4-2 マイクロプログラミング

日本電気株式会社 中央研究所

服 部 光 宏

1. ま え が き

マイクロプログラミングは 1951 年ケンブリッジ大学の M. V. Wilkes により“コンピューター制御部の系統のかつ秩序だった設計手段”として提案された。

コンピューターの機械語で記述されたプログラムにおいて、一機械命令は主としてレジスターからレジスターへの情報転送を単位ステップとするステップ列により実行される。このときのレジスター間の情報転送は、一般に加算器や他の論理回路を通して行われる場合もあるし、直接行われる場合もある。

一般にこれらの転送の実行順序は論理回路により制御される（ハードワイヤードロジックと呼ばれる）が、Wilkes はプログラム実行がプログラムの構成要素である機械命令を順次実行するのになぞらえて、機械命令の実行を制御記憶に記憶された上記ステップ列の実行と考えてマイクロプログラムを定義し、その実現方法としてダイオードマトリックスによる読出し専用メモリー (ROM, read only memory) を提案した。

その後マイクロプログラム方式のコンピューターは 1950 年代後半から 1960 年代前半にかけて多くの専門家の間で研究されるようになり、わが国でも KT パイロットコンピューターの試作が発表されている。また同じ順スタードロジック方式（マイクロプログラムと通常のプログラムが主記憶を共有する方式）が提案され、この方式にもとづくコンピューターが発表されている。

マイクロプログラム方式が商用として実現され、その重要さが真に認識されたのは、1964 年の IBM System /360 および Tucker¹²⁾ によるエミュレーションの概念発表の後、すなわちコンピューターに対する一般の注目がコンピューター CPU から入出力機器やそれらを利用したソフトウェアに向けられるようになり、ソフトウェアの蓄積およびソフトウェア開発の重要さが認識されるようになってからである。

1960 年代終りからの半導体 LSI 技術の進歩により、

書替え可能制御メモリー (read-write control memory writable control store) が商品として実用化されるようになり、Flynn により論じられた動的マイクロプログラミング (dynamic microprogramming) の応用が最近特に注目されるようになった。現在のマイクロプログラミングは Wilkes の提案と本質的には変わっていないが、動的マイクロプログラミングは Wilkes の意図にはなかったものである¹³⁾。

動的マイクロプログラミングも含めて、Husson は次のようにマイクロプログラミングを定義している⁶⁾。マイクロプログラミングとはデータ処理システムの制御機能を設計し実現する技術であり、複数の信号群の列により固定したあるいは動的に可変なデータ処理機能を解釈実行するものである。これら複数の信号群は語単位に形成され固定したあるいは動的に可変な制御メモリーに記憶され、実行しつつある機能間の情報の流れを制御する信号群の状態を表示するとともに、状態間の秩序正しい遷移をも制御するものである。

以下にマイクロプログラミングの特徴、応用および今後の研究課題について簡単に述べる。

2. マイクロプログラミングの特徴

マイクロプログラミングはハードワイヤードロジックによるコンピューター制御部をマイクロ命令で記述されたプログラムにより実現しようとするものであるから、その特徴は一般にハードウェアとソフトウェアの中間に位置するものと考えられる。

2.1 ハードウェアとの比較

(1) 設計の統一性と容易性

コンピューター設計に際し、マイクロプログラム方式という確立された方式を採用することにより設計が統一になる。また一度マイクロ命令セット、あるいはそのプロセサーがきまれば、対象とする制御部設計はマイクロプログラミングという作業の下で進められるため、各設計段階における文書類の作成をはじめ、論理設計、実装設計、デバッグおよび保守などにおいて非常に有利である。

(2) 融通性、可変性

“4-2 Microprogramming” by Mitsuhiro Hattori (Central Research Laboratories, Nippon Electric Co., Ltd., Kawasaki)

コンピューターシステムにおいてシステムの融通性、可変性は本質的に重要な問題であり、これらへの要請は、システム設計時あるいはシステム開発後に種々のかたちであられる。

たとえば、1つのシステムの機器構成を変えて使用したい場合、下位のシステムも同時に使用したい場合、あるいは上位のシステムやまったく新しいシステムを導入する場合などにおいて、ソフトウェアの互換性を保つ必要が生じる。また設計時の仕様変更や設計ミスにともなう設計変更、制御部の制御方法の改良、新しい命令や新しい端末機器の付加が必要になることもあるし、コンピューターを1つの利用目的だけにあった専用コンピューターに変えたり、1台のコンピューターを複数個の目的に応じて最適のアーキテクチャーを選択して使用することなどが要請されることもある。

