論　文

MPG マイクロプログラム・シミュレータ*

馬場敬信** 萩原　宏*** 藤本裕司****
高橋　訓***** 碇谷幸夫******

Abstract

A new microprogram simulator has been implemented to facilitate the logical simulation of the object microprograms generated by the MPG microprogram compiler.

The MPG is the processor of a high level microprogramming language, called MPGL, and allows the microprogrammer to write microprograms, in a sequential and procedural fashion, for the described target machine. As a subsystem of MPG, the simulator accepts a machine description and object microprograms and prints out the trace listing and the evaluation data.

Therefore, the characteristics of the MPG microprogram simulator are as follows: (1) The object code of the microprogram compiler is simulated, and the debugging aids for high level microprogramming language are provided. (2)It is applicable to a wide range of machines by using the machine description as the common data base with the compiler. (3) It facilitates the logical check of both a target machine and its microprograms. (4) The variables to be printed out may be determined without specification of microprogrammers by using the information from the compiler.

This paper first describes the system configuration of the MPG microprogram simulator and the format of the machine definition table. The procedures for interpreting an object microinstruction and executing microoperations are then described. The relation between the MPGL description and the simulator is described in detail. Finally, the results of an actual simulation are shown.

1. はじめに

ダイナミック・マイクロプログラミングの発展には，効率よくマイクロプログラム（以下，μP）を作成するための記述処理システムを実現することが重要な課題である．そして，このような μP を記述対象とするシステムにおいては，記述された μP を処理してオブジェクト・コードへの変換を行う機能と共に，μP のデバッグのための機能が要求される[15]．これは，μPが通常のマシン・レベルのプログラムより更に細かい，ハードウェア・レベルでのオペレーションの制御を行うため，ユーザが多数のリソースにわたるマイクロ操作の並列性，実行のタイミングなどを考慮しなければならないことによる．

このため，特に水平型のマイクロ命令を持つ計算機では，記述処理システム中に μP の論理シミュレータを持つものが多いが，これらの多くは，個々の計算機専用のシミュレータである．汎用を目ざすシステムとしては，CAS シミュレータ[1]，LFM[17]，GPMS[7]などがあげられる．CAS シミュレータは IBM System/

* Microprogram Simulator of MPG by Takanobu BABA (University of Electro-communications), Hiroshi HAGIWARA (Faculty of Engineering, Kyoto University), Yuji FUJIMOTO (Fujitsu, Ltd.), Satoshi TAKAHASHI (IBM Japan, Ltd.), and Yukio IKARIYA (Hitachi, Ltd.).
** 電気通信大学電気通信学部
*** 京都大学工学部
**** 富士通（株）
***** 日本 IBM（株）
****** （株）日立製作所

360 の μP 作成援護システム CAS のサブシステムである．また，LFM は PL/1 言語によってインプリメントされたシミュレータでコンパイラ方式を採用している．GPMSは，計算機の設計評価を主目的として作成されたハードウェア・シミュレータである．また，最近では，インタラクティブに μP のデバッグを行う方式も考えられ[3]，また，新たな μP シミュレータの提案もされている[14]．

これらの諸システムに比して，MPG シミュレータは，μP 記述処理システム MPG[9)~12)] のサブシステムとして，『高級言語で記述され，コンパイルされたオブジェクト・μP のシミュレーションを目的とする』点に特徴がある．次に，この目的に沿って作成された MPG シミュレータの特徴を列挙する．

(i) コンパイラのオブジェクト・マイクロ命令を入力として，高級言語による μP 記述に即応したトレース情報を出力する．

(ii) 計算機記述に応じたシミュレーションを行うことにより，幅広い計算機に適用可能とする．このため，特に間接符号化 (indirect encoding) 方式[8]，間接機能制御方式*を取扱えるようにした．

(iii) コンパイラと共通の計算機記述を使用することにより，ジェネレーションとシミュレーションで参照されるターゲット・マシンの記述を統一した．

(iv) 機能レベルでの計算機記述のチェックを行うためのトレース情報を出力する．

(v) ターゲット・マシンの各リソース，及びオブジェクト・マイクロ命令の使用頻度を出力して，記述を改良する際の資料を提供する．

(vi) (i)，(iv)，(v)のようにチェックの対象によって異なるトレース情報や評価データの出力法を，種々の制御パラメータを組合せることにより指定できる．

特に，(i)と(iii)は，CAS，LFM 等の従来のシミュレータと比較して，MPG シミュレータが高級言語で記述された μP に対するシミュレータであるために新たに実現された機能である．

