PRO·LINE SOFTWARE

POWER 64™

Toolkit for BASIC programmers includes:
MOREPOWER 64

by Brad Templeton

PRO·LINE SOFTWARE

POWER 64™

Toolkit for BASIC programmers includes: MOREPOWER 64

POWER 64™

Programmed for the Commodore 64™ Computer and Commodore 1541™ Disk Drive.

Program by: Brad Templeton

Manual by: Jim Butterfield

Manufactured and distributed exclusively by:

Pro-Line Software Ltd.,
755 The Queensway East, Unit 8,
Mississauga, Ontario, L4Y-4C5.
(416) 273-6350

FOREWORD.

Brad Templeton showed me his first POWER system in 1979. It was an early version, but had many of the features of the present package. I quickly became addicted to its use ... Basic programs came together more easily with POWER in place.

I was using POWER long before it became commercially available; this made me a logical candidate to write the documentation.

POWER has been available for PET/CBM computers for some time. Now there's POWER 64 for the Commodore 64 computer.

This manual is written to help you make good use of the wide range of features available in the POWER 64 package.

Jim Butterfield,
Toronto.
February, 1983.

TABLE OF CONTENTS.

TABLE OF CONTENTS. (CONTINUED)

POWER 64

1. INTRODUCTION TO POWER 64.

Welcome to the POWER 64 system! POWER 64 adds new capability to your Commodore 64 computer. With POWER 64, you'll write programs better, faster and more easily. Your Basic programs will test and debug more conveniently; and changes and updates become a snap.

POWER 64 makes Commodore 64 Basic programs easy to use and change. It does this with special 'instant action' keyboard features and with additional commands. The 64 is a friendly machine; its screen editing makes it easy to use. With POWER 64, the friendship will ripen into... well, closer friendship.

The first thing you'll notice about the POWER 64 system is its cursor action. Not only do cursor movements repeat automatically; they seem to reach beyond the screen to dig out program lines above and below the screen itself.

Next, you'll spot the 'instant keywords'. Tapping a single (shifted) key can produce a complete Basic keyword to speed your programming and perhaps improve your spelling. More: you can define a single key to mean anything you like: a complete line of Basic, if you like. Even this isn't the limit of what you can do with a single key: one key can invoke a whole subroutine of activity!

A whole family of special commands are featured in the POWER 64 system. They will be described in detail later in this manual. For the moment, here's a quick summary, with optional information enclosed in parentheses:

AUTO (line no.)(,increment)
which gives automatic line prompting.

DEL line-range
which deletes a range of Basic program lines.

DUMP
which dumps the values of active Basic variables, so that you can see them and change them.

FIX and PTR
which restore Basic pointers.

OFF
which disables POWER 64.

RENUM (increment)(,new start)(,line-range)
which renumbers all or part of a Basic program.

SEL R, I, K or P + or -
which turns on or off several 'instant key' activities.

TRACE (LIST, #, ↑, #↑ or LIST↑)
which engages a powerful Trace facility for testing Basic programs.

WHY
which helps to identify the cause of a Basic error.

EXEC file-number
which assigns control to a tape or disk file, allowing powerful features such as a Basic program merge.

/..pattern.. (/line-range)
which searches for certain words or phrases in your Basic program.

S/..pattern../..substitute.. (/line-range)
which substitutes words or phrases with other information.

TEST
which creates a special area to work with Basic programs different from the one being worked in.

BACK
Which restores the original Basic program after a TEST.

2. STARTING UP.

** NOTE: How you start POWER 64 depends on whether you have a cartridge version or disk version. Below you'll find out about loading the disk version.

Before POWER 64 can do anything, you have to load it from disk and activate it. Until you put POWER 64 in gear, your computer will be precisely the same as always. Each time you switch the computer off, POWER 64 deactivates. When you switch on again, you'll need to call in the POWER 64 system using the method shown below.

First you must place your POWER 64 system disk into your disk drive. You load POWER 64 just like any other Basic program you may have run. First, turn your computer and disk drive on. Then put the POWER 64 disk, face up, into the disk drive and close the door as described in your disk manual.

The cursor should be flashing on the screen of your 64 as usual. Type the command:

```
LOAD"POWER",8
```

...followed by the RETURN key. This will load in a small program that will help bring POWER 64 into your machine. While it loads, you should see the strings SEARCHING FOR POWER and LOADING appear on the screen. After a few seconds, the 64's cursor should appear again. If any of this does not happen, you should go to Appendix B, which talks about troubleshooting.

Once this program has loaded, you should type RUN and press the RETURN key. POWER 64 itself will now begin loading into your machine. When it is finished doing so, the POWER 64 system advises you that it is now enabled by printing the copyright message. Then the word "READY." will print and the cursor will flash, awaiting your next command.

This sequence may seem quite ordinary to you, but in the 64 it opens up a whole new world - the world of programmer POWER.

If the copyright statement does not print, or your computer does not print READY and flash the cursor, you have a problem. This is unusual; refer to Appendix B for troubleshooting procedures.

