座　談　会

マイクロプログラミングの現状と将来の展望

萩原　宏[1)]†，相磯秀夫[2)]，飯塚　肇[3)]，小高俊彦[4)]，
竹内廣文[5)]，溝口哲也[6)]，宇都公訓[7)]††

萩原(司会)　本日はお忙しいところを，どうもありがとうございました．マイクロプログラミングの現状と将来ということですので，まず過去を振り返って，それから現状について話合っていただき，将来の展望あるいは夢というようなものを考えてみたいと思います．

マイクロプログラミングの発達

萩原　マイクロプログラミングの過去をざっと眺めてみますと，M. V. Wilkesの論文が出たのが1951年で，実際に EDSAC 2 が完成したのが，たしか 1957年だと思います．それからヨーロッパの研究機関でマイクロプログラミングをとり入れた機械が作られるようになり，アメリカでは 58年から 60年頃に M. V. Wilkes のモデルに準じたような計算機がいくつか発表されているようです．その後61年から64年まで，アメリカ，英国，イタリア，日本，ソ連，オーストラリア，フランスなど世界のあちこちで興味が持たれ研究されたようです．1964年の Datamation の2月号で L. D. Amdahl は stored logic という言葉を使っています．具体的には，例えば PB 440 とか TRW 133 などで実現されていますが，現在の言葉で言えば，これらはダイナミック・マイクロプログラミングの機械だと思います．当時それが発展しなかったのは，制御記憶にコアメモリを使っていて，マイクロプログラム制御の速度が非常に遅かったからです．

1964年になりますと，IBM のシステム 360 で固定記憶 (Read-Only Storage) にマイクロプログラムを入れた制御方式が採用され，その後あちこちで盛んに使われるようになりました．第2世代の機械のエミュレーションということにも都合がよかったようです．1967年 Datamation の1月号で A. Opler がファームウェアという言葉を作り出して，再びマイクロプログラム制御計算機の特徴が見直されてきたという感じがします．

それから S. S. Husson がマイクロプログラムに関する部厚い本を書き，1昨年7月，IEEE の Transactionでマイクロプログラミングが特集されたということで今日に到っています．

日本におけるマイクロプログラミングについて，溝口さん，何かございませんか．

溝口　私どもの会社では，萩原先生の御指導をいただき，KTP (KTパイロット) という我が国最初のマイクロプログラム制御方式の計算機を開発いたしました．この KTP の固定記憶の内容はパンチカードの穴の有無によって与える方式を採っていましたので，命令の仕様変更，追加等が簡単に行なえるという，当時としては画期的な計算機だったように思います．このKTPをパイロット・モデルとして，その後 TOSBAC-3400 シリーズが生まれ，さらに小型の TOSBAC-5100 シリーズ，通信制御装置 DN-231 等が続々と開発されました．昨年 TOSBAC-5400 の 150 を出すなど，既に 10 年以上にわたって各種のマイクロプログラム制御の計算機を世の中に送り出しておりますが，今後ともこの方向は大きくは変らず，大型機にも積極的にマイクロプログラム制御を採り入れて行くことになると思います．

萩原　小高さん，何かございませんか．

小高　日立では，8000 シリーズはほとんどすべてマイクロプログラム制御です．8300, 8400, 8500 を66年に，続いて 8700, 8350, 8450, 8250 を発表しま

1) 京都大学，2) 慶応義塾大学，3) 電子技術総合研究所，4) 日立製作所(株)，5) 日本アイ・ビー・エム(株)，6) 早稲田大学
† 司会，†† 幹事

したが，これらはすべてマイクロプログラミングを応用しております．8800 では，簡単な命令は，ワイアード・ロジック (wired logic)，複雑なものはマイクロプログラム・ロジックになっています．8500 まではすべて read-only タイプだったんですが，8700 ではライタブル (writable: 書き換え可能な) なワイアメモリを使っています．8350 では一部使っています．8250 は IC メモリを使ったマイクロプログラミングを採用しています．