処理方式としては，CAS，LFM がコンパイラ方式であるのに対し，MPG ではインタプリタ方式を採用し，テーブル形式に変換した計算機記述を参照しながら，ビット・パタン形式のマイクロ命令をインタプリティブに解釈し実行する．これは，MPG シミュレータがコンパイラと共通の計算機記述を参照し，コンパイラのオブジェクト・マイクロ命令を入力とするため，計算機と μP の記述の変更に最も対処しやすい方式として選んだものである．ただ，インタプリタ方式の欠点として，シミュレーションに要する時間が長くなるため，処理の高速化を工夫する必要がある．

また，システム作成に当っては，これを書換え可能制御記憶を利用して行うことにより処理の高速化を図ることが検討された．しかし，MPG の目的が高級言語の処理であるため，汎用性からソフトウェアによる作成を行った．

以下，本論文では，まず MPG のサブシステムとしての MPG シミュレータのシステム概要，及び計算機記述の内部テーブル構造の概略について述べる．次に，計算機記述に基づく解釈と実行の手法について述べ，更に，μP 記述及び計算機記述とシミュレータとの関係について述べる．最後に，実験結果から本シミュレータが MPG のサブシステムとして十分な処理機能をもつことを示す．

2. システム概要

Fig. 1 に示すように，MPG シミュレータは，計算

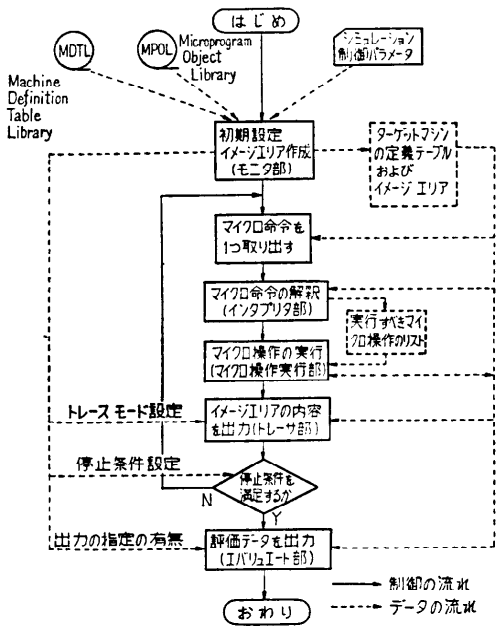


Fig. 1 System flow of MPG microprogram simulator.

* 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）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．機能レジスタの内容をいったんセットして変更することなく，何ステップものマイクロ操作を実行する間に何度もその内容が利用される場合，特にレジデュアル制御 (residual control) と呼ばれることがある[16]．

機記述及びMPGコンパイラの出力であるオブジェクト μP を入力として，記述された計算機上での μP の実行の論理シミュレーションを行う．

まず，モニタ部は制御パラメータを読み込み，パラメータの指定に従ってターゲットマシンの記述とビット・パタン形式のオブジェクト・μP を外部記憶より主記憶に読み込む．更に，モニタ部はターゲットマシンのイメージエリアをホスト・マシン上に作成すると共に，パラメータの指定に従ってシミュレータ各部の初期設定を行う．次に，マイクロ命令を1つずつ取り出しインタプリタ部によって解釈し，実行すべきマイクロ操作及びその実行のタイミングをマイクロ操作実行部に渡す．ターゲットマシン上でマイクロ操作がすべて実行された後に，トレーサ部によってイメージエリアの内容の出力が行われる．もし，シミュレーションの停止条件が満足されれば，エバリュエート部によって評価データを出力した後，シミュレーションを終了する．

本シミュレータは，HITAC 8350 のアセンブリ言語を使用して作成されており，総命令数 8000，所要メモリ約 88 k バイトである．

3. マシン定義テーブル

MPG では，処理の汎用化を目的としてターゲットマシンの記述を行い，これをマシン定義テーブル (MDT) と呼ぶ内部テーブル形式に変換して，コンパイラとシミュレータの共通のデータベースとして使用する．MDT の機能はシミュレータの立場から次のように要約される．

(i) イメージエリア作成のために，レジスタのビット長等の静的な構造に関する情報を与える（モニタ部に関連）．

(ii) オブジェクト・マイクロ命令の解釈に必要な制御部の定義を行う（インタプリタ部に関連）．

(iii) オブジェクト・マイクロ命令の実行に必要な被制御部におけるマイクロ操作の定義を行う（マイクロ操作実行部に関連）．

これらの機能を実現するため，MDT は次のように構成されている．

(i) 制御部と被制御部の記述ブロック[8]に応じた部分テーブルを構成要素とする(**Table 1**)．イメージエリア作成のための情報は VT (Variable Table) に集める．

(ii) MDT の参照は各部分テーブル相互に関連した形で行われる場合が多い．このため，部分テーブル間で関連する部分はポインタで結合し，処理効率の向上を計った．

(iii) 被制御部の各テーブルはマイクロ操作単位の処理を行いやすいように，マイクロ操作とこれを制御するマイクロオーダの記述（これを制御記述と呼ぶ），及びマイクロ操作の種類を表すコードの対を構成要素とする(**Fig. 2** 参照)．

Table 1 Sub-tables and microoperations in MDT.

