

FORTH LANGUAGE
for the Commodore 64
with Graphics, Sound and Assembler

A DATA BECKER PRODUCT

520994

Published by:

The One Good Use
Abacus  **Software**

Abacus Software

(C) 1985 DATA BECKER

FORTH

ON: **520994**

HPD: FEBRUARY 13, 1986

Abacus Software • P.O. Box 7211 • Grand Rapids, MI 49510 • (616) 241-5510

COPYRIGHT NOTICE

Abacus Software makes this package available for use on a single computer only. It is unlawful to copy any portion of this software package onto any medium for any purpose other than backup. It is unlawful to give away or resell copies of any part of this package. Any unauthorized distribution of this product deprives the authors of their deserved royalties. For use on multiple computers, please contact Abacus Software to make such arrangements.

WARRANTY

Abacus Software makes no warranties, expressed or implied as to the fitness of this software product for any particular purpose. In no event will Abacus Software be liable for consequential damages. Abacus Software will replace any copy of the software which is unreadable if returned within 90 days of purchase. Thereafter there will be a nominal charge for replacement.

First Printing, March 1985

Printed in U.S.A. Translated by G. Dykema

Copyright[©]1985 Abacus Software, Inc.

P.O. Box 7211

Grand Rapids, MI 49510

Copyright[©]1985 DATA BECKER, GmbH

Merowingerstr. 30

4000 Dusseldorf, W.Germany

ISBN# 0-916439-32-1

Table of Contents

1. Introduction.....	1
2. Forth-64.....	3
2.1 General.....	3
2.2 First time using Forth-64.....	4
2.3 Input from the terminal.....	6
2.4 The screen file SCR.....	7
3. Forth editor.....	9
3.1 General.....	9
3.2 Editing illustrated with a simple example.....	10
3.3 Overview of the editing buffer.....	12
3.4 Description of editor commands.....	13
3.4.1 Screen commands.....	13
3.4.2 Line commands.....	14
3.4.3 Character commands.....	15
3.4.4 Ending the editing.....	17
3.5 Editor function keys.....	18
3.6 Notes.....	18
4. Programming in Forth.....	19
4.1 General.....	19
4.2 The stack.....	20
4.3 Examples.....	21
5. Music with the sound vocabulary.....	25
5.1 Descriptions of sound commands.....	25
5.2 A first composition with sound.....	27
6. Graphics with the graphics vocabulary.....	29
6.1 Lo-res (with example).....	29
6.2 Hi-res (with example).....	30
6.3 Descriptions of graphics commands.....	35
7. The Forth assembler.....	37
7.1 General.....	37
7.2 The assembler vocabulary.....	39
7.3 A small assembler program.....	41
8. Miscellaneous.....	43
Appendix 1: Memory map.....	44
Appendix 2: The entire Forth vocabulary.....	45



1. Introduction

"Forth should be able to do what other programming languages can do--only more elegantly."

This was the claim that Charles H. Moore made about the language which he developed, Forth. As a matter of fact, the steadily growing base of Forth enthusiasts seems never to tire of producing examples illustrating how programs can be realized faster and in less space (and often more elegantly) than in other languages. For example, compilers are written in Forth and also the famous turtle graphics in Logo have been implemented in Forth--quickly and elegantly. Forth is particularly well applied to the solution of real-time problems since Forth is fast, compact, and efficient.

But Forth is not only a programming language but also an operating system. With Forth, a complete system is available to the user in which one can program, assemble, and edit, with which one (in the case of Forth-64) can do graphics and make music, with which memory dumps and program patches can be made, and in which desired extensions can be made by the user which are then immediately resident in the system. Forth consists of hundreds of words which all denote a program of their own which is immediately executed when called.

To be sure, Forth is not a language which is very easy to understand since it is quite different from all other high-level languages. Normally, Forth uses only the stack as an interface for parameter passing and performs no special parameter checking. In addition, Forth uses what is called Reverse Polish Notation or RPN for arithmetic. RPN can take a while to get used to.

{This page left blank intentionally}

2. Forth-64

2.1 General

Forth-64 is based on the FIG-FORTH standard of the Forth Interest Group (FIG) of San Carlos, California. Naturally, current versions of Forth offer many more words, just as this version offers some three times as many words in its basic vocabulary as does the standard. In addition, all of vocabulary of the second Forth standard, Forth 79, has been included in this Forth. Furthermore, some words in the most recent generation of Forth, Forth 83, have been included in Forth-64. Unfortunately, some words in Forth 83 have different programs connected with them in contrast to earlier Forths and some words have been renamed. Where such ambiguities exists (for example, the words PICK and ROLL have a slightly altered function, and the old word 0= has been replaced with NOT in Forth 83), the old syntax and semantics have been retained in Forth-64.

Independent of the standard, Forth-64 contains a number of very useful words with which the SID (Sound Interface Device) and VIC (Video Interface Controller) can be addressed making it easier to use these devices for graphics and sound.

Forth-64 naturally includes the special Forth assembler with which one can create assembly language programs (in the Forth notation), and Forth has an easy-to-use editor.

We also haven't forgotten the words with which one can address external devices (disk) and process strings (similar to BASIC), and the many programming aids such as the DUMP, HELP, and TRACE commands.

Should the user still require additional words for his special needs, these can be defined at any time and are--as any other Forth word--immediately available for use.

At this point we will only hint at the ability to assign strings (Forth words) to function keys or any other key.

Forth-64 makes use of the function keys itself and has assigned often-used functions to the function keys in the editor and even standard modes. The HELP command displays the current key assignments, also when the assignments have been changed by the user.

One last feature of Forth-64 is the fact that all of the examples given in the user's manual are also included on the disk. The programs are useful and can be immediately loaded, tried out, and changed as desired.

2.2 First time using Forth-64

The Forth distribution diskette contains 3 files:

- GFORTH (The loader program)
- FIG-FORTH64 (The Forth program)
- SCR (an included screen file)

Forth is loaded with: LOAD "GFORTH",8,1 . *NOTE: SEE COVER PAGE*

Forth-64 starts itself, meaning you don't have to type RUN or some similar command after the program has loaded.

Forth-64 announces itself with the usual initial message containing the version, date of creation, and program name.

The first commands which you can now give are such things as VLIST, HELP, or .S .

VLIST generates a list of all the words in the current context and Forth vocabularies. The context dictionary may be ASSEMBLER, EDITOR, GRAPHIC, SOUND, or FORTH itself. After loading, the context vocabulary is Forth.

If the command listing produced by VLIST takes too long for you, you can interrupt it by pressing the STOP key. This is true of the most of the commands which take a while to execute.

The command .S returns the condition of the stack. This is hopefully empty immediately after loading Forth.

When you enter a number (such as 3 CR), and then take a look at the stack, you will see this number there until another Forth word such as CLEAR CR or . CR retrieves it.

Enter the command HELP. It shows you the current key assignments. This produces an assignment of ASCII codes to Forth words. Any word can be assigned to any key as desired with the limitation that the assigned word not require any values from the stack.

Perhaps you would like to make a small change in your system. Let us assume that you do not like the color combination on the screen.

Simply enter the following sentence:

```
BLUE BORDER  CYAN SCREEN  BROWN PEN CR
```

and everything looks much nicer, right?

Forget BASIC, forget POKEing values into addresses that no human can remember!

In Forth-64, the color codes 0 to 9 are assigned to constants--words. Write BLACK when you mean BLACK instead of 0!

You can even define a new word for your favorite color combination,

: FCN BLACK BORDER RED SCREEN WHITE PEN ; CR

2.3 Input from the terminal

In Forth it was planned from the beginning to have the input operate in the KEY mode, meaning that each character is accepted and processed as soon as it is typed. This burdens Forth with a wholly unnecessary task which the operating system performs all by itself when the appropriate routine is called. Forth-64 does not process input in the key mode. The input of characters is done in the same way it is done in BASIC, meaning that Forth does not "see" the line until you send it an edited line with CR.

A small modification to this input function has been made however (with the result of greater convenience for the user):

The first character the user enters after Forth responds with "OK" is checked to see if it is contained in an assignment table which creates a connection between input characters and Forth functions. If this is the case, then this function is performed and more input will be expected upon completion.

This continues until a key is pressed which is not contained in this table. In this case the normal line input routine is called and the character first entered is, of course, retained.

An example of a key assignment can be found in Section 5 - "Music with the sound vocabulary".

2.4 The screen file SCR

Sooner or later it will no longer be any fun to enter your larger programs by hand each time. You want to be able to write these to a file with the help of an editor in order to be able to load them again when required. Forth recognizes only a single file. This is called SCR in Forth-64. SCR is a relative file which can be of any desired size, preferably on its own disk. Forth manages this file virtually, meaning that you can access random places in it, page forwards and backwards, and use them as you would memory in the computer, without giving it any thought. Forth takes care of the input and output transfer by itself. You need only inform the computer when you are done with an editing session so that it can make sure that all of the memory has been saved on disk. The word DONE performs this operation.

The screen file is divided into individual logical pages (screens). A page contains 16 lines, each of 64 characters. A page is also the logical entity used in editing. As we already said, one can select any page at any time, look at it and edit it. Two screens are always reserved for error messages in Forth. These are screens 4 and 5. When creating a new screen file, do not forget to copy these pages to the new file. If you don't do this you will not get any error messages when you encounter an error.

Before you edit your own programs, get one of your own data disks for the sake of security. This will also enable you to learn a couple of new Forth words:

- insert original disk
- EMPTY-BUFFERS 4 LIST 5 LIST CLOSE-SCREEN CR
- insert new disk in drive
- " N:disk name,id" DOS CR
Note: The spaces behind the " are relevant.
This command formats the disk, erasing it!
- CREATE-SCREEN CR
A screen file with 2200 sentences is created and screens 4 and 5, which are in the disk buffer, are copied to the disk. It requires several minutes to create the file.

Now you have a Forth data disk which you can use right away. You can of course make as many data disks as you like in this manner.

{This page left blank intentionally}

3. Forth Editor

3.1 General

One of the peculiarities of Forth is that the user cannot divide his text and programs into files as desired but must write these to a single virtual file called "SCR" in Forth-64. This virtual Forth storage is further divided into screens (logical pages). The user can select and work with desired pages of this file.

