解　説

2. マイクロプログラマブル・アーキテクチャ

2.2 マイクロプロセッサにおけるマイクロプログラミング技術†

飯塚　肇††　緑川博子††

1. はじめに

“マイクロプロセッサ”と“マイクロプログラム”，どちらも同じ“マイクロ”であっても実は直接の関係はない．マイクロプロセッサにおける“マイクロ”とは，もちろんサイズが小さいとか，機能が小さい（現時点ではそうは言えないが）とかを意味しているのに対し，マイクロプログラムにおける“マイクロ”とは，プログラムに使用する命令の機能レベルが低く，ハードウェアに直結したものであることを意味する．したがって，マイクロプログラムをマイクロプロセッサに適用する際，一般の計算機の場合にくらべて格段の相違があるわけではない．そこで，本稿では話題を少し広げ，マイクロプロセッサにおける命令セットの設計とインプリメンテーション，及びマイクロプログラマブル マイクロプロセッサの応用について述べる．また，マイクロプログラムによる CISC（高機能命令セットコンピュータ）に対抗する方式として話題の RISC（単純機能命令セットコンピュータ）についても両者を対比しながら若干述べてみたい．ただし，網羅的サーベイではなく，基本的考え方と代表的例を中心とする．

2. 命令セット設計とマイクロプログラム

2.1 マイクロプログラム制御の効用

計算機の命令セットは，ユーザ側からみれば使いやすいこと（コンパクトで実行効率の良いプログラムを製作しやすいこと），製造側からみればインプリメントしやすいこと（高速実行できるハードウェアを低コストで設計できること）が基本である．よく知られているようにマイクロプログラミング技術は比較的高機能の命令でも安価に実現する手法として開発された．非常に単純なプロセッサの場合を除きマイクロプログラム制御の有効性は一般の計算機においてこれまでに良く立証されている．

マイクロプロセッサにおけるマイクロプログラミングの利用も同じ目的で1970年代の後半に16ビットマシンにおいて始まった．ランダム論理に対するメモリの規則構造性はディスクリート素子の場合にも増して有効と考えられたし，後で制御記憶の内容を変えて制御論理を変えられる柔軟性も設計変更の困難なマイクロプロセッサでは特に有効である．また，部品としてのマイクロプロセッサを各システムのニーズに適合させる手段としてもマイクロプログラミングは効果的であった．その典型的例はビットスライスマイクロプロセッサである．

2.2 RISC 方式

上記のマイクロプログラム制御の基本は，ハードウェアに直結した単純な命令によって比較的高機能の機械命令をインタプリート実行することにある．しかし，インタプリートは本来低速な操作であって高速実行には向かない．マイクロ命令を用いて直接プログラムを作成すれば高速実行が可能であるが，従来それはプログラミングが困難であり，効率の良いコードを生成するコンパイラも実現不可能とされてきた．

これに対して，マイクロ命令を直接プログラミング向きに設計し，最新のコンパイラ技術を利用すればその最適化コンパイラの作成が可能となるというのが RISC の立場である（図-1）．すなわち，RISC はその名(Reduced Instruction Set Computer) の示すとおり，高速実行のためのインプリメントが可能なマイクロ命令同様の単純な命令セットを有するコンピュータである．ここで高速実行のためのインプリメントとは，具体的にいうと，たとえば分岐命令において図-2に示すように次の命令を1個実行してから分岐することで命令実行パイプラインの乱れを少なくすること（遅延分岐），命令実行のクリティカルパスを分析して

† Microprogramming Technologies for Microprocessors by Hajime IIZUKA and Hiroko MIDORIKAWA (Department of Information Sciences, Seikei University).

†† 成蹊大学工学部

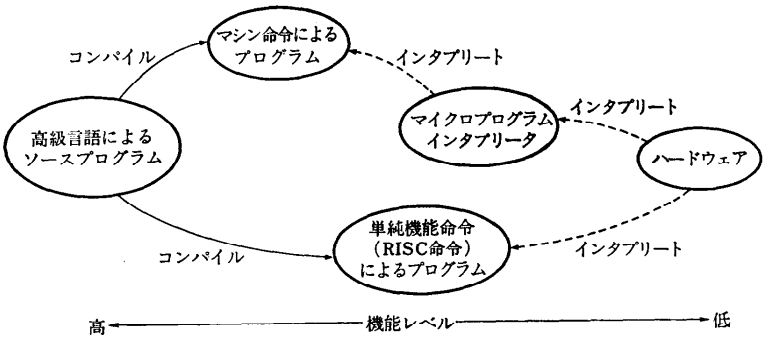

図-1　マイクロプログラム制御と RISC 方式の位置づけ

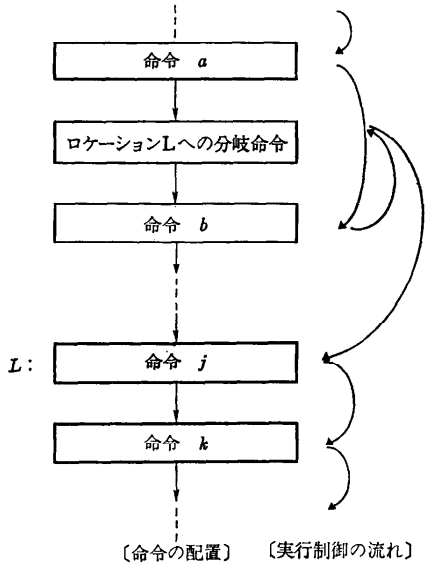

図-2　遅延分岐

マシンサイクルを長くするような命令を排除することなどの方策である.

