The Evolution of Forth

Original https://www.forth.com/resources/forth-programming-language/

Abstract

Forthは、プログラミング言語の中でも、企業や学術的なスポンサーの支援を受けない草の根的な活動によって発展・普及したユニークな言語である。当初は一個人が考案・開発したものであったが、その後、アプリケーションの問題を解決するためのツールを開発し、それを商品化したプロのプログラマと、Forthの無償配布に関わったホビーストの二つの大きな影響を受けながら発展してきた。これらの影響により、従来のプログラミング言語とは明らかに異なる言語が生み出された。

著者

エリザベス・D・ラザー

Elizabeth Ratherは、FORTH社の共同設立者であり、プログラミング言語Forthの第一人者である。エリザベスは、チャック・ムーアが1970年代初頭にNRAOで働いていたときの同僚であった。彼がForthを開発した際、彼女は史上2人目のForthプログラマとなった。それ以来、彼女はこの言語の第一人者となり、その主要な支持者の一人となっています。ForthのANSI規格(1994年)を作成したANSI技術委員会の委員長を務める。彼女はForthに関するいくつかの本の著者であり、その使用法に関するトレーニングセミナーを数多く開催している。

ドナルド・R・コルバーン

Don Colburn は最も早い時期の Forth ユーザの一人である。彼は Forth Interest Group の創立者の一人であり、最初のパブリックドメイン figForth の開発に貢献しました。その後、Creative Solutions, Inc. (CSI) を設立し、1984 年に MacForth™ を発表しています。MacForth は、発売当時、Macintosh 上で動作する最初のプログラミング言語でした。Forth の ANSI 規格(1994 年)を作成した ANSI 技術委員会のメンバーでもある。2009 年に死去。

チャールズ・H・ムーア

Chuck Moore は Green Arrays, Inc.の会長兼 CTO である。1971 年に FORTH 社を共同設立し、1980 年代半ばには Forth ベースのチップ(RTX2000)を開発し、その派生型は現在でも NASA で広く使われている。コンピュータ・カウボーイズ社では、マイクロプロセッサ「Sh-Boom」を設計し、その後、インターネット・アプライアンスメーカーの iTv を共同設立した。1990 年代には、独自の CAD ソフトウェアを使用して、ネットワーク・インターフェイスを備えた F21 プロセッサを含む、いくつかのカスタム VLSI チップを設計しました。最近では、colorForth を発明し、VLSI 設計ツールを移植しています。S40 マルチコンピュータチップの開発時には、IntellaSys 社の CTO を務めました。

1. チャールズ・ムーアのプログラミング言語

1.1 初期の発展

ムーアのプログラミングのキャリアは 1950 年代後半にスミソニアン天体物理観測所で、エフェメライド、軌道要素、衛星局の位置などを計算するプログラムによって始まった。[Moore, 1958], [Veis, 1960]。彼のソースコードはカードトレイ 2 枚を埋め尽くした。この大きなプログラムの再コンパイルを最小限にするため、彼はプログラムを制御するカードを読み取る簡単なインタプリタを開発した。これにより、彼は再コンパイルすることなく、複数の衛星のために異なる方程式を構成することができた。このインタプリタには、現代の Forth に受け継がれているいくつかのコマンドとコンセプトがあり、主にスペースで区切られた「単語」を読むコマンドと、数字を外部形式から内部形式に変換するコマンド、それに IF ... ELSE 構成がある。彼は、自由形式の入力が、特定の列にフォーマットするとい

う、より一般的なFortranのやり方、これは列間違いにより果てなく再実行を繰り返す結果になるやり方よりも効率的で(より小さく、より速いコード)、信頼できるものであることを発見しました。

チャールズ・ムーア FORTH, Inc.

1961年、ムーアはMITで物理学の学士号を取得し、スタンフォード大学の大学院に入学した。彼は、スタンフォード線形加速器(SLAC)でパートタイムのプログラミングの職に就き、それまでの最小二乗フィッティングの仕事を拡張して、当時建設中の2マイル電子加速器のビームステアリングを最適化するコードを書きました。この研究の重要な成果は、Algol(1964年)でコード化されたCURVEと呼ばれるプログラムで、汎用の非線形微分補正データフィッティングプログラムであった。このプログラムを制御するために、彼は自分のインタプリタを拡張し、パラメータ渡しのためのプッシュダウンスタック、明示的に値を取得・保存できる変数、算術・比較演算子、手続きを定義・解釈する機能などを管理できるようにしたものを使用した。

1965年、ニューヨークに移り住み、フリーランスのプログラマとなる。Fortran、Algol、Jovial、PL/I、各種アセンブラを駆使し、文字通りカードデッキを持ち歩きながら、必要に応じて再コード化するなど、インタプリタを可能な限り使い続けていた。60年代終盤にはミニコンピュータが登場し、それに伴ってテレタイプ端末が登場したが、ムーアはこの端末用に演算子を追加して文字の入出力を管理するようになった。あるプロジェクトでは、Fortran-Algol変換器とファイル編集ユーティリティを作成した。このとき、Fortranのソースでは必要なかった単語と単語の間のスペースが、ムーアにとって重要な意味を持つようになった。

1968年、新婚のムーアは、小さな町の環境を求めて、ニューヨーク州アムステルダムのモハスコ・インダストリーズ社に入社した。ここで彼は、2250グラフィックディスプレイを搭載したIBM1130ミニコンピュータ用のコンピュータグラフィックスプログラムを開発した。このコンピューターには、16ビットのCPU、8KのRAM、最初のディスク、キーボード、プリンター、カードリーダー／パンチ(ディスクのバックアップとして使用!)、Fortranコンパイラーが搭載されていた。彼は、2250用のコードを生成するために、プログラムにクロスアセンブラを追加し、さらにプリミティブエディタとソース管理ツールも追加しました。このシステムは、IBMのソフトウェアが静的な2D画像しか描けなかった時代に、アニメーションの3D画像を描くことができた。また、遊びで、初期のビデオゲーム「Spacewar」を書いたり、「Algol Chess」プログラムを、初めて「FORTH」と呼ばれる新しい言語に変換したりもした。彼は、FORTHが非常にシンプルになったことに感動した。

FORTHという名前は、ムーアが分散型小型コンピュータを特徴とすると見ていた第4世代(次世代)コンピュータのためのソフトウェアを示唆するものであった。当時彼が使っていたOSは、ファイル名を5文字に制限していたため、「U」は捨てられた。FORTHは、1970年代後半まで大文字で表記されていたが、これは大文字専用のI/Oデバイスが普及していたためである。頭字語ではないので、小文字が普及すると、「Forth」という使い方が一般的に採用されるようになった。

ムーア氏は、2250をプログラミングするためのForthベースの1130環境が、1130のソフトウェアが開発されたFortran環境よりも優れていることに気づき、1130コンパイラに拡張した。その結果、ループコマンド、ソースを1024バイトのブロックに分けて管理する概念とその管理ツールなど、現在Forthで認識されているコンパイラの機能のほとんどが追加された。

最も重要なのは、辞書が追加されたことだ。手続きは名前を持つようになり、インタプリタは名前のリンクリストを検索して一致するものを探しました。これはスタンフォード大学のコンパイラーから学んだことで、1980年代までForthで広く使われていた。辞書の項目内には、そのルーチンで実行されるコードのアドレスを含む「コードフィールド」があった。これは間接スレッドコード実装(5.2節参照)で、Dewarの間接スレッドコードに関する論文がCommunications of the ACM [Dewar 1975]に掲載される5年前から使用されていたものである。間接スレッドコードの使用は重要な技術革新でした。なぜなら、単語を見つけた後には間接ジャンプが唯一のオーバーヘッドとなるからです。辞書の項目は、他の「高水準」ルーチンへのポインタか機械命令で構成されていた。

最後に、ルーチンをネストするための簡単なメカニズムを提供するために、「リターンスタック」と呼ばれる第2のスタックが追加された。リターンアドレス用に予約されたスタックを持つことの利点は、呼び出しの前後に「バランス」を取る必要がなく、もう1つのスタックをパラメータ受け渡しに自由に使えることであった。Forthに関する最初の論文はモハスコ社で書かれた [Moore,1970a]。1970年、モハスコ社は、新しいUnivac 1108で受注システムのための専用線ネットワークを処理するという野心的なプロジェクトをMoore氏に任せた。彼は、Forthを1108に移植し、トランザクション処理を行うCOBOLモジュールとのインタフェースを確保した。1108のForthは、アセンブラでコード化されていた。入出力のメッセージをバッファリングし、各行を処理するタスク間でCPUを共有する。また、入力を解釈し、適切なCOBOLモジュールを実行した。このバージョンのForthには、タスクを定義し管理する仕組みや、現在使われている仕組みに似たディスクブロックバッファを効率的に管理する仕組みが追加されていた。しかし、不況のあおりを受けて、モハスコ社は1108プロジェクトの完成を待たずに中止を決定した。Mooreは直ちにその旨を伝

え、怒りの詩を書き、出版されることのなかったForthに関する本 [Moore, 1970b]を書きました。それは、Forthソフトウェアの開発方法を説明し、シンプルさと革新性を奨励するものでした。

1.2 哲学と目標

MooreにとってForthは、既存のソフトウェアツールに対する不満に対する個人的な反応であり、彼はそれを一種の「バベルの塔」[Moore, 1970a]と見なしていた。

> アセンブラはコンパイラとスーパーバイザを記述するための言語、スーパーバイザはジョブ制御のための言語、コンパイラはアプリケーションプログラムのための言語、アプリケーションプログラムはその入力のための言語を定義しているのです。ユーザはこれらすべての言語を知らないかもしれませんし、知ることもできないかもしれません。しかし、これらの言語は、ユーザとコンピュータの間に立ち、ユーザができることとそのコストに制約を課しているのである。
>
> この膨大な言語の階層を作るには、人と機械の膨大な時間が必要であり、保守するにも同様に大きな労力を必要とします。これらのプログラムを文書化するのにも、その文書を読むのにも膨大なコストがかかる。そして、これだけ努力しても、プログラムはバグだらけで、使いづらく、誰も満足しないのである。

ムーアはForthを、「広大な階層」全体を単一の層に置き換え、最小限の文書からなるプログラマとForthのインターフェース(インターフェースはシンプルで自然であるべきなので最小限のもの)と、プログラム自体からなるForth-マシンのインターフェースの2要素だけで構成するものと考えたのである。彼の考えは、自分の経験に照らして自分の必要性を考慮した、まったく個人的なものであった。以下は、彼の未発表の著書[Moore, 1970b]からの抜粋であるが、この見解が述べられている。

> 私は長年にわたって多くのプログラムを書いてきた。良いプログラムを書こうとしてきたし、自分の書き方をかなり批判的に観察してきた。私の目標は、必要な労力を減らし、生成される品質を高めることであった。

> その中で、私自身が同じ過ちを繰り返していることに気がつきました。振り返れば一目瞭然なのに、文脈からではなかなか気づかないミス。プログラミングの処方箋を書けば、少なくとも問題点を思い出すことができるのではないかと思ったのです。そして、その結果が自分にとって価値あるものであれば、他の人にとっても価値あるものになるはず.....。

何よりも、彼が「基本原則」と呼んでいた指針は、"Keep it simple!"であった。彼はキャリアを通じて、この原則を宗教的な献身を込めて守ってきた。

> プログラムに追加する機能の数が増えれば増えるほど、プログラムの複雑さは指数関数的に増していく。プログラムの内部的な整合性はもちろんのこと、これらの機能間の互換性を維持する問題は、簡単に手に負えなくなる。基本原則を適用すれば、これを回避することができます。基本原則を無視したオペレーティングシステムをご存じかもしれませんね。
>
> 基本原則を適用するのは非常に難しいことです。内外のあらゆる圧力が、プログラムに機能を追加しようと謀っているのです。結局のところ、半ダースの命令しか必要ないのだから、なぜそうしないのか？唯一の対抗力は「基本原理」であり、これを無視すれば対抗力は存在しない。

単純化の大敵は、プログラマが将来のニーズを推測してそれを提供しようとする一般化のサイレンコールであると彼は考えた。そこで彼は、基本原則に "憶測で物を言うな!"という副次的な言葉を付け加えた。

> 使われるかもしれないコードをプログラムに入れてはいけない。拡張機能をぶら下げるようなフックを置いてはいけない。あなたがやりたいと思うことは無限であり、それぞれが実現する確率は0である。もし後で拡張機能が必要になったら、後でコーディングすればいいのです - そしておそらく、今やるよりも良い仕事ができるはずです。そして、もし他の誰かがその拡張機能を追加したら、その人はあなたが残したフックに気づくでしょうか？あなたのプログラムのこの点を文書化しますか？

このやり方は、当時も今も常識から外れている。もう1つは、「自分でやれ!」という異端児的なものである。

> 従来のやり方は、多かれ少なかれ、標準的なサブルーチンを使うようにと強制されていた。私は、「自分でサブルーチンを書きなさい」と言う。

> 自分でサブルーチンを書く前に、書き方を知らなければならない。これは、現実的には、以前に書いたことがあるということであり、そのため、始めるのが難しいのです。しかし、一度試してみてください。同じサブルーチンを何台ものコンピュータと言語で何十回も書いているうちに、かなり上手になるはずだ」。

ムーアは、これを驚くほど忠実に実行した。1970年代を通じて、彼は18種類のCPUにForthを実装し(表1)、必ずそれぞれに独自のアセンブラ、独自のディスクおよびターミナルドライバ、さらには独自の乗除算サブルーチン(多くのマシンで必要とされていた)を書きました。これらの関数についてメーカー提供のルーチンがある場合、彼はそれを読んでアイデアを得ましたが、そのまま使うことはありませんでした。Forthがこれらのリソースをどのように使うかを正確に把握し、フックや一般性を省き、また、熟練と経験によって(彼は、ほとんどの乗除算サブルーチンは、これまで書いたことがなく、これからも書くことがない人が書いたと推測しています)、彼のバージョンは常に小さく、速く、通常は著しく速くなりました。

さらに、彼は、自分自身の問題解決に満足することはなかった。数年後にコンピュータやアプリケーションを見直し、主要なコードルーチンを書き直すことがよくあった。また、自分の作ったコードを、改良を目的とした再確認なしでそのまま再利用することは決してなかった。FORTH社(2.2節参照)のマーケティング部門であるラザー社は、ムーア氏が同様のプロジェクトを行ったことがあるので、このプロジェクトは簡単だろうという前提でしばしば仕事に入札したが、彼が過去のコードをすべて破り捨ててやり直すのをなすすべなく見ているだけであった。

現在、ムーア氏はForthベースのマイクロプロセッサの設計を、独自のForthベースのCADシステムを使って行っているが、それは、1979年以来ほぼ継続的に書き直し(時には自分のハードウェアで作り直し)続けてきている。

ムーア氏は、自分をアプリケーションプログラマと呼び、それを高尚な職業とみなしていた。「アプリケーションプログラマ」のためにツールを作る「システムプログラマ」は、その構成員に対して恩着せがましい態度をとっていると、ムーアは感じていた。彼は、プログラマからシステムを、プログラマから自分自身を守るためにシステムプログラマが築い

た障壁を回避するために、自分の職業人生の大部分を費やしてきたと感じており、Forthにはそうさせないようにすると決意した。Forthは、知的で高度な技術を持つプログラマのために設計され、束縛するのではなく、力を与えることを意図していた。