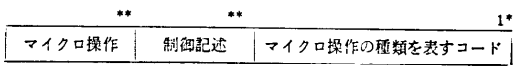
	内部テーブル名	内部テーブルに定義される内容
制御部	FDT	マイクロ命令の各フィールドの名称とそのビット長
	TIM	マイクロオーダの実行されるタイミング
	PST	マイクロ命令のパリティ・ビット
	MT	マイクロオーダ・ネモニックとビット・パターンの対応
	AGS*	制御記憶アドレス・レジスタへのアドレス生成(G)**
被制御部	VT	レジスタ等の各変数についてその属性とビット長を登録し同時に，次の TCT から SPAST への見出しとなる．
	TCT*	条件のテスト (E)
	TRT*	リソース間の転送 (R)
	FFST*	フリップ・フロップのセット (S)
	CCT*	カウンタの値の増減 (C)
	MST*	メイン・メモリの読出し (D) と書込み (W) スクラッチパッド・メモリの読出し(L)と書込み(M)
	SPAST*	スクラッチパッド・メモリのアドレス・レジスタへのアドレス生成 (A)
	CTH*	ワイヤッド・ロジックによる定数の発生 (H)
	CTF*	マイクロ命令のフィールドを利用した定数の発生(F)
	OPT*	ALU による演算 (P)
	EOT	OPT に定義された演算を構成する基本的な演算機能

* マイクロ操作の定義を行う内部テーブルを表す．
** 括弧内はマイクロ操作の種類を表すコードである．

部分テーブル間のポインタの関係を **Fig. 3** に示した．特に破線は制御部のマイクロオーダから被制御部のマイクロ操作へのポインタを表し，後述するように，解釈と実行の処理を結ぶ重要な機能をもつ．

4. マイクロ命令の解釈と実行

先に述べたように，MPG シミュレータはコンパイ



* バイト単位で長さを表す．
** マイクロ操作の種類により長さは異なる．

Fig. 2 Microoperation in MDT.

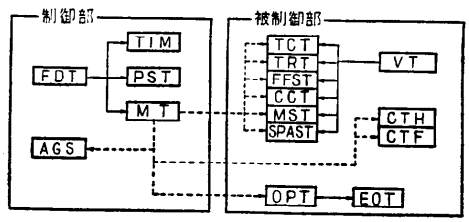


Fig. 3 Construction of MDT.

2	1	1	4	4	12	12	12	16	4
フィールド名	フィールドのビット長	マイクロ操作実行タイミング	MDT 中の MT へのポインタ	CMDR 内のこのフィールドのビットパタンへのポインタ	これから実行するマイクロ命令のマイクロオーダ (M_0)	1つ前に実行したマイクロ命令のマイクロオーダ (M_{-1})	2つ前に実行したマイクロ命令のマイクロオーダ (M_{-2})	M_0 に対する MEP へのポインタ	フィールド名を含む制御記述へのポインタ

Fig. 4 Microinstruction decoding table.

ラのオブジェクト・μP のシミュレータであることから，計算機の記述と μP の変更に最も柔軟に対処できる方式としてインタプリンタ方式を採用した．また，幅広い μP 制御計算機のシミュレータとして間接符号化方式や間接機能制御方式のシミュレーションを可能としている．以下，本章では，MPG シミュレータにおける解釈と実行の手法について，MDT を利用した処理の汎用化の手法，高速化のための手法などを中心に述べる．

4.1 マイクロ命令の解釈法（インタプリタ部）

インタプリタ部の目的は，オブジェクト・マイクロ命令を解釈して実行すべきマイクロ操作と実行のタイミングを決定することである．この目的に沿って，

（i） 同一マイクロ命令内のマイクロオーダ間の関係を処理する(間接符号化方式)．

（ii） 主記憶の読み出し，書込みのように，マイクロオーダの指定からマイクロ操作の実行までに2マイクロ命令以上を要するものを処理する．

（iii） 各マイクロ操作実行のタイミングを決定する．

などの機能が要求される．

インタプリタ部では，幅広い記述能力をもつ制御記述[8]の解釈を行うことにより，これらの機能を実現している．次に BNF 記法により，その仕様を示す．

```
〈CONTROL DESCRIPTION〉::=/*〈CONTROL CONTENT〉*/
〈CONTROL CONTENT〉::=♯|〈POINTED MNEMONICS〉
〈POINTED MNEMONICS〉::=〈POINTED MNEMONIC〉|
  〈POINTED MNEMONICS〉, 〈POINTED MNEMONIC〉
〈POINTED MNOMONIC〉::='〈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 EXPRESS-
  ION〉)
〈MNEMONIC IDENTIFIER〉::=〈IDENTIFIER〉
```

