55-7

資料 9

第5世代の電子計算機に関する調査研究報告書

——計算機の適応・学習機能——

昭和55年5月

JIPDEC

財団法人 日本情報処理開発協会

この資料は，日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和55年度に実施の「第５世代の電子計算機に関する調査研究」の一環として作成したものであります。

目　　次

第１章　緒　　言

第1章　緒　　言

1.1 はじめに

本委託調査研究報告書は日本情報処理開発協会の依頼を受けて，昭和54年度に行った調査研究の成果の一部をまとめたものである。調査研究の課題は「計算機の適応および学習機能に関する調査研究」であり，第5世代計算機において必須な機能と考えられる適応ならびに学習機能を計算機アーキテクチャ・レベルで実現できる可能性とその効果について追求したものである。

計算機における適応性ならびに学習機能については，とらえ方によって様々な意味をもち，今まだ定かな定義が与えられていないのが実情である。このため，これらの機能に関する具体的な研究成果の発表も明解な解説もほとんどない。今年度の調査研究の範囲は次節において示すが，本報告書は主として慶応義塾大学工学部　相磯研究室において行なわれた基礎的な研究成果[1]～[18]をまとめたものである。

1.2 調査研究の範囲

本調査研究の目的は，与えられた問題に対して汎用計算機に比べ高い処理効率が得られる応用問題適応型計算機の構築法について調査し，考察することである。

従来のいわゆる汎用計算機では，特定の応用，変化する応用に高い処理効率が得られなくなっている。与えられた問題に対して計算機を適応させるためには少なくとも，システムレベルあるいはアーキテクチャレベルでの改築が必要である。ここで，アーキテクチャとは，G. M. Amdahlによると[19]“プログラマから見たシステムの属性，言いかえれば，データの流れ，制御法，論理設計，

物理的実現法などの機構とは分離した概念的構造や機能的動きのことである。"と定義されている。この定義は一般に狭義のアーキテクチャと言われるもので，この他にも，最近では計算機ハードウェアシステムの論理的構造全体を示す広義のコンピュータアーキテクチャというものも考えられている。ここではアーキテクチャを狭義の意味に限定することとすると，現在望まれているのは命令セットの構築レベルでの適応性ということとなる。命令セットの構築レベルとは，C. G. Bell らにより Instruction Set Processor レベル，略して ISP と呼ばれている。[20] そこで以下，命令セットはISP と書く。

従来の汎用計算機では，ISPレベルで幅広く問題に適応するということが追求され，平均的な処理効率が得られた反面，特定の応用や発展する応用に追従できず，その結果適応できなくなっていると言える。ところで，ISPで問題に適応させる方法はいくつか考えられるが，コストパフォーマンス的に最も良い方法は，ダイナミックマイクロプログラミング(dynamic microprogramming)の概念を利用することである。

しかし，ISPレベルで問題に計算機を適応させる場合，その道具である汎用マイクロプログラム制御計算機の構造のみならず，その使い方，特に動的(ダイナミック)にISP構造を問題に応じて変化させていく方法が明らかにされなければならない。

本調査研究の目的とするところは，与えられた問題に対して汎用計算機に比べ高い処理効率を得ることのできる応用問題適応型計算機の構築法にあるが，特に本調査の範囲をISPレベルに絞り，その中でも ISP が自由に変化する可能性を持つ汎用マイクロプログラム制御計算機でのダイナミックマイクロプログラムの使用法を明らかにすることである。

ここでは，アーキテクチャを狭義に考え，ISPレベルでの適応を考察するが，今後より広くアーキテクチャを考え広い意味でのシステムアーキテクチャの適応性を追求する場合も ISP は基礎となると考えられる。

第 2 章　問題適応性

第2章　問題適応性

2.1 適応化の方向

計算機の誕生以来の研究を振りかえると，その研究のほとんどは，狭義の適応性を満足する方向で進められてきた。例えば，実行速度という因子のみについて考えれば性能改善は著しく，誕生当時のものと現在の最高速のものとでは10^4倍以上の開きがある。また物量という因子のみを考えれば，半導体技術の進歩により誕生当時のものと現在の技術を比べると，やはり容量化で10^4倍以上小さくなっている。このように適応性の因子を細分化して独立に考察すると，確かに着実に進歩している。

しかし，計算機を現実に動作させ，何らかの応用問題解決のために使おうとすると，これら因子を独立に考えることでは不十分になってくる。すなわち，各因子はそれぞれ相互関係を持ちお互いに影響を及ぼし合う。したがって，これらの因子が複雑に作用しあった結果として，一つのまとまった計算機システムという形で「適応性」を考える必要が生じてくる。

そこでこのような細分化された因子の集合として一つの計算機システムというまとまった形での適応を「広義の適応性」と呼ぶ。これも正確に定義するのは難しいが因子の適応のバランスが平衡している状態，すなわち工学的にパフォーマンス／コストが優れていることと定義する。幅広い問題に対して広義の適応性を満足する計算機システムの実現が現在かなり困難になってきているが，その理由はいくつか考えられる。一つには，応用分野の拡大で問題によっては計算機誕生時には考えられていなかったようなもの，例えば人工知能，パターン認識の問題などがあり，現在の固定した ISP 概念にもとづく計算機構造で対処しきれなくなってきている面がある。またもう一つの重要な理由として，人間機械系としての果てしない改善の要求が考えられる。

以上のような理由によりこれからの計算機システムには，拡大する応用問題ならびに要求が変化する問題に対してアーキテクチャが柔軟に対処できるような，従来の商用計算機には見られない柔構造アーキテクチャ，問題，要求に応じて自由にその構造が変えられるアーキテクチャ，可変構造性のあるアーキテクチャが必要とされている。

2.2 適応化技術

本節では可変構造性をその可変レベルと可変適応技術に分けて考察し，ISPレベルでの適応の位置付けを行い，また他のレベルでの適応ならびに適応技術との関係も明らかにする。

図2.1は計算機システムにおいて，システムを問題に適応させるレベルをいくつかの段階に分けたもの（これを“適応レベル”という）と，可変適応技術との関係を表わしたものである。適応化レベルは上の段階に行く程，応用問題に近づき人間にとっては考えやすくなる。また，計算機の構造を変化させることのできる技術（これを“可変適応技術”と呼ぶ）には，ソフトウェア，ファームウェア，ハードウェアの3つに大別して考えることができる。

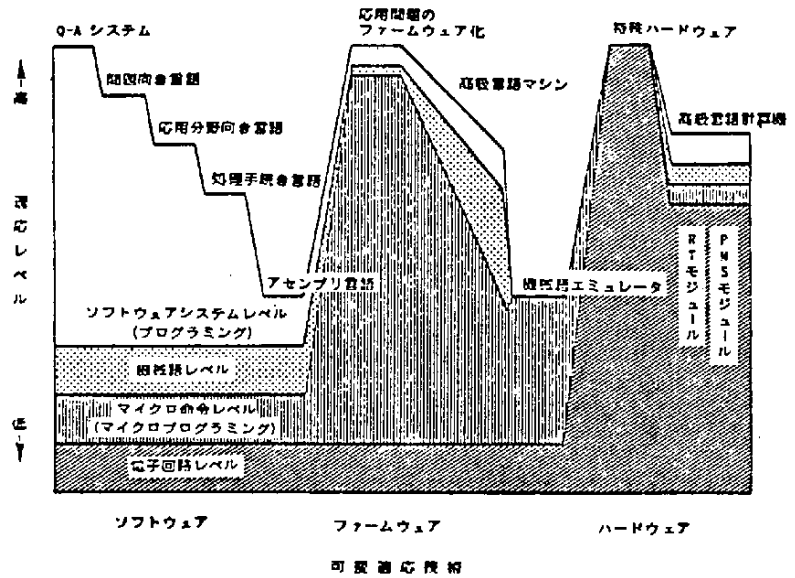


図2.1　計算機システムにおける適応レベルと適応技術

ここでソフトウェア技術では，機械語レベルは固定でその上のソフトウェア・システム・レベル（プログラミング・レベル）で可変で，問題に適応させることができる。また，ファームウェア技術では電子回路レベルは固定で，マイクロ命令レベル（マイクロプログラミング・レベル）で可変，ハードウェア技術では電子回路レベルで可変にさせることができる。これらの技術を使い適応レベルのそれぞれの段階に応じて計算機の構造を応用問題に適応させていくことが可能である。一般的には可変適応技術としてハードウェア技術を用いることは，実行効率の因子の適応度が上げられるが，コストの因子の適応度は下る。ソフトウェア技術はその逆であり，ファームウェア技術は中間に位置すると言える。また普通これらの技術は組み合せて使われる。

以下，それぞれにつき具体的な例を挙げ，どのように適応させるかを示す。

2.2.1 ソフトウェア

ソフトウェアによる可変性は命令セット（Instruction Set Processor 略してISP）レベルでは固定で，その上での可変性を得ることができる。可変性を利用して適応を持たせる例としてFORTRAN，COBOLなどの処理手続言語（Procedure-oriented languages）があり，更に，問題の記述のしやすさ（表現の仕方）という面では，適応度の高い応用分野向き言語（Application-oriented languages），例えばLISP（リスト処理向き），SNOBOL（記号処理向き），GPSS（シミュレーション向き）などがあり，また問題向き言語（Problem-oriented languages）として，RPG（レポート作成用），STRVDL（構造解析用）などがある。

このようにソフトウェアには，問題の表現方法についての適応度を上げることができる。しかも，適応を得るための可変性に対してのコストに関するオーバーヘッドは機構上ない。しかし，ソフトウェアで記述された問題は，コンパイラやアセンブラで固定された命令セットに変換され，実行されるわけであるから言語や記述されたものと，命令セットに著しく相違がある場合

は処理速度という点で適応しなくなる。すなわちISPレベルが固定であるとソフトウェアの適応性が生かせなくなる場合がある。ソフトウェアによって適応化できることはソフトウェアがもつ性格から当然のことといえ，ここではこれ以上の議論を行なわないが，もしISPレベルでの適応が成されれば，その特性はより適格に生かせるものと考えられる。

2.2.2 ハードウェア

ハードウェアによって計算機に適応を求めれば処理速度の因子では最も優れているが，ソフトウェアに見られたような可変性の自由度は少ない。

ハードウェアで応用問題に計算機の構造を適応させる場合，最も望ましいのは回路レベルから問題に適応したものを作り上げることであろう。しかし，これは一般に以下に述べる理由により非常に難しい。それは先ず第一に，応用問題と回路の間に概念上の開きがありすぎて対応がつけにくいことと，もしできたとしても適応範囲が著しく狭くなりコスト高になるからである。

そこで先ず考えられたことは経済的に採算のとれる方法，すなわち少しでも可変性をだし回路の標準的機能をインターフェースの統一されたモジュールとして用意しておき，それを用いて適応構造を構築する方法である。

モジュールの機能レベルならびに種類は，その時点での半導体技術，応用からの要求により決るが，技術的には低機能のものの方が作りやすいので，回路レベルより一段階上のレジスタとそれに関係するデータの処理機能を単位としたレジスタ転送レベル（Register Transfer Level 略してRTレベル）のような機能的には低いものから作られた。例えば，1967年にはWashington大学でマクロモジュールが開発され，[21] 1971年にはDEC社からRTMというモジュールセットが市販されている。[22] 当時の技術レベルからして当然のことながら，物理的な大きさや持続ケーブルの多さなどからこれを用いて応用問題に計算機構造を適応させた場合，ハードウェア・オーバーヘッドを大きくしてしまうという結果を招く場合もある。当時のデータによると，DEC社の

RTMを使い，特殊目的，例えばFFT専用機を作った場合，CDC 6600などの大型機と比べて同じくらいの速度が得られ，PDP−8のような計算機を作ったときは，コスト2倍，速度は40%程度であると報告されている。[23]

しかしながら最近の驚異的な半導体技術の進歩は，これらマクロモジュールやRTMに代わる新しいRTレベルの素子を生み出した。これは，一般にビットスライスのマイクロプロセッサと呼ばれるもので，1974年に発表されたIntel 3000シリーズに始まり，その後AMD 2900，Fairchildマクロロジックなどが発表されている。これらはほとんどがバイポーラ系技術を使っており，高速，小型で，しかもRTレベルでの汎用性を兼ね備え，従来のRTMやマイクロモジュールの欠点を克服している。

次にRTレベルモジュールより一段上の機能をもつPMS (Processor-Memory Switch) レベルモジュールについても簡単に触れておく。PMSとは計算機の要素をモジュールの一単位と考え，これを組み合わすことによって応用に適したシステムを作り上げるものである。Lockhead社のSUEプロセッサモジュールはミニコン程度の機能を持つ初めてのPMSモジュールで，実際にBBNによってARPA網の計算機間通信制御用システムに使われている。

また，カーネギーメロン大学のCM*[24]はDEC社のマイクロコンピュータLSI−11を中核に使ったPMSモジュールを組み合せた応用指向計算機であり，我国の電子技術総合研究所でもACE[25]というPMSレベルモジュールを使った応用指向計算機の研究が進められている。これら二つは特定の応用は考えずに，ある応用問題が与えられた時いかに適応した構造がとれるかを模索するための実験システムである。目下のところPMSレベルという性格上，各モジュール間の通信能力，通信方法における柔軟性，応用問題の並列構造性，応用問題からモジュール構造への変換手法ならびにソフトウェア開発法などを明らかにする必要があり，研究段階にとどまっている。

一方，最近の超LSI技術を背景としたカスタム・メイドICによる適応化に大きな期待がかけられているが，製造コストあるいは応用問題との写像

などに未解決な問題が残されている。

全般的に言えることであるがハードウェアによって適応性を得ると言っても，応用問題を先ず考えそれをLSI回路レベルにすぐ写像するのは難しいので，回路レベルの機能をRTレベルまで持ち上げ，このRTレベルのモジュールを使って応用問題に適応するような方法がとられてきたといえる。RTレベルでの適応性を得るということは，具体的には問題の性質を反映するようなISPを考え，それをRTレベルのモジュールを使って作り上げるということを行っているわけである。このような手法が広く行き渡るには，RTレベルのモジュールとしてどのようなものを用意するかが明られにされ，それを組み合せてシステムを作るための開発サポートシステムの完備を行うことが重要であると思われる。

またPMSレベルでの適応性を得るには，その性質上，問題の性質が並列性を考慮してシステムに写像できることが先ず重要である。並列性が検出でき各PMSに並列分散させる場合は，各PMSは1つ1つ独立して考えることができ，その中ではISPが重要となる。したがってその基礎となるのはやはりISPレベルでの適応性であると言える。

2.2.3 ファームウェア

前述したソフトウェアとハードウェアの中間に位置し，そのインターフェースの役割りを占めるのがファームウェア技術（またはマイクロプログラム技術といわれる）である。[26] 1960年代まではマイクロプログラムを格納する制御記憶が機能的に固定的なものが主流で，一度実現したマイクロプログラムを変えることは実質的に不可能に近かった。このような技術に支えられたマイクロプログラミングをスタティック・マイクロプラミングと呼んでいる。

しかしながら1970年初頭に高速のICメモリの発展を背景とした書き換え可能制御記憶（Writable Control Store：WCS）を備えたダイナミックマイクロプログラム可能な計算機が登場するに及び，これを境として可変構

造技術，すなわち問題適応技術としてのマイクロプログラムが重要視されるようになった。

マイクロプログラムによる可変性は基本的にはＩＳＰレベルにおいてである。しかしＩＳＰレベルといっても，考え方によっては大変広く考えることができる。いわゆる計算機初期の段階ではＩＳＰは，例えばIBM S/360の命令セットのような計算機の機械語と考えられていた。そこでマイクロプログラムを使い他の計算機の機械語をシミュレーションする技法が生まれた。これをエミュレーションと呼んでいる。しかしながら計算機の応用分野が広がるにつれ，機械語が唯一のものとは考えられなくなり，ISPとして考える範囲，機能，レベルも拡大解釈されるようになる。すなわちエミュレーションという言葉が，より広範囲に使われるようになった。FORTRANやCOBOLに向いたISPは，従来の機械語といわれるＩＳＰよりはるかに機能的にレベルが高く，――そのため，FORTRAN用ISPとは言わずFORTRAN用IML（Intermediate Language：中間言語）と言われる場合が多い――，すなわち応用問題により適応している。この考え方を押し進めるとさらに応用問題に近いレベルでのＩＳＰを設定できるわけで，これは応用問題のファームウェア化，またはユーザ・マイクロプログラミングなどと言われている。この技術は一応ソフトウェアと同等レベルまで可変適応度があると考えてよい。また，ISPとして実行時における適応性を追求するものを用意するだけでなく，デバックや保守のためのISPをも設定できる可能性があり，より広い範囲にわたり適応させることができる。マイクロプログラム化した場合の効果については，先ず実行時間の改善ということが考えられるが，表現のしやすさなどシステムの構造を問題に適応させた場合の利点に注目すべきである。

このようにファームウェア技術の特徴としては，ISPを問題に合せて可変にできることである。すなわちISPを問題に適応させることは，ハードウェアの技術を使ってでもできたが，この場合自由にその構成を変えないで変更

することはできなかった。ファームウェアでは変更に対しての自由度は大きく，WCSを使うとソフトウェア並みにその変更が可能となる。しかしながら，どのようにISPを問題に合わせていくのかを明らかにすることが最も重要な問題である。計算機の構造をISPレベルで問題に適応させる方法では，問題の環境をどのように検知し，ISPレベルで計算機の構造に写像するかが明確にされなければならない。

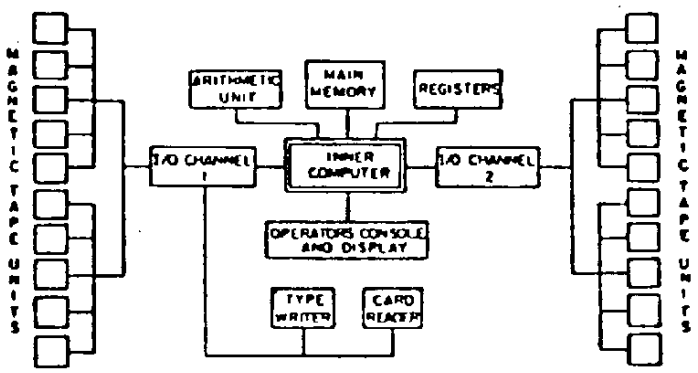
またファームウェアはソフトウェアとハードウェアの中間に存在するが，ファームウェアとソフトウェアとのインタフェースを考えるならば，ソフトウェアでは解くべき応用問題を最も記述しやすいように適応させているので，結局そのソフトウェアが最も効率よく動くようにファームウェアを使いサポートする。具体的には高級言語がコンパイラなどを使ってISPに落される時，落しやすいような構造にする。またファームウェアで実現されているISPにソフトウェアからパラメータなどの情報を受け渡す方法が明確にされなければならない。また，ハードウェアとのインタフェースを考えると，先ず多くのISPを効率よく記述できるようなマイクロプログラムを備えたマシンをどのように作るかということがあげられる。このような計算機は，汎用マイクロプログラム制御計算機と呼ばれるが，これにどのような機能を持たせるか，どのようなマイクロプログラムの形式にするかが重要となる。

第 3 章　可変構造計算機

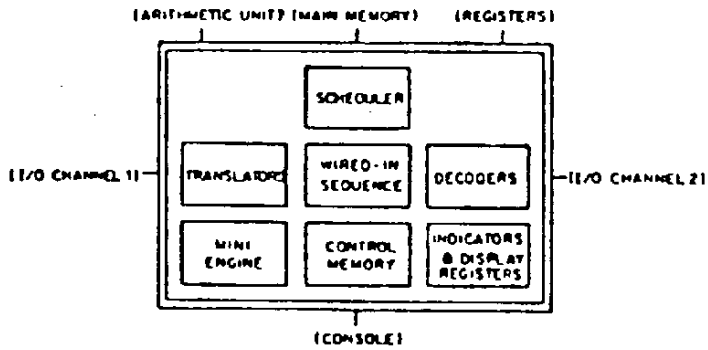
第３章　可変構造計算機

3.1　ダイナミック・マイクロプログラミングの概念

マイクロプログラムを動的（ダイナミック）に変更して問題に合わせたアーキテクチャを提供する方法の可能性は，既に1951年，M. V. Wilkes が述べているが，[27] その当時はそれをダイナミックに変更する必要性は述べていない。そのためマイクロプログラムの発展の初期においては，Wilkes の考えた通り，マイクロプログラムは制御論理に規則性が大きいことを利用し，設計を容易にするための道具として使われた。事実，IBMが1964年にIBM S/360で全面的にマイクロプログラムを用いたが，[28] マイクロプログラムは命令セットのエミュレーションのために用いられていたにすぎない。しかしこのあたりからマイクロプログラムを用いて IBM1401, 7000 シリーズなどのエミュレータが作られ，[29] マイクロプログラムの潜在的な能力が示されたり，マイクロプログラムに対する他の応用の可能性についての興味が高まり検討，提案が相次いで出されるようになる。高級言語計算機の実験がH. Weber によって EULERマシンのアイデアとして 1967 年に出されたり，[30] Opler は 1967 年にマイクロプログラムで実現される機能のことを「ファームウェア」（firmware）と呼び，マイクロプログラミング技術をこれまでの設計の道具という枠から脱脚し，ソフトウェアと同じような可変性の機能をマイクロプログラムで生かすべきであると指摘した。更に「適応性」の概念（Problem adaptability）は，Rakoczi が 1969年に，“Computer－within－Computer” という考え方を発表することにより明らかにした。Computer－within－Computer（内部計算機）の概念は，図3.1に示すような内部計算機と言われるマイクロプログラム制御計算機で各機能ユニットを制御し，いろいろな応用に合わせて内部計算機のマイクロプログラムを変えることによって適応システムを作ろうというものである。



TYPICAL FOURTH GENERATION SYSTEM CONFIGURATION



MAJOR COMPONENTS OF THE INNER COMPUTER

図 3. 1　内部計算機の概念図

1970年代に入り，Flynnらが「ダイナミック・マイクロプログラミング」という言葉を初めて使い，動的にマイクロプログラムを入れ換えるモデルを示した。

Reigelは，ダイナミック・マイクロプログラミングを用い，フィードバック・ループを扱った適応計算機の可能性を示した。この概念図を図3.2に示す。

これは，動的かつ自動的な適応性を持ったマシンの提案で，ダイナミック・マイクロプログラムの可能性を表わしている。また，Lesserは並列システムで，個々のプロセッサのみならず，各々のプロセッサ間の結合方式をもダイナミッ

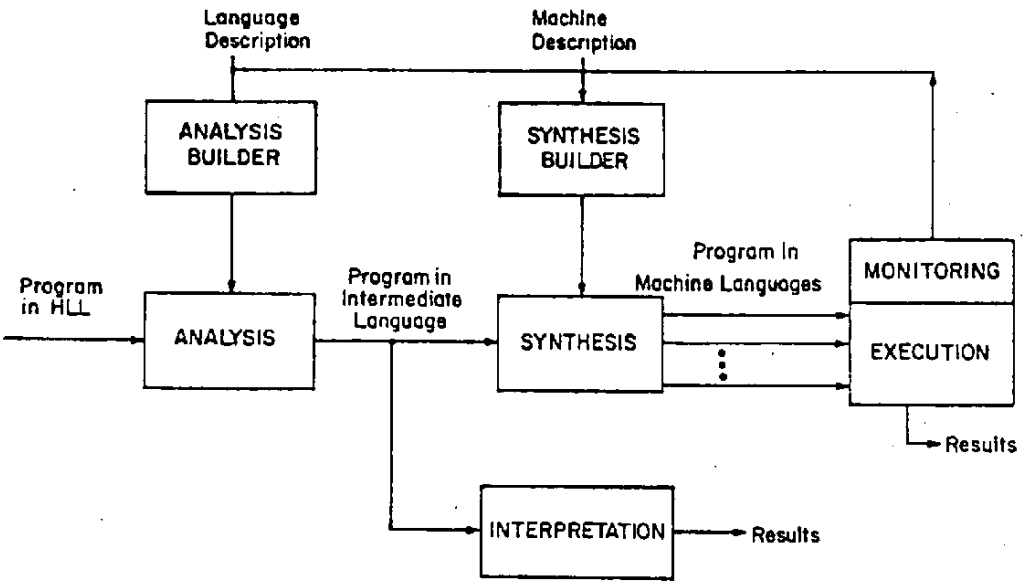


図 3. 2　適応計算機の概念図

クに問題に合わせて変化させていく方法を提案している。

以上述べたように，ダイナミック・マイクロプログラムの概念は 1970 年代に入り，応用問題に密着した効率よい計算機を求める要求と，半導体の進歩による実現の可能性という 2 つの背景をもとに次第に整理された。実用的な見地から始まった機械語のエミュレーションの考え方は言語のエミュレーション，応用問題そのもののエミュレーションへと拡大され，自動適応性，並列構造マシンなどの概念が登場した。

3. 2　ダイナミック・マイクロプログラミングの方法

3.2.1　制御記憶の入れ換え法

制御記憶の入れ換え法は以下に示す 2 つに分類できる。

(1)　オフライン的方法

オフライン的方法とは電気的に書き換え不可能ということであり，従

ってこの場合，マイクロプログラムを入れ換えるには人手あるいは機械的にということになる。初期のマイクロプログラム制御計算機ではこの方式をとるものが多く，例えばIBM S/360ではメタル・マスクROM，トランス形ROM，容量形ROMなどが使われた。これらはその性質上，ユーザ・レベルで自由に内容を書き換えることは無理であつた。またその後開発されたDSC Meta 4では渦電流方式ROMを使い，書き込みの自由度が増しておりユーザもオフラインでよければ任意のマイクロプログラムを書き込むことができた。しかし普通のコアメモリやICメモリなどに比べれば自由度はかなり犠牲になる。それにもかかわらず使われた理由は，低価格，読出しの高速性，情報の破壊に対する安定性などにあつた。

(2) オンライン的方法

オンライン的方法とは電気的に書き込み可能という意味であり，これはさらに2つの方法が考えられる。

第1の方法はマイクロプログラムを主記憶に記憶し，そこから取出し実行する方式をとる。このタイプのものにはバロース B 1700 などがある。またこの方式を発展させマイクロプログラムは主記憶に入れておき，さらにキャッシュメモリをつけ高速化をねらう方法をとったETLのACEなどがある。

