マイクロプログラムの記述とシミュレーション

倉　地　　　正[†]

1. まえがき

Wilkes によってマイクロプログラムによる電子計算機の制御方式が提案されて以来，そのすぐれた特徴のためにこの方式の計算機が数多く開発されてきたが，最近さらに性能を向上させるために高級言語に適するような複雑な命令が採用されたり，診断機能をファームウェア化するなどマイクロプログラム制御方式の採用は広まる一方である．それに伴ないマイクロプログラムを開発するためのデザインオートメーションシステムも次第に強化確立されてきており，最近では設計製造に不可欠な要素になっている．このシステムは種々の目的を有するが，その第一は誤りのないマイクロプログラムを設計することにある．一般に固定記憶に貯蔵されるマイクロ命令は通常の計算機命令に比べてビット数が多く，またゲート類を制御するマイクロオーダを複数個同時に実行する．従ってマイクロプログラムの設計は並列処理を考慮しなければならず，その確認の手段としてシミュレータが重要である．特に近年は IC 化した固定記憶が広く使われ一たん内容を記入した後の変更は不可能なので事前に完全に誤りのないマイクロプログラムを得るためにシミュレーションは不可欠といってよい．次にマイクロプログラムの内容を理解しやすく，設計者間での情報交換を正確かつ容易にするために共通の記述言語が必要である．一般の計算機用言語がアセンブラ言語からコンパイラ言語へと発達しているようにマイクロプログラム用の言語も次第に高級化してきている．この他に正確な製造や検査資料を得るためにマイクロプログラムのマスタファイルから固定記憶の内容を書き込むための制御テープや，検査機の制御テープなどの自動作成もシステムの重要な目的の一つである．

ここではまずマイクロプログラム設計用デザインオートメーションシステム全体の説明を行い，次に記述言語の例を紹介し，さらにシミュレータの方式とこの分野の将来方向について述べる．

2. マイクロプログラム用 DA システム

実用になっているマイクロプログラム設計用デザインオートメーションシステムはいままでにもいくつかの報告がなされているが，その一例として TOSBAC 5400 モデル 150 用に開発された μ-SIM-II システムについて紹介する．本システムはマイクロプログラムの設計から製造資料および検査資料作成の段階までをサポートする総合システムであり，記号形式のマイクロプログラムのアセンブル，固定記憶装置への番地の自動割付，マイクロプログラムのシミュレーション，固定記憶用 IC へのビットパタン書込テープの作成，固定記憶装置の内容の検査テープの作成，変更通知の作成などの機能を有している．μ-SIM-II のシステムフローチャートを Fig. 1 に示す．以下に構成要素の各プログラムについて順次説明する．

(1) マイクロプログラムアセンブラ

このプログラムはシンボリックに記述されたマイクロプログラムとその翻訳の方法を指定するゲートネームテーブルを入力とし，翻訳結果をビットパタンの形で中間ファイルに出力する．この記述方法などについては 3・1 節でさらにくわしく説明する．

(2) アロケータ

マイクロプログラムは，設計段階では一定の機能を有するブロックごとに記述され，ジャンプアドレスなどは記号番地で定義されている．アロケータはこれらの記号番地を入力プログラムの中にあらかじめ含ませてあるオリジンカードあるいはこの段階で指定されるアロケーションカードの指示によって固定記憶装置のアドレスに割付けてゆく．パリティビットを自動発生するのもアロケータの仕事である．この出力は ROS (固定記憶) のマスタファイルとなりシミュレーションおよび製造検査資料の作成に使用される．