Just as screens can be selected "at random" for editing, these same screens can be compiled in any order, determined either through references at the end of each page or through a command page.

The processing entity is always such a screen!

The format of a Forth screen is standardized. It always consists of 16 lines each of 64 characters. The maximum number of screens depends on the amount of storage space available on the disk.

An additional standard is that screens 4 and 5 must contain error messages. These screens should not be overwritten with programs or else no meaningful messages will be given when an error occurs.

If you take a look at these screens (4 LIST 5 LIST CR), you will see that some lines are empty. These lines can be filled with error messages as the user sees fit.

Line 15 on page 4 has another special significance. It should contain a brief text identifying the disk, perhaps its contents and the date of creation. This line is evaluated by the documenting words TRIAD and INDEX. The user can receive its message with the command 15 MESSAGE CR.

The following example will give you a look into how editing is done in Forth and how you can change the disk identification line. You can best learn about the many possibilities not explained in the example here from the description of the EDITOR vocabulary in section 3.4- "Description of the editor commands".

3.2 Simple editing

We enter the following command to edit screen 4:

```
4 EDIT CR
```

Screen 4 with the corresponding error messages will be displayed:

```
SCR # 4
0 P ( ERROR MESSAGES )
1 P EMPTY STACK !
2 P DICTIONARY FULL !
3 P HAS INCORRECT ADDRESS MODE
4 P ISN'T UNIQUE
5 P
6 P DISC RANGE ?
7 P FULL STACK !
8 P DISC ERROR !
9 P
10 P
11 P
12 P
13 P
14 P
15 P MASTER DISKETTE FORTH64
```

The line numbers are set from 0 to 15 and are not, as in they are in BASIC, freely chosen. The P behind the line number means PUT or PLACE and has the function that the text behind it (separated by a space) will be written on the corresponding line.

You may have noticed that the usual "OK" message is not printed. This makes sense because you can now position the cursor at the spot to be changed and make the appropriate changes. By sending the P command with CR, the start of the next line is not destroyed. You can then continue to edit as usual.

If you do not want to move the cursor, you can also write your lines directly, without using the pattern. If you have screens which contain more than 7 lines with over 35 characters, the top lines will scroll away and you must edit the first lines in this manner.

At this point you should be strongly advised to alternate lines when writing Forth programs (with the exception of lines 0 [contains screen information] and 15 [here it doesn't hurt anymore]). Otherwise Forth programs become very difficult to read and Forth programs which are difficult to read become hard for even the author to understand after a while.

Once you have changed screen 4 according to your tastes, save the changes to disk with the command SSAVE (F7).

You can get to the next or previous screen with the commands +EDIT (F1) and -EDIT (F2), accordingly. When you are done editing, enter the command DONE (F8). This implicates, among others, SSAVE, and even saves all changes in the disk buffer to disk if you forget SSAVE at the end of editing a screen.

To see if your changes were accepted, take a look at the index of screens 4 and 5 with the INDEX command:

```
4 5 INDEX CR
```

The comment lines of screens 4 and 5 as well as the disk identification line will be displayed on the screen.

For additional practice, you could define your favorite screen color combination on a screen and then load this screen after you have loaded Forth from the distribution diskette. Screens can be loaded with the LOAD command. For example:

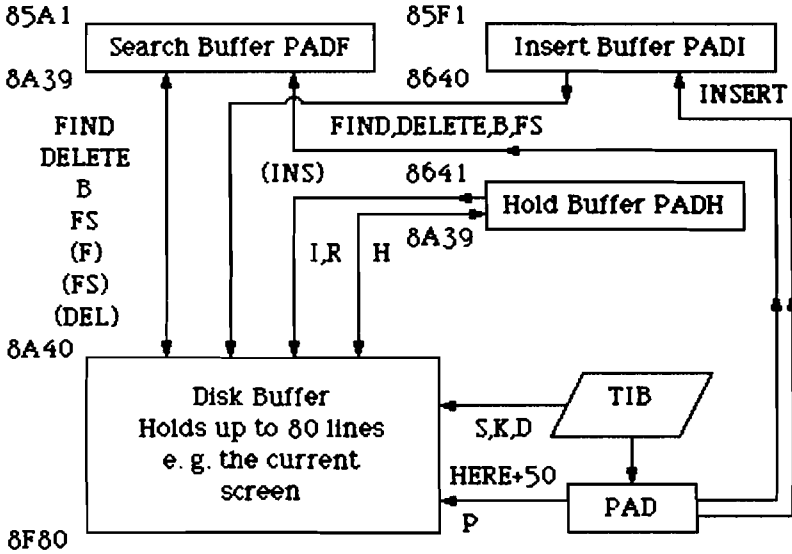
```
10 LOAD CR
```

if your program is in screen 10.

If screen 10 was the last screen to be edited or if the value 10 is contained in the variable SCR, you need only press the RUN key on the computer. This key is assigned the function ELOAD.

If Forth discovers an error while loading a screen, such as a word it does not recognize, you need only enter the word WHERE and the screen in question will be displayed for editing with a marker at the spot where the error occurred.

3.3 Overview of the editor buffer



All buffers up to PAD are used only by the editor and can be used for other purposes. They are erased with **EMPTY-BUFFERS** before using the editor.

3.4 Description of editor commands

3.4.1 SCREEN commands

The following commands operate on one or more screens.

EDIT n --
Selects the nth screen for editing. OK is turned off. Internal cursor position is set to the start of the 0th line (variable R#), the screen is displayed, and the cursor is positioned on the first line. Displayed screens can be edited.

E --
The current screen (number in SCR) is displayed again (as with EDIT).

+EDIT --
The next screen is displayed for editing.

-EDIT --
The previous screen is displayed for editing.

WIPE --
The current screen is erased (only in the disk buffer). After command E, a blank screen is displayed.

SSAVE --
The current screen is saved on disk, assuming it was changed.

SCRATCH --
The changes in the screen being edited are made invalid in the disk buffer. The last-saved condition can be redisplayed for editing with the E command.

3.4.2 Line commands

Commands which manipulate one or more entire lines.

- P i --
Places the following text in the ith line of the current screen. The internal cursor position is connected in series to the next page (important for searching).
- S i n --
Spreads the current screen at the ith line by n lines. The lines behind n are lost.
- R i --
Replaces the current line and the lines following with the contents of PAD. PAD can be filled by the commands H or XH.
- I i n --
Inserts the contents of PAD at the ith line. The last lines of the screen are lost.
- H i n --
Copies n lines at line i of the current screen to PAD.
- K i n --
Erases n lines at line i on the current screen.
- D i n --
Deletes n lines at line i on the current screen. Lines are placed in PAD.
- XH s i n --
"External Hold." Copies n lines at line i of Screen s to PAD. This allows you to use sections of other screens in the screen currently being edited without having to select the screen from which the extraction is made.

3.4.3 Character commands

The following commands operate on individual characters or on the internal cursor position R#. Search and delete commands operate at the cursor position saved in R#. The search and insert buffers are not initialized when the editor is called in order to preserve their contents over multiple editor calls. The appropriate parameters must therefore be given for the first search or insert call.

TOP --
 Sets the internal cursor position to 0 (line 0, column 0).

L i --
 Sets the internal cursor position to the start of the ith line (line i, column 0).

!R# n --
 Sets the cursor pointer to n.

+!R# n --
 Advances the cursor pointer n positions.

V --
 Writes the line to which the cursor pointer points on the screen. The cursor position is displayed through reverse output of the character in question.

X --
 Erases the remainder of the line at the cursor position.

FIND --
 A search is made for the text, enclosed in English pound signs (£), following the command, at the current cursor position. The text is placed in the search buffer. The delimiter can be found in the variable DELI and can be changed as desired. If an empty string is specified (for example: FIND ££ CR), the search buffer does not change. A search will be made for the string last given in a search command. If no delimiters (English pound signs) are found in the following string, the entire remainder of the input line will be used as the target string.

INSERT --
 The text following the command and enclosed in English pound signs is inserted at the current cursor position. The text is also placed in the insert buffer. The delimiter is again in the variable DELI and can be changed

as required. If an empty string is entered (for example: INSERT ££ CR), the contents of the insert buffer are not changed and existing contents are then used. If no delimiter is found in the text following the command, the entire remainder of the input line is inserted (Beware of length conflicts!)

B

--
The cursor is set back the number characters found in the search buffer.

DELETE

--
A search is made for the text following the command and enclosed in English pound signs, starting at the current cursor position, and deleted if found. The text is placed in the search buffer. The delimiter is found in the variable DELI and can be changed as required. The same applies for empty strings or those not enclosed in delimiters as for FIND.

FS

--
A search is made for the text following the command, enclosed in English pound signs, and if found, this text is replaced by the text enclosed in the English pound signs following the first. Both texts are placed in the search and insert buffers, respectively. The delimiter is once again found in the variable DELI and can be changed if desired. The same applies as with DELETE and INSERT for empty strings or strings not enclosed in delimiters.
Example: FS £CAT£DOG£

(FIND)

--
Like FIND, but the text to be found is taken from the search buffer. (FIND) is the actual search routine for FIND and can be used for multiple searches for the same target.

(INSERT)

--
Like INSERT, but the text in the insert buffer is inserted. What was said about (FIND) applies here too.

(FS)

--
Like FS, but the strings are taken from the search and insert buffers.

(DELETE)

--
Like DELETE, but the text is taken from the search buffer.

3.4.4 Ending the editing

DONE

--
Switches to the Forth vocabulary. The editing mode is exited. First, however, lines which were not already saved are written to disk. The OK message is reactivated and the editor function keys are deactivated.

3.5 Editor function keys

The function keys have special assignments in the Forth editor. The assignments can be viewed through the HELP command.

