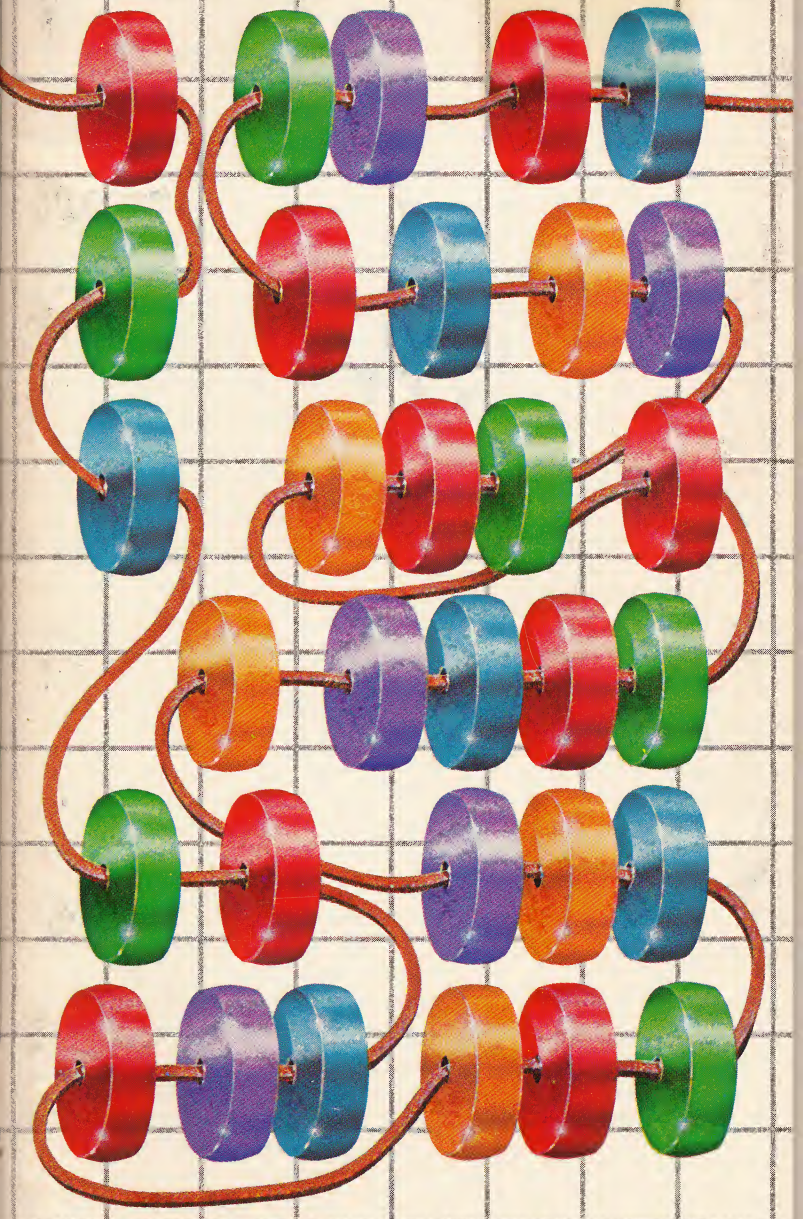
$2.95
in U.S.A.

UNDERSTANDING FORTH

An Introduction and Overview

An Alfred Handy Guide



by Joseph Reymann

UNDERSTANDING FORTH

by Joseph Reymann

AN ALFRED HANDY GUIDE

Computer Series Editor:
George Ledin Jr.

CONTENTS

ALFRED PUBLISHING CO., INC.
SHERMAN OAKS, CA 91403

Editorial Supervision: Joseph Cellini

Cover Design: Paula Bingham Goldstein

Alfred Publishing Co., Inc.
P.O. Box 5964
15335 Morrison Street
Sherman Oaks, CA 91413

Library of Congress Cataloging in Publication Data

Reymann, Joseph.
Understanding FORTH.

(An Alfred handy guide)
Bibliography: p.
1. FORTH (Computer program language) I. Title.
QA76.73.F24R49 1983 001.64'24 83-9406
ISBN 0-88284-237-4

1. WHY GO FORTH?

A computer is a specialized piece of machinery, often referred to as the *hardware*. But without directions, a computer cannot do a single thing. It must be *programmed*, that is, given instructions in a very detailed manner concerning the tasks it is supposed to do. The program, often referred to as *software* to distinguish it from the machine, is a sequence of steps in a highly-specialized code which only the computer can easily understand. To make the job of giving the computer these instructions easier, different types of human-understandable *computer languages* were devised.

The first step up from the direct machine code is called an *assembler*. This level gives the machine code human-like labels, so a programmer can more easily generate the complex codes. The programmer must still generate one assembler instruction for each machine instruction needed.

The next step up is a *high-level language*, one which is more like human communication, and whose procedures are more like human procedures. Each high-level instruction may generate many machine instructions necessary to perform a particular function. FORTH is one of these high-level languages.

There is no faster-running, more efficient computer program than a well-written one in the machine code of the computer on which it runs. Why then would anyone write programs in anything other than machine code?

One major reason is that machine code is oriented to the computer and its machine-like procedures, and is therefore more difficult for humans to understand. The various computer languages are oriented more to human communication and procedures. So, programming in a high-level, human-oriented language is usually easier for the average programmer, and therefore faster. Saving programming time means saving money in commercial programming. It also makes more efficient use of the programmer's time, an increasingly scarce commodity as computers proliferate faster than programmers. These savings and efficiencies are usually achieved at the expense of a program which runs somewhat slower and takes up more memory space in the computer than would an equivalent machine-code program.

One of FORTH's major benefits is that it is a fast-programming, high-level language which remains close to machine code in efficiency and therefore in execution speed. While its words are human language oriented, its procedures are closer to machine-procedural, and it executes very quickly.

FORTH is also the easiest programming method for finding and correcting programming errors (*debugging*) that I have ever used. A major reason for this is that each FORTH word may be executed separately by the computer with the programmer remaining in control. Overall, FORTH is one of the fastest means of producing efficient, compact, tested programs.

Finally, FORTH is the closest language I have seen to machine instructions in allowing the program-

mer complete, explicit control over all the resources of the computer, while still giving the advantages of faster programming which most high-level languages provide.

2. FOR WHAT COMPUTERS IS FORTH BEST SUITED?

The way today's computers are built, FORTH is best suited to the smaller end of the computer-size spectrum: micro- and minicomputers. One reason for this has nothing to do with FORTH itself. The larger machines have many hardware features needed in conventional programming, such as dozens of registers for advanced techniques like indexed addressing and parameter passing. FORTH has other methods of accomplishing the same tasks, so many of the hardware features of the larger machines are not needed by FORTH and are therefore wasted.

One of FORTH's major advantages is its ability to produce compact programs. The larger computers usually have vast quantities of memory, so there is no particular concern with saving memory. Large-system programmers are more concerned with abundant error-checking and other memory-consuming techniques not common in FORTH.

3. THE WHAT AND HOW OF COMPUTER PROGRAMMING

Every computer made today operates internally in a number system based not on ten, as is the decimal system we are all familiar with, but on the binary system in which there are only two digits: 0 and 1. The reason for this is that present electronic devices can recognize only two states: *on* and *off*. Since a computer is essentially a collection of such two-state devices, the computers have to operate in a number system which contains only two digits: the binary system. If you were to visualize a car odometer clocking off the miles in a binary number system, it would count 0, 1, 10, 11, 100, 101, 110, 111, 1000, etc. Each number position is called a *bit*, short for *b*inary dig*it*. Everything in the computer must eventually be represented in binary numbers, both the data on which the computer operates, and the sequence of instructions which tell the computer what to do with the data (the *program*). So, in a particular computer, the sequence 11000011 may be interpreted by the computer program either as the decimal number 195, or as an instruction to jump to the location stored in memory immediately following that instruction.

Programming the early microcomputers was accomplished by feeding in the various program instructions bit by bit from front-panel switches, a very tedious process. Eventually, someone wrote a program which allowed the computer itself to translate more human-oriented expressions, such as JMP, into the bits necessary to cause the computer to recognize an instruction to move (jump) to a different part of the program. As a result, programming could be done much more quickly by typing on a keyboard, even though the actual program was still written in the explicit instruction set which the computer recognized.

In the 1950s, it was realized that the programmer did not necessarily need to know exactly what the computer did at each step. All that was necessary was to get the right answer. An analogy in people-terms might be a teacher asking a student to add a column of figures; the teacher may not care whether a calculator is used, or the sum is obtained with pencil and paper. The first high-level languages (HLLs) were written, allowing programmers to direct the computer in terms quite different from the internal instruction sequences actually executed by the computer. The programmer could now instruct the computer to PRINT SALARY without knowing where in the computer's innards the salary was stored or in what form, or exactly what the machine needed to do to send something to the printer. In short, these languages were *human-procedure oriented.*

There have been many HLLs written, each either tailored to some particular need, or written with some purpose in mind. The most widely known of the early languages are:

- FORTRAN—written to ease computation of scientific formulas;
- ALGOL—written as an international, machine-independent language;
- COBOL—written to ease the handling of business-oriented problems;
- BASIC—written as a student instructional aid.

4. FOR WHAT APPLICATIONS IS FORTH BEST SUITED?

When a program must be written to perform a certain task, it may be written in almost any HLL. It will perform better in one language than in another, but it will work in almost any one. However, each HLL has a type of work for which it excels. One of FORTH's strong points is in controlling equipment, whether machine or electronic. This involves what are called *real-time situations*. There are several factors which contribute to this: FORTH's fast execution speed, the ability to optimize a program by writing critical portions of it in machine instructions, the close control which a FORTH system gives the programmer, and the compact size of FORTH programs.

Where have you seen FORTH at work, perhaps without realizing it? One of the most popular hand-held language translators is programmed in FORTH, as is one of the text processing programs for the IBM Personal Computer. Several video games hide FORTH behind their colorful screens, and FORTH controls at least one communications satellite and the ground stations which support it. The inventor of the latest technique in computer intuition, SAVVY, said his advanced method could be effectively programmed only in FORTH. The list goes on: an airline automated baggage conveyor, a major hospital's pulmonary laboratory, a military battlefield computer, and other real-time situations.