すなわち，制御記述は間接符号化方式を記述するマイクロオーダの論理式(〈MNEMONIC EXPRESSION〉) を基本的な構成要素とし，これに異なるマイクロ命令間のマイクロオーダの関係を表すポインタ(〈MNEMONIC POINTER〉) が付けられる．更に，制御記述の先頭にあるマイクロオーダの属するフィールドから，対になったマイクロ操作の実行のタイミングが得られる[8]．

このような制御記述解釈の時間的なオーバヘッドを少なくするため，**Fig. 4** に示すテーブルがあらかじめモニタ部によって作成されている．図中，フィールド名からビット・パタンへのポインタまでがあらかじめ書込まれている．M_0 はビット・パタンから得られたマイクロオーダ・ネモニックが置かれる．M_{-1}, M_{-2} には(ii)の処理を行うため，既に実行されたマイクロ命令のマイクロオーダを保存する．マイクロオーダには冗長ビットが許されるため[8]，各 M_0, M_{-1}, M_{-2} には4つまでのマイクロオーダが置ける．

インタプリタ部は，このテーブルを利用して次のような手順で処理を行う（**Fig. 5** 参照）．

（i） マイクロ命令を1つ読み込む．形式は **Fig. 6** に示す通りで，ビット・パタン以外にコンパイラより付加された情報（後述）がある．

（ii） MDT の制御部の部分テーブルを利用してマイクロオーダ・ネモニックに変換し，これと対になったマイクロ操作へのポインタ（MEP と呼ぶ）

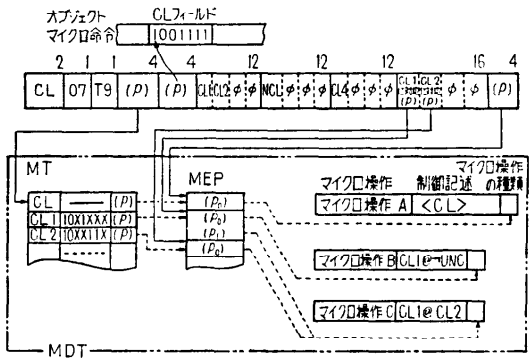


Fig. 5 An example of interpretation.

4	120	48	48	4	4
マイクロ命令の番地	マイクロ命令のビットパタン (EBCDIK コード)	マイクロ命令実行により使用される変数 (UV)	マイクロ命令実行により更新される変数 (DV)	セグメント番号	マイクロ命令の使用頻度書込み領域

計 228 バイト

Fig. 6 Format of object microinstruction.

と共にテーブルに書込む.

(ⅲ) MEP によってポイントされる制御記述を逆ポーランド形式に変換し,テーブル中の M_0, M_{-1}, M_{-2} にあるマイクロオーダが,その制御記述を満足するか否かをチェックする.

(ⅳ) 満足した制御記述に対する MEP を実行のタイミングに従って並べた上で,マイクロ操作実行部に渡す.

4.2 マイクロ操作の実行 (マイクロ操作実行部)

マイクロ操作実行部の目的は,インタプリタ部で決定される 13 種類 (Table 1 参照) のマイクロ操作を実行のタイミングに従ってホスト・マシン上で実行することである. 従って, MDT の被制御部の部分テーブルのセマンティクスに依存して, 次のような処理機能が必要となる.

(ⅰ) ALU の演算, 転送のビット位置, 及び分岐先番地について, 機能レジスタの値による指定を処理する(間接機能制御方式).

(ⅱ) 計算機によって特に異なる部分の多い ALU の演算機能と次マイクロ命令アドレスの生成法を幅広く取扱える.

また, プログラマから要求される機能として

(ⅲ) 各リソースを使用するたびに使用頻度を更新し, エバリュエート部を通じてプログラマに提供する.

(ⅳ) 実行時に新たに必要となる主記憶をプログラマの指定なしに確保する.

などがあげられる.

