Magagine: 1974, Astron. Astrophys. Suppl. 15, 497-511.

FORTH: A NEW WAY TO PROGRAM A MINI-COMPUTER

C.H. MOORE

National Radio Astronomy Observatory* Charlottesville, Virginia, U.S.A.

ほとんどのプログラミング言語は、小さなコンピュータで使うには不器用であったり、高価であったりします。FORTHはそのどちらでもない対話型高レベル言語です。FORTHは、コアと時間を非常に有効に使い、通常は低レベル言語に制限される機能を利用することができます。

問題を記述するために、プログラマはアプリケーション指向の語彙を定義する。プログラマはこの語彙をディスク上にソースとして作成する。ユーザはこれをcoreにコンパイルし、FORTHの標準的な機能と同様に、自分の問題を簡単かつ柔軟に解決するために使うことができる。

FORTHは数多くのコンピュータに実装されています。8Kのコアメモリ、ディスク(またはテープ)、端末が必要です。FORTH基本部分は、I/O、マクロアセンブラ、コンパイラを、わずか2Kワード のコアに常駐させている。もう1Kは自動バッファに使われ、ディスクをコアメモリの延長のように見せている。

1. PHILOSOPHY

近年のミニコンピュータの開発により、その性能は向上し、コストも下がり、大型コンピュータやカスタムハードウェアを駆逐しつつある。しかし、小型コンピュータは従来の高級言語を効率的に使用することができず、通常はアセンブラ言語でプログラミングされる。この低レベルのアプローチは、熟練したプログラマと多大な労力を必要とする。プログラミングのコスト、遅延、品質に関して、ほとんどすべての人が不満を持っている。これは、標準的な技術を効果的に実装しようとする断固とした試みにもかかわらず、根強く残っているのです。

FORTHは、コンピュータとの効果的なコミュニケーションという問題に対する新しいアプローチです。FORTHは、動詞、名詞、定義語を要素とする英語風の言語である。動詞はコンピュータの一連の操作を引き起こし、名詞は操作される対象であり、定義語は以前に定義された言葉の観点から、あるいは機械命令の観点から、新しい言葉を定義させる。

FORTHは、プログラマが自分の特定の問題を記述するために拡張できる基本的な語彙を提供する。基本語とは、語彙を構成し、並べ替え、テストするために必要な語であり、簡単で便利なものである。このようにFORTHは、通常ならば、アセンブリ言語、コンパイラ言語、ジョブコントローラ言語、アプリケーション言語を必要とする範囲をこれ一つでカバーする言語を提供します。

FORTHは多くのレベルで有用です。プログラマにとっては、コアに常駐するアセンブラとコンパイラを提供し、ソースから再コンパイルすることでオブジェクトプログラムをロードする必要がありません。このような機能は、通常、非常に大きなメモリを持つ大型コンピュータにしかないものです。また、プログラマには、プログラムの作成と修正を容易にするテキストエディタが用意されています。また、独立した(複数のプログラムを持つ)タスクを簡単に記述する方法を提供します。

ユーザにとっては、要求の厳しいリアルタイムアプリケーションも扱える効率的な語彙と、込み入った状況を診断する際に特に有用な広範な語彙を提供してくれます。さらに、必要に応じてデータブロックをディスクからコアへ、またディスクへ自動的に移動させるディスクルーチンにより、コアが提供するよりもはるかに多くのメモリに簡単にアクセスできるようになります。

最後に、プログラマはアプリケーションの語彙を定義する必要がありますが、 ユーザは自分の特定の問題に合うように自分で語彙を拡張することができま す。FORTHは特に、従来の専門用語を避け、シンプルで自然な方法で問題を記述することを可能にしています。

2. LANGUAGE

FORTHシステムには5つの重要な要素があります。まず、これらについて簡単に説明し、次にその実装について述べます。その前に、FORTHは基本的に語彙であり、その語彙の中で何が「単語」を構成してい るのかを理解することが重要です。

単語とは、空白で区切られた任意の文字列のことである。単語を構成できない特別な文字や、単語を開始できない文字はありません。したがって、算術演算子を表す文字や、句読点に似た文字も、空白で区切られていれば単語となり得る。例えば、次のようなものが単語となります。

```
FORTH re-start + IF, ? */. 2@ 3.14 1-0
```

ここでは、単語は大文字かアポストロフィで示される。つまり、FORTHと"?"の文字列は単語を示す。

3. DICTIONARY

FORTH言語は辞書によって構成されています。これは、プログラムが使用するメモリのほとんどすべてを占めます。その機能は、語彙の中の各単語の定義を提供することである。これには、動詞が行うべき操作も含まれる。また、この辞書には、最も頻繁に参照される名詞も含まれている。