ムーア氏の哲学の正味の結果としてのシステムは、小さく、単純で、明確で、極めて柔軟性に富むものでした。この哲学を実践するためには、柔軟なソフトウェアが不可欠です。将来の拡張のためにフックを残しておくのは、一般に要件が変わったときに何かを一から再実装するのが難しすぎ、時間がかかるからです。ムーア氏は、(単純で柔軟性に富むツールを使用して)コンピュータに「何かを」できるように教えることと、巨大な汎用OSで「何でも」できるようにすることの間に、明確な違いがあることを見抜いていた。ムーア氏は、前者に専念することで、自分のビジョンに沿った理想的なツールセットを手に入れたのである。

2. Development and Dissemination

1970年代初頭には、Forthは重要なアプリケーションに使用できるようになっただけでなく、他のプログラマや組織の注目を集めるほどの成熟度に達していた。彼らのニーズに応え、Mooreはより多くのコンピュータにForthを実装し、より大規模なアプリケーションを扱えるようにしたのです。

2.1 NRAOにおけるForth

ムーア氏は1971年、アリゾナ州キットピークにある国立電波天文台(National Radio Astronomy Observatory: NRAO)が運営する11メートル電波望遠鏡のために、初の完全なスタンドアロン型Forthの実装を開発しました。このシステムは、2台の初期のミニコンピュータ(16KBのDDP-116と32KBのH316)をシリアルリンクでつないで動作させていました。マルチプログラムシステムであり、マルチプロセッサシステムでもあるこのシステムは、望遠鏡の位置決めと追跡動作、データの収集と磁気テープへの記録、天文学者が以前に記録したデータを分析するための対話型グラフィック端末をサポートする役割を担っていました。このシステムは、マルチプログラミング性により、タイミングが合わなかったり、他の干渉を受けることなく、これらすべての機能を同時に実行することができました。

また、磁気テープをソースとして、ミニコンピュータそれ自身の上でソフトウェアの開発を行うという、当時としてはユニークなシステムであった。このForthシステムは、アプリケーションの開発をサポートするだけでなく、自分自身をもサポートしていたのである。Forth自体はForthで書かれており、必要な時に新しいシステムカーネルを生成する「メタコンパイラ」を使っていた。

このようなソフトウェアの能力の本質を当時の状況も踏まえて考えるためには、初期のミニコンピュータのメーカー提供のシステムソフトウェアが極めて原始的であったということを理解しておくことが重要だ。主なツールはメインフレーム上で動作するクロスアセンブラとFORTRANクロスコンパイラでした(ただし、FORTRANクロスコンパイラは、ターゲットマシンのメモリが小さいため、複雑な処理をするには効率が悪すぎました)。オンラインプログラミングのサポートは限定的で、紙テープから読み込むアセンブラと、紙テープで管理するソースに限られていた。デジタル・イクイップメント社が PDP-11 用に発表した RT-11 OS はフォアグラウンド/バックグラウンド動作が制限されており、H316 ファミリでは並行処理ができない状態であった。NRAOの天文学者は、オペレーターが望遠鏡を操作し、ライブデータが流れ込む中、グラフィカルにデータを分析するようなマルチユーザオペレーションは前代未聞であった。

11m望遠鏡を運用するNRAOのタクソン部門の責任者であるエドワード・K・コンクリン(Edward K. Conklin)は、ムーアがバージニア州シャーロッツビルのNRAO本部を拠点としていたため、ソフトウェアのメンテナンスが困難であることを理解しました。そこで1971年、彼はアリゾナ大学のシステムアナリストであるエリザベス・ラザー(Elizabeth Rather)を呼び寄せ、パートタイムで現地サポートをさせることにした。ラザーは、この重要なシステムが、文書化されていない、たった1人の人間しか知らない独自の言語で書かれていることに愕然とした。彼女は最初、FORTRANで全部書き直して、コントロールできるようにしようと思った。しかし、そのための時間も予算もない。そこで彼女は、できる限りシステムを学び、文書化することにした。

2カ月ほど経った頃、ラザーは何かとんでもないことが起きていることに気づき始めた。オンライン・コンピュータは信じられないほど原始的で、言語も奇妙で、地元の専門家やリソースもないにもかかわらず、彼女は週に一度、FORTRANコンピュータで数時間過ごすだけで、大型メインフレームを実質無制限に使える週の残りの期間より多くのことを成し遂げることができたのである。

彼女はなぜそうなったかを考えた。その答えは自明で、Forthの対話的な性質にあるように思えた。ファイルを開いたり閉じたり、コンパイラ、リンカ、ローダ、デバッガをロードして実行したりといった手続き的なオーバーヘッドによって、プログラマの注意が途切れることがないのです。しかし、それ以上のことがある。たとえば、ForthのOS、コンパイラ、その他の内部機能で使われるツールはすべてプログラマが利用できる。そして、チャック・ムーアが意図したように、その制約は最小限であり、その態度は寛容である。Forthの信奉者たちは、こうした生産性向上の源泉や規模について、今でも議論を交わすのが大好きなのです

ラザーはすぐに大学を去り、NRAOが施設を共有している光学観測所であるキットピーク国立天文台(KPNO)と共同で、NRAO用のForthシステムのメンテナンスとKPNO用システムの開発(これは後にKPNOの156″Mayall望遠鏡や他の機器で使用された[Phys. Sci. 1975])に取り組み始めたのです。次の2年間で、彼女は最初のForthマニュアルを書き [Rather, 1972] 、天文台や関連する天文団体で多くの論文やコロキアムを行いました [Moore, 1974a]。

1973年、ムーアとラザーはツインコンピュータシステムをディスクベースの PDP-11 コンピュータに置き換えた [Moore, 1974a&b]。これはマルチユーザシステムで、望遠鏡の制御とデータ取得のタスクに加え、4つの端末をサポートした。これは非常に成功し、制御部分は1991年現在でも使われている(データ取得と解析機能は、年々激変している実験装置や技術に依存するところが大きい)。このシステムは非常に先進的で、世界中の天文学者がこのソフトウェアのコピーを求め始めました。スチュワード天文台、MIT、インペリアルカレッジ(ロンドン)、セロ・トロロ(チリ)米州天文台、ユトレヒト大学(オランダ)などに導入された。その使用は急速に広まり、1976年には国際天文学連合によってForthが標準言語として採用された。

2.2 商用ミニコンピュータシステム

1973年にアップグレードされたシステムが完成した後、ムーアと彼の同僚のラザーとコンクリンは、FORTH, Inc.を設立し、この言語の商業的利用を模索した。FORTH社は、当時使われていたほとんどのミニコンピュータ(表1参照)用のマルチユーザ版Forth [Rather, 1976a]を開発し、データベースアプリケーションから画像処理などの科学アプリケーションまで、広く多様な市場でカスタムアプリケーションのコンポーネントとして販売した。1970年代のミニコンピュータとアプリケーションは、Forthが発展し安定するための環境を提供し、その後の独立した実装者による技術革新は、すべてこのテーマに対する比較的小さなバリエーションに過ぎなかった。このため、これらのシステムの設計と構造を詳しく見ていくことにする。

Year	Model	Customer	Forth Applications
1971	Honeywell H316	NRAO	Data acquisition, on-line analysis w/graphics terminal
1971	Honeywell DDP116	NRAO	Radio telescope control
1972	IBM 370/30	NRAO	Data analysis

Year	Model	Customer	Forth Applications
1972	Varian 620	KPNO	Optical telescope control and instrumentation
1972	HP2100	KPNO	Instrumentation
1973	Modcomp	NRAO	Data analysis
1973	PDP-11	NRAO	Radio telescope control, data acquisition, analysis, graphics
1973	DG Nova	Steward Observatory	Data acquisition and analysis
1974	SPC-16	Steward Observatory	Ground control of balloon-borne telescope
1975	SDS920	Aerospace Corp.	Antenna control
1975	Prime	General Dynamics	Environmental controls
1976	Four-Phase	Source Data Systems	Data entry and data base management
1977	Interdata Series 32	Alameda Co.,	CA Data base management
1977	CA LSI-4	MICOA	Business systems
1978	Honeywell Level 6	Source Data Systems	Data entry and data base management
1978	Intel 8086	Aydin Controls	Graphics and image Processing
1980	Raytheon PTS-100	American Airlines	Airline display and workstations

Table 1. Computers for which Chuck Moore personally implemented Forth systems.

1978年には、レベル6と8086におけるForthの実装は、両CPUにおける最初の常駐ソフトウェアとなり、メーカーのシステムを何ヶ月も先取りした。

2.2.1 環境の制約

1970 年代のミニコンピュータは、現在の最小のマイクロコンピュータに比べ、はるかに性能が劣っていた。1970 年代前半には、すべてのシステムがディスクを搭載しているわけではなく、1/2 インチのテープが唯一の大容量記憶装置であることがよくありました。メモリサイズも 16K バイトから 64K バイトまであったが、64k バイトが大きいとされていた。1970 年代前半は、ミニコンピュータのプログラミングはほとんどアセンブリ言語で行われていた。1970 年代半ばには Fortran や BASIC のコンパイラが登場し、DEC の RT-11 のようなメーカー提供の実行環境がフォアグラウンド／バックグランド動作をサポートするようになった。PDP-11 や Nova は 8 ユーザをサポートすることが期待されましたが、アクティブユーザ 8 人におけるシステムのパフォーマンスは低いものでした。

このハードウェア上で、ムーアの Forth システムは、アセンブラ、エディタ、高級 Forth 言語へのインタラクティブなアクセスを含む統合開発ツールキットと、64 ユーザを目に見える性能低下を起こさずにサポートするマルチタスク、マルチユーザ動作環境を組み合わせ、すべてランタイムオーバーレイなしで常駐させていた。

システムの時間的に重要な部分はアセンブラで書かれていたが、ほとんどのアプリケーションが非常に高い性能を必要としていたが、ムーアは Forth 開発環境全体を約 2 週間で新しいコンピュータに移植することができた。Forth のコンピュータは、ターゲットシステムのアセンブラと約 60 のプリミティブのコードがあれば、別のコンピュータの Forth を生成することができる。移植の最初のステップは、ターゲットアセンブラを設計し書くことだったので、ムーアは、他の誰よりも、異なるプロセッサのためのアセンブラを書いている可能性がある。

ミニコンピュータ市場は非常に細分化されていたため、新しいアーキテクチャに簡単に移植できることが重要だった。ミニコンピュータ市場は極めて細分化されており、多数の CPU が利用可能で、それぞれが多数のディスクコントローラとドライブの組み合わせでサポートされていました。対照的に、現在のマイクロコンピューター市場は、ごく少数のプロセッサファミリに支配され、PC/AT のようなデファクト・スタンダードに準拠するのが普通である。

ミニコンピュータを出荷するのは現実的ではないため、インストールは現地で行われた。LSI-11 が発売されると、ムーア氏は 1 台購入し、機内持ち込み用のスーツケースに入れ、もう 1 つのスーツケースに 8 インチフロッピーディスクドライブを 1 台だけ入れて搭載した。このポータブル・パーソナル・コンピュータは、1982 年までムーアと行動を共にし、新しい Forths を生み出すための「フレンドリな」ホストとして機能した。

2.2.2 アプリケーションの要件

主な環境上の制約が、メモリの制限と幅広いCPUアーキテクチャに対応する必要性であるとすれば、アプリケーションの要件は性能の必要性によって支配されていた。この時期、Forthが成功を収めた主な応用分野を以下に挙げる。

1. 商用・業務用データベースシステム

Arthur A. Gravinaの指導のもと、Cybek社のために開発されたこのシステムは、Data General Nova上で複数の端末をサポートし、高速なトランザクション処理を行うものであった。最初の1台は、1974年、Pacific Telephoneのサービスビューローである Vernon Graphics, Inc.のために開発された。300MBのデータベースに対して、32台の端末でトランザクションを処理するものであった。最初の1週間で、このシステムは1日に10万件以上のトランザクションを処理した(要件は4万件だった)。その後、64台の端末と600MBのデータベースをサポートするシステムにアップグレードしたが、レスポンスタイムに目立った劣化はなく、1秒未満を維持した。

サイベック社はその後、このシステムを銀行や病院管理などのビジネス用途に販売し、現在のバージョンはマクドネル・ダグラス社の一部門が販売している。アイオワ州のSource Data Systems社も同様の取り組みで、病院管理などの用途にNCR社が販売する多端末データ入力システムを開発しました。

このようなシステムの性能は、オペレーティングシステムの問題が支配的で、主にネイティブForthのブロックベースファイルシステムがデータファイルを非常に高速に読み書きできることが重要です。

2. 画像処理。

FORTH社は、海軍兵器研究センター、NASAゴダード宇宙飛行センター、英国王立グリニッジ天文台などのために、一連の画像処理アプリケーションを開発しました。これらの中心は、異なる種類のハードウェアに存在する画像に対して、標準化された操作(例えば、強調、ウィンドウ表示など)を実行する必要性であった。このアプローチには、カプセル化(基本オブジェクトは「画像」であり、特徴的なパラメータとメソッドを持つ)、継承(以前に定義された画像クラスの特徴を継承する新しい画像を追加できる)、操作メソッドの動的結合など、現在オブジェクト指向プログラミングに関連している多くの機能が含まれていました。このアプローチの主要な設計者であるムーアは、この分野の学術的な研究成果を知りませんでした。彼は、後のOOPSの作者と同じ目標を達成するために、独自に同様の解決策を導き出しました。

画像処理システムは、例えば、512×512×16の画像で512KBを占める大容量のデータを高速に操作・移動する必要があることも特徴である。Forthデータベースシステムの特徴である高速なディスク性能に加え、FFTなどのアルゴリズムに対応した高速な処理速度が要求された。多くのミニコンピュータはハードウェアの浮動小数点演算を備えていないため、Forthは柔軟な整数演算と固定小数点演算、さらに特殊な配列プリミティブを備えていた。

3. 計測と制御分野

FORTHは NRAOで最初に開発され、計測と制御のために使われましたが、今日でもFORTHは広く計測器や制御のために使われています。FORTH社はさらにいくつかの天文システムを手がけました(ワイオミング大学[Gehrz, 1978]、ミネソタ大学、ハワイ大学、イリノイ大学、カリフォルニア工科大学、それにイギリスの王立グリニッジ天文台とセント・アンドリュース大学など)。さらに、プリンストン応用研究所(Princeton Applied Research)(現在はEG&Gの一部門)やニコレット機器(Nicolet Instruments)など、多くの商業機器メーカーが社内開発用の言語としてForthを採用した。

これらのアプリケーションは、高いデータレートを特徴としています(場合によっては20KHzという高データレートになる)。利用可能なプロセッサのCPU速度に大きな負担をかけるものであった。また、オペレータの操作や機器の制御と同時にデータ取得を行うため、高速なマルチタスク処理と高速な割り込み応答が不可欠でした。

2.2.3 影響を受けたもの

1978 年以前の Forth の進化は、Moore 氏自身によって完全に支配されていました。これまで見てきたように、ムーア氏は熱狂的なミニマリストであり、ゼロベース設計の原則に専心していました。つまり、すべての機能、すべての命令はその存在を正当化するか、さもなくば冷酷に廃棄されなければなりません。

ムーア氏は、もともと自分が使うためにシステムを開発した。ムーアは、当初自分が使うために開発したシステムだったが、ラザーや他の初期のユーザもこのシステムを気に入り、自分と同じように生産性を向上させることができたことに、少し驚いた。しかし、FORTH社を設立し、このシステムをオープンに販売した後も、サポートツールや一般的なプログラミングインターフェースの選択と設計は、彼の個人的な好みに支配された。