この結果, RISC では命令制御論理が簡単になるので素子量の制限が厳しいマイクロプロセッサでは非常に好都合であることは言うまでもなかろう（通常チップ内の制御用領域は 10% 以下. ちなみに MC 68020 では 68%）. さらに, 命令制御論理の簡単化によって空いた面積は大容量のレジスタファイルとして用いられるので, チップ外とのデータ転送量も減少し高速化に寄与させられる.

RISC は命令のインタプリートを前提とした本来の意味でのマイクロプログラム制御ではないが, マイクロプロセッサにおいてマイクロプログラムと同様の目的をもった重要な技術といえる.

3. マイクロプロセッサにおけるマイクロプログラム技術の特徴と種類

3.1 特　徴

マイクロプロセッサのアーキテクチャ設計に影響を与える主な特性は次の 4 点である.

a. [素子数の制限] 1チップに集積できる素子数は初期のマイクロプロセッサにくらべて最近は著しく増大したが, 高性能なマイクロプロセッサはその実現に多くの素子を要し, 素子数, 特に, ランダムロジックの量をできるだけ減らすことは現時点でも大きな設計目標である.

b. [ピン数の制限] パッケージング技術の進歩によってマイクロプロセッサ用パッケージのピン数もかなり増大したが, 集積度の向上に比べれば増加度ははるかに小さい. ピン数の増加は信頼度の減少や消費電力の増加を招く可能性もあり, プロセッサの性能向上に見合ったインタフェースピン数の向上は望めない.

c. [チップ外とチップ内通信速度の差が大きい] したがって, チップ外との通信量を減らせるアーキテクチャが必要である.

d. [汎用性に対する要求大] マイクロプロセッサのアーキテクチャには, 設計側からのみならず応用側からも一般計算機に対するよりずっと大きい汎用性と柔軟性が求められる.

マイクロプロセッサにおいて, マイクロプログラミング技術はこれらの要求を満たす有効な手段として用いられているが, 同時にその適用はこれらの条件にそったものでなければならない.

3.2 種　類

以下では, マイクロプロセッサにおけるマイクロプログラミング技術の適用形式を, 大きく従来型のものと RISC 型に分けた上で, それぞれについていくつかの視点でその特徴を上記要請に対応させて論じる.

3.2.1 インタプリート式マイクロプログラム制御

（1） 目的による分類

第一は，比較的機能レベルの高い ISP（機械）命令セットを実現するための古典的利用形式である．通常マイクロプログラムはチップ内の ROM に内蔵され，要件 a を達成するために2レベル制御などの工夫が行われている．また，要件 d のために外部からマイクロプログラムを供給する例や，部分的に書き換え可能な制御記憶（WCS）を利用するものもある．すでに確立した技術になっていることとマイクロプログラムの内容は企業秘密的な面もあるので詳細に述べた文献は少ないが，後に2，3の代表的例について述べる．

第二は，要件 d を主たるねらいとするものである．マイクロプロセッサは一般の計算機と異なり，汎用部品的色彩が強い．この点を表に出す場合には機能の制御性を高める必要があり，ハードウェア細部まで制御できるマイクロプログラミングは好都合である．その典型的例はいわゆるビットスライスマイクロプロセッサであるが，集積度の増加を利用したビットスライスではないものもある．この型の場合は，その目的からしてマイクロプログラムは普通チップ外から供給される．また，マイクロプログラムはハードウェアを熟知した設計者以外によって作成されることになるので，マイクロ命令は比較的わかりやすくプログラミングしやすいものとなっている．

（2） マイクロ命令供給方式による分類

チップ内蔵とチップ外からの供給の二方式のほかそれらの中間に位置するいくつかの変形が考えられる．要件 d の制御性の点からは外部供給方式が明らかに有利である．また，要件 a からもマイクロプログラム量が大きい場合は全マイクロプログラムのチップ内蔵は困難になる．しかし，要件 b のために長語長のマイクロ命令をチップ外から供給して高速実行することも実現が困難である．そこで，チップ内制御記憶の一部を WCS とし最初にマイクロプログラムをロードしておいて実行する方法や，チップ内にマイクロプログラム用キャッシュをもつ方法なども考えられている．

（3） マイクロ命令形式による分類

マイクロ命令形式は制御性の点からは水平型が良く，プログラミング性と制御記憶量からは垂直型が優れていることは良く知られている．しかし，前記のようにチップ外からの供給の場合は要件 b と c のために水平型が用いられることはない．チップ内蔵の場合でも制御記憶量を減らすために2レベル型マイクロプログラムが使用されることがある．

この2レベル型のハードウェア素子量についての利点は次のように論じられる．今，1レベルの水平型で実現したときの制御記憶の構成を図-3（a）とする．ここで，次の命令アドレスフィールドの長さは $a=\lceil \log_2 m \rceil$ であるから制御記憶量は，

$$M_1=m(c+\lceil \log_2 m \rceil)$$

と表せる．一方，垂直型のマイクロ命令で水平型のナノ命令を間接的に呼び出す場合は，図-3（b）に示すような構成となる．この場合，$b=\lceil \log_2 n \rceil$ となるから，制御記憶の総量は

$$M_2=m(a+b)+nc=m(\lceil \log_2 n \rceil+\lceil \log_2 m \rceil)+nc$$

となる．ナノ命令の重複度が大きく，n/m が小さければ $M_1 \gg M_2$ となって2レベル型の効果が表れる．この具体的例は後に述べる．

一方2レベル型の欠点としては，1マイクロ命令の実行にマイクロ記憶とナノ記憶を逐次的にアクセスしなければならないために通常マシンサイクルが長くなるという問題がある．したがって，2レベル型ではこの遅延を減らすように命令のプリフェッチや並行アクセスなどを用いてマシンサイクルの短縮に十分注意する必要がある．