5. ORIGINS AND DEVELOPMENT OF FORTH

Most HLLs are conceived and written by groups of people, with either corporate or academic backing. FORTH is an exception. A single programmer, trained in physics, was frustrated by the HLLs of the sixties. More particularly, he felt that these HLLs and the computers on which they operated forced complexities and constraints on him which were unnecessary and wasteful. Over a period of several years he evolved a system for his own use which allowed him virtually complete control over the resources of the computer. Charles Moore called his system concept FORTH (he considered it a *fourth*-generation system, but the computer on which he then operated could only accept five-letter labels). His first use of the full system provided programming and control for a major observatory telescope.

After his brilliant concepts were published, many individuals, companies, and organizations utilized them and traveled more or less parallel paths in implementing FORTH on various computers. Since the late seventies, there has been a major effort to standardize FORTH implementations from different companies in a manner similar to other languages. This has culminated in the publication of the FORTH-83 Standard by the FORTH Standards Team.

6. FORTH'S BASIC CONCEPTS

There are some fundamentals to consider before we go into the FORTH language.

PARAMETER PASSING

There are many ways in the various HLLs to pass information between various portions of a program.

One of the most common is to use *named variables*: SUM perhaps, or GROSS:SALARY. FORTH has the capability to use these, as well as named constants, like PI.

To create a variable named SUM, type:

```
VARIABLE SUM
```

When a variable is created, most FORTH systems will set it to 0 as a starting value. Creation of a constant is similar, except that you need to furnish the value, as in

```
7 CONSTANT DAYS/WEEK
```

In many high-level languages, you would start a program by naming a group of variables to use in passing data around by name inside the program. The major method of communicating data inside a FORTH program is the *stack*, a sequential last-in-first-out list of data. The stack is usually visualized by analogy to a cafeteria's stack of trays mounted on a spring-loaded carrier. If a tray is added to the top, it pushes the others down. Only the last tray added to the stack can be removed, and the other trays beneath it move up. A data stack operates in similar fashion. If several numbers are put onto the stack, the last one on is at the top of the stack.

FORTH contains several words to manipulate the stack. For example, SWAP exchanges the stack's top two values, and DUP duplicates the top entry. All of the arithmetic operations expect their numbers to be on the stack, and they leave their results on the stack. The FORTH words @ and ! bring data to the stack from memory and return data to memory.

ORDER OF OPERATIONS

Another concept which makes FORTH unusual is the order in which arithmetic operations are done. Most people by now have used one of the popular hand calculators. In some, you operate in the same fashion you did in arithmetic class: to add 1 and 2 you enter 1, press +, enter 2, then press = to obtain the result. This operation, called the *algebraic method*, is not used in FORTH. Rather, FORTH uses a method followed in some other popular calculators, the *Reverse Polish Notation*, named after the country of origin of the mathematician who first proposed it. This method would do the same calculation as follows: 1 2 +. The two numbers are first put onto the stack, and then the operation is executed. This is the manner in which FORTH operates.

FORTH WORDS

Programming in FORTH consists of defining new FORTH *words*, each of which accomplishes a specific function, by combining some of the few dozen predefined FORTH words. For example, the FORTH word EMIT sends a single character to the operator's ter-

minal screen, while KEY gets a character from the terminal's keyboard. Some, such as the arithmetic operators + − * and / , are more or less standard in the computer field. Others, SWAP and EXPECT for example, are unique to FORTH. Some of the words are written directly in the machine code of the computer in which the system resides. These words are called *primitives*; they execute very quickly. Other basic FORTH words, called *secondaries*, are written in terms of these primitives.

Each defined word is placed in a *dictionary*, together with the actions which define that word. The dictionary is a table of words similar to a normal English dictionary. If a word is written directly in machine instructions, then the name of the word is followed by the machine code which does its work. If the word is defined in terms of other FORTH words, the *location* in the dictionary of each of these words follows the new name being added to the dictionary.

A FORTH program is rarely written solely in the word set furnished with the FORTH system. A programmer works in FORTH by defining new words in terms of the already-defined words. For example, let's define our first FORTH word. Suppose we want a word SQUARE which multiplies any number it operates on, by itself. We could define this word to be DUP*. DUP, a predefined word, takes a number from the stack and creates another copy of it on the stack, so that now the stack has two numbers on it, both the same. The FORTH word * multiplies together these top two stack entries. The result is that any number on the stack when SQUARE is executed is multiplied by itself. SQUARE is now usable by any further words defined by the programmer in writing a program.

In this way, the FORTH language grows in a direction unique to each application. Unlike most HLLs, FORTH is *extensible*; it is not limited to a fixed set of words. *Each FORTH program is actually a new language*, written by the programmer for a specific application by defining new FORTH words. The names of these new words are chosen by the programmer, so the final program may be as readable as the programmer chooses to make it. FORTH, then, more than any other HLL, is a series of building blocks which allow the programmer to define his or her own language.

This Handy Guide lists the most common FORTH words in the Appendix, with a brief description of what each does.

FORTH SYNTAX

The concept of a "correct syntax," with precise rules of word order and punctuation, which must be carefully followed in most languages, is virtually nonexistent in FORTH. A FORTH program is merely a group of FORTH words, one after the other, separated by at least one space.

AUXILIARY STORAGE

Another concept bearing special relation to FORTH is that of *virtual memory*. This concept is not

unique to FORTH, but its implementation in FORTH is unusual. If the computer on which FORTH is running has memory resources which include some method of auxiliary storage, usually either floppy or hard disks, or tape equipment, the existence of this equipment is more or less transparent to the FORTH programmer. FORTH includes the ability to store or recall information from disk storage without the sometimes complicated file procedures required by other languages.

Information is stored on disk in *blocks*, 1024 *bytes* (a byte is equivalent to 8 bits) in length, numbered sequentially on the disk. A typical small disk might be capable of holding 350 such blocks. There are no file names, no directory, and no block numbers stored on the disk. Only the information itself is stored, accessed solely by its position on the disk. The programmer calls for 9 BLOCK and the FORTH system brings into the computer's memory the information stored at that position on the disk, for use by the program. This procedure is considerably faster than the cumbersome file procedures of some other HLLs, and it contributes to the very fast execution of FORTH programs. The ability to name programs is still preserved, however, by defining the block containing the program as a constant with the name by which the program is called:

```
38 CONSTANT MY-PROGRAM
```

Then, calling MY-PROGRAM LOAD loads that program into memory. This is considerably shorter and easier than the file procedures in many HLLs.

COMPILERS VS. INTERPRETERS

High-level language implementations are described as being either *compilers* or *interpreters*, depending on how they translate the human-understandable program called *source code* into a machine-understandable one. A compiler is a language which takes the source code and permanently changes it into machine-executable form, or *object code*, which is stored inside the machine. It is that object code which the computer follows when the program is run. The source code is therefore no longer necessary, and does not take up space in the computer. Since the translation process takes place before the user's program is run, the program execution time is not slowed down by this function.

By contrast, an interpreter takes the source code and interprets it into machine code one line at a time, *as the user's program is running*. Since the process of interpreting takes some time, and the interpreter may have to interpret one line of source code many times because the program returns to it often, an interpreted program runs much slower than a compiled one.

FORTH combines the best of both techniques. The definition of a FORTH word is *compiled* into the dictionary, but only a few special definitions (see Section 17 below) are compiled directly into machine code. FORTH words defined in terms of other FORTH words are compiled into a list of locations, or *addresses*. In this way, the time-consuming portion of the translation from human-understandable high-level FORTH words

into machine-understandable code is done before the program is run. No portion of the program run time is wasted on this function. The compiled list of addresses can be easily interpreted when a program is run. Rather than separating words and looking them up to find their proper function, the interpreter in FORTH merely retrieves the address and looks there for the machine code. The time-wasting work has already been done. In this way, FORTH obtains the benefits of both compilation *and* interpretation.

7. NUMBERS IN FORTH

In keeping with FORTH's primary use in control of machinery and equipment, FORTH uses a simplified but adequate method of representing numbers. There are no decimal points in a standard FORTH system. Numbers are integers, usually in a single-length format, which can represent numbers in the range of +32,767 to −32,768, adequate for the majority of uses.

FORTH can perform both arithmetic operations and other manipulations on these numbers:

+ Adds together the top two numbers on the stack.

− Subtracts the top two numbers on the stack.

* Multiplies the top two numbers on the stack.

/ Divides the top two numbers on the stack, leaving only the quotient.

/MOD A division which returns both quotient and remainder.

MOD A division which returns only the remainder, not the quotient.

*/ A very useful scaling operator which first multiplies then divides.

DUP Duplicates the top stack entry.

@ Obtains the contents of the address which is the top stack entry.

! Stores the second stack entry at the address pointed to by the top stack entry.

SWAP Interchanges the top two stack entries.

OVER Duplicates the second stack entry on top of the stack.

DROP Erases the top item from the stack.

ROT Brings the third stack entry to the top of the stack, pushing down the other two.

To provide a greater range of positive numbers, or where algebraic signs (+ or -) are not necessary, unsigned numbers in the range 0 to 65,535 may be used, with their own operators:

U* Multiplies two stack entries without considering sign.

U/MOD An unsigned division giving quotient and remainder.

U. Prints out the top stack entry, without considering sign.

Some operations require double-length numbers, and most FORTH systems have the ability to work with these. Double-length numbers lie in the range of +2,147,483,647 to −2,147,483,648. There are some similar operators to manipulate these numbers:

D+ Adds two double-length numbers on the stack.

D- Subtracts two double-length numbers on the stack.

2DUP Duplicates a double-length number on the stack.

2SWAP Interchanges two double-length numbers on the stack.

2DROP Erases a double-length number on top of the stack.

2OVER Reproduces the second double-length entry on top of the stack.

In a few cases, where a very narrow range of numbers is needed, half-range numbers may be used to conserve space. If signed numbers are necessary, these numbers may range from −128 to +127. If unsigned numbers are used, the range is 0 to 255. These numbers take half the storage space in the computer as full-range numbers, and one-quarter the space of double-length numbers. Since they take up the same amount of space as an alphabetic character (one byte), they have a set of character operators to manipulate them:

C! Stores a byte-value from the stack into memory.

C@ Obtains the byte contents of an address on the stack.

There are no separate arithmetic operators for these numbers.

In a FORTH system, a number is entered onto the stack merely by typing it. With your FORTH system running, typing 100 followed by RETURN or ENTER will cause the number 100 to be put on top of the stack so that you can operate on it. If you wanted to store it from the stack into a variable, for example, you would enter the name of the variable, followed by ! .