ムーアは、主にコンサルタントとして、FORTH社の他の社員の支援を受けながら、カスタムアプリケーション開発の最初のステップとして、顧客のコンピュータにForthシステムをインストールする仕事をしていた。顧客は、主にアプリケーションに興味を持っていたので、移植を迅速かつ安価に完了させることが急務であった。Forthは非常にシンプルであるため、アプリケーションの性能を損なうことなく、移植を実現することができた。

これらのプロジェクトは、それぞれ独自の教訓、ツール、テクニックを提供してくれた。ムーアは、過去のすべてのプロジェクトのリストをマイクロフィルムに焼いてブリーフケースに入れて持ち歩き、過去のユニークなプリミティブやドライバのコードを得るために、しばしばそれを参照した。よく使う単語は、システムの標準的な備品になるかもしれない。また、よくある問題を解決するための改良技術も、システムの中に組み込まれていった。

しかし、このような継続的な進化は、FORTH社にとって顧客サポートの頭痛の種となった。インストールされたシステムのどの2つとして同じものがなかったからである。ほとんどの場合、インストールにはラザーが講師として教える5日間のFORTHプログラミングコースが含まれており、ラザーは毎晩、システムが教えられたとおりに動作しているかどうか確認しなければならなかった。

2.3 初期のマイクロプロセッサシステム

1976年、RCA社の半導体部門のRobert O. Winderは、新しいCDP-1802 8ビットマイクロプロセッサにForthを実装するためにFORTH社に依頼した [Rather, 1976b], [Electronics, 1976]。この新製品は「microFORTH」と呼ばれ、その後Intel 8080、Motorola 6800、Zilog Z80に実装され、FORTH社から既製品として販売されることになった。microFORTHは、米国、英国、日本において、多くの組み込みマイクロプロセッサの計測・制御アプリケーションに採用され、成功を収めた。

2.3.1 環境とアプリケーション

microFORTHは、FORTH社にとって初めての既製品、かつ通信販売によるソフトウェアパッケージであり、ミニコンピュータシステムはすべてオンサイトでインストールされていた。この通信販売は、8インチIBMフォーマットフロッピーディスクが急速に標準化されたことと、各CPUタイプの開発システムの数が比較的少なかったことによって実現された。

これらのマイクロプロセッサは、すべて8ビットデバイスで、開発システムのメモリは通常16Kバイトであった。ターゲットシステムは通常カスタムボードで(ただしインテルのシングルボードコンピュータシリーズがすぐに普及した)、ソフトウェアはディスクや(通常)ターミナルのない組み込み環境でPROMから実行することが期待された。これは、常にディスクがあり、開発用のコンピュータと同じ(あるいは同一の)コンピュータ上でプログラムが動作することが期待されていたミニコンピュータ環境とは大きく異なるものであった。

ほとんどのマイクロプロセッサメーカーは、ターゲットと同じマイクロプロセッサ、最大64Kバイトの RAM、ターミナル用のシリアル回線、パラレルプリンタポート、2台の8インチフロッピーディスクドライブからなる開発プラットフォームを提供していた。ソフトウェアのサポートは主にアセンブラであった。Intel はすぐに PL/M を導入した。デバッグ用にインサーキットエミュレータと個別のユーティリティが導入された。

microFORTH は、PL/M とは異なり、ほとんどのマイクロプロセッサのファミリで利用でき、したがって高い移植性を提供するアセンブラの対話型代替品として主に販売されました。

2.3.2 Language definition

2.3.2 言語定義

最初は8ビットスタック幅と128バイトブロックバッファで実験しましたが、すぐにミニコンピュータシステムと同じ基本的な内部アーキテクチャを維持することに決定されました。しかし、プログラムの構成は大きく変化した。

microFORTH には、PROM から実行するように設計されたターゲット核(nucleus)が付属していた。この核は1K程度の大きさで、単精度演算などのプリミティブと、ごく基本的な機能を含んでいた。開発環境は、対話的にコードを書いてテストし、そのコードをランタイム核に適合するようにコンパイルすることをサポートしていた。`VARIABLE` は ROM と RAM のデータ空間分離をサポートし(`CONSTANT` は PROM に配置された)、定義語はユーザ定義の構造を ROM/RAM どちらにも配置できるようにした。また、従来はコンパイル時に `VARIABLE` を初期化できましたが、この機能の互換性は外されました。ROM のコンパイル時にターゲット RAM を初期化するには「シャドウ」テーブルを設定しなければ難しく、そのために ROM の領域を割り当てするには貴重すぎると判断されたためです。

マルチプログラミングのサポートは当初は取り除かれましたが、後に新しい高速なタスクスワップアルゴリズムを使って復活しました。データベースツールは完全に消滅しました。

FORTH 社は、新しいミニコンピュータの CPU で Forth を生成するために使われたメタコンパイラをリリースすることはなかった。しかし、このメタコンパイラの一種は、ターゲットアプリケーションの ROM 化コードを生成するために使われ、microFORTH の不可欠な一部となった。このことは、次節で述べるように重要であった。

2.3.3 影響を受けたもの

microFORTHの主要アーキテクトはディーン・サンダーソン(Dean Sanderson)である。サンダーソンはムーアと密接に仕事をし、彼の基本的な哲学のほとんどを共有していましたが、スタイルとアプローチの違いは避けられませんでした。しかし、主要な新しい影響は、より広範な顧客基盤からもたらされ、その結果として、microFORTHの広範なマーケティングによってもたらされた。マルチプログラミングを復活させたのは顧客の圧力であり、この大きな顧客基盤は標準化団体を形成する原因にもなりました。

2.4 言語定義

1970年代初期から中期にかけてFORTH社によって作られた商用ミニコンピュータ・マイクロコンピュータ実装は、今日使われているForthの原理と要素を初めてカプセル化したものです。このため、これらを簡単にまとめておくことにする。

2.4.1 設計原理

代数学がFORTRANの「メタファー」であったように、Forthは英語の散文をモデルとして考えられました(ただし、その後置記法はドイツ語のような動詞を末尾に置く言語に似ている傾向があると指摘する人もいます)。その要素(「単語」)は、名前付きデータ項目(乱暴に言えば名詞に相当する)、名前付き手続き(動詞に相当する)、および定義語(カスタマイズされた特性を持つデータ項目を作成できる特殊な動詞の一種)である。単語は、以前に定義された単語に基づき定義されるか、(組み込みアセンブラを用いて定義された)機械語コードで定義されます。

Forthの「単語」は、機能的には他の言語におけるサブルーチンに類似している。また、他の言語におけるコマンドにも相当します。Forthは言語的要素と機能的要素の区別を曖昧にしています。

単語は名前によって(キーボードから、またはプログラムソースから)参照される。そのため、「単語」という用語は、プログラム(および言語)単位とそのテキスト名の両方に適用されます。テキストの解析において、Forthは空白文字(一部のファイルベース・システムでは「空白類(white space)」文字)で囲まれた文字列を単語と見なします。これらを除いて、単語に含めることができない、または単語を開始することができない特別な文字はありません。多くのプログラミングチームが可読性を高めるために命名規則を採用しています。テキストで出会う単語は、定義済の単語(すなわちForthルーチン)、数字、未定義の単語の3つに分類されます。

Forth には明示的な型付け機構はありません。この特徴は、時に新参者を驚かせますが、経験豊富な Forth プログラマには一般に賞賛されています。

2.4.2 構造化プログラミングの規範

アーキテクチャ的には、Forth 語はダイクストラ[例: Dijkstra, 1969]と「モジュール式プログラミング」 [Parnas, 1971]によって明示された「構造化プログラミング」の原則に厳密に準拠している。これらの原則をまとめると、次のようになります。

すべてのプログラムは、自己完結したモジュールの線形列として記述される。モジュールは1つの入口と出口を持ち、理想的には、1組の入力と1組の出力が与えられたときに、1つの機能を実行する。

モジュールは以下を含むことができる。

- 他のモジュールへの参照
- 決定構造 (`IF THEN` 文)
- ループ構造
- トップダウン設計とボトムアップのコーディングとテスト(Forth における構造として強く奨励されています)

ムーア氏が画像処理システムで OOP のような機能を独自に開発したときと同様、ムーア氏は構造化プログラミングに関する現代の文献をよく知りませんでした。これらの原則は、1973 年にラザーが初めて彼に注意を促したもので、彼女は Forth に関するセミナーで Forth と構造化プログラミングの明らかな関係についていくつかのコメントを受け取っている。ダイクストラの論文の一つを読んだムーアは、"私には良いプログラミングの実践のように思える" と観察した。

実際、基本的な実装について知っている Forth の上級プログラマは「ごまかす」方法を知っているが、そうしたやり方は嫌われるし、言語の構造によってサポートや奨励されていないのは間違いない。

2.4.3 Forth の構成要素

1970 年代初頭のムーアの Forth システムは、わずか 4K バイトの核をベースに構築されていた。この小さなプログラムには、ディスク(またはテープ)および端末ドライバ、辞書の検索と構築の機能が含まれていました。この核を使って、アセンブラ、エディタ、マルチユーザサポート、数百の一般コマンドなど、プログラミング環境の残りをソースからコンパイルしていた。システムの起動は、ソースから実行形式へのコンパイルを含めて、わずか数秒しかかからなかった。

核のコンパイルには、同じくForthで書かれたメタコンパイラが使われた。このシステムのソース全体は約40ページだった。

これらのシステムは、ホストOSやエグゼクティブを必要としない「ネイティブ」であった。これは、OSがなかった時代には必要なことであった。その後、Forthのネイティブ環境でのI/Oサービスは、汎用OSが提供するよりもはるかに高速であるため、大きな利点とみなされるようになった。

以下、Forthの主要な構成要素について簡単に説明する。

2.4.3.1 辞書

Forthプログラムは、システムで使用されるほぼすべてのメモリを占める、拡張可能な辞書として構成される。辞書は、古典的には可変長項目のリンクリストとして実装され、各項目は単語を定義する。各定義の内容は、単語の種類(データ項目、定数、一連の演算など)に依存する。マルチユーザForthシステムでは、各ユーザがプライベート辞書を持つことができ、各辞書は共有のリエントラントシステム辞書に接続される。

2.4.3.2 プッシュダウン・スタック

Forthは2つのプッシュダウンスタック、またはLIFOリスト(マルチプログラムバージョンでは、各タスクに1組)を維持します。これらは、Forth の単語間でデータを渡すため、および論理フローを制御するために使用されます。スタックは1セルで構成され、1セルは8ビットおよび16ビットコンピュータでは16ビット幅、680×0ファミリのような32ビットプロセッサのほとんどの実装では32ビット幅である。拡張精度の数値は スタック位置2つ分を占め、最上位が一番上になります。スタック上の項目は、アドレスであったり、様々な種類のデータであったりする。スタックの大きさは不定であり、通常、メモリの低位方向に伸びる。

Forthはスタックを明示的に使用することで、オペランドが演算子の前にくる「後置記法」を採用しています。演算結果をスタックに残すため、演算を簡単に連鎖させることができ、一時保存のための変数を定義する必要がほとんどありません。

2.4.3.3 インタプリタ

Forth はインタプリタ方式で、プログラムの実行は小さな機械語ルーチン(しばしば2〜3命令のみで構成されている)によって制御され、抽象的な機械機能へのポインタ(トークン)のリストを解釈するものです。このアーキテクチャは、例えばBASICやPROLOGで使われているような古典的なインタープリタよりもはるかに高速で、当時設計対象であったリアルタイムアプリケーションでも十分な性能を発揮することができる。

この内部エンジンは、しばしば「内部」または「アドレス」インタープリタと呼ばれ、ソースやユーザ入力を処理するForthのより伝統的なテキストインタープリタとは区別されています。テキストインタープリタは、端末や大容量記憶装置からスペースで区切られた文字列を取り出し、それぞれの単語を辞書で調べます。単語が見つかったら、アドレスインタプリタを呼び出して実行する。アドレスインタプリタは、単語の定義にコンパイルされたアドレスの列を処理し、それぞれが指す定義を実行することで単語を実行する。テキストは凝縮されたものも含めてメモリには保存されない。単語が見つからない場合は、数値として変換し、スタックにプッシュしようとする。数値変換に失敗した場合(非数字の文字のため)、インタプリタはエラーメッセージを表示して中止する。

アドレスインタプリタには2つの重要な性質がある。まず、高速であることです。これは、アドレス1つにつき必要な機械語命令が1〜2個と少ないためである。2つ目は、Forthの定義が非常にコンパクトになることです。これは、個々の参照が1つのセル(またはコンピュータ・ワード。「ワード」がForth言語の要素を示すため、Forthユーザはハードウェア・ユニットとしての「ワード」の使用を避ける傾向があります)しか必要としないためです。これに対して、多くのコンパイラが構築するサブルーチン・コールは、`CALL`または`JSR`命令とアドレスの前後に、サブルーチン内のレジスタを保存および復元する呼び出しシーケンスを処理する命令が通常必要となります。Forthのスタック・アーキテクチャは、明示的な呼び出しシーケンスを必要とせず、ほとんどの実装では、特定のシステム状態変数を専用レジスタに割り当て、その他のすべてのレジスタをコードワードで使用するためのスクラッチレジスタとして指定するグローバルなレジスタ割り当てを行います。

2.4.3.4 アセンブラ

ほとんどの Forth システムには、実行する CPU 用のマクロアセンブラが含まれています。`CODE`を使用する場合、プログラマは他のアセンブラと同様に CPU を完全に制御し、`CODE`定義は完全なマシンスピードで実行されます。アセンブラを使用すると、プログラマはCPUに依存するコードを明示的に、マシンに依存しないインターフェース規約で管理可能な部分の中で使用することができます。アプリケーションを別のプロセッサに移行するには、`CODE`ワードのみを再コード化する必要がありますが、これは他のForthワードと全く同様に相互作用します。

Forthアセンブラは、次の2つの目的を持ったユニークな設計になっています。

1. プログラマによるプロセッサの制御を損なうことなく、アセンブラ表記をできるだけ標準化し、プロセッサ間の移植性を向上させること、および
2. 常駐可能なコンパクトなアセンブラを実現し、対話的なプログラミングやデバッグを容易にすること。

古典的なForthアセンブラでは、オペコード自体がForthの単語であり、スタック上に渡されたオペランドによってアドレス情報を与えながら命令を組み立てていく。このため、アドレッシング・モード指定子はオペコードの前に置かれる形式となる(Forthの他の場所で使用されているポストフィックス表記と一致する)。ムーアはアドレッシングモードの表記法も標準化したが、通常はCPU製造業者の命令ニーモニックを使用していた。レジスタは、重要な内部システム機能に割り当てられたレジスタを除き、一般に番号で呼ばれました。例えば、スタックポインタは通常Sというレジスタにあり、2バイト幅のスタックの2番目の項目を2S)というフレーズでアドレス指定する。

Forthアセンブラは、高レベルForthと同じように構造化プログラミングをサポートしています。ラベルの付いた場所への任意の分岐は推奨されません。一方で`BEGIN` ... `UNTIL`や`IF` ... `ELSE` ... `THEN`といった構造がアセンブラで利用できます(適切な条件分岐と無条件分岐を組み立てるマクロとして実装されています)。このような構造は、アセンブル時にスタックがアドレス情報を運ぶために利用できるため、実装が容易である。

従来のアセンブラは、コードをファイルに残し、リンカで高級言語コンパイラのファイルと統合してから、プログラムをメモリにロードしてテストする必要がありました。Forthの常駐型アセンブラは、コードを実行可能な形でメモリに直接アセンブルするため、リンクのステップを省くことができます。