大型の計算機も思い切ってマイクロプログラムでやったものですから，開発時点で難しい問題をいくつか解決しなければなりませんでした．これからもライタブルな制御記憶 (Control Storage) を使用したマイクロプログラム計算機を開発していくことになるだろうと考えています．

萩原　竹内さんのところはどんな具合いですか．

竹内　システム 370 には，158，168，135，145，125 などのモデルがありますが，全面的にマイクロプログラムを使っております．しかも，168 の一部を除いてすべて WCS (Writable Control Storage: 書換え可能な制御記憶) を使用しています．この方向は今後も変らないでしょうね．

萩原　飯塚さん，現状について何か御意見はございませんか．

飯塚　IBM が 360 を出した時点がマイクロプログラミングの第2世代とすれば，現在は第3世代に入ったような感じを持っています．今までと違う点は，マイクロプログラム制御の利点の中でもフレキシビリティがとくに注目されていますね．ですから，マイクロプログラム制御の重要性が高まっているという気がしていて，例えば，小型機をマイクロプログラムで何個も組合わせて複合プロセッサタイプの大型機を作るとか，その辺から新しい計算機システムが生まれる可能性が出てきているようです．

マイクロプログラミングの利点

溝口　マイクロプログラムによる制御は，コスト／パフォーマンスとか，柔軟性とか，あるいは，互換性とか保守性などの点でハードウェアによる制御に比べて優れていますが，なかでも小型から大型まで one machine concept にもとづく計算機シリーズの開発が比較的容易になってきた点が大きいと考えています．メーカから見ると，ファミリ・シリーズ内ではハードウェア，ソフトウェアともかなりの部分が共通に使用できるという意味で重複投資が避けられるし，また，ユーザにとっては，大型機と同一の仕様をもつ計算機システムを安く手に入れることができるし，ファミリ・シリーズ内では互換性があるということで，ユーザ自身のプログラムを変更することなく段階的にシステムのグレード・アップができるという大きなメリットがありますね．

竹内　それから，アーキテクチャがまだきちんと決まっていない時点で製品を開発して行く上で融通性があるし変更も可能である，そういう点で非常にいいんじゃないかと思いますね．

相磯　そういう意味で，メーカにとってはシステムの寿命が長くなる．つまり，実装だけを変えれば新しい計算機になり得るというところが非常に魅力になっているんじゃないでしょうか．また，既存のプログラムを実行できるというエミュレーション機能が，将来開発する計算機にとってはエッセンシャルであると思われますから，それにうまく対処できる計算機という意味でも，マイクロプログラミングは重要だということでしょうね．

萩原　ハードウェアの設計もずっと楽ですね．

竹内　とくに設計変更ですね．ハードウェアの設計変更はゼロにはならないでしょうから，それをマイクロプログラムの変更でカバーする，いわんやライタブルになれば，書き込みのためのメディアを変えてやるだけで非常に安価に技術変更を受け容れることができます．メーカにとっても魅力があると思いますね．

相磯　マイクロプログラミングを採用することによって，例えば，それぞれの計算機で違っていたコントロールの複雑なところは全部マイクロプログラムに吸収できます．その他の部分は LSI に適していて標準化できる．だから，将来の計算機は LSI だと言われますけれども，そういう意味でもうまくフィットしているように見えますね．

萩原　保守の点からいって，いかがでしょうか．

竹内　保守の点からいっても非常に優れていますね，マイクロ診断ができますから．

萩原　あのような診断はマイクロプログラミングを使わないかぎり考えられなかったことですね．

竹内　たとえば，ディスク・ドライブの1つが故障しても，システム全体を停止させずにテストをやれる，これはやっぱりマイクロプログラムのお蔭だと思います．ハードウェアでは不可能に近いでしょう．

萩原　ハードウェアで実現しようとすれば，とても

膨大なものになりますね.

飯塚 メーカさんの利点が挙げられたのですが, 我我のような研究所, どちらかというと使う側からは, マイクロプログラミングの最大の利点は計算機の非常に細かいところを制御できることですね. 昔は研究所でもいろいろ計算機を組立てていましたけれども, 今はとても大変なことになって, ちょっと作れません. その点マイクロプログラム計算機は細かいところまでオーガニゼーションがいじれて, コンピュータ・オーガニゼーションの研究にとても便利ですね.