第2の方法はマイクロプログラムを主記憶とは別のアドレス空間にあるWCSに置き，その書き換えには特別の命令を用意する。これはMLP-900やHP2100，HP 21MXなどのミニコンに見られる。これを更に発展させ2レベル化したものとしてQM−1などがある。

ここで，マイクロプログラムを主記憶とは別の空間に入れておくこと（また特に高速の記憶素子を使うこと）は一般のプログラムとは異なり，非常に頻繁に使われるという性質から考えて納得のいく方法である。

しかし，もしマイクロプログラムを多く書く必要が生じたり，多くの

応用問題向きエミュレータを用いる場合は，経済的理由，切り換えのオーバーヘッドを減らすという理由で，主記憶に置く方法が優れてくる。B 1726 の方法は主記憶の一部が高速メモリになっており，アドレッシングの方法はマイクロプログラム，ソフトウェア・プログラムで区別することはなく，アドレスの値（番地）で区別するだけである。この方法では重要なマイクロプログラムをどこに置くか，の管理はプログラマが行なわねばならないが，経済的で興味ある方法である。また，キャッシュメモリをつける方法はこれを自動的に行うものである。一般的にダイナミック・マイクロプログラムと呼ばれているものは，このオンライン的方法をさす。特に本調査研究で注目している応用問題に適応させるためには，このオンライン的方法が重要な役割を果す。

3.2.2 ISPの構成方法

ここでは適応ISPが何らかの方法で作られた場合，それをどのように構成し，動作（ISP間の遷移の方法：逐次制御法）をさせるかの方法につき整理しておく。

第1の方法は，ISPの構成制御もマイクロプログラムで行う方法である。例えばバローズ B 1700 では作られたISPの読出し，解読，実行，すべてマイクロプログラムで行っており，どのようなビットアドレッシングや強力なフィールド処理能力を採用するかが重要となる。

第2の方法は拡張命令方式をとることであり，ミニコンなどには多く見られる。この方法ではある特定の計算機のエミュレーションを主体に設計し，そこにユーザが作ったISPを，例えば特別のWCSへ制御を移す命令を用意することによって，拡張という形で付け足す方法である。

両者を比べた場合，後者はISPの形式（オペコードの幅，オペランドの数，長さなど）に制限がつく場合が多く，ISPレベルからマイクロプログラムレベルへのパラメータの転送がうまくいかない場合がある。しかし商業的に見た

場合，ハードウェアは前者に比べ後者の方が低価格に実現できる。

3.3 マイクロプログラム計算機の機能

マイクロプログラムによって実行速度が上がる理由としては，制御記憶の速度が主記憶のそれと比べ高速であるということが考えられるが，それよりも問題に合わせてハードウェアの細部の制御を行うことができ，ハードウェアを効率よく使えるということの方が重要である。確かに，機械語のエミュレーションなどではその速度差を利用して効率を上げている場合もある。しかし最近では半導体メモリの低価格化と高性能化が進み，制御記憶と主記憶の速度差はほとんどなくなってきており速度差だけでは効率向上を大幅に望むことは不可能になっている。特に適応化させる対象が高級化し，応用問題レベルで適応させる場合，ハードウェア細部の制御を行う機能が必要となる。

以下，マイクロプログラムでのみ制御できる機能と，マイクロプログラム計算機で使われる特長的な技法をいくつか述べる。

(1) マイクロ命令の形式

マイクロ命令の形式には水平型と呼ばれハードウェアに密着し，ゲートレベルでの制御を行うことができ，従って多くのフィールドを持つものと，垂直型と呼ばれデコード機能を強化し少ないフィールドで多くのハードウェアレベルのゲートを制御するものがある。

水平型の例としては，IBM S/360のマイクロコード（50ビット以上）などがあり，垂直型の例としてはバローズB 1700など（16ビット）がある。どちらの方式がよいかは一概には言えないが，一般に水平型は実行効率はよいが価格的には高くなる傾向にあり，ビットの使用効率を上げるのは難しいとされている。特に水平型の場合ハードウェアのゲートにかなり密着しており，高級な概念を記述するのは難しくなる。従って，機械語のエミュレーション，特に限定された機械語のエミュレーションを行い論理設計を簡単にし，

保守効率を上げるというようなWilkesの提唱した使い方の場合向いていると言えよう。また垂直型の場合は高度な概念のエミュレーションやダイナミック・マイクロプログラムを頻繁に使う応用では，概念の写像のしやすさ，転送のオーバーヘッドが少ないという点で向いている。しかしデコードの程度を上げるということは効率の低下をまねくので，逆にあまりに低級なもの，例えば通常の機械語のエミュレーションやコントローラとしてマイクロプログラム制御を利用するような場合に適さない。

この中間のものとして2レベル方式がある。例えばバロースのインタープリタ（B1700）やQM−1に採用されている。これは水平型，垂直型の両者の長所を必要に応じて使い分けるというものである。バロースインタプリタではマイクロ命令は16ビットで，特定のオペコードの場合，ナノ記憶が更にアクセスされナノプログラムが動く。この場合ナノ命令は54ビットであり，マイクロ命令もナノ命令によって作っていると見てもよい。

以上マイクロ命令の形式として重要なことは，いかに少ないビット幅で高度な効率よい処理を行うか，可変構造性を得るかということであるが，これらの要求は互いに矛盾するので難しくなる。2レベル方式は確かに一つの解決法であると思われるが，システムの複雑化と階層構造を強めることによる効率の低下などいくつかの問題点もあり，今後の検討に待つ点も多い。

(2) 環境の設定——レジデュアル制御——

ある応用問題をエミュレートすること，即ち機械から見た場合の適応させるべき環境が決まると，ハードウェアの環境をこれに合わせる必要が出てくる。ここでハードウェアの環境とはデータの長さ，ALUの構成，演算精度，ISPのデコード，アドレス計算などであるが，これらはエミュレーション対象が決まればそう度々変化するわけではない。そこでマイクロ命令からこのようなハードウェアの環境を決定する要素を抜き出し，初めにこれを特別なレジスタなどに設定すると後は変更されるまでその値が使え，マイクロ命令のセマンティクスがそのレジスタの値により動的に決定されるという方式が

考案された。これをレジデュアル制御（Residual Control）というが，B－1700，QM－1 など最近の汎用マイクロプログラム制御計算機では，可変構造性と適応性を与える機能として重要な役目を果している。B－1700 ではALUのレジディアル制御のためいくつかの機能があるが，その1つに演算精度を制御する機能がある。演算精度を制御するレジデュアル制御用レジスタがあり，ここに1～24の値がセットでき，ALUの演算精度はこのセットされた値をとる。このようなレジデュアル制御は，高度な適応性を得るために不可欠な機能である。

(3) マッピング

適応の対象となつているISPを実行するには，それをデコードして各ISPをエミュレートしているルーチンへマッピングし，それをマイクロプログラムで解釈実行しなければならない。またオペランド・フェッチを行うための実効アドレスの計算も行わねばならない。これらは一般に時間がかかり，しかも頻繁に使われるので，このための機能を備え効率を上げる工夫が考案された。MLP－900に見られる“language boad”は，デコードとオペランドフェッチを高速に行うための組合せ回路である。“language boad”は各エミュレータ毎に設計される。またB－1700ではEX命令があり，Mレジスタと呼ばれるレジスタに入れた値と，次に実行されるマイクロ命令の論理和をとつてから実行できる。従つて，マルチウェイ・ジャンプが可能でこれもデコードの高速化の機能の一つと見てよい。

(4) ビットアドレッシングとフィールド処理

現在の汎用計算機で扱えるデータは，多くの場合有限固定である。しかし問題によつて扱うデータ長が異なり，その取り扱い法も違うのは明らかである。従つて，高度な適応を得てしかもメモリ空間を有効利用するためには，データの最小単位ごとの取り扱いができることが必要となる。B－1700ではビットアドレッシング機能といい，1ビットごとにアドレスが付けられ，ビットごとの取り扱いが可能である。主記憶の任意の場所から任意のビット長

を一命令で取り出せる。またフィールド処理能力と呼び，任意のメモリフィールドを任意のメモリフィールドに移す機能も重要であり，強力なシフタ（マルチシフタ：一命令で任意桁のシフトが可能なもの）やマスク機能が必要とされる。

以上，いくつかのマイクロプログラム計算機特有の機能を簡単に述べた。この他にも適応を得るための機能として，マイクロ命令の並列動作，マイクロ命令レベルでの割り込み，サブルーチンスタックなど重要な機能は多い。これら多くの機能は，今後述べるような適応化の実験を積み重ねることにより，真に必要なものが明らかにされよう。

3.4 マイクロプログラマビリティ（Microprogrammability）

マイクロプログラマビリティとはマイクロプログラムのしやすさを表わす相対的な指標であり，マイクロプログラムを作る場合のトランスレータ，デバッガ，シミュレータなどのサポートソフトウェアの機能と制御記憶の内容を変えるハードウェアの機能に分けて考えることができる。結局，応用問題に応じた制御記憶が構成できるかということが重要なので，サポート・ソフトウェアが十分そろっていて（これをユーザ・マイクロプログラム可能という），高速に読み書きができるような制御記憶を持った機械がマイクロプログラマビリティが高いということになる。図3.3は実際作られた機械のマイクロプログラマビリティがどれくらいあるかを表わしている。

一般に HP，Varianなどのミニコンに共通していることは，そのほとんどが商用機であるため，ある特定のマシンのエミュレーションを想定して，マイクロプログラム制御部の設計がなされていることであって，応用問題に適応したISPはそれに追加していくという方法をとることにある。

研究用として作られるものはともかくとして，商用計算機において当初ミニコン程度のものからマイクロプログラムの利用をユーザ・レベルで許していく

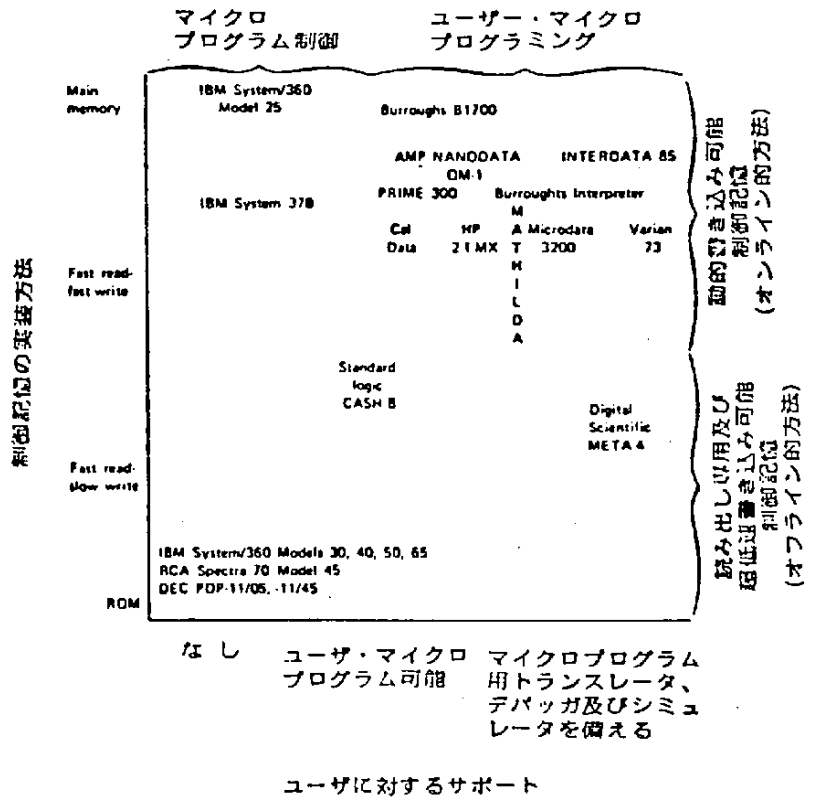


図 3.3　マイクロプログラマビリティ

という傾向が強かった理由としては，ミニコンはシステムに組み込むなどパーソナル・ユーズ指向が強いためシステム・ソフトウェアが簡単なものが多く，マイクロプログラムを動的に変更した場合の混乱が少ないと考えられたことに他ならない。

しかし今後研究が進んでも，製作される汎用マイクロプログラム制御計算機が大型化していくことは少ないと考えられる。高度に分散化されたシステムの中での適応化マシンという考え方，即ち各種の応用問題に対して１台で中央集

権的にこなすのではなく，それぞれの問題群に対して適応したマシンをいくつか持ち，それを結合することによりユーザにとっては1台のシステムとするような分散処理システムが今後の方向の一つであることは，半導体技術の動向から見ても明らかである。従って，今後作られる，あるいは研究されるべきマシンとしては，IC化しやすい分散処理システムのコンポーネントとしての汎用マイクロプログラム制御計算機がより重要となろう。

第 4 章　適応化の実験

第4章　適応化の実験

4.1 適応化の方法

計算機の構造（ここではISP）を解くべき問題に適応させる方法としての手順をまとめると，

1. 問題の性質を知る。
2. 問題の性質により，最適ISPの構造を決定する。
3. 制御記憶に決定されたISPを登録（ロード）し，適応化を終了させる。

のようになる。

それぞれの手順は次のとおりである。

4.1.1 適応化の手順

〔1〕 問題の性質を知る

問題の性質を知る最も望ましい方法は，解くべき問題のアルゴリズムを解析して知ることである。これをここでは直接法，または静的環境情報による方法と呼ぶが，現在計算機上で解かれる問題は多種多分野に渡り，複雑，巨大化しており，個々の部分的な性質は明らかになっても，問題を全体でとらえた場合の性質を明確に把握することは困難になっている。そこで問題の性質を知る方法として間接的な方法が考えられた。間接的に問題の性質を知る方法は，実行時における計算機の動作状態の偏りに注目するものである。Knuthらは，FORTRANプログラムの動作状態のデータを解析し，プログラムの実行時にはプログラムの4％以下の部分で，実行時間の50％以上が費やされているということを報告している。[41] またDarden, Hellerらも多くのプログラムを調べた結果，コードの3％の部分で実行時間の60％を費やすという事実を示している。

また，SoalはISPの動作状態についての偏りについて報している。[43]

このように，計算機の実行動作をもとにして間接的に問題の性質を知ることができるが，例えばプログラムの偏りが知れると動作時におけるプログラムの状態がわかる。これは，間接的にではあるが，解くべき問題においてどの部分が重要であるかなどがわかることで，問題の性質を知ったことになる。また，問題によつては動作させないとその性質が明らかにならないものもある。以上の方法を動的環境情報による方法と呼ぶ。

〔2〕 最適 ISP の構造の決定

問題の性質が知れると最適ISPの構造が決定できる。ISPの構造とは，ひとつにはISPの種類（どのような動作を行なうISPを用意するか）と，ISP間で変数受け渡しのためのオペランドの種類などの構成法を含むが，特に後者はエミュレーションされるユニバーサル・ホスト・マシンの構造に大きく依存するということに注意すべきである。

〔3〕 制御記憶への登録

静的（スタティック）な方法と動的（ダイナミック）な方法が考れられる。静的な方法とは，解くべき問題を実行させる前に 制御記憶に最適ISPを登録する方法であり，動的な方法とは，実行途中に入れ換える方法である。

〔4〕 方法の分類

問題の性質を知る方法と制御記憶への登録の方法には，それぞれ2つの方法があるので，これを組み合わせ，適応性の方法として合計4つの方法が考えられる。

すなわち，

1. 静的環境情報による静的な方法
2. 静的環境情報による動的な方法
3. 動的環境情報による静的な方法
4. 動的環境情報による動的な方法

である。

静的環境情報による静的な方法とは，アルゴリズムなどから問題の性質を知り，実行前に制御記憶を適応化しておく方法である。問題の性質が実行前にわかっていても，実行途中で制御記憶を入れ換えて適応化させた方がよい場合があり，これが静的環境情報による動的な方法である。また，動的環境情報による方法では，動的な動作を測定することによって適応化の方法を知るのであるが，これでも一度実行した結果をもとに次回からその結果から適応化させる方法と，実行中に制御記憶を入れ換え，途中から適応化させる方法がある。

4.1.2 チューニングと学習

動的環境情報により環境を知る方法では，計算機を一度動作させ，その状態を測定しその結果を解析することによって，問題の性質を知る方法である。従って，何らかの方法で一度ISPを設定し，これをもとにして改良することにより適応ISPを作り出す。そのためこのような方法はチューニング(tuning)と呼ばれる。

チューニングは人手で行ってもよいが，自動的に行う方法の開発が望まれる[10]。またチューニングにおける“学習”の概念は重要である。ここでの学習とは，基本的には普通使われている学習と差異はない。すなわち，一度経験した事を覚えておき，次回同じような状態に遭遇した時，その経験を生かすということである。これをチューニングにあてはめるなら，一度チューニングフェーズで発見された適応ISPを覚えておき，次回のチューニングをより効率よく行うことである。なお，このような機構を備えたマシンを“Adaptive Machine”とか“Learning Machine”と呼ぶと，本調査研究の目標は，これら“Adaptive Machine”，“Larning Machine”構築法の基礎を調べることであると言える。

4.2 静的環境情報による静的適応方法

静的環境情報による静的適応方法とは，問題のアルゴリズム，仕様などからその問題の解かれるハードウェア環境を知り，問題が解かれる前に環境を適応させる。即ちマイクロプログラムによってハードウェア環境を問題に適応させる方法である。ここでは具体的な例として，機械語のエミュレーション，高級言語計算機，応用問題のマイクロプログラム化を取りあげ，考察する。

4.2.1 機械語エミュレーション

機械語エミュレーションとは，ある特定の計算機の機械語をマイクロプログラムでシミュレーションすることである。エミュレーションの手順を図4.1に示す。エミュレーションする対象は機械語であり，その動作仕様は明確で，適応させる際の目標も速度をパラメータとして考えればよい。また適応はエミュレーション動作前に行えばよく，静的方法ということになる。しかしこのように，問題の性質がすべて明らかであり，目標もはっきりしていても，実際適応させるのは常に容易であるとは限らない。

エミュレーションにおける問題点は，

(i) ハードウェア・リソースの対応（写像）

(ii) セマンティックスの解釈

(iii) デコード・プロセス

(iv) 入　出　力

にあると考えられる。ここでハードウェアのリソースの対応とは，被エミュレーション・マシンとエミュレーション・マシンが持つハードウェア資源の対応がどのようにとられるのかである。例えば，もし被エミュレーション・マシンよりもエミュレーション・マシンの方がハードウェア資源をより多くもてば，効率のよいエミュレーションが期待できる。

(ii)のセマンティクスの問題については，図4.1における各ルーチンの意味

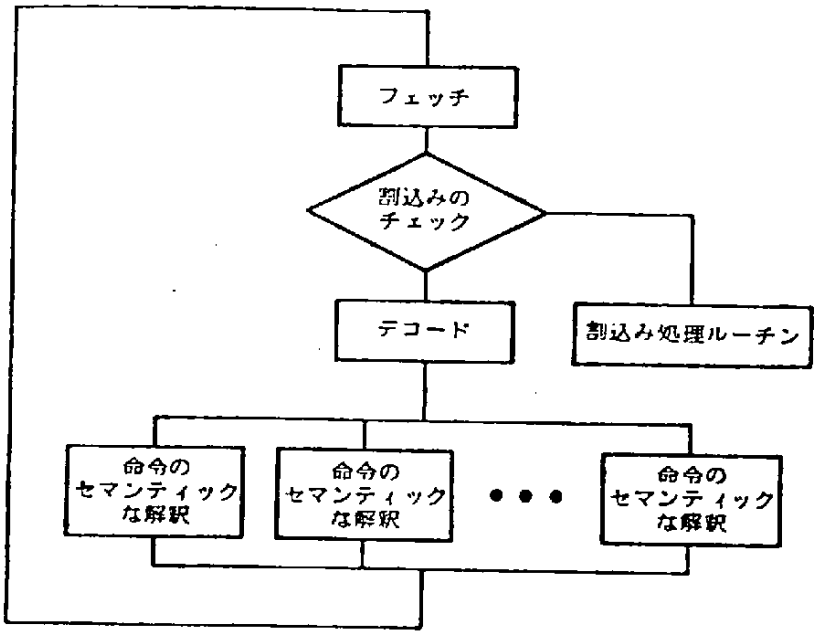


図 4.1　エミュレーションの手順

を解釈する場合に起る問題である。この場合，マイクロ命令の持つ能力がどれ程であるかが重要になる。被エミュレーション・マシンの持つ機械語とエミュレートするマシンが持つ構造ならびにマイクロ命令のレベルが似ている場合はよいが，被エミュレーション・マシンよりマイクロ命令レベルが低すぎると効率低下が生じる。

(iii)のデコード・プロセスについての問題は，フェッチ・デコード・ルーチンが毎回走ることを考えると，効率を向上させるためには，マッピングのための特別な工夫——例えばランゲジ・ボードなどの用意——が望まれる。

最後の入出力の問題については，先ずタイミングを考慮する以外に，割り込み制御などがあり，これらを完全に効率よくエミュレーションするのは一般に難しい。

エミュレータを実用にする場合は入出力はもちろんのこと，オペレーティング・システム（以下OSと略称する）の構造も一部エミュレーションする

必要がある。汎用マイクロプログラム制御計算機の素子にもよるが，10倍以下とするのは現在難しいと思われる。

4.2.2 高級言語計算機

高級言語計算機（又は高級言語マシン）とは，高級言語で書かれたプログラムを効率よく実行する機構を備えた計算機である。[44] 従って，その実現の方法はいろいろ考えられるが，高級言語計算機においてもマイクロプログラミングの概念が重要な働きをする。

高級言語をソフトウェアで処理する場合の方法は，一般に大きく分けてインタープリタ方式とコンパイラ方式の２つの方法が考えられている。ここでインタープリタ方式とは逐次処理方式であって，高級言語を１ステートメントごと解釈実行していく方法であり，これをマイクロプログラムで処理するとはこのインタープリタをすべてマイクロプログラム化することである。確かにこのような方法をとってもソフトウェアと比べ約２～10倍程度の効率向上が望める。しかし，マイクロプログラムを使い高級言語計算機を実現する場合でも，コンパイラ方式をとった方がインタープリタ方式をとるより効率向上が望めるのは，ソフトウェアの場合と同様である。コンパイラ方式の場合，ソフトウェアのみの時と大きく異なる点は，コンパイラによって変換されるコードが，ソフトウェアの場合は一律同じコードであるのに比べ，マイクロプログラム・マシンでは，高級言語の性質を反映したコードにそれぞれ変換し，それをマイクロプログラムで解釈実行するという方式をとることである。

方式としては，機械語に代わって高級言語の性質を反映したコード（これを中間言語 Inter-Mediate Language，ＩＭＬという）をエミュレーションすると考えればよいので，前節で述べた機械語エミュレーションと似ている。しかしながら，高級言語のIMLは機械語に比べると制御構造とデータ構造がかなり複雑になっている。例えば，制御構造では if then else や while 等

のループ，begin—endによるブロック構造，リカーシブ・コール，コルーチン等，データ構造ではリスト，ベクトルなどがある。従って，機械語エミュレーションの場合よりもエミュレーション対象の中間言語のレベルがかなり問題に近い。

従来の汎用計算機でソフトウェアで実現する場合は，これを機械語のマクロのようなもので処理しているが，機械によっては，これ（機械語）がさらにマイクロプログラムで処理される。それゆえ直接マイクロプログラムで処理することによつて，階層構造が減り効率向上が望める。従って，処理する高級言語としては，概念的に高級なものの方が効率向上の度合は大きい。FORTRAN，COBOLなどよりAPL，LISPといった言語の方が効率は大きい。例えば，Hassitt の行なった実験によると，[45] APLマシンでA←B＋C（ここでB，Cを長さnのベクトルとする。）を実行した場合

APLマシン …… 785 + 320n （マイクロ秒）

IBM 360/25 …… 41 + 387n （マイクロ秒）

と10倍以上，その他HP 2100でLISPマシンを実験的に作ったところ，[46] 3～7倍くらいの効率向上がなされた。これに比べFORTRANなどの実験は少ない。その理由は，FORTRANは言語レベルが現在の汎用計算機の機械語を考慮して作られているため，高級言語マシン化をしてもあまり効率向上が望めない点にあると思われる。

効率以外の点では一般的にコンパイラが高級言語マシンでは簡単となる。例えば，IBM360/30のマイクロプログラムで実現されたEulerマシンでは，機械語のそれに比べメモリ要求量は1/6程度であると報告されている。[30] またオブジェクト量も減少する。

IMLが高級であることは，デバックに与える影響も大きく，デバックのためのデータがオーバーヘッド少なく取れ，虫取りが容易となる。また汎用マイクロプログラム制御計算機との関係では，より高級言語の性質を反映した機能の強化が望まれる。例えば，高級言語では多くのデータ構造を取り扱わねば

ならないが，そのような場合“タグ”の概念が有用となる。しかしタグを効率よく処理するには，マイクロ命令にタグを考慮したものがあることが望ましい。

最後に商用機に与える影響について簡単に触れておく。先ず中型以上の計算機では複数個の言語が走らなければならない。この場合，ダイナミック・マイクロプログラミングの概念が有用である。例えば，バロース B1700 は2つ以上の高級言語マシンを1つのハードウェア上にもつ数少ない商用機である。この場合，マイクロプログラムを言語に合わせ動的に入れ換える機能を持つたオペレーティング・システムなどが用意されている。

また，小型機以下では今後パーソナル化が進むと思われ，この場合1つの言語しか走らなくてもよい。またこの小型計算機を使いネットワーク化を押し進めた分散処理システムについては，第3章でも述べたように今後中型機以上のシステムに与える影響も大きく，マイクロプログラムによつて専用化されたマシンがいくつか組み込まれることになろう。

4.2.3 応用問題のファームウェア化

前節までに述べてきたエミュレーションの概念をさらに押し進めると，最後にたどりつく方法は，解くべき問題をすべてマイクロプログラムで構成するということになる。いくつかの実験によると処理効率は機械語で書いた場合に比べ，約3～10倍以上の効率向上が期待できる。[47][48]

