マイクロプログラム・コントロール方式とその設計†

溝　口　哲　也††

1. まえがき

1951年ケンブリッジ大学のウィルクスによって提案されたマイクロプログラミング技術は，その後多くの人達によって改良されてきているが，コントロールに関する基本的な考え方はほとんど変わっていない．

しかし，実用に際し，(1) マイクロプログラムを記憶する ROS (Read Only Storage) のワード・サイズおよび容量を減らすために，ROS ワードのコード化および番地の共有化等が，(2) ROS のアクセス・タイム・ロスを少なくするために，ROS コントロールの高速化が，(3) システムの仕様の拡大および変更に対処するために，ROS の拡張および書換え方法が，(4) ROS の信頼性改善および障害検出のために，ROS 自身に誤り検出および訂正機能を持たせること，等が考案されている．

2. ROS の語構成（マイクロ命令形式）

マイクロプログラムを記憶する ROS の各語には，1マイクロステップ（通常は1マイクロステップ=1マシン・サイクルであるが，1マイクロステップ=Nマシン・サイクル（Nは正整数）のものもある．）の間に同時に実行される各種の基本的なマイクロオペレーションを指令する情報が含まれているが，これらの情報の表現形式によって，ROS の語構成は，

① ダイレクト・コントロール形式

② エンコード・コントロール形式

さらに

{ダイレクト・エンコード形式
{インダイレクト・エンコード形式

あるいは

{ミニ・エンコード形式
{ハイ・エンコード形式

の2つに分類することができる．

（語句の説明）

ダイレクト・コントロール；

ROS ワードを構成する各ビットがシステムの中の各コントロール・ゲートと1対1に対応するようにしたコントロール方式である．コントロール・ゲート数の少ない小規模なシステムに向いている．

エンコード・コントロール；

ROS ワードを構成するビットをいくつかのフィールドに分割し，各フィールドごとにコード化し，同一のフィールドに属するコードは，相互に排他的な制御信号として働き，システムの中の各コントロール・ゲートに1対1で対応するようにしたコントロール方式である．コントロール・ゲート数の多い，中から大までの規模のシステムに向いている．

ダイレクト・エンコード；同上

インダイレクト・エンコード；

フィールドを構成するビットの意味が，他のフィールドのビットの状態によって変わる方式である．

ミニ・エンコード；

コード化の量の少ないコントロール方式．ダイレクト・コントロールの一部をコード化した比較的小規模なシステム向き．

ハイ・エンコード；

コード化の量の多いコントロール方式．この中には，Standard Computer 社の MLP-900，RCA Spectra 70/45，Honeywell H-4200 等のようにちょうど機械命令語と同じようなフォーマットをしたものがあるが，これらはミニ命令形式と呼ばれている．

a）ダイレクト・コントロール方式

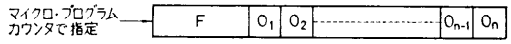
この方式には，エンコード・コントロール方式に比べて

（1） ROS のワード・サイズが大きくなる．（ROS のコスト・アップ）

（2） 拡張性に乏しい．（将来の拡張に備えて，余裕のある語構

† Microprogram control technology and design, by Tetsuya Mizoguchi (Electronic Computer Hardware Development Dept. Tokyo Shibaura Electronic Co., Ltd.)

†† 東京芝浦電気株式会社電算機ハードウェア開発部



F＝マイクロ命令のオペレーション・コード

例えば
- 算術演算
- 論理演算
- シフト・オペレーション
- データ転送

On＝オペランド・フィールド

例えば
- オペレーションの細目指定
- オペランド・アドレス指定

図 1 ミニ命令形式の例

成にするには経済的に不利.)

という大きな欠点があるが，一方

（1） 保守診断が容易，

（2） 変更が容易，

（3） ある種の ROS（たとえばダイオード・マトリクス等）では記憶素子数を減らすことができる，

（4） デコーダが不要，

（5） デコーディングによる時間遅れがない，

などエンコード・コントロール方式よりも優れた点もある．

一般的には，このダイレクト・コントロール方式はコントロール・ゲート数の少ない小規模なシステムに向いている．

b）エンコード・コントロール方式

システムの中に存在するコントロール・ゲートを開閉する制御信号のうち同時に付勢する必要のないもの，あるいは同時に付勢してはいけないもの，つまり，互いに排他的なものをグループ化し，たとえば，同一グループに属する制御信号が M 個ある場合には，$M \leqq 2^m - 1$ (m, M は正整数) を満足する最小の整数 m のビット幅 (m ビット) を有するコントロール・フィールドを M 個の制御信号つまり M 個のコントロール・ゲート用に，ROS ワードに割り付け，コード化したコントロ