Forthアセンブラは、高レベルのForth語のように機能する短い名前付きルーチンを書くために使用されます：ルーチンの名前が呼び出されると、それが実行されます。他の Forthルーチンと同様に、コードルーチンはスタック上の引数を期待し、その結果をそこに残します。コード内でプログラマは、定数(値を得るために)、変数(アドレスを得るために)または他の定義されたデータ型を参照することができます。コードルーチンは、他のForth単語と同様に高レベルの定義から呼び出すことができますが、それ自身は高レベルまたはコード定義を呼び出すことはありません。

これらの機能により、Forthプログラマは短く、簡単にテスト可能なモジュールとしてコードを書くことができ、それは自動的にアプリケーションに統合されます。プログラミングは完全に構造化されており、アセンブラと高級プログラミングの両方で一貫した使用ルールとユーザインターフェイスを備えています。単語は、プログラマの頭の中にある望ましい動作が新鮮である間に、段階的にテストされます。ほとんどの新しい単語は、入力値をスタックに置き、テストする単語をタイプし、スタックに残された結果を表示して検証するだけで、テストすることができる。

その結果、コンピュータの完全な制御、必要な部分では高性能化でき、そしてあらゆるレベルで対話型プログラミングを行うことにより、全体的な開発期間の短縮が実現される。

2.4.3.5 ディスクのサポート

古典的なForthは、大容量記憶装置を1024バイトずつの「ブロック」に分割しています。このブロックサイズは、セクタサイズが異なるディスク間で便利な標準として選択されました。少なくとも2つのブロック・バッファがメモリ内に維持され、ブロック管理アルゴリズムは、すべてのブロックが常にメモリ内にあるように見えるようにします。`n BLOCK`コマンドは，ブロックnのメモリアドレスを返し，必要ならそれを読み出す。内容が変更されたバッファにはマークが付けられ、再利用の際には自動的にそのブロックが書き出されるようになっている。このアルゴリズムは、データとソースの保存に便利な仮想メモリを提供し、必要な物理ディスクのアクセス回数を最小限に抑えます。FORTH社のデータベースアプリケーションは、ブロックからデータファイルを構築し、ファイルは指定されたブロックの範囲にまたがるものとして定義される。データアクセスは、選択されたファイル内の名前付きフィールドに対して行われる操作を通じて行われる。

ネイティブForthでは、ディスクドライバがブロックの番号から物理アドレスを計算するため、ブロックシステムは高速で信頼性が高く、ディレクトリは必要ありません。ディスクを多用するアプリケーションでは、バッファを追加することで性能を向上させ、より多くのブロックをメモリ内で見つけることができます(バッファはディスクキャッシュになります)。

1980年代には、後述するように、従来のOS上でForthシステムが動作するようになった。これらの多くはホストOSのファイルの内部でブロックをサポートしているが、中にはブロックを完全に放棄したものもある。ブロックは、ネイティブシステムと非ネイティブシステムの両方で大容量記憶装置にアクセスするための互換性のある手段を提供するので、ANS Forth(セクション5.1)は、大容量記憶装置のサポートが利用可能であればブロックを利用できるようにすることを要求しています。

2.4.3.6 マルチプログラミング

初期のForthシステムは、コンピュータが複数のプログラムシーケンスを同時に実行できるという点で、マルチプログラミングをサポートしていました。1973年、ムーアはこの機能を拡張し、それぞれが端末と独立したサブディクショナリとスタックを持つ複数のユーザをサポートするようにした。このようなプログラム列の1つを実行したり、ユーザをサポートするエンティティをタスクと呼ぶ。今日のForthの多くはマルチプログラミングをサポートしており、これらのほとんどはムーア氏のアプローチの変種を使用している。

この方法は、協調的なノンプリエンプティブアルゴリズムを使用してCPU時間を割り当てるものです。タスクはI/O操作の完了を待つか、`PAUSE`という単語を使用するとCPUを放棄する。放棄する時間はラウンドロビンタスクキューをちょうど1周する間となる。

ムーアのシステムでは、I/Oに割り込みが使われていた。割り込みは、Forthエグゼクティブの介入なしに、アセンブラマクロを使用してレスポンスコードに直接ベクトル化された。割り込みコードは、最もタイムクリティカルな操作(例えば、数値の読み取りやカウンタのインクリメントなど)だけを実行し、その後、割り込みを待って中断されていたタスクを再び有効にしました。このタスクは、次にラウンドロビンタスクループに入ったときに動作を再開し、そのときにイベントによって発生した高レベルの処理を完了し、作業を続行することになります。

理論的には、このノンプリエンプティブアルゴリズムは、論理的または計算集約的な活動でCPUを独占するタスクに弱いのですが、実際にはリアルタイムシステムはI/Oに支配されているので、これが問題になることはほとんどありません。CPUに負荷のかかる処理が発生した場合、`PAUSE`は性能を「調整」するために使用されます。

Event	VRTX	OS9	PDOS	polyFORTH
Interrupt response	91	43.75	93.4	7.0
Context switch	128	186.25	93.4	36.0
Suspend task	180	316.25	184.7	6.8
Copy memory (80 bytes)		212.5		97.0

Table 2. Performance comparisons of several real-time OSs on a M68010 [Cox, 1987]. 時間は平均値で、単位はμsです。polyFORTHはノンプリエンプティブタスクスケジューリングを採用しているため、その性能の優位性は明らかです。

コンサルタントのビル・コックス(Bill Cox)は、このようなノンプリエンプティブアルゴリズムにはいくつかの利点があると指摘しています[Cox, 1987]。まず、タスクスケジューラ自体が単純化され、1タスクあたり1マシンインストラクションで済むため高速化される。第二に、タスクは既知で明確に定義された時間でのみ中断されるため、保存・復元すべき「コンテキスト」が少なくなり、コンテキストスイッチ自体が高速化される。第三に、タスクがCPUを制御するときとしないときを正確に把握した上でタスクコードを書くことができ、共有資源の管理がかなり簡素化される。Coxはいくつかのリアルタイム OSの性能を比較し、その結果を表2に示した。

タスクはシステム起動時に構築され、各タスクには実行する機能に応じた固定メモリが割り当てられた。システム再起動の時間は数秒であり、タスクの再構成は容易であったと言える。

2.4.3.7 数値計算

1970 年代末まで、浮動小数点演算を提供するミニコンピュータはほとんどなく、実際、多くは ハードウェア乗除算器を備えていませんでした。しかし、当初からForthは計算を多用する仕事に使われていた。例えば、電波望遠鏡の制御では、天体の位置を示す天球座標から方位/仰角座標系への変換を1秒間に1回、中間位置を1秒間に5回補間し、データ取得とオペレーターの作業を同時進行で行う必要があった。

ムーアのアプローチは、Forthに整数を効率的に操作する機能を組み込むことだった。例えば、*/というコマンドは、2つの単精度整数を掛け、3つ目の整数で割って、倍精度整数の中間積を生成する。これは、ほとんどの乗算、除算の機械命令の動作方法を反映しており、次のような計算が可能である。

12345 355 113 */

これは12345に355/113という比を掛けており、この比はPIを8.5×10-8の誤差で表現している[Brodie, 1981]。比率による乗算は、合理的な近似だけでなく、較正やスケーリングにも理想的である。同様に、`/MOD`は単一の除算を実行し、商と余りの両方を返します。このような単精度、倍精度、混合精度の演算が豊富にあるため、整数演算は多くの言語で使われているものよりずっと使い勝手がよい。

Mooreは内部的に角度を14ビット、15ビット、30ビットの固定小数点バイナリ分数で表現しました。彼は、角度フォーマット(例えば、dd:mm:ss)との間で変換するプリミティブのセットと、これらのフォーマットに対する超越関数をサポートする数学ライブラリを、主に [Hart 1968] のアルゴリズムに基づいて提供しました。高速フーリエ変換のような演算は、複素数をスケーリングされた整数のペアとしてサポートする特殊なプリミティブに基づいて、一部のアプリケーションで提供されていました。

今日、高速な浮動小数点プロセッサは一般的です。ANS Forthのように、多くのForthが浮動小数点をサポートしています。しかし、単純なマイクロコントローラ上の組み込みシステムのような多くの場合、Forthの整数演算は今でも単純で高速なソリューションを提供します。

2.4.3.8 データ型

おそらく、ムーア氏の個人的な哲学が最もよく表れているのは、データ型付けに対するアプローチでしょう。基本的に、彼はデータオブジェクトを好きなように操作することに全責任を負いたいと考えていた。この点について追及されれば、彼は「もし、私がAという文字に1を加えたいのなら、コンパイラがそれをできないなどと言う筋合いはない」と言うだろう。

Forthの標準語は、スタック上の値を返す単精度および倍精度の`CONSTANT`と、ポインタを返す`VARIABLE`をサポートしています。`CREATE`は、スペースを確保できるデータ領域の先頭を指定します。`CREATE`された実体が返すポインタは、配列へのインデックスとしてインクリメントすることができます。定数や変数に保持される値の性質は完全に任意であり、通常、明示的な型チェックは行われません。文字列は通常、その長さを最初のバイトに格納してメモリ上に保持されます。この構造体のアドレス、あるいは実際の文字列のアドレスと長さは、スタック上で渡すことができる。

`CONSTANT`、`VARIABLE`、`CREATE`は「定義語(defining words)」、つまり特徴的な動作をする新しい単語を定義するものです。Forthは、プログラマが新しい定義語を構築するためのツールも提供しており、コンパイル時(テーブルの設定や初期化など)と実行時(インデックスを受け取って構造体のベースアドレスに自動的に適用するなど)の両方でカスタム動作を指定することができます。

3. チャック・ムーアなきForth

microFORTHは、70年代後半に盛んにマーケティングされ、多くの注目を集めました。その副次的な効果として、Forthに恋する活発で熱狂的なホビイストのグループが成長したのです。彼らの後を追うように、FORTH社に対抗してForthのバージョンを販売する新しい会社が現れた。同時に、ムーア自身もForthのハードウェア実装に惹かれるようになり、FORTH社でのソフトウェア制作にはあまり関与しなくなった(1982年にハードウェアに専念するために退社)。このセクションでは、このような様々な新しい影響の下でのForthの発展について考察しています。

3.1 The Forth Interest Group

1970年代後半、北カリフォルニアは、コンピュータ革命の初期に沸き返った。"ホームブリューコンピュータクラブ"のような興味を持った人たちのグループが集まり、興味や経験を共有しました。「ラジオ・エレクトロニクス」などの雑誌には、ビデオ・ディスプレイ・ターミナルの作り方や、マイクロコンピュータ・システムの作り方などがステップバイステップで掲載されていた。

メモリが高価で、VLSIの集積度が低かったため、典型的な「自作(homebrew)」コンピュータは、非常に資源制約の多い環境であった。第一世代のコンピュータに遡ると、エディタ、アセンブラ、リンカを同時にサポートするにはメモリが不足していた。大容量記憶装置は遅くて高価なため、多くの自作システムでは入出力に紙テープやオーディオカセットテープを使っていた。BASIC言語の製品もあったが、一般に非常に遅く、重要なプログラムをサポートすることはできなかった。こうして、懸命な探検家や「アーリーアダプター」の拡大するニーズに応えるために、何か別のものが必要な段階が整った。

Forthは、資源に制約のあるシステムの最小限の設備を利用するために生まれ、育てられた。Forthは、一般的なソリューションのような過剰な荷物も、既存のファイルやオペレーティングシステム、大容量の記憶装置も必要としなかった。Forthがより困難な組み込みコンピュータのアプリケーションに使用されるようになると、北カリフォルニアの自作コンピュータ愛好家の注目を集めるようになった。

ベイエリアのセキュリティシステムメーカーで成功したBill Ragsdaleは、microFORTHの利点に気づき、1978年にFORTH社に6502用のmicroFORTHのバージョンを作るように依頼しました。FORTH社は、より人気のある8080、Z80、6800のCPUに比べ、6502上のmicroFORTHの市場需要ははるかに少ないと考え、これを拒否しました。

そこで、ラグスデールは、microFORTHの知識と6502に精通し、microFORTHを6502に移植できる人を探した。彼は、陸軍のプロジェクトでAMI 6800開発システムにmicroFORTHを使っていて、個人的に6502用のスタンドアロンエディタ／アセンブラ／リンカパッケージを開発していたロバート・セルザー少佐(Maj. Robert Selzer)を見つけた。セルザーは6502 Forthアセンブラを書き、陸軍のmicroFORTHメタコンパイラを使って、Joltシングルボードコンピュータ用の最初の6502スタンドアロンForthをターゲットコンパイルしたのです。

セルザーとラグスデールはその後、6502のページゼロとスタックインプリシットアドレッシングのアーキテクチャ上の特徴を利用するなど、このモデルに大幅な修正と改良を加えました。後のパブリックドメインのバージョンを特徴づける多くの拡張は、可変長の名前フィールドや辞書リンクリストスレッディングの修正など、この時期に行われました。Jolt

上のメタコンパイラは、大幅に変更されたカーネルをメモリ内の上位アドレスをターゲットに構築することができました。そして、ブート可能なイメージは、新しいカーネルによって低いブートアドレスに再コンパイルされ、置き換え用としてディスクに書き出されることになります。この時点で、ラグスデールは、組み込みセキュリティシステムという専門的なニーズに対応するシステムを手に入れた。

この時期、Forth Interest Group (FIG) が、ラグスデール、キム・ハリス、ジョン・ジェームズ、デーヴィッド・ボールトン、デイブ・ベンゲル、トム・オルセン、デイブ・ワイランドによって始められました [FIG 1978]。彼らは、一般的なコンピュータアーキテクチャに実装可能な、公に利用可能なForthシステムである「FIG Forthモデル」という概念を導入しました。

FIG Forth Modelは、ラグスデールの6502システムから派生したものです。多種多様なアーキテクチャに迅速に実装し公開を単純化できるようにするため、Forthメタコンパイラのソースコードを、標準的な6502アセンブラに入力すると、元のカーネルイメージを再現するテキストに変換するトランスレータが作成されました。このようにすることで、メタコンパイラもそのソースコードも公開する必要がなくなりました。ここが重要なポイントだ。Forthのメタコンパイルは、完全に理解するのが難しい処理である。3つの異なる実行フェーズとオブジェクト領域を直接操作する必要があり、カジュアルなユーザが望むものでも、必要とするものでもないのだ。

Forth Interest Groupは、アセンブラのリストを公開することで、Forthのランタイム環境を、異なるコンピュータ・アーキテクチャのアセンブリ言語に容易に複製および/または翻訳できる方法でカプセル化することができた。このようにして、互換性のあるForthシステムの開発や、Forth製品の新しいベンダーの出現を促進することが、実装者のオリジナル・チームの意図でした。

6502のFIGモデルが発表された後、FIGの実装者は8080と6800のマイクロコンピュータ、PDP-11とComputer Automationのミニコンピュータ用の互換バージョンを発表しました。何年にもわたって、有志が他のプラットフォームやドキュメントを追加していきました。1982年のMitch DerickとLinda BakerによるForth Encyclopedia [Derick, 1982]は、FIG Forthに関する333ページの完全なマニュアルで、ほとんどの単語のフローチャートが掲載されています。1983年、FIGのニュースレターであるForth Dimensionsの広告 [FIG, 1983]には、以下のようなものが掲載されていました。RCA 1802, 8080, PACE, 6502, 8086/88, 6800, 6809, 9900, Nova, Eclipse, VAX, Alpha Micro, Apple II, 68000, PDP11/LSI11, Z80。

現在、Forth Interest Groupのメンバーは15カ国以上、数千人にのぼります。1980年以来、FIGはFORML(Forth Modification Laboratory)と呼ばれる年次会議を主催しており、Forthに役立つことを意図した新規または未実証の提案を共有し議論する教育フォーラムであり、Forthの技術的側面に関する議論も行われています。その議事録は、Forth Interest Groupから入手できます。