萩原 12年ばかり前にKTPを作ったとき, その通りのことを実感しました.

相磯 そういう意味で, これから大学などで, コンピュータ・オーガニゼーションの研究が盛んになるんじゃないかと思っています. アメリカではそろそろそういう傾向が現われているようです.

宇都宮 昔はセンターの計算機を使って研究できましたけれども, 最近はマイクロプログラミングの計算機がないとやりたいこともやれない時代になったような気がします. とくに我々のような研究室ですと.

萩原 ということはWCSをもった計算機ということですね.

宇都宮 はい, 手軽に変更可能ということで.

萩原 マイクロプログラムの機械の教材としての価値について, 相磯先生, いかがでしょうか.

相磯 マイクロプログラミングをユーザにオープンすれば, コントロールに関する論理設計をユーザがやることと同じなんですね. 以前から, 私はソフトとハードの設計は同じだと言ってるんですけれども, 一般のソフトウェアの設計者はどうもハードウェアはとっつきにくいと言いますね. これからのソフトウェア, とくに制御プログラムを書く人はマイクロプログラム, つまりハードウェアのことを知らなければだめだろうし, また, そうあるべきだと思いますね.

そういう意味でも, ハードとソフトのちょうど接点になるマイクロプログラムは計算機を教育する上で大変いい教材になるだろうと思います.

それから, もう一つ注意しなければならないのは, ワイアード・ロジックとマイクロプログラムの関係が割合強調されていないんじゃないかと思えるんです. ワイアード・ロジックで設計することもマイクロプログラムで設計することもまったく同じなんですね.

溝口 私もワイアード・ロジックとマイクロプログラミングとはまったく同じだと思うんです. 機械語を一連のマイクロオペレーションに分解していく時点でマイクロプログラムにしようがワイアード・ロジックにしようが同じなんですね.

相磯 私どもの卒論の学生に設計させるとき, まずワイアード・ロジックでやらせておいて, マイクロプログラムに変換させるんです. ほぼ1対1に対応していますよ.

小高 マイクロプログラムの流れ図ですと, 論理図よりはるかに判りやすいですね.

宇都宮 教育の効率という立場からはマイクロプログラムの方が優れていますね.

竹内 実際, 簡単にインプリメントできて結果を見ることができますからね. いろいろなマイクロプログラムを書いて動かしてみる, そういった意味では, WCS がいいですね.

宇都宮 大学としてはWCSの計算機を入れて, マイクロプログラムを組ませ, 実際に動かしてみることが是非必要ですね.

飯塚 今はそういう意味でいい計算機がありませんね. 一応ライタブルになってはいても, 実際買おうとするといろいろ制限条件がついていてうまくいかない. もう少し都合のいいのが欲しいですね.

宇都宮 確かにそういう目的をはっきり意識して設計された機械はありませんね.

マイクロプログラミングの問題点

相磯 マイクロプログラミングの利点はたくさん挙げられましたけれども, 欠点として, 経済性の問題, もう一つ, 速度が遅いんじゃないかという心配, この2つをどうお考えになっていますか. たとえば, ICメモリを使ってユーザ・マイクロプログラミングを実現したとして, コストはどのように評価されるんでしょうか.

小高 IC メモリは現時点ではかなり高い. 大型計算機でマシンサイクルを上げ, それに見合うようにマイクロプログラミングした場合, ライタブルで, たとえば, 32 キロバイトといった容量では, かなり難しいことは確かです. 大型計算機ではまだまだ出来ないという気がします. 小型機ですとマシンサイクルが遅いですから, TTL の IC メモリが安いので, 大容量のものを使えば可能だと思いますけれども.

宇都宮 IC メモリや LSI メモリは安くなる可能性が強くて, とくに自動設計が盛んになればかなり安くなりそうですので, マイクロプログラミングの将来は

ますます有望だと思いますね.

竹内　WCSで使うRAS (Random Access Storage) は同じものを作ればいいのですから，部品当りのコストを非常に安くできるわけですね.