FORTHはこの辞書の中の単語を見つけ、単語を追加し、辞書の限定された部分を選択するコンテキストを認識する。

単語は「定義語」によって辞書に追加されるが、その中でも特によく使われるのは3つである。1つは";"であり、端末からのメッセージの中で遭遇したとき、それに続く単語を辞書に入力することを示す。この新しい単語の、以前に定義された単語から見た定義も、辞書に登録される。たとえば

```
: APPLE MEDIUM SIZE ROUND RED FRUIT;
```

は、APPLEの定義かもしれません。

もう一つの定義語は「CODE」である。これは、その次の単語が機械語命令によって定義されることを示すもので、この機械語命令も辞書に登録されることになる。例については後述します。

3つ目の定義語はINTEGERで、これも同様に次の単語を辞書に登録する。しかし、この場合、単語は名詞であり、その値のために辞書の位置が確保される。たとえば

```
100 INTEGER DATA
```

は名詞 DATA を定義し、その値を 100 に設定します。

4. STACK

プッシュダウン式のスタックは、ユーザとユーザが使う単語、およびある単語と別の単語の間の通信を提供します。例えば，ユーザは数字(あるいは他のオブジェクト)をスタックに置き，次に単語をタイプすることで，スタックに作用して望みの結果を得ることができます．例えば、次のように

```
12 2400 * 45 / .
```

と入力すると、12 と 2400 という数字をスタックに置き、それらを掛け合わせ (*)、45 をスタックの一番上に置き、前の結果をそれで割って (/)、その結果 (.) をタイプします。

最近、電子計算機でプッシュダウン式のスタックが普及したため、この概念はより身近なものとなっている。

もう一つのスタックである「リターンスタック」は、特定のプログラムパラメータを保存するために使用されます。

5. INTERPRETER

FORTHには2つのインタプリタがある。上位のインタプリタは、端末から単語を読み、辞書を検索し、見つけたエントリを実行する。

低レベルのインタプリタは、他の単語との関連で定義された単語を実行する。例えば、先ほどのAPPLEの定義。低レベルインタプリタはAPPLEを見ると、MEDIUM、SIZE、ROUND、RED、FRUIT、';'の順に実行することになる。これらの単語はすでにコンパイルされているので、この作業は極めて単純である。つまり、APPLEという単語を定義するときに、その定義に含まれる各単語を辞書で検索し、その結果の辞書アドレスをAPPLEのエントリに配置してある。APPLEが実行されたとき、インタプリタはこれらのアドレスを拾って、必要なコードを実行すればよい。APPLEを定義するテキストは、インタプリタ言語によくあるように保存されない。

低レベルインタプリタには、いくつかの重要な特性がある。まず、高速であること。実際、いくつかのコンピュータでは、単語そのものが意味するコードに加えて、追加で実行される命令数は各単語に対してわずか2つでしかない。第二に、コンパクトな定義を解釈することである。定義で使われる各単語がコアで占める場所は1箇所だけである。ある単語を他の単語で定義する場合、その定義を解釈するコンピュータに依存しないためである。

その結果、FORTHの語彙のほとんどは低レベルのインタープリタによって定義され、解釈されることになる。高レベルのインタプリタ自体も、このように定義されている。

6. ASSEMBLER

CODEという定義語を使って、プログラマは指定された機械語命令を実行させる語を定義することができる。このような定義は、I/Oの実行、算術演算の実装、その他機械に依存する処理を行うのに必要である。

これは、FORTHの重要な特徴である。FORTHは、コンピュータに依存したコードを、管理しやすい大きさに分割し、特定のインターフェイスの規則で明示することを可能にします。アプリケーションを別のコンピュータに移植するには、CODE語だけを再コード化すればよく、他の語とはコンピュータに依存しない形で相互作用します。

アセンブラが解釈する言葉の中には、命令ニーモニックがあります。各ニモニックは、対応する機械語命令をアセンブルするように定義されている。例えば、次のようなフレーズ

```
X LDA,
```

は、LDA命令をXの適切なアドレスでアセンブルし、辞書に置く。

このアセンブラは非常にコンパクトです。その主なコストは、ニーモニックが占める辞書のスペースで、通常250コアロケーションです。プッシュダウン・スタックは、従来のアセンブラで必要だったシンボルテーブルを大幅に削減する。これは、コアの位置に名前を付ける必要がないためです。動詞は、名前を付けた場所からパラメータを取り出すのではなく、スタック上でパラメータを見つけます。同様に、条件付きジャンプは、後で説明する方法で、アセンブル時にプッシュダウンスタックを使用します。ただし、変数とロケーションには名前を付けることができ、そのような名前は通常通り辞書で見つけることができます。

7. SECONDARY MEMORY

FORTHの最後の重要な要素は、ブロック-1024バイトの二次記憶装置です。2つのコアバッファが提供され、ブロックは必要に応じて読み込まれます。ブロックがコアで変更されると、そのバッファが再利用されるときに、ディスク上のイメージが自動的に置き換えられます。プログラマは、自分のデータがいつでもコアにあると考えることができるため、非常に小さなコストで、非常に大きなサービスを提供することができます。