4.3 静的環境情報による動的方法

本節では静的環境情報のもとで動的にアーキテクチャに帰還をかけ，計算機アーキテクチャを解くべき問題に適応させる方法について簡単に述べている。具体的な適応例としては，ソフトウェア開発における最大の困難事であるプログラムのデバッギングに適応する機能を，ダイナミック・マイクロプログラミ

ング技術を駆使して実現した例をとる。[7][8]

プログラミングのデバッキングに関しては以前から種々の方法が提案され，有益なシステムも開発されているが，[50]~[52]一般に最もやっかいな虫（バッグ）はプログラムの実行段階で発見されるものである。このような実行段階でのデバッギングを能率よく行うために，今までも高機能デバッガの開発研究が数多く行われている。当然のことながら，デバッガの高機能化はシステムの実行効率の低下をもたらし，実用的な見地から限界といわれている。[52][53]

本節では，デバッガの高機能化と実行効率の改善という難問題をダイナミック・マイクロプログラミング技術を活用し，計算機アーキテクチャ・レベルで解決することを考え，実際にYHP 21−MX 上に高機能デバッギング・システム" SNOOPY"を実現し，その有用性を実証した例をとりあげる。

なおここで，静的環境情報によるとは，デバッグのための動作指示がなされると，どのように適応化させるかがわかるからであり，デバッグ動作中に動的に制御記憶を書き換えて適応させるので，動的な方法ということになる。

4.3.1 高機能デバッギング・システムの背景

〔1〕 デバッギングシステム

信頼性が高く効率のよいプログラムを作成することは，計算機誕生以来大きな課題の1つとなっている。従来このような目的のために行われてきた研究としては，大別して2つの方法がある。1つは合成法（Synthesis）にもとづくもので，他方は確認法（Verification）にもとづくものである。合成法にもとづくものはプログラム作成時に誤ったプログラムは走らないという考えによるものである。確認法にもとづくものはプログラム作成時に完全なプログラムをつくることはできず，必ず誤りはあるという前提にもとづき誤りをいかに速く，完全に除去するかという考え方でプログラムを完成させるものである。現在とられている方法の多くは確認法によるものである。

ところで確認法の代表的な手法に，テストとデバッグがある。[54] プログラムは先ずテストによりプログラムの誤りの有無を見つけ，デバッグによりプログラムの誤りを見つける。テスト，デバッグとも机上で行うことは可能である。しかし，両者とも計算機にその動作の一部あるいは全部を行わせることは可能であり，このようなものをデバッギング・システムとよぶ。

〔2〕 機能的要求

(1) 事象モニタ

従来のデバッガの多くは，ユーザが指定するロケーションまで被デバッグ・プログラムを実行させ，探索，変更を行うという，いわゆるブレーク・ポイント方式のものが多く，“虫”発見までに多くの試行錯誤を必要とした。

これに対してPL/1の“ONステートメント”に見られる事象モニタの機能は，ある事象を起す場所を一度で発見でき，デバッギングにおいて非常に有用である。このような事象モニタ（以下，モニタ・アクションと呼ぶ）では，そのモニタ結果に対する動作（以下，アクションと呼ぶ）を指定することが望ましく，適宜組合せれば“虫”発見が容易になる。

(2) デバッグ向き機能としてのバックトラック機能

従来のプログラムのテストとデバッグの過程を考察すると，テストによって誤りの存在を判断し，デバッグによって誤りを引き起した“虫”のありかを調査し取除くというものであり，“虫”の発見にはプログラムの実行状態を把握できるように（例えば，テスト文を入れる），プログラムを変更しながら何度もプログラムの再試行を行わなければならない。このプログラムの再試行は，プログラムの大きい場合にはオーバーヘッドとして多くの時間的な損失をまねいた。そこで，プログラムの再試行を行わずにプログラムの状態を任意の時点に忠実に逆にもどる機能（バックトラック機能という）は，このような時間的なオーバーヘッドがなく，デバッギングにおいて非常に有力な道具となる。

(3) 被デバッグプログラムの完全把握

被デバッグプログラムの状態を完全に且つ正確（デバッギングシステムの影響を受けず）に，把握できる機能が重要である。入出力関係，特に割込みなどの事象把握は，システムプログラムの開発に有用である。

〔3〕 システム的要求

高いレベルでのマンマシンインタフェースを保証することが重要である。そのためには会話形式であること，高いレベルのデバッグ用コマンド，又は言語を備えること，更に機能的拡張が容易に可能なことなどが望まれる。

4.3.2 デバッギング適応形計算機の実現例

〔1〕 従来の方法

前節で述べたように高機能デバッガの核となるのは事象モニタであり，従来はソフトウェア的な手段で実現する場合が多かった[49][50]。しかし，この方法では本質的にはシミュレーション技法となるのでデバッギング時間が非常に大きく，入出力動作などの正確なシミュレーションは困難である。ハードウェアモニタを使う方法では，モニタ時間はオンラインで行えるが，デバッギングで重要なモニタ機能の融通性とシステムの自由度を保証するには，コストの面でオーバヘッドが大きくなる欠点がある。ファームウェアによる方法では，前2者の欠点を補うことが可能であるが[4]，単に情報をログアウトするだけのスーパートレーサでは無駄が多く，デバッギングに適応しているとはいえない。

〔2〕 ダイナミック・マイクロプログラミングによる適応化

YHP 21－MX 上に実現した"SNOOPY"を例にとり，適応化の方法とその評価を具体的に示す[7][8]。SNOOPY開発に際して考慮した点は次のとおりである。

<1> 設計方針

(1) デバッギングを会話型に行うことを前提とし，不要なトレース情報

をなるべくとらないようにする。

(2) 事象は個々の命令の動作結果によるものであって，事象ごとにモニタすべき内容を動的に決める。具体的にはダイナミックマイクロプログラミングを利用し，個々のデバッグコマンドごとにマシンの状態を適応させる手法をとる。

(3) デバッグの対象とするプログラム形式は，オブジェクトプログラムとする。対象とするものをソースプログラムとするかオブジェクトプログラムとするかは，それぞれ一長一短があるが，特に後者にはコンパイラやアセンブラ自身の虫を取り得るという柔軟性があるので，ここではオブジェクトプログラムを対象とした。又，ユーザとしては，アセンブリ言語使用者（システム・プログラム開発者）を考えているが，高級言語使用者にも適用できるような配慮も加える。

<2> 機能およびコマンド言語

(1) コマンド形式

SNOOPY のコマンドは表4.1のモニタ動作を伴う if then else形と do case 形ならびにアクション，特殊コマンド及び特殊キーを単独に用いる形式からなる。

(i) if then else形コマンド

SNOOPYコマンドの基本形であり，モニタの対象とする事象とモニタ結果に応じて2通りのアクションを指定できる。又 then 節，else 節には任意レベルの if then else 形コマンドを記述でき，連続モニタが可能である。

(ii) do case 形コマンド

if then else 形コマンドの拡張形であり，複数の事象の同時モニタが可能である。

表 4.1　SNOOPYのコマンド形式

	if then else 形	do case 形
format	if MONITOR ACTION,n,{S/P} then ACTION begin (if then else COMMAND) end else ACTION begin (if then else COMMAND) end	do case : (expression),n,{S/P} (relational operator) (expression) ACTION (relational operator) (expression) ACTION (relational operator) (expression) ACTION (relational operator) (expression) ACTION
comment	・n：事象の回数 ・S：モニタ範囲 (ステップ数) ・P：モニタ範囲 (ラベル＋変位) ・多数の if then else を記述可能 ・then，又は else 節を省略した場合、ACTION に "PAUSE" が割当てられる。	・n：事象の回数 ・S：モニタ範囲 (ステップ数) ・P：モニタ範囲 (ラベル＋変位) ・マルチモニタ用の形式で、現在 4 レベルまでの事象を同時にモニタ可能

(2) 主 な 機 能

(i) モニタ動作

モニタ・アクションとして指定可能なモニタ動作は，実行される命令のオペコード，オペランドをモニタするための Appearance 形と，変数およびハードウェアレジスタの内容の状態変化をモニタするための Expression 形からなる。

(ii) バックトラック機能

バックトラック準備用の履歴保持用コマンドと実際にバックトラック動作を起動させるコマンドが準備されている。

(iii) エディット機能

一般的な探索・変更機能のほかにデバッグ向きのオペレーションサーチ機能を備えた。これはコマンドで起動され，指定される変数に関連するオペレーションの履歴情報を知らせるものである。

(iv) コマンドマクロ機能

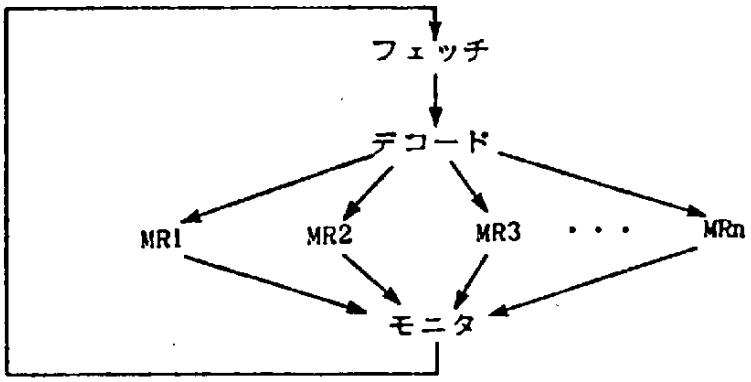
システムレベルでコマンド言語の拡張性を考慮すると，実行効率が著しく悪くなるためコマンドマクロで擬似的な拡張性を与える。

そのほかにデバッグ・セッション中に常に働かせておきたい機能を宣言するコマンドや，変数の内容の変化をグラフで表わすグラフ機能などがある。

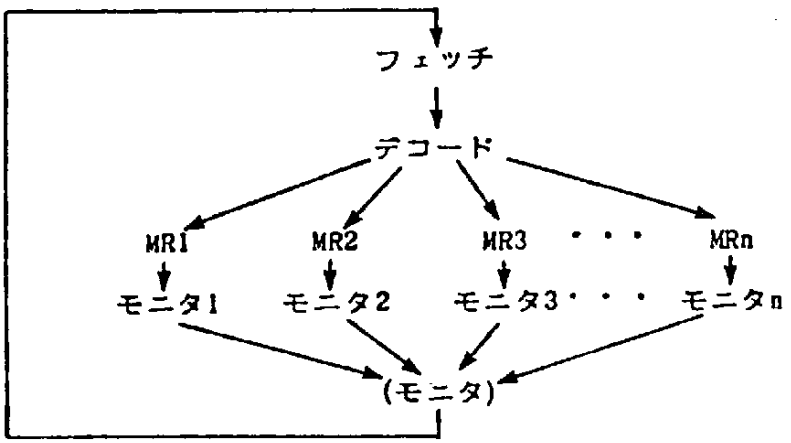
<3> 適応化の方法

(1) 考え方

ダイナミックマイクロプログラムの特質を生かしたモニタの構成法の基本的な考え方は，中間言語（IML）のエミュレータの中にデバッギングのためのマイクロプログラムをモニタコマンドに適応させて挿入することである。図4.2(a)に示すのは従来のファームウェア・モニタの方法であるが，この方法ではモニタは一ケ所で行なわれている。



(a) モニタの方法1



(b) モニタの方法2

図 4.2 モニタの方法

ところが，ある目的のためのモニタはすべてのIMLに関係するとは限らない。そこで，ここではこのような事実に注目して図 4.2(b)に示すような方法を得る事とする。これはIMLレベルでのモニタはそれぞれの命令に従属するので，モニタ動作はIMLの中に組み込むという思想に基づいている。

次にデバッギング動作を見ると，モニタはモニタ指示により一定時間にモニタされ，その結果を解析する事から別のモニタ指示がされるというように，逐次的に異なるモニタが行われている。可能となるモニタ動作は一度に行われない。すなわちモニタには"局所性"がある。

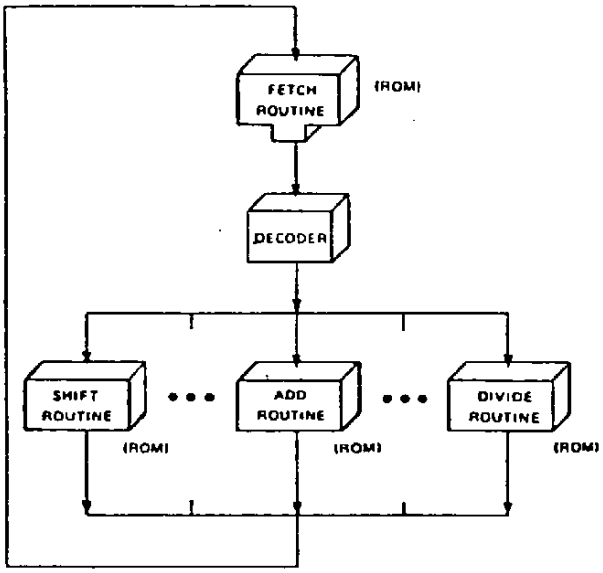
従ってモニタ部分にはモニタ指示に従い，それぞれ構成し直す。例えば，ある時間まではフリップフロップのモニタが必要であっても，それ以後は必要なくなった場合，もはやそのモニタルーチンは無用である。以上の事をまとめると次のようになる。

- モニタはIMLに依存するので集中モニタはモニタ効率を上げるためには望ましい方法ではない。
- モニタ動作はモニタ指示によって決定されそれはその都度異なる。

ここではこの考え方をモニタシステムに反映させた結果，図 4.3の方法をとった。モニタ部分はIMLの中に組み込まれ（制御記憶の中に同居さす），モニタ指示の都度，この部分を構成しなおす。これはダイナミックマイクロプログラムにより行われ，モニタ指示ごとに，モニタシステムの最適化を試み効率を上げることが可能となる。
（図 4.3(a)～(c)を参照）

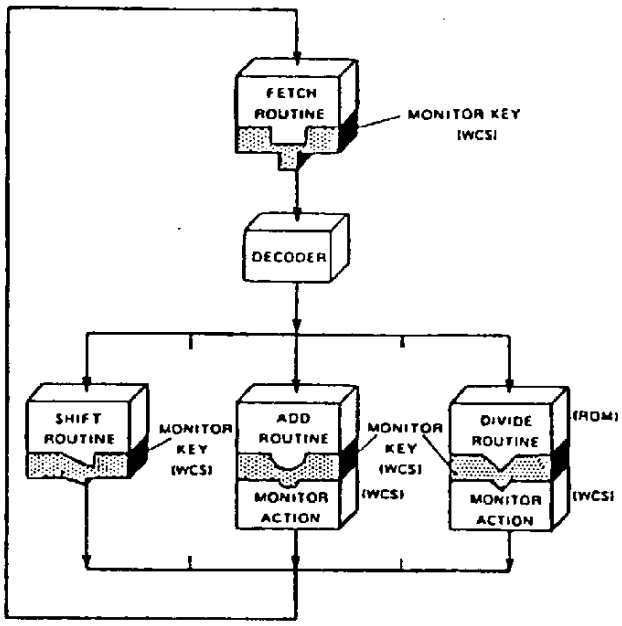
(2) YHP 21－MX での適応化法

SNOOPYのデバッグ機能を計算機のシステム・アーキテクチャ・レベルで実現し，計算機を問題に適応させる方法を，YHP 21－MXでの例に沿い，以下に述べる。



ノーマルモード　(SW ： オフ)

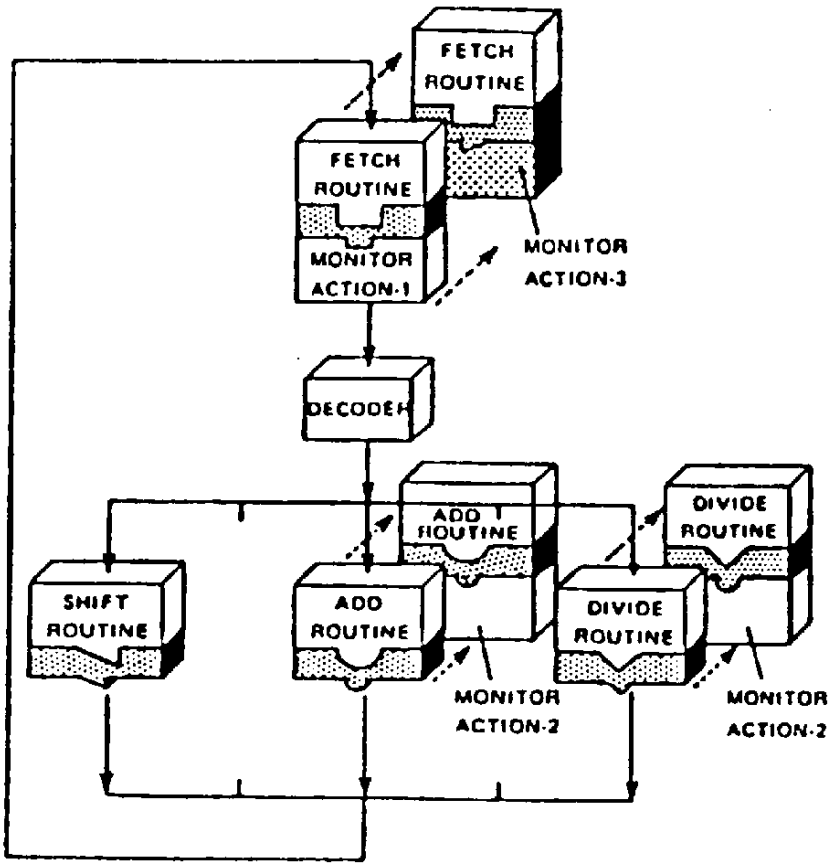
(a) ファームウェア・モニタ



モニタモード　(SW ： オン)

(b) ファームウェア・モニタ

図 4.3　ダイナミック・マイクロプログラムを用いたモニタの概念図



(c) WCS の動的再構成

図 4.3　ダイナミック・マイクロプログラムを用いたモニタの概念図（続　き）

(a) YHP21−MX における命令実行に伴う制御記憶の動き

YHP21−MXは，制御記憶に書換え可能メモリ（WCS）を備えたユーザ・マイクロプログラマブルなミニコンピュータであり，その機械命令（1語＝16ビット）は，YHP2100の命令と互換性のある基本命令セットと，拡張命令および浮動小数点命令からなる。

制御記憶（1語＝24ビット）は16モジュール（1モジュール＝256語）から構成されており，モジュール0にフェッチルーチンと基本命令用マイクロルーチン，モジュール14，15に浮動小数点命令，拡張命令用マイクロルーチンがそれぞれ含まれている。

命令の解釈実行に伴う制御記憶のコントロールの動きを図4.4に示す。フェッチ・ルーチンの最後のJTABというのは，命令デコード用の特殊なマイクロ・オーダーで，命令レジスタ（IR）のビット15〜18の内容（オペコード）に従って，それぞれの命令実行マイクロ・ルーチンへ分岐させる機能をもっている。すなわち，命令デコードの部分はマイクロプログラマに開放されておらず，ハードウェア的な手段で行っている。

ユーザ・マイクロプログラムの実行は，ある限定された（JTABによってMTABLを介して所定のアドレスに分岐できるような）ビットパターンの機械を設定することによって，通常の機械語の解釈実行と同様に処理される。図4.5にその様子を示す。

(b) システムの状態遷移

SNOOPYには8種のモードがあり，その関係を図4.6に示す。又，使用するスイッチ及びテーブル類を表4.2に示す。コマンドはシステムが待ちモードのとき受付けられ，コマンドインタプリタがモニタ機能，バックトラック機能，エディット機能，出力機能の中のどれを要求しているか判断し，コントロールを対応するモードへ移す。

BASE SET INSTRUCTION FETCH SEQUENCE

(ARROWS INDICATE JUMP DESTINATIONS)

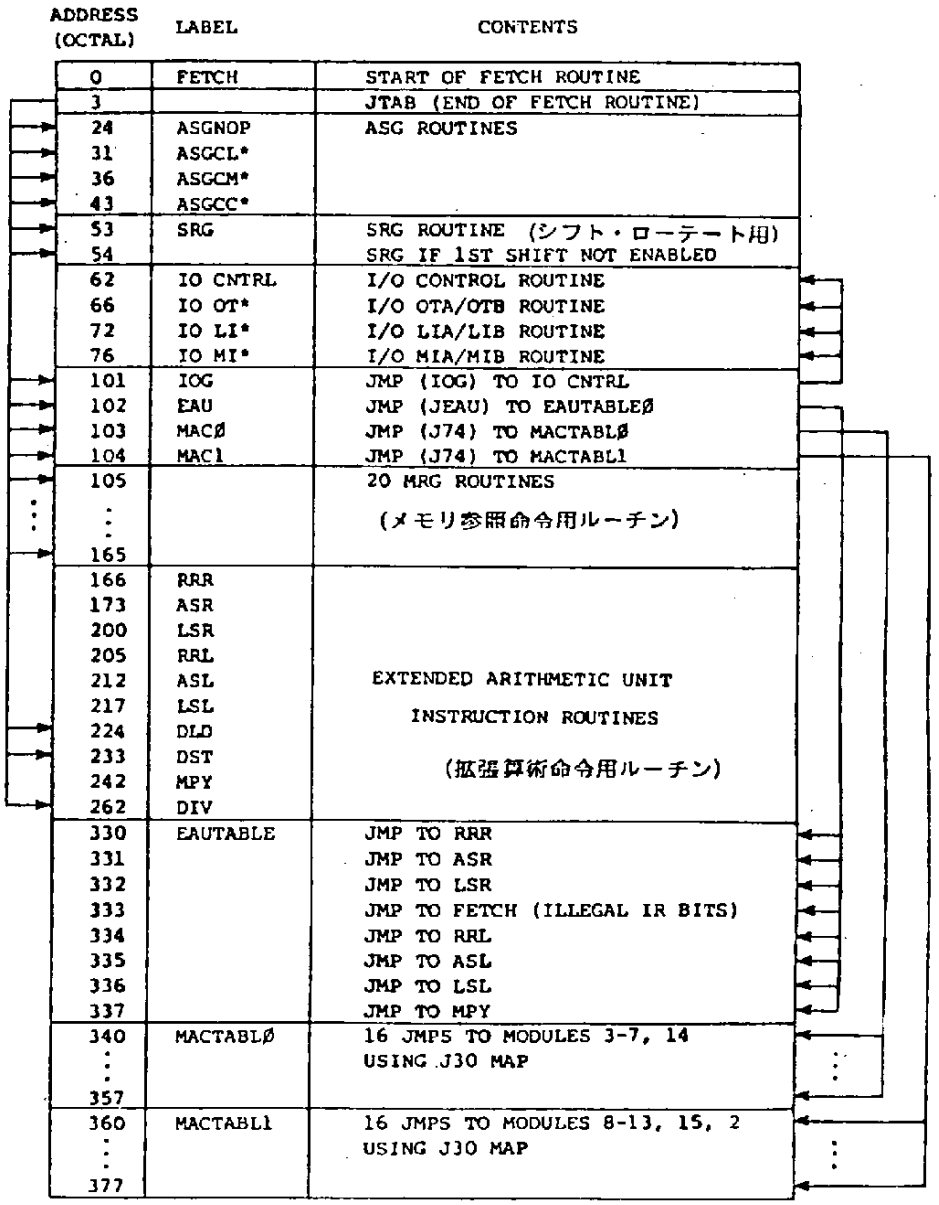


図 4. 4　YHP－21MXの制御の流れ

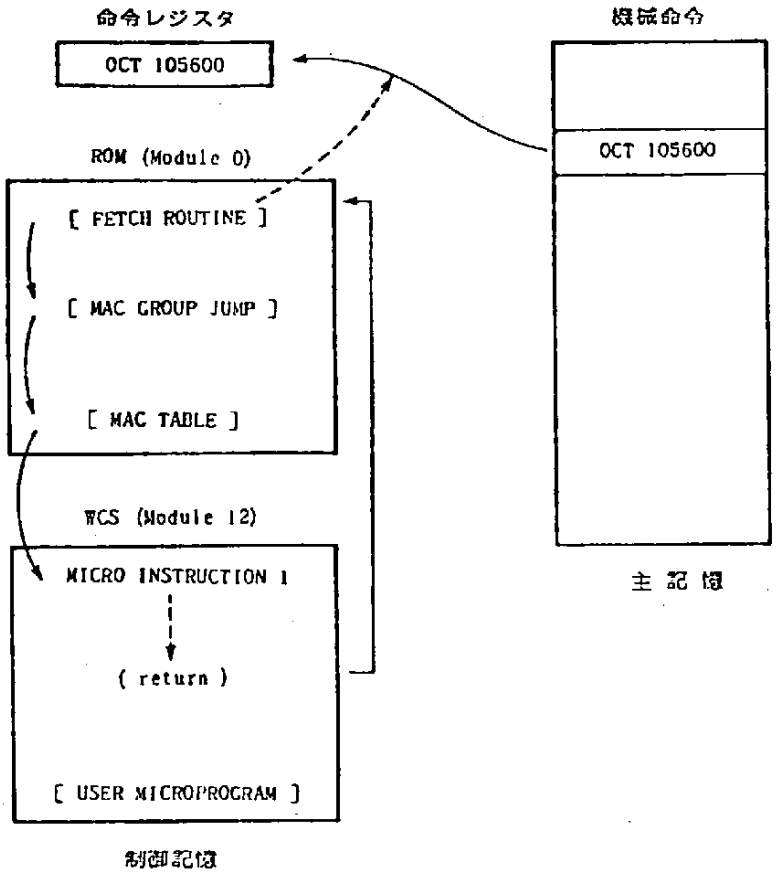


図 4.5　ユーザ・マイクロプログラムの実行

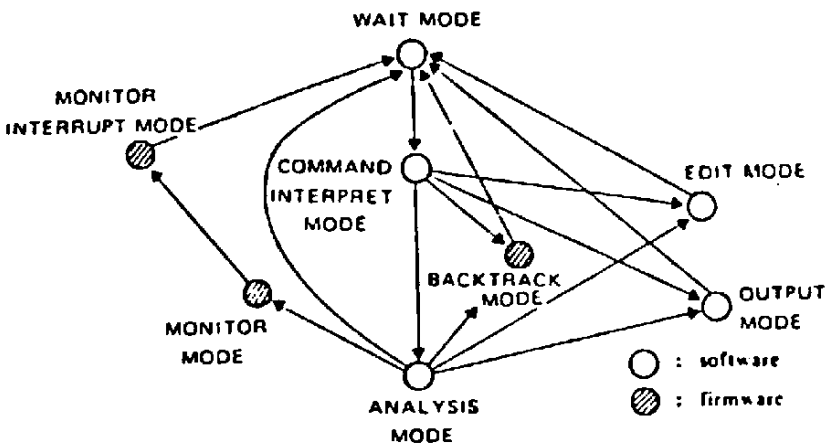


図 4.6　システムの状態遷移

表 4.2　スイッチおよびテーブル

名　称	用　途
モニタスイッチ (SW)	モニタモードのときセットされ、それ以外のときリセットされる。これはインストラクションのエミュレートをモニタプローブ内だけで行なうかどうか判断するために使われる。
モニタテーブル (MTABL)	実行すべきモニタアクション、アクションのシーケンスを制御するテーブル。
パラメータテーブル (PRMTBL)	モニタアクション、アクションで使用されるパラメータを保持するテーブル。
WCS 状態語 (WSW)	現在 WCS にロードされているモニタアクションのファイル名を保持する。