3.2.2 RISC 型制御

RISC においては，普通チップ外部から単純な命令列を供給する．命令サイクルが短いのでチップ外に命令キャッシュを用意する場合もあるが，チップ内の基本構成は本質的にはあまり変わらない．ただし，マイ

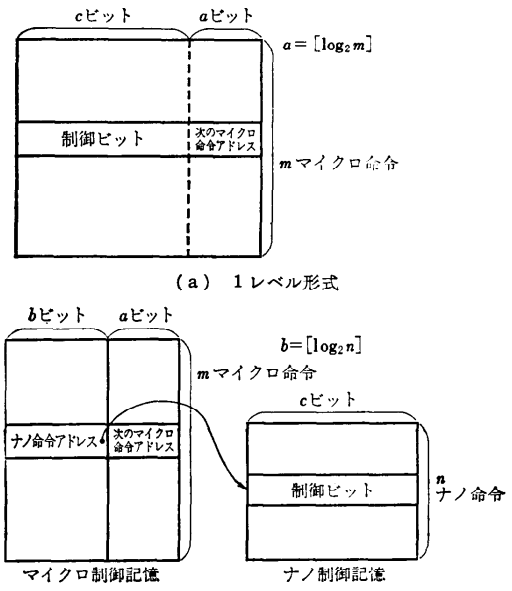

図-3 制御記憶の構成

クロコード化 RISC と呼ばれる次のような変形も提案されているが（ジョージメイソン大の MIRIS)[2),11)] いまだチップ化はされていない.

マイクロコード化 RISC は，いわば前記2レベルマイクロプログラム制御方式において機械命令のレベルを除いたもので，1マシンサイクルで実行される垂直型マイクロ命令（これが RISC 命令に相当し，MIRIS では 32 ビット）によってプログラムする（またはコンパイラがそのコードを生成する）．ただ，一般の RISC では命令の実行制御はハードワイアドなのに対し，この場合はマイクロ命令と1対1に対応するナノ命令（MIRIS では 64 ビット）によって実行される．こうすることによってマイクロプログラム制御の柔軟性を RISC に与えることをねらいとしている．

4. アーキテクチャの実例

以下では，代表的なマイクログラム制御マイクロプロセッサと RISC マイクロプロセッサについてそのアーキテクチャを述べる．

4.1 マシン命令のインプリメンテーション

（1） モトローラ MC 68000[1)]

マイクロプロセッサへのマイクロプログラム制御の導入は，ビットスライス マイクロプロセッサを別とすれば事実上 16 ビットプロセッサから始まった．MC 68000 は初期のマシン命令インプリメンテーションのためのマイクロプログラム制御マイクロプロセッサの代表的チップである．マイクロプログラム制御を採用したために不採用の Z 8000 などに比べて開発期間が短縮され，それが市場で有利な地位を占めた一因になったといわれている．

図-4 に MC 68000 の制御構造を示す．図から分かるようにチップに内蔵されたマイクロプログラムは **3.2.1** で述べた2レベル方式を採用している．制御語の長さ（c）は 70 ビット必要で，マイクロ命令数は約 650 語，異なる制御語の数 (n) は約 280 である．したがって，n/m は約 0.4 となり，**3.2.1** に示した式に値を入れてみれば分かるように2レベル方式の採用によって制御ビットの数は1レベルの場合に比べ，50% 強ですんでいる．

（2） LSI-11[3)] と MicroVax[4)]