ブロックは、語彙を定義するテキストを格納するために使用されます。このブロックは、ユーザが要求したときにコアにコンパイルされる。編集ボキャブラリは、ブロックを64文字で16行にフォーマットします。これにより、ユーザは自分のソースコードを修正したり、再コンパイルしたりすることができる。

また、ブロックはデータの保存にも使われる。小さなレコードを1つのブロックにまとめたり、大きなレコードを複数のブロックに分散して格納したりすることが容易にできる。ブロックはコンピュータに関係なく同じ形式なので、アプリケーションをあるコンピュータから別のコンピュータに移すのも簡単です。

8. EXAMPLE

これまで説明したFORTHの関数は、約100の定義された単語からなる基本辞書によって利用可能になっています。この基本辞書は2Kのコアロケーションを占める。16ビットコンピュータでは、2次記憶装置はそれぞれ512ロケーションの2つのコアバッファを必要としま

す。したがって、FORTHは3Kのコアで、端末I/O、仮想記憶形式のディスクI/0、インタプリタ、数値変換、算術、アセンブラ、コンパイラを提供する。

これらに関連する語彙を付録Iに列挙する。これは十分に広範なもので、これだけでプログラマの仕事を非常に簡単なものにしている。これを見るために、ある問題を考えてみよう。

CODE定義によって、2つの単語を定義したとする。

- READ はあるデバイスから 16 ビットの数値を読み取り、スタックにプッシュします。
- SEND は 16 ビットの数値を他のデバイスに送信します(そしてスタックから削除します)。

最初にすることは、それらが動作することを確認するためにテストすることです。

```
READ .
```

は数値を読み取り、それを印字します。8進数で表示するには、次のようにします。

```
READ OCTAL .
```

同様に

```
100 SEND
```

は出力デバイスに 100 を送ります。

これらの言葉の自然な組み合わせは、READ SENDかもしれません。この組み合わせはPASSと名付けることもできます。

```
: PASS READ SEND ;
```

というように、PASSと入力すると、あるデバイスから別のデバイスに数字が渡されます。負の数を送らないようにするには、次のように定義します。

```
: POSITIVE READ 0 MAX SEND ;
```

と定義すれば、負の数を 0 で切り取ることができます。 PASS が定義されると、複数の数値、例えば32個が渡されることがあります。

```
: MANY 32 0 DO PASS LOOP;
```

MANY と入力すると、PASS が 32 回実行されます。繰り返しの回数を可変にするには、

```
: TIMES 0 DO PASS LOOP,
```

と定義する。ループの上限を与えるには、

```
17 TIMES
```

と書く。ここで、これらの数値を読み取り、ディスクに保存することを望むとする。まず、数値が保存されるブロックを定義します。

```
100 INTEGER DATA
```

はブロックを識別する変数 DATA を指定し、それを 100 で初期化します。 そして、この変数を使用するストア演算子を定義します。

```
  : STORE DATA @ BLOCK + ! UPDATE ;
```

スタック上に2つの数値、サンプル番号(ブロック内の位置となる)と値を期待します。

- DATA @ はブロック番号を取得します。
- BLOCKはブロックの内容のコアアドレスを生成し、必要であればディスクから読み取ります。
- +は、このアドレスにサンプル番号を加算します。
- ! は、その値をデータブロックに格納します。

- UPDATEはFORTHにブロックの書き換えが必要であることを知らせる。

```
100 3 STORE
```

と入力すると、ブロックの4番目の場所(0から数えて)に100が格納されます。 これで、次のように

```
: SAMPLE READ SWAP STORE ;
```

SAMPLEを定義することができます。

```
0 SAMPLE 1 SAMPLE 2 SAMPLE
```

と入力すると、その言葉の意味するところをほぼ実現することができます。これをループさせると

```
: RECORD 32 0 DO I SAMPLE LOOP;
```

ここで、'I' はループを制御するインデックスを SAMPLE のパラメータとしてスタックに置く。RECORD がタイプされるたびに 32 個の数値がディスクに記録されます。

Finally, consider establishing a task that will do this every 10 seconds for an hour. Assume the word DELAY has been defined so that 最後に、これを1時間ごとに10秒だけ行うタスクを設定することを考える。DELAYという単語が

```
1 DELAY
```

が1ミリ秒待つように定義されているとします。 そして、次のようは定義

```
: MONITOR 600 0 DO RECORD 1 DATA + ! 10000 DELAY LOOP ;
```