(3) ROS デコーダ

マスタファイル上ではマイクロ命令はエンコードされた形で記憶されている．これをシミュレーションを

† 東京芝浦電気株式会社電算機ハードウェア開発部

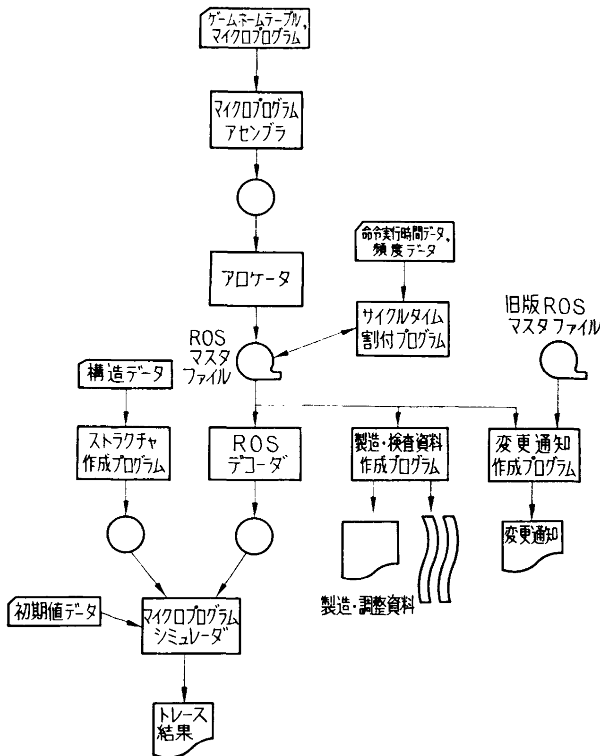

Fig. 1 System Flow Chart of μ-SIM-II

行なう時能率がよいように機能単位のゲート番号に分解しておくのが ROS デコーダの役割である.

（4） ストラクチャ作成

これはゲート番号の動作を定義するプログラムである. すなわち特定のゲート番号が与えられた場合にその動作をシミュレートするサブルーチンの名前(番号)と対象となるレジスタまたはバスのアドレスおよびシミュレートする順番を示すレベルを指定するテーブルを作成する. このプログラムによってハードウェアやマイクロ命令のフィールドの設計変更にも容易に対処することができる.

（5） マイクロプログラムシミュレータ

シミュレートする計算機命令とそれに使われるレジスタ類の初期値を与えるデータを入力としてシミュレーションを実行する. 実行すべきマイクロステップの内容は ROS デコーダの出力ファイルから得られる. これにより与えられたゲート番号に対応するサブルーチンがコールされ, インタプリティブにシミュレーションが進められ, ステップごとにレジスタや制御フリップフロップの内容をプリントアウトする. これはマイクロプログラムのデバッグに使われるばかりでなく, 計算機の調整用や保守用の資料としても使われる.

シミュレータはマイクロ命令のアドレスごとのダイナミックな使用頻度の統計をとる機能を有している. これは後に述べるマイクロ命令の実行時間を決定するプログラムのデータとして用いられる.

シミュレータは割込仕様やハードウェア制御の部分の機能が異るため CPU 用と IOCP 用と別々のものが開発された. シミュレータ以外のプログラムは両者に共通である.

（6） サイクルタイム割付プログラム

TOSBAC 5400 モデル 150 のマイクロ命令の実行時間はその種類により異なっている. 制御回路をあまり複雑にしないためにはこの種類が多くないことが望ましい. 本プログラムは各マイクロ命令の実行時間計算プログラムで計算した結果あるいは人手で与えた命令実行時間データと, シミュレータによる頻度統計あるいはギブソンミックス値計算基準による頻度を入力として最適なマシンサイクルの種類を決定する. こうして決定されたサイクルタイムはエンコードされてマイクロ命令の特定のフィールドに記入される. このサイクルタイムをもとにしてギブソンミックス値を計算することもできる.

（7） 製造・検査資料作成

固定記憶に使用される PROM（プログラマブル ROM）の書込テープやそのビットパタンをアドレスごとあるいはセンス線ごとにソートした製造資料を出力する. また製造された固定記憶の内容が正しいかどうかを検査するために試験機の制御テープを出力するプログラムも用意されており, これらの資料作成に人手の介入を省いてミスの入る余地を無くしている.

（8） 変更通知作成

設計変更があった場合に版数の異るマスタファイルの内容を比較して変更箇所を指示する資料を出力する.