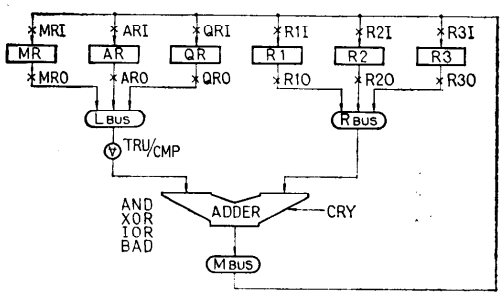
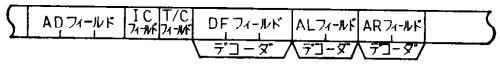


図 2 簡単なブロック・ダイアグラム



AR フィールド

ROSBITS	MICRO-ORDER	FUNCTION
00	NULL	
01	R1O	R1 —RBUS→ ADDER
10	R2O	R2 —RBUS→ ADDER
11	R3O	R3 —RBUS→ ADDER

DF フィールド

ROSBITS	MICRO-ORDER	FUNCTION
000	NULL	
001	R1I	ADDER —MBUS→ R1
010	R2I	ADDER —MBUS→ R2
011	R3I	ADDER —MBUS→ R3
100	MRI	ADDER —MBUS→ MR
101	ARI	ADDER —MBUS→ AR
110	QRI	ADDER —MBUS→ QR
111	UNDEFINED	

AD フィールド

ROSBITS	MICRO-ORDER	FUNCTION
000	NULL	
001	AND	LAI∧RAI AND
010	XOR	LAI∀RAI EXCLUSIVE OR
011	IOR	LAI∨RAI INCLUSIVE OR
100	BAD	LAI+RAI BINARY ADDITION
101 ～ 111	UNDEFINED	但し LAI=Left Adder Input, RAI=Right Adder Input

AL フィールド

ROSBITS	MICRO-ORDER	FUNCTION
00	NULL	
01	MRO	MR —LBUS→ ADDER
10	ARO	AR —LBUS→ ADDER
11	QRO	QR —LBUS→ ADDER

T/C フィールド

ROS-BITS	MICRO-ORDER	FUNCTION
0	TRU	GATE TRUE TO LEFT ADDER INPUT
1	CMP	GATE1'S COMPLEMENT TO LEFT ADDER INPUT

IC フィールド

ROS-BITS	MICRO-ORDER	FUNCTION
0	NULL	NO CARRY
1	HT1	INSERT CARRY (HOT1)

図 3 ROS の語構成の一部

ールである.

たとえば，図2において，MRO, ARO, QROをROSワードのALフィールドに，R1O, R2O, R3OをARフィールドに，MRI, ARI, QRI, R1I, R2I, R3IをDFフィールドに割り付けていくことにより，図3に示すようなROSワードを構成できる.

この方式には，ダイレクト・コントロール方式とは逆に

（1） ROSのワード・サイズを小さくできる，(ダイレクト・コントロール方式の場合に比べて一般に1/3〜1/4に短縮できる.)

（2） 拡張性に富んでいる，(デコーダの金物さえ追加すれば，システムの要求に応じて，割合簡単にコントロール・ゲート数をふやせる.)

という大きな利点があるが，一方

（1） デコーダが必要，

（2） デコーディングによる時間遅れが生じる，

という不利な面もある. IBM System/360など，中規模から大規模な電算機システムには，ほとんどエンコード・コントロール方式が採用されている.

エンコード・コントロール形式のROS語を設計する際の注意としては，システムの仕様変更，追加等でマイクロオペレーションが後で追加になってもデコーダの配線変更が生じないように，余分のデコーダを実装しておくなど設計当初から機能拡張には十分注意を払って，設計しておくことが望ましい.

以上，各種のマイクロ命令形式を述べてきたが，これらは必ずしも排他的ではなく，通常これらを混用する場合が多い.

3. ROSのアドレス指定について

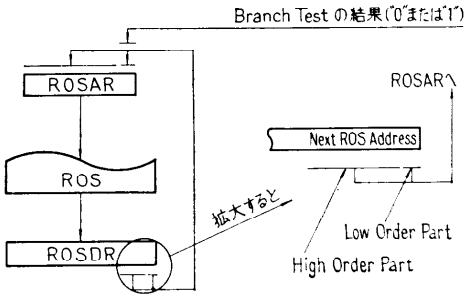
3.1 ROSのアドレス指定法

ROSのアドレス指定法には，次に述べる2つの方法がある.

（1） 現在実行中のマイクロ命令 (ROS語) のシーケンシング・コントロール・フィールドで，次に実行するマイクロ命令のアドレスを指定する. この方法によるとマイクロ命令を実行する順序と無関係にROSに記憶しておくことができる.

