COU IDENTIFIER>
  <CONDITIONAL TRANSFER> ::= |-<TRANSFER CONDITION>
        <CONDITIONED TRANSFERS>
  <TRANSFER CONDITION> ::= <TRANSFER CONDITION ID>
        <DIMENSION>
  <TRANSFER CONDITION ID> ::= <REG IDENTIFIER> |
        <COU IDENTIFIER> | <FF IDENTIFIER>
  <CONDITIONED TRANSFERS> ::= <CONDITIONED TRANSFER> |
        <CONDITIONED TRANSFERS> <CONDITIONED TRANSFER>
  <CONDITIONED TRANSFER> ::= ( <CONDITION VALUE> )
        <UNCONDITIONAL TRANSFER>
  <CONDITION VALUE> ::= <INTEGER> |
        <INTEGER> : <INTEGER>

** Constant Table **
  <CONSTANT TABLE> ::= CT <CT BODY> TC;
  <CT BODY> ::= SPACE | <CT ROW> | <CT BODY> <CT ROW>
  <CT ROW> ::= <HARDWARE CT ROW> | <FIELD CT ROW>

** Operator Table **
  <OPERATOR TABLE> ::= OPT <OPT BODY> TPO;
  <OPT BODY> ::= <DIRECT OPERATOR BLOCK> IND
        <INDIRECT OPERATOR BLOCK>

** Algorithm Description Section **
  <ALGORITHM DESCRIPTION SECTION> ::= ADS <ADS BODY> SDA;
  <ADS BODY> ::= <DECLARE BLOCK> <ALG-PRO BLOCK>
  <ALG-PRO BLOCK> ::= <ALGORITHM BLOCK> | <PROCEDURE BLOCK> |
        <ALG-PRO BLOCK> <ALGORITHM BLOCK> |
        <ALG-PRO BLOCK> <PROCEDURE BLOCK>
  <DECLARE BLOCK> ::= SPACE | <DECLARE STATEMENT> |
        <DECLARE BLOCK> <DECLARE STATEMENT>
  <DECLARE STATEMENT> ::=<EXTRN STATEMENT> ; |
        <AVAIL STATEMENT> ; | <EQU STATEMENT> |
        <ADDR STATEMENT> ;
  <ALGORITHM BLOCK> ::= <ALG STATEMENT> ;
        <ALG-EXEC STATEMENTS> GLA;
  <PROCEDURE BLOCK> ::= <PRO STATEMENT> ;
        <PRO-EXEC STATEMENTS> ORP;
  <ALG-EXEC STATEMENTS> ::= <ALG-EXEC STATEMENT> |
        <LABEL> : <ALG-EXEC STATEMENT> |
        <ALG-EXEC STATEMENTS> <ALG-EXEC STATEMENT>
  <PRO-EXEC STATEMENTS>::= <PRO-EXEC STATEMENT> |
        <LABEL> : <PRO-EXEC STATEMENT> |
        <PRO-EXEC STATEMENTS> <PRO-EXEC STATEMENT>
  <LABEL> ::= <IDENTIFIER>
  <ALG-EXEC STATEMENT> ::= <ASSIGN STATEMENT> |
        <FF-SET STATEMENT> ; | <GOTO STATEMENT> ; |
        <IF STATEMENT> ; | <CASE STATEMENT> ; |
        <PROC-CALL STATEMENT> ; |
        <COUNTER CONTROL STATEMENT>
  <PRO-EXEC STATEMENT> ::= <ALG-EXEC STATEMENT> |
        <RETURN STATEMENT> ;
```

(昭和 51 年 4 月日 28 受付)
(昭和 51 年 9 月 30 日再受付)