図 4.7に示すようにもしコマンドが，if then else型，もしくは，do case型であるなら，実行すべきモニタアクションおよびアクションのシーケンスを制御するために，モニタテーブル（MTABL）とパラメータテーブル（PRMTBL）を作成し，解析モードへ移る。

解析モードではまずWCS状態語（WSW）とMTABLの最初のエントリを比較して，WCSを書き換える必要があるか判断する。もしWCSの内容を書き換える必要があるなら書き換え，モニタ・スイッチ（SW）をONにする。そして，モニタ・モードへ移りモニタ動作が開始される。モニタ終了するとSWをOFFにして解析モードへ戻る。そこで，MTABLを参照して次に何をなすべきかを決定する。もし，アクションが指定されているなら，それに対応するモード(エディット，出力，バックトラックのいずれか）へ移ることになる。アクションではなくモニタ・アクションであれば，再びSWをONにしてモニタ・モードへ移る。このように，MTABLを参照しながら一連のコマンドが処理されていく。

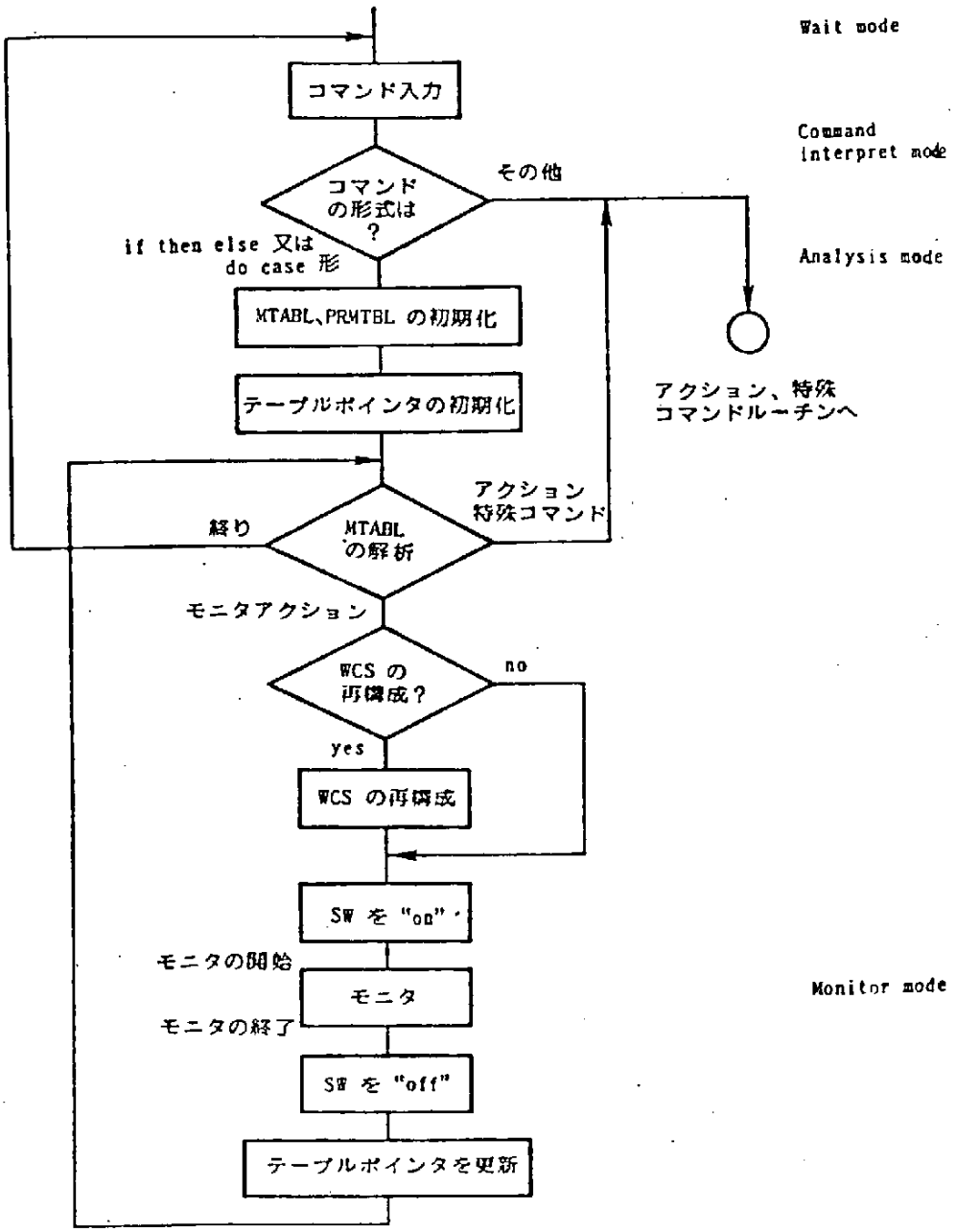


図 4. 7　コマンドの処理過程

(c) ファームウェアモニタ

① 対象

YHP 21−MX の命令体系は，基本命令，拡張命令および浮動小数点命令の 3種類から成るが，SNOOPY ではモニタの対象を基本命令のみとしている。

② 被モニタ・プログラムとシステムの分離

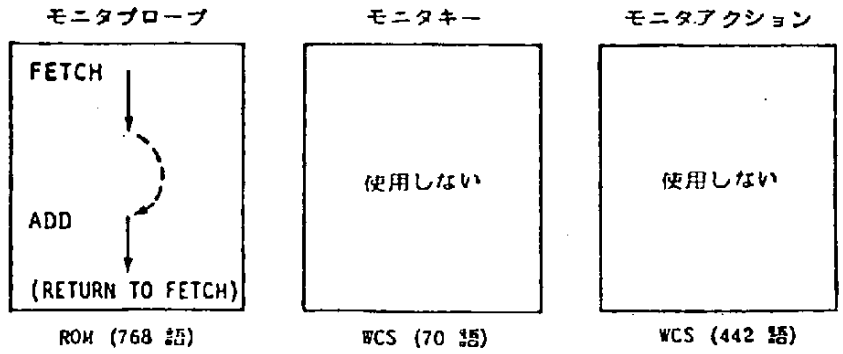
――タグスイッチSWの設置――

被モニタ・プログラムとシステムの分離は，ダイナミック・マイクロプログラミングを用いれば必要ないが，今後の計算機アーキテクチャに重要な要素になると考えられるタグの効率をも実験的に測定するため，フロント・パネルのSレジスタのビット15をタグスイッチSWとして備えている。

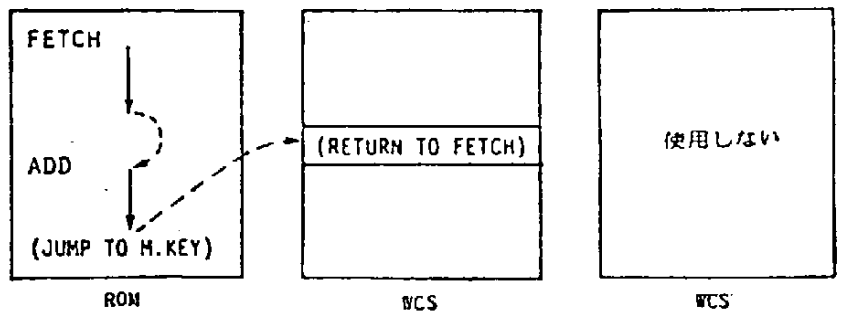
③ ファームウェアモニタの構成

前節で述べたように，機械命令のデコードにはJTABというマイクロオーダーで使用するために，それぞれのマイクロ・ルーチンの先頭アドレスは固定されている。したがつて，マイクロ・ルーチンの任意の場所にダイナミックにモニタ・プローブを挿入してモニタ・アクションを動作させることはできない。そこで既存の制御記憶の内容を検討し，モニタ機能を埋込めるように再設計し，図 4.8のような3部分からなるファームウェア・モニタを実現している。

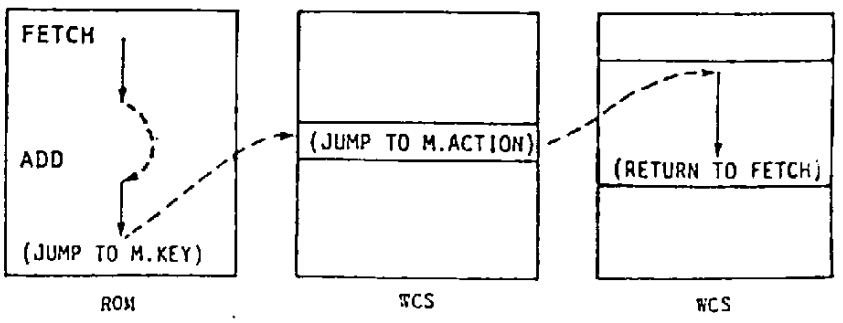
モニタ・プローブ（MP）は，YHP 21−MX 機械語をインタプリートする既存のマイクロプログラムに，モニタ・アクション（MA）で必要な情報を採取するための70個のエントリポイントを付加したものであるが，この部分は書換えの必要がないのでROMで実現している。モニタアクションテーブル（MAP）は，MPとMAの連絡をとるためのものでMP のエントリポイントと同数の要素



(a) ノーマル・モード　(SW ： オフ)



(b) モニタ・モードでの非モニタ状態　(SW ：オン)



(c) モニタ・モードでのモニタ状態　(SW ：オン)

図 4.8　ファームウェア・モニタ

からなる。MA で必要な情報に関係したエントリポイントだけをリンクする役割を果し，WCS上で実現している。又，MA はコマンドで指定されたモニタを遂行するマイクロプログラムからなり，WCS 上に実現されている。

図 4.8 (a)～(c)はモニタを行う場合とそうでない場合の制御を示しており，SWが"ON"でモニタモードのときでも，モニタの対象となるエントリポイントは，MATに依存することが分かる。

(d) WCS のダイナミックな再編成

図 4.9 (a)に示すプログラムの一例は，簡単なソーティングのプログラムの一部分である。LOC1からLOC2まで実行させた際に，どこかでオーバフローを起した。どこでオーバフローを起したかを知るだけでなく，どのようにコントロールが動いたかを調べるために，まず(b)のコマンドでLOC1までバックトラックし，(c)のコマンドを入力した。結果は(d)のようになり，オーバフローは，（JSB＋6）番地の命令を実行したときに発生し，又サブルーチン SUBR へは正常に飛んで，メインへのリターンは RETN番地からであったことも同時にモニタできた。この(c)のコマンドを実行する間にWCSの内容がどのように変化したかを図 4.10 に示す。オーバフローを起す可能性のある命令AD＊（加算），IN＊（インクリメント），DIV（除算）と，コントロールの動きをチェックするためのフェッチルーチンに対応するMATの内容だけが，MA へのJMP命令になっている。又，WCSの内容が各事象モニタごとにダイナミックに変化し，それぞれに最適な制御記憶の構成になることが分かる。

(e) バックトラック

バックトラックを行うために，あらかじめプログラム実行の流れに関する情報と，実行途中で失われるデータを残しておかなけれ

(a)

```
***** SAMPLE PROGRAM **********
*
***** INITIALIZE DATA ***********
INIT    LDA     P10     ; LOAD +10 TO A-REG.
        ADA     M20     ; ADD -20 TO A-REG.
        STA     DATA,I  ; STORE A-REG. TO (DATA)
        ISZ     DATA    ; INCREMENT DATA, SKIP IF ZERO
        ISZ     M20     ; INCREMENT M20, SKIP IF ZERO
        JMP     INIT    ; JUMP TO INIT
***** PRINT INITIAL DATA *********
JSB     JSB     SUBR    ; JUMP SUBROUTINE TO SUBR
LOOP    LDB     M10     ; LOAD -10 TO B-REG.
        INB             ; INCREMENT B-REG.
LOC1    STB     N10     ; STORE B-REG. TO N10
          .
          .
          .
***** PRINT SUBROUTINE SUBR *****
SUBR    NOP
        CLB             ; CLEAR B-REG.
          .
          .
          .
RETN    JUMP    SUBR,I  ; JUMP TO (SUBR)
```

(b)

```
BACK INIT
```

(c)

```
if #P".EQ.SUBR+1 then ..... A
  begin
    if #O".NE.1.AND.#P.EQ.RETN then ..... B
      begin
        if #O.EQ.1, LOC1
        then "OVERFLOW NOT IN SUBR, BUT TILL LOC1"
        else "NOT OVERFLOW IN SUBR & TILL LOC1"
      end
    else "OVERFLOW IN SUBR, OR NOT RETURN VIA RETN"
  end
else "NOT JSB SUBR"
```

* program counter
** overflow register

(d)

```
OVERFLOW NOT IN SUBR, BUT TILL LOC1
LOCATION = JSB+6
```

図 4.9　デバッギングの例

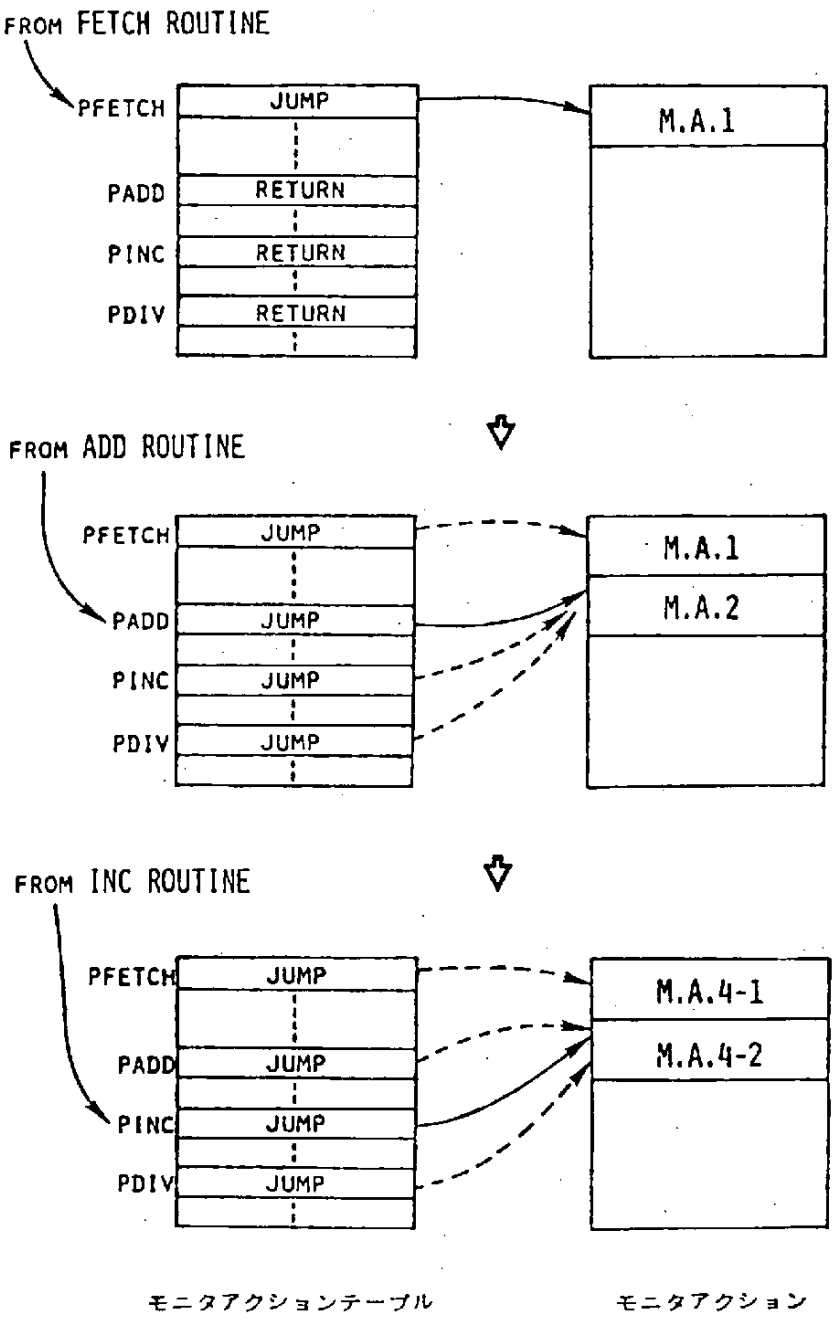


図 4.10　WCSの動的再編成

ばならない。そこでファームウェアモニタの1つのモニタアクションとして，それらの履歴を残す機能を用意した（252マイクロ命令ステップ）。表4.3に使用した履歴保持用の3種類のリングスタックを示す。実行される命令がJMP命令など，制御の流れを変えるときにはCTに"1"がスタックされ，それ以外のときには"0"がスタックされる。又，演算によって失われる情報がオーバフローフリップフロップの内容等1ビットのものはDT2にセーブされ，変数の途中結果等1ワードのものはDT1にセーブされる。

表4.3　バックトラック用リング・スタック

テーブル名	用　途	データ単位	容　量
control table (CT)	実行の流れを記憶するためのスタック	1 bit	100 words
data table 1 (DT1)	1ワード単位の情報を保持するためのスタック	1 word	2000 words
data table 2 (DT2)	1ビット単位の情報を保持するためのスタック	1 bit	100 words

実際にバックトラックを行う機能は，アクションとしてマイクロプログラムで実現されており（335マイクロ命令ステップ），モニタアクションで作成されたテーブルを参照しながら1ステップずつ逆もどりさせる。その過程で失われた情報はテーブルから元にもどされ，演算で解決するものはそれを行う。図4.11にバックトラック機能のフローチャートを示す。

<4> 割込みを扱うプログラムのデバッグ

従来のソフトウェアによるデバッガにおいて，割込みを扱うシステム・プログラムのデバッグは最も困難であった。それは，割込み機構がハードウェア，又はファームウェアで実現されているため，完全なシミュ

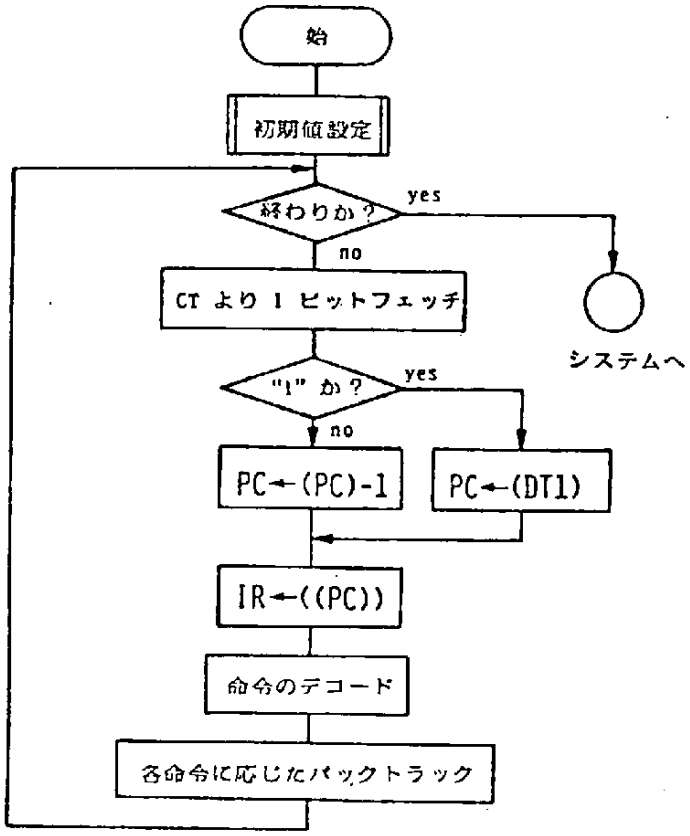


図 4.11　バックトラックの処理

レーションが難しく，又実現できても実時間で再現することは速度的に困難であった。

YHP 21−MXの割込み機構は図 4.12に示すように，ファームウェアで割込み原因を調査し，割込みベクトルを通してその原因に対応する割込み処理ルーチン（ソフトウェア）へ制御を移すように構成されている。SNOOPY ではマイクロプログラムで構成されている割込み用ルーチンの中に，モニタアクションとしてデバッグ機能を埋込み，マイクロプログラムで割込み原因および割込み処理ルーチンの動作をモニタできるので，実時間処理の環境でも十分追従できるものとなった。又，割込み処

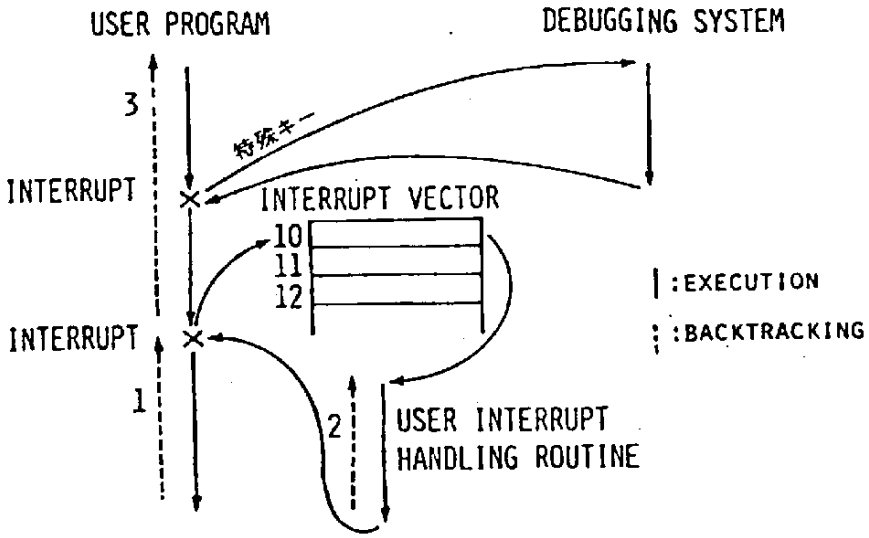


図 4.12　割込み機能のモニタ

理を含むプログラムのバックトラックも図 4.12に示すように，一般のプログラムの場合と同様に可能となった。

〔3〕 SNOOPY(42)の評価

(1) ファームウェアモニタ

ファームウェアモニタの速度的な効率を調べるため，3種のモード；

(i) スタンダード・モード（YHP 21—MX標準制御記憶装備）

(ii) ノーマルモード（MPのみ）

(iii) モニタ・モードでモニタなし（MAT でリターン）

の制御記憶を用いた場合について実行時間の測定を行った。測定に用いたプログラムは，YHP 21—MX DOSⅢシステムプログラムのアセンブラ，FORTRAN コンパイラ，及びユーザ・プログラムとしてのFORTRANプログラム（シンプソンの公式）である。表 4.4 にその結果を示す。これからわかるようにマイクロプログラムだけで，モニタを構成した場合の基本的なオーバヘッドは，約 2倍以下である。

表4.4　モニタ・アクションの速度評価

PROGRAM	STANDARD MODE	NORMAL MODE	NON-MONITORING IN MONITOR MODE	RATIO
21MX DOS-Ⅲ ASSEMBLER	104.0sec	161.7sec	205.1sec	1:1.55:1.97
FORTRAN Ⅳ COMPILER	7.0	8.3	8.9	1:1.17:1.27
FORTRAN PROGRAM	1.6	1.7	1.9	1:1.06:1.19

(2)　バックトラック

従来の方法[50][51]では，セーブ用に使う記憶容量は少なくてすむが，バックトラック可能なステップ数がプログラムに依存して異なり，しかもバックトラック時間とバックトラックのステップ数に全く比例関係がないという欠点があった（例えば，1ステップもどるのに5,000ステップもどるより時間を要する場合がある）。又，実行中に割込みがかかるとバックトラックできなかった。SNOOPYではセーブ用に使う記憶容量は多少多いが，最大1,600ステップ（拡張可能）のバックトラックが可能であり，その処理速度もファームウェア化されているため極めて速い。表4.5から分かるように，バックトラック用モニタアクションのためのオーバヘッドは，約2～4倍（T3：T2）であり，アクションもソフトウェアと比べ約10倍の効率向上を示している。

(3)　マイクロプログラム化の評価

マイクロプログラムを使ってデバッガを作成すると(1)，(2)で述べたように，速度的に有利なだけでなくデバッギングシステム全体を小さくする。これは，従来主記憶にあった事象モニタのためのトレーサが，すべて制御記憶に移るからである。デバッギングシステムが大きくなると，

表4.5　バックトラックの実行時間

(a)バックトラックモニタアクションの実行時間

INSTRUCTION	STANDARD MODE	NORMAL MODE	MONITOR ACTION	
			MIN	MAX
LDA	1.95μs	3.90μs	12.0μs	17.9μs
ADA	1.95	3.90	15.6	24.1
ISZ(SKIP)	2.60	4.88	12.7	18.5
ISZ(NO SKIP)	2.60	4.55	9.10	17.2
JMP	1.95	3.90	11.7	17.6

(b)バックトラックアクションの実行時間

INSTRUCTION	MICROPROGRAM		SOFTWARE *	
	MIN	MAX	MIN	MAX
LDA	10.1μs	16.6μs	90.3μs	131μs
ADA	17.2	26.7	174	260
ISZ(SKIP)	13.0	19.5	117	156
ISZ(NO SKIP)	9.75	15.3	99.7	139
JMP	9.43	15.9	76.3	115