（2） マイクロプログラム・カウンタで次に実行するマイクロ命令のアドレスを指定する. この方法では，分岐先のマイクロ命令以外は全て実行される順序に，ROSに記憶されている.

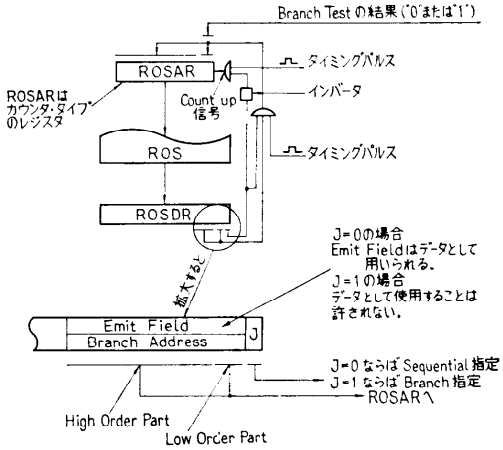
これらの2つの番地指定法を図解すると図4および図5のようになる.



i) 条件付分岐以外の場合
Next ROS Address=(High Order Part)+(Low Order Part)

ii) 条件付分岐の場合
Next ROS Address=(High Order Part)+(Branch Test の結果)

図4 ROSのアドレス指定法 (1) の解説図



i) シーケンシャル指定の場合 (J=0)
Next ROS Address=(ROSAR)+1

ii) 無条件分岐の場合 (J=1)
Next ROS Address=(High Order Part)+(Low Order Part)

iii) 条件付分岐の場合 (J=1)
Next ROS Address=(High Order Part)+(Branch Test の結果)

図5 ROSのアドレス指定法 (2) の解説図

3.2 ROSの分岐アドレスの決め方

マイクロプログラムでよく用いられる分岐の数は，

分岐条件の数によって異なるが，分岐テストによる場合は2ウェイあるいは4ウェイの分岐が一般に用いられる．一方，ファンクショナル分岐と呼ばれるデータ・パスからの情報（たとえば命令語のOPコード）をもとにして分岐する場合には，その数は計算機によって種々様々である．

ここでは，まずROSワードがKビットのブランチ・テスト・フィールドを1個のみ持つ場合を考えてみよう．分岐テストは互いに排他的な 2^K-1 個のテストの中からブランチ・テスト・フィールドで指定された1つのテスト（たとえば，加算結果にオーバーフローが生じたか否か）を選び出して行なわれ，そのテスト結果（"0"または"1"で示される）が分岐アドレス決定の情報として用いられる．図4および図5で示したように，分岐テストの結果をたとえば分岐アドレスの最下位桁の決定に用いれば，Next ROS Addressは次のようになる．

分岐テストの結果が"1"ならば

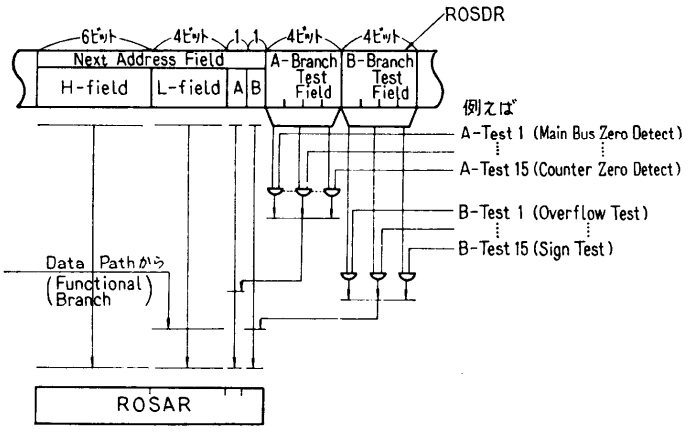
Next ROS Address＝$a_{N-1}a_{N-2}\cdots a_2a_11$

分岐テストの結果が"0"ならば

Next ROS Address＝$a_{N-1}a_{N-2}\cdots a_2a_10$

ただし $a_{N-1}a_{N-2}\cdots a_2a_1$ はROSワードのNビットのアドレス指定フィードの上位N-1ビット．

次に，ROSワードが2つのブランチ・テスト・フィールドをもち，さらにファンクショナル分岐の機能をももつ場合の分岐アドレスの決め方を図6で図解する．



分岐の種類	Next ROS Address
A-Branch のみ指定（2ウェイ分岐）	H+L+ABT+B
B-Branch のみ指定（2ウェイ分岐）	H+L+A+BBT
A-, B-Branch ともに指定（4ウェイ分岐）	H+L+ABT+BBT
Functional Branch のみ指定（16ウェイ分岐）（F-Branch と略称）	H+FB+A+B
A-, F-Branch ともに指定（32ウェイ分岐）	H+FB+ABT+B
B-, F-Branch ともに指定（32ウェイ分岐）	H+FB+A+BBT
A-, B-, F-Branch ともに指定（64ウェイ分岐）	H+FB+ABT+BBT