はそのタスクを指定します。MONITOR と入力すると、1時間のタスクが始まり、360ブロックのデータを記録します。

この例は、FORTH がいかに単純な単語を単純な方法で組み合わせて、複雑な操作を行えるかを示しています。この単純さは、定義のネスト(入れ子)の便利さによるものである。MONITORは、4段階のネストを経て、FORTHの基本的な言葉に到達していることに注意してください。ネストはオーバーヘッド(通常10-20μs)を発生させますが、速度が重要な場合、最内部のループは常にコーディングが高級言語で行われる際と同じようにコーディングすることができます。

プッシュダウンスタックの使い方、すなわち、オペランドをスタックに格納するために、演算を与える前にオペランドを指定しなければならないというスタイルに習熟している限りにおいて、単語を慎重に選ぶことができれば、読みやすい定義を提供することができる。また、適切な演算子を定義するだけで、実際のデータ転送を気にすることなく、簡単にデータをディスクに保存することができることにも注意してください。

9. PROGRAM

FORTH言語の基本部分は、いろいろな方法で実装することができる。現在ではFORTHでコーディングされているが、これは他のアプリケーションと同様、このアプリケーションに適したものである。FORTHのサイズとオーバーヘッドを最小にしながら、FORTHの能力を最大にすることが継続的な目標である。前述したように、FORTHの基本部分は約3Kの16ビットコアを必要とします。実際のところ、PDP 11/40では、低レベルインタープリタのオーバーヘッドは1ワードあたり4.6usと、標準的なサブルーチンコール(5.9μs)より少ないのです。

FORTHプログラムは2つの部分から成っています。第一は、約1Kワードのプリコンパイルされたオブジェクトプログラム で、これには自分自身をコンピュータにロードするためのブートストラップルーチンも含まれています。このプログラムは、FORTHの典型であるように、ほとんどが辞書である。これは、他のすべての単語を定義することができる最小限の語彙を定義しています。

第二の部分は、基本語彙のソース記述(8ブロック)である。これは、オブジェクトプログラムによって、別の1Kワードにコンパイルされる。語彙は、変更や追加を容易にするため、また文書化するために、できるだけ多くの語彙をソースの形で残しておく。この語彙をディスクから再コンパイルするのに数秒しかかからないので、これには何のコストもかからない。この約100語の基本語彙に、アプリケーションのための語彙が追加される。

10. DICTIONARY

辞書は、可変長のエントリのリストである。エントリはリストの最後に追加することができ、また、指定されたエントリの後のすべてのエントリを忘れることができる。これによ

り、シンプルで効果的かつ動的なコア割り当てが可能になる。

ある単語を見つけるために辞書を検索する必要がある場合、まず最後のエントリが調べられる。一致しない場合は、そこから前のエントリへのポインタが取得され、そのエントリが調べられる。この手順は、一致する単語が見つかるか、辞書を使い果たすまで繰り返される。この特殊な検索は、単語が再定義された場合、その最新の定義が最初に見つかるという重要な特性を持っている。さらに、検索の開始点を変え、ポインターを適切に設定することによって、辞書の異なる部分を検索することができ、その結果、いくつかの異なる語彙を利用することができる。

各辞書項目は5つのフィールドを持つ。これらのフィールドと、16ビットコンピュータでの実装を図1に示す。

- 10.1. 定義される単語は、2つの場所を必要とする。これらの4バイトは、文字カウントと単語の最初の3文字を含む。この情報により、任意の長さの単語を最小限の混乱で区別することができる。
- 10.2. 辞書検索で使用されるポインタ。これは、直前の辞書エントリのアドレスを含む14ビットのフィールドである。
- 10.3. エントリに割り当てられた優先順位。これは、検索ポインタを含む場所を共有する2ビットのフィールドである。後述するコードを含む。
- 10.4. そのワードに対して実行されるコードのアドレス。低レベルのインタプリタは、エントリに遭遇するたびにこのコードを実行する。このフィールドは、異なる種類のワードを区別する。例えば、INTEGERで定義されたすべてのワードは同じアドレスを持ち、同じコードが実行されます。
- 10.5. ワードに関連するパラメータ。このフィールドは、任意の数のコアロケーションを使用することができ、辞書エントリが可変長となる主原因である。動詞の定義、および名詞に割り当てられたスペースを含む。

11. STACK

FORTHは16ビットのプッシュダウンスタックを実装しています。このようなスタックの記述は容易に入手可能である。Knuth (1968)はThe Art of Computer Programming, volume 1でそれらを説明している。数値がスタックに置かれたり、スタックから外されたりするとき、スタック上の残りの数値は実際には移動されない。むしろ、静的コア配列に置かれた最後の単語を示すポインタが調整される。

スタックポインタに対する正のアドレスがスタック上の番号を特定するように、また、ダブルワード項目がコア内の相対位置と同じ位置にスタック上に配置されるように、スタックがコア上で低位の方向に伸びることが重要である。