3.2 パーソナルコンピュータ用の商用およびパブリックドメインシステム

アップルコンピュータは、サンフランシスコのベイエリアにおけるコンピュータへの熱狂から生まれ、資源に制約があるが、全く新しい世代のコンピュータを生み出しました。BASICはROMで入手できたが、Forthは Apple][で多くの人気テキストエディタやゲームを書くのに使われ、その少ないメモリとディスクの制約の中で重要なプログラムを常駐開発することを可能にした。メモリとディスクが何メガバイトもある現在、16Kのメモリと100Kのディスクストレージ上で40カラム幅の画面を使い重要なプログラムを開発することがどんなものであったかを想像するのは難しいことです。

低価格のForthシステムのベンダーが一夜にして現れ、それぞれがお気に入りのパーソナルコンピュータをサポートし、そのほとんどがFIGモデルをベースにしたシステムでした。例えば、1979年に Miller Microcomputer Services は TRS-80 用の MMSFORTH を発表し[TRS-80, 1979]、1980年までに Computerworld は MMS がその製品に対して100以上のユーザグループを持っていると報告した [Taylor, 1980]。

IBMが最初のPC製品を提供してパーソナルコンピュータビジネスに参入したとき、彼らはForthで書かれた人気のあるApple][テキストエディタEasyWriterのバージョンをIBM製品として配布することを選択した。ラボラトリー・マイクロシステムズ(LMI)は1982年に商用のIBM PC Forthシステムを発表した。その後、多数の商用およびパブリックドメインのForth製品が登場し、大規模なソフトウェア製品開発が始まりました。

IBM-PC用の最初の商用Forthの導入後、LMIは、32ビットリアルモード実装(1983年2月)、OS/2ベースのForth(1988年2月)、Windows版(1992年)など、PC用の最先端のForthシステムを生産する戦略を継続した。LMIの創設者であるレイ・ダンカンは、マイクロソフトOSの権威として知られるようになりました[e.g., Duncan, 1988]。

FORTH, Inc.のPC版はpolyFORTHであり、ミニコンピュータ製品のマルチユーザサポートとデータベースツールをmicroFORTHのROM化されたアーキテクチャと組み合わせたものであった。FORTH社は1984年までに、1台のPCで最大16人のユーザをサポートし、

polyFORTH を最初はネイティブ OS として、後には MS-DOS とのコ・レジデント OS として動作させるようになりました。1980 年代後半には、NCR のような polyFORTH のユーザは、 80386 ベースの 1 台の PC で 150 人ものユーザをサポートするようになりました。

1978 年、セルツァー少佐は、コルバーンがセルツァーの 6502 の仕事について 2 つの記事を書く代わりに、彼がラグズデールに書いた 6502 Forth のコピーをドン・コルバーンに渡しました。その後、コルバーンは、完成前の FORTH-77 標準に基づくバージョンの基礎としてこれを使いました(著者が知っている唯一の FORTH-77 実装です)。1979 年の秋、コルバーンは 68000 のプロトタイプのために FIG 互換のシステムを生成した。MultiForth と呼ばれるこの製品のマルチタスク、マルチユーザ版は、68000 の生産出荷にかなり先駆けて、1980 年 1 月に Motorola にデモされた。1982 年にヒューレット・パッカードのデスクトップコンピュータ部門が 68000 を中心に新世代のデスクトップコンピュータを設計したとき、HP の部品番号で配布された最初のサードパーティ言語製品は MultiForth だった。

コルバーンの会社クリエイティブ・ソリューションズも、1984 年 1 月の Mac のデビュー直後に、128K Apple Macintosh 用の最初の常駐型開発システムである MacForth を発表している。MacForth は、Macintosh の「Toolbox ROM」ルーチン全体に常駐型プログラミング環境で直接アクセスできるユニークなシステムであり、包括的なアプリケーション例とともに、Macintosh アプリケーションプログラマの第一世代の大半が MacForth でプルダウンメニュー、ウィンドウ、グラフィック、マウスの作成方法と使い方を習得したのである。初期の Macintosh では、大容量の表計算ソフト、2D および 3D のレンダリングとデザインパッケージ、CAD/CAM デザインツール、ゲーム、医療診断、画像補正プログラム、会計パッケージ、デスクトッププラネタリウム、プロセス制御アプリケーションなどが MacForth で作成されました。

1985 年までには、個人から数百万ドル規模の組織まで、70 社以上の Forth システムのベンダーが存在しました。

1982 年、ローレンス・フォースリーは Institute for Applied Forth Research(現在では単に Forth Institute と呼ばれている)を設立した。この組織は、ニューヨーク州ロチェスターのロチェスター大学で毎年開催される Forth アプリケーションに関する会議を主催し、Forth のアプリケーション、新しい開発および技術、Forth の特定分野の調査に関する査読付き技術定期刊行物である Journal of Forth Application and Research を出版しています。

1989 年には、George Shaw らが SIGForth という Forth に関する ACM Special Interest Group を結成し、ニュースレターや年次カンファレンスも主催しています。

System(s)	Company	Primary Products & Markets
CFORTH83, Forthmacs, SunForth	Bradley Forthware	C で書かれたポータブル Forth、Atari、Macintosh、Sun 用バージョン。Sun Microsystems Open Boot 関連のコンサルティングとサービス。
cmFORTH	Silicon Composers	C. Moore による Novix Forth およびその他のプロセッサ用のパブリックドメインシステム、他の人々によって Harris および SC-32 Forth プロセッサに移植されたもの。
Cyrano	Opto-22	独自の組み込みコントローラ用Forth
F-PC	T. Zimmer 他	IBM-PC ファミリ用の広範なパブリックドメインシステム
F83	Laxen and Perry	IBM-PC ファミリ用のパブリックドメインシステム、後に他のプラットフォームへ移植
HS/Forth	Harvard Softworks	IBM-PC ファミリ
JForth	Delta Research	Amiga
MacForth	Creative Solutions, Inc.	Apple Macintosh、NuBusインターフェースボード
Mach2	Palo Alto Shipping	アップルマッキントッシュ
mmsFORTH	Miller Microcomputer Services	IBM-PCファミリ、ビジネスおよび商業用アプリケーション
MPEForth	MicroProcessor Engineering (UK)	PCおよび組み込みシステム
mvpFORTH	Mountain View Press	様々なプラットフォーム上のパブリックドメインシステム
Open Boot	Sun Microsystems	SPARCワークステーション上のプログラマブルROMベースFORTH
polyFORTH	FORTH, Inc.	PCおよびその他のプラットフォーム上の産業用システム、対話型クロスコンパイラ、 コンサルティングおよびカスタムプログラミングサービス

System(s)	Company	Primary Products & Markets
UR/Forth	Laboratory Microsystems, Inc. (LMI)	DOS、OS2、Windows が動作する IBM-PC ファミリ、各種システム用クロスコンパイラ。

Table 3. Some major suppliers of Forth systems, services and related products.

Byte 誌は 1980 年 8 月号で Forth を特集した。この号は、これまでで最大の売り上げを記録し、何度も再版された。

3.2.1 設計原理

FIG Forth は、性能よりも移植性に最適化されています。ごく少数のプリミティブだけがアセンブラでコード化され、残りのロジックは高水準の Forth を使って実装されていました。その結果、動作はかなり遅く、辞書検索などの一部の操作は、代表的な商用実装よりも 10 分の 1 ほど遅くなってしまいました。

その他、内部的な決定も同様に、初心者を意識したものであった。例えば、初期の FORTH 社製システムでは、単語名を名前の長さと最初の 3 文字でコンパイルしていた。これは、単純な切り捨てよりも衝突率が低く、ほとんどの場合において適切であった。しかし、FIG では 31 文字までの可変長の名前を使うことで、サイズと使い勝手の良さを両立させました。これは、当時は多少議論を呼んだが(図 1 参照)、1980 年代半ばには、ほとんどのシステムがこの使い方に転換していた。

パーソナルコンピュータの登場は、Forth の実装者にホスト OS の下で動作することを学ぶ動機を与えた。最初のノンネイティブ・システムは、1980 年に Micromotion 社の Martin Tracy 氏(Apple][用)と Laboratory Microsystems 社の Ray Duncan 氏(Z80 上の CP/M 用)によって開発されました。LMI のシステムはフルスクリーンエディタも備えていた。1981 年、LMI はソフトウェアおよびハードウェア浮動小数点のサポートを追加し、またネイティブコード変換や辞書検索を高速化するためにハッシュテーブルに辞書検索をキャッシュするなどの性能強化のパイオニアとなった。

非ネイティブの Forth 実装の登場は、今日でも Forth の実践で論争の的になっている、大容量記憶装置にホスト OS のファイルを使用する問題を導入した。従来のブロックを捨てて、ソースやデータを直接ファイルで操作する方法と、ブロックをホスト OS のファイルにマッピングする方法の 2 つが主流です。前者は、特定の OS(例えば MS-DOS)用のシステムに集中している実装者に好まれ、後者は、FORTH 社のように、ネイティブとノンネイティブの両方の製品をサポートしている団体に好まれるアプローチである。

クリエイティブ・ソリューションズの MacForth は、オリジナルの 128K Macintosh でプログラム開発に利用できるメモリ量を最大化するため、トークンスレッディングやネームヘッドの分離など、非常にコンパクトなオブジェクトイメージ戦略を採用していました。また、実行イメージのランタイムリロケーションや、ランタイムシステムにおいてメタコンパイルを行わずにワードネームを除外する機能など、斬新な機能を備えていました。MacForth は、画面ベースのテキストエディタ、コンパイラ、インタプリタ、アセンブラを 20k バイト以下のメモリに組み込んだシームレスなプログラミング環境を提供した。

3.2.2 それによる影響

FIG モデルはパブリックドメインであり、さまざまなコンピューターシステムに移植されました。FIG Forth の内部設計は基本的にすべてのマシンで同じだったため、FIG Forth で書かれたプログラムは、辞書項目や他の実装依存の機能の内部を直接操作する「システムレベル」プログラムであっても、かなりの程度の移植性を享受していたのです。FIG Forth は、多くの人にとって Forth の最初の入門書であったため、"Forth らしさ" に関連づいています。

しかし、FIG Forth は、この時代のすべての商用実装を代表するものではありませんでした。商用ベンダは、これまで見てきたように、性能を重視し、移植の容易さよりも性能やサイズを最適化する実装戦略を選択する傾向が強かった。

FORTH を標準化するための最初の大きな努力は、1977 年にユトレヒトで開かれた会議で、天文系の FORTH ユーザ数人と FORTH 社(当時は唯一の商用ベンダー)が参加したことです。彼らは FORTH-77 と呼ばれる予備的な標準を作成し、翌年も会合を開くことに合意しました。1978 年と 1979 年にカリフォルニアのカタリナ島で行われた会議では、FORTH Interest Group と他の製作者の代表が参加し、FORTH-79 と呼ばれるより包括的な標準が作成されました。FORTH-79 は非常に大きな影響力を持ちましたが、多くの Forth ユーザとベンダーはそれに含まれる欠陥に気づきました。1982 年に 2 つのミーティングが開かれ、標準規格をアップデートし、1983 年に FORTH-83 という新しい規格がリリースされました。FORTH-79 と FORTH-83 はともに、16 ビット、2 の補数、アライメントなしのリニアバイトアドレスの仮想マシンを規定し、実装技術に関する多くの仮定を含んでいました。

```
DEA- EDI---
I AM AFR--- THA- THE LET--- IN THE LAS- ISS-- ABO-- FOR-- INC--
HIS LET--- ( LIK- THI- ONE ) SHO-- THA- SAV--- ONL- THR-- LET--
WE STI-- DON- SEE THE NEE- FOR 31 CHA------ NAM-- IN THE GEN---


YOU-- TRU--
```

```
CHU-- MOO--
FOR-- INC-
```

Figure 1. “Letter to the Editor” of Forth Dimensions [Moore, 1983] concerning the practice of storing names of Forth words as a count and first three characters.

残念ながら、FORTH-83のいくつかの変更は、既存のコードとの重大な非互換性を生み出しました。例えば、"true"フラグの正式な表現は常に1でしたし、NOTという単語はブール値フラグを反転させるものでした。FORTH-83では"true"は-1になり、NOTはビット単位の補数になっています。その他にも、FORTH-83の床付き除算の仕様や、ループ構造のパラメータの仕様に深刻な曖昧さがあった。これらの非互換性の影響は、賛否両論を巻き起こしました。ほとんどの実装者は、FORTH-83が改善されたことに同意し、新しい標準を採用しましたが、転向しなかった人々や、標準化プロセス全体に懐疑的な人々も存在します。例えば、表3に挙げたシステムのうち、ほとんどはFORTH-83とかなりの互換性を持っています。注目すべき例外はMacForth、mmsFORTH、mvpFORTHで、これらはすべてFORTH-79にとどまっています。

1981年、Prentice Hall社は当時FORTH社の社員であったLeo Brodie [Brodie, 1981]によるStarting FORTH を出版しました。Starting FORTHは、明晰で、かつ面白い(ブロディは重要なFORTHプリミティブを印象的な漫画で表現した)、この言語への徹底的な入門書であった。この本は11万部以上売れ(一時期、Prentice Hall社のコンピュータ部門でベストセラーになった)、初めてForthを知る多くの人々や、それとの互換性を求めて躍起になっているベンダーに強い影響を与えた。初版は、主にFORTH社のpolyFORTHをベースにしていたが、FIG Forthや他の方言での脚注や例題を多く含んでいた。第2版(1987年)は、FORTH-83規格をベースにしている。

パーソナルコンピュータ市場でもう一つ大きな影響を与えたのは、Forthのパブリックドメイン版と商用版の競争である。1980年代半ば、FIGモデルは徐々にパブリックドメインのF83(Henry Laxen、Mike Perryらが「No Visible Support Software」という名前で活動して作成しました)、もともとIBM-PCでリリースされたマルチタスクシステムに取って代わられました。多くの独立したプログラマによって、他のさまざまなプラットフォーム用のバージョンが開発されている。このシステムは非常に広く普及しているため、その名前からFORTH-83規格と混同する人が多いようです。実際、FORTH-83とほぼ互換性があるものの、F83はその機能において、限られたFORTH-83規格をはるかに超えている。1980年代後半には、Tom ZimmerらがF-PCというさらに大規模なPC用のパブリックドメインシス

テムを作成しました。これは数メガバイトのソースコードとユーティリティを含んでいます。しかし、これらを除けば、ほとんどのパブリックドメインのForthは、かなり限定的なものです。

パブリックドメインのForthは、確かにForthが広く知られるようになるのに貢献しました。しかし、その影響力はいいことづくめではありませんでした。Embedded Systems Programming Magazineの編集者であるTyler Sperryによれば[Sperry, 1991]。

> 問題は、自分自身の最小限のForthシステムを比較的簡単に実装できることです。カーネルは結局のところ、数百バイトのコードに過ぎないのだから.....。残念ながら、Forthインタプリタを立ち上げることはSmall Cコンパイラを書くようなもので、よくできたライブラリがなければおもちゃに過ぎない。パブリックドメインやシェアウェアの最大の問題は、ライブラリが部分的にしか完成しておらず、ドキュメントも大雑把であることが多いことです。これでも状況を控えめに言っただけです。