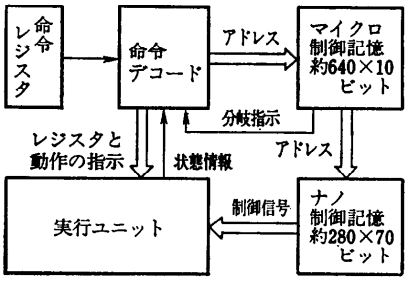

図-4 MC 68000 の制御構造

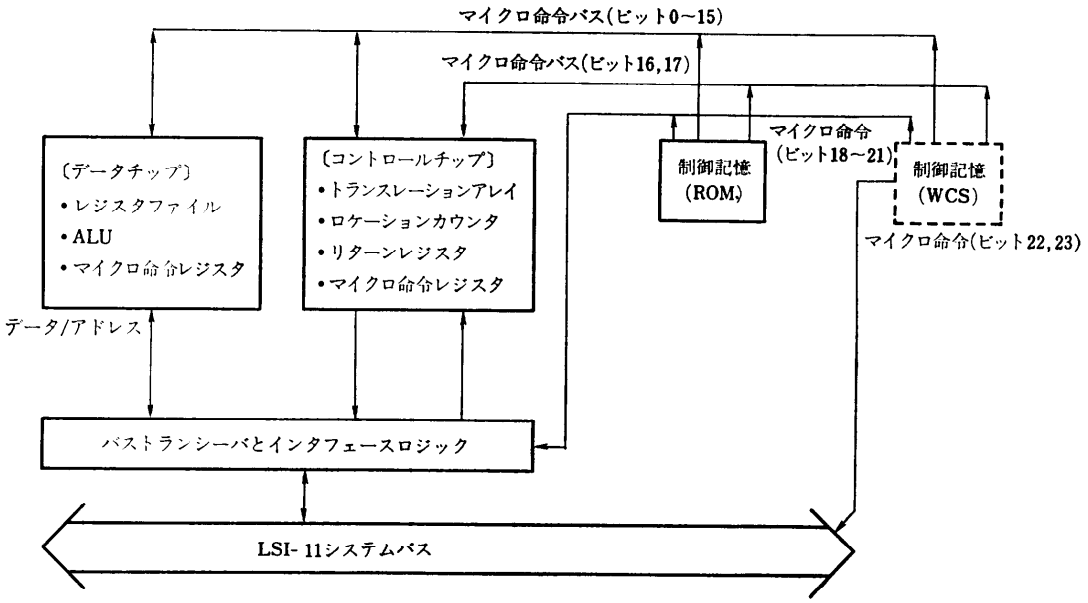

図-5 LSI-11 の制御構造

LSI-11 は PDP-11 を，また MicroVax は Vax-11 をマイクロプロセッサ化したもので，前者は 1970 年代後半に，後者は 1980 年代になって開発された．機能的には大きな違いがあるが，いずれもチップ外部からマイクロ命令を供給する形式のマイクロプログラム制御方式を採用している．この間の進歩をみるために両者を対比しながらマイクロプログラム制御機構をみてみよう．

LSI-11 は，図-5 に示すようにデータチップとコントロールチップに対し外部の ROM（基本命令用 1 k 語，2 k 語まで拡張可能）または WCS から読み出したマイクロ命令を供給する．マイクロ命令長は 22 ビットであるが，うち 16 ビットが垂直型マイクロ命令を構成し，データチップとコントロールチップに共通に与えられる．残り 6 ビットのうち 2 ビットはコントロールチップのみに加えられ，4 ビットはチップ外でバストランシーバなどの制御に用いられる．コントロールチップ内に各マシン命令に対するマイクロルーチンの先頭アドレスをハードウェア的に発生するトランスレーションアレイをもったごく標準的な構成になっている．

一方，MicroVax は図-6 に示すように 3 個の本体チップに制御記憶から 39 ビット（及び 1 パリティビット）の長さのマイクロ命令を供給する．制御記憶は 5 個の同一のチップから成るが，各チップは図-7 に示すようなパッチャブルチップと呼ばれる ROM, RAM, CAM の混在するユニークなものである．マイクロプログラムは 16 k 語のマスク ROM 内に格納されているが，後から WCS にパッチプログラムを入れることができるようになっている．図から分かるように ROM と RAM は 8 ビットスライスであるが，CAM は各チップに 32 ロケーションあって，合計 160 のパッチアドレスを記憶できる．

パッチ動作は以下のとおりである．図において選択信号（MS）は通常 0 になっていて ROM から読み出されたデータがマイクロ命令として出力されている．今，CAM のいずれかの語がマイクロアドレスとマッチしたとすると，そのチップのマッチ信号（MA）は 1 となり，これは他のチップの MA と OR を取られて本体チップのマイクロシーケンサに与えられる．すると，マイクロシーケンサはこの ROM から読み出されたマイクロ命令をアボートして，今度は MS 信号を 1 にして前と同じマイクロアドレスを読み出す．以後は分岐命令によって再び ROM に戻るまでマ

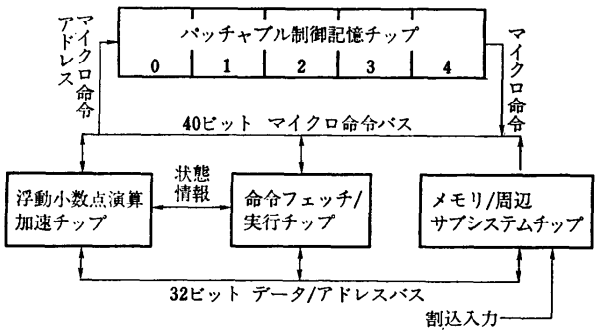

図-6　MicroVax のブロック図

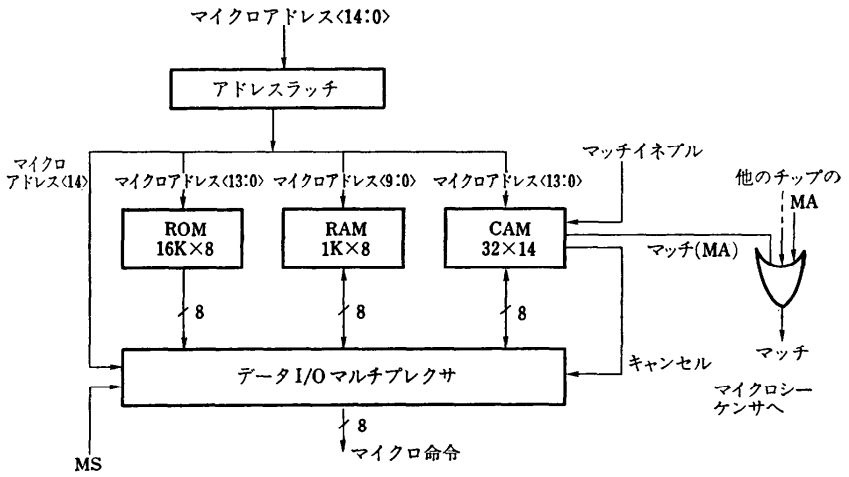

図-7　パッチャブル制御記憶の構造

イクロ命令は RAM から読み出される．

なお，WCS のアクセスにはマイクロアドレスの下位 10 ビットをそのまま用いているのでパッチアドレスがコンフリクトを起こす可能性があるが，現実には複数個のアドレスがパッチアドレスの候補として利用できる場合が多く，ほとんど問題とはなっていない．