There are two ways to do arithmetic operations in a computer: one is by writing a mini-program to do them in software; the other is by means of a special hardware component which handles math directly, like a calculator. On computers which contain calculating hardware (more and more computers now do), it is easy to extend FORTH's number range to include decimal fractions such as 234.5678. But doing decimal calculations in software slows down program execution considerably, by as much as a factor of ten to a hundred. Since FORTH's basic tenet is improving computer effi-

ciency, the standard FORTH system operates on integer numbers.

8. DISPLAYING RESULTS

FORTH can send its results to either a terminal screen or a printer. Although the various systems available have different ways of doing this, some output methods are common to most of them.

All numbers inside FORTH are stored as binary values. For ease of interaction with its human operator, FORTH can be told to display or print numbers in one of several number bases. The usual ones are DECIMAL, HEX (base 16), OCTAL (base 8), and BINARY.

The FORTH word . will display the data value on top of the stack as a signed number in whatever base the system has been told to utilize. It is a *destructive print*, which means that it removes the value from the stack. A *non-destructive print*, which leaves the value on the stack, may be obtained by using DUP . in the program.

The word ." will display or print a *string* of characters, such as headings of a table, operator messages, and such.

FORTH also has special formatting words to allow display or printing of unusual numbers, such as double-length values or numbers containing non-numeric characters (dollar sign, dashes for telephone or social security numbers, etc.).

9. DEFINING A FORTH WORD

We said earlier that programming in FORTH consists of defining progressively more powerful words built pyramid-like onto words defined earlier. This characteristic, shared by few other languages, is called *extensibility*. The programmer is not limited to a given word set; he or she is free to construct the one most helpful in solving the problem at hand.

The start of a FORTH definition is the colon. In fact, words written in high-level FORTH (as opposed to machine code-defined words) are also called *colon definitions*. The end of a FORTH colon definition is marked by a semicolon.

Let's look at our earlier FORTH definition written formally and see what it means:

```
: SQUARE   DUP *  ;
```

The opening colon informs FORTH that this is the definition of a new word to be added to the dictionary. SQUARE is the name of the word. After compiling a heading, FORTH searches the dictionary to find the location of each of the words making up the colon definition, and compiles the address of each word into the dictionary after the new name. So, after the heading for SQUARE, the dictionary addresses of the FORTH words DUP and * would be stored. Then, the ending semicolon tells FORTH to stop compiling. The dictionary entry for each word, therefore, is (1) a heading containing the full or truncated name of this word, (2) a backward link to the word before it in the dictionary, and (3) the addresses of all the previously-defined words which make up the action to be taken by this new word.

When the word SQUARE is executed, either within a running program or when typed in by the operator, FORTH searches its dictionary to find SQUARE. Once that is found, the address interpreter starts executing the words which define the operation of SQUARE. It extracts the address of the word DUP and executes it. This code duplicates on the stack whatever data entry is on top of the stack, giving two identical data entries on top. Then FORTH extracts and executes the code at the address of * , multiplying these two stack values together. The net effect, of course, is that SQUARE multiplies the top stack value by itself, the mathematical operation called squaring.

Each FORTH word has a name. About the only rules for choosing names are:

- they may contain any ASCII character except space or carriage return;
- they must be unique;
- they must conform to the length limitation (usually 31 characters) of the FORTH system in use.

Any printable character may appear in a name. The following are all valid FORTH names:

```
SUM                    Z
CONTACT                L123*
@                      +(*&*!
```

However, a few cautionary notes are in order. Some systems allow the programmer to control the portion of a FORTH name which is stored in the dictionary. Such systems may use, for example, only the first three letters plus a count of all letters in the name. In these systems, care must be exercised to avoid inadvertent duplication; the words CONTACT and CONTROL could both be stored in the dictionary as 7CON (count of characters plus the actual first three characters). If CONTACT was entered into the dictionary first, followed directly by CONTROL, all references to CONTACT would actually be compiled as references to CONTROL, since the dictionary is searched from most recent entry to oldest entry. Some systems which truncate a FORTH word in the dictionary contain a variable, WIDTH, which allows the programmer to override the truncation by selecting how many letters will be used in the dictionary entry.

Another caveat in naming words is that you should avoid choosing names which consist entirely of digits. If you need a word for which the obvious name is 100, use HUNDRED instead so that 100 will still be available to enter that number onto the stack.

Similar names may be used if a distinguishing feature is put at the beginning, before any possible truncation: 1TABLE, 2TABLE is fine.

Note that there are no "invalid" names, as that term is used in many languages. You may even violate the naming guidelines, as long as you accept the inevitable results. For example, you may deliberately choose duplicate names for some valid purpose.

This freedom to choose names of FORTH words means that a program may be as readable or as obscure as the programmer chooses. We'll see that in the examples below.

For efficient use of computer memory, FORTH itself uses many single-character names for often-used primitives or words. Each of the following symbols are FORTH words:

```
!  -  *  (  +  =  @  :  ;  /  <  ?  >  .  '
```

10. EXAMPLE #1

For practice, let's define a word that does even more work than SQUARE. Suppose we want to evaluate a quadratic equation of the form

$$Y = AX^2 + BX + C$$

by typing in any number we select. Let's assume that A = 3, B = −2, and C = 7. In FORTH, we would write a word QUAD to do that:

```
 3 CONSTANT A
-2 CONSTANT B
 7 CONSTANT C
 : QUAD   DUP >R   A *   B +   R> *   C +   .   ;
```

If we wanted to evaluate this equation for an X-value of 4, we would type in 4 QUAD, and the computer would immediately display the answer 47.

Let's see how this works. Typing 4 (or any number) puts that number on top of the data stack. DUP makes a copy of it, and >R sends the copy of 4 to a special stack called a *return stack* for temporary storage. At this point, both the normal data stack and the return stack have 4 (the value of X) on top. Executing the name of a constant calls its value to the stack, so executing A puts the value of 3 on the data stack. Now, the top values of the data stack are 4 and 3. * multiplies them together, leaving 12 on the stack. B + puts the value of −2 on top of the stack, then adds it to 12. The value on top of the stack (10) now represents AX + B. R> brings back to the data stack the value of X (4) we put there temporarily, and * multiplies that by the AX + B value beneath it on the stack. We now have a value of 40 on the stack, representing X(AX + B) or AX^2 + BX, and executing C + adds the value of 7, completing the calculation. The word . prints out the answer on the system's terminal.

11. HOW YOU WRITE A FORTH PROGRAM: EXAMPLE #2

Suppose you were assigned the task of automating a kitchen dishwasher, to be run by a FORTH program. A first step would be defining the problem in its broadest terms: to wash dishes—drain, wash, rinse twice, dry.

Let's define that as a FORTH program:

```
: WASH/DISHES   DRAIN  WASH  RINSE  RINSE
    DRY ;
```

That is the top-level operating program. Continuing the FORTH programming process consists of increasing the definition of the program until you get to words which actually manipulate hardware components. So, the next step might be:

```
: DRAIN   DRAIN/VALVE ON   PUMP REVERSE
    30 SECONDS   RESET ;
: WASH   FILL   DETERGENT ADD   AGITATE
    DRAIN ;
: RINSE   FILL   AGITATE   DRAIN ;
: DRY   HEATER ON   15 MINUTES   RESET ;
: FILL   FILL/VALVE ON   ?LEVEL
    FILL/VALVE OFF ;
: AGITATE   PUMP FORWARD   5 MINUTES
    RESET ;
: RESET   FILL/VALVE OFF   DRAIN/VALVE OFF
    PUMP OFF   HEATER OFF ;
```

The final step would be to define the machine-specific functions (on/off, forward/reverse) to manipulate valves, detergent-tray, and motor controls, and the timing terms SECONDS and MINUTES. We will assume that the machine designers have told us we have computer ports 60 to 66 through which we can operate valves and controls and read levels. A value of 1 turns a control on (-1 for pump reverse), and 0 turns it off. For the timer, a non-zero value creates a delay of that number of seconds. The level port indicates 0 when not full, 1 when full:

```
60 CONSTANT FILL/VALVE
61 CONSTANT DRAIN/VALVE
62 CONSTANT PUMP
63 CONSTANT DETERGENT
64 CONSTANT TIMER
65 CONSTANT HEATER
66 CONSTANT LEVEL
: ON   1 OUTPUT   ;
: OFF   0 OUTPUT   ;
: SECONDS   TIMER OUTPUT   ;
: MINUTES   60 * SECONDS   ;
: FORWARD   ON   ;
: REVERSE   -1 OUTPUT   ;
: ?LEVEL   BEGIN LEVEL INPUT UNTIL ;
```

What we have done here is to first specify the system level functions, then progressively further define each of them until we reach something that is executable by the computer. This style is called *top-down programming*. In recent years it has become the favored method of program design because it leads to conceptually correct, efficient solutions.

However, note that FORTH words are defined in terms of previously defined FORTH words. Therefore, *entry* of the program must be in reverse order, so that the word is already defined in the dictionary when used in another word. The constants which designate machine-control ports are defined first, then the words which directly manipulate them, and progressively up to the last definition entered, WASH/DISHES.

Notice that in writing the program, we have actually written a *new language*, specific to this particular application. This is a major characteristic of FORTH: rather than use the same commands throughout a program, each program is actually a distinct language tailored to that specific job.

Note also that the amount of work done by successive individual words increases. In most of the high-level languages, a program line of code near the end of a program does about the same amount of work as a line near the beginning. A FORTH program is a pyramid constructed on the foundation of primitives, with each level of the pyramid doing progressively more work than the lower one.

12. READABILITY

Since a word must be defined before it is called inside another word, the program above would be typed into the computer in the reverse order as described above. The completed program would look as follows:

```
60 CONSTANT FILL/VALVE
61 CONSTANT DRAIN/VALVE
62 CONSTANT PUMP
63 CONSTANT DETERGENT
64 CONSTANT TIMER
65 CONSTANT HEATER
66 CONSTANT LEVEL
: ON   1 OUTPUT  ;
: OFF  0 OUTPUT  ;
: SECONDS   TIMER OUTPUT  ;
: MINUTES   60 * SECONDS  ;
: ADD   DUP ON   10 SECONDS   OFF ;
: FORWARD   ON  ;
: REVERSE   -1 OUTPUT  ;
: ?LEVEL   BEGIN   LEVEL INPUT   UNTIL ;
: FILL   FILL/VALVE ON   ?LEVEL
    FILL/VALVE OFF ;
: AGITATE   PUMP FORWARD   5 MINUTES   RESET ;
: RESET   FILL/VALVE OFF   DRAIN/VALVE OFF
    PUMP OFF   HEATER OFF ;
: DRAIN   DRAIN/VALVE ON   PUMP REVERSE
    30 SECONDS   RESET;
```



```
: WASH   FILL   DETERGENT ADD   AGITATE
    DRAIN ;
: RINSE   FILL   AGITATE   DRAIN ;
: DRY   HEATER ON   15 MINUTES   RESET ;
: WASH/DISHES   DRAIN  WASH  RINSE  RINSE
    DRY ;
```

The underlined words in the program above are the only words, aside from the colon and semicolon, which are part of the generic FORTH language. The remaining ones are chosen by the user. The resulting program is very descriptive. At every stage, it is easy to understand what is going on.