辞書はコアの高位に向かって伸びるので、スタックと辞書を互いに伸ばせるようにすると便利である。そうすれば、一方が使用しないすべてのコアを他方が使用できるようにな

り、スタックサイズの上限が効果的に排除される。

しかし、スタックには下限がある。なぜなら、スタックは決して空より小さくなってはならないからである。FORTH は入力された各単語をチェックして、スタックに実際にあるよりも多くのパラメータを使っていないかどうかを確認します。これは、端末というエラーを起こしやすい環境での診断を可能にし、新しい単語をテストすることを可能にします。しかし、FORTHは、正しく動作すると思われる定義でコンパイルされた単語のチェックには時間をかけません。

12. INTERPRETER

FORTHの高位インタプリタは、言語の見た目に責任を負っている。入力文字列から単語を読み取って、単語が何であるかを決定する。単語の仕様は意図的に単純化されている。単語はスペースで区切られた文字列である。

数字は単語の一種である。数値は特殊な単語で、-で始まり、数字、'.', または ':'(時間や角度を表す)数字が含まれることがある。'.'を持たない数字は 16 ビット整数に変換され、スタックに格納されます。

```
12345 -0 23:59
```

"." を持つ数値は、32 ビットの整数に変換され、スタックの 2 つの位置を占め、高次の部分が上になります。

```
1. -1.5 3.14159265
```

小数点の位置は保存されますが，数値のスケーリングには使用されません．一般的に、ユーザは正しい小数点以下の桁数を入力する必要があり、数値は暗黙の小数点以下の桁数を持つ整数として扱われます。

入力文字列はいくつかの場所から来ることができる。一つはターミナルである。ターミナル入力ルーチンは最大80文字をバッファに読み込みます。このルーチンは3つの制御文字を認識する。

- EOT はメッセージを高レベルのインタプリタに渡します。
- BACKSPACE はバッファの最後の文字を削除します。
- CANCELはメッセージ全体を破棄します。

これらにより、ユーザは入力ミスを修正することができます。

メッセージバッファは出力にも使用されます。TYPEという単語は、文字列をバッファに格納します。

端末出力ルーチンは、バッファが一杯になったとき、あるいはさらなる入力が要求されたときに、それを送信します。入力文字列は、ディスクから来ることもできる。

```
3 LOAD
```

を入力すると、ブロック3に格納されている1024文字の文字列が高レベルのインタプリタに渡されます。あるブロックから別のブロックをロードするために、現在のブロック番号と文字ポインタがリターンスタックに保存されます。メッセージバッファはブロック0として識別される。各テキストブロックは単語 ';S' で終わらなければならず、これはリターンスタックから前のブロックとポインタを取り戻すものである。

FORTH はすべてのテキストを 7 ビットの ASCII 文字として、偶数パリティ付きで保存します。これは、コンピュータ間の互換性を保つために非常に重要な規則です。

高レベルのインタプリタは、":"ワードと同じように定義され、低レベルのインタプリタに解釈される。これは次のフレーズで定義されています。

```
BEGIN WORD FIND NUMBER EXECUTE QUERY END
```

ここで、BEGINは無限ループの開始を意味します。

WORD は入力文字列から単語を抽出します。

FINDは辞書を検索します。もしその単語に一致するものが見つかれば、NUMBERをスキップする。それ以外の場合は

NUMBERは入力された単語を2進数に変換しようとする。失敗した場合は、単語そのものと、その後に"?"を付けてエラーメッセージとして印字する。

次にEXECUTEがその単語のコードを実行する(またはコンパイルする)。

QUERYはさらにテキストが利用できるようになるまで待ちます。

ENDはBEGINに戻り、この手順を繰り返す。

このインタプリタはいくつかの状態で動作する。一番下の状態では、EXECUTEはワードで指定されたコードを実行させる。しかし、新しい単語を定義する単語":"は、単語が実行さ

れるのではなく、コンパイルされるように状態を変更し、単語がそれを元に戻すまで、状態を変更する。単語をコンパイルするということは、その単語の辞書エントリのアドレスが、現在定義されているエントリに入れられるということである。同様に、数値のコンパイルとは、その数値を辞書に置くことを意味し、その前に、解釈時にスタックに戻すための別のエントリが置かれる。

図2は、ABSという単語が定義されているエントリを示しています。

```
: ABS DUP 0 < IF MINUS THEN ;
```

そのパラメータフィールドには、それを定義するワードのエントリのアドレスが含まれています。これは、サブルーチンコールの文字列に似ている。ただし、アドレスには16ビットすべてを使用することができ、16ビットコンピュータで一般的なアドレッシングの問題を排除することができます。

低レベルのインタプリタは、常にこのような定義を解釈しています。コンピュータが次に実行される命令を指すレジスタを持っているのと同じように、インタープリタは次に解釈される単語を指すポインタ(I)を持っています。