最近の超高集積 LSI では設計誤りの修正はきわめて高価につく．上記の機構はマイクロプログラム制御機構の本質にかかわるものではないが，商用マイクロプロセッサのためのテクニックとしては PROM を用いるものより優れ，実用的価値はあると思われる．

(3) 32 ビットマイクロプロセッサ

標準的な 32 ビットマイクロプロセッサはチップ内に制御用 ROM を内蔵したマイクロプログラム制御方式を採用している．すでに確立した技術なのでそれを詳しく論じた文献は見当たらないが，2，3の例を簡単に紹介する．

インテル 80386[5] では命令プリフェッチユニットがフェッチしてキューにいれたマシン命令コードを命令デコードユニットが取り出していったん幅の広い命令に解読し，キューに入れる．この解読ずみ命令中にはマイクロコードの先頭アドレスも格納されているので実行ユニットはこれを取り出してマイクロルーチンを起動実行する．マイクロ命令の実行は2クロックを要するもののマイクロ命令フェッチと実行をオーバラップさせ，かつ遅延分岐を採用することで実質マイクロサイクル当たり1クロック (16 MHz のとき 62.5 ns) しかかからない．

32 ビットチップでは本体プロセッサのみならずコプロセッサでも同様のマイクロプログラム制御が用いられている．NS 32081 浮動小数点演算プロセッサでは IEEE 表現に例外ケースが多いことを配慮して，プログラムカウンタを使用しないマイクロ命令形式が採用されている[6]．図-8 に示すように 12 ビットの ROM アドレスのうち下位7ビットは前のマイクロ命令 (20 ビット) 中で定義され，次の4ビットは条件コードによって定められる．したがってマルチウェイ分岐がどのマイクロ命令でも行えることになって例外ケースが高速に処理される．

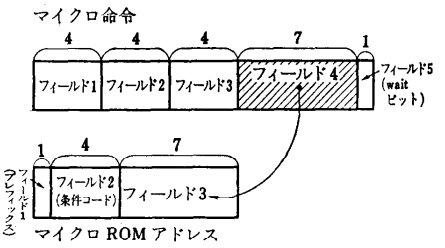

図-8 NS 32081 のマイクロ命令とマイクロアドレス形式

このほかメモリ管理ユニット MC 68851[7] では部分的にマイクロプログラム制御を用い，必要に応じてマイクロシーケンサを起動する方式を取っている．

4.2 マイクロプログラマブル マイクロプロセッサ

汎用のマイクロプログラマブル マイクロプロセッサには二つのタイプが考えられる．一つはシステム構成部品としての汎用性を全面に打ち出したいわゆるビットスライス マイクロプロセッサであり，もう一つはユニバーサルホスト指向のものである．

4.2.1 ビットスライス マイクロプロセッサ

よく知られているようにビットスライス マイクロプロセッサは演算器とレジスタファイルを中心に構成され，チップ外部から与えられるマイクロ命令によって動作を制御される．マイクロ命令のフィールドは演算操作コード，演算レジスタ番号，イミディエイト入力などで構成され，あまり細かい制御ビットはない．Amd 2900 シリーズが有名で広く用いられている．ビットスライス マイクロプロセッサは汎用性を与えるためにマイクロプログラム制御が有効に使用された例といえよう．なお，5. でビットスライス マイクロプロセッサを用いて 32 ビットマシン CPU を構成する例について述べる．

4.2.2 ユニバーサルホスト指向 マイクロプロセッサ

マシン命令をあらかじめ定義しないで，個々の応用に適したマシン命令セットをマイクロプログラムでインタプリート実行するいわゆるユニバーサルホストプロセッサは 1970 年代に広く研究され，商用機も製作されたが必ずしも成功には至らなかった．そのマイクロプロセッサ化 (たとえば，PULCE[8]) も試みられたが，実用にはなっていない．しかし，マイクロプログラムを応用に応じて作成できる汎用のマイクロプログラマブル マイクロプロセッサは，システムを応用に応じてチューニングすることで高性能化するための有効な手段であることに間違いはなく，マイクロプログラムの作成環境を整備することで効果的に利用できる可能性もある．ここでは完全なユニバーサルホストを指向しているわけではないが，チップ外部からマイクロ命令を与えてユーザがマイクロプログラミングできる NCR/32 マイクロプログラマブル マイクロプロセ

ッサ[9]について簡単に述べておこう．なお，そのマイクロプログラム利用の効果については，次章で具体例を述べる．

図-9 にチップの内部構成を示す．マイクロ命令は16ビットで3ステージのパイプラインによってチップ外部から供給される．各マイクロ命令はチップ内部の制御用 ROM 内のマイクロコードによって実行される．マイクロ命令セットは柔軟で比較的分かりやすい構成になっていて，マイクロアドレス空間も 64 K あるから，かなり大きなマイクロプログラムを利用できる．

4.3 RISC マイクロプロセッサ

RISC マイクロプロセッサはカリフォルニア大(バークレイ)の RISC I と II，スタンフォード大の Mips などが有名である．後者は商品化もされ，非常に高性能といわれている．ここではマイクロプログラム制御マイクロプロセッサとの構成的差をみるために古典的な RISC について簡単に述べる[10),11)]．RISC は基本的に 32 ビットの ALU と 78 (II では 138) 語のレジスタファイル，それに若干の制御用レジスタか