* HP-2100 アセンブリプログラム

ユーザがデバックできるプログラムの大きさに制限がつくことがあるのでこれは大きな利点である。

次にSNOOPYの機能は，原理的には本文中で述べたようにソフトウェアでも実現可能である。しかし，実行効率が悪いと原理的に実現できても実用には耐えられない。例えば，CDC6600大型計算機上に作られたデバッガ“HELPER”はソフトウエアで事象モニタ，バックトラック機能などを備えているため，デバッガの速度が通常の速度に比べ100～200倍と遅く，実用上大きな問題であると報告されている。[50] マイクロプログラム化するとこのようなことはなく，SNOOPYでは〔3〕で述べたように事象モニタ機能やバックトラック機能も十分実用に耐えられると考えられる。

以上述べたように，システムの作成，使用においてファームウェア化は有用であるが，代償として制御記憶の増加は免れない。しかし，SNOOPYでは，ダイナミック・マイクロプログラミングの本来の機能を活用し，その増加を最少に抑えることができる。

4.3.3 ソフトウェア，ファームウェア，ハードウェアのトレードオフ

YHP21－MX のホストマシンとする実験を基にして，高機能デバッギングシステムに関連して今後の計算機アーキテクチャにどのような配慮を行うべきかについて調べる。

(1) 状態指示機構

IBMのS/370に備えられているようなPSW(Program Status Word)の概念は，モニタ効率を上げるには絶対に必要なものである。PSWが持つ意味は２つに分類でき，１つはALUの演算結果がどのような状態か（＋か－か０かなど）を示すものと，もう１つはOSモードとユーザモードなどの区別をつけるプログラムの性質を示すものであるから，デバッグ時には，PSWが容易にアクセスできるようにしなければならず，これは専用レジスタなどに置くべきである。又，ユーザが指定する任意の条件で実行時に，PSWがセットされることが望ましく，これはマイクロプログラムで制御す

べきである。

(2) 事象モニタ機構

ユーザが指定した事象をチェックするような，専用ハードウェア機構があるとよい。すなわち，事象チェックのための演算は，実行のために使われる演算器とは独立の演算器で行った方がよい。YHP21－MXではこのような機構がなかったため，PSWに相当するALU状態フリップフロップを，事象モニタのたびにマイクロプログラムで退避しなければならず，そのオーバヘッドは大であった。又，別の方法として実行時にPSWにセットするか否かの区別を，マイクロ命令で制御できるようにしておく方法も考えられ，この方法では演算器は1つあればよく，コスト的に有利である。

(3) モードの切換え

〔3〕の表4.4から明らかなように，実行モードとデバッグモードの切換えスイッチのオーバヘッドは約1.5倍である。もしスイッチがハードウェアで備えられているとモニタのオーバヘッドは約1.2倍（モニタモード／ノーマルモード）となるので，スイッチはハードウェア的機構として備えるべきである。しかし，切換え時の計算機の状態（例えば，レジスタなど）の退避は，自由度を増すためマイクロプログラムで行うことが望ましい。

(4) レジスタ，スタック

モニタのためのパラメータを格納するため，モニタ用のレジスタがあるとよい。又，モニタ専用のLIFOスタック，バックトラック用のリングスタックを設けることが望ましい。

(5) コンパイラとのインタフェース

コンパイラの開発において，シンボルテーブルを残しておくなど，コンパイラもデバッグのことを考慮して作成すべきである。

4.4 動的環境情報による静的方法

4.4.1 アーキテクチャ・チューニング

ダイナミック・マイクロプログラミングにアーキテクチャ・チューニングの可能性があることは既に第3章で述べた。ここでは，その具体的な方法について述べる。

(1) Snyderの方法[55]

H.P.社のD.C.Snyderによれば，コンピュータの動作特性を知る最も簡単な方法の1つは，単にシステムが動いている間に10〜20回 HALTスイッチを押し，その度にプログラムのアドレスレジスタの内容（P Valueという）を書きとめることである。普通はこのP Valueの範囲には目立ったところがあつて，ここをチューニングすればよい。又，どのくらい改善できるかの推定法としては，例えば20回のうち7回がある特定のルーチンであるならば，もしこのルーチンを無限に速くした時のスピードアップは7/20，すなわち35%であるといえるだろう。

この理論は多少機細さに欠ける感じを与えるかもしれないが，一つの目安とみることもできる。

(2) Abd−All らの方法[56]

Snyder の方法によれば，解析，自己修正にかなり人間の手を煩わした。Abd−All の方法も注目するもの（環境）は実行時アドレス・プロファイル（Profile）であるが自動的に解析を行い，自己修正を行うための考慮がなされている。先ず，アドレストレースによりプログラムのループを見つける。これがスタティックな方法と異なるのは，ループの動特性も合わせて検出することで，ループの回数，ループ中で参照されるメモリロケーションの回数をデータとして収集する。解析アルゴリズムは，ループ中で最もよく使われるメモリの領域を，マイクロプログラムだけでアクセスできるMGPレジスタ（マイクロ・ジェネラル・パーパスレジスタ）に置き換え

てやることである。実際には上位いくつかが置き換えられる。自己修正機能としては，ループに入る前にメモリーレジスタ転送を一度だけ行い，ループ中のメモリ参照命令はレジスタ参照命令に置き換わるようにすることであり（Preload という），図4.13(a)に示すように再びループを抜け出る時にレジスターメモリ転送を行ない，もとに戻す（Restore という）。実験結果を図4.13 (b)に示す。この実験は HP 2100 を使って行われた。

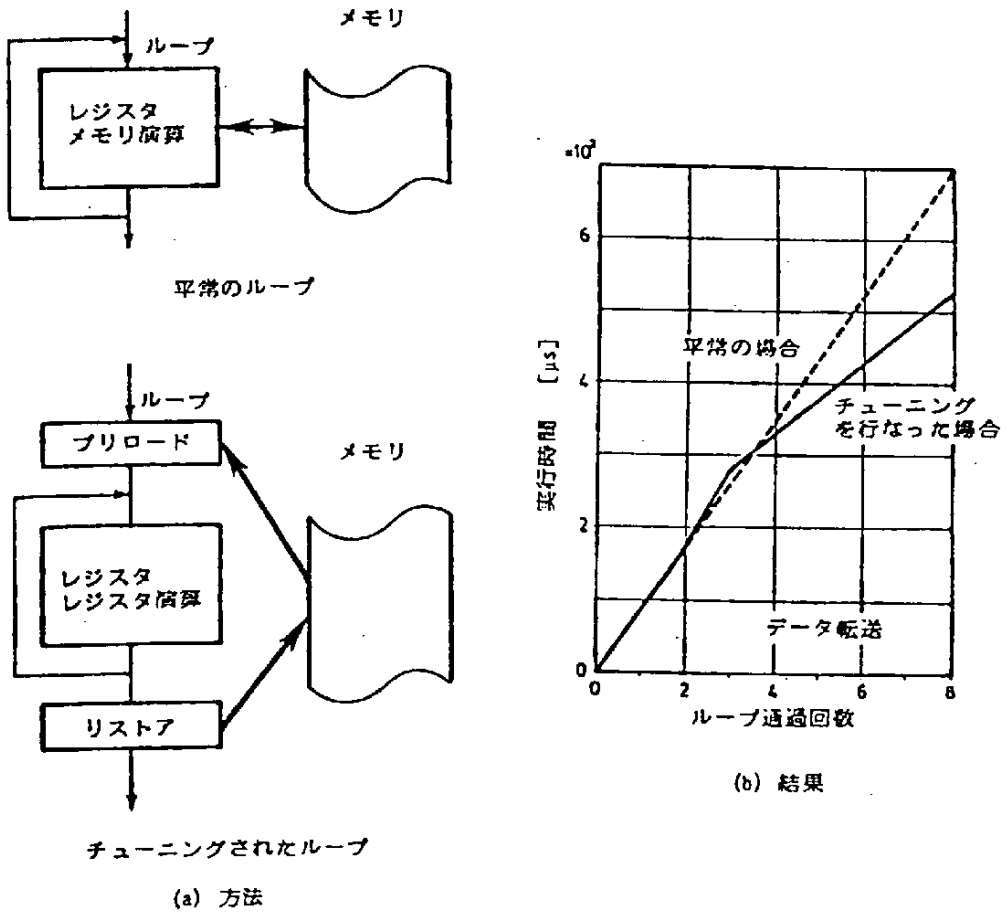


図 4. 13　Abd－Alla の方法と結果

Abd－All らによると，この方法で4倍ぐらいまで実行速度が改善できると報告しているが，この方法ではループが少ないか，あるいは dirty ル

ープのある構造のきたないプログラムでは効果が少ない。またチューニングの結果に一般性はない。すなわちチューニングは特定の問題のみに対して行われ，他のチューニングにその結果を役立たせることはできない。

(3) Sakamura の方法[3]

前述した2つの方法は，環境としてアドレスのパターンに注目した。Sakamura の方法はアドレスパターンではなく，インストラクションのパターンに注目する。

図4.14に示すのはインストラクションパターンの例である。これは，B 1700 上にインプリメントされたPASCALマシンのコンパイラを実行させた時，PASCAL マシンの中間言語がそれぞれどのくらい実行され，又実行された後でどの中間言語に移るかを示している。この遷移に注目して遷移の大きな命令の組をインストラクション・パターンと呼ぶ。そこで，このインストラクション・パターンに注目する理由は，ある程度一般性を持ったチューニングが重要であると考えたことにある。一般性のあるチューニングとは，一度行ったチューニングの結果を覚えておき，次回からこの結果を使ってチューニングを行い，チューニングのオーバーヘッドを減らすことができる方法であり，チューニングにおける学習の基本概念といえる。

4.4.2 学習の概念を持つ自動アーキテクチャ・チューニングの原理と実現方法

本節では学習の概念を持つ自動アーキテクチャチューニングの原理[10]について述べる。

<1> 基 本 原 理

ISPレベルでの計算機アーキテクチャの自動チューニングに関するメカニズムの簡略化したモデルを図4.15に示す。アーキテクチャの最適化，すなわちチューニングに関して要求される基本的機能は次のとおりである。

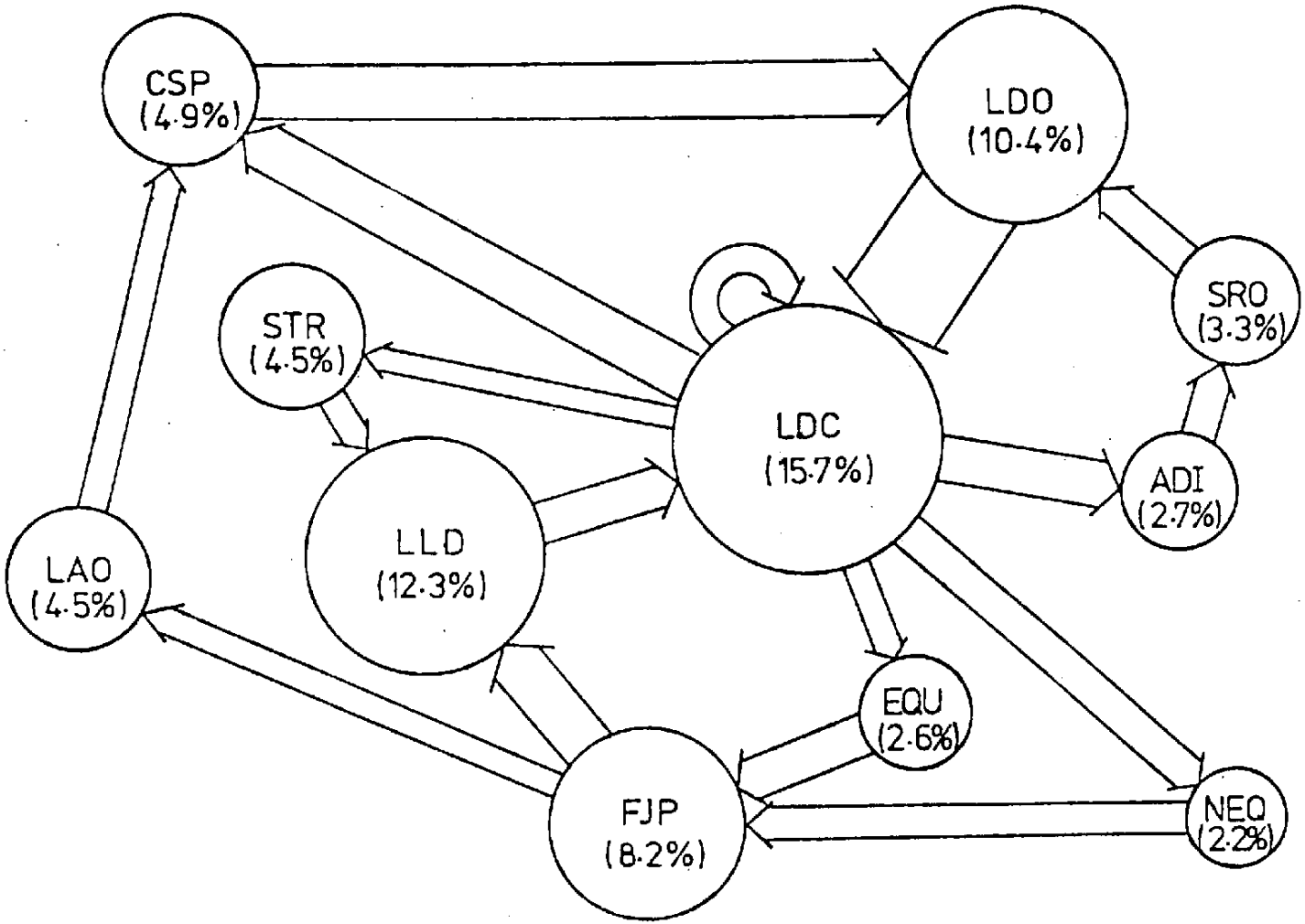


図 4. 1 4　　PASCAL マシンの命令パターン

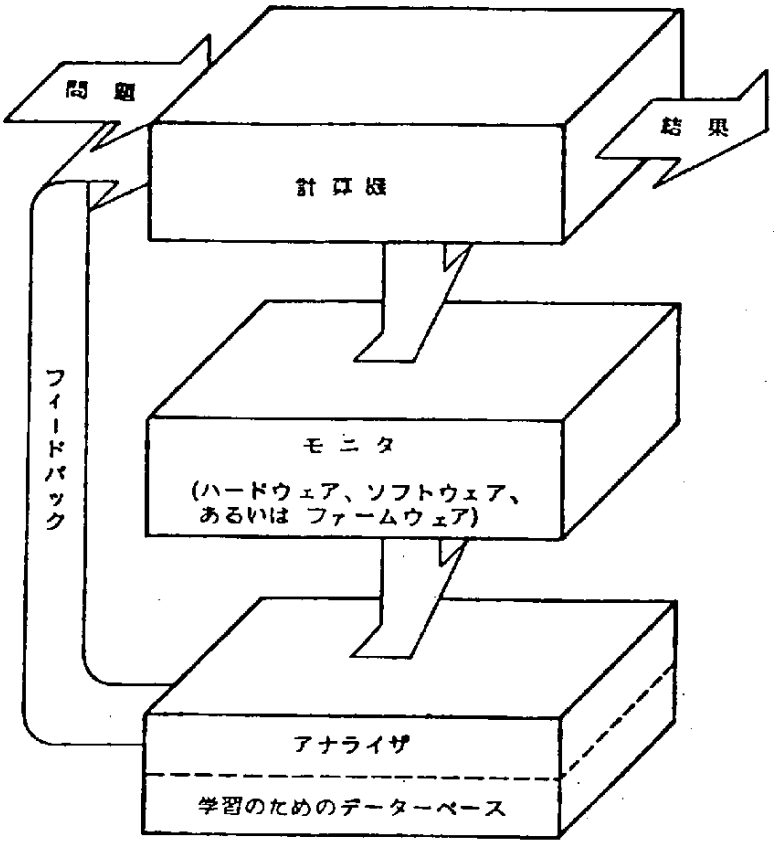


図 4.15　自動チューニング機構の概念

(1)　プログラムのモニタ

計算機で解かれる問題の動作特性に関する詳細情報が，最適化プロセスを行うのに必要である。この情報は，機械命令の相対頻度，命令列（実行される順序に注目した）の相対頻度，およびアドレスとデータの値の相対頻度などである。モニタはハードウェア，ソフトウェアあるいはファームウェアで作られ，この情報を収集する。

(2)　データの解析

モニタから集められた情報は，計算機で解析される。この計算機は別の計算機であってもよい。解析の機能は，実行時間とメモリ容量を減少さ

せる新しい命令を作るために，最適化できる命令パターンの全ての可能な候補を見つけることである。これは応用プログラムと実行プロフィルの解析によって行われる。

(3) 計算機へのフィードバック

アナライザによって合成された新しい命令は，新しいマイクロプログラムに変換される。これによって元の命令に比べて小さいメモリ容量ですみ，実行時間を短くできる。合成されたマイクロプログラムは新しい拡張アーキテクチャを形成するプログラムを実行することによって計算機のWCSへロードされる。これは計算機アーキテクチャの動的修飾である。ダイナミック・マイクロプログラミング技術がこの動的修飾を大変効果的に行える点に注目すべきである。

(4) チューニング動作の学習

本方法の特徴ある点は，アナライザにデータベースを持たせることである。上述のチューニング・プロセスは，通常目的のパフォーマンスの改善が得られるまで繰返される。従って，アナライザはチューニングを行うたびにデータベースにチューニングの結果を保持する。アナライザはデータベースの内容を参照することによって，チューニングの繰返し数を最小化することができる。

＜2＞動的な計算機の振舞いに基づく中間言語の最適設計

一般にマイクロプログラム制御計算機では，解こうとしている問題は高級言語で書かれていて，これは機械語あるいはある高級言語向けに設計された中間言語の列に翻訳される。機械語，すなわち命令セットは対応するマイクロプログラムに翻訳される。問題をより低レベルの言語で記述する方が実行効率が良いことは良く知られている。即ち問題が高級言語で書かれるより機械語で書かれる方がパフォーマンスが良く，さらに機械語で書かれるよりマイクロプログラムで書かれた方がパフォーマンスが良い。実行効率の改善はISPレベルで行われ，この手続きが「アーキテクチャ・チ

ューニング」と呼ばれる。

最大の効率を得るには，全てのプログラムをファームウェアで作ることである。しかし，プログラムの全てをファームウェア化することはコスト・パフォーマンス的には最良ではない。プログラムの実行の動的振舞いを考慮し，プログラムの部分をファームウェア化すると良い。この場合，最適な戦略はプログラムで最も実行頻度の大きい部分にマイクロプログラミングを適用することである。最適化する部分を自動的に検出するために，プログラムは新しいアルゴリズムに従っていくつかのブロックに分割する。次に，ブロックの荷重を測定しファームウェア化するブロックを決定する。これは実行荷重の最も高い所から始める。

＜3＞ IML命令パターンのチューニング・アルゴリズム

チューニングのプロセスを行うのに重要な点は，プログラムの特性についての詳細な情報が必要である点である。各IML命令の使用度は，プログラム実行中のいつでも一様というわけではないので，命令の相対頻度と命令列の相対頻度はこの情報として必要なものである。命令の一連の対あるいは組は新しい命令を作り，「命令パターン」と呼ぶ。従って，プログラムは以下に示す静的解析による命令パターンの有限個の列で表現できる。

命令パターンの重みづけ

各々の命令パターンに重みづけを行うためには，命令列の頻度を測定すればよい。この場合，命令パターンの途中には分枝命令によってブロックに分解される。ブロックの実行頻度はハードウェアあるいはファームウェアモニタで測定される。命令パターンは測定されたブロックの頻度から計算することができる。

ファームウェア化される命令パターンの選択

ファームウェア化する命令パターンを選択するには，次の2つの戦略が考えられる。

(1) 最大実行効率を得る方法

次項で述べるチューニングの効果を予測する式を使って，見つけられた命令によるパフォーマンスの改善度($\hat{\mu}$)を予測することができる。先ず$\hat{\mu}$の値が最大の命令パターンを選ぶ。ある命令パターンは他と重なっていたり，一方が他方を包含していることがあるので，ある命令パターンを選ぶことによって他の命令パターンの荷重が変化することがある。従って次に，再度静的解析を行い，命令に重みづけを行い，次の命令パターンを選ぶため$\hat{\mu}$を計算しなおす。上記のステップを$\hat{\mu}$の合計がある値を越えるまで繰返す。

(2) 最大の経済効果を得る方法

この方法は上で述べた方法と同様で，ただ$\hat{\mu}$の代わりに$\hat{\mu}/\hat{\theta}$を使用する。ここで$\hat{\theta}$は，命令パターンをファームウェア化した場合に必要とするWCSの予測値であり，これもチューニングの効果予測式から求まる。チューニングは次の条件が満たされるまで繰返される。

$$\Sigma\ \hat{\mu} > y \quad \text{あるいは} \quad \Sigma\ \hat{\theta} > y'$$

新命令の合成

選択された命令パターンから新しい命令を作る。命令列を1つのマイクロプログラムで作る。これによってメモリ容量と実行時間を元の命令列より少なくできる。チューニングは現在のマイクロプログラム最適化手法[14]に基づいており，メモリ参照回数の減少，内部リソースの有効利用，並列処理などを行なう。

フィードバック

合成された命令は，WCSに格納され，コンパイラのコード生成部に登録される。コンパイルされているコードは機械コードレベルで編集を行い，対応する新しいコードに変換する。

<4>チューニングの予測[14]

もしWCSの容量に制限がないならば，プログラム実行中に検出されたパターンに対応する新しい命令を合成することにより，最高のチューニング

ができる。しかしながら，WCSの量はシステムの価格に直接響くし，マイクロプログラムを書く手間も少なくしたい。この手間を定量的に評価するのは難しい。しかしながらマイクロプログラミングの手間がマイクロプログラムのステップ数に比例すると考えると，手間はWCSの量に比例すると考えられる。従って最大の効果があり，かつ最小のWCS量ですむ命令パターンを選択したい。このためファームウェア化の効果，すなわち実行効率の改善度とWCSの増加量との比を予測する方法を命令パターンのファームウェア化に際し開発しなければならないが，もしこのような予測が可能であるならば，命令パターンの選択はチューニングのオーバーヘッドに基づいて行われる。

ここで，マイクロプログラム化された新しい命令による効率の改善度(μ)が命令パターンの荷重(ξ)及びWCSの増加量(θ)の関数と考える。すなわち，

$$\mu = f(\xi, \theta) \qquad (4.1)$$

さらにθは命令パターン長さ(γ)の関数と考える。すなわち，

$$\theta = g(\gamma) \qquad (4.2)$$

この関数f及びgを実験的に決定する。関数fを定めるために，$\log\xi$と$\log(\mu/\theta^m)$との関係を求めた。図4.16に示すように，ほぼ直線的な関係がある。すなわち図より，

$$\log(\mu/\theta^m) = n\log\xi + a \qquad (4.3)$$

よって

$$\mu = A\,\theta^m\,\xi^n \qquad (4.4)$$

ここで，A，m及びnは中間命令セットによって決定される定数である。同様に，θとγの関係は，図4.17に示すように一次式

$$\theta = b\gamma + C \qquad (4.5)$$

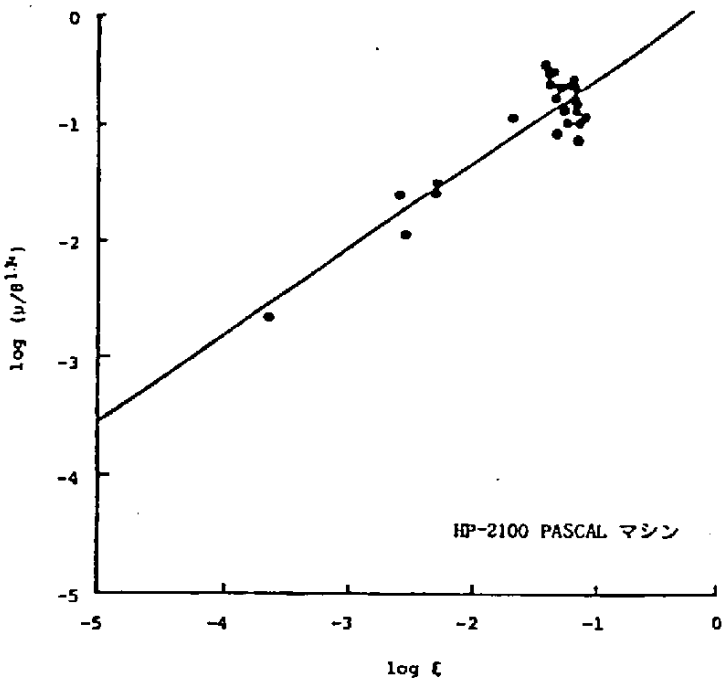


図 4.16　ξ と μ/θ^{m} との関係

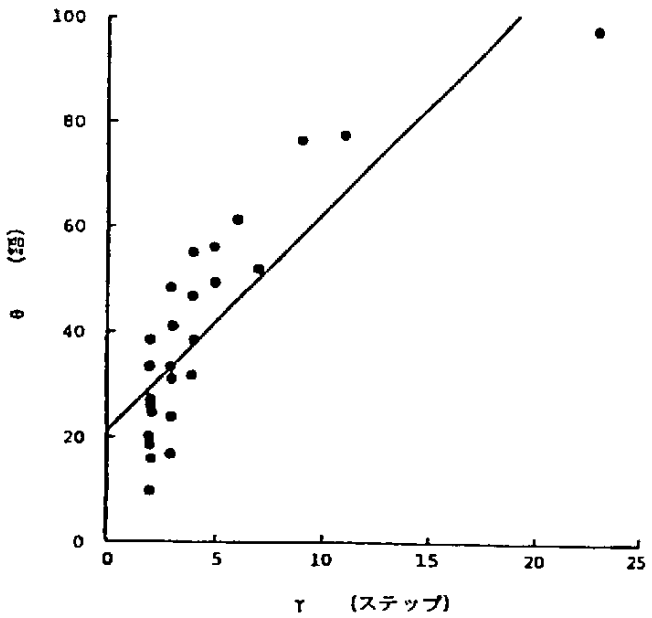


図 4.17　γ と θ との関係

で表わされる。ここで，b及びCは中間命令セットで決まる定数である。

式（4.4）は，命令パターンの荷重と，対応するマイクロプログラムの量からファームウェア化の効果が予測できる。従って，この関係を「チューニング曲線」と呼ぶ。明らかに定数α，A，m，nは正で，荷重ξおよび WCS の増加量θが大きくなると改善度は大きくなる。