但し ABT は A-Branch Test の結果（"0" または "1"）を示す．
BBT は B-Branch Test の結果（"0" または "1"）を示す．
FB は Data Path から送られてきた情報そのものである．

図6 ROSの分岐アドレスの決め方の解説図

4. マイクロプログラムの共有（ROSの番地の共有化）

通常のプログラムにおいては，メモリを効率よく使用する一つの手段として，使用頻度の高いプログラムを共有化（サブルーチン化）する技術が一般によく用いられるが，マイクロプログラミングにおいても同様にROSを効率よく使用するために，使用頻度の高いマイクロプログラムあるいはマイクロ命令を共有化する技術がよく用いられる．

共有化にとって重要なことは，共有されるマイクロプログラム（以下，共有ルーチンと呼ぶ）から共有ルーチンを呼び出したマイクロプログラムへリターンするときに使用するアドレス情報（あるいはアドレス情報の一部）を前もって決定し，共有ルーチンに入る前にセーブしておくことである．一般によく用いられる共有化の方法としては，(1) 図7に示すようなリターン・アドレス・レジスタ (ROSRA) を設け，リターン・アドレスをセーブする方法と，(2) 図6で図解したアドレス分岐機能を用いる方法がある．ROSRAを用いる方法では，リターン・アドレス情報として（i）共有ルーチンに入る前のROSアドレス・レジスタ (ROSAR) の内容の一部あるいは全部，（ii）データ・パスからの情報等を使用することができる．共有ルーチンから元のルーチンへリターンするときに，前もってROSRAにセーブしておいたリターン・アドレス情報をROSARにリストアすれば，図8に示すように正しくリターンすることができる．一方，アドレス分岐機能を用いる方法もまた，図9に示すように正しくリターンすることができる．

5. ROSコントロール方式

一般にマイクロプログラム・コントロール方式は高性能計算機に向かないといわれてきたが，その主な理由は次の2点

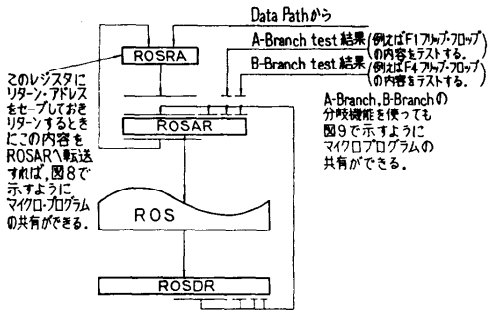


図 7 マイクロプログラムの共有方法の解説図

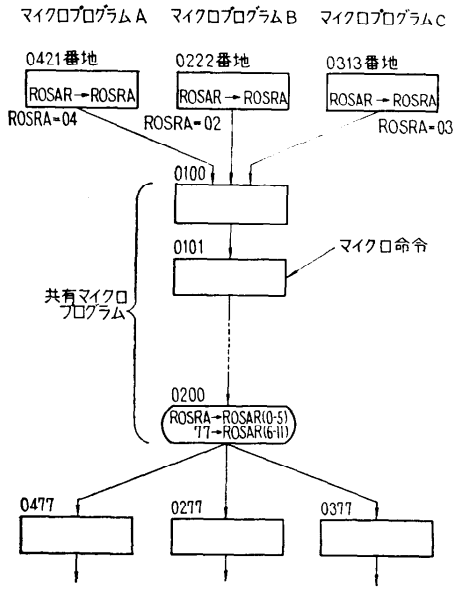


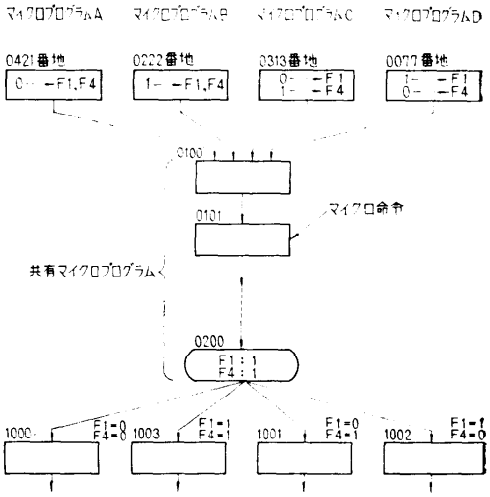
図 8 マイクロプログラムの共有方法 (1) の例

であろう.