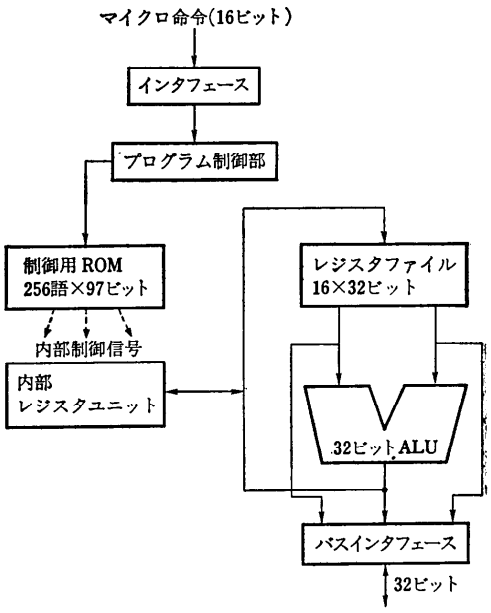

図-9 NCR/32 プロセッサチップの内部構成

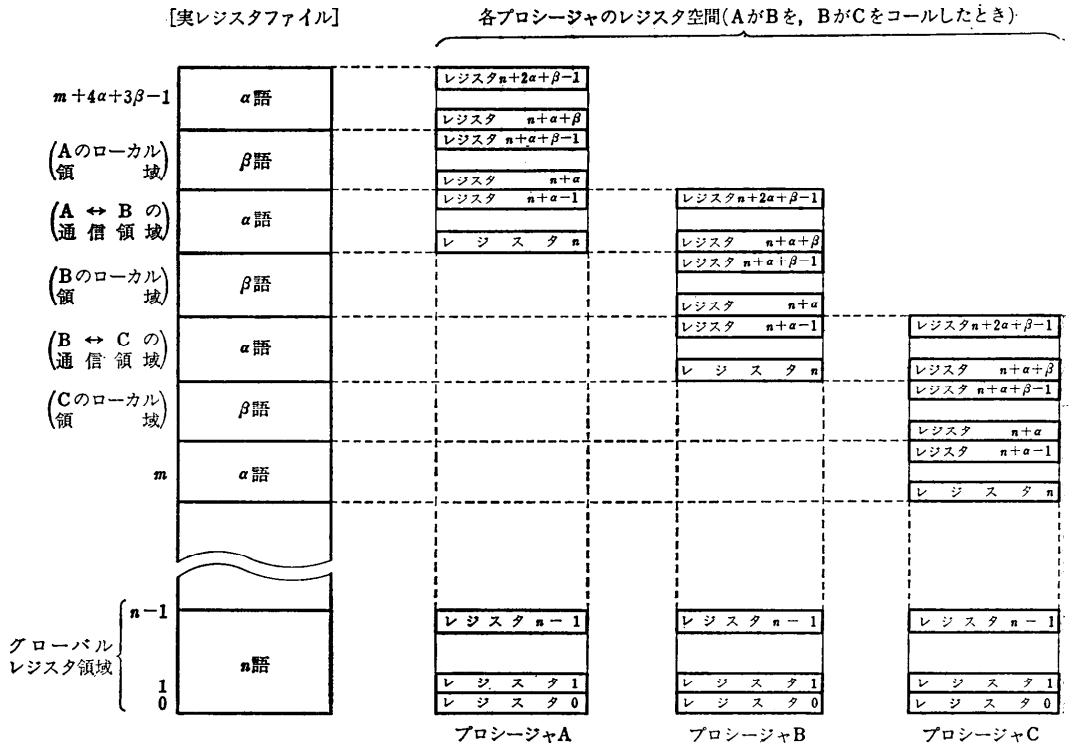

図-10 レジスタウィンドウ

ら構成される．命令は32ビットの固定長で31 (IIでは39) 個しかない．高速化のために遅延分岐とレジスタファイルをウィンドウに分けて使用する方法（図-10）を用いている．チップ内の制御用領域はわずか6% (IIでは10%) にすぎず，空いた領域は主にレジスタファイルに用いられて高速化が達成されている．

5. マイクロプログラマブル マイクロプロセッサの応用

5.1 ビットスライス マイクロプロセッサによるCPU設計

ビットスライス マイクロプロセッサは，一般のマイクロプロセッサに比べて，消費電力や価格が多少高めで，部品数が多くなるという短所があるものの，汎用のものよりも高速で，データやアドレスのサイズが自由に設定でき，マイクロプログラミング機能を利用して命令セットアーキテクチャをユーザが自由に定義できるという利点がある．ここでは，そういったシステムの例としてTI社のAS888ビットスライスチップを用いて32ビットCPUを構成している例をあげる[12]．

システムの概略は図-11のようになっている．ALU部は8ビットビットスライスプロセッサAS888を4個カスケードに接続したALUと16ビットレジスタファイルなどから成る．CCU部はAS890とマイクロプログラムメモリ，制御レジスタ，命令レジスタ，マッピングPROMなどから成る．命令レジスタがデータバスからマシン命令を取り込み，このうち操作コード部をマッピングPROMへ送り，このアドレスにより対応するマイクロプログラムルーチンが読み出される．マイクロプログラムメモリはパイプライン制御されており，マイクロ命令のフェッチと実行は並行して行われる．マイクロ命令は127ビットの水平型で