上で述べたアルゴリズムのフローチャートを図 4.18 に示す。

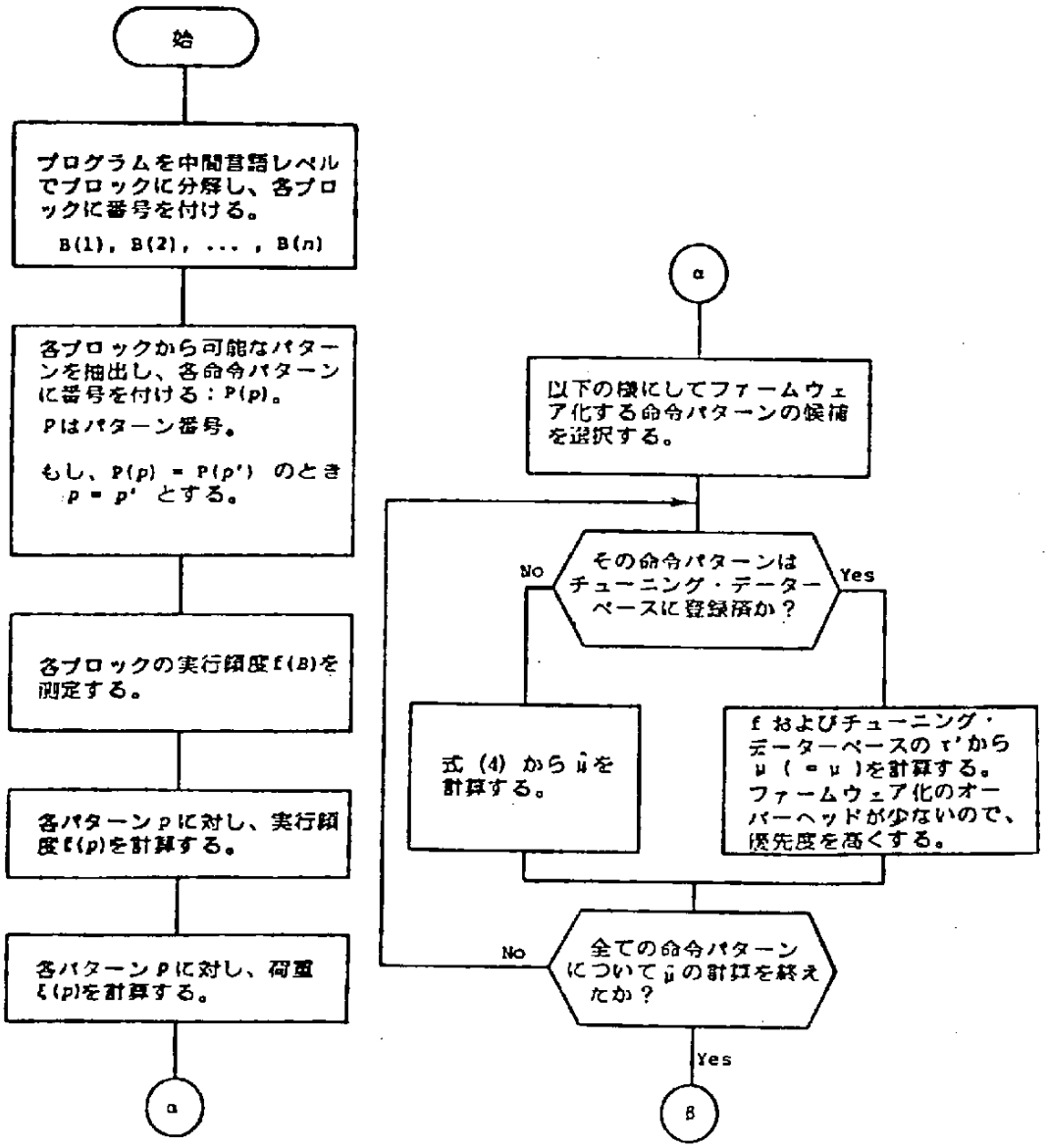


図 4.18　チューニング・アルゴリズム

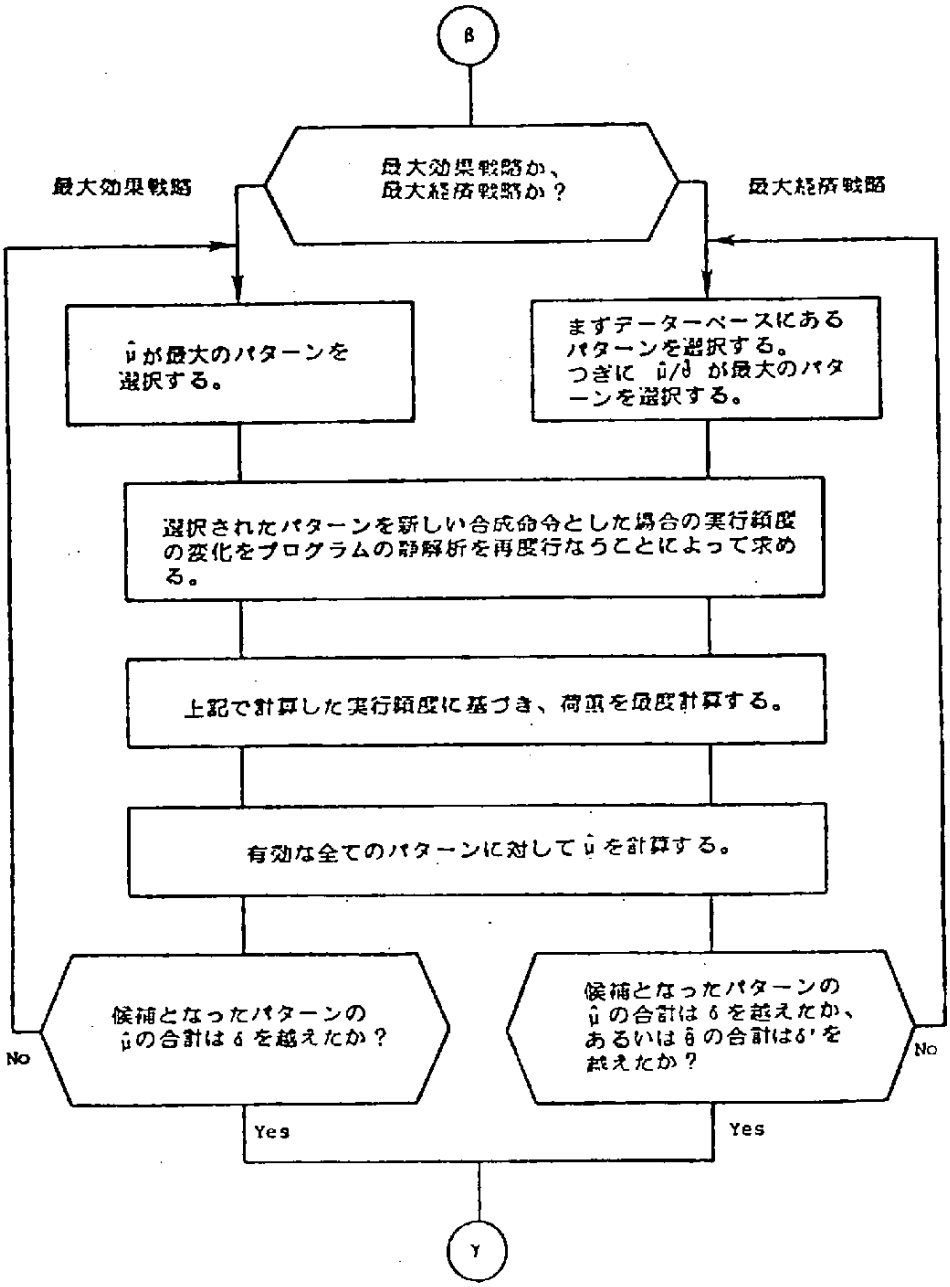


図 4.18　チューニング・アルゴリズム（続　き）

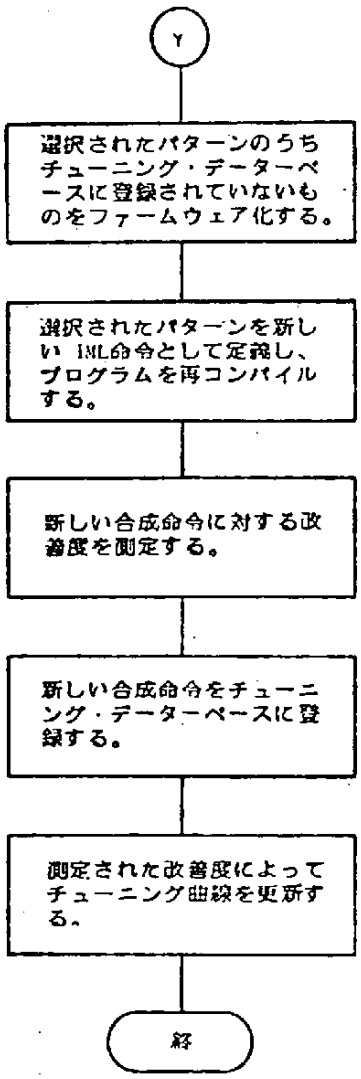


図4.18　チューニング・アルゴリズム（続　き）

4.4.3 実験システム

モニタ，アナライザ，シンセサイザから成るチューニングメカニズムを，“Automatic Performance Evaluator（APE）”と呼ぶ。[12]

前節で述べた原理の効果を証明するために開発された，２つの簡単なAPEメカニズムについて述べる。

＜1＞実験システム１－HP2100とハードウェアモニタ

この実験システムでチューニングされるホスト計算機は16ビットのマイクロプログラム可能なミニコンピュータである。その構成を図4.19に示すが，制御記憶は256×24ビット語の機械語をインタープリトするROMボートと，ユーザマイクロプログラムのためのROMと同等量のWCSボート３枚から成つている。APEのモニタとしてはCOMRESS社のDYNAPROBE 7900＋800ハードウェアモニタを使用している。ハードウェアモニタすなわちD7916 カウント／タイプハードウェアモニタ及びD 8028 マップ／ストアタイプハードウェアモニタによって，ある事象の累積時間を計測し，これからの出力を解析，合成するためにPDP－11V03を使用している。測定されるプログラムはHP2100ホスト計算機で実行される。D7916及びD8028 ハードウェアモニタは，命令パターンの荷重を測定するためにプローブを通してホスト計算機からの信号を集める。ハードウェアモニタによつて集められた信号は，PDP－11V03に入力され，チューニングの解析が行われる。PDP－11V03のLSIバスはHP2100のI／0バスに接続され，I／0インターフェースを経由してフィードバックループを形成する。

本システムの主な特徴の１つは，モニタ機能に対するオーバーヘッドが全くないことである。これは，高速のハードウェア・モニタを使用し，チューニング解析にはホスト計算機とは別にLSI－11を使用しているためである。

＜2＞実験システム２－バロースB 1700

バロース社のB－1726は24ビットデータのマイクロプログラム可能な計算機である。この計算機は４つの汎用レジスタ，32のスクラッチパッド等多くの内部リソースを持ち，ビットアドレッシング，レジスタのサブフィールドのアクセスが可能である。マイクロプログラムを格納するWCSの容量は，16ビット語で４K語である。ここに，ファームウェアモニタを持つ

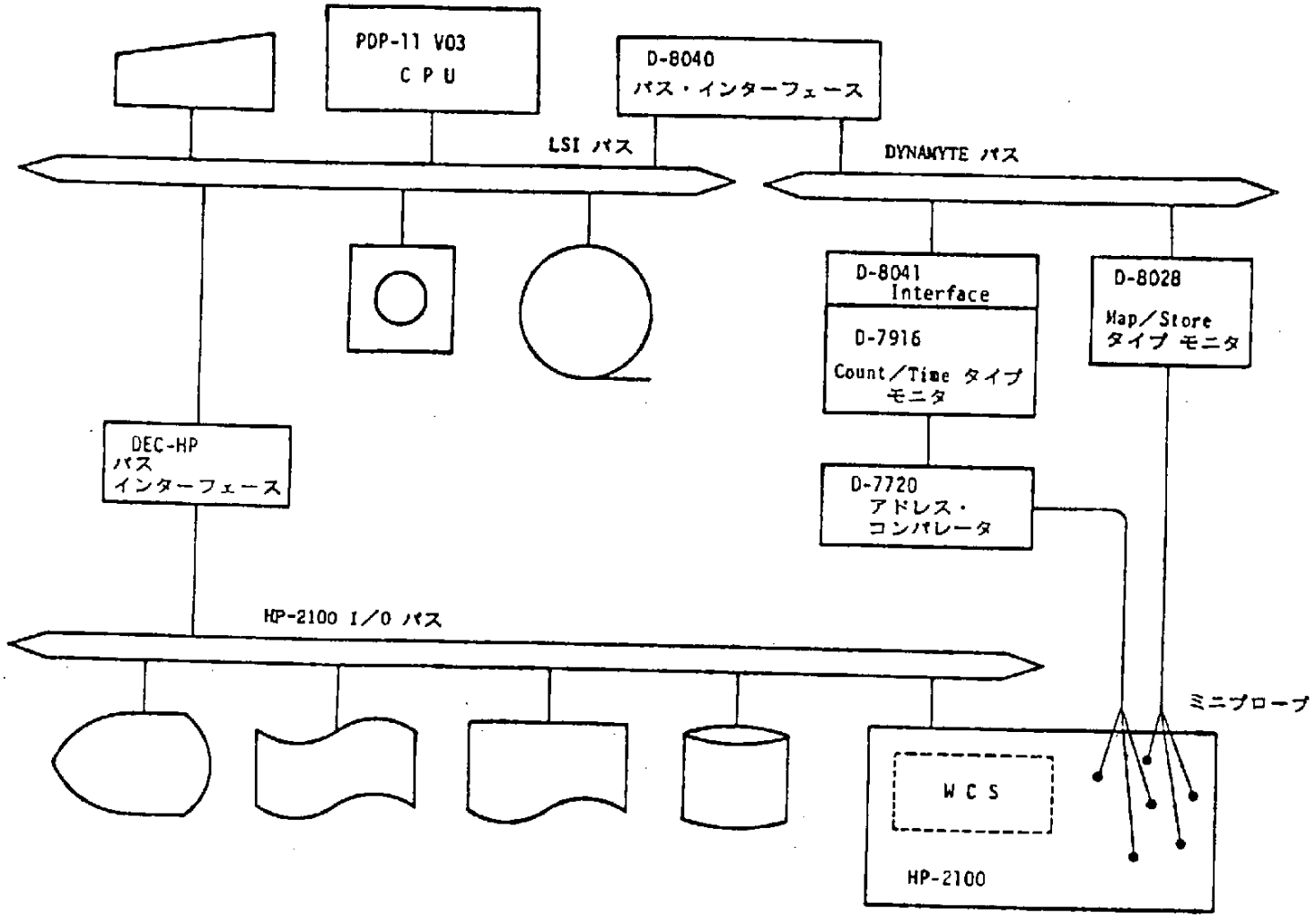


図 4. 19　　実験システム(1)の構成

たいくつかのSコードインタープリタを作成した。命令パターンの頻度を測定するファームウェアモニタは，Sコードインタープリタの命令フェッチルーチン中に埋込まれている。従って，B−1700には特定のハードウェアは付加されていない。このため，モニタリングのためのオーバーヘッドは元来の計算機の速度に対し，命令フェッチ部のみが約2倍遅くなっている。しかしながらチューニングはそれほど毎々行うわけではなく，もし行った場合でもモニタのためのマイクロプログラムは，Sコードインタープリタを再構成する時外してしまうので，全体としてのオーバーヘッドはそれ程大きくはない。

<3> 実験に使用した中間言語

チューニングされるIMLとして次の2つのものを使用した。

(1) 60の命令から成る PASCAL のための IML

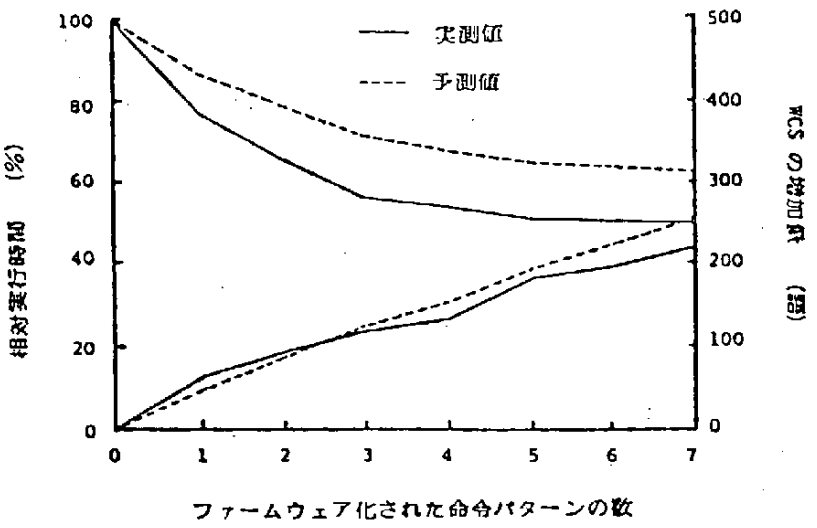
これは，K.V. Nori， V. Amman らによるもので[67]，HP−2100, B 1700の両者でエミュレートした。

(2) HP−2100 の FORTRAN IML，すなわちHP−2100 上でインタープリートされる機械語の例。

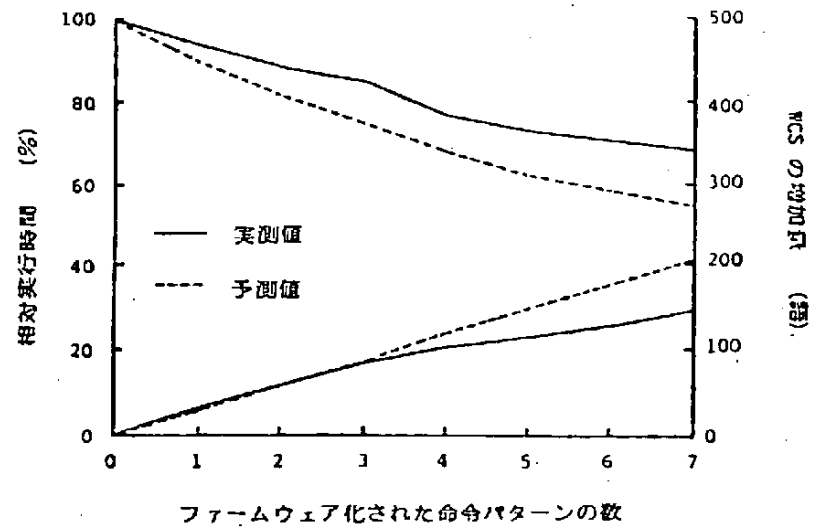
4.4.4 実験の結果

<1> チューニングの結果

図 4.20〜図 4.23 にチューニングの結果を示す。図 4.20の例をとれば，HP−2100 によりエミュレートされる PASCAL マシン上で実行されるソーティングのプログラムのチューニングの結果である。選択された戦略が最大実行効率戦略の場合，プログラムの実行速度は2倍になる。一方WCSの増加量は210ワードである。この場合7つの命令パターンをファームウェア化している。さらにオブジェクト・コードの量は392バイトから284バイトに減少している。これは，プログラムの静的解析によって効果的なコード・コンパクションが行われることを意味する。最大経済効果戦略を



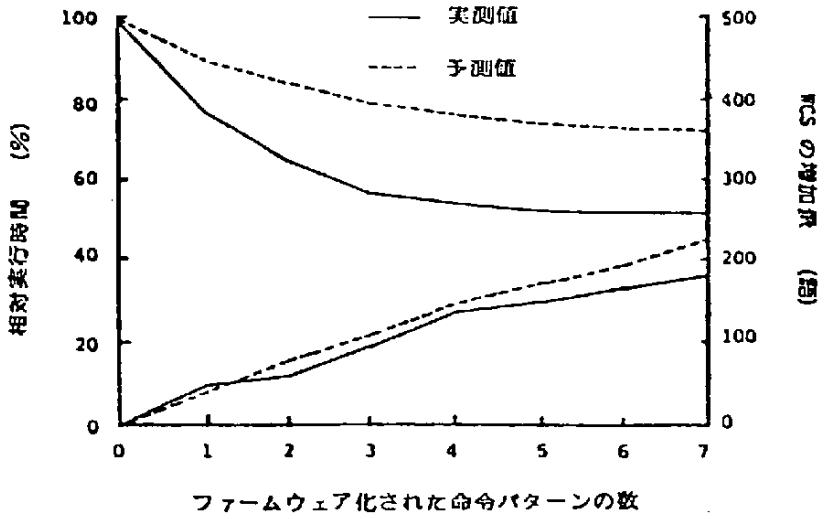
(a)　最大効果戦略（$\bar{u}$が最大である命令パターンを選択）



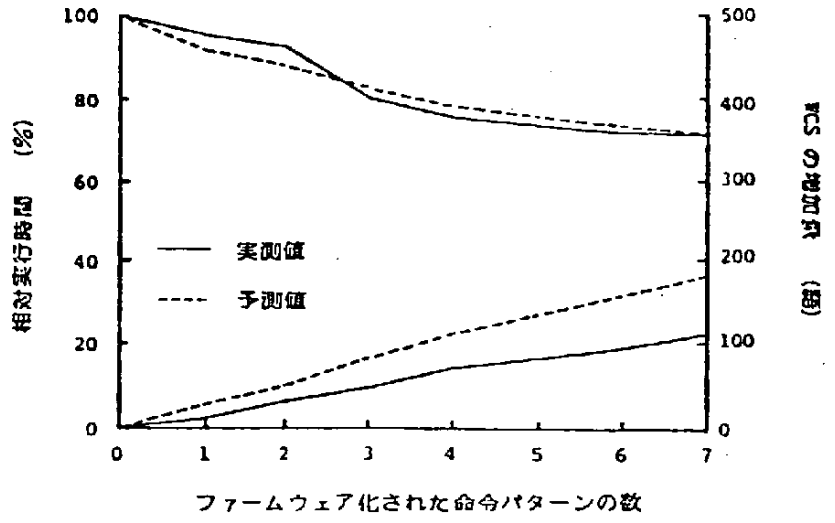
(b)　最大経済戦略（$\bar{u}/\theta$が最大であるパターンを選択）

図 4.20　　チューニングの結果

（HP－2100 PASCALマシンにおけるバブル・ソートの問題）

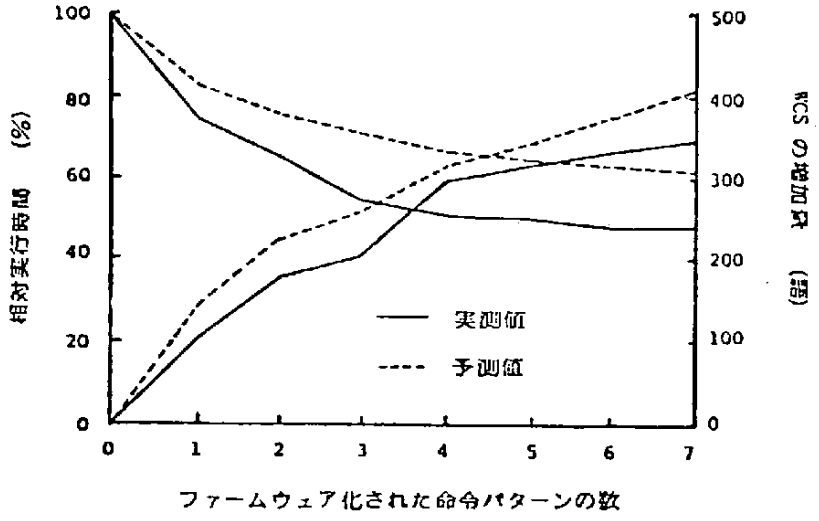


(a)　最大効果戦略（$\hat{u}$が最大である命令パターンを選択）

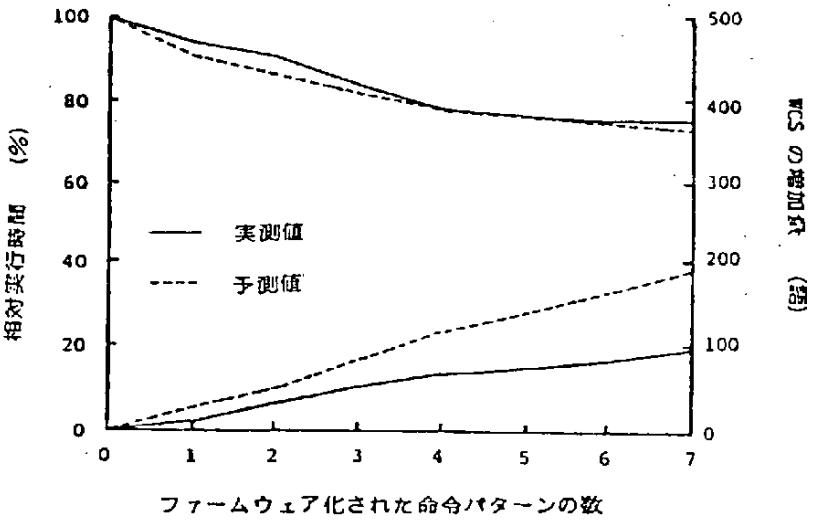


(b)　最大経済戦略（$\hat{u}/\hat{\theta}$が最大であるパターンを選択）

図 4.21　チューニングの結果
（B－1700 PASCALマシンにおけるバブル・ソートの問題）



(a) 最大効果戦略（$\bar{u}$が最大である命令パターンを選択）



(b) 最大経済戦略（$\bar{u}/\delta$が最大であるパターンを選択）

図 4.22　チューニングの結果
（B－1700 PASCALマシンにおけるマトリクス乗算の問題）

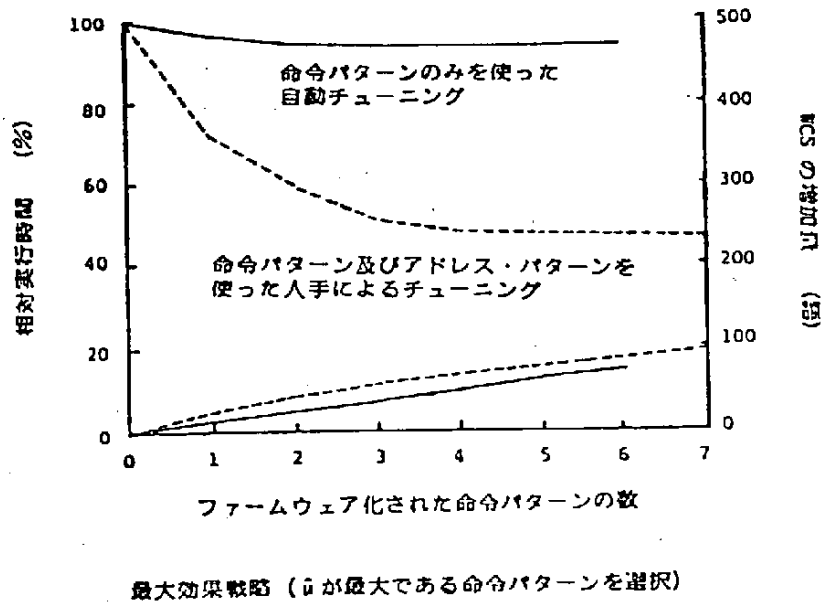


図 4.23　　チューニングの結果
（HP－2100 機械命令におけるバブル・ソートの問題）

選択した場合，WCS の容量は 130 語増加し，プログラムの実行時間は30％減少した。これから提案した計算機アーキテクチャの自動チューニングの原理が効率向上に効果があることが明らかになった。