相磯　それからもう一つ，スピードという点で，主記憶 (Main Storage) も制御記憶も IC ということになると，スピードの差でバランスがとれるかどうかという点はいかがですか．つまり，どれ位のスピードの差があればバランスがとれるかということですね.

いままでの計算機では，コアメモリで大体1マイクロ秒近辺のサイクルタイムですと，固定記憶は約数百ナノ秒で，数倍の開きがある．それでいて結構バランスがとれているように思えるんですけれども，両方とも IC メモリになってしまうと，どういうことになるんですか．IBM の場合，どの位の差があるんですか.

竹内　370 の例でいいますと，一口では言えませんが，135, 145, これらは主記憶，制御記憶ともまったく同じテクノロジ，同じスピードです．135 の場合，マイクロ命令には 275 から 1485 ナノ秒ぐらいまでのスピードの差があります．モデル 158 は主記憶には FET の IC メモリを使っているんですが，スピードが遅いんで，制御記憶の方はバイポーラ型のを使っています．スピードはサイクルタイムで 72 ビット 115 ナノ秒です．それに対して主記憶のスピードは，キャッシュ・メモリを持っていますので，どう評価していいのか分りません．168 の制御記憶も，主記憶の8バイト 480 ナノ秒に対し，サイクルタイム 80 ナノ秒とかいった近辺です．このモデルもキャッシュ・メモリを持っています．125 の場合ですと，CPU の部分だけを司るインストラクション・プロセッサ，マイクロプログラムの制御記憶へのロードを司るサービスプロセッサ，それからI/O用のプロセッサの3つのプロセッサがあって，全部違ったマイクロコードを使っていますが，スピードは主記憶も含めて大体 480 ナノ秒ぐらいです.

溝口　360 では，主記憶と固定記憶のスピードの比率は2倍から4倍ですね.

竹内　システム 360/20 の場合ですと，固定記憶のスピードは 600 ナノ秒です．それに対して主記憶が 3.6 マイクロ秒ですから，6倍ぐらいですね．40 では，固定記憶のスピードは 625 ナノ秒で，主記憶が 2.5 マイクロ秒ですから，4倍です.

溝口　高性能計算機にマイクロプログラム制御を採り入れた場合は，例えば，命令の先取り，解読，アドレスのトランスレーション等を司る先行制御ユニットと，命令を司るユニットとを別々のマイクロプログラムで独立に制御する方式が採られているのではないですか.

小高　そうですね．フィールドを分けてやっています．先行制御できる命令とそうでないものがありますので，完全に独立というわけにはいかないんです．今後性能を上げるためには，どうしてもそういう方向に行かなければならないと思います.

萩原　高速化するために先行制御をやるとするとどうしても独立のコントロールにしなきゃ，先行制御のうまみを発揮できないということなんでしょうね.

マイクロプログラミングと入出力

相磯　CPU とマイクロプログラムは非常に理解が容易なんですけれども，入出力の制御とマイクロプログラムについてはどういうふうにお考えですか.

竹内　例えば，遠隔に働いている通信制御装置は手足がないんですね．そのような機械はどうして診断したらいいのか．本来だったら，CPU の方からいろんなメッセージがきてそれに対してこの通信制御装置が処理を行なうわけですけれども，そのメッセージを遠隔で作ってやらなければいけないわけですよ．そういったメッセージを CE (Customer Engineer) がジェネレートするのにマイクロプログラミングが有効ですね．要するに，I/O のついている CPU はマイクロプログラミングを使わなくても診断できるんです．また，手足のない I/O 自身の診断はマイクロプログラミングを利用しなくちゃ十分な診断は難しいですね.

相磯　もう1つ，とくにユーザからみると，I/O に関してはインタフェイスの標準化ということがあると思うんですが.

溝口　データ・チャネルと I/O 制御装置との間のインタフェイスはすでに標準化されているわけですが，従来，I/O 制御装置については，I/O 機器ごとに別々に設計され，標準化されておりませんでした．しかし最近では，I/O 制御装置にマイクロプログラム制御が採り入れられ，設計も動作も簡単化され，いろいろな I/O 機器に対して，マイクロプログラムのみが異なる制御装置が開発されています.