CHR\$(x)	Key	Assignments
94	↑ : E	redisplay current screen
131	SHIFT RUN: ELOAD	loads the last-edited screen
133	F1: +EDIT	edit next screen
137	F2: -EDIT	edit previous screen
134	F3: (DELETE)	delete current search target
138	F4: WIPE	erase current screen
135	F5: (INSERT)	inserts the current INSERT buffer
139	F6: (FIND)	searches for current target
136	F7: SSAVE	saves edited screen
140	F8: DONE	ends editing

The function FS can be repeated as often desired by using F3 and F5. The search and insert buffers are also not cleared when screens are changed so that a substitution using function keys F1, F3, F5, and when done, F7, can be performed, for example.

3.6 Notes

If you want to edit, but Forth does not accept the edit commands entered, you have probably exited the editor. Enter the command i EDIT CR or E CR.

4. Programming in Forth

This Forth user's guide is not designed to teach Forth. The user who wants to learn Forth and write more complex Forth programs should get a good tutorial book on Forth. Starting Forth by Leo Brodie of FORTH, Inc. is an excellent book to start learning this unique language.

In this chapter we will clarify only the main characteristics of Forth. We will include small example programs which, in contrast to the programs in the following chapters, are not contained on the program disk.

4.1 General

The Forth language consists of words, as does any other language. Each of these words corresponds, so to speak, to a command which has something to do with the program. All of the words belonging to the basic Forth vocabulary are described in the appendix to this guide. There are hundreds of them for the user to become acquainted with. Some (like HELP or VLIST) have been described in the previous chapters.

Most of the words must have yet to be explained. Just as you can't tell anyone "Write!" without telling him what to write, it is just as senseless to give Forth the corresponding command . CR without telling it what should be displayed on the screen.

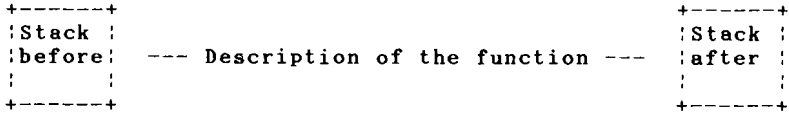
The instruction 5 . CR, on the other hand, can be processed and Forth outputs the number 5 on the screen. It should be noted that in Forth, one first gives the operand(s) and then the operator; therefore first the number 5 and then the instruction ".".

The same applies to arithmetic expressions. The sum of the two numbers 7 and 12 is calculated in Forth through the expression 7 12 + CR instead of $7 + 12 =$ as usual.

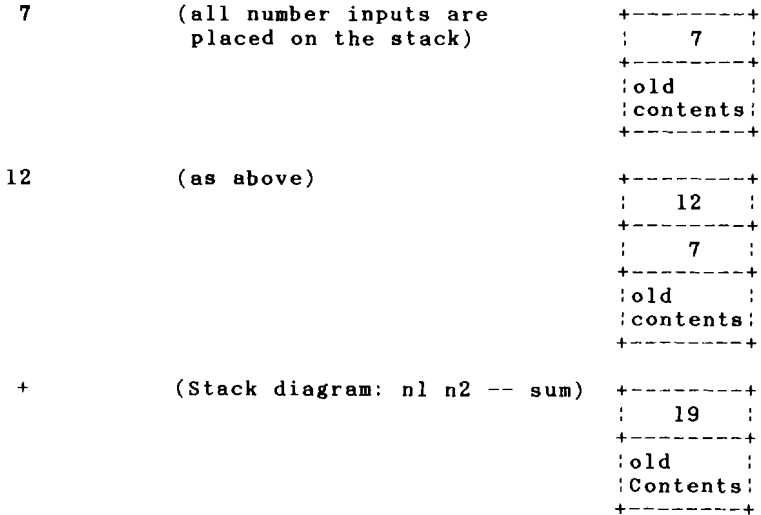
Now we must explain how Forth words know what they are supposed to process. To do so we must describe something called the stack.

4.2 The stack

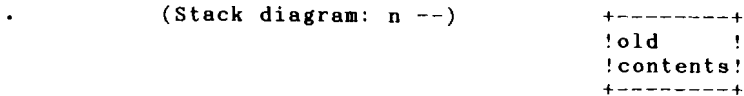
In Forth, all programs communicate over a single central location--the stack. Every Forth subroutine which requires parameters gets these parameters from the stack. The user has the responsibility of making sure the appropriate parameters are on the stack before the subroutine is called. Otherwise the called word will get some kind of garbage which can lead in the worst cases to a system crash. A Forth word can best be described by representing the logical flow of the function and in addition, clarifying the input and output parameters.



The Forth words in the appendix are also to be understood in this manner. Let us take a look at what happens to the stack when adding two number 7 12 + CR.



To output the results one writes:



The stack is now in the condition that it was in before the calculation.

4.3 Examples

We will now illustrate through the use of some short examples how one programs in Forth. The first is a kind of a language translator which may be thought of as an English-German dictionary. We need only the two related words `."` and `'"` as well as the definition delimiters required for every word, `':'` and `';'` .

We define:

```
: LOVE      ." LIEBE " ;
: I         ." ICH " ;
: QUEEN     ." KOENIGIN " ;
: MARY      ." MARIA " ;
```

Note: The space is an important delimiter in Forth. At least one space must separate each Forth word from neighboring words (a space is required after a quotation mark because it is a Forth word).

You can now ask for the translation of any learned word. If you type `QUEEN CR`, Forth answers with `KOENIGIN`. You can also translate sentences (though the translation will often be grammatically wrong): `I LOVE QUEEN MARY CR`

The name "dictionary" used to describe the list of Forth words takes on a literal meaning in this application.

Next we want to define a word called `2SORT` which sorts two numbers according to size. The stack diagram looks like this:

```
nl n2 --- min(n1,n2) max(n1,n2)
```

At the end of `2SORT` the larger of the two values will be at the top of the stack. The word definition for this is:

```
: 2SORT
    2DUP
    >
    IF
        SWAP
    ENDIF
;
STACK: nl n2
        nl n2 nl n2
        nl n2 f
        nl n2
                n2 nl
        nl n2 or n2 nl
```

The two values `nl` and `n2` must be duplicated once in the program because the comparison operator `>` removes two values from the stack and leaves an indicator 'true' (=1) or 'false' (=0) depending on whether `nl > n2` or not. The word `IF` removes the indicator from the stack. The branch between `IF` and `ENDIF` is only executed if the indicator was "true".

This branch exchanges the top two values on the stack.

We will use this word in our next definition (calculation of the greatest common denominator [GCD] of two numbers).

The Euclidean algorithm for determining the GCD of two positive integers is similar to the following:

- subtract the smaller of the two numbers (let us say n2) from the larger, n1, until the result, n3, of the subtraction is less than n2.
- subtract the result n3 from n2 until the new result, n4, is less than n3, and so on.
- end the procedure when the difference is 0. The number last subtracted is the GCD.

The stack diagram for the word GCD:

```
n1 n2 -- gcd
```

The word definition:

```
: GCD          n1 n2
  BEGIN        The loop will be repeated until
                no remainder occurs from the
                division
  2SORT        we make sure of the right ordering
  OVER        stack: n2 n1 n2
                small large small
  MOD         stack: n2 n3
  -DUP        we duplicate n3 only if n3>0
                Case 1: n3>0, then stack: n2 n3 n3
                else :                n2 0
  0=          Case 1: n2 n3 0
                else : n2 1
  UNTIL      Case 1: n2 n3 and back to repeat
                else : n2
;
```

The loop will be exited at UNTIL if a 0 is not found there.

The program is fast (about 10 times faster than an equivalent BASIC program) and can be used in any application just as any other Forth word.

At the end of this chapter we will introduce a program which adds two fractions and also reduces the result. This example will clearly illustrate the possibilities and dangers which Forth offers.

The stack diagram for the addition program for fractions is:

n1 d1 n2 d2 -- nsum dsum

in which n and d stand for numerator and denominator.

The Forth program:

```

: FRACTIONADD      ( N1 D1 N2 D2 ---- NSUM DSUM)
  DUP              N1 D1 N2 D2 D2
  5 ROLL           D1 N2 D2 D2 N1
  *                D1 N2 D2 P1
  3 ROLL           D1 D2 P1 AA
  4 PICK           D1 D2 P1 N2 D1
  *                D1 D2 P1 P2
  +                D1 D2 NSUM
  -ROT             NSUM N1 N2
  *                NSUM DSUM
  2DUP             NSUM DSUM NSUM DSUM
  GCD              NSUM DSUM GCD
  DUP              NSUM DSUM GCD GCD
  -ROT             NSUM GCD DSUM GCD
  /                NSUM GCD (REDUCED DENOMINATOR)
  -ROT             (REDUCED DEN.) NSUM GCD
  /                DENOMINATOR NUMERATOR
  SWAP
;

```

The program FRACTIONADD works, but it is difficult to read and follow. Forth programs should not be written this way. Programs become more readable if one breaks the problem down into smaller tasks and defines more words, such as the following:

Function	Stack diagram
NUMERATOR	n1 d1 n2 d2 --- numerator numerator=n1*d2+n2*d1
REDUCE	n d divisor --- num. reduced denom. reduced

The addition of fractions can then be formulated as follows:

: ADDFRAC

3 PICK	n1 d1 n2 d2 d1
OVER	n1 d1 n2 d1 d2
*	n1 d1 n2 d2 denominator
>R	n1 d1 n2 d2 (denom saved on return stack)
NUMERATOR	numerator
R>	numerator denominator
2DUP	num denom num denom
GCD	numerator denominator gcd
REDUCE	num reduced denom reduced

5. Music with the sound vocabulary

5.1 Description of the sound commands

Forth-64 contains a small vocabulary with which music or at least tones can be created. The vocabulary is called SOUND and contains the following words:

VOICE VOLUME ENVELOPE NOISE PULSE SAWTOOTH TRIANGLE and SID

Before you can call one of these words, you must first enter the command SOUND.

ENVELOPE n1 n2 n3 n4 n5 ---
 Definition of the waveform of the nth voice
 n1 - voice (1.. 3)
 n2 - attack (1..15)
 n3 - decay (1..15)
 n4 - sustain (1..15)
 n5 - release (1..15)

VOICE n1 n2 n3 n4 n5 n6 ---
 Creation of a tone in the nth voice
 n1 - voice (1.. 3)
 n2 - frequency (0..65535)
 n3 - wave type (17,33,65,129)
 n4 - key relationship low (0..255)
 n5 - key relationship high (0..15)
 The key relationship is relevant only
 for wave type 65=PULSE
 n6 - duration (-32768..32767)
 Duration of the created tone. A dummy delay
 loop will be executed once per entity.