(ⅰ)の中で, ALU の演算及び転送のビット位置の間接機能制御は, MDT において機能レジスタの値とマイクロ操作の対がテーブル化されているので, これを参照することにより行う. 機能分岐は, 機能レジスタの内容を制御記憶アドレス・レジスタ (CMAR) にセットして行う. (ⅱ)の ALU の演算は 18 種類の基本的な演算[8]に対するサブルーチンを MDT の OPT 及びEOT に従って組合せて行う. また, 次マイクロ命令アドレスの生成法は CMAR の各ビットに対して記述された生成法[8]による値をセットする. (ⅳ) は語長に応じた長さをもつアドレスとデータの対を主記憶のイメージエリアとして作成することにより実現した. この方法には, 更に, 使用する分だけのイメージエリアが作成されること及び, 計算機ごとに異なる主記憶の語長, 語数に柔軟に対処できる, などの利点がある.

また, 処理プログラム作成に当っては, すべてのマイクロ操作がターゲット・マシンの任意のビット長のリソースを対象として行われるため, この処理が煩雑であった. これは, ホスト・マシンの命令セットによるビット操作が容易でなかったことによる.

5. MPGL 言語による記述とシミュレーション

先に述べたように, MPGL 言語は計算機の記述部 (MDS) と μP の記述部 (ADS) とから成るため, 各部の論理チェックについてシミュレータに要求される機能が異なる. すなわち, MDS については主として記述された計算機のデータ・フロー, 実行のタイミング, マイクロオーダとマイクロ操作の関係などがチェックの対象となり, ADS については, 記述された μP の論理的な誤りがチェックの対象となる. 従って, 筆者等は, 4 章で述べたシミュレーションの結果を, プログラマの多様な使用目的に応じて出力することを考え, その出力法や, 評価データの項目について検討を行った.

5.1 マイクロプログラムの記述とシミュレーション

現在の μP の論理シミュレータでは, マイクロ命令実行ごとに, プログラマの指定したリソースの内容を出力するのが一般的である. しかし, ADS でシーケンシャルに高級言語で記述された μP は, 次のような理由からオブジェクト・マイクロ命令との対応付けを行うことが困難である.

(ⅰ) ADS の記述順序は, コンパイラの最適化によって変えられる場合があり, また, 1 つのステートメントが複数のマイクロ命令に構成されたり, 逆に複数のステートメントが 1 つのマイクロ命令に構成されたりする.

(ⅱ) オブジェクト・マイクロ命令を解読して含まれるマイクロ操作を得るには, MDS との詳細な対照を行うことが必要で, 実際的でない.

(ⅲ) 番地付けもコンパイラによって行われるため, 番地と ADS のステートメントとの対応も判りにくい.

また, 本来コンパイラの使用者がオブジェクト・コードに立入らないと, デバッグが不可能ということでは, コンパイラの意味がないと考えられる. このため, ソースからオブジェクトに至るコンパイルの過程で不変なもので, かつデバッグに有効と思われる次の概念に注目した.

(1) セグメント: ADS で先頭から最後のステートメントまでシーケンシャルに記述されたステートメント群は，やはり最初から最後までシーケンシャルに実行されるマイクロ命令群に変換される．これをセグメントと呼ぶ．

(2) 参照あるいは更新される変数: あるセグメント内に限ると，ADS でファシリティとして参照される変数 (UV) と更新される変数 (DV) は，オブジェクト・マイクロ命令のセグメント内においても，使用され，あるいは更新される変数となる．

これらのことから，各セグメント内で UV と DV を出力することが最も有効と考え，これを実現した．

更に，実際の処理に際し好都合であるのは，コンパイラの最適化前処理及び最適なマイクロ命令の構成に同じ概念を利用していることである．このため，コンパイラがオブジェクト・コードにこれらの情報を付加することにより (Fig. 6 参照)，シミュレータは単に出力すべきファシリティとその値の編集を行えばよく，コンパイラとシミュレータを合わせたトータルシステムとしての処理効率の向上が実現された．

同じ乗算の μP に対するマイクロ命令ごとのトレースとセグメントごとのトレースの結果の一部を，比較のため **Fig. 7** に示した．

また，ADS での μP 記述の改良のためのデータとして，マイクロ命令の実行ステップ数，各マイクロ命令及びハードウェア・リソースの使用頻度を出力する．

5.2 計算機の記述とシミュレーション

計算機の記述はレジスタ・トランスファ・レベルで行われるが，これをチェックするには次の2点が中心となる．

(i) オブジェクト・マイクロ命令のデコーディングは正しく行われるか．

(ii) マイクロ操作の実行は，正しく行われるか．

(i) は主として制御部について，インタプリタ部から出力されるマイクロ操作とその順序を見てチェックする．特に，コンパイラとシミュレータの双方で重要な役割を果す制御記述については，間接符号化の影響も含め，十分チェックしておく必要がある．(ii) については，特に水平型のマイクロ命令ではいくつかのマイクロ操作を同時に実行することが多く，マシン・レベルに比して，マイクロ操作間の並列性，実行のタイミングなどについてチェックする必要がある．