私どもの会社でも，TOSBAC-5600 あるいは TOSBAC-5400/150 用に MPC (Microprogrammed Peripheral Controller) を開発しましたが，これなどはマイクロプログラムのみを入れ替えることにより磁

気ディスク装置としても，磁気テープ装置としても用いることができます．このように，標準化された制御装置は I/O 機器ごとに設計していた制御装置に比べて，スケール・メリット等を考慮に入れると，はるかに安いコストで製作できるし，保守についても標準化できるなど，非常に大きなメリットがありますね．

竹内 370 の 125 では，テープはちょっと違いますけれども，通信アダプタ，プリンタ，カード，ディスクといった I/O 機器は I/O プロセッサという標準化されたプロセッサがコントロールしています．

相磯 そうすると，それを仲介にして I/O インタフェイスの標準化ができる可能性があるんじゃないでしょうか．もちろん 100% マイクロプログラムで解釈して，適当に標準化できるとは思っていないんですけれども．

溝口 つまり，I/O 機器と制御装置の間のインタフェイスは，見かけ上まったく同じにしておき，マイクロプログラムで I/O 機器ごとに解釈するやり方ですね．技術的には問題ないと思いますし，将来，方向としては大いに可能性ありと思います．

相磯 かなりうまい道具になるのではないかと思っているんですがね．

マイクロプログラムの開放

萩原 ユーザ・マイクロプログラムのために，メーカがユーザに，まぁ大学や研究所はいいとして，一般のユーザにマイクロプログラムを開放できるかということについて，これはいかがでしょうか．

竹内 私は考えられないんじゃないかと思うんです．といいますのは，機械 1 台ごとにお客さんに合うようにカスタマライズしてお渡しし，私どもの方では，どのお客さんにはどういう組合せの，どの技術変更レベルのものが入っているかを全部おぼえているわけです．そのお客さんがフィーチャの変更，追加をされますと，そのためのマイクロプログラムなりハードウェアを出していかなければなりません．それで，お客さんが勝手にいじってしまったら，我々のコントロールができなくなってしまいます．そういう意味から，少なくともメイン・プロセッサについては，我々メーカとしては開放できないんじゃないかと思います．

ただ，370 には，独立した通信制御装置がつくことになっていますけれども，そのためのマイクロプログラムは 370 から送るようになっています．そのマイクロプログラムをハイレベルの言語でお客さんが書き，CPU の方でコンパイルして 370 から通信制御装置に入れてやると，お客さんの意図したカスタマライズされた機械になります．それをもっと広げていけば，今後かなりのインテリジェンス (intelligence) を持つと考えられる端末処理装置などでも，アプリケーションに合わせたインテリジェンスを持たせることができ，そのインテリジェンスに関するものはお客さんがマイクロプログラムできる．ただし，機械言語レベルでなく，ハイレベルでさわっていける．そういう形での開放はますます強くなっていくと思います．

相磯 一般のビジネス計算機のようにアーキテクチャがきちっと決まっていて，その上に大きな OS がのっているというような場合には，アーキテクチャの近くをいじれば，多分 OS にまで影響するわけです．だから，もしマイクロプログラムで何か機能を変更したり追加したりすることを決心したら，OS までいじらなければ，あるいは，自分で OS を作るような立場にたたないとダメですね．そうすると，ライタブルなマイクロプログラミングということは，当分の間は，小さな計算機でなければ向かない．逆に，ミニコンなど小さな計算機ですと，つまり，非常に単純な機能でアプリケーションの範囲が広いと，それにマッチするプログラムをつくるということで，とてもうまいんじゃないかと思いますね．自分でファームウェアを作り，OS をいじるという決心がいるんじゃないかという気がしますね．