VOLUME n ---
 Define the volume level (0..15)

NOISE --- 129
 The constant 129 is placed on the stack for
 creation of noise.

PULSE -- 65
 The constant 65 is placed on the stack for the
 creation of a tone with a square wave.

SAWTOOTH -- 33
 The constant 33 is placed on the stack for
 creation of a tone with a sawtooth wave.

TRIANGLE -- 17
 The constant 17 is placed on the stack for
 creation of a tone with a triangle wave.

SID

-- D400

The address of the SID (decimal 54272) is placed on the stack.

The command VOICE is to be given after the volume and waveform have been set.

5.2 A first composition with sound

With the words just described you can play a few tones on your C-64:

Enter the following commands:

```
15 VOLUME CR          (maximum volume for the new music)
1 9 3 0 8 ENVELOPE CR (these are the settings recommended
                       for piano in the C64 user's guide)
```

and finally

```
1 7493 PULSE 255 1 2000 VOICE CR (create the def. tone)
```

You now hear the famous concert-pitch A!

If you play around a bit with the settings for ENVELOPE and VOICE, you will notice a greater or lesser change in the tone (after VOICE) in contrast to the last tone.

When you have discovered a combination which you want to keep, you should define it as a new word.

You can recognize through these examples how easy it is to experiment with Forth and how Forth programs can be tested as the individual parts are written.

Try out the second and third voices in the same way:

```
2 0 15 240 0 ENVELOPE CR
and 3 96 10 0 10 ENVELOPE CR
```

In the Commodore 64 User's Guide, the second setting in combination with the waveform TRIANGLE and the third setting in combination with the waveform SAWTOOTH are valid parameters for organ and trumpet sounds. Your own experiments will no doubt bring you still closer to this ideal.

You can use the following program as a starting point to create a composition for piano, organ, and trumpet--a combination seldom used today.

SCR # 15

```

0 P ( CHAMBER MUSIC #1, PIANO, ORGAN, TRUMPET DEFINITIONS )
1 P FORTH DEFINITIONS DECIMAL
2 P          ( ENVELOPE DEFINITIONS)
3 P : PIANO          ( -- WAVE )
4 P   SOUND 1 9 3 50 8 ENVELOPE
5 P ;
6 P
7 P : ORGAN          ( -- WAVE)
8 P   SOUND 2 0 15 240 0 ENVELOPE
9 P ;
10 P
11 P : TRUMPET       ( -- WAVE)
12 P   SOUND 3 96 0 0 5 ENVELOPE
13 P ;
14 P
15 P  -->          ( DEFINITION CONTINUED ON NEXT PAGE)

```

SCR # 16

```

0 P ( CHAMBER MUSIC #2, PLAY )
1 P FORTH DEFINITIONS DECIMAL
2 P : PLAY          ( MAKE MUSIC)
3 P   15 VOLUME     ( FULL VOLUME)
4 P   PIANO ORGAN TRUMPET ( ENVELOPES)
5 P   BEGIN
6 P     3 RND 1+ ( RANDOM VOICE)
7 P     CASE
8 P       1 OF 1 PULSE      ENDOF
9 P       2 OF 2 TRIANGLE ENDOF
10 P      3 OF 3 SAWTOOTH ENDOF
11 P     ENDCASE
12 P     5000 RND 5000 + SWAP ( FREQUENCY)
13 P     240 5              ( KEY RELATIONSHIP)
14 P     16 RND 4 - 1000 * VOICE
15 P   ?TERMINAL UNTIL ; ( UNTIL STOP KEY PRESSED)

```

Load this program with 15 LOAD CR and start it with the command PLAY. It plays until you press the STOP key (probably quite soon).

6. Graphics with the graphics vocabulary

6.1 Lo-res

You can not only make music with Forth-64, you can also create pictures and graphics. The GRAPHIC vocabulary helps you with this in hi-res while some words in the primary vocabulary of Forth help in lo-res.

You have already become acquainted with some words for outputting to the screen in chapter 2: the colors and the words BORDER, SCREEN, and PEN.

If you want to write a character at an certain spot on the screen or you want to know what character is located there, you need the words S! and S@. Both words require the screen coordinates on the stack in the form (line, column):

```
S@      l c -- a
        yields the code of the character in the lth line,
        cth column

S!      a l c --
        writes (pokes) the character a in line l, column
        c on the screen
```

The following is a small program which tests the random number generator:

```
SCR # 20
0 P ( RANDOM NUMBER TESTER ?RND )
1 P FORTH DEFINITIONS DECIMAL
2 P : ?RND
3 P   ( INITIALIZE FIRST SCREEN)
4 P   1024 1000 ASCII 0 FILL
5 P   BEGIN
6 P   1000 RND   ( RANDOM 0.999)
7 P   40 /MOD   ( COLUMN, LINE)
8 P   SWAP     ( EXCHANGE)
9 P   2DUP S@  ( CHARACTER)
10 P  1+ -ROT  ( ADD 1)
11 P  S!      ( SAVE)
12 P        ?TERMINAL UNTIL ( UNTIL STOP IS PRESSED)
13 P ;
14 P   ;S
15 P
```

You can load this program with LOAD 20 and execute it with ?RND CR .

6.2 Hi-res

Better quality pictures can be created in the hi-res mode. The words which you need to do this are found in the GRAPHIC vocabulary which is described in its entirety in the next section. Enter the word GRAPHIC CR before you enter any graphics commands, or Forth will not understand the following commands.

You can see what the high-resolution screen looks like before you have drawn something on it by entering the command &HI-RES or simply by pressing the F3 key. The picture which you see consists of many small, multi-colored squares. Clear the screen with the command &CLEAR.

You must enter this command blindly (if you have not switched back to the text mode [F4]), meaning that the letters which you type will not appear on the screen. Your input will be accepted and processed nevertheless.

The screen and writing colors can also be set in the hi-res mode (independent of the text screen), with the graphics words &PAPER and &INK. Enter:

```
GRAPHIC  YELLOW &PAPER  BLACK &INK  &CLEAR CR
```

If you take a look at the graphics screen with F3, you will see that it looks much nicer than before.

If you want to put a very small point in the extreme lower-left corner, enter the following instruction:

```
1 0 0 &S! CR
```

(If Forth communicates its lack of understanding with the message "&S! ?", tell it that the word &S! is in the GRAPHIC vocabulary--command GRAPHIC CR--otherwise Forth will look in the SOUND vocabulary for the graphics words.)

Note that in high-resolution mode the origin (0,0) is the in the lower left-hand corner and the coordinates in the x-direction go up to 319 and up to 199 in the y-direction.

Another graphics word which you will probably use quite often is &LINE. It draws or erases a line between two given points, depending on whether or not the first parameter is zero (if zero, the line is erased). Example: &HI-RES 1 10 10 50 50 &LINE CR

You can save your current graphics screen to disk with the command &SAVE or load it in with &LOAD, for example:

```
" 1ST PICTURE" 8 &SAVE CR
```

You can learn how to paint with the cursor and function keys in the following program.

SCR # 25

```

0 P ( PAINT PICTURE #1, PS, AMOVE )
1 P FORTH DEFINITIONS DECIMAL
2 P
3 P 1 VARIABLE PS          ( PEN STATUS -1/0/1      )
4 P                        ( PEN ABSOLUTE MOVEMENT )
5 P                        ( COORDINATES  XO,YO    )
6 P : AMOVE      ( -- )
7 P   GRAPHIC PS @
8 P   0< 0= IF
9 P           PS @
10 P          (XO) @
11 P          (YO) @
12 P          &S!
13 P          ENDIF
14 P ; -->
15 P

```

SCR # 26

```

0 P ( PAINT PICTURE #3, RMOVE )
1 P FORTH DEFINITIONS DECIMAL
2 P                                (PEN AT DX,DY MOVE )
3 P : RMOVE      ( DX,DY -- )
4 P   GRAPHIC
5 P   2 ?ENOUGH
6 P   (YO) +! (XO) +!
7 P   AMOVE
8 P ;
9 P -->
10 P
11 P
12 P
13 P
14 P
15 P

```

SCR # 27

```

0 P ( PAINT PICTURE #3, CURSER MOVEMENTS )
1 P FORTH DEFINITIONS DECIMAL
2 P
3 P
4 P : CRIGHT          1 0  REMOVE ;
5 P : CLEFT           -1 0  REMOVE ;
6 P : CUP              0 1  REMOVE ;
7 P : CDOWN           0 -1  REMOVE ;
8 P
9 P  -->
10 P
11 P   CHECK FOR DIAGONAL
12 P   MOVEMENT DEFINED
13 P : UPRIGHT        1 1  REMOVE ;
14 P
15 P

```

SCR # 28

```

0 P ( PAINT PICTURE #4, CHG-PS TOGGLEPIX )
1 P FORTH DEFINITIONS DECIMAL
2 P
3 P                                     ( PEN STATUS CHANGES )
4 P : CHG-PS
5 P   PS @
6 P   0=
7 P   PS !
8 P ;
9 P                                     ( PIXEL ERASE/SET )
10 P : TOGGLEPIX
11 P   GRAPHIC (X0) @ (Y0) @
12 P   2DUP &S@
13 P   0= -ROT &S!
14 P ;
15 P  -->

```

SCR # 29

```

0 P ( PAINT PICTURE #4, PEN-UP )
1 P FORTH DEFINITIONS DECIMAL
2 P
3 P                                     ( PEN RAISED UP )
4 P : PEN-UP
5 P   -1 PS !
6 P ;
7 P
8 P
9 P
10 P
11 P
12 P
13 P
14 P
15 P

```

When you have compiled these screens (25 LOAD compiles all of them up to screen 29, or to screen 31 if you compile the programs on the distribution disk), you can paint your first pictures by command. For example, set the starting position of your picture in the middle of the graphics screen with 160 (X0) ! 100 (Y0) ! CR and then use CRIGHT to move to a position to the right, CUP to move up, etc. A new point will be set or erased at the new location depending on the value of the variable PS. If PS has the value 0, it works like an eraser, and if PS is 1, like a writing pen and if PS = -1, nothing happens--the pen is off the paper.