3. マイクロプログラムの記述

マイクロプログラムを記憶する固定記憶装置は通常一語当り十数ビットないし百数十ビットの長さを有している. この設計をビットパタンのままで行なうのは

能率が悪くかつ設計者間あるいは検査や保守の担当者との情報交換という意味では都合が悪いので記号を用いた記述形式が用いられる．この方法としては計算機の命令を記述するのにアセンブラ言語やコンパイラ言語が用いられるのと類似の記述が使われる．ただマイクロプログラムの設計の場合には設計者が限定された専門家であるので記述自身の分り易さは多少犠牲にされることもある．その記述の仕方は対象となるハードウェアやマイクロプログラムの形式によって各種の変形があるが，ここでは2種類の例をとり上げてみる．

3.1 アセンブラ形式

アセンブラ形式はマイクロ命令の各々のフィールドの値にそれぞれ意味を持った記号名をつけてマイクロプログラムの記述を行う方式である．この記号名はデータ構造中の特定のゲートを開くことを指定する場合もありメモリへの読み書きの指定やカウンタの増減などの特定の機能の実行を指定していることもある．この形式の具体的な記述方法を前述の μ-SIM-II の方法を例にとりながら説明する．Fig. 2 に示すような回路が存在するとき，AレジスタとBレジスタの内容を

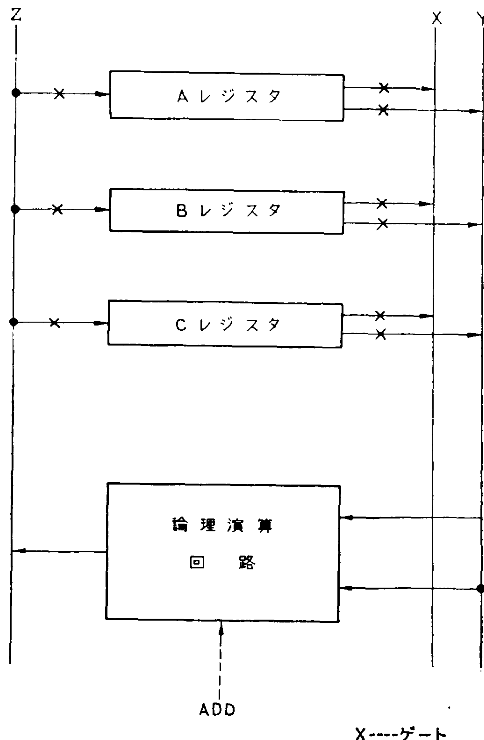

Fig. 2 Example of Microprogram Controlled Circuit

加算してCレジスタに格納するという操作を実行するためには

ARX, BRY, ZCR, ADD

のような記述が用いられる．ARX はAレジスタとXバスの間にあるゲートを開いてAレジスタの情報をXバスに送り出す事を指定している．BRY と ZCR も同様にレジスタとバス間のゲートの指定を行なう．4番目の ADD は論理演算回路において加算機能を選択することを指定する．この記号を変えることにより減算や論理和などの機能が選択できる．この場合にこれらのゲート指定記号を記述する順序は任意であって制約はない．マイクロ命令内でのゲート開閉のタイミングはマイクロ命令のフィールドとは独立に制御回路によって固定されている．並列に動作させることのできる回路が複数個ある場合にはそれに対応するゲート名をカンマで区切っていくらでも並べることができる．同じフィールドの異なる値，たとえば ADD と SUB を同時に指定してしまうようなミスはアセンブラでチェックされる．マイクロプログラムはここで説明したゲート記述部の他に命令の記号番地を定義するフィールドおよびジャンプアドレスを指定するフィールドを有する．これらは通常のアセンブラ言語のようにリロケータブルなプログラムを書くという目的の他に，記号番地フィールドにオクタル表示のアドレスを書くとそのマイクロ命令を指定されたアドレスに強制割付することを意味する．またジャンプアドレスフィールドに数字を書くとそれが命令に挿入され，ゲートフィールドの指定に従ってリテラルまたはジャンプアドレスとして使われる．これらは設計者が番地割付をも同時に考慮してマイクロ命令の共用などの最適化を行なう時に便利な機能である．ゲート指定の数が多くて一枚のカードに書ききれない場合には継続記号をつけて次のカードにまたがって記述する．