限られたパブリックドメインのForth実装しか見たことも使ったこともない人たちは、Forthそのものがおもちゃだと感じてしまうことが多いのです。そして、高品質の商用システムの供給者は、見込み客が「Forthはどれも同じだ」と思い込んでいることに対処しなければならない。この思い込みは、パブリックドメインのバージョンが極めて安価であることから、当然、価格面での抵抗を生むことになる。しかし、Forthのコミュニティでは、"When you've seen one Forth ... you've seen one Forth" というジョークがよく言われます。コードやドキュメントの質、ライブラリの性質や範囲、製品サポートなど、その幅は非常に広い。見込みユーザは、パブリックドメインと商用製品の両方を数多く評価することをお勧めします。

3.3 組み込みシステム

3.3.1 環境とアプリケーション

Forthは限られたハードウェア資源を最大限に活用する能力があるため、マイクロプロセッサの組込み用途に適しています。心拍の詳細な波形分析を行う RCA 1802 ベースの心臓モニタ(1979) は、異常の記録に使用した 1 インチ×2 インチのテープカセットと大差ない大きさでした。1980年代初めにロッキード社がC5B航空機の翼のパネルを成形するために使用した750トンの圧延プレスのような大型のものもあった。また、サウジアラビアのリヤドにあるキング・ハレド国際空港の大規模な施設管理システムに使用された約500台のネットワーク接続されたプロセッサのように、分散型もありました[Rather, 1985]。Forthは特

にItronやMSIデータといった企業が製造するハンドヘルド機器のファームウェアの開発で成功を収めた。1990年、Federal Expressはその荷物追跡システムで権威あるMalcolm Baldridge品質賞を受賞したが、そのデータ入力はFederalの世界中の5万人の配達人と代理人が携帯するForthベースのハンドヘルドデバイスによって行われた。

Forthは極めてモジュール性が高いため、徹底的かつ系統的なテストが可能であり、高い信頼性が要求されるアプリケーションに魅力的であった。その結果、多くの人工衛星やスペースシャトルの実験に使われるようになりました。McDonnell Douglas は polyFORTH を彼らの Electrophoresis in Space プロジェクト [Wood, 1986] で貨物室の工場自体(複数の68000 VME バスボード)、宇宙飛行士の制御コンソール(ラップトップ PC)、地上の分析コンピュータ(Compaq PC)を制御するために使用しました。1990年11月のコロンビア・シャトル便は4つの天文学のペイロードを搭載し、そのうち3つはForthでプログラムされました [Ballard 1991] 。1992 年 1 月のスペースラブ便は、オンボード制御と分析に polyFORTH システム [Paloski, 1986] と分析用に地上の Macintosh に MACH2 を使った微少重力前庭調査 (MVI) 実験を特集しました。

組み込み Forth の最も多量な供給者はおそらく Sun Microsystems で、その SPARC ワークステーションはすべて Mitch Bradley とその仲間によって開発された Open Boot と呼ばれるプログラム可能な Forth ベースのモニタを使用しています。Bradley は、Forth がこの目的のために成功したのは、Forth が以下を提供したからだと考えています [Bradley, 1991]。

1. バイトコード化されたポータブルドライバに使用する、CPUに依存しない「仮想マシン」。
2. ドライバをデバッグするための環境。
3. ハードウェアの立ち上げやデバッグに便利な、完全なプログラミング言語機能を持つ対話型コマンド言語。
4. ファームウェア自体のデバッグ環境の構築(ファームウェアはデバッグに手間がかかる)。
5. オペレーティングシステムソフトウェアのデバッグ環境。
6. 拡張性があり、新しいハードウェアの要求や機能を容易にサポートできる。
7. スピードとスペースのトレードオフのために実装を調整する大きな柔軟性。

少なくとも他の1つの主要なボードレベルCPUベンダーは、その製品ライン全体でOpen Bootファームウェアを採用しており、そのためのIEEE標準を開発するワーキンググループも存在する。

3.3.2 設計原則

70年代のミニコンピュータから80年代のPCまで、ほとんどのForthシステムは、完成したアプリケーションを実行するのと同じコンピュータ上での開発をサポートしてきました。70年代後半から80年代前半のマイクロプロセッサ・システムでさえ、同じCPU上で開発し(クロス開発とは異なります)、開発ツールを剥がし、ROM化できるターゲットを生成するための開発ソフトウェアの機能を備えていました。

ほとんどの組み込みシステムは、ディスクや端末、あるいはその両方を備えていないため、最も無駄のないForthプログラミング環境であっても使用することができません。それでも、いくつかのベンダーは、マイクロコントローラにオンボードのForthを提供しています。前述のロックウェル社のAIM65や、テキサス州のNew Micros社、コロラド州のVesta Technologies社、カリフォルニア州のOpto-22社などが販売しているマイコンボードがその例である。

しかし、PCがあまねく存在するにつれて、より快適で強力なForthのクロス開発環境のホストとしても普及するようになった。これらは一般に、古典的なForthのメタコンパイラをベースに、クロス開発をサポートするように改良されたものである。

従来のForth辞書は統合されており、「定義」は単語の名前(テキスト・インタープリタが実行する辞書検索で見つけることができる)、実行可能部分(通常、コロン定義、変数、定数など、特定のクラスの単語を実行するコードへのポインタ)、データ空間(定義の内容を構成する単語の1つ以上の値またはアドレスが含まれる)を含み、すべて古典的には連続したメモリ位置にある(ただし、セクション5.2「実装戦略」参照)。メタコンパイラは、これらをホストシステムのコンパイラが使用する部分(シンボルテーブルに相当)とターゲットで実行時に必要となる部分に構造的に分割する。ターゲット・プログラムをROM化するためには、コンパイラはROMとRAMのデータ空間を別々に管理する必要があり、通常は複数の辞書ポインタのセットを使用する。

4. Forthのハードウェア実装

Forthの内部アーキテクチャは、2つのスタック、レジスタのセット、および他の明確に定義された機能を持つコンピュータをシミュレートしています。そのため、誰かが実際のForthコンピュータをハードウェアで表現しようとするのは、ほぼ必然的なことだった。

その最初の試みは、1973年に英国マンチェスター近郊のジョドレルバンク電波天文台のジョン・デイヴィス氏によって行われた。デイヴィス氏のアプローチは、生産中止となったフェランティ社のコンピュータを再設計し、その命令セットをForth用に最適化することだった。

最初の実際のForthコンピュータは、ビットスライスされたボードレベルの製品であった。その最初のものは、1976年にスタンダード・ロジックというカリフォルニアの会社によって作られた。スタンダード・ロジック社のチーフ・プログラマ、ディーン・サンダーソン氏は、ボードレベルコンピュータの命令セットにわずかな変更を加えることで、Forthが「アドレスインタープリタ」で使用している、ある高レベルのコマンドから次のレベルに移るための正確な命令を実装することに成功したのだ。このシステムは、アメリカの郵便局で広く使われた。

1980年代初頭、ロックウェル社はオンチップROMにForthプリミティブを搭載したマイクロプロセッサ、ロックウェルAIM 65F11 [Dumse, 1984]を製造した。このチップは組み込み用マイクロプロセッサのアプリケーションでかなり成功裏に使用されました。しかし、このプロセッサの実際のアーキテクチャ (基本的には6502) をForthのサポートに適合させる試みは行われませんでした。

1981年、ムーア自身は実際のForthチップの設計を引き受けた。最初はFORTH社で、その後チップを開発するために設立されたNovixという新興企業で働き、ムーアは1984年にデザインを完成させ、最初のプロトタイプは1985年初頭に生産された[Golden, 1985]。この設計はその後ハリスセミコンダクター社が購入、採用し、同社のRTXプロセッサのラインナップの基礎となった。

1980年代初頭から、メリーランド州のジョンズ・ホプキンス応用物理研究所のグループは、宇宙計測に使用する一連の実験的なForthプロセッサを開発した [Hayes, 1987]。これらのうち最も成功したものは、カリフォルニア州パロアルトのSilicon ComposersからSC-32として販売され、1990年11月にコロンビア・スペースシャトルで飛行したホプキンス紫外望遠鏡の制御に使われた [Ballard, 1991]。現在も開発中の多くの宇宙観測機器の基礎となっている。

ムーア氏自身は、独自に、特殊な用途のためにForthベースのプロセッサの開発を続けてきた。

様々なForthプロセッサは、Forthソフトウェアシステムに影響を及ぼしてきた。これらのアーキテクチャを最大限に活用するために、チップ内部のアーキテクチャに最適化されたマシンコードを生成するForthコンパイラが、ムーア、FORTH社、ラボラトリー・マイク

ロシステムズ社によって開発されたのである。NovixとHarrisチップの`FOR` ... `NEXT`というネイティブループ構造(単一引数の上限からゼロまでカウントダウンする)は、他のForthでもこの構造が採用されるきっかけとなった。

5. 現在と将来の方向性

コンピュータ業界は、常に急激で大きな変化を繰り返してきた。Forthが最後に標準化された1980年代初頭以来、手頃な価格のパーソナルコンピュータの速度、メモリサイズ、ディスク容量は100倍以上に増加した。8ビットプロセッサはPCでは珍しくなり(組み込みシステムにはまだ広く使われていますが)、32ビットプロセッサが一般的になっています。オペレーティングシステム、プログラミング環境、ユーザインターフェイスもはるかに洗練されたものになっています。最近のForthの実装は、商用、公共用を問わず、これらの問題に対処しようとするものが多くなっています。

5.1 標準化への取り組み

本稿執筆時点(1992年11月)では，技術委員会X3J14(著者ラザーとコルバーンがメンバー)がANS Forthの完成を間近に控えた段階にある．TCの20人の投票メンバーの中には、ベンダー(FORTH社、Creative Solutions社、Sun Microsystems社、NCRの一部門)、いくつかの大きなユーザ組織(Ford Motor社、NASA)、そして多くの小さなユーザ組織、コンサルタント、専門家が含まれています。1987年に始まったこのグループは、FORTH-79とFORTH-83の問題点、および現代の問題点に対処してきました。FORTHのユーザと実装者の間で現在活発に議論され、技術的な活動が行われている分野であるため、規格案で扱われている問題のいくつかを以下に紹介します。

ANS Forthは、FORTH-79とFORTH-83の間の非互換性によって引き起こされたいくつかの分断を調整することを試みます。例えば、FORTH-79の`NOT`機能を実行するために0=を残し、FORTH-83の`NOT`機能を実行するために`INVERT`を導入し、`NOT`という単語を削除しています。これにより、どちらのバージョンにも依存するアプリケーションの作者は、プログラムを変更することなく、`NOT`が好ましい動作の同義語として定義されている簡単なシェルを追加することによって互換性を達成することができます。

また、実装オプションに関する制限を実質的にすべて撤廃し、CPUのワードサイズに依存しないようにし、ホストOSのファイル互換性、動的メモリ割り当て、浮動小数点演算などの機能に対して、オプションで多数の拡張ワードセットを提供します。ANS Forthが扱う重要な問題のいくつかを以下に示します。

5.1.1 セルサイズ

FORTH-79とFORTH-83は、スタック幅、アドレス、フラグ、数値を含む16ビットアーキテクチャを義務付けています。ANS Forthはサイズを「セル」という単位で規定しており、その幅は実装によって決まりますが、少なくとも16ビットでなければなりません。セル、文字、またはセルや文字の整数倍でアドレスを増分する単語を追加し、移植性を向上しました。

5.1.2 算術計算

大きな論争の中で、FORTH-83は切り捨て除算(floor division)を義務付けました。これは以前の使用法(符号付き除算を処理するアルゴリズムを指定しなかった)と互換性がないだけでなく、ほとんどのプロセッサのハードウェア乗除命令と矛盾しています。しかし、多くの人が、切り捨て除算の方が数学的に適切であり、そう規定することが重要であると強く感じていました。この件については、どちらの実装も多数存在することを認識し、TCはflooredまたはtruncated除算のいずれかを許可することを選択した。実装はどちらをデフォルトで使用するかを指定し、両方の方法をサポートするプリミティブを提供しなければならない。

5.1.3 制御構造

Forthのユニークな特徴の1つは、独自の内部ツールにアプリケーションプログラマがある程度はアクセスできることです。例えば、コンパイラ、アセンブラ、テキストインタプリタが使用する字句解析器は1つで、アプリケーションのコマンドやテキスト解析にも使用できます。同様に、ループや条件文などの制御構造を実装するツールは、カスタム構造語を作るために利用できる。1986年、Wil Badenは、標準のForth構造語と、これらの基礎となるツールから作られたいくつかの拡張が、D. E. Knuthの論文「Go to文による構造化プログラミング」 [Knuth, 1974] で提起された問題に対する解を含むあらゆる構造を作るのに十分であることを示した [Baden, 1986].

Structure	Description
DO…LOOP	Finite loop incrementing by 1
DO…+LOOP	Finite loop incrementing by
BEGIN…UNTIL	Indefinite loop terminating when is ‘true’
BEGIN… WHILE… REPEAT	Indefinite loop terminating when is ‘false’

Structure	Description
BEGIN…AGAIN	Infinite loop
IF…ELSE…THEN	Two-branch conditional; performs words following IF it is ‘true’ and words following ELSE if it is ‘false’. THEN marks the point at which the paths merge.
IF…THEN	Like the two-branch conditional, but with only a ‘true’ clause.

表4. Forthにおける標準的な制御構造。ANS Forthでは、プログラマは構成語を混ぜたり、新しい構造語を定義するためにそれらを使って新しい構造を形成することができます。

FORTH-79とFORTH-83は、表4に示した一般的な構造のための構文仕様と、構造プリミティブの「実験的」なコレクションを提供した。しかし、後者は広く採用されず、標準が期待したような構文チェックを行う実装はほとんどありません。F83では、スタック(コンパイル時に構造体をコンパイルするために使用される)が定義のコンパイル前と後で同じサイズであることを要求するという、限定的な形式の構文チェックを提供しており、スタックの不均衡は不完全な構造を示しているという理論です。残念ながら、この手法では、コンパイル時にスタックに値を残しておいて、定義の中でリテラルとしてコンパイルするという、非常に一般的な慣習の妨げになってしまいます。

一般的な手法では、コンパイル時に構造語がどのように動作するかという知識を利用して、構造語を独創的な方法で操作することがよくありました。ANS Forth技術委員会は、コンパイル時および実行時の構造語の動作を規定することで、構造語を任意の順序で組み合わせることができるようにし、これを公認しました。構造プリミティブのセットは「プログラミングツール」ワードセットで提供され、 `POSTPONE` ワードは、プログラマが新しい動作の一部を提供するために既存のコンパイラ指令を参照する新しい構造語を書くことを可能にするために提供されています。

5.2 実装戦略

1970年代にムーアによって開発されたオリジナルのForthシステムは、ソースをディスクからメモリ上で実行可能な形にコンパイルしていました。これにより、多くのコンパイル言語に特徴的なコンパイル-リンク-ロードに分離したシーケンスを回避し、常駐するForthエディタを使ってソースを修正し、再コンパイルして数秒でテストに利用できる、非常にインタラクティブなプログラミングスタイルが実現できていました。定義の内部構造は、図2に示すように、すべてのフィールドがメモリ上で連続した構造になっていた。FIGモデルとその派生モデルは、この構造の詳細を多少変更したが、その本質的な特徴は維持した。

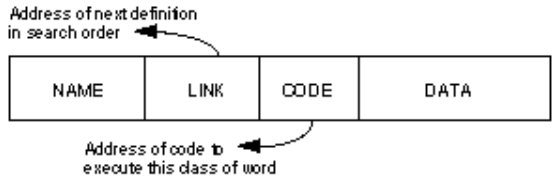

diagram: logical components of a Forth-language definition.