Drawing with these relatively long words is naturally somewhat tiresome. We will now draw with the cursor and function keys using the key assignment methods mentioned in chapter 2. To do this we must create a table (we call it PTABLE):

SCR # 30

```

0 P ( PAINT PICTURE #6, PRETURN,PTABLE )
1 P FORTH DEFINITIONS DECIMAL
2 P : PRETURN          ( SET LO-RES AND STANDARD KEY TABLE )
3 P GRAPHIC
4 P &LO-RES FFTABLE TO .NKEY
5 P ;
6 P 0 VARIABLE PTABLE  -2 ALLOT
7 P 29 CAP CRIGHT      ( ASSIGN CURSOR-RIGHT )
8 P 157 CAP CLEFT      ( ASSIGN CURSOR-LEFT )
9 P 145 CAP CUP
10 P 17 CAP CDOWN
11 P 133 CAP CHG-PS    ( CHANGE PS IF F1 )
12 P 134 CAP PEN-UP    ( PEN UP )
13 P 135 CAP TOGGLEPIX
14 P 136 CAP PRETURN
15 P 0 C,              --> ( END ENTERING OF KEY TABLE )

```

SCR # 31

```

0 P ( PAINT PICTURE #7 OF 7, PSTART )
1 P FORTH DEFINITIONS DECIMAL
2 P : PSTART          ( START PAINT )
3 P GRAPHIC
4 P 160 (X0) !
5 P 100 (Y0) !
6 P PTABLE TO .NKEY
7 P &HI-RES
8 P ; ;S
9 P
10 P
11 P FINISHED!
12 P
13 P
14 P
15 P

```

Compile the screens (30 LOAD CR, unless you have already compiled them with the 25 LOAD command), enter the command PSTART CR, and draw using the four cursor keys. Note: If you accidentally press a key to which a special function has not been assigned, the key assignment is deactivated until you press RETURN (see chapter 2).

6.3 Descriptions of the graphics commands

&CLEAR ---
Clear the graphics screen.

&HI-RES ---
Switch to the HI-RES mode.

&INK n ---
Set plotting color.

&LINE f x1 y1 x2 y2 ---
Draw/erase a line between x1,y1 and x2,y2 based on the condition of the flag f.

&LO-RES ---
Switch to the lo-res mode.

&LOAD st dev ---
Load hi-res graphics from device dev (disk=8), file st.

&PAINT addr ---
Initialize a byte with the paper and pen color at address addr in the graphics memory.

&PAPER b ---
Set background color.

&SAVE st dev ---
Save hi-res graphics to device dev (disk=8), file st. Example: " PICTURE1" 8 &SAVE CR .

&S@ x y --- f
Return the value of the pixel at x,y.

&S! f x y ---
Set (f=TRUE) or erase (f=FALSE) a point at x,y.

(&ADD) x y --- addr
Return the relative pixel address in the graphics memory for the coordinates x and y.

(&ECM) ---
Turn extended color mode on.

(&CADD) x y --- addr
Return the relative address of the coordinates x and y in the color memory.

(&MASK) x --- m
Return the mask of the xth bit of a byte in the graphics memory.


```

(&MCM)    ---
           Turn on multi-color mode.

(&SBM)    ---
           Set standard bit map.

(&SCM)    ---
           Turn on single-color mode.

(&TM)     ---
           Return to text mode.

(DX)      --- addr
           Temporary variable.

(DY)      --- addr
           Temporary variable.

(X0)      --- addr
           Temporary variable for x-coordinate.

(Y0)      --- addr
           Temporary variable for y-coordinate.

SET-AREA  addr1 addr2 ---
           Set address range for graphic mode in VIC.
           addr1=color range, addr2=pixel range
    
```

7. The Forth Assembler

7.1 General

Forth-64 contains a complete assembler. With this it is possible to program particularly time-critical functions and to call these as any other Forth word.

The user will not know (except through the speed with which a particular function is performed), whether a word is written in Forth or in assembler. The assembler need not be loaded separately--it is always available.

The Forth assembler naturally contains all of the 6502 commands. In addition, it even makes a kind of structured programming possible with the help of the words BEGIN, AGAIN, UNTIL, IF, ELSE, and ENDF. The words which return the conditions for the corresponding conditionals are called VS, >=, <, 0=, CS, and NOT and checking the individual bits of the status bytes.

There are some deviations from normal assembly language programming. For example, all mnemonics are concluded with a comma in order to prevent name conflicts with other Forth words. The same reverse-Polish notation exists in the assembler, which means that the operand is written first followed by the operator. A modifying term may come between the two (for immediate, indexed, or indirect addressing).

Some simple commands are:

4 * LDA,		LDA #4
XSAVE STX,	instead of	STX XSAVE
50) JMP,		JMP (0050)

If assembler programs are to be called as Forth words, a few things must be taken into consideration.

- Forth uses the X-register as a stack pointer. If the assembly language program makes use of the X-register in other ways, it must first save its value. The variable XSAVE is used for this purpose. It must naturally be restored at the end of the assembly language program.

- Since the stack serves as an interface for passing parameters, there are little aids for processing the information: the words BOT and SEC for passing parameters and the jump labels PUSH, PUSH0, PUT, PUTOA, and NEXT. The instructions BOT LDA and BOT 1+ LDA return the top two bytes on the stack. The second element on the stack can be obtained by indexing SEC instead of BOT.

- Assembly language programs can also begin with the word label. They are not then addressable directly from Forth but can only be used as subroutines from other assembly language programs.

- Finally, there are certain return points to Forth to which a jump must be made with JMP at the end of the assembly language program. If the stack is not to be changed, one uses the symbolic address NEXT, otherwise POP, PUSH, PUSH0A, or PUT as required.

7.2 The Assembler Vocabulary

This list does not describe all of the actual assembler mnemonics. Their exact meaning can be found in a book about 6502 assembly language programming. All of these commands are recognized by the Forth assembler provided a comma is added after the mnemonic.

Example: JMP, INX, ...

The structured words BEGIN, AGAIN, UNTIL, IF, ELSE, and ENDIF are similarly explained. Their meanings are the same as those for the corresponding Forth vocabulary.

This leaves the Forth-specific address modifiers as well as some global assembler labels.

IP	:	Address of the I(nterpretive) P(ointer)
W	:	Address of the code field pointer
N	:	Address of an 8-byte scratch area
R	:	Pointer to the return stack
XSAVE	:	Address of the save register for X
UP	:	Address of an 8-byte area for the user parameters
.A	:	Denotes the accumulator addressing mode
#	:	Immediate addressing
)	:	Indirect addressing
,X ,Y	:	Indexed addressing
X))Y	:	Indirect indexed addressing
BOT	:	Address of the low byte of a 16-bit quantity in ,X mode. The X-register points to the current data stack in the zero-page.
BOT 1+	:	Address of the high byte of a 16-bit quantity in ,X mode. The X-register points to the current data stack in the zero-page.
SEC	:	Address of the low byte of the second quantity on the data stack.
SEC 1+	:	Address of the high byte of the second quantity on the data stack.

PUT : Address of a routine which replaces the high byte of the data stack with the contents of the accumulator and the low byte with the top byte on the machine stack; used by NEXT.

PUSH : Like PUT, but the word is pushed on the stack.

SETUP : Address of a routine which transfers (pops) n 16-bit words from the stack to the temporary area N. The number n is expected in the accumulator.

BINARY : Address of a routine which removes the top word from the data stack and then executes PUT.

POP : Address of a routine which removes a word from the data stack.

POPTWO : Like POP, but removes two words.

PUSHOA : Like PUSH, but the high byte is not taken from the stack but is set to 0.

NEXT : Address of the inner interpreter. All routines jump to NEXT directly (NEXT JMP,) or indirectly (such as PUSH, PUT).

7.3 A Small Assembly Language Program

The C-64 user knows of the possibility to redirect output from the screen to some other device with the help of the CMD command:

```
OPEN 4,4
CMD 4
```

An assembly language routine in Forth which does something like this could look like the following:

```
SCR # 40
0 P ( SCREEN OUTPUT TO PRINTER : CMD )
1 P FORTH DEFINITIONS HEX
2 P CODE CMD      ( START OF AN ASSEMBLY LANGUAGE PROGRAM)
3 P   XSAVE STX,  ( SAVE STACK POINTER)
4 P   4 # LDA, TAX, 0 # LDY,
5 P   FFBA JSR,    ( SETPAR )
6 P   0 # LDA, FFBD JSR, FFC0 JSR,
7 P   4 # LDX, FFC9 JSR, ( CKOUT )
8 P   XSAVE LDX,
9 P   NEXT JMP,
10 P END-CODE
11 P
12 P CODE RESET   ( RESET SCREEN)
13 P   XSAVE STX, FFCC JSR, XSAVE LDX,
14 P NEXT JMP,
15 P END-CODE DECIMAL
```

You can now redirect output from the screen to the printer with the command CMD. You can send it back to the screen with RESET. A routine which outputs screens first to the screen and then to the printer looks like this:

```
SCR # 41
0 P ( PRINT ROUTINE: PRINT, ANALOG TO LIST )
1 P FORTH DEFINITIONS DECIMAL
2 P : PRINT      ( SCR -- )
3 P   RESET
4 P   DUP LIST  ( FIRST TO SCREEN )
5 P   CMD LIST  ( THEN PRINTER   )
6 P   RESET
7 P ;
8 P ;S
9 P
10 P
11 P
12 P
13 P
14 P
15 P
```

When you have loaded the word PRINT, you can print the above FORTH screen with the command: 41 PRINT CR .