コード NEXTは、

- I にあるエントリのアドレスをインデックスレジスタに格納する。
- Iをインクリメントします。
- エントリのコードフィールド経由で間接ジャンプする。

NEXTのコストは、FORTHとコンピュータのマッチングを測る良い指標になります。これは、マイクロプログラム命令の理想的な機会を提供するものです。

実行されるコードのアドレスだけでなく、エントリのアドレスがコンパイルされることが重要です。同様に、エントリはインデックスレジスタに置かれることが重要です。これにより、コードフィールドだけでなく、ワードのパラメータフィールドにもアクセスできるようになります。

実際にエントリを割り当てるアドレスは、コンピュータに依存します。これは、NEXTを最適化するための選択肢の1つです。最も適しているのはパラメータフィールドですが、負の相対アドレスが使用可能でなければなりません。それ以外の場合は、コードフィールドかエントリの先頭を選択することができます。

13. PRECEDENCE

FORTHコンパイラはそれ自体がFORTHで書かれているので、コンパイル中の言葉とコンパイラへの命令としての言葉を区別する方法がなければなりません。優先順位フィールドはこれを行う。これは、ある語がコンパイル中に実行されるかどうかを指定するものである。表1は、エントリの優先順位と変数STATEの値によって、ワードがいつコンパイルされ、いつ実行されるかがどのように決まるかを説明しています。

ほとんどの単語は優先順位が0であり、普通にコンパイルされる。しかし、一部の単語、特にIF, ELSE, THEN, DO, LOOP, ';'はコンパイルが困難なため、コンパイル中に実行しなければならないという問題が生じる。これらの語はコンパイラ指令であり、定義の外側では使ってはいけません。ほとんどの指令は優先順位が1です。

しかし、FORTHの性質上、さらに複雑なことが起こります。コンパイラ指令を定義できるようにするためには、コンパイラ指令さえもコンパイルされる状態を定義しておく必要があるのです。これは、IMMEDIATEという指令で、現在のエントリの優先順位を1にし、変数STATEも2にします。これで優先順位1の指令もコンパイルされるようになりますが(表1参照)、それでもコンパイルを止めるために実行しなければならない単語があります。というわけで、';'は優先順位2になっています。

14. ASSEMBLER

各FORTHプログラムには、そのコンピュータ用に設計されたアセンブラの語彙があります。コンピュータ製造元のニーモニックが使われますが、多少の変更はあります。

接尾辞は別の単語とします。

インデックスを表すのに','(仕事させすぎ)ではなく、`)`を使う。1行のテキストを複数の命令で共有することがあるため、読みやすくするために操作ニーモニックにカンマを付け加えます。

しかし、最も大きな変更点は語順です。アドレスが命令より先になります。これは、従来のアセンブラ形式に疑問を持ったことのないプログラマにとっては、衝撃的なことである。しかし、これは他のFORTHの操作と同じであり、アセンブラを非常に単純化するものです。

FORTHでは、別の種類の命令が定義されており、最も一般的なニーモニックを定義するために用いられます。必要であれば、いつでも追加のニーモニックを定義することができます。これらのニモニックは、主にCODEという単語の後に使われ、そのパラメータフィールドにあるコードを実行するエントリを定義しています。例えば

```
CODE + 0 S) LDA, 1 S) ADD, BINARY JMP,
```

このフレーズは、16ビットの加算操作を定義しています。その辞書エントリを図3に示す。単語は次のように動作します。

CODE +	constructs the entry for the word '+'
O S)	puts onto the stack at assembly time, an address that will reference the number on top of the stack at execute time.
LDA,	assembles a 'load A register' instruction, using the above address from the stack, and places the instruction into the entry for '+'.
1 S)	puts onto the stack the address of the second number on the stack.
ADD,	assembles an 'add to A register' instruction.
BINARY	puts onto the stack the interpreter address appropriate for operations that take two numbers from the stack and leave a single result there.
JMP,	assembles a jump instruction to the address on the stack.

コードはインタープリタ内のいくつかのポイントのうちの1つに戻ることで終了します。これらのそれぞれは、スタックに異なる影響を与えます。もちろん、コードはスタック自体を操作することもできます。

アセンブラマクロは、マクロの機能が他の命令の観点から新しい命令を定義することであるため、「：」ワードとして実装されています。例えば

```
: LDB, LDA, IAB, ;
```

は、'load A' 命令と 'interchange A and B' 命令という形で 'load B' 命令を定義するかもしれません。 FORTH の命令ニモニックはその命令をアセンブルさせるので、定義にニモニックを含めると、その定義もアセンブルさせることになります。