Suppose you were a programmer tasked with modifying this program. Wouldn't it be easy to determine at each step what the program was doing? Although the example is purposely chosen to be more simple than most FORTH programs, it illustrates that FORTH can be quite an aid to readability.

13. DEFINING DEFINING WORDS

FORTH has several words which define other words. Among these are : (colon) ,CONSTANT, and VARIABLE. We have already discussed the function of : . CONSTANT creates a class of words whose *value* is put on the stack whenever its name is invoked. For example, if we had defined

```
314159 CONSTANT PI
```

in a program, whenever we put the word PI in the program, the value 3.14159 would be pushed onto the stack.

VARIABLE creates a class of words whose *address* is put on the stack when the name is invoked in a program. If we had declared

```
VARIABLE SUM
```

then each time SUM was used in a program, the address where the SUM was stored would be pushed onto the stack.

FORTH also has the ability to create new defining words, which themselves define other words. These may be used, for example, to create new data types or to create new classes of defining words.

Suppose in a navigational program, we needed to be able to reference vectors. These vectors have two values, a direction and a magnitude. We can write a new defining word VECTOR, which will allow us to name a new class of variables, each having the same behavior when its name is invoked:

```
: VECTOR   CREATE , ,  DOES> 2@  ;
```

VECTOR may now be used to define new words. Both a *compile-time* and a *run-time* behavior are defined

for VECTOR. When a new vector is being defined, the CREATE portion of the definition is followed by the FORTH system. When the word is executed during running of a program, the DOES> portion is followed. During compilation, a phrase like

```
41 270 VECTOR WIND
```

will CREATE an entry named WIND in the dictionary and store (, ,) the two values after the name. When the program is run, each VECTOR has the property that whenever its name is invoked, the two numbers which define it will be placed on the stack (2@). Other VECTORs may be similarly defined for CURRENT, 1VESSEL, 2VESSEL, and so on. (In some FORTH systems, BUILDS> is used instead of CREATE.)

Although this example is intentionally simplified, the technique of defining new defining words is a very powerful and unique FORTH feature. Among other things, it allows definition of the different data structures of the computer scientists: lists, queues, arrays, linked lists, trees, and others.

14. FORTH DECISIONS AND STRUCTURES

Almost every computer program must have the ability to *branch*: to make "decisions" by testing a value and, depending on the result of that test, choose one of two or more program paths to continue along. FORTH includes several methods of making such decisions.

One technique is to use comparison tests. The basic tests are purely mathematical: > (greater than), < (less than), = (equal), 0< (less than zero), 0> (greater than zero), and 0= (equals zero) work on the values of mathematical data on the stack. These tests take one or two values from the stack and return a value of 1 (or some non-zero value in some FORTH systems) on the stack if the test is met, and a value of 0 if the test is not met.

Other words make use of these tests. The major word is IF, which looks at the value on the stack. A non-zero value will cause execution of the words after IF, and a zero value will execute the words after ELSE (if there is an ELSE) or will continue after THEN.

This allows another technique of branching: rather than using a comparison test to generate a value for testing, we can use the results of a computation. Since the words IF and UNTIL test for a non-zero value, any computation which can produce a zero or non-zero value can be used to generate values for branching. We'll see an example of computed branching values in the next section.

Many programs need to repeat a portion of the program until some condition is satisfied. There are several structures supported in FORTH for this purpose:

DO-LOOPS	A method of repeating a loop a known number of times.
BEGIN-UNTIL	An indefinite loop which repeats until some computed or tested value on the stack is non-zero.
BEGIN-REPEAT	A variant of BEGIN-UNTIL which supports a conditional exit at an interim point specified by WHILE.

For example, in our program in Sections 11 and 12 above, we used a word named ?LEVEL. This word was used to indicate a wait until the washer was full of water. It did this by repetitively checking a part of the washing machine which would read 0 until the tub was full, then read a non-zero value. The word which used the value input was UNTIL, which looks at the data value on top of the stack. If it is 0, UNTIL causes the program to jump back to BEGIN; if non-zero, it continues with whatever program code occurs after UNTIL.

15. EXAMPLE #3

Let's look at a method of using one of FORTH's structures to perform a repetitive calculation. In mathematics, it is sometimes useful to calculate the largest number which will divide evenly into two other numbers: the greatest common divisor (GCD). A method described by the ancient Greek mathematician Euclid provides a quick solution, using a repetitive division with the remainder as the new divisor, continuing until the remainder is zero. A FORTH program to compute GCD is shown below.

```
: HEAD   2DUP  CR  .' The GCD of ' . .' and
    ' . .' is ' ;
```
(puts higher number on top)

```
: ADJUST   2DUP >   IF SWAP THEN  ;
```
(prints head and adjusts number)

```
: GCD   HEAD  ADJUST
    BEGIN  OVER MOD  SWAP OVER  0= UNTIL
    . DROP  CR ;
```
(print GCD, drop remainder)

Use GCD by typing two numbers between 1 and 32,767 in any order, then GCD. For example, typing:

```
432 32760 GCD
```

produces the answer:

```
The GCD of 32760 and 432 is 72
```

Read the program from the bottom up. In GCD, first HEAD prints the legend, and ADJUST orders the two input numbers so that the lesser number is on top of the stack. Then the loop is started. OVER duplicates the second stack entry on top of the stack, and

MOD divides the two top stack entries leaving only the remainder (no quotient). SWAP adjusts the order for another turn at the loop, if necessary, and OVER gets the value of the remainder to test it for 0. The loop repeats until the remainder is 0, then prints the divisor, discarding the remainder.

16. EFFICIENCY AND OPTIMIZATION

Once a program is written, it is often helpful (or even required) to optimize its execution. Every HLL, including FORTH, produces programs which run slower than optimum machine-language programs. Although FORTH's time penalty is less than many HLLs, some critical operations must run with peak efficiency. Some HLLs do not allow writing program segments directly in machine code. It is difficult to optimize these programs. FORTH, however, makes this quite easy, by allowing the easy incorporation into the FORTH program of small segments written in machine code.

17. LEARNING CODE : EXAMPLE #4

In most programs written in a high-level language there is some portion where the computer spends most of its time, or which is time-critical. It is extremely helpful in increasing program run-time efficiency if these portions can be written in machine code, which has the fastest execution speed of any method.

FORTH has a simple interface to machine code. We learned above how to write colon definitions of FORTH words. However, FORTH words can also be defined directly in machine code. Most FORTH systems contain an assembler to make writing such words easy.

Let's look at an example. If we needed a word to multiply by 10 any number given to it on the stack, we could write it briefly in high-level FORTH like this:

```
: 10TIMES   10 * ;
```

This word is defined in a very concise way, and it will execute reasonably quickly. However, sometimes reasonably quickly is not quick enough. Since multiplication and division are often among the slowest operations in many computers, we might need to save some time in the program which uses 10TIMES. We can rewrite this operation in machine code, as in the following sample code for an 8080/Z80 FORTH computer:

```
CODE TIMES10  H POP  H PUSH  D POP
    H DAD  H DAD  D DAD  H DAD  H PUSH
    NEXT JMP
```

This code puts the number from the stack into two registers, then doubles one of them twice to multiply by four, adds in the number from the other register to achieve multiplication by 5, then doubles that product for multiplication by 10.

By measurement, the CODE definition TIMES10 runs several times as fast as the high-level 10TIMES. While most speed improvements won't be that dramatic because FORTH is so fast, this example provides us an illustration of optimizing a FORTH program by coding a small segment of it in machine code. When the machine-language segment is in the most-often-executed portion of a program, the execution time of the program can be dramatically improved.

Suppose we wanted to write a program to search through an entire 64K computer memory to find and print out the address of all locations where a certain two-byte data value is stored. We could write a high-level FORTH program to do this. Assuming we have placed the two-byte value on the stack:

```
: HSEARCH   0 0 DO  DUP I @ = IF I . CR
    THEN LOOP DROP ;
```

If we run this program and measure the time consumed, we find that most of the elapsed time is taken stepping through the DUP I @ = portion of the loop. If we can code that portion in machine language, the loop will run faster. Let's try, again using an 8080/Z80 FORTH computer as our example:

```
CODE COMPARE   R LHLD  M A MOV  H INX
    M H MOV  A L MOV  M E MOV  H INX  M D MOV
    H POP  H PUSH  A XRA  D SBC  A H MOV
    A L MOV  +1 JRNZ  H DCX  H PUSH  NEXT JMP
: MSEARCH    0 0 DO  COMPARE  IF I . CR
    THEN LOOP DROP ;
```

This version of SEARCH will take only two-thirds the time of the high-level-only definition. There are two types of time savings. The first is the time involved for FORTH's address interpreter to step to several high-level words, not necessary in the second definition. The second is the savings in fewer machine instructions being executed in the machine-language version than in the high-level one.

These two examples illustrate the ease with which a FORTH program may be optimized to minimize execution time. This ability to switch from high-level to machine code and back very easily is one of the unique advantages of FORTH. Typically, a programmer will write an entire FORTH program in FORTH words. Then, after testing, he will analyze the program to look for places to optimize, and replace those FORTH colon definitions with code definitions. As much of the program as is necessary may be written in machine code, with the balance left in high-level FORTH. In this way, the benefits of high-level programming are retained, but the overall program runs more efficiently.

18. RECURSION: EXAMPLE #5

In human language, it is considered improper to define something in terms of itself: "Food is something you get at a food store." With computers, it is sometimes very helpful to program something in terms of itself: a word which calls itself. This technique is called recursion, and it is aided by FORTH's stack-oriented operation.