図 2. Forth 定義の論理コンポーネントを示す図。古典的な実装では、これらのフィールドはメモリ上で連続しています。データフィールドはデータオブジェクトの値、プロシージャのアドレスまたはトークン、CODE 定義の実際のコードを保持します。 このモデルに従って実装された Forth システムは、以前に定義された単語へのポインタをそのパラメータフィールドにコンパイルすることによって高レベルの定義を構築し、そのような定義を実行するアドレスインタプリタは、これらのルーチンを介して、その場所を保持するために使用されるレジスタを介して間接ジャンプを実行することによって参照される定義を順番に実行するように進みました。これを一般に間接スレッドコードと呼んでいる。

しかし、さまざまな条件に最適化する必要があるため、この基本的な実装方法にはさまざまなバリエーションがあります。最も興味深いものをいくつか紹介します。

1. **直接スレッド型コード**。このモデルでは、コードフィールドには機械語コードへのポインタの代わりに機械語コードが含まれます。これは多少高速ですが、通常、いくつかのクラスのワードで余分なバイトがかかります。32 ビットシステムで最も一般的です。
2. **サブルーチンスレッドコード**。このモデルでは、コンパイラは、宛先アドレスを持つサブルーチンへのジャンプ命令をインラインに配置します。この手法では、16 ビットシステムでは、コンパイルされた参照ごとに余分なバイトがかかります。直接スレッド化されたコードよりも遅いことが多いのですが、ネイティブコード生成への移行を可能にする技術です。
3. **ネイティブコード生成** サブルーチンスレッドコードの一歩先を行くこの技術は、+や他の高レベルルーチンへのジャンプなどの単純なプリミティブのためのインラインマシン命令を生成します。その結果、サイズとコンパイラの複雑さの代償として、実行速度が大幅に向上します。ネイティブコードは、スレッドコードよりもデバッグが困難な場合があります。この手法は、RTX などの Forth チップ用に最適化されたシステムや、コードのコンパクトさがスピードよりも重要でないことが多い 32 ビットシステムで使用されます。

4. **最適化コンパイラ**。ネイティブコード生成の一種で、1サイクルで複数のForthプリミティブを実行できるForthプロセッサ(1.5節で説明)用に考案された。この方法で処理可能なパターンを探し、適切な命令を自動生成した。最適化の範囲はプロセッサの能力に支配され、例えばNovixやRTXプロセッサ用のpolyFORTHコンパイラは4要素の覗き窓を持っていました。
5. **トークンスレッド化**。 他の単語への参照を、テーブルへのインデックスのようなトークンを使ってコンパイルする手法で、絶対アドレスよりもコンパクトになります。トークンによるスレッド化は、例えば1980年代初頭に開発されたパナソニックのハンドヘルドコンピュータ用Forthのバージョンで使用され、MacForthでも重要な要素となっている。
6. **セグメント化アーキテクチャ**。80×86ファミリは、セグメント化されたアドレス空間をサポートしている。これを利用して、16ビットシステムで64Kを超えるプログラムをサポートするForthもある。同様に、8051やTI TMS320シリーズなどのハーバード・アーキテクチャ・プロセッサの実装では、コード空間とデータ空間を分けて管理している。

初期の規格では古典的な構造を前提としていましたが、ANS Forthでは実装技術に関する仮定を避けるために、定義の先頭とデータ空間の関係を仮定することや、定義済みの演算子以外でデータ構造の本体にアクセスすることを禁止するなどの特別な取り組みを行っています。このため、プログラマの間では、このような仮定をする自由を好む派と、別の実装方法で可能な最適化を行いたい派の間で論争が起こっています。

5.3 オブジェクト指向の拡張機能

Forthは、コンパイル時および実行時の動作と同様に、ユーザ定義の構造を持つカスタムデータ型をサポートしているため、長い年月を経て、プログラマは、前述の画像処理に対するムーアのアプローチ(セクション2.2.2、項目2)のようにオブジェクトベースのシステムを開発するようになりました。Pountain [1987]は、Forthにおけるオブジェクト指向プログラミングの1つのアプローチについて述べており、多くの実装者によって試行されている。いくつかのForthベンダーはオブジェクトベースのシステムを実装するために他のアプローチをとっており、これは現在Forthの最も豊饒な探求分野の1つとなっています。

1984年、Charles Duffは Neonという Forthで書かれたオブジェクト指向のシステムを発表しました [Duff, 1984 a, b]。1980年代後半にDuffがサポートを打ち切ると、シカゴ大学Yerkes天文台のBob Lowensteinがそれを引き継ぎ、Yerkという名でパブリックドメインのシステムとして利用できるようになった。より最近では、Michael Horeがサブルーチン・

スレッドコードを使ってNeonを再実装し、MOPSという名前で(これもパブリックドメインで)利用できるようになった。YerkとMOPSの両方は、この論文の最後に挙げた多くのForth指向の電子掲示板で、ダウンロード可能なファイルとして入手できます。

6. *事後*評価

FORTHの初期の開発は、他の多くのプログラミング言語とは多くの点で大きく異なっていました。言語というものは、構文と意味論に関する明確な公式仕様に基づいた完全な機能を備えて登場するのが一般的ですが、FORTHは、言語の基本的な前提条件のそれぞれが実際のアプリケーションの経験という金床で試される、長くてダイナミックな青年期を楽しんでいたのです。この間、ムーア氏は、多くのユーザに邪魔されることなく、言語がどうあるべきかという彼自身の現在の見解に合うように、しばしば日常的に革命的な変更を加えていた。彼は、最初のブートストラップローダから完成したアプリケーションまで、手元のマシンを完全にコントロールし、責任を負っていた。言語は、リソースに制約のある環境で技術的に困難な問題を解決する一人の人間の実際のニーズに向かって収束していったのです。

その結果、デファクト言語仕様として表現された問題解決の方法は、他の人々にも有用であることが証明されました。シンタックスチェックやデータ型付けなど、プログラミング言語に不可欠な形式的構造を自由に追加できるにもかかわらず、Forthを独自に実装した数百人のうち、ほとんどの人が実装していない。ANS Forth技術委員会が調査した彼らの努力の結果は、Moore氏の個人的なビジョンが驚くほど民主的に承認されたことを表しています。

6.1 目標を達成するために

明確に定義された目的を示す正式な言語設計仕様がなければ、言語の発明者とそれを使用した人々が述べた目的を評価するしかありません。個人の生産性と知の移植性は、Mooreの主要な表明した目的でした。Forthは、プログラマブルコンピュータの大部分に移植され、いくつかの異なるForth専用コンピュータアーキテクチャに具現化されています。

1979年、Chuck MooreはForthの10年の経験を振り返り、次のように述べた[Moore, 1979]。

> 私の当初の目標は、生涯で40以上のプログラムを書くことでした。私は、自分のスループットを10倍以上にしたと思う。もはやスループットがプログラム言語の制限を受けるとは思えないので、私は自分がやろうとしたことを達成したことになる。私の手元にあるツールはとても有効で、他の人の手元でも有効なようです。私はこのことを嬉しく思い、誇りに思っている。

今日、彼はこの評価を変える理由はないと考えている。

FIG Forthの開発者たちは、自分たちのシステムが、自分たちの組織の支部とともに世界中に広まり、世界中のForthプログラマに影響を与えるのを目の当たりにした。また、Forth製品の商用ベンダーを増やすという彼らの目標も達成された。

Forthベースの企業にキャリアと財産を捧げた多くの起業家のうち、その努力のために金持ちになったり有名になったりした人はほとんどいない。しかし、ムーアと同じように、自分たちの生産性が向上したこと、言語のパワーと柔軟性によって、不可能と思われたプロジェクトの目標が達成されたことに満足している人がほとんどである。また、この能力をクライアントや顧客に提供することで、繁栄を享受してきたのである。

生産性と移植性というムーアの基準を考えると、これらの目標を達成した最も良い尺度は、多種多様なコンピュータ上で少数のプログラマによってForthで書かれた多種多様な、多数のアプリケーションプログラムであると言えるかもしれません。

6.2 Forthの主な貢献

1984年、Leo BrodieはThinking Forthという Forthアプリケーションの設計に関する本を書きました[Brodie, 1984]。その中で、彼は多くのForthプログラマの設計とコーディングの実践を引用しています。エピローグでは、他の言語でのプログラミングスタイルや、一般的な問題解決へのアプローチに、Forthが大きな影響を与えたと、何人かがコメントをしています。以下は、一般的なForthユーザの観察に典型的な2つの例です。

> 優れたForthプログラミングの本質は、手続きを有用な自立した言葉にファクタリングする技術である。Forthの言葉のアイデアは、実験室のハードウェア設計に予想外の影響を与えた。

大きな、モノリシックな、万能の Interface を作る代わりに、私は Forth の言葉のように機能するシンプルな小箱を山ほど作っていることに気づきました：それらは標準入力と出力の決まったセットを持ち、ただ一つの機能を果たし、それほど努力せずに互いに接続できるように設計されており、ラベルを見ただけでその箱が何をするかわかるほどシンプルでした。

Forth は小さいから、そして Forth はユーザにマシンをコントロールさせるから、Forth は人間にアプリケーションをコントロールさせるのです。科学者が研究室のコンピュータの前に座って、パッケージソフトで「20 の質問」をして遊ぶことを期待するのは馬鹿げています。Forth は、コンピュータが科学者を指導するのではなく、科学者がコンピュータを指導することを可能にするのです。

—Mark Bernstein Eastgate Systems, Inc., Cambridge, MA

Forth は、私の考え方をいろいろと変えてくれました。Forth を学んでから、私はアセンブラ、BASIC、FORTRAN など他の言語でコーディングをしてきました。その中で、言葉を作ってグループ化するという意味で、Forth で行っている分解と同じような使い方をしていることに気づきました。

もっと根本的なことを言えば、Forth は私のシンプルさへの忠実さを再確認させてくれました。多くの人は、複雑なツールを使って問題を解決しようとします。しかし、もっとシンプルなツールもありますし、もっと便利です。

Jerry Boutelle ノーチラス・システムズ社(カリフォルニア州サンタクルーズ)オーナ

Mitch Bradley は、Forth ベースの Open Boot の設計が、Sun Microsystems で Unix カーネルの低レベルインターフェイスを担当する人たちの考え方に大きな影響を与えたと報告しています [Bradley、1991]。Open Boot の設計思想は、ドライバインターフェース、デバイスのネーミングシステム、そして初期のスタートアップとコンフィギュレーションメカニズムに影響を及ぼしています。また、Forth 構文を使用して、いくつかのバラバラのカーネル設定ファイルの構文を統一し、サブセットの Forth インタプリタを Unix カーネルに含めるという話も出てきています。Sun で Open Boot を使っている人たちは、シンプルな postfix 構文が決して「力尽き」たり 「窮地に追い込まれる」ことがないという事実に感銘を受けているのです。

6.3 誤りや望ましい変化

Forthは、どんな特定のアプリケーションのニーズにも適応できるカメレオンのような能力を持っています。実際、Forthでのプログラミングのプロセスは、必要な機能がすべて実装されるまで、アプリケーション指向の単語をどんどん追加していくことです。だから、どんなプロジェクトでも、どんなプログラミング集団でも、必要だと思ったことは即座に対応することができる。そこで、「間違い」を探すときに最も有効なのは、多くの実装者が変更や追加を選択したものは何か、そして、より広く受け入れられることを妨げたと思われる言語の特性は何かということです。

1987年に設立されたANS Forth技術委員会が最初に行ったことの1つは、数百人のForth実装者とユーザに対して、Forth言語の問題点に関する意見を調査することでした。その結果、3つのカテゴリーに分類された。既存の2標準規格の「誤り」(例えば、FORTH-83で導入された非互換性や、`DO`の引数の仕様のような変則性)、FORTH-83の時代遅れの制限(主に16ビットアーキテクチャへの依存)、ホストファイルアクセスや浮動小数点演算などの標準化の必要性、です。後者の機能は、当時はほとんどの商用システム、多くのパブリックドメインシステムで提供されていましたが、独自に開発されたため、使い方や実践にはばらつきがありました。ANS Forthは、これらの問題点に対応することを試みました。

しかし、今にして思えば、浮動小数点演算など、他の言語でカバーされている機能を標準装備していないことが、Forthの普及を妨げていたのかもしれません。商用システムで使えるというだけでは、そのような機能についての言及がない規格がForthの一般的なイメージとして定着してしまうからだ。このような観点から見ると、ANS Forthの取り組みは遅すぎたと言えるでしょう。

もう1つの難点は、Forthのアイデンティティが不明確であることだ。あからさまなスタックアーキテクチャやポストフィックス表記への依存など、外観が型破りなだけでなく、言語だけでなくOSやエディタ、ユーティリティなど、多くの人が独立した存在として見慣れている領域に広くまたがっているのだ。そのため、「Forthとは何か」という問いに、単純な答えを出すことは難しい。

Forthの統合された特性は、その実践者たちによって、その最大の資産と見なされています。ブラッドリー[1991]が表現しているように。

> Forthは私に、一つのプログラミング環境の中の異なる構成要素(例えば、コンパイラ、リンカ、コマンドインタプリタなどで使われる異なる構文)の間の「防火壁」は非常に迷惑であり、同じ構文を使って、いつでも、どのレベルでも、どんなことでもできる均一な環境の方がずっと快適であることを教えてくれた。

しかし、Duncan [1991]は、このように言語としてのForth、仮想マシンとしてのForth、プログラミング環境としてのForthがシームレスに統合されていることが、主流に受け入れられるための大きな障壁になっていると考えている。彼は、Smalltalk対C++に関しても同じことが観察されたと指摘しています。

> 私は数ヶ月間C++を使っていますが、C++の不満な点、つまり、言語がそれ自体で書かれていない(したがって、プログラミング環境のビルディングブロックをアプリケーションの一部として使う方法がない)、言語が真に拡張可能ではない(たとえば、ネイティブデータ型の演算子はオーバーライドできない)、言語とクラス階層について賢いプログラミング環境がない、などは、従来の言語の専門家がSmalltalkと比較してC++が資産として見ていることなのです

しかし、Forthのユーザが、その統合的で本質的にインタラクティブな性格が、プログラマとしての生産性の鍵であると確信している限り、それは変わりそうもない。

6.4 問題点

ある種の言語は「平準化」する傾向があります。つまり、専門家が書いたプログラムは、初心者が書いたものよりも著しく良い(より小さい、より速い、など) ものとなりにくいです。Chuck Mooreはかつてこう言いました [Moore, 1979]、「...FORTHは増幅器だ。良いプログラマはFORTHで素晴らしい仕事をするが、悪いプログラマは悲惨な仕事をする」。定量化されたことはありませんが、この観察は多くのForthプロジェクトで、幅広いプログラマ層にわたって繰り返され、Forthコミュニティの中で「民間の知恵」の地位を獲得してきました。

このような傾向から、Forthは「手に負えない」という評判が立ち、「Forthの大失敗」(1980年代初頭のエプソンのVALDOCSプロジェクトが有名)が大きく報道されたこともある。しかし、よくよく考えてみると、このプロジェクトやその他の失敗したForthプロジェクトの根本原因は、他の言語を使ったプロジェクトを破滅させるのと同じ問題、つまり、不十分な定義、不十分な管理、非現実的な期待なのである。

また、前述のサウジアラビアの空港の施設管理システムなど、Forthの成功例も数多くあります。このプロジェクトでは、30万行の実行可能なFortran、PLM、およびアセンブリ言語のソフトウェアが含まれると見積もられていましたが、完全に再設計し、Forthで再コード化し、わずか18ヶ月で顧客が満足するようにテストしました [Rather、1985年]。その結果、動作は10倍以上速くなりました。

Vesta Technologies のシニアプロジェクトマネージャである Jack Woehr は、Forth プロジェクトをうまく管理するには、一般的に優れた管理方法と、Forth プログラマがその並外れた生産性に抱く特別な誇りを理解すること以上のことは必要ないとしています [Woehr, 1991]。Forthは、高度に熟練した専門家の小さなチームが、より若いプログラマの大きなグループよりも、より良い仕事を、より短時間で、より少ない総費用で行うことができると信じる管理スタイルに報いているのです。

6.5 現在の言語と将来の言語への示唆

20年にわたるForthの経験から何を学ぶことができるでしょうか。Forthは、言語開発者を導いてきた多くの仮定に対する生きた挑戦である。例えば、厳密な構文や強力なデータ型付けがないことは、Forth プログラマが主要な利点として挙げている特徴的な点です。Forthのシステムとプログラマーとの非公式でインタラクティブな関係は、C言語などの従来のツールに比べ、開発期間を短縮することが多くのプロジェクトを通じて示されています。また、コンピュータの性能は飛躍的に向上しましたが、Forthは、簡単なプログラミング、コンパクトなサイズ、高速なパフォーマンス(互いに相容れないと思われがちな特性)を兼ね備え、特に組み込みシステム向けのソフトウェア開発者に支持され続けています。

References

ANS 1991
ANS Forth, document number X3.215-1994, available from Global Engineering Documents, 2805 McGaw Ave., Irvine, CA, 92714.