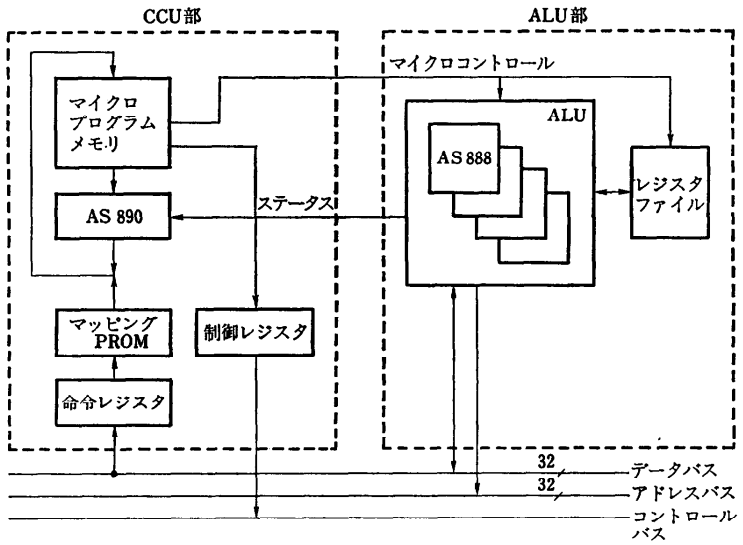

図-11 AS 888による32ビットCPU

表-1 フェッチ及び乗算の処理時間の比較

		AS888 32ビット	AS888 16ビット	Z8001	8086-1	80286	68000L
クロックレート (MHz)		11.1	10.98	4	10	10	8
フェッチ	データ幅	32	——	16	16	16	16
	サイクル数	4	——	3	4	4	4
	合計時間 (ns)	360	——	750	400	400	600
乗算	サイズ	32×32	16×16	16×16	16×16	16×16	16×16
	サイクル数	35	19	70	128	21	≤74
	合計時間 (μs)	3.150	1.729	17.5	12.8	2.1	≤9.25

ある．

クリティカルパスから見積もったマシンサイクルは90 ns で，これをもとにフェッチ，乗算の処理速度を他のマイクロプロセッサと比較すると**表-1** のようになる．

5.2 マイクロプログラムを用いた PROLOG システム

マイクロプログラミング機能は特定の高級言語にチューニングしたマシン(高級言語マシン)の作成に古くから用いられてきたが，ここでは PROLOG の実行性能をあげるための手法としてマイクロプログラマブルマイクロプロセッサを用いた例[13]を示す．

通常の PROLOG プログラムの実行では，WAM (Warren's Abstract Machine) コードという中間形式を経て，ホストマシンのマシン命令プログラムに変換され，実行時に各マシン命令はマイクロプログラムによってインタプリートされる．しかしこの例では，**図-12** に示すようにプログラムを WAM コードから直接ホストのマイクロプログラムに変換し，内部の WCS にダウンロードして実行する方式を採っている．使用しているマイクロプロセッサは，**4.2.2** で述べた NCR/32 である．

このシステムの構成を**図-13** に示す．NCR/32 は図-12 中のバイナリオブジェクトファイルがダウンロードされた 64 K 語の WCS をアクセスしながら処理を行う．

主記憶は 2 M バイトで，PROLOG を処理するためのヒープ，スタック，トレイル (TRAIL) のほか，WAM のレジスタで，NCR/32 の実際のレジスタに割り当てることのできなかったレジスタなどはスクラ

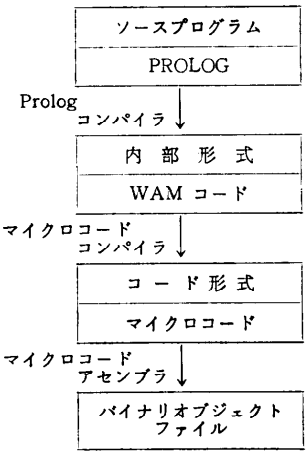

図-12 Prolog プログラム開発手順

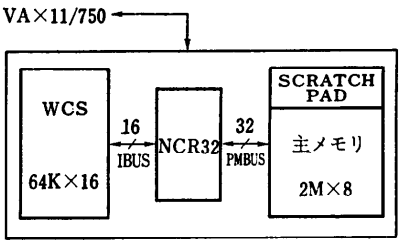

図-13 Prolog システムの構成

ッチパッド上に割り当てられている．また WCS の約 3% の領域には WAM コードで表せないような組み込み関数を常駐させている．

システムの性能比較を**表-2** に示す．これは Deterministic Concatenate (APPEND 処理) の例であるが，既存の汎用マシンの中では最高速になっている．これは，PROLOG をインタプリートではなくマイクロプログラムにコンパイルした効果である．ただし，いくつかのベンチマークで用いたプログラムは**表-3** に示すように 64 K 語の WCS に十分入るサイズであるが，さらに大きなプログラムの場合，WCS に入らないことがあり得る．これがこの方式の欠点で，この場合はスワップ操作が必要となり，実行時間は長くなる．

以上の例は，PROLOG に適した中間形式 (WAM コード) から直接マイクロプログラムへコンパイルすることにより，専用マシンを用いずとも，マイクロプログラマブル プロセッサにより，ある程度の実行性能が得られるということを示している．