(i) ROS コントロールが逐次制御であるために,オーバラップ・オペレーションができない.

(ii) ROS アクセス・タイムが遅い.

(i)の理由に対しては,確かに ROS コントロールは逐次制御であるが,1つのマイクロ命令で実行できる機能をふやすことにより,あるいはたとえば CPU (Central Processing Unit) を複数個のユニットに分割し,それぞれのユニットを独立に ROS でコントロールすることにより,オーバラップ・オペレーションが可能になる.(ii)の理由に対しても高性能計算機のマシン・サイクル・タイム (100 ns 以下) 程度のアクセス・タイムの速い ROS を使用し,かつ ROS アクセス・タイムとマシン・サイクル・タイムをオーバラップさせれば,あるいは ROS ワードの先取りをすれば,ROS アクセス・タイムを実効的にゼロに近付けることができるので,(ii)は高性能計算機に ROS コントロール方式を採用する際の妨げには必ずしもならない.なお,ハードウェア・コントロールとマイクロプログラム・コントロールは排他的なものではないので,IBM System/370 Model 165 のように両者を使い分けて高性能計算機を実現している例もある.



但し,F1 は A-Branch Test に,F4 は B-Branch Test に属しているものとする.
図6参照.

図 9 マイクロプログラムの共有方法 (2) の例

ここでは,ROS コントロール方式の一般論として直列方式と並列方式について述べたあと,並列方式の考え方をさらに進めN個のマイクロ命令を先取りする方式について述べ,最後に,マイクロ命令の実行速度を上げる手段としてよく用いられるマシン・サイクル・タイムを可変制御する方式について述べる.

(1) 直列コントロール方式

この方式では,現在実行中のマイクロ命令が完全に実行を終了するまで,次に実行するマイクロ命令のアドレッシングが行なわれず,従って ROS のアクセス・タイムの分だけマイクロ命令の実行時間が引き延ばされる.もし ROS のアクセス・タイムがマイクロ命令の実行時間に比べて,非常に小さい場合はこの遅れはそれほど問題にはならない.

(2) 並列コントロール方式

現マイクロ命令の実行中に,つまりデータ・パスの動作中に,次に実行するマイクロ命令を読み出

し，分岐が行なわれる場合は予測を行なって，どちらか一方のアドレスで指定されたマイクロ命令を読み出す．予測がはずれたときは，もう一方のアドレスで指定されたマイクロ命令を読み出す．この方法は分岐点が少ない場合あるいはかたよった確率で分岐する場合は良いが，一般的に分岐の多いマイクロプログラム（CPU のマイクロプログラムは分岐点が多い）では予測がはずれるケースが多くなり，時間損失が多くなる．

（3） 直列方式と並列方式の比較（処理速度の比較）[1)]

（i） 系の実行時間が外部条件の制約をうけるとき．

$$D+\sum_{j=0}^{M} A_j < C$$

ただし，

D：全データ処理に要する時間，

A_j：直列方式でJ番目のマイクロ命令のアクセスに要する時間，

M：マクロ命令を実行するのに必要とされるマイクロ命令の数，

C：外部制約条件によって決定されるマクロ命令の最小実行時間．

上式が成立するならば，直列方式の方がよい．

（ii） 系の実行時間が外部条件によって制約されないときは，分岐点での時間損失を考慮に入れても次の関係が成立すれば並列方式の方が優れている．

$$\sum_{i=0}^{N} P_i \cdot T_i < \sum_{j=0}^{M} A_j$$

ただし，

P_i：分岐点で間違った予測をする確率，

T_i：並列方式で，分岐点で間違って予測されることによる損失時間，

N：マクロ命令を実行するのに必要とされるマイクロプログラム中の分岐点の数．

（4） N個のマイクロ命令を先取りする方式

一分岐点での分岐先が最大N個まである，マイクロプログラムにおいても，ROS をN個のモジュールに分割して，同時にN個のマイクロ命令を読み出す機能を設けておけば，分岐点で予測する必要性はなくなり，従って ROS のアクセス・タイムを実効的に，ほとんどゼロに近付けることが可能になる．前にも述べたように CPU のマイクロプログラムには，一般的に非常に多くの分岐点があるが，ほとんどは2ウェイか4ウェイの分岐であるので，図 10 に示すように ROS を4個のモジュールに分割し，4ウェイ・インターリーヴしておけば十分であろう．また，ROS モジュールごとに ROS アドレス・レジスタを設け，同時に，あるいは独立に ROS モジュールをアクセスできるようにすればマイクロ命令の先取り機能はさらに拡張される．