アセンブル時にスタックを使用することで、次のような構文も可能になります。

```
IF ... ELSE... THEN
```

は、コンパイラだけでなく、アセンブラでも使用できます。

- IF は条件付きジャンプをアセンブルし、そのジャンプ命令のアドレスをスタックに残します。
- ELSEは無条件ジャンプをアセンブルし、そのアドレスをスタックに残し、IFで組み立てたジャンプに現在のアドレスを挿入します。
- THENは、ELSE(ELSEが省略された場合はIF)でアセンブルされたジャンプ命令に現在のアドレスを挿入します。同様に

```
BEGIN... END
```

はループをアセンブルします。END は BEGIN によってスタックに残されたアドレスへのジャンプをアセンブルします。

これらのマクロは、アセンブル時にスタックを操作することで、ラベルの必要性をなくし、ネストさせたり、オーバーラップさせたりすることができます。

```
BEGIN... IF ... SWAP END THEN...
```

BEGINとIFはスタックにアドレスを残しています。SWAP はそれらを交換して、END が BEGIN からのアドレスを取得し、THEN が IF からの アドレスを取得するようにします。

15. SECONDARY MEMORY

FORTHの二次記憶技術は、基本的にブロックの先頭から相対的にアドレス指定する形式をとっています。ブロックはコアに取り込まれ、できるだけ長くそこに置かれ、必要なら再書き込みされます。このようなアドレッシングは、スタックを介してパラメータに簡単にアクセスできることが重要なポイントになる。他の言語で実装するのは面倒だし、コストもかかる。

ディスク付きコンピュータのFORTHプログラムには、次のような定義があります。

```
: BLOCK CORE BUFFER READ ;
```

これは指定されたディスクブロックの、コア中の先頭アドレスを取得するために使われます。

- COREはコアのバッファをチェックして、希望するブロックがあるかどうかを確認します。

- BUFFERは最も古いバッファをチェックし、変更されていればそれをディスクに書き戻す。
- READはディスクから目的のブロックを読み出す。

テープはディスクの代用となり得るが，その場合，テープの動作に若干の遅れが生じる．テープの最初の通過では、テープ上の各ブロックのレコード番号を提供するコアマップが作成されます。各テープレコードの513番目の単語を使ってブロック番号を特定する。FORTHは、テープの位置を知ることで、求めているブロックのレコードにテープを進めたり戻したりすることができる。ブロックはテープの終りに書き直され、新しい位置を示すようにマップが更新されなければならない。

あるブロックが連続的に重なることなく他のブロックを参照できるようにするには、2つのコア・バッファが必要です。2つ以上のバッファが有用であることは稀である。FORTHは513ワードのバッファを使い、513ワード目にブロック番号を入れますが、ディスクには512ワードしか保存されません。UPDATEという単語は、ブロックが変更されたことを示すために、このステータス・ワードの符号ビットをセットします。これにより、ワードを個々にブロックに入れることができ、それぞれのワードに対してディスクアクセスを必要としない。

16. STATUS

FORTHは過去数年にわたり開発されてきました。最近、観測装置の制御、データ収集、表示など、多くのアプリケーションに FORTHが選ばれています。いくつかの天文台では、IBM 360、Honeywell 316、Varian 620、Data General NOVA、Hewlett Packard 2100、DEC PDP 10、PDP 11、Modular MODCOMP II コンピュータ上でFORTHを使用している。

FORTHが小さなシステムを実現することに成功したことで、さらなる配当が得られました。FORTHの基本部分は、明示的に言及された約500の機械語命令を含むだけです。新しいコンピュータのためにFORTHをコーディングすることは、これらの命令を機能的に同等なものに置き換えることを意味します。従って、FORTHに慣れたプログラマは、数週間で再コード化することができる。

FORTHの目標は、プログラマに提供するサービスを最大にし、時間とメモリ使用量の点でコストを最小にすることである。FORTHは、プログラマがユーザに、これまでにないレベルの効率性と汎用性を持つシステムを提供できるよう支援することを目指しています。

C.H. Moore
National Radio Astronomy
Observatory
Edgemont Road
Charlottesville, Virginia 22901, USA

Appendix I. Some basic FORTH words.

'l', 'm', and 'n' indicate numbers on the stack; 'a' indicates an address on the stack

Words concerned with the dictionary:

HERE	Address of next available word.
LAST @	Address of last entry.
WHERE	Type name of last entry.
n ,	Compile number into dictionary.
VOCABULARY word	Define the name of a vocabulary.
FORGET word	Forget all entries following 'word".
n DP + !	Leave n locations for an array.

Words concerned with the stack:

l m n DUP	Leave l m n n on the stack
" OVER	Leave l m n m on the stack
" DROP	Leave l m on the stack
" SWAP	Leave l n m on the stack
" ROT	Leave m n 1 on the stack
n .	Type (and discard) number.
a ?	Type number at address a.
a COUNT	Fetch the count field of a string.
a n TYPE	Type n characters, starting at address a.