μ-SIM-II においては各ゲート指定記号の翻訳の仕方をゲートネームテーブルによって定義できるようになっておりアセンブラを汎用なものにしている．これは設計変更によってマイクロ命令のフィールドの定義を変更する場合にも便利である．ゲートネームカードは

1) ゲートネーム (12文字以内)
2) ビットパタン (12文字以内)
3) 先頭のビット位置
4) ビット長

の4個のフィールドから成り立っている．ゲートネー

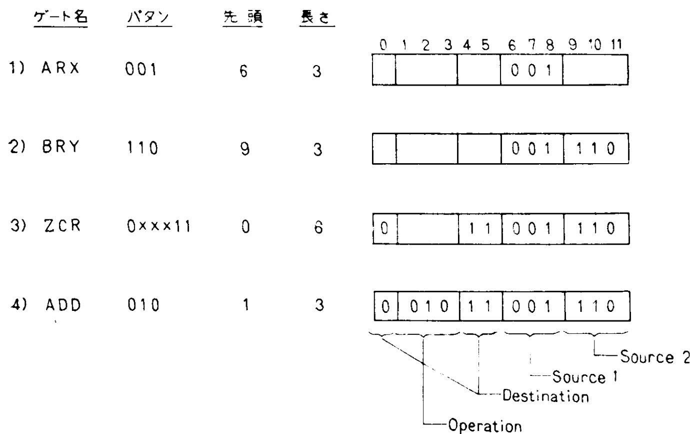

Fig. 3 Process of Gate Name Assembly

ムは前述のマイクロ命令の記述のところで用いたシンボリックなゲート指定記号に対応するものである．ビットパタンは1，0，Xの3種の文字で構成され，指定されたゲートネームをここに書かれたビットパタンに変換することを意味する．ここでXはその位置の内容を変化させないことを意味するのでマイクロ命令の特定のゲート指定が実際には離れたフィールドになっている場合に一枚のカードで指定するのに都合がよい．3）と4）のフィールドでそのゲート名をアセンブルした結果を格納するマイクロ命令上の位置を指定する．翻訳はマイクロプログラム記述に現われるゲート名ごとに順番に行なわれる．Fig. 2で示した例のマイクロ命令がアセンブルされる過程をFig. 3に示す．RCAで開発されたALSIMシステムにおけるマイクロプログラムの記述もほぼ同様の形式を採用している[1]．

アセンブラ形式ではマイクロプログラムの記述が簡潔で，その動作の内容に無関係に任意のマイクロコマンドに同じ規則で名前をつけることができ，汎用性のあるアセンブラが容易に作成できる．反面ドキュメンテーションあるいは情報の交換の手段としてはやや直観的でないという欠点を有するが，コメントカードの使用によりある程度カバーすることができる．

3.2 コンパイラ形式

マイクロプログラム記述のもう一つの形態はコンパイラ形式である．ここでいうコンパイラ形式とはマイクロプログラムの翻訳の方法がコンパイラ言語の手法によるのではなく，記述の仕方がコンパイラ言語のそれに似ているという意味である．この意味ではレジスタトランスファ形式と言ってもよい．この形式の記述は種々の提案が行なわれ研究されているが，実際の計算機の設計に使用された例としてはIBM社のCAS (Controls Automation System) がある[2]．CASにおけるマイクロプログラムの記述言語は論理装置の構造と動作を記述する論理設計言語と類似である[3]．

CASにおいてはメモリ，フリップフロップ，レジスタなどの記憶素子はFacilityと呼んでおりあらかじめ宣言する必要がある．

FCL ACC 16

というステートメントによって16ビットのACCという名前のレジスタを宣言することができる．また

SBFCL EXP=SEL (1, 7) ACC

というステートメントによってACCの1ビット目から7ビット目の部分にEXPという名前をつけることができる．

組合せ論理回路の記述はレジスタ単位の全体的な演算から信号線一本一本に対する論理演算まで各種のレベルの記述が可能である．演算の種類はデータの転送，特定のデータフィールドの選択，データの連結，論理演算，算術演算，比較演算などが許されている．