（5） マシン・サイクル・タイムを可変制御する方式

マイクロプログラムの処理時間を短縮し，計算機の処理速度を上げるためには，マイクロプログラムを構成する個々のマイクロ命令のステップ長をできる限り短くする必要がある．通常，マイクロ命令の実行に要する正味の時間はマイクロ命令の種類によって異なっている．

マシン・サイクル・タイムを M_t，マイクロ命令の正味の実行所要時間を E_t とすると，

マイクロステップ長$=n\times M_t \geqq E_t$ の関係式が成り立つ．ただし，同期式の場合は，

M_t＝定時間，n＝正整数，一般には，$n=1$ が多いが，$n\geqq 2$ の場合もある．非同期式の場合は，

M_t＝可変時間，$n=1$.

この式から分るように，マイクロステップ長とマ

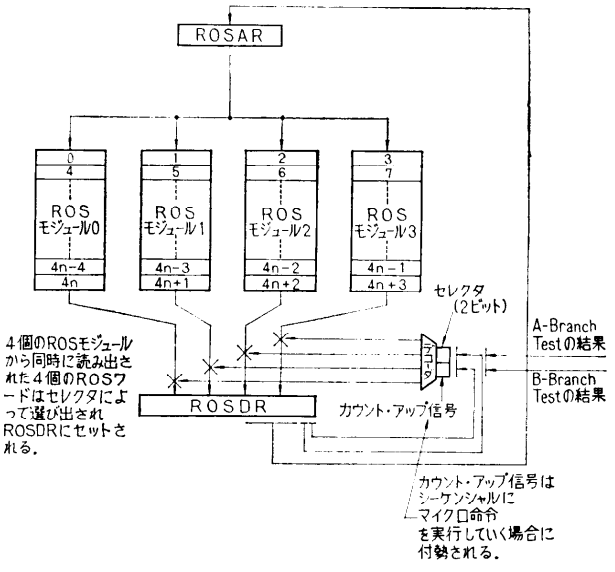
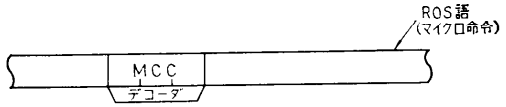


図 10　4ウェイ・インターリーヴした ROS

イクロ命令の実行所要時間 E_t の差の分 ($\Delta T = n \times M_t - E_t$) が時間損失となるのである．特に同期式の場合は M_t が定時間であるため，M_t が可変である非同期式の場合より一般に時間損失が大きくなる．

ここで述べるマシン・サイクル・タイムを可変制御する方式はこのような時間損失 ΔT をできるだけ小さくするために考案された方式で，TOSBAC-3400 シリーズ，TOSBAC-5400 Model 150 等に採用されている．

個々のマイクロ命令のステップ長として，$\Delta T = 0$ を満足する任意のマシン・サイクル・タイムを与えることができれば理想的であるが，経済的理由で実現は難しい．そこでマイクロプログラムの実行中に現われる各マイクロ命令の実行回数と実行所要時間を考慮して，予め定めたいくつかのマシン・サイクル・タイムの中から最も適したもの，つまり ΔT を最小にするマシン・サイクル・タイムをマイクロ命令のステップ長として選び，ROS ワード (マイクロ命令) の MCC (Machine Cycle Timing Control) フィールドに記憶しておき，マイクロ命令を実行するときに，MCC フィールドの情報によりマシン・サイクル・タイムを制御する．図 11 はマシン・サイクル・タイミング・コントロール・フィールドの一例を示す図である．



MCC Field ROS Bits	Micro-order	Machine cycle timing control comment
000	TC0	$E_t \leqq T0$ ならば $M_t = T0$
001	TC1	$T0 < E_t \leqq T1$ ならば $M_t = T1$
010	TC2	$T1 < E_t \leqq T2$ ならば $M_t = T2$
⌇	⌇	⌇
111	TC7	$T6 < E_t \leqq T7$ ならば $M_t = T7$

但し，E_t = マイクロ命令の実行所要時間
M_t = マシン・サイクル時間

図 11　マシン・サイクル・タイミング・コントロール・フィールドの例

6. ROS の拡張及び書換え方法

ROS コントロール方式を採用する場合には，設計当初からシステムの仕様変更や追加等に備えて ROS の拡張および書換え方法を十分考慮しておく必要がある．ROS の拡張には

(1) ROS のワード・サイズを拡張する，(ビット方向への拡張.)

(2) ROS の容量を拡大する，(ワード方向への拡張.)