マイクロプログラムの記述と効率

萩原 その場合のマイクロプログラムの記述ですけれども，ビット・パターンで書くというのは別としても，どのレベルで書くことになりますかね．

小高 アセンブラ・レベルじゃないですか．

萩原 ユーザが書くときもですか．

飯塚 ハイレベルで書ければいうことはないんですが，そういう言語から生成されるコードの効率が非常にいいとはちょっと考えにくいですね．

萩原 効率はそれほどよくなくてもいいんじゃないですか．というのはですね，FORTRAN が出てきたときの理屈を考えてみますと，コンパイラ言語で書いたプログラムなんか，効率が悪くて使いものにならないんじゃないかという議論があったんですが，現実には，こんな大きな仕事は FORTRAN がなければとても出来ないということになった．つまり，マイクロ

プログラムで相当のことをやらせようとしますと，アセンブラ・レベルでは大したことはできなくて，ハイレベルの言語で書かなければならない，そのために効率が落ちてもやむを得ない，そういうことになるんじゃないですか．それで，メーカの立場から言えば，ハイレベルで書かれたプログラムを効率よいコードにコンパイルすることを目指すことをやったらいいんじゃないでしょうか．

小高　難しい注文ですね．

萩原　難しいかも知れませんが，そういうことが必要ではないかと思いますね．

瀧口　マイクロプログラミングを普及させるためには，ハイレベル言語が使えなければなりませんね．

飯塚　たしかにハイレベル言語がなければ普及しないと思うんですけれども，FORTRAN の場合とマイクロプログラミングの場合は違うんじゃないかと思うんです．マイクロプログラムはできるだけ効率よく機械を制御して能率を上げるためのものですから，効率の悪いコードしか出来ないのであれば，何もマイクロプログラム語まで落さなくたって，普通の機械語に落して使えばいいという気がしますね．マイクロプログラムである以上，それで出来たコードは非常に効率のいいものでないといけません．とくに何回も使われるような場合には．

宇都宮　使い方にも関係しますし難しい問題ですね．ただ，マイクロプログラミングは，本来効率を最大の目的として導入されたわけですから，それが変らない限り，それに反するアプローチの仕方をするのはうまくありませんね．

萩原　その辺はマイクロプログラミングの本質をえぐるんで，もっと突込んで議論すると非常に面白いとは思うんですが，さて，どこからつついたらいいのでしょうか．

竹内　一挙にユーザ・マイクロプログラムにまで行ったんですが，メーカの中とか，OEM 同士とかでマイクロプログラムが公開され，組まれるようになることが先決だと思いますね．

宇都宮　その場合には，アセンブラでも構わないですね．

竹内　アセンブラですね．

相磯　結局，竹内さんのおっしゃることは，まずハードウェア設計者のレベルでオープンすることが大切で，そうしたら，アセンブラレベルでも我慢できるはずだということですね．

萩原　つまり，使い捨てのマイクロプログラムでないかぎり，やっぱり効率のいいものでなければダメなんですね．

相磯　そうですね．

萩原　使い捨てのマイクロプログラムが考えられるかどうかということですね．使い捨てのマイクロプログラムが考えられると，ハイレベル言語も考えられることになりますね．

飯塚　確かにそのとおりですね．

小高　例えば，オペレーティング・システムの一部をマイクロプログラム化しても，性能が改善されるかというと難しい問題ですね．客観的に考えますと，命令の呼び出し分だけいらないから，その時間だけ節約できるということになるはずですけれども．

萩原　そのあたり，私はこういう問題，つまり，今の機械語命令で考えるアルゴリズムとマイクロ命令で考えて処理するアルゴリズムとは違ってくると思うんですよ．いまおっしゃった命令の呼び出しの節約だけなら，先行制御をやったら同じじゃないですか．

ですから，多分マイクロプログラムに適したアルゴリズムが開発されなければならないんでしょう．例えば，sine, cosine を普通の機械コードで計算する場合は例の公式を利用するんですが，マイクロプログラムでやる場合には，別のアルゴリズムがあるでしょう．FFT などではこの方法が有効だと思います．これは一例ですけれども，このようにマイクロプログラムに適した処理をするアルゴリズムが開発されなければならない．そうすると，とても違ってくると思いますよ．