これらのチェックを行うためには，マイクロ命令レ

```
ST-NO.  SOURCE STATEMENT
        ADS MULT
   1      AVAIL SPM0 (X"39");
   2      EXTRN ALG END (X"A20");
   3       ALG MULT (X"A50");
        *** SPM0 (X"18"): ADDRESS OF MULTIPLIER
        *** SPM0 (X"19"): ADDRESS OF MULTIPLICAND
   4            SPM0 (X"10"):=MM (SPM0 (X"18") '8 : 29');
   5            SPM0 (X"11"):=MM (SPM0 (X"19") '8 : 29');
   6            G:=XL 8"20";
   7            U:=XL 32"0";
   8            L:=XL 32"0";
   9       L0 : SPM0 (X"10"):=⟨SRLA⟩ SPM0 (X"10");
  10            IF SC '0'=BL 1 "0" THEN GOTO L 1;
  11            U:=U ⟨+⟩ SPM0 (X"11");
  12       L 1 : U:=⟨SRL⟩ U;
  13            L:=⟨SRC⟩ L;
  14            >G;
  15            IF G¬=XL 8"0" THEN GOTO L 0;
  16            MM (SPM0 (X"18") '8 : 29'):=U;
  17            SPM0 (X"18"):=SPM0 (X"18") ⟨+⟩ XL 32"4";
  18            MM (SPM0 (X"18") '8 : 29'):=L;
  19            GOTO END;
  20       GLA;
        SDA;
```

(a) Source listing of a sample microprogram.

HITAC 8350 MICRO-PROGRAM SIMULATION
MULTIPLICATION MICROPROGRAM

EXEC ADDR	CM-AR	G	SC	U	L	SPM0 (10)	SPM0 (11)	BUSA
0A50	0A51	00	00	00000000	00000000	00000000	00000000	00000100
0A51	0A52	00	00	00000000	00000000	00000007	00000000	00000007
0A52	0A53	00	00	00000000	00000000	00000007	00000000	00000200
0A53	0A54	00	00	00000000	00000000	00000007	00000004	00000004
0A54	0A55	20	00	00000000	00000000	00000007	00000004	00000020
0A55	0A56	20	00	00000000	00000000	00000007	00000004	00000000
0A56	0A57	20	00	00000000	00000000	00000007	00000004	00000000
0A57	0A58	20	02	00000000	00000000	00000003	00000004	00000003
0A58	0A59	20	02	00000000	00000000	00000003	00000004	00000000
0A59	0A5A	20	02	00000004	00000000	00000003	00000004	00000004

(b) Simulation in microinstruction trace mode.

```
HITAC 8350    MICRO-PROGRAM SIMULATION
              MULTIPLICATION MICROPROGRAM
SEGMENT TRACE MODE
          ***USED VARIABLE***                    ***DEFIND VARIABLE***
                        ***SEGMENT. NO: 0001***
SPM0 (18)=00000100   MM (000100)=00000007   *   SPM0 (10)=00000007   SPM0 (11)=00000004
SPM0 (19)=00000200   MM (000200)=00000004   *   G=20                U=00000000
                                            *   L=00000000
                        ***SEGMENT. NO: 0002***
SPM0 (10)=00000007                          *   SC=02               SPM0 (10)=00000003
                        ***SEGMENT. NO: 0003***
SPM0 (11)=00000004   U=00000000             *   SC=02               U=00000004
```

(c) Simulation in segment trace mode.

Fig. 7 Comparison of two trace modes.

ベルより更に細かいマイクロ操作レベルでの出力が必要となり，マイクロ操作実行部で各マイクロ操作実行ごとに行うこととした．

6. シミュレーション結果と考察

μP の作成援護システムの一部としてシミュレータの機能を十分活用できるように，**Table 2** に示すシミュレーション制御パラメータが用意されている．また，筆者等はこれらの制御パラメータを用いて MPG のサブシステムとして本シミュレータを使用することにより，用いた手法の有効性について検討を行った．特に μP シミュレータに対しては，シミュレーションに要する時間が最も重要と考えられるため，まず，(i)～(iii)でこの評価を行い，次に，(iv)～(vi)で処理の内容について述べる．

(i) シミュレータ各部の処理時間

2つの μP をシミュレートした結果を **Table 3*** に示す．モニタ部は，外部記憶からの読み込み，及び初期設定に時間を要していると考えられる．他の各部の処理時間は，ほぼ実行ステップ数に依存している．また，インタプリタ部の処理時間は 2～4 割を占めており，より一層の処理の高速化が望まれる．ただし，計算機の記述と μP に対する柔軟性は MPG のサブシステムとして必須であるため，コンパイラ方式の採用は難しく，たとえばインタプリタ方式で解釈結果のマイクロ操作のリストを保存するなどの改善策が考えられる．

(ii) ターゲット・マシンによる相違

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 2100 A の記述を行い，コンパイラの出力したオブジェクト・μP のシミュレーションが正しく行われる事を確認した．また，処理時間にも計算機による違いはほとんど生じていない (**Table 4**)．

(iii) 専用シミュレータとの比較

HITAC 8350 には，SIM 350 と呼ばれる専用のシミュレータがある．これと MPG シミュレータは，

Table 2 Simulation control parameters.