の 2 つのケースがある．

(1) ROS のワード・サイズを拡張する方法としては，

a) エンコード・コントロール方式を採ることにより ROS ワードの各フィールドに十分余裕をもたせておく，

b) マルチワード・マイクロ命令形式を導入する，

等が考えられる．

(2) ROS の容量を拡大する方法としては，

a) ROS アドレス・レジスタおよび ROS ワードのアドレス指定部に予め 1〜2 ビットの余裕をもたせておく，

b) ベース・アドレス方式を導入する，

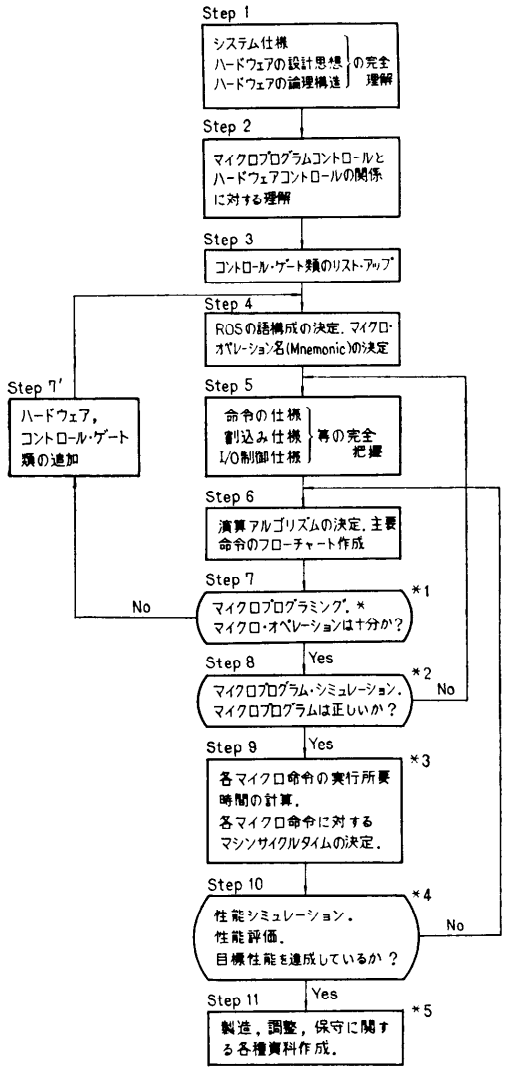
c) 主メモリの中に ROS を設け，しかもアドレス・レジスタを共用し，バウンダリ・レジスタで境界を指定できるようにしておく，

等が考えられる．

ROS の書換えについては，マイクロプログラム方式が本格的に計算機に採用され始めた初期の頃は，記憶媒体をたとえばカードの型で装置から着脱可能にし，フィールドで記憶情報の更新が割合簡単にできる方法等が採られたが，最近では主メモリあるいは I/O 機器 (たとえばディスク・カートリッジ，カセット・テープ等) から記憶情報を簡単にローディングできる WCS (Writable Control Storage) がマイクロプログラム制御メモリとして用いられるようになってきている．たとえば IBM System/370 Model 158 では従来の Model 155 で使用されていた ROS に替って，モノリシック回路による WCS が採用され，設計変更や機能の追加が容易になっている．書換え可能な制御メモリ WCS は従来の読取り専用の制御メモリ ROS に比して，種々の特長を有しているが，中でもダイナミック・マイクロプログラミングの概念を確立したという点で，マイクロプログラミング技術の発展に大きく貢献している．

7. マイクロプログラムの設計手順

効率のよい，システィマティックな設計を行なうには，標準的な手順に沿って，設計するのが望ましい．ここでは，筆者らが採っている設計手順を示そう．



設計の自動化

マイクロプログラムアセンブラにより
- *1 の ROS Address の割付け
- *3 の時間計算及び最適マシンサイクルタイムの割付け
- *5 の ROS Bit pattern の作成

等は自動化できる．
マイクロプログラムシミュレータにより
- *2 のマイクロプログラムシミュレーション
- *4 の性能シミュレーション
- *5 のマイクロプログラムトレース・リスト

等は自動化できる．
＊マイクロプログラムの記述言語としては一般的には
1. マイクロプログラム・コンパイラー言語（メタ・アセンブラ）
2. マイクロプログラム・アセンブリ言語
3. マイクロオペレーション言語

がある．

図 12 マイクロプログラムの設計手順

□各ステップの説明

Step 1：マイクロプログラマはまず，システム仕様およびハードウェアの設計思想を完全に理解した上で，ハードウェアの論理構造（各ゲート，各データ・パス，ユニット間のインターフェイス，タイミング等）を熟知しなければならない．

Step 2：システムの中に同期的に動作する部分と非同期的に動作する部分（CPU と主メモリとデータ・チャネル）が存在するため，マイクロプログラム・コントロールとハードウェア・コントロールが共に採り入れられるケースが多い．