ファームウェア化

萩原　計算機システムはハードとソフトからなっていて，これまではハードウェアの方は本当にハードウェアで固まったものばかりで，いろんな機能はソフトウェアに背負わせてきた．だから，本当に大エス (OS) で小エスにならなかった．それから，いろんなものをソフトウェアに背負わせてきたために，ソフトウェアの開発が大変になって，ソフトウェアの危機というような議論がなされるようになった．ですから，ハードとソフトの接点であるファームウェアが生かされなければならない．それはどういうところで取り上げられていくんでしょうか．

竹内　今までは，とにかく CPU を出来るだけ効率よく使うということで，割込み処理や I/O 処理が複雑に行なわれ，OS が膨大になった．LSI の時代になっ

て，CPU を作るハードウェアが非常に安くなってくると，当然マルチプロセッサが出てきますね．そのマルチプロッサセの各々がマイクロプログラムを持つようになるということで，現在の OS がもっと簡単になっていくという考えも一部の人の間にはありますね．

宇都宮 OS の専門家は，OS はますます大きくなるだろうと言いますけれども，私は，OS をなくすというのは極論としても，少なくても小さくするという方向で努力しなくてはならないと思います．

相磯 小さくするということはファームウェア化するということですか．

宇都宮 そうです．その意味で，現在 OS をやられている人にソフト屋という意識を捨てていただいて，ハードに近づくというか，ファームウェアに積極的に接近するという方向でやっていただきたいと思います．

萩原 OS がソフトであるかどうかは非常に疑問ですね．OS は見方によっては普通の意味でのハードよりもっと固いところがあるわけですよね．

小高 OS の内容がはっきりしないとファームウェア化するのも難しいですね．

萩原 ある意味からは，OS の中の制御プログラムのアルゴリズムが確立されていないということでしょうね．

宇都宮 そういう面からの OS の解析というのは今後の課題ですね．

萩原 今の OS の制御プログラムの方はそれとして，言語プロセッサの方は少し様子が変わるんじゃないかという気がしますが，いかがでしょうか．たとえば，会話型の処理ですけれども，今の会話型プロセッサはインタプリティブにやっているのが多いですね．今のハードウェアでインタプリティブにやるのは非能率的なんで，ハイレベル言語のプロセッサをファームウェアで作れば，会話型処理をやる場合には，とても有効だと思いますね．

小高 竹内さんのところで APL を会話型で直接インプリメントなさった例がございますね．

宇都宮 たしか，360 の 25 といいましたね．

竹内 25 ですか．あれは制御記憶にコアを使っているんです．制御記憶も CE によりカードリーダからマイクロを入れることができ，簡単にマイクロプログラムを変更することができるんです．多分，大学の研究室か研究所でやられたものだと思います．

溝口 バローズの B-1700 では，ハイレベル言語を SDL と呼ばれる中間レベルの言語に変換して，それをマイクロプログラムで処理していますね．

宇都宮 今後は，そういうハイレベルの言語の専用機械がどんどん出てくるんじゃないでしょうか．

竹内 そういう方向になって行かざるを得ないんじゃないですか．

宇都宮 先程の OS と同じで，どのような機能をどのように分担させるかが非常に大きな問題になると思います．今のソフトウェアの危機なんかも，一つにはそういう検討がじゅうぶん為されてなくて，やみくもにソフトを開発して来たからじゃないんですか．

溝口 FORTAN のインデクシングとサブルーチンのみをマイクロプログラム化したら，15% のスピードアップになったという話を聞いたことがありますが，ファームウェア化というのは，ソフトウェアをマイクロプログラムでどこまでサポートするかということを，システム全体から考えて決定することが重要であると思いますね．今後の方向としては，ソフトウェアはますます巨大化し，開発コストも上がる一方であるのに対し，ハードウェアは半導体技術等の進歩によって価格低下の傾向にあるということを考え合わせると，意外に早く本格的なファームウェア・マシンが登場してくるかも知れませんね．