制御パラメータ名	機能	指定事項
START	シミュレーションの開始条件	ターゲットマシン名 マイクロプログラム名 開始番地
TRACE	シミュレーション・リストの出力法	マイクロ命令ごと セグメントごと ある範囲のマイクロ命令
SNAP	イメージエリアの内容のスナップショット・ダンプ	出力のタイミング 出力の対象
PRINT	マイクロ命令ごとのトレースで出力する変数名の指定	変数名
SET	初期値の設定	変数名と初期値
STOP	シミュレーションの終了条件	変数名とその値 終了番地 マイクロ命令実行ステップ数
EVALUATE	シミュレーション終了後のイメージエリアの使用頻度の出力	レジスタ，ターミナル等メモリ マイクロ命令
END	制御パラメータの終了	——

Table 3 Execution time of the each section.

μP ＼ シミュレータの各部	モニタ部	インタプリタ部	マイクロ操作実行部	トレース部	合計
A(マイクロ命令数 8 / ステップ数 131)	32	9	1	15	57
B(マイクロ命令数 73 / ステップ数 461)	36	41	4	34	115

(単位はすべて秒)

Table 4 Comparison of the execution times for different target machines and trace modes.

計算機	マイクロプログラム ＼ トレース法	マイクロ命令ごと	セグメントごと
HITAC 8350	C(637/100)*	124	130
	D(1154/523)	190	236
HP 2100 A	E(419/104)	83	88

* 上段：実行ステップ数
下段：実行セグメント数

(1)インタプリタ方式である，(2) HITAC 8350 のアセンブリ言語で作成された，(3) MDS で HITAC 8350を記述してマイクロインストラクション・トレースで使用した時の MPG シミュレータと SIM 350 とは，ほぼ同じ機能**をもつ，などの共通点がある．

この2つを使用してそれぞれ4種類の μP をシミュレートした***．その結果，1マイクロ命令の平均実行時間は，MPG が CPU 時間 0.15 秒，使用時間 0.18 秒であり，SIM 350 が CPU 時間 0.023 秒，使用時間 0.30 秒であった****．CPU 時間については MPG が約 6.5 倍となっており，これは汎用と専用の差を表していると考えられる．一方，使用時間がほとんど変らないのは，主としてトレースリストの出力法によると思われる*****．

(iv) 解釈と実行の処理について

---

* Table 3 は処理プログラム中にタイマ・マクロを挿入して各部を使用するごとに時間を計測した．また，次の Table 4 は処理の始めから終りまでを一括して計測した．

** 出力の形式，制御パラメータの指定などについて．

*** 4種類の μP の実行ステップ数の合計はそれぞれ約 2000 ステップである．

**** CPU 時間・CPU の動作時間，使用時間・入出力を含む全所要時間．

***** SIM 350 がファイル定義による出力を行うのに対し，MPG は入出力マクロ命令による出力を行う．

(ii) で述べたように異なるタイプのターゲット・マシンに対する解釈と実行の処理が行え，間接符号化や間接機能制御の処理が行えた．更に，これらの処理時間は (i) と (iii) の実験から十分実用的なものであることが判った．

(v) トレース法について

MPG で特にコンパイラ用に開発したセグメントごとのトレース法とマイクロ命令ごとのトレース法の比較を Table 4 に示した．セグメントごとに UV，DV を編集する処理を行うことから，処理時間は若干増加している．更にセグメントごとの出力については，出力すべきファシリティを指定する必要がなく，また冗長な出力や指定もれがない，などの利点が確かめられた．

(vi) 評価データの出力

5. に述べた評価データは

(1) MDS の各リソースは有効に利用されたか

(2) ADS の記述に冗長性があるか

(3) μP の実行所要時間はどの程度か

の3点について利用される．(1)については各リソースの使用頻度からよく使用されるデータを主記憶からレジスタに移すなどが行われた．(2)については，実行頻度の多いセグメントに重点を置いて，ADS 記述の冗長性のチェックが行われた．(3)は実行マイクロ命令ステップ数によって与えられる．