{This page left blank itentionally}

8. Miscellaneous

A few suggestions when programming Forth:

- Use short words.
- Use the words !CSP and ?CSP.
- Check at the beginning of each word to see if enough space is left on the stack for the word to function properly (word ?ENOUGH). Most system crashes result from stack overflow or underflow.
- Execute the BOOT parameters once you have tested your new words and want to keep them. If your system should crash later, then you can recover the complete system with RUN/STOP + RESTORE and a bit of luck.

The BOOT parameter can be set as follows:

```

HERE FENCE !
HERE 28 +ORIGIN !      ( FENCE )
HERE 30 +ORIGIN !      ( DP   )
LATEST 12 +ORIGIN !    ( TOP NFA)

```

If you have established a new vocabulary, you must actualize it with the appropriate reference:

```

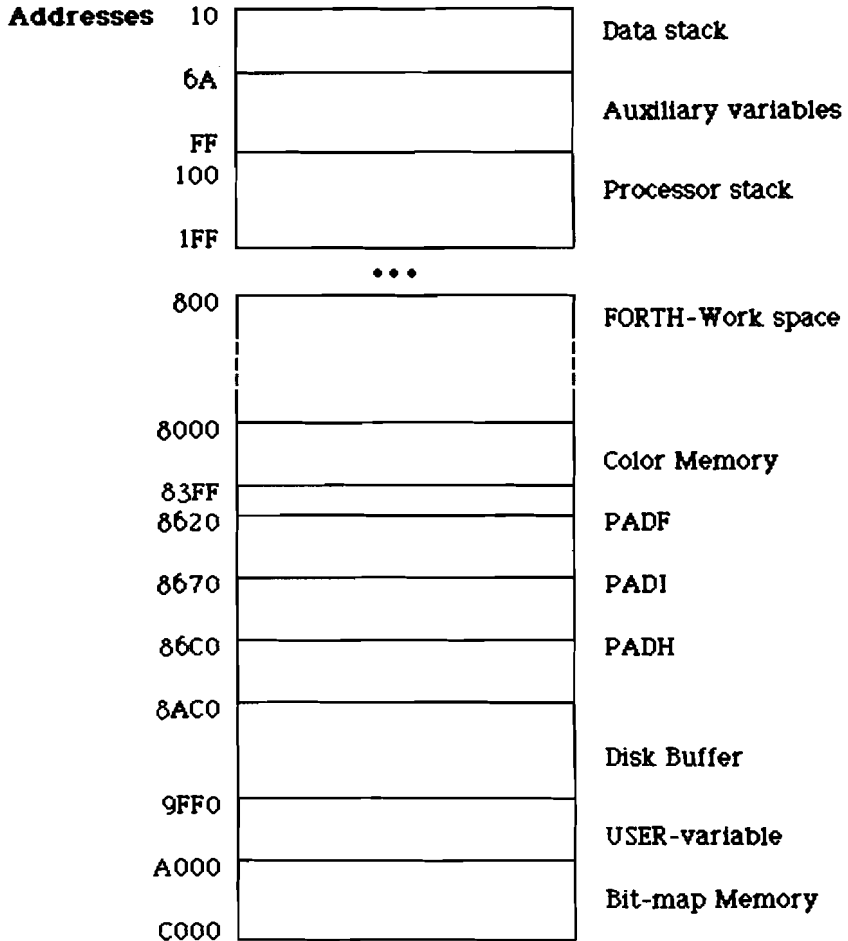
' vocabulary name 6 + 32 +ORIGIN !

```

- The error message NO CHANNEL from the disk drive can be removed with the command CLOSE-SCREEN. If this doesn't work, the assigned channel must be freed with a selective CLOSE.
- Forth words may never contain characters with an ASCII value greater than 127, in particular no upper-case letters in screen mode 2 of the C-64 (upper/lower case mode).

Strings such as ." This is text that works!", may contain such characters without difficulty.

Appendix 1 Memory Map



All operating system routines are available without limitation provided they do not access the data stack in the zero page.

Appendix 2

The Forth vocabulary

The following list includes only words in the dictionary "FORTH". Words in the other vocabularies EDITOR, SOUND, GRAPHIC, and ASSEMBLER have already been described in their own chapters.

*** Symbols used:

addr	memory address
b	8-bit byte (left byte=0)
c	7-bit ASCII character (right-justified, msb=0)
d	32-bit (double precision) integer
f	logical indicator (flag), 0=false, else true
ff	false flag (=0)
n	integer (16-bit)
tf	true flag (<>0)
u	unsigned 16-bit integer
st	string variable (corresponds to n,b for address and length of a string)

In some places, numbers will appear in the stack diagram instead of the above symbols. These are constants which are to be used as synonyms of system address in the C-64. They are given in hexadecimal form.

Upper-case letters on the right-hand side indicate certain attributes of the definition:

P	precedence bit set (definition is "immediate")
U	user variable

! n addr ---
Stores the word n at addr (pronounced "store").

!CSP Stores stack pointer in CSP (corresponds to ?CSP).

" --- st
Places the following text concluded with a quotation mark in the PAD buffer as a string constant.

d1 --- d2
Creates the next character for output from a double-length integer. The result d2 is the quotient of the division by BASE. Used between <# and #> (see also #S).

#> d --- addr count
Ends the numeric conversion for output. The result on the stack is the address and length (in bytes) of the string to be printed. Can be directly accepted by TYPE.

#S d1 --- d2
Creates ASCII text in the text output buffer by using #, until the double word d2 has the value zero. Used between <# and #>.

\$! st1 st2 ---
Store string st1 in the string variable st2.

\$+ st1 st2 --- st
Concatenation of two strings; result in PAD.

\$2C+ st n --- st
Lengthens the string st by the word n (2 characters).

\$C+ st b --- st
Lengthens the string st by the character b.

\$LEFT n st --- st
Yields the first n characters of the string st in PAD.

\$LEN st --- b
Returns current length of a string.

\$MID n1 n2 st --- st
Returns the characters n1 to n2 of the string st (in PAD).

\$MLEN st --- n
Returns the maximum character length of the string variable st.

- (FIND) addr1 addr2 --- pfa b tf ;if found
 addr1 addr2 --- ff ;if not found
 The dictionary is searched starting at the name-field address addr2 for the string at addr1. If the string is found, the address of the parameter field, the length byte of the name field, and the true indicator are all placed on the stack, else only the false indicator is left on the stack. This word is called by -FIND and is normally used only by the system.
- (KEY) --- c
 Run-time routine of KEY. Returns the code of the next-pressed key (without waiting for CR).
- (LINE) n1 n2 --- addr count
 Returns the length and internal buffer address of line n1 on screen n2.
- (LOOP) n ---
 The run-time procedure compiled by the LOOP instruction; increments the loop counter by 1 and checks the exit condition (system routine).
- (NUMBER) d1 addr1 --- d2 addr2
 Converts the ASCII text at address addr1+1 to the current base. The new value is constructed in d1 and placed in d2. addr2 is the new address of the first unconverted character. Used by NUMBER. (system routine)
- (SETFPA) chann dev ---
 Serves as label for assembler routines in order to set file parameters (system routine).
- (SETFNA) st ---
 Serves as label for assembler routines in order to pass a new filename to the operating system.
- * n1 n2 --- prod
 The result is the product of the top two stack values.
- */ n1 n2 n3 --- n4
 n4=n1*n2/n3. The 31-bit intermediate result n1*n2 is divided by n3. The accuracy is therefore greater than the sequence n1 n2 * n3 /
- */MOD n1 n2 n3 --- n4 n5
 n5 is the quotient, n4 the remainder of the operation n1*n2/n3. A 31-bit intermediate result allows higher accuracy.
- + n1 n2 --- sum
 n1 and n2 are added.

- 1+ n1 --- n2
 Increments n1 by 1.

- 10* n1 --- n2
 Multiplies n1 by 10 (fast assembler
 multiplication).

- 2! d addr ---
 Store double word at address addr.

- 2+ n1 --- n2
 Increment n1 by 2.

- 2@ addr --- d
 Copy double word at address addr to the stack.

- 2CONSTANT d ---
 A definition word. Defined in the form d 2CONSTANT
 cccc. When cccc is called, the double-precision
 constant d is placed on the stack.

- 2VARIABLE d ---
 A definition word. Defined in the form d 2VARIABLE
 cccc. When cccc is called, the address of the
 variable which contains the double word is placed
 on the stack.

- 2DROP d ---
 Removes two words from the stack.

- 2DUP d --- d d
 Duplicates a double word on the stack.

- 2SWAP d1 d2 --- d2 d1
 Exchanges two double words on the stack.

- ;
 Used in a ':' definition in the form:
 ; cccc ... ;
 Defines a new word in the dictionary which is
 equivalent to the instructions represented here
 with "...". The definition is ended with ; .
 The context vocabulary is set to the current
 vocabulary; words whose precedence bit is set are
 executed instead of compiled. P

- ;
 The ':' definition and the compile mode are ended. P

- ;
;S Used in order to stop the compilation of a screen P
 at any location.

- < n1 n2 --- f
 Sets the flag to true if n1<n2, else false.

- ?DISC Checks the condition of the disk drive and outputs this on the screen. Called when the light on the drive flashes.
- ?ENOUGH n ---
A check is made to see if at least n values can still be stored on the stack.
- ?ERROR f n ---
Outputs the error message n if the flag f is true. Line n from SCR#4 is printed. n may also be larger than 15, resulting in lines from the following screens being printed.
- ?EXEC Outputs an error message if the system is not in execute mode.
- ?LOADING Outputs an error message if loading is not currently going on.
- ?PAIRS n1 n2 ---
Outputs an error message if n1<>n2. This instruction checks to see if two structured language elements belong together or not.
- ?STACK Outputs an error message if a stack overflow occurs.
- ?TERMINAL --- f
A test is made to see if the RUN/STOP key is being pressed. A false flag indicates that this is not the case.
- @ addr --- n
The contents of address addr are placed on the stack.
- ABORT
Initializes the two stacks and sets the execute mode. The computer uses standard input and output and outputs the start-up message.
- ABS n --- u
The absolute value of n is generated.
- AGAIN P
Used in a ':' definition in the form:
BEGIN ... AGAIN
Jumps back to the corresponding BEGIN. The stack remains unchanged. The loop can only be exited through the sequence R> DROP .