Baden, 1986
Baden, W., “Hacking Forth.” Proceedings of the Eighth FORML Conference, pub. by the Forth Interest Group, 1986.

Ballard, 1991
Ballard, B. and Hayes, J., “Forth and Space at the Applied Physics Laboratory,” in Proceedings of the 1991 Rochester Forth Conference. Rochester, NY: The Forth Institute, 1991.

Bradley, 1991
Bradley, M., private communication, 7/8/91.

Brodie, 1981
Brodie, L. Starting FORTH. Englewood Cliffs, NJ: Prentice Hall, 1981.

Brodie, 1984
Brodie, L. Thinking Forth. Englewood Cliffs, NJ: Prentice Hall, 1984.

Cox, 1987
William C. Cox, “A case for NPOSs in real-time applications.” I&CS Magazine (pub. by Chilton), November, 1987.

Derick, 1982
Derick, M. and Baker, L., The Forth Encyclopedia. Mountain View, CA: The Mountain View Press, 1982.

Dewar, 1970
Dewar, R., “Indirect Threaded Code.” Communications of the ACM, 18, 6, 1975.

Dijkstra, 1969
Dijkstra, E.W., “Structured Programming,” Software Engineering Techniques, Buxton, J.N., and Randell, B., eds. Brussels, Belgium, NATO Science Committee, 1969.

Duff, 1984
C. Duff and N. Iverson, “Forth Meets Smalltalk,” Journal of Forth Application and Research,2, 1, 1984.

Duff, 1984b
C. Duff, “Neon – Extending Forth in New Directions,” Proceedings of the 1984 Asilomar FORML Conference pub. by the Forth Interest Group, 1984.

Dumse, 1984
Dumse, R., “The R65F11 and F68K Single Chip FORTH Computers,” Journal of Forth Application and Research, 2, 1, 1984.

Duncan, 1988
Duncan, R. (Gen’l Ed.). The MS-DOS Encyclopedia. Redmond, WA: Microsoft Press, 1988.

Duncan, 1991
Duncan, R., private communication, 7/5/91.

Electronics, 1976
“RCA may offer memory-saving processor language.” Electronics (Feb. 19, 1976, p. 26).

FIG, 1978
Forth Dimensions, 1, 1, June/July 1978, pub. by the Forth Interest Group.

FIG, 1983
Forth Dimensions, 5, 3, September/November, 1983, pub. by the Forth Interest Group.

Gehrz, 1978
Gehrz, R.D., and Hackwell, J.A., “Exploring the Infrared Universe from Wyoming.” Sky and Telescope (June, 1978).

Golden, 1985
Golden, J., Moore, C.H. and Brodie, L. “Fast Processor Chip Takes Its Instructions Directly from Forth,” Electronic Design (March 21, 1985).

Hart, 1968
Hart, J.F. et al., Computer Approximations. Malabar, FL: Krieger, 1968; 2nd ed., 1978.

Hayes, 1987
Hayes, J., Fraeman, M.E., Williams, R.L. and Zaremba, T., “A 32-bit Forth Microprocessor,” Journal of Forth Application and Research, 5, 1, 1987.

Knuth, 1974
Knuth, D.E., “Structured Programming with go to statements.” Computing Reviews, 1974, #4.

Moore, 1958
Moore, Charles H. and Lautman, D.A., “Predictions for Photographic Tracking Stations – APO Ephemeris 4” in SAO Special Report #11, G.F. Schilling, ed. Cambridge, MA: Smithsonian Astrophysical Observatory, 1958.

Moore, 1970a
Moore, Charles H. and Leach, G.C. FORTH – A Language for Interactive Computing. Amsterdam, NY: Mohasco Industries Inc. (internal pub.) 1970.

Moore, 1970b
Moore, C.H., Programming a Problem-oriented Language. Amsterdam, NY: Mohasco Industries Inc. (internal pub.) 1970.

Moore, 1974a
Moore, C.H., “FORTH: A New Way to Program a Computer,” Astronomy & Astrophysics Supplement Series, 15, 3, June 1974. Proceedings of the Symposium on Collection and Analysis of Astrophysical Data at NRAO, Charlottesville, VA, Nov. 13-15, 1972.

Moore, 1974b
Moore, C.H. and Rather, E.D., “The FORTH Program for Spectral Line Observing on NRAO’s 36 ft Telescope” Astronomy & Astrophysics Supplement Series, 15, 3, June 1974. This is the Proceedings of the Symposium on the Collection and Analysis of Astrophysical Data given at NRAO, Charlottesville, VA, November 13-15, 1972.

Moore, 1979
Moore, C.H., “FORTH, The Last Ten Years and the Next Two Weeks…” Address at the first FORTH Convention, San Francisco, CA, November 1979, reprinted in Forth Dimensions, 1, 6, 1980.

Moore, 1983
Letter to the Editor of Forth Dimensions, 3, 1, 1983.

Paloski, 1986
Paloski, W.H., Odette, L., and Krever, A.J., “Use of a Forth-based Prolog for Real-time Expert System.” Journal of Forth Application and Research, 4, 2, 1986.

Pountain, 1987
Pountain, R. Object Oriented Forth. New York: Academic Press, 1987.

Parnas, 1971
Parnas, D.L., “Information Distribution Aspects of Design Methodology.” Proc. IFIP 1971 Congress. Ljubljana, Yugoslavia.

Phys. Sci. 1975
“Graphics in Kitt Form.” Physical Science, Nov. 1975, p. 10.

Rather, 1972
Rather, E.D. and Moore, C.H., FORTH Programmer’s Guide, NRAO Computer Division Internal Report #11, 1972. A later version, with J.M. Hollis added as a co-author, was Internal Report #17, 1974.

Rather, 1976a
Rather, E.D. and Moore, C.H., “The FORTH Approach to Operating Systems,” Proceedings of the ACM, Oct. 1976 pp. 233-240.

Rather, 1976b
Rather, E.D. and Moore, C.H., “High-level Programming for Microprocessors”, Proceedings of Electro 76.

Rather, 1985
Rather, E.D., “Fifteen Programmers, 400 Computers, 36,000 Sensors and Forth,” Journal of Forth Application and Research (3, #2, 1985). Available from The Forth Institute.

Sperry, 1991
Sperry, Tyler, “An Enemy of the People.” Embedded Systems Programming (4, 12), December, 1991.

Taylor, 1980
Taylor, Alan, “Alternative Software Making Great Strides.” Computerworld, 12/?/80.

TRS-80, 1979
Press release published in “Software and Peripherals” section of Minicomputer News, 8/30/79.

Veis, 1960
Veis, George and Moore, C.H., “SAO Differential Orbit Improvement Program” in Tracking Programs and Orbit Determination Seminar Proceedings. Pasadena CA: Jet Propulsion Laboratories, 1960.

Woehr, 1991
Woehr, Jack J., “Managing Forth Projects,” Embedded Systems Programming (May, 1991).

Wood, 1986
Wood, R.J., “Developing Real-time Process Control in Space.” Journal of Forth Application and Research, 4, 2, 1986.

Bibliography

Brodie, L. Starting FORTH. Englewood Cliffs, NJ: Prentice Hall, 1981. [online edition]

Brodie, L. Thinking Forth. Englewood Cliffs, NJ: Prentice Hall, 1984. [Thinking Forth project on Sourceforge.net, more download links]

Feierbach, G. and Thomas, P. Forth Tools & Applications. Reston, VA: Reston Computer Books, 1985.

Haydon, G.B. All about Forth: An Annotated Glossary. La Honda CA: Mountain View Press, 1990.

Kelly, M.G., and Spies, N. FORTH: A Text and Reference. Englewood Cliffs, NJ: Prentice Hall, 1986.

Knecht, K. Introduction to Forth. Howard Sams & Co., Indiana, 1982.

Kogge, P.M. “An Architectural Trail to Threaded Code Systems.” IEEE Computer (March, 1982).

Koopman, P. Stack Computers, The New Wave. Chichester, West Sussex, England. Ellis Horwood Ltd. 1989

Martin, T. A Bibliography of Forth References, 3rd Ed. Rochester, NY: Institute for Applied Forth Research, 1987.

McCabe, C.K. Forth Fundamentals (2 volumes). Oregon: Dilithium Press,1983.

Moore, C.H. “The Evolution of FORTH – An Unusual Language.”Byte (August 1980).

Ouverson, Marlin (ed). Dr. Dobbs Toolbook of Forth, Redwood City, CA: M&T Press, Vol. 1, 1986 Vol 2, 1987.

Pountain, R. Object Oriented Forth. New York: Academic Press, 1987.

Rather, E.D. “Forth Programming Language.” Encyclopedia of Physical Science & Technology (Vol. 5). New York: Academic Press, 1987.

Rather, E.D. “FORTH.” Computer Programming Management. Auerbach Publishers, Inc., 1985.

Terry, J.D. Library of Forth Routines and Utilities. New York: Shadow Lawn Press, 1986

Tracy, M. and Anderson, A. Mastering Forth (2nd ed). New York: Brady Books, 1989.

Winfield, A. The Complete Forth. New York: Wiley Books, 1983.

Addendum to Section 2.2.2

NRAOと日付とロバスト性

1989年9月、ラザーとコンクリンは、FORTH社にNRAOの11-M望遠鏡から「ホットライン」の電話がかかってきたことにショックを受けた。カレンダーが突然誤作動を起こしたのだ。この問題は、Y2Kバグの一種であった。FORTHのプログラム(そしてそれ以降の多くのFORTHシステム)は、1900年1月1日の0から数えるユリウス日数として、内部で日付を保持していたのである。そのため、日付は内部的には16ビット整数で表現することができ、その方が小さいだけでなく、計算もはるかに簡単です。しかし、16ビットの符号付き日付は、1989年9月17日(32767日目)に「ロールオーバー」してしまう。FORTH社ではずっと以前から使われていたのだが)単純な修正として、符号なし演算を使えば、16ビットの日付は2079年6月5日まで使えるようになった。このプログラムは、何世代にもわたって新しいハードウェアに移植されましたが、十分な機能を持ち、多くの天文台ではまだ使われていない機能を実現していました。ワイオミング大学のプログラムは、その後ミネソタ大学にも採用され、1999年現在も4つの天文台で天文学者によって使用されています。

Addendum to Section 2.4.3.3

インタープリタ

ここで説明した「アドレス・インタプリタ」は、長年にわたってForthの支配的な実装戦略でしたが、これはこの言語固有の機能ではありません。1990年代後半までに、ほとんどの商用システムは、サブルーチンスレッディングとコード最適化を伴う直接コードコンパイ

ルの組み合わせに切り替えています(セクション5.2参照)。これにより、ほとんどのプラットフォームで大幅に高速化され、32ビット以上のプロセッサではインタプリタ型コードよりもさらに小さくなります。

Addendum to Section 3.2

パーソナルコンピュータ用の商用およびパブリックドメインシステム、表3

残念ながら、これらの中にはもう運営されていないものもあります。Bradley Forthware は現在 Firmworks で、Open Firmware のソフトウェアとサポートを専門に扱っています。Don Colburn は 1995 年に退職し Creative Solutions を売却しました。MacForth 製品ラインは現在 MegaWolf, Inc. によって維持・サポートされています。明るい面では、MicroProcessor Engineering (MPE) と FORTH, Inc. が Windows ベースの Forth と組み込みシステム用の Forth クロスコンパイラの新しいラインを持っており、Tom Zimmer (Andrew McEwen と一緒に)は人気のあるパブリックドメインの Win32Forth を開発しました。また、Unix や Linux 用の GPL やパブリックドメインの Forth もいくつかあり、特に gForth、bigForth、そして iForth が有名です(リンクは www.forth.org をご覧ください)。

Addendum to Section 3.3.1

環境とアプリケーション

最近の Forth の宇宙アプリケーションのリストは、NASA の Space-Related Applications of Forth を参照してください。Sun Microsystems の Open Boot は Open Firmware (IEEE1275, 1994) に発展し、今日では Sun 以外にも Apple や IBM などの非インテル PCI バスシステムで、ブートレベルの「プラグアンドプレイ」ファームウェアとして主流になっています。

Addendum to Section 4

Forth のハードウェア実装

ハリス社の RTX-2010 は放射線強化デバイスとして提供されており、現在でも軍事・航空宇宙用途で使用されています。Forth チップは他にもいくつか開発されており、たとえば Patriot Scientific 社の PTS1000(ただし Patriot 社はマーケティング上の理由から、これを Java チップとして宣伝している)がある。 チャック・ムーア氏は現在、自身の新会社 IntellaSys のためにマルチコアの Forth チップを設計しています。

Addendum to Section 5.1

標準化への取り組み

1994 年に ANSI 規格（X3.215.1994）が発行され、その後、同じ技術内容が ISO/IEC 15145:1997 として「fast track」承認された。ANSI は、5 年ごとに規格を再評価することを要求しており、この記事の執筆時点では、この取り組みが現在進行中である。 Forth の拡張機能一式は、1980 年代後半に Sun Microsystems 社の Mitch Bradley 氏らが行った作業を基にした規格、Open Firmware（旧 IEEE 1275）の基礎となった。

Addendum to Section 5.2

実装戦略

1990 年代後半までに、ほとんどの Forth のプロフェッショナル版といくつかのフリーウェア版は、コードの最適化を伴う直接コードコンパイルを採用し、よく最適化された C 言語に匹敵する性能を実現しました。SwiftForth と SwiftX を参照してください。

Addendum to Section 5.3

オブジェクト指向の拡張機能

ほとんどの Windows ベースの Forth はオブジェクト指向の拡張機能を提供します。SWOOP と呼ばれる SwiftForth の OOP パッケージについて読んでください。

参考文献と参考文献の補遺

Conklin, Edward K. and Rather, Elizabeth D. Forth Programmer’s Handbook. Los Angeles, California: FORTH, Inc. (3rd ed., 2007)Rather, Elizabeth D. et al. Forth Application Techniques. Los Angeles, California: FORTH, Inc. (5th ed., 2008)Open Firmware (formerly IEEE-1275) Open Firmware

Conklin, Edward K. “Smart Cards and the Open Terminal Architecture,” Dr. Dobb’s Journal (Dec. 1998).