Consider the mathematical operation of computing a number's factorial, the product of all numbers from that number down to 1. The mathematical symbol for this is ! : 4! would be 4 X 3 X 2 X 1. Computer people talk in terms of *algorithms*, which merely means the method of computation. One algorithm for this computation would be a loop structure similar to our example above, based on the following:

```
N! = N * (N-1) * (N-2) * (N-3) * . . . * 2 * 1
```

A different algorithm using recursion might be:

```
N! = N * (N-1)!
```

Recursive algorithms such as this sometimes have the advantage of taking fewer instructions to perform the work. Since the FORTH word ! has been defined to mean something else, we'll choose a different name.

```
: FACTORIAL   ABS  DUP 1-   DUP 2 = NOT
    IF  FACTORIAL  THEN  *  ;
```

Typing in 7 FACTORIAL U. causes the computer to display 5040 (assuming the BASE is DECIMAL).

Note that the definition of FACTORIAL contains a call to itself. What this definition will do, when given a number (ABS makes certain the number is positive), is to compute the next lower number and, unless it is 1, repeat this process. The stack will therefore contain a series of numbers from N down to 2 (since multiplying by 1 doesn't change a number), which are then multiplied together and the product displayed.

In some FORTH systems, you cannot call a word until its definition is complete (after the semicolon). In these systems, there may be a word RECURSE or MYSELF which you can use to force the same action.

19. INTERRUPTS AND INTERRUPT-DRIVEN SYSTEMS

Programming often requires keeping track of several activities at once, servicing each activity as needed. This technique is called *multitasking*. For example, at a satellite ground control station, you might have the following:

- a clock display that needs to be updated every second;
- a radio receiver that needs to be read whenever a piece of data arrives from the satellite;
- an operator keyboard which needs to respond whenever the operator presses a key;
- an antenna which is rotating horizontally and vertically at the same time, toward the aiming points the computer has given it.

The occurrence of these events is at relatively unpredictable times as far as the computer is concerned, since the computer may be executing a million instructions each second. One technique for handling this situation is to check each activity periodically to see if it is ready for more processing; this is called *polling*. Its problem is that an activity might require immediate attention while the computer is polling or servicing another activity of lesser importance. It is helpful to have a technique which will notify the computer when an event requiring processing has happened. Many computers have the capability to accept such *interrupt signals*, which literally interrupt what the computer is doing and force it to service the particular activity which needs immediate attention.

FORTH easily accomodates interrupt systems, although FORTH systems from different suppliers handle interrupts in different ways. Such systems can also *prioritize* the incoming interrupts so that the more important gets serviced first. Because FORTH executes faster than the code produced by some other high-level languages, it is a preferred method of programming interrupt-driven systems.

20. TESTING AND DEBUGGING

In any computer program of significant length, the complexity will usually lead to one or more programming errors. Each portion of the program must be tested to find and eliminate these errors or *bugs*. FORTH, more than any other HLL, makes testing and debugging easy.

In many HLLs, the programmer switches back and forth between various utility programs in generating a program. He might, for example, use an editor to construct the program's source code, then a compiler to translate the source into object code, then switch to an operating system with a debugger to test run the program with some diagnostic tools. Most FORTH systems operate initially in an interactive mode, where the user has access to all of these system utilities while developing the program. This means that, each time you define a new word, you may immediately execute it. If a word requires two data values on the stack, you can place dummy values of the correct range there, execute the word, and check the result to see if it is what you expected.

An average FORTH programmer will type in a series of definitions which make up one minor module

of the program. (Remember, it is typed in bottom-up.) Then, each successive level of that module will be executed to test it, word by word, with the results carefully checked. The ease of testing, changing, and retesting induces the programmer to test as he goes along. In this way, the entire program may be tested from the ground up, with assurance that each higher level will rest on a thoroughly tested foundation.

There is no switching back and forth among the utilities; they are always present and available. This alone is a considerable time saver during the program development phase.

21. TRANSPORTABILITY

The computer industry periodically considers the problem of *transportability*: installing a program written to run on one computer on a different model. Why?

In the past twenty years, there has been a tremendous change in computer equipment and capabilities. The room-filling computer of 1963 is replaced in 1983 by a more powerful integrated circuit measuring only one by three inches. The average commercial life of a computer may be three years or less. This creates a considerable problem with computer programs and data bases. A computer user is faced with a dilemma: he either sticks with his present computer as it becomes technologically obsolete, or he transfers to a new computer with greater power. The new computer rarely has the same instruction set, so the user will probably have to throw out his programs, perhaps salvaging some of his data base with considerable effort and money.

It would be preferable if the same computer programs which manipulated the old computer, could be used on the new one. FORTH offers a possible solution.

Some professional FORTH systems are written in FORTH itself. Transporting such a system to another computer consists primarily of writing the primitives in the new computer's assembly language, and optimizing the machine-coded portion of the system to the instruction set of the new machine.

In the new FORTH system, the part of it which drives the peripheral equipment (printer, disk storage, etc.) will probably need to be re-written because the equipment is different. However, the user's programs should be transportable to the new machine because the FORTH system overall appears the same to a user program.

With the release of the FORTH-83 standard, mentioned in Section 5, this possibility of transportability is greatly enhanced.

22. ROMABLE PROGRAMS

When a computer is dedicated to only one specific task, such as control of a piece of machinery or equipment, its program is often stored in ROM (Read-Only-Memory), a type of semi-permanent computer storage which cannot be accidentally overwritten or erased in normal computer operation. Several features of FORTH greatly facilitate such ROMing of programs, where saving memory space usually has a cost benefit.

First, a FORTH program is naturally compact. Each use of a defined word uses only a single-address space of two bytes. This corresponds to a *subroutine call* in machine language, which might take three bytes. In that sense, FORTH can be even more compact than machine code. This efficient use of memory space means that less program storage space is needed.

Second, FORTH's extensibility means that the power of successive FORTH words increases, so that fewer program lines may be necessary compared to other HLLs or machine code.

Third, the entire FORTH system may not be needed in a particular application. For example, the assembler and compiler are needed only during program development. Why include them in the ROM, where they take up space without benefit? In FORTH, the nucleus system may be pared down to the absolute essentials, eliminating any unused sections of code: printer and disk drivers, for example, are not needed if the machine being programmed will have no printer or disks attached. Even unused primitives in the nucleus may be omitted from the final product, so that not a single excess byte is used. This allows considerable space savings. It is possible to have the FORTH kernel (the basic FORTH system) pared down to less than one thousand bytes of memory in an application.

23. INSIDE FORTH

As we've seen, FORTH is a language which operates on *addresses*. Your computer's memory is very similar to the pigeon holes of a roll-top desk. Each pigeon hole has an address, which is merely a sequential number. You can store data into any pigeon hole by locating it by number.

When you define a new FORTH word and enter it into the computer, it is stored starting at some particular address. While you, for convenience, are allowed to refer to a word by name, the computer uses the address of that word to refer to it.

When a new word is being defined, the FORTH system searches its vocabulary for each of the earlier-defined words which is used in the new definition. The address of each of those words is stored into the new definition. Let's use our word SQUARE as an example. When the programmer defines this word, its name is added to the end of the dictionary, followed by the *addresses* of the words DUP and * .

The heart of the FORTH system which executes a program is the *address interpreter*. Since any computer can only run its own instruction set, all high-level FORTH definitions eventually have to lead to executable machine code. Definitions of FORTH words are either machine code or addresses of other FORTH

words, so FORTH needs a method of interpreting the address and getting to executable code.

A major portion of the address interpreter is the section called NEXT. This is the part of the system which obtains the next FORTH word to be executed and directs the computer to its location. Depending on the computer and the method of FORTH implementation, NEXT may be one machine instruction or several in length. This is of critical importance, because NEXT is the most often executed portion of the FORTH system during the running of a program. Its efficiency, therefore, determines how close to the execution speed of machine code a FORTH program can come.

Since FORTH stores only a two-byte address for each use of a word after it is defined, it is possible to write very compact programs. Even in machine language, most calls to subroutines take at least three bytes. Comparing equally well-written programs in the major high-level languages, a FORTH program will often take less memory than one in another HLL.

In accordance with the goals of its inventor—simplicity and lack of constraints—FORTH typically does not include the depth of error-checking common in some other HLLs. If, for example, a certain FORTH word needs two operands on the stack to operate correctly, nothing in the basic FORTH system checks to make sure that there are in fact two operands on the stack before the word is executed. However, such error-checking can be built into a given application by the programmer.

One of the major advantages of FORTH is that the programmer is more in control of the computer and its resources than in most other HLLs. While retaining most of the benefits of a high-level language, FORTH remains much closer to the machine-code level, so the programmer has control over the system.

A characteristic of FORTH most appreciated by system programmers is the ease of defining new data types. Rigid typing of data is not usually built into the basic FORTH system, as it is in, say, Pascal. However, the typical data structures of computer scientists—lists, arrays, queues, linked lists, and trees—can easily be accommodated. In addition, new types and structures can easily be implemented.

24. FORTH'S ADVANTAGES AND DISADVANTAGES

In summary, let's list the major advantages of FORTH compared to machine language and other high-level languages:

1. fast programming;
2. faster execution than many high-level languages;
3. ease of testing;
4. most compact programs of any method;
5. closer to machine-language control of computer resources;
6. allows advanced programming techniques (recursion and re-entrancy);
7. easily accomodates multiple users;
8. ease of optimization;
9. ease of interface with machine-code segments;
10. more readable programs possible;
11. promotes structured, modular, top-down programming.

To provide a balanced approach, let's include FORTH's disadvantages:

1. little error-checking built in;
2. less hand-holding to prop up the programmer;
3. does not co-exist well with other programming systems on the same computer, so all concurrent users must be working in FORTH;
4. few string-handling facilities built into the basic FORTH system.

25. FORTH-LIKE VARIANTS

There are several other languages which behave more or less like FORTH. STOIC is a system which operates under a computer operating system called CP/M. IPS (Interpreter for Process Structures) is a FORTH offshoot developed in Germany for uses in real-time situations, such as satellite control. URTH was developed at the University of Rochester. SNAP is a special version of FORTH developed for a handheld computer.

26. HOW TO GET STARTED IN FORTH

For those interested in learning more about FORTH, the next step is to read the book *Starting FORTH* (see Bibliography), a most readable self-instruction book. If you have access to one of the popular microcomputers, a FORTH system is probably available for it (see the next section). It is preferable to read the book with an operating FORTH system handy, so that you can type in the book's examples and see the results yourself. However, remember that the book was written to cover general principles, not a particular implementation, so your system may differ slightly from the book.