ACC=REGA+(REGB/REGC)

という記述によってレジスタBとCを接続したデータとレジスタAの内容との論理和を取りレジスタACCにセットするという意味を持たせている．他の言語と同じようにサブルーチンやファンクションを定義することもできる．

マイクロ命令中の特定のタイミングで動作を行なわせることを指定するためにタイミングポイントを定義する機能がある．たとえばP2からP4というタイミングポイントの間にKレジスタの内容を増加させるINCKというコマンドを記述すると

COM INCK (P2, P4)

のようになる．

コントロールの流れを司どる機能としてメモリからマイクロ命令を取出す機能や，マイクロ命令中で他のステートメントへ移るためのGO TO機能などが用意されている．

CASは以上で説明したようなコンパイラ形式の記述の他に使用者の選択によっては**3.1**節で述べたよう

なアセンブラ形式の記述も許している．さらに出力リストは入力した記述をそのまま言語形式でプリントするのではなくフローチャート形式に編集されるので制御の流れが把握しやすい．

コンパイラ形式の記述はアセンブラ形式に比べると各々のマイクロコマンドの動作が直観的で分り易く，情報伝達の媒体としては優れている．一方入力の記述言語をビットパタンに変換する翻訳プログラムは入力の種類が多いためにアセンブラ形式に比べて規模が大きくまた汎用になりにくい．特に演算関係以外のマイクロコマンドの表現方法には工夫が必要である．能率のよい翻訳方法と汎用性のある言語の開発につれ広く採用されるようになるであろう．

4. マイクロプログラムのシミュレーション

マイクロプログラムシミュレータに要求される機能はマイクロ命令の実行にしたがって計算機内の各レジスタの状態が変化する様子をトレースするほかに，主記憶の内容の変化，割込機能のシミュレーションや場合によっては接続されている周辺機器の動作もトレースする必要がある．さらにシミュレータの動作を制御するために時間の管理，レジスタやメモリの初期状態のセット，トレースやダンプのコントロールを行なう機能が要求される．シミュレータの方式には，コンパイラ方式とインタプリタ方式がある．コンパイラ方式はマイクロ命令のビットパタンをシミュレーションに先立って計算機の命令語に変換しておきそれを実行することによりマイクロプログラムの動作をシミュレートする．インタプリタ方式はビットパタンをあまり変形しない形で持っており実行時にその意味を解釈しながらシミュレートしてゆく方式である．コンパイラ方式の方がスピードの点では有利であり，インタプリタ方式の方が開発が容易であり融通性に富むが，マイクロプログラムの場合にはあまり実行時間に対する要求は厳しくないのでスピードはそれほど重視しなくてもよい．

マイクロプログラムシミュレータの汎用化は常に論議される課題であるが，割込機能が機種ごとに異なっており，またシミュレーションを実行しながらマイクロ命令の使用法の違反のチェックも要求されるので中中難しい．命令の正常動作のみをシミュレーションの対象とする場合には汎用化も可能である．次に周辺機器コントローラ用に開発したインタプリタ方式の μ-SIM-M を例にとりながらマイクロプログラムシミュレータの構成を眺めてみる．

4.1 ROS デコーダ

複雑にエンコードされたマイクロ命令のビットパタンをシミュレーション実行時にデコードするのは不経済であるので，シミュレーションに先立ち標準形にデコードしておく．デコードされた各フィールドはシミュレーションを実行するルーチンの番号や対象とするレジスタに対応する．こうして作成された命令のイメージは中間ファイルに記録されシミュレーションプログラムに渡される．マイクロ命令が RAM（ランダムアクセスメモリ）に含まれており，実行の途中で書換えられる場合にはコンパイラ方式やこの ROS デコーダのような方式は使用できず，シミュレーション時にビットパタンの翻訳をしなければならない．