マイクロプログラム方式は、マイクロプログラム部分の静的あるいは動的な変更、置換えにより融通性、可変性をもてるため、これらの要請に応じられる可能性が最も大きいと考えられている。

(3) 教育の容易さ

システムが複雑になるに従ってそのシステムの開発、保守が困難になり、システムの作成者、保守要員の教育必要度が増してくる。マイクロプログラミングでは従来のハードウェアによるものと比較して統一的に扱えるため、これら要員教育が容易になる。またハードウェアとソフトウェアの間にマイクロプログラミングという1つの階層を設けたことにより、従来ハードウェアに深い知識をもつ専門家の間で議論されていたコンピューターアーキテクチャーについて一般に教育することが可能になる。

(4) 実行効率

ハードワイヤード方式のコンピューターの制御部では、必要ならば部分的にハードウェアを増やすことにより並列演算を行わせることができるのに対し、マイクロプログラム方式ではマイクロ命令が逐次処理されるため、一般にマイクロプログラム方式におけるコンピューターの処理効率は良くないと考えられる。

(5) コスト

一定の機能をマイクロプログラムで実現する場合、一般に制御記憶のアクセスタイムは主記憶のそれよりも速くする必要があるため、構成が単純で固定したシステムや融通性や可変性が要求されないシステムでは、ハードウェアコストが高つくき、システムのコストパフォーマンスが良くないとされている。しかしながらマイクロプログラミング方式ではLSI化できる部分が多く、最近のLSI技術の進歩により、この考えは必ずしも正しく

ないと見るむきもある。

(6) 操作効率

ハードウェア機能をマイクロプログラムで実現することによりシステムライブラリーが増加し、システムモニターが複雑化するため、これらの管理面でシステムプログラマーやオペレーターの負担が増すことになる。

2.2 ソフトウェアとの比較

(1) 実効効率

一般にマイクロプログラム方式のコンピューターでは、1つの機械命令は命令フェッチ、命令解析、オペランドフェッチ、命令実行の順にマイクロプログラムルーチンを実行することにより解釈実行されるから、機械命令で記述されたプログラムの実行に対してマイクロプログラムの実行ははるかに速いといえる。

(2) 階層レベル

上記のように機械命令はマイクロプログラムの実行により解釈実行されることは階層レベルの違いを意味している。このことはコンピューターの故障診断や、タイムシェアリングシステムにおける機密保護や記憶域保護に必要な機能を実現するのにマイクロプログラミングが適している要因にもなっている（他のユーザーやソフトウェアによりマイクロプログラムがアクセスされたりこわされたりすることがない）。

(3) マイクロプログラム生産性

一般に1つのマイクロ命令は、その構成要素であるサブフィールドで指定されるマイクロコマンドの組合せにより構成されており、おのおののマイクロコマンドの機能単位は機械命令の機能単位と比較して非常に小さく、またマイクロコーディングするにはハードウェアの知識を必要とする。したがって最近のようにソフトウェア作成が高級プログラミング言語で行われるのと比較して、マイクロプログラム生産性は非常に悪い。

3. マイクロプログラミングの応用

前章で述べたマイクロプログラミングの特徴は、簡約すると、ハードウェアと比較して融通性、可変性に富むことと、ソフトウェアと比較して実行効率が優れていることである。したがってマイクロプログラミングのこのような利点を生かした多くの利用方法が考えられており、そのうち一般のコンピューター制御、入出力機器制御およびコンピューターの故障診断などについては、すでに実用化されている。これらの応用を考える際に、概念的に最も本質的なものは（広義の）エミュレーションであるといつてよい。

コンピューターシステムをグレードアップする（上位システムで置き換える）際に、または新しいコンピュー

ターシステムを開発あるいは導入する際に最も重要な問題として、いままで使用されてきたコンピューターシステムとの互換性をいかに保つかという問題がある。この問題へのアプローチとしては、既存ソフトウェアの機械命令を新しいコンピューターの命令でシミュレートする方法と、既存ソフトウェアを新しいコンピューター上のソフトウェアに変換する方法とが考えられてきた。前者は実行効率の点で、また後者は技術的な困難さのために、特殊な場合を除いて実用化されていない。