27. SOURCES OF FORTH SYSTEMS

There are several companies and sources for FORTH systems. Some are professional systems, highly refined to be fast and compact; some are hobbyist systems, to teach the principles and let you write your own programs. With some, you get a disk which runs immediately on your computer, while with others, some work may be necessary to install or customize the program to the particular system components you have.

The following partial list is a suggested starting place. It is reasonably up-to-date as of the writing of this book. There may be other companies now offering FORTH systems, and some of the ones listed below may no longer offer products. A recent computer magazine may give you later information. The list is compiled from available literature and advertising materials, and no endorsement is implied.

FORTH systems are available for the following computers: Alpha Micro; Apple; Atari; Attaché; Commodore Pet and VIC-20; CPM; Cromemco; Data General Nova; DEC LSI-11, PDP-11, and VAX; Eclipse; Heath H-89 and Z-100; Hewlett-Packard HP-85 and System 200; IBM 370, 470, and Personal Computer; Jonos C2100; KayPro; North Star Horizon and Advantage; Osborne; Pace; Radio Shack TRS-80; Rockwell AIM; Texas Instruments; Vector Graphics; Xerox 820; Zenith Z-89.

FORTH is also available for computers based on the following CPU chips: 1802, 8080, Z80, Z800, Z8000, 6502, 6800, 6809, 8080, 8086/8, 9900, 68000

FORTH SUPPORT GROUPS

FORTH, Inc., 2309 Pacific Coast Highway, Hermosa Beach, CA 90254

The company started by the inventor of FORTH, specializing in professional-level FORTH systems, with utility programs for mathematics, trigonometry, data-base management, graphics; custom programming services. Computers supported: IBM PC, PDP/LSI-11, 8086/8, 8080/5, Z80, 68000, Intel, Omnibyte.

FORTH Interest Group (FIG), P. O. Box 1105, San Carlos, CA 94070

A user's group which publishes *FORTH Dimensions*, an interesting journal of FORTH development and programs.

Mountain View Press, P. O. Box 4656, Mountain View, CA 94040

A retailer specializing in FORTH, with literature and FORTH systems for many computers.

LIST OF FORTH SYSTEM SUPPLIERS

Armadillo Int'l. Software, P. O. Box 7661, Austin, TX 78712

FORTH for the IBM PC and TRS-80 Color Computer.

Capstone Computing, Inc., 5640 Southwyck Boulevard, Toledo, OH 43614

FORTH for Nova and Eclipse.

Computer Methods, 7822 Oakledge Road, Salt Lake City, UT 84121

CometFORTH for Cromemco computers.

ComType,Inc., P. O. Box 374, San Dimas, CA 91773

Supplier of professional and consumer systems for Z80 and Z800 computers under license from FORTH, Inc.; custom programming services. Computers supported: North Star, Otrona Attache, Jonos C2100, STD-buss-based Z80 systems, and other Z80 systems.

CP/M Users Group, 1651 Third Avenue, New York, NY 10028

STOIC for microcomputers.

Creative Solutions, Inc., 4801 Randolph Road, Rockville, Maryland 20852

MultiFORTH for 6800-based computers, including Hewlett-Packard 200-series.

FORTHright Engineering, Inc., 7901 E. Boojum St., Tucson, AZ 85730

HyperFORTH for 68000 CPU, with cross-assemblers for several other CPUs.

Human Engineered Software, 71 Park Lane, Brisbane, CA 94005

VICFORTH for the VIC-20.

Inner Access Corp., Box 888, Belmont, CA 94002

FORTH for PDP-11, CP/M, Cromemco, and Z8002; also offers FORTH workshops, consulting services.

Laboratory Microsystems, 4147 Beethoven St., Los Angeles, CA 90066

FORTH for IBM-PC, 8086/8, and Z80.

MicroMotion, 12077 Wilshire Blvd., #506, Los Angeles, CA 90025

FORTH for Apple, CPM, Cromemco CDOS, H-89, KayPro, North Star (CPM and DOS), Osborne, TRS-80 Mod. II, Vector Graphics, Xerox 820, Z-89.

Microsystems, Inc., 2500 E. Foothill Blvd., Pasadena, CA 91107

proFORTH for 8080/8085, Z80 CPUs.

Miller Microcomputer Services, 61 Lake Shore Road, Natick, MA 01760

Supplies MMSFORTH for TRS-80 and IBM-PC, with FORTH application programs for utilities, word processing, data-base management, games, and computer communications; custom programming services.

Nandgate Laboratories, P. O. Box 270426, Tampa, FL 33688
OmniFORTH for North Star, TRS-80/III.

Nautilus Systems, P. O. Box 1098, Santa Cruz, CA 95061
FORTH for TRS-80.

Peopleware Systems, Inc., 5190 W. 76th St., Minneapolis, MN 55435
P-FORTH on an STD processor card.

Perkel Software Systems, 1452 N. Clay, Springfield, MO 65802
MarxFORTH for Atari, TRS-80, North Star, CP/M, Polymorphic.

Quest Research, Inc., P. O. Box 2553, Huntsville, AL 35804
FORTH-32 for the IBM PC.

RTL Programming Aids, 10844 Deerwood SE, Lowell, MI 49331
RTL for 6502, 8080, 6809, Z80, 8086/8, and 68000.

Satellite Software International, 288 West Center, Orem, UT 84057
SSI*FORTH for IBM PC.

Shaw Laboratories, Ltd., 24301 Southland Drive, Suite 216, Hayward, CA 94544
Task-FORTH for CP/M, Micropolis, and North Star.

Software Works, Inc., Box 4386, Mountain View, CA 94040
FORTH for North Star disk systems.

Supersoft Associates, P. O. Box 1628, Champaign, IL 61820
Stackwork's FORTH for 8080 or Z80.

Talbot Microsystems, 1927 Curtis Avenue, Redondo Beach, CA 90278
T-FORTH and firmFORTH for 6800, 6801, 6809, 68000 CPUs.

Timin Engineering Co., 6044 Erlanger St., San Diego, CA 92122
FORTH development systems.

Transportable Software, Inc., PO Box 1049, Heightstown, NJ 08520
FORTH for PDP-11, IBM PC, TRS-80.

Unified Software Systems, P. O. Box 2644, New Carrollton, MD 20784
UNIFORTH for Z80, 8086/8, LSI/PDP-11, 68000, and 16032.

Ward Systems Group, 8013 Meadowview Dr., Frederick, MD 21701
FORTH for IBM 370, 4341, 3033.

28. BIBLIOGRAPHY

The following books and articles are recommended for those interested in further reading.

Starting FORTH, by Leo Brodie: Prentice-Hall, Inc. . Englewood Cliffs, NJ, 1981.
An excellent self-teacher for FORTH users.

FORTH Encyclopedia, by Mitch Derick and Linda Baker: Mountain View Press, CA, 1982.
A valuable reference on the exact functioning of most FORTH words.

Threaded Interpretive Languages, by R.G. Loeliger: Byte Books, Petersborough, NH, 1981.
An excellent treatment of the internal operation of FORTH-like languages.

"An Architectural Trail to Threaded-Code Systems", by Peter M. Kogge: *IEEE COMPUTER Magazine*, March, 1982.
A theoretical discussion of basic concepts.

BYTE Magazine, August, 1980 issue. Vol 5, No. 8.
An issue devoted to FORTH, with several tutorial articles.

29. GLOSSARY

ACM Association for Computing Machinery, a professional society of computer scientists, programmers, and data-processing specialists.

acronym A word from the first or first few letters of several words. Many computer terms are acronyms.

address In computers, the numerical designation of one of the pigeonholes or units of memory into which data may be stored or from which data may be read. Also used to designate the different input and output ports through which the computer communicates.

ALGOL ALGOrithmic Language, one of the first and most important developments in higher-level programming languages. Used mainly for describing algorithms and as the notation of choice for presenting computer-programming concepts in print.

algorithm A precise sequence of steps that define a specific computation. Also, it is a general method of solution to a computable problem.

alphanumeric Any human-readable character representable in computer memory. (*See* Character set.)

APL A Programming Language, a powerful, array operations-oriented higher-level programming language.

array A set of related items, usually variables, that are grouped together under a single name.

ASCII American Standard Code for Information Interchange, a standard way in which character sets are encoded on (mostly non-IBM) computers. (*See also* EBCDIC.)

assembler A low-level symbolic programming language that uses mnemonics instead of the numeric instructions the computer's own machine code uses.

assignment statement A statement, found in all programming language, which involves the transfer of a value to a variable.

base The number of digits used in a number system. The decimal number system uses base 10. The binary system uses base 2.

BASIC The simplest and easiest to learn of the higher-level programming languages. It's an acronym for Beginner's All-purpose Symbolic Instruction Code.

baud A unit of measurement of the transmission speed of serial data communications, about equal to one bit per second.

benchmark A program or programs processed on several computers to provide some means of comparison between various equipment or operating systems.

binary Referring to a number system based on only two digits, 0 and 1. Counting in binary would proceed: 0, 1, 10, 11, 100, 101, 110, 111, 1000, etc.

bit binary digit, a 0 or a 1.

blank The visible character that represents a single vertical space in any string.

block The basic storage unit on disk or tape memory in FORTH. Usually 1024 bytes (characters) in length.

BNF Backus Naur Form, a "metalanguage," that is, a language or notation used to define and describe programming languages. Named after John Backus (the principal developer of FORTRAN) and Peter Naur (the editor of the ALGOL-60 Reports).

boolean A value that may be either *true* or *false*. Also referred to as *logical*. Named in honor of the English mathematician George Boole.

bug An error that prevents a program from running, or from running correctly. To rid a program of such errors, programmers perform a ritual called "debugging."

buffer A temporary memory storage area set aside to hold data for transmission or processing. It may be inside or outside the computer, as for example, a box inserted between computer and printer to allow the

computer to process printed output faster than the printer can accept it.

byte A sequence of bits, usually 8 bits. Two bytes from a "half-word" in computer memory; and 4 bytes, a full (32-bit) word. Word size, however, depends on computer design.

C An algorithmic language developed at the Bell Laboratories.

card A computer data-storage medium in which data are represented by rectangular punched holes usually in 80 vertical columns.

card reader A device that inputs information stored on computer cards.

character set A symbol representable in computer memory. It may be alphabetic (like a letter), numeric (like a digit), or special alphanumeric (like a punctuation sign or any other special symbol).

chip Slang for an integrated circuit, derived from the chip of silicon from which many integrated circuits are fabricated.