4.2 命令の取り出し

デコードされた形式のマイクロプログラム全体を常時シミュレータのコアメモリの中に持つのは不経済であるので，固定記憶の全アドレス領域を固定長のブロックに分けコアメモリへはその時点でアクセスしている数ブロックのみを取り出しておき残りはランダムアクセスファイルに保持するバッファメモリと類似の方式を採用している．バッファメモリのスワップ方式はいろいろな方法が開発されているがここでは新しいブロックが必要になった場合に最も長時間アクセスされなかったブロックを置きかえる方法を取っている．

4.3 シミュレーションの方法

デコードされたマイクロ命令が固定記憶イメージからとり出されるとシミュレーション実行制御ルーチンは対応する命令実行ルーチンへコントロールを渡す．そこでレジスタからバスへの転送や加算機などに対応した各種の基本的な機能を実行する基本機能ルーチンが順番にコールされ，それらはコアメモリ上に割付けられた対象レジスタのイメージをとり出して処理を行ない目的レジスタに相当するアドレスへ結果をセットする．この構成法によって命令の機能変更が容易に行なえまた命令ルーチンが理解しやすくなっている．基本機能ルーチンはデバッグスイッチの指定により実行内容をプリントするので命令ルーチンのデバッグが容易である．

4.4 割込とディレール情報の処理

マイクロプログラムのシミュレーションにおいてハードウェア割込はマイクロプログラムが特定のアドレスに達した時あるいはシミュレータのクロックが一定時間経過した時に発生することが要求される．一方シ

ミュレーションを効果的に実施するためには，割込要求と同じような条件でレジスタの内容を変更したりトレース状態を変更したりする機能も要求される．μ-SIM-M においてはこれらを全てディレール情報と称して統一的に取扱う．ディレール情報には

1) ハードウェア割込を指定するもの
2) コンソールのスイッチ操作を指定するもの
3) レジスタ・メモリ類の初期状態を指定するもの
4) レジスタ・メモリ類の状態のダンプを指定するもの
5) 命令のトレース区域の指定・シミュレーションの打切りなどの制御に関するもの

などに分類される．これらはシミュレーション開始時にディレールカードとして読み込まれるが，読み込むと直ちに実行するか，特定のアドレスに制御が達した時に実行するか，あるいはクロックが特定の値に達した時に実行するかを指定することができる．これらのディレール情報を処理するためにアドレスキュー，タイムキューおよび割込キューの３種類のリスト形式のキューが用意されている．

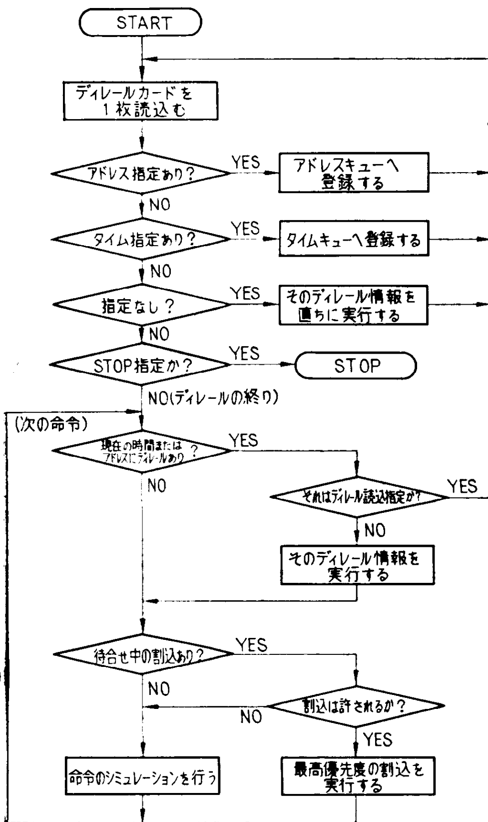

Fig. 4 Flowchart of the derail information handling

ディレールカードの読込が終了するとマイクロ命令の実行に移るが，各命令の実行の前に現在のアドレスまたは時間で実行すべきディレール情報があるかどうかを調べ，もしあればそれを実行する．次に待合せ中の割込があるかどうか，および割込がマスクされていないかどうかを調べ，許されていれば最高優先度の割込を処理する．通常は割込が発生するとアドレスが変更されるので，新しいアドレスに対して上記の処理をくり返す．割込がなかった場合には命令自身のシミュレーションを実行して次のアドレスへ進む．