(狭義の)エミュレーションとは他のコンピューターの機械語を自分のコンピューターのソフトウェアとハードウェアおよびマイクロプログラムによりシミュレートすることである。(狭義の)エミュレーター実現にはマイクロプログラムの利用が不可欠である。実際問題としてエミュレーター実現には、コンピューター間の命令セットの違いのほかに、システム構成、入出力機器機構、入出力媒体のフォーマット、エラー処理機構などの違いが問題になることが多い。

狭義のエミュレーション対象機能である機械命令の持つ機能を一般のエミュレーション対象機能、あるいは対象システムにまで拡張解釈し(これを仮想機能あるいは仮想システムと呼ぶ)、シミュレートするコンピューター(ホストマシンあるいはホストシステムと呼ばれる)の上に仮想システム(仮想機能)を実現することを(広義の)エミュレーションと呼ぶ。広義のエミュレーションにおける仮想機能の例としては、

- (1) 浮動小数点演算
- (2) あるコンピューターの科学計算用サブルーチン
- (3) 磁気ディスク制御機能
- (4) FORTRAN コンパイラ
- (5) オペレーティングシステムのシステムモニター
- (6) オンラインシステム等の応用システム
- (7) あるコンピューターの上にインプリメントされたすべてのソフトウェアシステム

などがそのレベルに応じて考えられる。

4. マイクロプログラミングの研究課題

(1) ソフトウェアへの寄与

Opler は 1967 年コンピューターの制御記憶に格納された特殊目的のマイクロプログラムを“ファームウェア”(firmware)と名づけ、ファームウェアが第4世代ソフトウェアの特に効率の面で重要な役割を演ずることになると予想した⁹⁾。この予想は LSI 技術の進歩による slow-write fast-read (SW/FR) メモリーの実現に期待したも

のであるが、現段階でのファームウェア利用は Opler の予想したほどにはいたっていない(第4世代では、マンパワーの点でファームウェアへの投資がソフトウェアへの投資を上まわると考えた)。

効率の良いソフトウェアを実現するうえでファームウェアが期待できるものとしては、よく使用される科学計算用サブルーチンや特殊なアプリケーションパッケージ(たとえばソートプログラム)などがある。また 1960 年代のはじめから研究されてきた高級プログラミング言語用コンパイラの全ハードウェア化は融通性や信頼性の点で問題があったが、マイクロプログラミングの実用化により別のかたちで現実的なものになりつつある。

たとえば EULER という ALGOL に似た言語や APL 言語のプロセサーのファームウェア化や、会話型プログラミングシステムにおいて、PL/I の算術演算の中間言語を解釈実行する命令を付加した例などがある。また 1972 年には FORTRAN や COBOL などのプログラミング言語に応じてアーキテクチャーを変えられるコンピューターシステムとしてバロース B 1700 が発表されている。

さらにデーター管理システムや速い応答時間を必要とする各種のアプリケーションシステムを指向したコンピューターについても研究されているが、これには技術的な問題のほかに保守や管理の面でも大きな障害を含んでいる。

(2) 多重プログラミング, 多重処理

多重プログラミングはコンピューターリソースの効率良い利用法として広く利用されている。また多重処理は実行効率を上げるためと、複数個の同型プロセサーの組合せによりシステム全体の性能を向上させるために考えられてきた。これらは元来マイクロプログラミングと無関係に発展したものであるが、マイクロプログラミングにより、これらの処理に固有な非同期処理やプログラム実行管理に適したアーキテクチャーを構成することが考えられる。また特殊なアプリケーションシステムを指向した大規模な多重処理システムを構成する手法として、複数個の特殊化された(マイクロプログラミングにより個々に最適化された)プロセサーの組合せにより全体の性能を上げることが考えられている。

(3) 障害管理と信頼性向上

マイクロプログラミングによるコンピューターの故障診断は、

- (a) 故障および故障箇所検出の分解能が良い。
- (b) ソフトウェアによる診断と比較して検出速度が速く、必要な診断プログラムも少なくすむ。
- (c) 診断時に動作する必要のあるハードウェアが