COBOL COmmon Business-Oriented Language, the most widely used programming language for commercial applications.

code A specific way for representing information and for manipulating symbols.

coding Writing a program in code, that is, in a specific programming language, and, usually, on special sheets called "coding forms."

collating sequence The order in which a list of items is sorted, such as ascending numerical order (for numbers), or alphabetical order (for names).

compiler A special program that translates the complete user's program (written in a higher-level programming language) into machine language. The user's program is referred to as "source code," which the compiler converts into "object code," strings of 0s and 1s (binary instructions) that can be executed directly by the CPU.

computer A system consisting of (at least) a CPU, memory, I/O units, and a power supply.

constant A data value that usually does not change during program execution, such as *pi*.

control character A non-printing part of a character set, used to control the computer or its equipment rather than to communicate with a human user.

core Magnetic memory made of ferrous doughnut-shaped rings strung on wires. As electric current goes through the wires, the rings are magnetized in one direction or the other, the two magnetic states

being represented by 0 and 1. Core memory is also referred to as principal, main, or primary memory (or store), and it is not volatile.

CPU Central processing unit, the portion of a computer which does the arithmetic work, which twenty years ago filled one or more electronic equipment racks. Now, the CPU of small computers is a single integrated circuit perhaps one by three inches.

CRT Cathode Ray Tube, the video display (television-like) part of a computer terminal.

cursor An often-blinking underscore marking the position at which the next character is to be entered on a CRT screen.

data Coded information.

data base The complete set of data that can be used to make decisions, calculations, and tabulations. Its importance to the organization that collected it is measured by how much the organization's operations depend on frequent access to a continually updated data base.

data processing The activity of using computers and other devices to deal with the acquisition, storage, and manipulation of information.

debugging The process of eliminating errors that prevent a program from running, or prevent it from running correctly.

digits Symbols used to represent numbers in a particular system. In the binary system, there are two digits, called bits: 0 and 1. In the decimal system, there are ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

disk An auxiliary (secondary) memory device that resembles a long-playing hi-fi record.

diskette A thin magnetically coated circular object that is almost always kept in a square envelope or sleeve, and used as a low-cost mass storage medium for smaller computer systems, especially microcomputers. It is also called *floppy disk.*

DPMA Data Processing Management Association, a professional organization of data-processing specialists, programmers, and computer scientists.

EBCDIC Exended Binary Coded Decimal Interchange Code, a code used on IBM and other computers. (*See* ASCII.)

EDP Electronic Data Processing.

EPROM Erasable Programmable Read-Only Memory. Also referred to as EROM.

execute To run a program: to make the computer accept the program and carry out the program's instructions.

FIG FORTH Interest Group, an association of users and suppliers of FORTH systems.

file A collection of records.

firmware Software stored in a fixed way, on a ROM, PROM, or EPROM. Firmware is software "firmed" in hardware.

field A set of spaces, within a record, in which each space is dedicated to a specific purpose. For example, an employee's record may contain name and address fields.

floating-point A type of numeric operations using a decimal point which can appear at any position in a number, as opposed to fixed-point, where the decimal point remains at a certain position (such as in dollars and cents calculations, which always use two decimal digits).

floppy disk Diskette.

flowchart A diagram or schematic drawing of the steps to be executed in a program. The flowchart uses standard symbols, such as arrows (to indicate flow of control), connectors, rectangles, and diamonds, to make the purpose of and relationship among the steps more visually understandable.

FORTRAN FORmula TRANslation, the first and most widely used higher-level programming language for engineering and scientific applications.

FST FORTH Standards Team, a voluntary industry group working to standardize the various FORTH implementations.

GIGO Garbage In, Garbage Out (when the input is bad, the output will likely be bad also).

global A descriptive term applied to data reachable from any point in a program, as opposed to *local.*

GPSS General Problem Systems Simulation, a higher-level programming language for developing simulation systems.

graphics system Hardware and software that enable the computer to deal with drawings and pictures.

hardcopy Printout produced on paper, or on any tangible, permanent medium.

hardware The physical components and parts of a computer system (the chips, CRTs, wires, nuts, bolts, keyboards, panels, lights, tapes, disks, etc.)

hex or hexadecimal A number system based on sixteen digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

high-level language A language that is used to program computers which incorporates English-like statements and mathematical notation. APL, BASIC, COBOL, FORTRAN, LISP, PASCAL, PL/I, and many others are examples of high-level languages.

IC Integrated Circuit.

ID IDentification.

identifier The name of an object in a program, such as the name of a variable or a subroutine.

increment Amount by which a given quantity is increased.

indenting Setting-off portions of a program or of the output to improve readability and visual appeal.

initialization The process of assigning initial values to variables.

input Data entered into the computer through a peripheral device, such as a card reader or a terminal.

instruction A single order within a program. It may be a statement, a command, a subprogram call, or some other order that requires the program to perform a particular task.

interactive Conversational, the case in which the computer responds when it is prompted with a proper request.

interface The (shared) connection or relationship (or boundary) between two components of a system, such as the hardware–software interface. Also, the point at which two components are joined together.

interpreter A special program that translates each one of the statements of the user's program (written in a high-level programming language) into machine language as the program is entered, executing each statement immediately.

interrupt A special computer signal which stops processing of the current program to force execution of a higher-priority program segment.

I/O Input/Output.

ISAM Indexed Sequential Access Method, one of several ways in which data may be organized into files. (*See also* SAM.)

iteration Looping, repetition.

JCL Job Control Language, a language used to make the computer perform systems tasks. An example of a system task is to load a program into main memory.

joystick A stick or bar located on or near a terminal keyboard. Moving the stick moves a dot on the terminal's screen.

jump Branch, detour, skip, or transfer. Change the order of execution of a sequence of statements, due to a condition requiring that change. Normally, statements are executed in sequential order, one after another; but a certain condition may require a change in this order of execution, and a jump will result.

K Short for the Greek word *kilo*, meaning 1000. In binary computers, since $2^{10} = 1024$, the term K is used to measure memory in 1024-byte chunks. For example, 16K = 16,384 bytes.

keyboard The portion of a terminal that contains the keys. Terminal keyboards are very much like electric typewriter keyboards. Keyboarding means entering data using a keyboard. The action of pressing a single key is a "keystroke."

keypunch A keyboard-operated device that punches holes in computer cards.

keyword A name, usually appearing in a programming language, that has been expressly reserved to have some unique purpose. Also referred to as "reserved word."

language Communication in a distinctly human manner, with symbols used in standard ways to represent standard meanings.

LED Light-Emitting Diode, common technology used to display characters by glowing light (usually red or green) in the readout area of electronic calculators.

line printer An output device that prints an entire line (80 to 144 characters) at a time.

LISP LISt Processor, a powerful high-level programming language commonly used in artificial intelligence and word-processing applications.

list A data structure: a sequence of data items.

local A descriptive word applied to data reachable only within a small segment of a program.

logging on or in The act of starting a session at a computer terminal, or of alerting the computer to the beginning of an oncoming job.

loop Repetition (of a list of statements located between the beginning and the end of the loop).

LSI Large Scale Integration. Technology by which many thousands of electronic components are built on a single silicon chip.

machine language Low-level programming language, specific to each computer, and consisting of strings of 0s and 1s. Also called binary code.

mainframe The computer body that houses the central processor, or the central processor itself. *See* CPU.

Memory The part of a computer system whose function is to store data or instructions. Memory may be primary (main, core, or principal) or secondary (auxiliary or peripheral).

MICR Magnetic Ink Character Recognition, a technique associated with the printing and reading of characters using a special magnetic ink. (See the bottom of your bank checks for an example.)

microcomputer A small computer built around a microprocessor.

microprocessor A central processor usually implemented on a single LSI IC chip.

minicomputer A small computer, larger than a microcomputer but smaller than the regular mainframe computer.

MIS Management Information Systems.

mnemonic Memory-aiding. An abbreviated name that helps remember the actual longer name. For example, HRDWRE may be a mnemonic for hardware.

mode The form of operation, such as execution mode.

modem MOdulator-DEModulator. A data set—a device that connects data-processing equipment to a communications channel.

module An independent and usually self-contained portion of the whole, such as a program segment that can be executed apart from the entire program itself.

multiplex To simultaneously transmit two or more messages or instructions through a single channel of communication.

multi-tasking A method of scheduling separate programs to appear that the computer, which can run only one program at a time, is processing several tasks simultaneously. The computer can process hundreds of thousands of instructions in each second, so may often be waiting for slower peripherals (printer, human operator, disk systems, etc.). This waiting time is used to process a different program segment.

network A set of devices interconnected with each other through communications channels.

OCR Optical Character Recognition: characters printed in a special type font readable by both people and computers.

octal Base-8 arithmetic notation that uses the eight octal digits (0, 1, 2, 3, 4, 5, 6, and 7).

off-line (1) Computer resources not directly and promptly accessible to a computer during the execution of a program. (2) A type of data processing done at a different time than when the data is collected.

on-line Method of processing while being in direct communication with or under the direct control of the CPU.

operating system OS, software that controls the execution of computer programs. Examples are: Disk Operating Systems (DOS), Time-Sharing Operating Systems (TSOS).

operation Any action defined by a single computer instruction of higher-level programming language statement. Examples of arithmetic operations are addition, subtraction, multiplication, and division.

output Data transmitted by the computer to a peripheral device, such as a line printer or a terminal.

overflow Condition encountered when specified limits are exceeded. Usually applies to memory capacity.

paper printer Peripheral device that outputs on paper.

PASCAL A higher-level programming language devised for teaching computer science and good programming habits. It is a simple language yet rich in data-typing and data-structuring facilities.

password Special word used to restrict access, for security reasons, to a computer system.

PC Abbreviation for either printed circuit or personal computer.

peripheral Any device or component that can be connected to the computer.

PL/I A higher-level programming language (Programming Language/One) that combines both scientific and business facilities.

PL/M Programming Language for Microcomputers, a high-level language derived from PL/I, and developed by INTEL for its microcomputers.

plotter A peripheral device used to draw line drawings, diagrams, and pictures.

pop To remove a data value from a stack.

POS Point Of Sale terminal, a cash-register-like device interfaced with a computer that keeps track of all transactions.

program A sequence of instructions, written in a programming language, and according to the requirements of a specific computer system, which it will

direct to carry out the necessary tasks for solving a problem.

programming The process of writing a program in a particular programming language. Also, the process of reducing the solution of a problem to a program.