マイクロプログラミングの将来

溝口 言語プロセッサをファームウェア化する前に，言語プロセッサ向きの機械命令が出てくるんじゃないかと思いますね．

相磯 そうですね．将来の夢にもつながるんですけれども，マイクロプログラムの 1 つの大きな利点はハードウェア機能に可変性を与えていることなんですね．つまり，計算機をアダプティブにする，あるいは大げさに言えば，Learning Machine にするというところにつながっているんじゃないかという夢を持っています．ですから，今おっしゃったように，同じハードなんだけれども，いろいろな方向に適した命令体系をもってくる．しかも，それが環境によってダイナミックに変わるというようなことだと思うんです．アダプティブ・コンピュータだとか，ラーニング・マシンなんていうのが仮に実現できたとすれば，やっぱりマイクロプログラミングがエッセンシャルになっているんじゃないかと想像しているんですがね．

竹内 マイクロプログラミングでもって，オペレーションに最適なようにデータやフローまで変えること

を考えてもいいんですよね.

相磯　そうですね. さらに, 計算機システムとして, 使い方というか環境がいろいろ変わると, それに一番適したような姿に変わるというのがもっともいいんじゃないかと思います.

竹内　それもダイナミックに変わるわけですよね.

相磯　そうそう. それは夢ですけれどもね.

溝口　確かに夢は多いですね, マイクロプログラミングには.

宇都宮　一挙にそこまで行かなくても, まず, 人間と計算機が一緒になって学習して, 人間の経験の蓄積がより優れたマイクロプログラムに反映されるようにする. それから, ダイナミックに変わるということでしょうね.

竹内　アプリケーションに合ったシステムを自分自身で作り上げる. しかもダイナミックに変化できるものを. それはマイクロプログラミングを使わないと不可能だと思いますね.

相磯　そのためにはユーザがもっとしっかりしてくれないとダメなわけですね. ユーザがマイクロプログラミングを理解してくれなければダメでして, 現状では, マイクロプログラムはメーカのものになっているようなものですからね.

飯塚　圧力をかけないといけないですね.

萩原　というよりは, 若い人を教育するのに, マイクロプログラムを積極的に教えていかなければいけませんね. 先程相磯さんがおっしゃったように, ワイアード・ロジックもマイクロプログラムもなにも違いはないんだということから, ハードウェア・プロセッサも努力して自分で設計できる, そういう能力のある人がユーザになって行けば一番いいんじゃないかと思うんですがね.

宇都宮　そうなれば, プログラムといいますか, 現在のソフトウェアもハードウェアとイコールになってしまうわけですね. 現在, ユーザはソフト屋さんだということを自慢しているわけではないんでしょうが, そんな意識が強くてハードを知らなくても当然だという顔をしている. そこを直していただかないと困りますね. 大学側にも問題があるのかも知れませんが.

竹内　今までは, general purpose の計算機ということであったわけですけれども, 今後の計算機としては, お客さんごとに違った計算機を提供していくようにしなければいけないと思いますね.

小高　ですから, 同じ機械でも中味を変えたら新しいプロセッサになるといったようなことができるようになると実に楽しいと思いますね.

溝口　マルチパーソナリティですか.

宇都宮　そういうふうに言われると, 小さな計算機だけがそうなりそうに聞こえるんですけれども, 大型計算機も, よほど特殊な場合を除いて, そうなるんじゃないでしょうか.

竹内　お客さんのアプリケーションはみな違いますからね. アプリケーションに合ったものを製造して提供する. その結果, 生産者側としては, 計算機にフレキシビリティを持たしておいて, マイクロプログラミングでカスタマライズするということになりますね.

相磯　計算機システムの将来の開発として, 私は大きく分けて2つの方法が多分採られるだろうと思います. 1つは現在の大型計算機あるいはビジネス計算機をずっとこのまま伸ばして行く. もう1つは, 今おっしゃったような特殊なものの複合体のようなもの. どこかで一致するとすれば, その1つのきっかけはマイクロプログラミングだろうというわけです. 将来パッと一致すれば大変望ましいことですがね.

これからはマイクロプログラムの時代だと思っています.

萩原　話は尽きないようですが, 時間になったようです. どうもいろいろありがとうございました.

(昭和47年12月21日開催)