ソフトウェアによる場合と比較して少なくすむ。

- (d) 診断速度が速いため、またソフトウェアと階層レベルが異なるため、ユーザープログラム実行時に診断プログラムを組込むことが可能である。

などの長所をもつため、広く一般に採用されている。また診断が容易で再実行も容易な誤動作については自動再実行も可能である。

さらに多数のユーザーが共有して使用するシステムでは、誤動作や誤操作の対策としてデータ保護のために特殊なアドレッシング機構や特権命令が設けられている。今後のマイクロプログラミングの応用として、タイムシェアリングシステム、オンラインのアプリケーションシステムや制御用コンピューターを用いたシステムなど、障害予防や障害時対策機能がシステムを実現するうえで本質的に重要となるものについて、その高度な信頼性を保つための研究が必要である。

(4) マイクロプログラムの生産性

上記のようなマイクロプログラミングの応用が実用化されるためには、効率の良いマイクロプログラムを効率良く生産し、保守することが必要である。現在マイクロプログラム作成には主として専用あるいは汎用のマイクロプログラムアセンブラーとマイクロプログラムシミュレーターを中心とした作成システムが使用されている。

このような作成システムを使用した場合でも、マイクロプログラムの生産性はソフトウェアの生産性と比較して極端に悪く、また保守にも問題がある（フィールド長の短い特殊なマイクロプログラムプロセッサについては高級言語によるマイクロプログラミング支援システムが考えられている）。したがってマイクロプログラムの生産性をソフトウェアの生産性に近づけるためには、マイクロプログラム記述、最適化、デバッグシステムなどの研究が必要であり、これについての試みがいくつか発表されている。

5. む す び

マイクロプログラミングの発展、ハードウェア、ソフ

トウェアと比較してのマイクロプログラミングの特徴、応用としてのエミュレーションおよび今後の研究課題として話題になっているものをいくつか述べた。

マイクロプログラミングの概念が発表されてからすでに 20 年以上経過しており、その応用範囲もその間のソフトウェアおよびハードウェアの進歩により広がってきた。しかしながらマイクロプログラミングがいわゆるコンピューターハードウェア（中央演算処理装置や入出力機器）の専門家以外の人々の間で広く注目されるようになったのは 1970 年代になってからであり、その意味で今後新しい利用面が考えられる可能性がある。

おわりに、より系統的でありかつ詳細な解説を望まれる読者のために解説的な文献および参考文献が多くっている文献を上げておいた。本稿がこれらを読むための発端になれば幸いである。

(昭和 49 年 1 月 12 日受付)

〔参 考 文 献〕

- 1) B. Beizer: The Architecture and Engineering of Digital Computer Complexes Vol. 1, Plenum Press, New York (1971) 239-292
- 2) Y. Chu: Computer Organization and Microprogramming, Prentice-Hall, Englewood Cliffs, New Jersey (1972)
- 3) J. A. Clapp: The Application of Microprogramming Technology, ACM SIGMICRO Newsletter, **3**, 1 (1972) 8-47
- 4) J. A. Clapp: Annotated Microprogramming Bibliography, ACM SIGMICRO Newsletter, **3**, 2 (1972) 3-38
- 5) D. M. Davies: Readings in Microprogramming, IBM Syst J., **11**, 1 (1972) 16-38
- 6) S. S. Husson: Microprogramming; Principles and Practices, Prentice-Hall, Englewood Cliffs, New Jersey (1970) 1-144 (元岡達監訳: マイクロプログラミング, 日本経営出版会, 1973)
- 7) L. H. Jones, et al.: An Annotated Bibliography on Microprogramming Late 1969—Early 1972, ACM SIGMICRO Newsletter, **3**, 2 (1972) 39-55
- 8) マイクロプログラミング特集号, 情報処理, **14**, 6 (1973)
- 9) A. Opler: Fourth Generation Software, Datamation, **13**, 1 (1967) 22-24
- 10) R. F. Rosin, G. Frieder & R. H. Eckhouse: An Environment for Research in Microprogramming and Emulation, Comm. ACM, **15**, 8 (1972) 748-760
- 11) Special Issue on Microprogramming, IEEE Trans., **C-20**, 7 (1971)
- 12) S. G. Tucker: Emulation of Large Systems, Comm. ACM, **8**, 12 (1965) 753-761
- 13) M. V. Wilkes: The Growth of Interest in Microprogramming; A Literature Survey, Computing Surveys, **1**, 3 (1969) 139-145