Words concerned with arithmetic:

n CONSTANT word	Define 'word' so that its value (n) is placed onto the stack.

n INTEGER word	Define 'word' so that the address of its parameter field is placed onto the stack.
n a SET word	Define 'word' to store the number into the address.
a @	Fetch the number from address a.
n a !	Store the number into address a.
n a +!	Add the number into address a.
DECIMAL	Specify number base.
OCTAL	Specify number base.
HEX	Specify number base.
n MINUS	Leaven -n on the stack
n ABS	Leave
m n +	Leave m+n on the stack
m n *	Leave m*n on the stack
m n MAX	Leave max (m, n) on the stack
m n MIN	Leave min (m, n) on the stack
m n -	Leave m-n on the stack
m n /	Leave m/n on the stack
m n MOD	Leave m mod n on the stack
l m n */	Leave l * m / n on the stack
n 0=	Leave if n=0 then 1; otherwise 0.
n 0<	Leave n < 0 then 1; otherwise 0.
m n <	Leave m< n then 1; otherwise 0.
m n >	Leave m > n then 1; otherwise 0.

Words concerned with the interpreter:

WORD	Read the next word in the input string.

QUESTION	Type an error message-the last word read followed by?
n LOAD	Read text from block n.
;S	End of text.
IMMEDIATE	Set the precedence of the last entry to 1.
: word	Define 'word' and begin compiling its definition.
I	Put the current value of the loop index on the stack.
	(the following have precedence 1)
n IF	Skip to ELSE (or THEN) if number is 0 (false).
ELSE	Skip to THEN.
THEN	Mark end of skip.
m n DO	Begin loop; limit (m) and index (n) are placed on the return stack at execute time.
LOOP	End loop; increment index by 1 and stop at limit.
n +LOOP	End loop; increment index by n and stop at limit.
	(the following have precedence 2)
;	End compilation.
;CODE	End compilation and begin assembling code.
EOT	End of input message from terminal. Await input and continue compilation.

Words concerned with the assembler:

CODE word	Define 'word' to execute the instructions in its parameter field.
NEXT	Execute next word-interpreter return address.
PUT	Replace stack with register return address.
PUSH	Push register onto stack return address.
POP	Discard top of stack return address.

BINARY	'POP' then 'PUT' return address.
S)	Index relative to stack.
E)	Index relative to entry.
X)	Index relative to index register.
I)	Specify indirect address.
R)	Index relative to return stack.
I	Location of next entry for interpreter.
DP	Address of next-available-word in dictionary.
condition IF	Jump if condition false; leave address of jump on stack.
ELSE	Jump; leave address on stack; supply address to IF.
THEN	Supply address to ELSE.
BEGIN	Begin loop; leave address on stack;
condition END	Jump to BEGIN if condition false.

Words concerned with secondary memory:

n BLOCK	Obtain core address of block n.
UPDATE	Mark last block 'changed''.
FLUSH	Re-write changed core buffers onto disk.
ERASE-CORE	Mark buffers empty.

Table 1 Words are compiled if their precedence is less than the value of STATE:

		Precedence		
Situation	STATE	0	1	2
During execution	0	execute	execute	execute
Compiling:	1	compile	execute	execute
after IMMEDIATE	2	compile	compile	execute

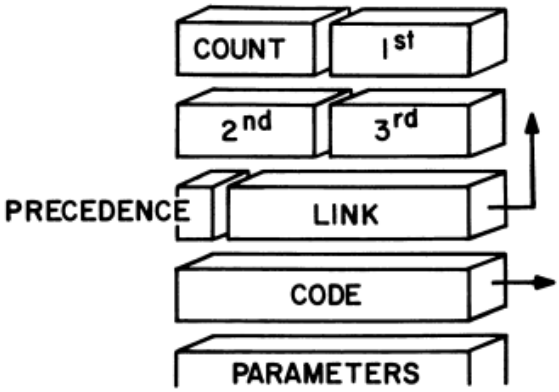

Figure 1 Format of a Dictionary Entry for a 16 bit Computer.

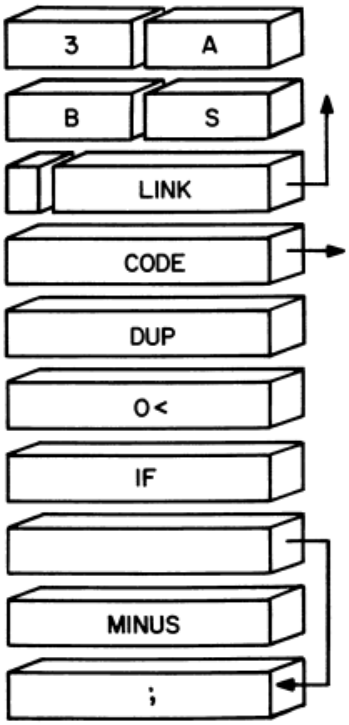

Figure 2 Entry for ';' word ABS. Addresses of entries have been compiled into the parameter field.