BLK --- addr
User variable. Contains the number of the block which is currently being compiled. Zero indicates input from the terminal.

BLOAD vaddr st n2 n3 ---
Binary load of a file from disk.
vaddr is the start address at which the file is to be loaded.
st denotes the filename.
n2 and n3 indicate the channel (0 or 1) and the device number (=8) of the file to be loaded. Note: When using channel 1, the file is placed at the original address. The value of vaddr is then irrelevant, but may not be omitted.

BLOCK n --- addr
Places on the stack the memory address addr of the block buffer which contains block n. If block n is not currently in memory, it will be written to the "oldest" buffer. If the contents of this buffer are marked for "UPDATE," its contents are first written to disk (see BUFFER, R/W, UPDATE, FLUSH).

BLUE --- 6
Returns the constant 6 as the color value for blue.

BMOVE addr ---
Moves a string from PAD to addr.

BORDER b ---
Sets the color of the screen border to color b.

BROWN --- 9
Returns the constant 9 for the color value of brown.

BSAVE addr1 addr2 st n2 n3 ---
Binary save of a file to disk.
addr1 and addr2 indicate the range (from-to) which is to be written to the file.
st denotes the filename.
n2 and n3 denote the channel (0 or 1) and device number (=8) of the file to be saved.

BUFFER n --- addr
Finds the next disk block buffer and assigns it to block n. If the current contents of the buffer are marked as updated, they will first be written to the disk before the next block is read. The block is not read from the disk. The address returned on the stack is the start address of the data to be read.

- C!** **b addr ---**
Stores the byte **b** in the address **addr**.
- C,** **b ---**
Stores the byte **b** in the next free space in the dictionary. The dictionary pointer is incremented by 1.
- CAP** **b ---**
Writes a three-byte entry of the form **b (1 byte) n** (the CFA of the following word) in the dictionary. Required for key assignment, for example: **133 CAP ?DISC .**
- CASE** **n ---- n**
CASE and **ENDCASE** together with **OF** and **ENDOF** offer the ability to execute case conditionals in Forth. These words are used as in the following example:
CASE
n1 OF instruction for alternative 1 ENDOF
n2 OF instruction for alternative 2 ENDOF
...
nk OF instruction for alternative k ENDOF
ENDCASE
Alternative **i** is then executed if **ni** agrees with the value placed on the stack prior to execution of the **CASE** instruction. **n** is removed from the stack by **ENDCASE**.
- C/L** **--- n**
Returns the number of characters per line (64).
- C@** **addr --- b**
Places 8 bits of **addr** on the stack (see **@**).
- CFA** **pfa --- cfa**
Converts the parameter-field address to its code-field address.
- CIA1** **--- DC00**
Returns the address of the **CIA1** in the C-64.
- CIA2** **--- DD00**
Returns the address of the **CIA2** in the C-64.
- CLEAR** **---**
Clears the stack.
- CLIT** **--- b**
Exactly as **LIT**, only works with a byte instead of a word. **CLIT** executes faster than **LIT**.
- CLOSE** **n1 n2 ---**
The device **n2** addressed over channel **n1** with **OPEN** is closed.

CR

Causes a linefeed at the terminal.

CREATE

A word with which one can construct new Forth words. For example:

```
CREATE cccc
```

This creates a header for the word to be defined, cccc. The actions which this word is to execute must be compiled into the word with COMPILE, for example.

The smudge bit is set. The word cannot be addressed (it will not be recognized) until the smudge bit is cleared (command SMUDGE).

CREATE-SCREEN

Establishes a file SCR with a length of 2200 sentences (137 records). The message RECORD NOT PRESENT can be ignored here.

In addition, the screens 4 and 5 (error messages) which should be in the disk buffers are copied into the file.

CSP

```
--- addr U
User variable. Contains the stack pointer.
```

CURRENT

```
--- addr
User variable. Contains an indirect reference to the name field address of the definition which was just defined. (see CONTEXT)
```

CYAN

```
--- 3
Returns the constant 3 as the color code for cyan.
```

D+

```
d1 d2 --- dsum
Adds the double words dsum=d1+d2.
```

D+-

```
d1 n -- d2
The double word d1 contains the sign of n.
```

D.

```
d ---
Outputs a double word to the output device. (pronounced "D-dot")
```

D.R

```
d n ---
Outputs the double word d as a signed number in an n-character field.
```

DABS

```
d --- u
Returns the absolute value of the double word.
```

DECIMAL

Sets the I/O conversion base to 10.

GREEN --- 5
 Constant for the color value of green.

HELP ---
 Returns the function key assignments.

HERE --- addr
 The next free address in the dictionary is placed on the stack.

HEX ---
 Sets the conversion base to 16 (hexadecimal).

HLD --- addr
 User variable which contains the last character of a string of numerical output conversion.

HOLD c ---
 Used between <# and #>. Transfers the ASCII character to the string for numerical output conversion.

I --- n
 Used inside a DO ... LOOP in order to place the loop index on the stack.

ID. addr ---
 Outputs the name of the word with NFA addr.

IF f --- ;run-time
 Used in a ':' definition in the form:
 IF (tf) ... ENDIF
 IF (tf) ... ELSE (ff) ... ENDIF
 If f is true, the commands immediately behind IF are executed; if f is false, a jump is made to ELSE (if present) or ENDIF.

IMMEDIATE
 Marks the last-defined word as "immediate." This means that the word will be executed during compilation instead of being compiled. The user can force the compilation of an immediate definition through a preceding [COMPILE] instruction.

IN --- addr
 User variable. Contains the byte offset in the current input text buffer from which the next text will be fetched. The instruction WORDS uses and changes the contents of IN.

NOK ---
Turns Forth's OK message off.

NOP ---
A routine which does nothing (dummy routine).

NOTABLE --- addr
Variable which should contain the value 0.
Its address is saved in the variable FTABLE if no
function keys are assigned.

NUMBER addr --- d
The string which begins in address addr and whose
length is immediately before the text is converted
to a double-precision word based on the current
conversion base. The placement of decimal point
(if any) is stored in DPL. Otherwise the decimal
point has no function. An error message will be
given in the event of a conversion error.

OK ---
Turns Forth's "OK" message on.

OPEN st n2 n3 ---
A file with filename st is opened on device n2
over channel n3.

OPEN-SCREEN ---
Opens file SCR (containing the screens). Usually
need not be given by the user because Forth
manages this file by itself.

OR n1 n2 --- or
The result is the logical ORing of n1 and n2.

ORANGE --- 8
Returns the constant 8 as the color value for
orange.

OUT --- addr
User variable. Contains a value which is used and
changed by EMIT. The user can read or change the
value in order to create formatted output.

OVER n1 n2 -- n1 n2 n1
The second word on the stack is copied onto the
top of the stack.

PAD --- addr
Returns the address of the text output buffer.

PAPERCOLOR --- n
A variable containing the color value of the paper
for graphics.

R> --- n
Places the top word of the return stack on the data stack.

READ addr count n1 n2 --- f
count bytes are read from device n2 over channel n1 at address addr; error flag 0=OK.

RED --- 2
Returns the constant 2 as color value for red.

REPEAT
Used in a ':' definition in the form: P,C2
BEGIN ... WHILE ... REPEAT
At run-time this results in an unconditional jump to the corresponding BEGIN.

RND n1 --- n2
Returns a random number in the range 0 to n1-1.

RNDNR --- n
A variable used for random number generation with RND.

ROLL b ---
Rotates the top b stack elements one position down and puts the former bth element on top. ROLL is a generalization of ROT (which is equivalent to 3 ROLL).

ROT n1 n2 n3 --- n2 n3 n1
Rotates the three top words on the stack.

RP!
Initializes the return stack pointer.

S! b1 b2 b3 ---
Writes the ASCII value b1 in line b2, column b3 of the text RAM (address 1024 ff.).

S->D n --- d
The top stack word is converted to a double precision value with the same sign and magnitude.

S0 --- addr U
Variable with which the data stack is initialized.

S@ b1 b2 --- b3
Returns the ASCII value in text RAM at line b1, column b2.

VICCR2 --- D016
 A constant which returns the address of the second control register in the VIC.

VIC-ADDPTR --- D018
 A constant which returns the address of the address pointer register in the VIC.

VIC-BORDER --- D014
 A constant which returns the address of the VIC register which contains the border color of the screen.

VIC-BG0 --- D015
 A constant which returns the address of the VIC register which contains the (normal) background color.

VIDEOAREA --- n
 A variable which contains the address of the video area (HEX 400).

VOC-LINK --- addr U
 User variable. Contains the address of the vocabulary last used. All vocabularies are chained though these fields. A FORGET can therefore have effect on multiple vocabularies.

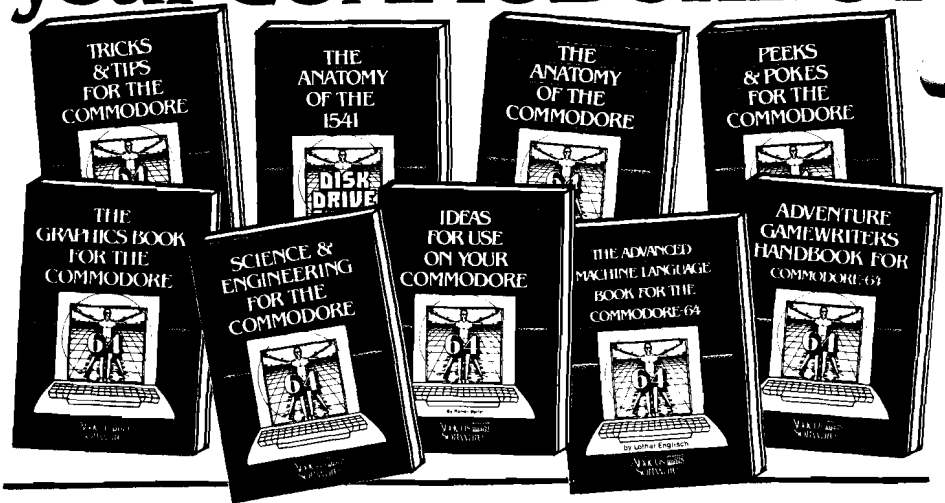
VOCABULARY E,L
 A definition word. Used in the form:
 VOCABULARY cccc IMMEDIATE
 in order to define a new vocabulary cccc. A later call of cccc makes it the CONTEXT vocabulary, the first vocabulary searched for an INTERPRET instruction. The instruction sequence:
 cccc DEFINITIONS
 makes cccc the CURRENT vocabulary in which new definitions will be placed. Vocabulary names are by convention declared as immediate.