マイクロプログラマは，この両者の関係（たとえば優先度等）を詳しく知っておく必要がある．

Step 3：マイクロプログラマは ROS ワードによって制御されるすべてのコントロール・ゲートのリストを作成しなければならない．このリストの中にはレジスタの入出力ゲート，アダー入出力ゲート，シフト・コントロール用ゲート等が含まれている．

Step 4：Step 3 で得たリストをまず次の 3 つのフィールドに大別する．

a）Static Control Lines

レジスタ入出力ゲート，アダー入出力ゲート，シ

フト・コントロール用ゲート等の各種データ・パス用ゲートを制御する信号群.

b) Sequencing Controls

次に実行する ROS ワードの番地を決定する各種の逐次制御信号群.

c) Emit Field

マイクロプログラマが自由に定数（2進数）を発生できるように ROS ワードの中に確保する必要のある定数フィールド.

ダイレクト・コントロール方式を採用する場合には，上に述べた3つのフィールドに含まれる各種信号を ROS ワードの各ビットに割当てていけばよい.

一方，エンコード・コントロール方式を採用する場合には，Static Control Lines の中の互いに排他的なもの同志をグループ化し，同一グループに属する全ての信号を2進コードで表示できるだけのビット幅をもつサブフィールドを作成する.

したがって，ROS ワードは Static Control Lines フィールドを構成する各サブフィールドと Sequencing Control フィールドと Emit Field で構成されることになる.

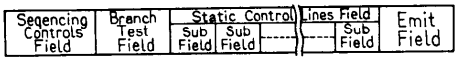


図 13 ROS の語構成（マイクロ命令のフォーマット）

次にコード化の有無にかかわらず，ROS ワードによって与えられる各種信号に対し，マイクロ・オペレーション名を与える.

〔Step 4 での注意〕

システムによっては，あるハードウェアが動作モード（たとえば CPU モードと I/O モード）によって全く別の機能として働く場合があるので，マイクロプログラマはコントロール・フィールドの定義およびフィールドの幅を決定する際には常に動作モードのことも考慮しなければならない.

Step 5: 命令の仕様，割込み仕様，I/O 制御仕様等を見直し，完全把握に努める.

Step 6: 乗除算の詳細な演算方式を決定するとともに，主要命令（四則演算等）のフローチャートを作成する.

Step 7: 欠くことのできないマイクロオペレーションがもれていれば，それを補い，あるいはマイクロオペレーションを新設することにより，効率のよいマイクロプログラミングができるならば，それを追加する. もちろん，不要なマイクロオペレーションがあれば，それを削除することも忘れてはならない.

Step 8: でき上ったマイクロプログラムを，前もって作成したマイクロプログラム・シミュレータ（プログラム）を使って，シミュレーションを行ない，仕様通りの結果が得られるかどうかをチェックする.

Step 9: 各マイクロ命令の実行に要する時間を計算する.

a) マシン・サイクル・タイムを可変に制御する方式が採られている場合，計算結果に応じて，各々のマイクロ命令に対し最適のマシン・サイクル・タイムを割り当てる.

b) マシン・サイクル・タイムが固定（1種類）の場合，計算結果がマシン・サイクル・タイムより小さいか等しいなら，マシン・サイクル・タイムがそのマイクロ命令の実行時間となるが，計算結果がマシン・サイクル・タイムより大きい場合はマシン・サイクル・タイム以下になるように，そのマイクロ命令を複数個のマイクロ命令に分割するか，あるいは1マイクロ命令に基本マシン・サイクル・タイムの整数倍のマシン・サイクル・タイムを与える必要がある.

Step 10: マイクロプログラム・シミュレータを使って各命令の性能を求め，評価する. 目標性能を達成していない場合は，Step 6 にもどり目標性能を達成するように努力しなければならない. 通常，この努力は極めて実りが多い.

Step 11: 製造資料として使う ROS ビット・パターン，調整および保守に役立つマイクロプログラム・トレース・リスト，教育用のマイクロプログラム説明書等を作成する. ROS ビット・パターンおよびマイクロプログラム・トレース・リストはいずれも自動作成できるものである.

8. おわりに

以上，ウィルクス・モデル以後に改良あるいは開発されてきたマイクロプログラム・コントロール方式，およびマイクロプログラムの設計手順について述べてきたが，マイクロプログラミング技術はこれからも，種々の新方式が開発され，また，LSI, WCS 等の出現により，計算機本体，I/O 制御装置，ミニコン，電卓あるいはエミュレーション，ファームウェア，診断等といった分野にますます応用されていくであろう.

（昭和 48 年 3 月 15 日受付）