＜2＞ 発生頻度の高い命令パターン

表 4.6 は最大実行効率戦略及び最大経済効果戦略でよく現われる命令パターンのリストである。第 1 の戦略で検出された命令パターンは，第 2 戦略の場合に比べ長い命令列になっている。長い命令パターンの多くは意味を付記してある。すなわち，例えばレジスタに配列要素の内容をロードする命令パターン等は解かれる問題の環境を示している。これらの命令パターンの特徴は，将来の高レベル言語向きの計算機のISPアーキテクチャの設計に大変重要な役割を果たすと思われる。他方，最大経済効果戦略で検出された短い命令パターンには，ほとんど重要な意味を持っていない。

表 4.6　発生頻度の高い命令パターン

（HP−2100　PASCALマシンにおけるバブル・ソートの問題）

最大効果戦略

No.	パターンNo.	命令パターン						意　味
1	88	LAO	LDO	CHK	DEC	IXA	IND	スタックに配列要素をロードする。
2	13	LDO	LOD	LEQ	FJP			二つの値を比較し、一方が他方より小さい、あるいは等しい場合分岐しない。
3	40	LDO	INC	SRO	UJP			ある変数を q だけ増やし分岐する。
4	70	LDO	SRO					
5	41	LAO	LDO	CHK	DEC	IXA		配列のアドレスを計算をする。
6	19	LDO	STO					
7	6	SRO	LDC	STR				

最大経済戦略

No.	パターンNo.	命令パターン			意　味
1	17	DEC	IXA		
2	15	LDO	CHK		
3	8	LDO	LOD		
4	31	LDO	INC	SRO	
5	10	LEQ	FJP		
6	70	LDO	SRO		
7	73	IND	SRO		

＜3＞チューニング曲線と学習

実験において得られたチューニング曲線を図4.24および図4.25に示す。実験の結果，チューニングの効果を予測する方法は，提案したアルゴリズムに有用であることがわかる。

4.5 動的環境情報による動的方法

4.5.1 数値計算におけるフォールト・トレラント・コンピューティング

本節では，動的環境情報による動的方法につき，精度落ちをアーキテクチャレベルで防止する方法を例に取って説明する。[15][16]

プログラマが遭遇する最もやっかいな問題の1つに，精度落ちがある。計算機では演算はたいてい固定長データを取扱うことを基礎としているので，プログラムの実行中に精度落ちや丸め誤差などの重大な誤りが発生し，その発生を予測することは非常にむずかしい。これは，この問題に対してはどんな理論的な解析も適用されないことを意味する。精度落ちによる上位桁の消失は，しばしば誤った結果をもたらす。たとえば，ガウスジョルダン法を用いる時，ある行の要素が一斉に有効桁を失うことがある。

図4.26に示すプログラムは基本アルゴリズムをそのままプログラムにしたものであるが，つねに正解をもたらすとは限らない。もし，桁を失った要素をピボットとして掃出しが行われると結果は誤ったものになる。このような好ましくない影響を抑えるために，絶対値が最大の要素をピボットとして掃き出しを行なうようにプログラムを書かなければならない。図4.27は枢軸選択法の例であるが，これは図4.26に示したプログラムの変形である。言いかえればユーザはつねにいくつかの「技法」を用いてプログラムを書くことを要求されている。それらのプログラムは簡単なアルゴリズムを用いた正直なものではなく，多くの非本質的なステップを含んだ複雑なものである。 計算機が計算分野で基本的な役割を果たしているにもかかわらず，数値計算に関

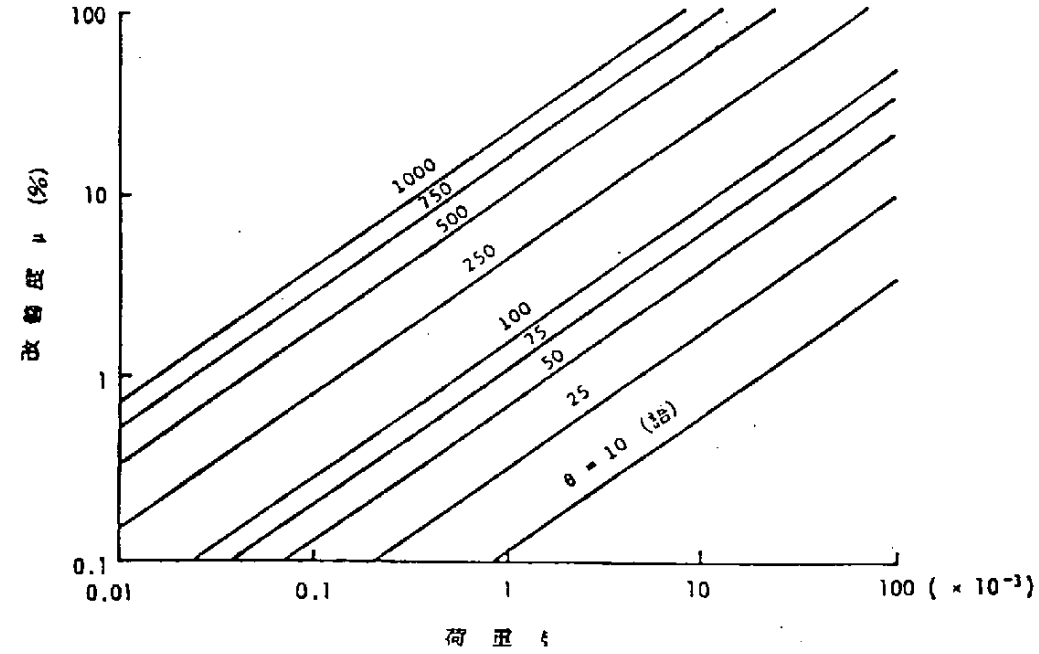


図 4. 24　　チューニング曲線
（HP−2100 PASCALマシン）

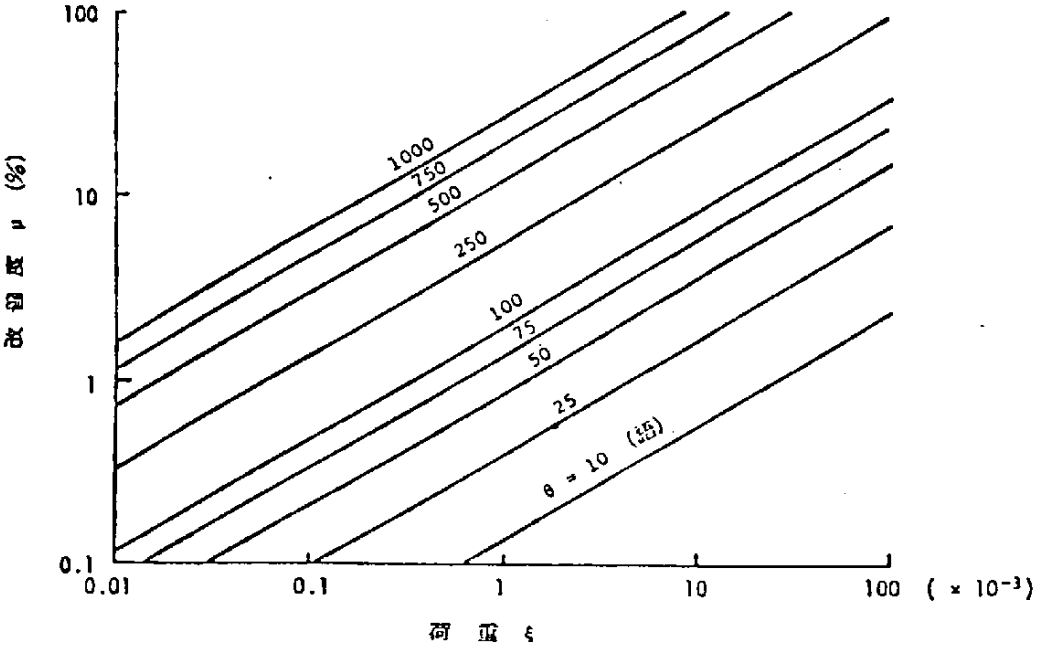


図 4. 25　　チューニング曲線
（B−1700 PASCALマシン）

```
                .
                .
                .
      N1 = N+1
C
      DO 100 K=1,N
      K1 = K+1
C
      DO 200 J=K1,N1
      A(K,J) = A(K,J)/A(K,K)
  200 CONTINUE
C
      DO 300 I=1,N
      IF(I .EQ. K) GO TO 300
      DO 400 J=K1,N1
      A(I,J) = A(I,J)-A(K,J)*A(I,K)
  400 CONTINUE
  300 CONTINUE
C
  100 CONTINUE
                .
                .
                .
```

図 4.26　ガウス・ジョルダン法による連立方程式の解法

して計算機アーキテクチャにおいて目立った進歩がないことは注目すべきである。精度落ちの発生という問題を計算機アーキテクチャで解決することは，ソフトウェアの生産性の向上にも本質的に貢献する可能性がある。

4.5.2　精度落ちを防ぐための自動復活機構

〔1〕 可能なアプローチ

精度落ちを復活させるには次の3つの方法が考えられる。

(i)　アルゴリズムはそのままで，実行する計算の順序を変える。

(ii)　アルゴリズムを改良する。

(iii)　プログラムの実行中に精度落ちを検出後，より大きい精度で再計算を行なう。

既存の計算機では演算は，単精度，倍精度のような段階的に指定される有限精度に基づいている。可変精度の計算ができる機構も，精度落ちを検出する機構も持たない。従って，(i)，(ii)の方法がユーザプログラムのレベルでのみ適用されている。

```
      .
      .
      .
      DO 10 I=1,N
       JNO(I) = I
   10 CONTINUE
C
      N1 = N+1
      DO 100 K=1,N
      K1 = K+1
C
C     SEARCH MAX. PIVOT
C
      IMAX = K
      JMAX = K
      AMAX = DABS(A(K,K))
C
      DO 20 I=K,N
      DO 30 J=K,N
      IF(DABS(A(I,J)) .LE. AMAX) GO TO 30
      AMAX = DABS(A(I,J))
      IMAX = I
      JMAX = J
   30 CONTINUE
   20 CONTINUE
C
      IF(IMAX .EQ. K) GO TO 50
      DO 40 J=K,N1
      AMAX = A(IMAX,J)
      A(IMAX,J) = A(K,J)
      A(K,J) = AMAX
   40 CONTINUE
C
   50 CONTINUE
      IF(JMAX .EQ. K) GO TO 70
      DO 60 I=1,N
      AMAX = A(I,JMAX)
      A(I,JMAX) = A(I,K)
      A(I,K) = AMAX
   60 CONTINUE
C
      I = JNO(K)
      JNO(K) = JNO(JMAX)
      JNO(JMAX) = I
C
   70 CONTINUE
C
      DO 200 J=K1,N1
      A(K,J) = A(K,J)/A(K,K)
  200 CONTINUE
C
      DO 300 I=1,N
      IF(I .EQ. K) GO TO 300
      DO 400 J=K1,N1
      A(I,J) = A(I,J)-A(K,J)*A(I,K)
  400 CONTINUE
  300 CONTINUE
C
  100 CONTINUE
      .
      .
      .
```

桁落ち影響を最小にするための付加部分

図 4.27　ガウス・ジョルダン法による連立方程式の解法
（完全枢軸選択）

これらの２つの方法をシステム・レベルで，復活機構を実現し，適用するためには，計算機システムの設計が進んだアルゴリズムに基づく必要がある。さらに，この機構を現在のハードウェアおよびソフトウェア技術で実現した場合にオーバヘッドは非常に大きくなる。そこで(iii)の方法が一番良いと思われる。

〔2〕 システムレベルでの問題解決

<1> 基本原理

精度落ちの発生を検出する機構を工夫すること，再計算を開始するための位置の決定，および開始位置へバックトラックするために計算機が準備する方法を見つけることが重要となる。

(1) 精度落ちの検出

加減算の結果の桁数が，あらかじめ決められた精度より小さいとき精度落ちが発生したとする。

(2) 再計算の開始位置

開始位置は，乗除算の行われた地点とする。それらの結果が精度落ちした加減算の演算項に用いられている。

(3) 再　計　算

再計算の手続きは要求される精度が得られるまで繰返される。

(4) バックトラックの準備

再計算を正しく行うために，精度落ちが発生した加減算と再開始位置の間で，変更された変数は元の値が戻されていなければならない。従って，変更される変数はプログラムの実行中に前もって保存される。もし，精度落ちが検出されると演算桁数は，精度落ちした分だけ増やされなければならない。

再計算のために必要な変更された変数が元に戻された後に，図4.28に一例を示すように再計算が開始位置から行われる。

再計算の開始位置をこのように選んだのは，以下の理由である。

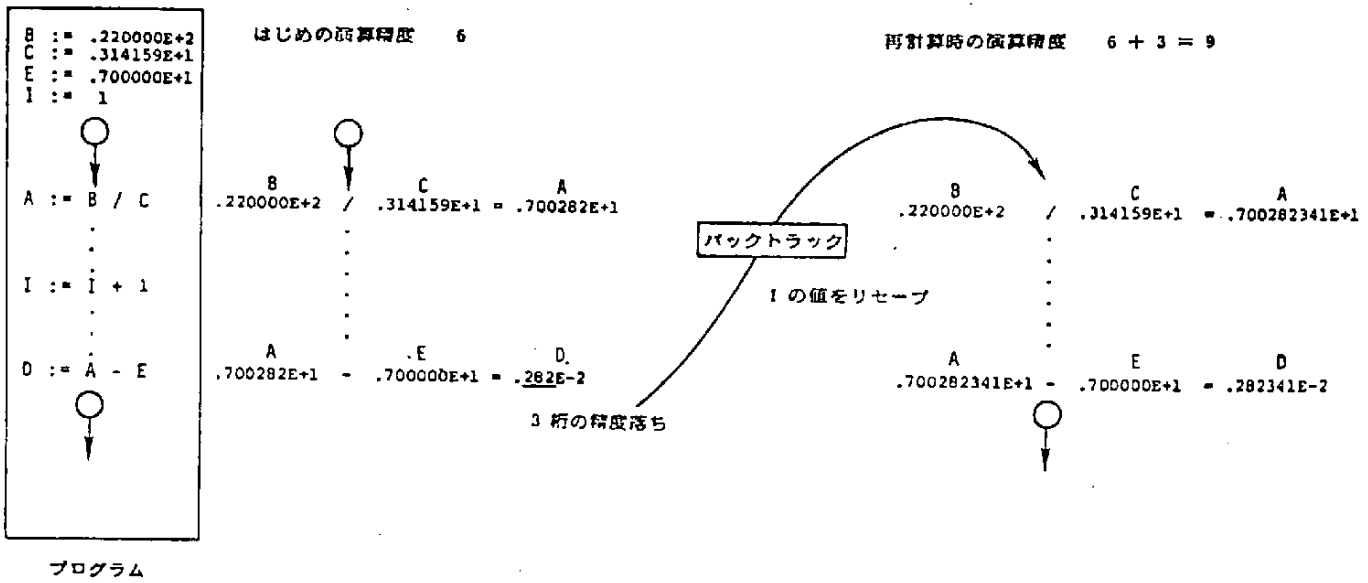


図 4.28　再計算による精度復活

○精度落ちが発生した加減算のオペランドが，それ以前により精度良く定義されていないと，精度落ちが復帰できない。

○計算機の演算の性質から，加減算あるいは単なる代入演算では，最下位桁の右側に有効な桁を補うことができない。

○乗除算の結果の桁数は，元のオペランドの桁数より長くなることができる。

上述の機構が実現されるためには，計算機システムは任意長のデータを扱い，任意精度の演算をサポートする基本アーキテクチャを持たなければならない。

＜2＞静的解析

再計算を開始するところの演算の位置，すなわち再開始位置は後述するように静的解析によって決定される。

再計算を正しく行なうためにプログラムの実行中に保存すべき変数もまた静的解析によって見い出される。プログラム内の各加減算に対し，再計算の再開始位置の候補が決定される。しかし，精度落ちを生じる加減算の2つのオペランドのうち，少なくとも一方が乗除算で定義されていなければ，再演算を行なうことができない。

オペランドを定義する乗除算がいくつかあるときは，再計算の開始位置として2つのオペランドの各々に対して，最後の乗除算が選ばれる。またプログラム中の各命令が図4.29に示すグラフの中で節で表わされるならば，上述の基本原理は(a)，(b)および(e)の場合はそのまま適用できる。

もし求めた演算が，(b)，(c)の場合のようにループ内にある時は，再開始位置はループの入口の位置に変更される。この場合は，たとえ計算機が一回だけ乗除算へバックトラックし，そこから再計算を開始しても丸め誤差がループ内で累積し，信頼ある結果が得られない。

再開始位置が決定された後，基本的原理に従って実行中に保存すべき変数が静的に定義される。しかし，プログラム中にループがあるときは

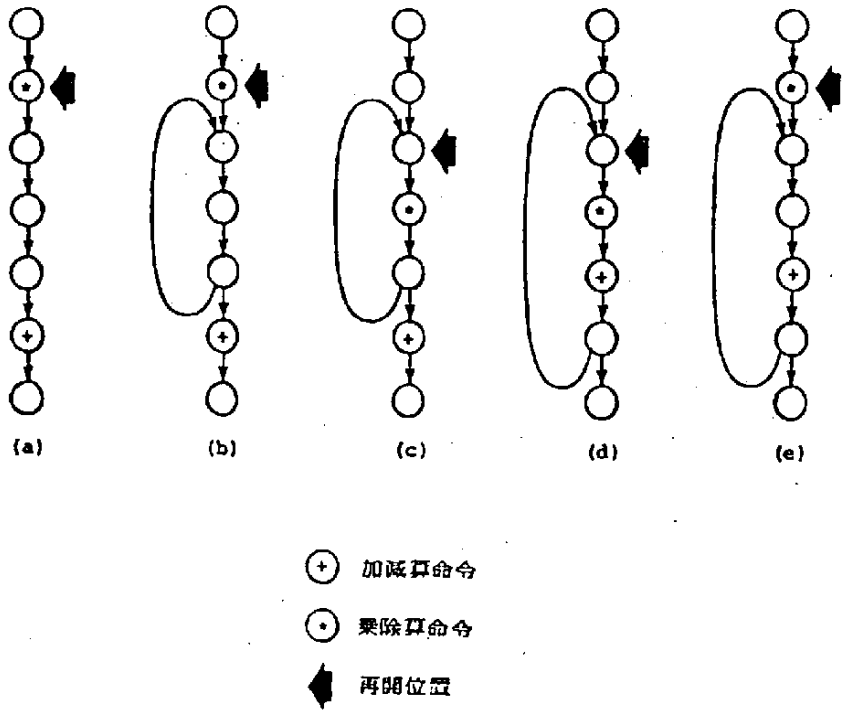


図 4.29　プログラムのグラフ・モデル

ループ内で定義され，かつ参照されるすべての変数は保存することとし，そのループ内の演算に対しては基本原理による解析を行わない。

<3> 変数の保存と再計算

静的解析は，再計算の再開始位置と保存すべき変数を決定する。これらのデータに基づいて計算機は命令の実行時に再計算のために必要な情報を保存しなければならない。そして，ある変数が乗除算によって陰あるいは陽に定義されるならば，定義された位置とその時の環境が実行中に保存される。

保存すべき変数は関係する命令あるいはループの入口の命令が実行される時に保存される。そして，精度落ちが検出されると，再開始位置が保存された情報から選ばれ，変数が再格納され，再開始位置へバックトラックが行なわれる。演算桁数は必要な桁数だけ増され，再計算開始の

準備ができる

4.5.3 精度落ちの自動復活機構を備えた仮想計算機のアーキテクチャ

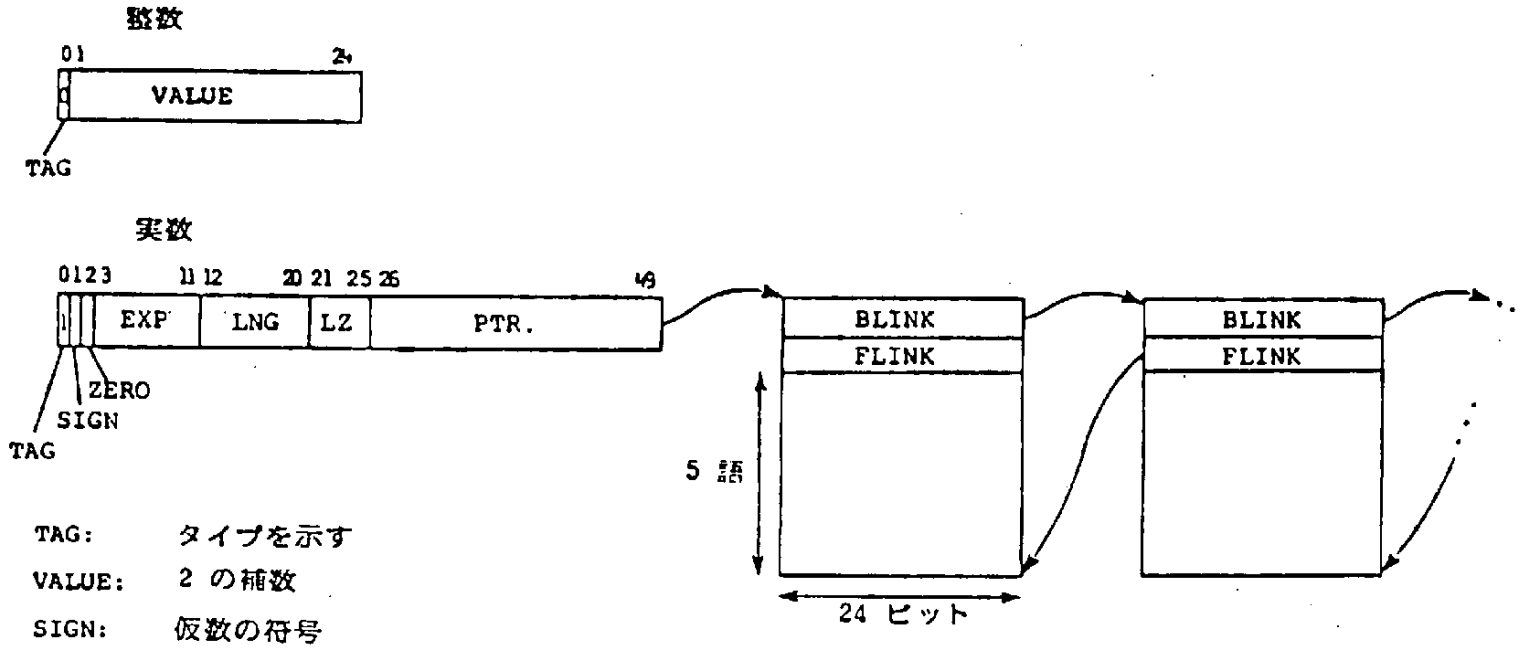
自動復活機構を使って精度落ちの発生を防ぐために，命令の実行中に演算桁数を制御できる機能を持たなければならない。この機能はマイクロプログラミング技術を利用して可能であるが，特にダイナミックマイクロプログラミングが重要な役割を果たす。

マイクロプログラムの計算機では，演算桁数がレジデュアル制御の一つとして指定されることがある。たとえば，B－1700のコントロールパラレルレングスレジスタは，ファンクションボックスのデータ長を指定する。演算桁数はレジデュアル制御の情報を新しい値に変えることによって制御することができる。この場合，プログラムの実行中にこれができなければならない。ダイナミックマイクロプログラミング技術が，プログラムの実行中にマイクロプログラムのシーケンスを全く変えることなく，レジデュアル制御情報の内容を変更することを可能にする。これはダイナミックマイクロプログラミングの本来の可能性によって計算機の動作をダイナミックに変えることを意味する。

〔1〕 データ構造

自動復活機構を用いて精度落ちを防ぐために，演算が可変精度で行なわれることが要求される。従って，実数データはデータ長を示す部分を持つ。データ長は演算中に変化するので，仮数部の位置は固定できない。そこで仮数部は指数部と分離される。指数部と仮数部は図4.30に示すようなリンクド・ブロック によって管理される。

長い桁数のデータの桁数を正規化するには時間がかかるので，仮数部のリーディングゼロの数を持たせ，正規化の必要をなくしている。さらに値のゼロを示すタグを設け，演算のスピードアップを図っている。1ビットのタグによって示された整数型のデータも用意している。