VLIST
 Outputs all of the defined names in the dictionary, starting with the CONTEXT vocabulary.

WARNING --- addr U
 User variable. Contains a value for controlling the system messages, where
 1: error messages are output from disk screen 4
 0: only error numbers are output
 -1: (ABORT) is executed

WHERE ---
 WHERE is automatically initialized when an error occurs while loading screens. The corresponding screen is automatically displayed for editing.

Required Reading for your COMMODORE 64



TRICKS & TIPS FOR YOUR C-64 - treasure chest of easy-to-use programming techniques. Advanced graphics, easy data input, enhanced BASIC, CP/M, character sets, transferring data between computers, more.
ISBN# 0-916439-03-8 275 pages \$19.95

GRAPHICS BOOK FOR C-64 - from fundamentals to advanced topics this is most complete reference available. Sprite animation, Hires, Multicolor, lightpen, IRQ, 3D graphics, projections. Dozens of samples.
ISBN# 0-916439-05-4 350 pages \$19.95

SCIENCE & ENGINEERING ON THE C-64 - starts by discussing variable types, computational accuracy, sort algorithms, more. Topics from chemistry, physics, biology, astronomy, electronics. Many programs.
ISBN# 0-916439-09-7 250 pages \$19.95

ANATOMY OF 1541 DISK DRIVE - bestselling handbook available on using the floppy disk. Clearly explains disk files with many examples and utilities. Includes complete commented 1541 ROM listings.
ISBN# 0-916439-01-1 320 pages \$19.95

ANATOMY OF COMMODORE 64 - insider's guide to the '64 internals. Describes graphics, sound synthesis, I/O, kernel routines, more. Includes complete commented ROM listings. Fourth printing.
ISBN# 0-916439-00-3 300 pages \$19.95

IDEAS FOR USE ON YOUR C-64 - Wonder what to do with your '64? Dozens of useful ideas including complete listings for auto expenses, electronic calculator, store window advertising, recipe file, more.
ISBN# 0-916439-07-0 200 pages \$12.95

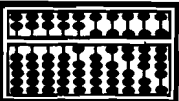
PEEKs & POKes FOR THE C-64 - programming quickies that will simply amaze you. This guide is packed full of techniques for the BASIC programmer.
ISBN# 0-916439-13-5 180 pages \$14.95

ADVANCED MACHINE LANGUAGE FOR C-64 - covers topics such as video controller, timer and real time clock, serial and parallel I/O, extending BASIC commands, interrupts. Dozens of sample listings.
ISBN# 0-916439-06-2 210 pages \$14.95

ADVENTURE GAMEWRITER'S HANDBOOK - is a step-by-step guide to designing and writing your own adventure games. Includes listing for an automated adventure game generator.
ISBN# 0-916439-14-3 200 pages \$14.95

Call today for the name of your nearest local dealer Phone:(616) 241-5510

Other titles are available, call or write for a complete free catalog. For postage and handling include \$4.00 (\$6.00 foreign) per order. Money order and checks in U.S. dollars only. Mastercard VISA and American Express accepted. Michigan residents include 4% sales tax. CANADA: Book Center, Montreal Phone: (514) 332-4154

You Can Count On  **Abacus Software**

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510

Break the BASIC language barrier



VIDEO BASIC-64 - ADD 50+ graphic and sound commands to your programs with this super development package. You can distribute free RUN-TIME version without paying royalties!
 ISBN# 9-916439-28-7 \$59.95

BASIC COMPILER 64 - compiles the complete BASIC language into either fast 6510 machine language and/or compact speedcode. Get your programs into high gear and protect them by compiling.
 ISBN# 0-916439-17-8 \$39.95

MASTER-64 - professional development package for serious applications. Indexed file system, full screen management, programmer's aid, BASIC extensions, 100 commands.
 ISBN# 9-916439-21-6 \$39.95

PASCAL-64 - full Pascal with extensions for graphics, sprites, file management, more. Compiles to 6510 machine code and can link to Assembler/Monitor routines.
 ISBN# 0-916439-10-0 \$39.95

ADA TRAINING COURSE - teaches you the language of the future. Comprehensive subset of the language, editor, syntax checker/compiler, assembler, disassembler, 120+ page guide.
 ISBN# 0-916439-15-1 \$59.95

FORTH-64 - loaded with hires graphics, complete synthesizer control, full screen editor, programming tools, assembler.
 ISBN 0-916439-32-1 \$39.95

C LANGUAGE COMPILER - a full C language compiler. Conforms to the Kernighan & Ritchie standard, but without bit fields. Package includes editor, compiler and linker.
 ISBN# 0-916439-28-3 \$79.95

ASSEMBLER MONITOR-64 - a macro assembler and extended monitor package. Assembler supports floating point constants. Monitor supports bank switching, quick trace, single step, more.
 ISBN# 0-916439-11-9 \$39.95

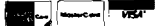
XREF-64 - indispensable tool for BASIC programmer cross-references all references to variable and line numbers.
 ISBN# 0-916439-27-5 \$17.95

OTHER TITLES ALSO AVAILABLE - WRITE OR CALL FOR A FREE COMPLETE CATALOG

Call today for the name and address of your nearest local dealer.

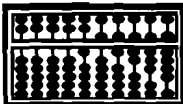
PHONE: (616) 241-5510

For postage and handling include \$4.00 (\$8.00 foreign) per order. Money order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted. Michigan residents incl 4% sales tax.



FREE PEEKS & POKES WALL POSTER INCLUDED WITH EVERY SOFTWARE PURCHASE

You Can Count On
Abacus



Software

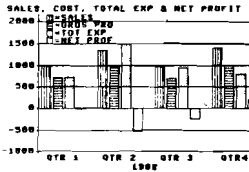
P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510

Make your '64 work fulltime

MAKE YOUR OWN CHARTS...

CHARTPAK-64

produces professional quality charts and graphs instantly from your data. 8 chart formats. Hardcopy in two sizes to popular dot matrix printers \$39.95
ISBN# 0-916439-19-4

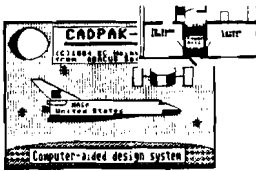


Also Available CHARTPLOT-64 for unsurpassed quality charts on plotters.
ISBN# 0-916439-20-8 \$84.95

DETAIL YOUR DESIGNS...

CADPAK-64

superb lightpen design tool. exact placement of object using our Accu-Point positioning. Has two complete screens. Draw LINES, BOXES, CIRCLES, ELLIPSES; pattern FILLING; freehand DRAW; COPY sections of screen; ZOOM in and do detail work. Hard copy in two sizes to popular dot matrix printers.
ISBN# 0-916439-18-6 \$49.95



CREATE SPREADSHEETS & GRAPHS...

POWER PLAN-64

not only a powerful spreadsheet packages available, but with built in graphics too. The 275 page manual has tutorial section and HELP screens are always available. Features field protection; text formatting, windowing; row and column copy, sort; duplicate and delete.
ISBN# 0-916439-22-4
\$49.95

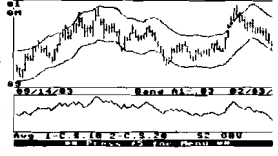
Coordinates: E119	POWER PLAN-64		
	A	B	C
1	Sales	440	640
2	Distributors	47.2	54.2
3	Retailers	27.9	35.4
4	Mail Order	18.5	23.7
5			
6		93.6	113.3
7			
8	Expenses		
9	Materials	8.2	9.2
10	Office	2.0	2.8
11	Shipping	4.4	5.0
12	Advertising	12.9	13.8
13	Payroll	10.3	10.7
14			
15		38.0	41.5
16			
17	Profit	55.6	71.8

FREE PEEKS & POKES POSTER WITH SOFTWARE
For name & address of your nearest dealer call (616) 241-5510

CHART YOUR OWN STOCKS...

TAS-64

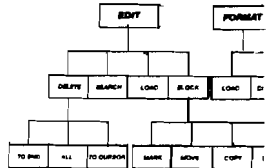
sophisticated technical analysis charting package for the serious stock market investor. Capture data from DJN/RS or Warner services or enter and edit data at keyboard. 7 moving averages, 3 oscillators, trading bands, least squares, 5 vol ume indicators, relative charts, much more. Hardcopy in two sizes, most printers.
ISBN# 0-916439-24-0 \$84.95



DO YOUR OWN WORD PROCESSING

TEXTOMAT-64

flexible wordprocessing package supporting 40 or 80 columns with horizontal scrolling. Commands are clearly displayed on the screen awaiting your choice. Quickly move from editing to utilities. Will work with virtually any printer.



ISBN# 0-916439-12-7 \$39.95

ORGANIZE YOUR DATA...

DATAMAT-64

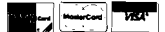
powerful, yet easy-to-use data management package. Free form design of screen using up to 50 fields per record. Maximum of 2000 records per diskette. Complete and flexible reporting. Sorting on multiple fields in any combination. Select records for printing in desired format.

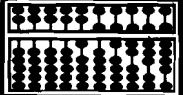
INVENTORY FILE	
Item Number	Description
Onhand	Price
Location	
Record. Pt.	Record. Qty.
Cost	

ISBN# 0-916439-16-X \$39.95

Other titles available. For FREE CATALOG and name of nearest dealer, write or call (616) 241-5510. For postage and handling, include \$4.00 (\$6.00 foreign) per order. Money Order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted. Michigan residents include 4% sales tax.

CANADA: Book Center, Montreal (514) 332-4154



You Can Count On  **Abacus Software**

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510