表-2 Deterministic Concatenate 実行時の性能比較

	マシン	システム	LIPS 数
既存のマシン	NCR 132	Warren,コンパイル方式	53K
	DEC 2060	Warren,コンパイル方式	43K
	SUN-2	Quintus コンパイラ	40K
	IBM 3033	Waterloo	27K
	VAX-780	マイクロコード方式	15K
	LMI/Lambda	Uppsala	8K
	VAX-780	PROLOG	2K
	VAX-780	M-PROLOG	2K
	VAX-780	C-PROLOG	1.5K
	SYMBOLICS 3600	インタプリート方式	1.5K
	PDP 11/70	インタプリート方式	1K
	Z-80	Micro Prolog	0.12K
	Apple-II	インタプリート方式	0.008K
計画中のマシン	Berkeley PLM	TTL/コンパイル方式	425K
	TICK & WARREN	VLSI	415K
	Aquarius I	TTL/コンパイル方式	305K
	日本第5世代 HPM	マイクロコード方式	280K
	SYMBOLICS 3600	マイクロコード方式	110K

表-3 プログラムサイズ
ベンチマークサイズ

名 前	サイズ(バイト)
nrev	8K
qs 4	8K
palin 25	16K
diff	16K
query	8K
ckt 2	32K
con 1	2K
con 6	2K
hanoi	2K
mumath	8K
pri 2	4K
queens	8K

6. おわりに

本稿では，マイクロプロセッサにおけるマイクロプログラミング技術についてその位置付け，特性，アーキテクチャの実例，応用などについて述べた．一般にマイクロプログラミングに相対する技術とされているRISC 方式も見方を変えればマイクロプログラミング技術の一つの変形であることも示した．

だれでもマイクロプログラミング技術の将来はどうかということに興味があるであろうから，最後にこの点について若干のコメントをしてみたい．ただし，以下は筆者の全くの私見である．マイクロプログラミングと RISC はいずれもハードウェアに近い低レベルの命令を活用する点では同じねらいである．その命令セットをユーザに見せないことも同じであるが，RISC ではコンパイラには見える．しかし，両者の最大の違いは，マイクロプログラミングがインタプリータベースなのにたいし，RISC ではコンパイラ方式であることだろう．コンパイラが良いコードを生成できるならば，その方がずっと高速に実行できることは明らかである．この点では将来においても RISC は優位であろう．特に，集積度の関係から大きなマイクロプログラムインタプリータをチップに内蔵することが困難な GaAs マイクロプロセッサには向いていよう．

しかし．インタプリート方式にも有利な点はある．柔軟性は大であるし，与えられたマシン命令セットに対してマイクロプログラムを作成できるので環境条件の変化に強く，既存ソフトウェアや将来技術への対応性は良い．チップ内の制御領域が多いといっても多くは ROM であってこれを除けば RISC とさして変わらない．したがって，マイクロプログラミング技術が今後もマイクロプロセッサのマシン命令の有効なインプリメント方法として利用され続けるのは間違いなかろう．

最近宣伝が行き届いて RISC の利点が強調されているが，マイクロプログラミング技術を駆逐するようなことはないと思う．ただ，RISC が ISP アーキテクチャ設計に一石を投じたことは事実であって，これから新しいマイクロプロセッサのアーキテクチャを設計するとすれば，コンパイラを初めとしたソフトウェア技術からデバイス技術まで，その現状のみならず将来までよく見越して設計をする必要があろう．今後の可能性としては，両者はともにその利点を生かして場合に応じて使い分けられよう．また，一つのマイクロプロセッサに双方の機能を併用したアーキテクチャも興味ある可能性と思う．

参 考 文 献

1) Stritter, S.: Microprogrammed Implementation of a Single Chip Microprocessor, Proc. Micro 11, pp. 8-16 (1978).
2) Dubose, D. K. et al.: A Microcoded RISC, Comput. Arch. News, Vol. 14, No. 3, pp. 124-128 (1986).
3) LSI-11 WCS Users Guide, DEC (1978).
4) Calcagni, R. E. and Sherwood, W.: Patchable Control Store for Reduced Microcode Risk in a VLSI Vax Microcomputer, Proc. Micro 17, pp. 70-76 (1984).
5) El-Ayat, K. A. and Agarwal, R. K.: The Intel 80386-Architecture and Implementation, IEEE Micro, Vol. 5, No. 6, pp. 4-22 (1985).
6) Gavrielov, M. and Epstein, L.: The NS 32081 Floating-point Unit, IEEE Micro, Vol. 6, No. 2, pp. 6-12 (1986).
7) Cohen, B. and McGarity: The Design and Implementation of the MC 68851 Paged Memory Management Unit, IEEE Micro, Vol. 6, No. 2, pp. 13-28 (1986).
8) Iizuka, H. et al.: Development of a High-performance Universal Computing Element-PULCE Proc. NCC, Vol. 47, pp. 1255-1264 (1978).
9) NCR/32-000 32-Bit Microprogrammable Microprocessor, NCR (1983).
10) Patterson, D. A. and Sequin, C. H.: A VLSI RISC, IEEE Computer, Vol. 15, No. 9, pp. 8-21 (1982).
11) Tabak, D.: RISC Architecture, Research Studies Press (1987).
12) Lazko, F. et al.: 32-Bit CPU Design with the 'AS 888/'AS 890, ACM Sig Micro Newsletter, Vol. 17, No. 2, pp. 8-13 (1986).
13) Fagin, B. et al.: Compiling Prolog into Microcode--A Case Study Using the NCR/32-000, Proc. Micro 18, pp. 79-88 (1985).

(昭和62年8月10日受付)