TAG: タイプを示す

VALUE: 2 の補数

SIGN: 仮数の符号

ZERO: ゼロを示す

LNG: 仮数のワード数

EXP: 2^{24} をベースとする指数

LZ: リーディングゼロのワード数

PTR: 仮数の先頭ブロックへのポインタ

BLINK: 後方ブロックへのポインタ

FLINK: 前方ブロックへのポインタ

図 4.30 データ構造

〔2〕 仮想計算機の動作と命令セット

仮想計算機の動作シーケンスを図4.31に示す。演算によって，そのデスティネーション・オペランドが実行前に保存されるかどうかチェックされる。このチェックは静的解析によってオペレーション・コード中にすでにセットされている情報によって成される。

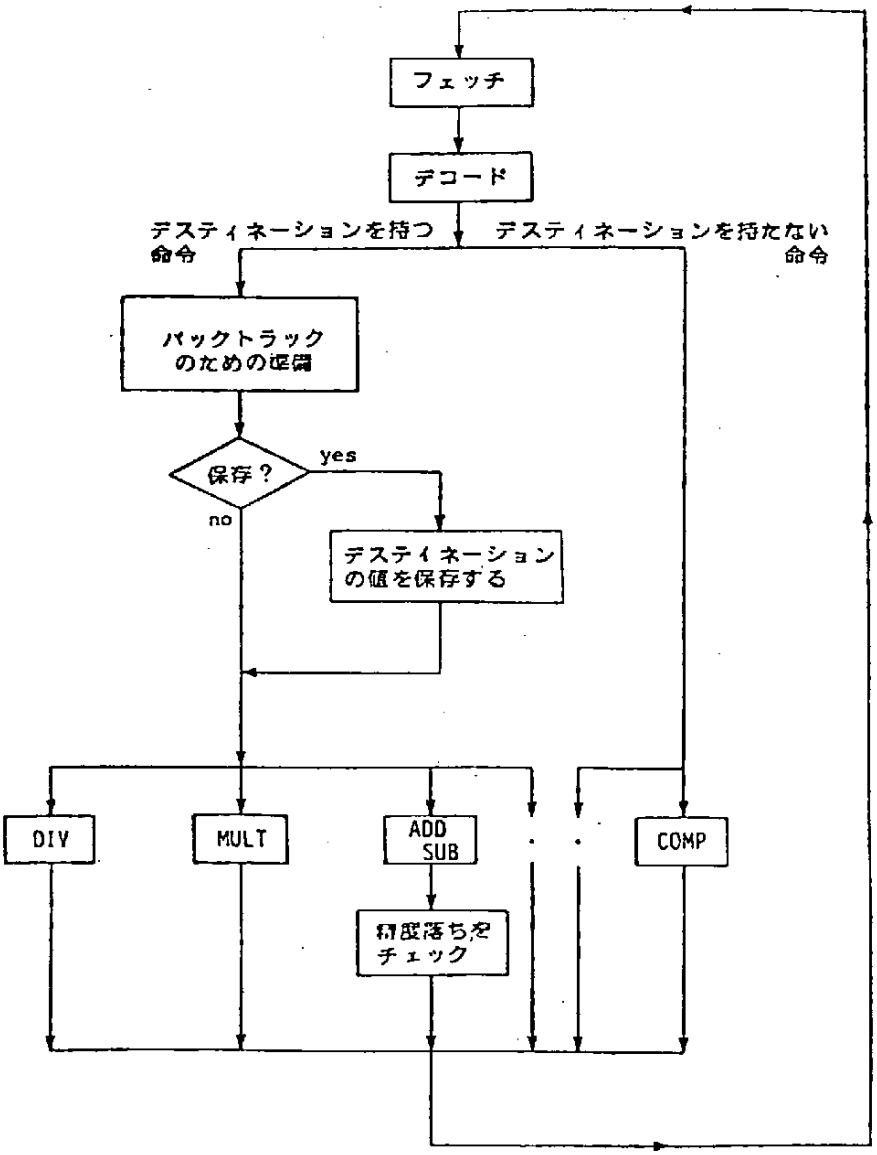


図4.31　仮想計算機の動作シーケンス

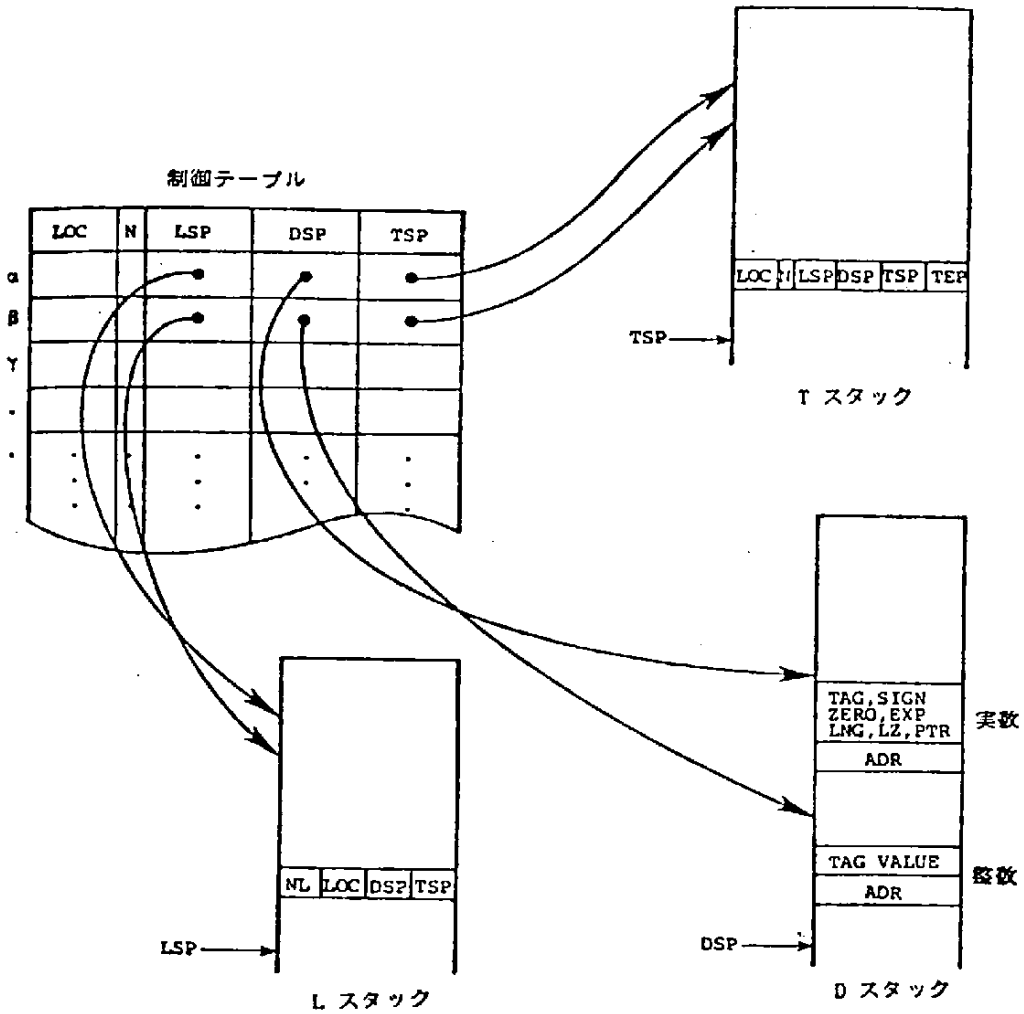
仮想計算機は，図4.32に示すように再計算のためにデータ・スタック（Dースタック），テーブル（Tースタック），ループ・スタック（Lースタック）と呼ばれる3つのスタックと1つの制御テーブルを持つ。

保存すべき変数は，Dースタックにプッシュされる。制御テーブルに関する前のデータは，Tースタックにプッシュされる。ループの入口の位置およびループに関する情報は，Lースタックにプッシュされる。制御テーブルは，定義された位置とその位置に関する情報の追跡する。

実際には，乗除算が実行されたとき，プログラムカウンタおよびD，L，Tースタックの各スタックポインタが制御テーブルのデスティネーションオペランドの欄に記憶される。現在のネストレベルがゼロならば，制御テーブルのNフィールドがゼロにリセットされる。命令が加減算あるいは代入演算ならば，制御テーブルのソース・オペランドの欄がデスティネーション・オペランドの欄に記憶される。精度落ちが検出されると，制御テーブルのソース・オペランドの欄が，再計算の再開始位置を得るために調べられる。

もし，ソース・オペランドの欄の内容が空ならば，再計算は不可能である。なぜならばソース・オペランドの値は乗除算によって定義されなかったからである。もし内容が空でないならば，制御テーブルに示された位置までDースタックおよびTースタックの内容が戻される。もしNフィールドがゼロでないならば，再計算の再開始位置およびDースタック，Tースタックの内容を戻す位置は，制御テーブルを使うかわりに，Lースタックに記憶された情報から得る。

次に，バックトラックが成された後，再開始位置がプログラム・カウンタにセットされる。演算桁数が，精度落ちした桁数だけ増やされ，再計算の開始の準備ができる。



LOC ： 乗除算命令あるいはループの入口の位置

N ： ループ指示子（LOC のネストレベルが 0 ならば 0）

LSP ： L スタックのポインタ

DSP ： D スタックのポインタ

TSP ： T スタックのポインタ

NL ： ループのネストレベル

ADR ： 保存された変数のアドレス

図 4.32　再計算用テーブルおよびスタック

4.5.4 実験と評価

〔1〕 バロースB1700での実験

前節で述べたように，計算機アーキテクチャレベルで精度落ちの発生を防止できるように設計された適応型計算機の仮想計算機をバロースB1700計算機システムでエミュレートした。

表4.7は仮想計算機をB－1700で実現した場合のマイクロプログラムの大きさを表している。大きさは約4k語であり，B1700上のFORTRAN マシンなどとは同じ大きさである。

提案した適応型計算機の性能を評価するために，比較対象として，バロース社がFORTRAN Sーランゲージインタプリタと呼ぶFORTRANマシンを使う。

表4.7 マイクロプログラムの大きさ

	Variable precision arithmetic	Recomutation control	others	total
MICRO INSTRUCTION COUNT	1,763	319	1,521	3,603

〔2〕 ベンチマーク問題

以下のベンチマーク問題が考えられた。

(i) 二次方程式の解法

(ii) 正弦関数のテーラー級数による決定

(iii) 係数行列にヒルベルト行列を持つ連立方程式の解法

これらの問題は，適応型計算機のためにFORTRAN形式の高級言語からコンパイルされた。B1700FORTRANマシンでは，問題はバロース社が提供するFORTRANで記述し，FORTRANコンパイラでオブジェクト・コードにされた。

表4.8はソース・コード，オブジェクト・コードおよび適応型計算機で必要なデータ・サイズを示している。適応型計算機では精度落ちの発生

を防ぐ必要がないのでソース・コードの大きさはB1700FORTRANに比べて半分以下である。このためにオブジェクト・コードもより小さくなっている。ソース・プログラム・レベルでのステップ数の減少はソフトウェアの生産性の改善を意味する。

１８桁以下のオペランドを持つ問題では，B1700FORTRANマシンの方が適応型計算機よりデータ・サイズがより小さいが，もしオペランドの長さが１８桁以上になると，データ・サイズに大きな違いはなくなる。

表4.8　解くべき問題のプログラムとデータの大きさ

	SIZE OF SOURCE PROGRAM (steps)				SIZE OF OBJECT PROGRAM (bytes)				SIZE OF DATA (bytes)				
	A	B		C	A	B		C	A	B		C	
		a	b			a	b			a	b	a	b
Adaptive computer	3	17	17	10	73	232	232	140	186	765	1,498	1,659	11,854
B-1700 FORTRAN machine	6	17	380	35	99	248	4,453	729	54	72	1,206	243	729

A　：二次方程式（18桁）
B-a：テーラー級数（18桁）
B-b：テーラー級数（32桁）
C-a：連立方程式（4元、18桁）
C-b：連立方程式（8元、18桁）

〔3〕実行時間と誤差

表4.9に実験の結果を示す。適応型計算機では，初期のオペランドの精度が十分ならば前もって決めた精度が得られている。しかし，従来のFORTRANマシンでは誤差が生じている。もし精度落ち防止がとられていなければ，さらに大きな誤差が生じる。

実行時間としては，適応型計算機での全処理時間，バックトラック，再計算およびその準備のオーバーヘッド時間を含んでいるにもかかわらず，FORTRANマシンより少ない。ユーザがソース・プログラム・レベルで可変精度にしないと，B−1700FORTRANマシンでは１８桁以上のオペランドを扱うことができないことに注目すべきである。例えば，正弦関数のテーラー級数を３２桁で計算しようとすると，B1700での実行時間は，

適応型計算機に比べて2,500倍以上の時間が要求される。これは，従来の計算機ではいかに非本質的なステップで多くの時間が費やされているかを示している。

表4.9　実　験　結　果

実行時間　(sec.)					
	A	B		C	
		a	b	a	b
Adaptive computer	0.0299	0.0128	0.0289	0.0413	0.3206
B-1700 FORTRAN machine	0.0316	0.0430	75.7	0.1433	0.9633
Time ratio	1 : 1.06	1 : 3.36	1 : 2619	1 : 3.47	1 : 3.00

相対誤差					
	A	B		C	
		a	b	a	b
Adaptive computer	0	0	0	0	0.19×10^{-11}
B-1700 FORTRAN machine	0	0.36×10^{-15}	0.63×10^{-25}	0.22×10^{-14}	0.26×10^{-7}

A　：二次方程式　(18桁)
B-a：テーラー級数　(18桁)
B-b：テーラー級数　(32桁)
C-a：連立方程式　(4元、18桁)
C-b：連立方程式　(8元、18桁)

表4.10は，再計算のオーバーヘッドを示している。オーバーヘッドは，全実行時間の10％以下である。実験のほとんどは，1回のバックトラックしか生じなかった。しかし，二次方程式とテーラー級数の計算では予期したほど多くのスペースを必要としなかったが，連立方程式の解法には，行列の要素が増すにつれて非常に多くのスペースが必要となった。表4.11はその様子を示したものである。

したがって，ここで提案している適応型計算機は代数方程式を解くような単一変数の計算により適しているといえる。

表4.10　再計算オーバヘッド

	A	B		C	
		a	b	a	b
$\frac{\text{overhead time} \times 100}{\text{total execution time}}$	1.75％	9.48％	5.56％	5.71％	5.30％

表4.11　再計算用スタック容量

	A	B		C	
		a	b	a	b
D-stack	0	35	35	2,214	23,522
T-stack	87	464	638	1,015	7,830
L-stack	10	10	10	210	730

A　：二次方程式（18桁）
B-a：テーラー級数（18桁）
B-b：テーラー級数（32桁）
C-a：連立方程式（4元、18桁）
C-b：連立方程式（8元、18桁）

第 5 章　結　　言

第5章　結　　言

本調査研究報告書は固定的な機能を備えた従来の汎用計算機と異なり，計算機アーキテクチャ・レベルで適応させていく，いわゆる問題適応型の可変構造計算機の実現可能性について，具体的な実験例を通して示したものである。本報告書に示す成果の多くは主として著者らの研究によるものであり，計算機の適応ならびに学習機能に関する一部を示したにすぎないが，将来の計算機に必須な機能であることが明確になったものと確信する。

計算機の適応ならびに学習機能，特に後者の学習機能については研究が緒についたばかりで，更にその可能性と効果について検討する必要があるが，これらの研究は今後急速に発展するものと予想される。第5世代の計算機においても，これらの機能は大きな特徴になるものと考えられるが，種々の観点から広い意味での適応性ならびに学習概念を追求する必要があるものと思われる。

謝　　辞

本調査報告書をまとめるに当って，ウォータル大学客員助教授 所 真理雄博士，電子技術総合研究所計算機方式研究室長 飯塚 肇博士ならびに慶大相磯研究室の学生諸氏には多大の御協力をいただいた。ここに深謝申し上げます。

また，日本情報処理開発協会開発部長 山本欣子 氏をはじめ開発部・技術調査部の諸氏に御支援をいただいたことを付記し，厚く御礼申し上げます。

昭和55年5月

慶応義塾大学工学部電気工学科　　相磯秀夫

東京大学理学部情報科学科　　坂村　健

参　考　文　献

(1)　相磯秀夫
〝ELM(ETL's Learning Machine)の提案〟
電総研　計算機方式研究室内部資料，（昭和４４年１１月）

(2)　坂村健，山田光一，相磯秀夫，飯塚肇
〝合成命令によるマイクロプログラム計算機の効率向上について〟
昭和50年度 情報処理学会第16回大会59

(3)　坂村健，飯塚肇，相磯秀夫
〝ダイナミックマイクロプログラミングによる最適命令セットの合成手法に関する考察〟
1975年9月 電子通信学会電子計算機研究会 EC75－28

(4)　坂村健，北越秀明，武田行雄，相磯秀夫
〝ダイナミックマイクロプログラミングによるデバッギングマシン－SNOOPY－〟
1976年6月 情報処理学会計算機アーキテクチャ研究会 CA21－2

(5)　坂村健，矢ヶ部一之，小花貞夫，相磯秀夫
〝マイクロプログラミングによる任意精度計算〟
昭和52年度 電子通信学会総合全国大会1253

(6)　Ichikawa, I., Sakamura, K. and Aiso, H.
〝ARES - A memory, capable of associating stored information through relevancy estimation〟
National Computer Conference AFIPS - Conference Proceedings
Vol.46, pp947-954 (1977)

(7)　Sakamura, K., Kitasusa, H., Takeyari, Y. and Aiso, H.
〝A DEBUGGING MACHINE - AN APPROACH TO AN ADAPTIVE COMPUTER〟
INFORMATION PROCESSING 77, Vol.7, pp23-28 (1977)

(8)　坂村健，北越秀明，武田行雄，相磯秀夫
〝デバッギング適応型計算機〟
電子通信学会論文誌　IECE'77／9 Vol.J60－D No.9

(9)　坂村健，相磯秀夫
〝計算機アーキテクチャの自動最適化に関する考察〟
電子通信学会論文誌　IECE'77／11 Vol.J60－D No.11

(10)　坂村健，相磯秀夫
〝A LEARNING MACHINE〟
1977年11月 情報処理学会計算機アーキテクチャ研究会 CA29－3

(11) 坂村健
・パネル討論
新らしい計算機アーキテクチャー非ノイマン機能
昭和52年度 情報処理学会第18会全国大会

(12) 坂村健，諸限立志，飯塚肇，相磯秀夫
〝ハードウェアモニタによるアーキテクチャチューニング
—Learning Machineにおける知識データベースの構成法—〃
1978年4月 電子通信学会電子計算機研究会 EC78—3

(13) Ichikawa, I., Sakamura, K. and Aiso, H.
〝A Multi-microprocessor ARES with associative processing capability on semantic data bases〃
National Computer Conference AFIPS - Conference Proceedings
Vol.47, pp1033-1039 (1978)

(14) 坂村健，諸限立志，飯塚肇，相磯秀夫
〝ファームウェア化による計算機性能改善度の予測〃
電子通信学会論文誌 IECE'78／9 Vol.J61—D No.9

(15) 坂村健，中野光一，加藤秀夫，相磯秀夫
〝数値計算適応型計算機のアーキテクチャ
—精度落ち自動防止機構を中心として—〃
1978年11月 電子通信学会，情報処理学会共催
電子計算機・アーキテクチャ研究会

(16) Sakamura, K., Nakano, K., Kats, Y. and Aiso, H.
〝A New Approach to An Adaptive Computer - An Automatic Recovery Mechanism To Prevent The Occurence of Subtract Errors〃
International Symposium on Computer Architecture(1979)

(17) Sakamura, K., Morokuma, T. and Aiso, H.
〝Automatic Tuning of Computer Architectures〃
National Computer Conference AFIPS—Conference Proceedings
Vol. 48, (1979)

(18) 坂村 健
〝問題適応型可変構造計算機に関する研究〃
慶應義塾大学学位論文，（昭和５３年度）

(19) Amdahl, G. M., Blaauw, G. A. and Brooks, Jr. F. P.
"Architecture of the IBM System/360"
IBM J. R&D., Vol.8, No.2, pp87-101, (1964)

(20) Bell, C. G. and Newell, A.
"The PMS and ISP descriptive systems for computer structures"
Proc. SJCC, pp351-374, (1970)

(21) Clark, W. A.
"Macromodular computer system"
Proc. SJCC, 30 pp335-336, (1967)

(22) Bell, C. G., Eggert, J. L., Grason, J. and Williams, P.
"The description and use of Register-Transfer modules(RTM's)"
IEEE Trans. Comput. C-21 No.5, pp495-500, (1972)

(23) Bell, C. G. and Grason J.
"The Register-Transfer Modules design concept"
Computer Design, Vol.10, No.5, pp87-94, (1971)

(24) Swan, R. J., Fuller, S. H. and Siewiorek, D. P.
"CM*-A modular, multi-microprocessor"
Proc. NCC, 46, pp637-644, (1977)

(25) Iizuka, H., Ishii, O., Fujii, K., Ohomoto, R. and Furuya, T.
"ACE - A new modular computer architecture"
Proc. US-J Computer conf., 2, pp36-41, (1975)

(26) Husson, S. S.
"Microprogramming - principles and practices"
Prentice-Hall (1970)

(27) Wilkes, M. V.
"The best way to design an automatic calculating"
Computer Inaugral conference, pp16-18, (1951-07)

(28) Tucker, S. G.
"Microprogram control for System/360"
IBM Syst. Jour., Vol.6, No.4, pp222-241, (1967)

(29) Rosin, R. F.
"Contemporary concepts of microprogramming and emulation"
Computing Surveys, Vol.1, No.4, pp197-212, (1969)

(30) Weber, H.
"A microprogrammed implementation of EULER on IBM System/360 Model 30"
CACM, Vol.10, No.9, pp549-558

(31) Opler, A.
"Fourth generation Software"
Datamation, Vol.13, No.1, pp22-24, (1967)

(32) Rakoczi, L. L.
"The computer-within-a-computer. A fourth generation concept"
IEEE Computer group news, Vol.2, No.8, pp14-20, (1969)

(33) Flynn, M. J. and MacLaren, M. D.
"Microprogramming revisited"
Proc. ACM 22nd National Conf., pp457-464, (1967)

(34) Reigel, E. W. and Lawson, H. W.
"At the Programming Language - Microprogramming interface"
SIGPLAN-SIGMICRO interface meeting, pp2-22, (1973)

(35) Lesser, V. R.
"Dynamic control structure and their use in emulation"
SLAC-127 Stanford University, (1972)

(36) Wilner, W. T.
"Design of the Burroughs B1700"
Proc. FJCC, pp489-497, (1972)

(37) Lawson, H. W. and Smith, B. K.
"Functional characteristics of a multilingual processor"
IEEE Trans. C., 20-7, pp732-742, (1972)

(38) Rosin, R. F., Frieder, G. and Eckhouse, R. H.
"An environment for research in microprogramming and emulation"
CACM, 15-8, pp748-760, (1972)

(39) Davis, R. L., Zucker, S. and Cambell, C. M.
"A building block approach to multiprocessing"
Proc. SJCC, 40, pp685-703, (1972)

(40) Agrawala, A. K.
"Foundation of Microprogramming Architecture,
Software and applications"
Academic Press, (1967)

(41) Knuth, D. E.
"An empirical study of FORTRAN programs"
Software-practice and experience, Vol.1, pp105-133, (1974)

(42) Darden, S. C. and Heller, S. B.
"Streamline your software development"
Computer Decisions 2, pp29-33, (1970)

(43) Saal, H. J. and Shustek, L. J.
"Microprogrammed Implementation of Computer Measurement Technique"
MICRO 5 ACM, pp42-50, (1972)

(44) Chu, Y.
"High level language computer architecture"
Academic Press, (1975)

(45) Hassit, A. and Lageschulte, J. W.
"Implementation of a high-level language machine"
CACM, Vol.16, pp199-212, (1973)

(46) 島田俊夫，山口喜教，坂村健
"LISPマシンとその評価"
信学誌 Vol.59−D, No.6, pp406−413, (1976)

(47) Tucker, A. B. and Flynn, M. J.
"Dynamic microprogramming processor organization and programming"
CACM, Vol.14, No.4, pp240-250, (1971)

(48) 飯塚肇
"マイクロプログラム化による効率向上について"
情報処理学会 ダイナミックマイクロプログラミングシンポジウム報告集
pp153−159, (1973)

(49) Rustin, R.
"Debugging techniques in large systems"
Prentice-Hall, pp58, (1970)

(50) Kulsrud, H. E.
"Extending the interactive debugging system HELPER"
Debugging techniques in large systems, pp77, (1970)

(51) Blair, J.
"Extendable non-interactive Debugging"
Debugging techniques in large systems, pp93, (1970)

(52) Gasser, M.
"An interactive debugger for software and firmware"
Preprints of the 6th annual workshop on microprogramming, pp113, (1973)

(53) Sager, G. R.
"Emulation for system measurement/debugging"
Preprints of IFIP TC.2, pp89, (1975)

(54) 宇都宮公訓
"マイクロプログラミングのデバッギングへの応用"
情報処理学会 ダイナミックマイクロプログラミングシンポジウム報告集 (1973)

(55) Synder, D. C.
"Computer Performance Improvement by Measurement and Microprogramming"
Hewlett Packard Journal, February 2, (1975)

(56) Abd-Alla, A. M. and Karlgaard, D. C.
"Heuristic Synthesis of microprogrammed computer architecture"
IEEE Trans. Comput. C-23, 3, pp802-807, (1974)

(57) Nori, K. V. and Ammann, U.
"The PASCAL 'P'compiler: Implementation Notes"
Institut fur Informatik, Eidenossiche Technische Hoehschule,
Zurich Report No.10

(58) Forsythe, G. E.
"Pitfalls in Computation, or why a Math Book Isn't Enough"
The American Mathematics Monthly, Vol.77, pp931-956, (1970)

(59) Knuth, D. E.
"THE ART OF COMPUTER PROGRAMMING, 1, Fundamental Algorithms"
ADDISON-WESLEY PUBLISHING COMPANY, pp251, (1975)

昭 和 55 年 5 月 発 行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園3－5－8
機 械 振 興 会 館 内
T E L　(434) 8211(大代表)

印刷所　山　陽　株　式　会　社
東京都港区虎ノ門1－9－5
T E L　(591) 0248