PROM Programmable Read Only Memory.

punch cards Cardboard cards, usually of 80 vertical columns each, in which characters are entered by way of combinations of punched holes.

push To add a data value to a stack.

query Request, question, inquiry (for information). Query languages have been developed to deal with data bases.

queue A data structure: a list of data items, in which list deletions are made at the head (front) and additions at the tail (back) of the queue. A queue is a First-In-First-Out (FIFO) or Last-In-Last-Out (LILO) list.

RAM Random Access Memory (Read/Write Direct Access LSI Memory).

random-access memory A type of memory wherein each portion is accessible by the computer in the same amount of time, as opposed to, for example, rotating memory, where there may be a wait until some data becomes available again.

real time A method of processing in which the computer responds sufficiently fast to affect the activity under the computer's control, as opposed to off-line processing. Missile guidance is an example of a real-time computer application.

record A set or block of data items, structured into fields. A component of a file.

recursion A programming technique in which a process is defined in terms of itself.

re-entrant A type of program which can be used by several people more or less simultaneously.

relocatable A computer program which can be located anywhere in memory and still run correctly. Many programs must be loaded at a single specific location and so are not relocatable.

remote access Peripheral equipment connected to the central processor but physically distant from it.

response time The time it takes a system to react to a prompt.

robot A device equipped with an on-board computer and necessary peripheral equipment, and capable of carrying out specified tasks and of making the needed

independent decisions to accomplish these tasks, but without human intervention.

ROM Read Only Memory.

RPG Report Program Generator, a special programming language in which specification sheets are filled to produce business reports.

run To execute a program.

run time The time it takes a program to complete the tasks it is supposed to do.

SAM Sequential Access Method. (*See* ISAM).

screen The video (CRT) surface on which information may be displayed.

scrolling The vertical (up and down) motion of information on a screen.

SIMULA SIMSCRIPT Higher-level programming languages used in simulation applications.

SNOBOL StriNg Oriented symBOlic Language, a higher-level programming language used in string-processing applications, such as work with voluminous texts.

software The programs used by a computer system.

stack A temporary data storage method which allows data values to be stored so that the last one in is the first one retrieved, like a stack of cafeteria trays.

statement The fundamental unit of a computer program written in a higher-level programming language. A statement usually consists of a single instruction, such as an assignment, input, or output.

storage, store Memory.

string A sequence of data in which each unit is a printable character (see alphanumeric).

subprogram A portion of a program.

subroutine A group of statements that may be treated as a unit and "called" when needed by the main program.

syntax The usually rigid patterns of words and phrases in computer languages, which the computer may interpret as proper commands.

system The conglomerate of devices, people, data, methods, practices, and whatever else may be needed to accomplish certain objectives.

systems analysis The study of systems, specifically business-oriented computer systems.

table The arrangement of data into rows and columns.

tabulation The act of arranging data to form a table.

tape Either paper tape or magnetic tape: a storage medium for data or instructions.

terminal A peripheral device through which a person may communicate with a computer, and which can be used to send and receive information.

throughput The usable work done by a computer.

timesharing A method of processing in which the computer is shared (or, due to its fast response time, appears to be shared) by several users simultaneously.

transaction Any business activity, such as a sale, expenditure, purchase, shipment, reservation, or inquiry.

truth value The value of a boolean, which can be only "true" or "false", as for example the result of a comparison: A greater than B?

turnaround time The actual time between submitting a job (program and data) to the computer and receiving the output.

UPC Universal Product Code, also called 10-digit bar code, used on most products, and designed to speed processing of sales at check-out counters.

utility Software prepared by a computer manufacturer to enable some operation efficiencies in routine tasks.

value A constant, or the quantity being stored in a variable.

variable An object that has a name (a location in memory) and that holds or stores a particular value at a particular time.

VDT Video Display Terminal, a television-like device with a keyboard.

vendor A company that sells or supplies hardware, firmware, or software.

virtual memory A method of making exterior, non-random-access memory appear to be on-line, random-access memory.

VLSI Very Large Scale Integrated circuit.

volatile memory Memory that lasts only a brief time, and which does not provide protection from destruction or loss to any data stored.

wafer A slice of silicon, with integrated circuits built on it.

word A group of bits placed together and treated as a unit in a single location in memory. A word usually consists of at least 4 bytes (32 bits).

WP Word Processing, a computer system specifically designed for typing and text-editing applications, such as writing, and formatting letters, reports, and books.

APPENDIX: FORTH WORD SET

In reviewing a computer language, it is helpful to look at the command set of words provided with the system. This list contains words which are defined in most FORTH systems to do the tasks described. Although not complete, this list is representative of typical programming words.

FORTH Word	Action
ABS	Changes the top data value into an absolute (unsigned) value.
ALLOT	Allocates a number of bytes in the dictionary, for a list or table, for example.
AND	A logical (not arithmetic) combination of two data values.
BEGIN	Starts a loop of an unpredictable number of times through.
BLOCK	Brings data stored on disk or tape into the computer's memory.
C!	Stores a byte value into the address on top of the stack.
C@	Fetches a byte value from the address on top of the stack.
CONSTANT	Defines a name for a value which doesn't change.
CONTEXT	Allows selecting among several vocabularies within the dictionary.
D+	Adds two double-length numbers on the stack.
D-	Subtracts two double-length numbers on the stack.
DECIMAL	Sets to 10 the base for displaying numbers.
DEPTH	Calculates how many data values are on the stack.
DO	Starts a loop of a known number of times through.
DOES>	In a defining word, marks the run-time behavior that class of words will have.

DROP	Discards the data value on top of the stack.
DUP	Makes another copy of the top value on the data stack.
ELSE	Marks a program path to execute if a decision value is "false."
EMIT	Sends a character to the operator's display.
EMPTY-BUFFERS	Erases the storage area reserved for disk or tape blocks.
EXPECT	Obtains a number of characters from the operator's keyboard.
FLUSH	Writes data in memory out to the disk or tape storage.
FORGET	Removes some user-defined words from the dictionary.
HEX	Sets to 16 the base for dispaying numbers.
I	Defined two ways: when editing, allows inserting text; when running a program, copies data value from top of return stack to top of data stack.
IF	Marks the start of a branch or decision choice.
KEY	Obtains one character from keyboard.
LEAVE	Marks a midpoint exit for a DO-loop.
LIST	Displays the text contents of a storage block.
LOAD	Compiles into the dictionary a program stored in text form on disk or tape.
LOOP	Marks the end of a DO-loop.
MAX	Leaves on the stack the higher of the two top values.
MIN	Leaves on the stack the lesser of the two top values.
MOD	Divides two numbers, leaving only the remainder.
NOT	Reverses a truth value from "true" to "false" and vice versa.

OCTAL	Sets to 8 the base for displaying numbers.
OR	A logical combination of two data values.
OVER	Duplicates the second stack value on top of the stack.
REPEAT	An alternate ending for a BEGIN-loop, used when WHILE provides a mid-loop exit: BEGIN. . . WHILE. . . REPEAT.
ROT	Rotates the stack's third data value to the top.
SPACE	Inserts a space character formatted output.
SWAP	Interchanges the top two values on the data stack.
THEN	Marks the end of an IF. . . ELSE. . . THEN decision path.
U*	An unsigned multiplication of the top two stack values.
U.	Displays the top stack value as an unsigned number.
U/MOD	An unsigned division of the top tow stack entries, leaving quotient and remainder.
UNTIL	Marks the end of a BEGIN loop. BEGIN. . . UNTIL.
UPDATE	Marks a block of data in memory as having been changed.
VARIABLE	Defines a name for a value which changes.
VOCABULARY	Allows delineating separate vocabularies within the dictionary.
WHILE	Marks a mid-loop exit in a BEGIN/REPEAT loop.
>R	Moves a data value from the data stack to the return stack.
R>	Moves a data value from the return stack to the data stack.
?DUP	Duplicates the value on top of the stack, but only if it is not zero.

:	Creates a dictionary entry for the word following the colon.
;	Ends the dictionary entry for that word.
*	Multiplies two numbers on top of the stack.
/	Divides two numbers on top of the stack, leaving only the quotient.
/MOD	Divides two numbers, leaving both quotient and remainder.
*/	Multiplies two numbers together, divides result by top-of-stack value.
+	Adds two numbers on top of the stack.
-	Subtracts two numbers on top of the stack.
!	Stores a data value into an address.
+!	Adds the number on the stack to the contents of an address.
@	Fetches a data value from an address.
'	Searches the dictionary to locate a word.
=	Compares two values on the stack for equality.
0=	Tests if top stack value is zero.
<	Tests if second stack value is less than top stack value.
0<	Tests if top stack value is greater than zero.
>	Tests if second stack value is greater than top stack value.
0>	Tests if top stack value is greater than zero.
1+	Increases value on top of stack by one.
1-	Decrease value on top of stack by one.
2+	Adds two to value on top of stack.

2-	Subracts two from value on top of stack.
2/	Divides stack`s top value by two.
2DUP	Duplicates a double-length number on the stack.
2SWAP	Interchanges two double-length numbers on the stack.
2DROP	Erases the double-length number on top of the stack.
2OVER	Copies the second double-length stack entry onto the top of the stack.

GET ON THE ALFRED COMPUTER MAILING LIST!

KEEP UP-TO-DATE!

Send us your complete **name** and **address,** and we'll send you catalogs, newsletters, and new product listings, as they become available.

Or fill out and mail this coupon:

Name

Address

City State Zip

Handy Guide Titles You Own

Comments:

Send to: ALFRED PUBLISHING CO., INC.
Post Office Box 5964
Sherman Oaks, California 91413

Alfred Handy Guides

Practical, economical, and concise
Alfred Handy Guides tell you what you need to know quickly and easily—without a lot of reading!

Perfect for today's fast-moving adult on the run, they fit anywhere—in pocket, purse, gadget bag, guitar case. Always where you need them!

The Alfred Handy Guide Series to Computers

The Computer Companion
How to Buy a Personal Computer
How to Buy a Word Processor
How to Use Atari Computers
How to Use the IBM PC
How to Use VisiCalc/SuperCalc
Understanding APL
Understanding Artificial Intelligence
Understanding Atari Graphics
Understanding BASIC
Understanding COBOL
Understanding Data Base Management
Understanding Data Communications
Understanding FORTH
Understanding FORTRAN
Understanding LISP
Understanding LOGO
Understanding Pascal

Other Alfred Handy Guide Series:

Cooking
Health
Music
Photography

Look for new titles and new series.
For more information:

Alfred Publishing Co., Inc.
P.O. Box 5964
15335 Morrison St.
Sherman Oaks, CA 91413

ISBN 0-88284-237-4