シミュレータ全体の流れを Fig. 4 に示す．

5. 将来の課題

以上マイクロプログラム用のデザインオートメーションについての現状の説明を行なったが，将来の発展の方向について若干述べてみる．

まずシステム自身の汎用化が促進されねばならないと考える．これは新しいハードウェアが開発される度にそのためのシステムを開発するという無駄をさけ，その代りにシステムの性能あるいは機能の向上に努力を向けるという意味で大切である．現在でも限定された範囲内では融通性のあるシステムは存在するが，割込機能その他のマイクロプログラムと独立に動く部分までを含めて汎用であるというシステムはないと思われる．これは多分レジスタトランスファレベルの論理設計言語システムとマイクロプログラムシミュレータの結合によって解決されるものと思われる．

次にマイクロ命令におけるゲートフィールド割付の自動化の問題がある．これは記述言語で書かれたマイクロプログラム全体を分析し，その中で使われるマイクロコマンドの種類を抽出して必要なマイクロ命令のビット長および各コマンドのフィールドを決定する問題である．

重要ではあるがまだ実用化しているとはいえないテーマにマイクロプログラムの最適化がある．最適化の目標はマイクロプログラムの実行速度の短縮と必要な記憶容量の縮小とがある．この２つは両者が同時に達成される場合もあるが，多くは背反である．プログラム最適化は最近コンパイラなどには採用されているが，マイクロプログラムの最適化の研究も報告されている[4]．その中で使われる手法には不要な操作や重複している操作を除いたり，並列に実行できるものを同

じ命令の中へ移したり，一見異なった働きをしているルーチンの中へ影響のない操作を挿入して共通化を行なうことなどがある．マイクロプログラムのみでなく回路の構造をも変更して行なうような巧妙な最適化は不可能であろうが，今後ますますマイクロプログラム制御の計算機が増え，マイクロプログラムの規模も増大するであろうから自動的に質の高いマイクロプログラムを得るための研究はますます重要である．

シミュレータの高速化という課題に対しては，単にマイクロプログラムのデバッグという目的だけならば前述の μ-SIM-II が TOSBAC 5400 モデル 150 の全命令を一通りシミュレートするのに TOSBAC 5600/170 を用いて 30 分以内で可能であるからそれほど要求は強くないが，オペレーティングシステムも含めたシステムパーフォーマンスの評価という観点からはさらに速いシュミレータが要求され，ハードウェア化されたマイクロプログラムシミュレータなども一つの解答になるだろう[5]．

最後に設計の全てのデザインオートメーションに対して共通にいえることであるが，設計工程の短縮のためにはシステムの TSS 化が有効であり，今後広く採用されるものと思われる．

6. むすび

マイクロプログラムの設計を行なうためのデザインオートメーションシステムに関する概要および記述言語とシミュレーションの方式や問題点について説明してきた．ここで触れなかった方式の中にも多数参考になるものもあろうが，この小文が多少なりとも今後マイクロプログラム関係のシステムを開発される方の参考になれば幸いである．

参考文献

1) S. Young: A Microprogram Simulator, Proc. of DA Workshop, 1971, pp. 68～81.
2) B. R. S. Buckingham: The Controls Automation Systems, Proc. of Simposium on Switching Theory and Automata, 1965, pp. 279～288.
3) 元岡達也：論理設計言語，信学誌，Vol. 50-12, pp. 2353～2360 (1967. 12).
4) R. L. Kleir, et al.: Optimization Strategies for Microprograms, Trans. on Computers, IEEE, Vol. C-50, No. 7, pp. 783～794.
5) M. Yamamoto, et al.: A Microprogrammed Computer Design and Evaluation System, 1st USA-Japan Computer Conference Proc., pp. 139～144 (1972).

(昭和 48 年 2 月 2 日受付)