Fig. 8 に Fig. 6 の μP に対するファイナル・ダンプと評価データの出力リストの一部を示す．

```
***REGISTER, FF, ···DUMP***
U=00000000                L=0000001C
G=00                      SC=00
        ***SPM0  DUMP***
SPM0 (10)        00000000        00000004
SPM0 (18)        00000104        00000200
        ***MM DUMP***
MM (000100)      00    00    00    00
MM (000104)      00    00    00    1C
MM (000200)      00    00    00    04
```

(a) Final dump.

```
*FREQUENCY TABLE OF USED REG, FF, ···*
   U=000072                  L=000066
   G=000065                  SC=000164
*FREQUENCY TABLE OF CONTROL MEMORY*
   CM (0A50)=000001          CM (0A51)=000001
   CM (0A52)=000001          CM (0A53)=000001
   CM (0A54)=000001          CM (0A55)=000001
   CM (0A56)=000001          CM (0A57)=000032
   CM (0A58)=000032          CM (0A59)=000003
   CM (0A5A)=000032          CM (0A5B)=000032
   CM (0A5C)=000001          CM (0A5D)=000001
   CM (0A5E)=000001          CM (0A5F)=000001
```

(b) Evaluation data.

Fig. 8 A part of final dump and evaluation data.

7. むすび

本論文では，高級言語による μP 作成援護システム マイクロプログラム・ジェネレータ (MPG) のサブシステムとして開発した MPG マイクロプログラム・シミュレータについて，そのシステム概要，内部テーブル構造，処理手法などについて述べた．

本シミュレータは，高級言語で記述され，コンパイラによって処理されたオブジェクト・μP のシミュレーションを行う．また，その処理は内部テーブル形式に変換された計算機記述を，コンパイラとの共通のデータベースとして使用する．従って，本システムの特徴は

(i) 高級言語によるマイクロプログラミングのためのデバッギングエイドを提供．

(ii) マシン定義テーブルによる処理の汎用化，及びジェネレーションとシミュレーションの結合．

(iii) 計算機記述から μP 記述までの多様な論理チェックが可能．

(iv) コンパイラの処理結果を利用したリストの出力．

などに要約される．

また，実際に MPG のサブシステムとしてコンパイラと共に使用することにより，上記の特徴の有効性が確認されると共に，処理速度が実用に十分なものであることが判った．このような事から，本システムに実現された手法は，今後ダイナミックマイクロプログラミングの発展が予想される中で，記述言語の高レベル化，汎用化に役立つものと考える．

参考文献

1) B. R. S. Buckingham, W. C. Carter, W. R. Crawford, and G. A. Nowell : The Control Automation System, 6th Annual Symposium on Switching Circuit Theory and Logical Design (1965).

2) S. Young : A Microprogram Simulator, Proc. of DA Workshop, pp. 68～81 (1971).

3) M. Gasser : An Interactive Debugger for Software and Firmware, Sixth Annual Workshop on Microprogramming, pp. 113～119 (1973).

4) C. Vickery : Software Aids for Microprogram Development, Seventh Annual Workshop on Microprogramming, pp. 208～211 (1974).

5) R. Petzold, L. Richter, and H. P. Pohrs : A Two Level Microprogram Simulator, Seventh

Annual Workshop on Microprogramming, pp. 41～47 (1974).
6) M. S. Zucker : LOCS : An EDP Machine Logic and Control Simulator, IEEE Trans. on Electronic Computers, Vol. EC-14, No. 6, pp. 403～416 (1965).
7) M. Yamamoto, M. Hattori, M. Yano, K. Hakozaki, K. Kagiyama and K. Fujino : A Microprogrammed Computer Design and Evaluation System, First USA-JAPAN Computer Conference, pp. 139～144 (1972).
8) 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).
9) 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)
10) 馬場，萩原：マイクロプログラムの自動作成について，情報処理，Vol. 19, No. 1, pp. 61～69 (1978)
11) T. Baba : A Microprogram Generating System-MPG, Proc. IFIP Congress 77, pp. 739～744 (1977).
12) 馬場：マイクロプログラム記述言語とその処理システムに関する研究，京都大学博士論文 (1978)
13) 日立製作所：HITAC 8350 RCM 処理システム・マニュアル
14) たとえば，情報処理学会第 17 回大会，No. 103, 285, 366, 381, 383 など (1976).
15) 倉地：マイクロプログラムの記述とシミュレーション，情報処理，Vol. 14, No. 6, pp. 397～403 (1973).
16) 萩原：マイクロプログラミング，p. 343，産業図書，東京 (1977).
17) 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)，第2報 pp. 93～152 (1974).

(昭和52年5月18日受付)
(昭和52年11月10日再受付)