Now you're ready to go; the features of POWER 64 are literally at your fingertips.

Most users start up the POWER 64 system as soon as they turn on their computers, and leave it in for the entire session. If you wish to disconnect POWER 64, however, you can simply type OFF followed by a Return character: this will kill the POWER 64 features and restore you to normal Commodore 64 activity.

Keep in mind that POWER 64 is most useful when you are developing and testing new programs. Programs written using POWER 64 will still run on any similar computer, even ones not equipped with POWER 64. If you switch on with the intention of using your computer only for a program run - not testing or writing programs - you won't need to engage the POWER 64 system. You may wait and call it in later when you need its features.

Reminder

Every time you power up, remember to **POWER** up by loading from your POWER 64 disk.

3. THE POWER 64 CURSOR.

A Note about this Manual.

This guide is meant to be used along with your Commodore 64 computer. Turn on your machine and try things out as you read. POWER 64's features are quite natural, and you'll learn them best by using them.

So if you're not sitting by your computer with the power on, stop reading this manual and go there.

If you don't have a Commodore 64 computer, buy one immediately. Then put your POWER 64 disk into it and engage POWER 64 by loading and RUNning it. We will get to MOREPOWER in Appendix "A". Now read on.

Scrolling Power.

First we'll describe a dramatic feature of POWER 64 which is more than just useful. It will amaze and mystify your friends - not to mention making them green with envy until they, too, get a POWER 64 system. If you've used your 64 before, you know you can move the cursor quickly around the screen by holding down the cursor movement keys. Now, with POWER 64, your cursor will seem to reach out beyond the top and bottom of the screen!

Load any Basic program, from disk or tape; or type in a brief program on the keyboard. It doesn't need to be a working program, since we plan only to list it at this time, but it would be nice to have a program of thirty or more lines, so that we have more than a screenful. If you don't happen to have a program with you at the moment, you could type in the one from Appendix G.

List the program in the usual way with the LIST command. The first lines of the program will disappear off the top of the screen, and you'll end up with the last twenty lines or so of the program showing on the screen.

It is one of Murphy's axioms of programming that the particular line that you happen to want to look at will have vanished from the screen. A related law says that any attempt to list selectively, say LIST 200-300, will probably get the wrong lines anyway. POWER 64 to the rescue!

After you've done a LIST, the cursor will be blinking at you from the bottom of the screen. Move it up to the top using the cursor-up key and POWER 64's automatic repeat. You're moving toward the earlier lines of your program. Now - KEEP GOING. When the cursor reaches the top of the screen, the cursor-up key will cause earlier lines of your program to appear. Eventually, if you keep holding down the cursor-up key, you'll reach the start of your program. And of course, you can stop anywhere along the way.

If you've been moving up to look at earlier parts of your program, later lines will have disappeared off the bottom of the screen, of course. You can get them back easily: just move the cursor down to the bottom of the screen with the cursor-down key, and the later lines will scroll into sight.

It's handy, it's powerful, and it's quite dramatic. Sit in front of your 64 and scroll programs up and down to your heart's content. It's rather fun and probably more educational than watching television. Try to avoid humming, "Ol' Man POWER, he jus' keeps scrollin' along... "; it's considered poor taste unless you do it very quietly.

A quick technical note: POWER 64 knows which new line to write to the screen by looking at the screen - top or bottom as the case may be - for the previous line number. This means, for example, that if you type a new line for your program, say line 125, near the bottom of the screen and then perform cursor-down, you'll get the lines following 125.

4. THE INSTANT ACTION KEYBOARD, PART 1: INSTANT KEYWORDS.

POWER 64 is so designed that a single key can do a lot of work. We'll try out the simplest "instant action" feature first - the instant keywords.

Hold down the Shift key and press the letter L on the keyboard. Surprise! The word LIST appears. Release the Shift key and press RETURN and the program you have in memory will be listed.

The keyboard is set so that a single shifted key prints an entire Basic keyword. Shifted F, for example, causes the word FOR to print; shifted N produces NEXT, and so on. Hold down the shift key and practice speed typing. You'll find an appendix with a complete list of what keywords come from each key, and you'll learn to use the common ones quite quickly. Sometimes, the CONTROL or COMMODORE KEY versions of a letter have been defined in addition to the shift version, and that's all in the table. In the meantime, explore the keyboard.

It's important to understand that the instant key action produces output to the screen only. When you type shifted L, the word LIST appears on the screen - nothing more. That's all you need, however: when you press RETURN your computer will take its instructions from the screen, and do the LIST you want. Similarly, if you're typing in a program you might enter the line number and then press shifted F to generate the keyword FOR ... keep typing to complete the line and you'll have saved a little time and effort. The FOR will be saved as part of your program line.

POWER 64 is smart enough to recognize that you sometimes want to type graphics or shifted alphabetic characters. If you enter a line like PRINT "(character)" where the character is graphic or shifted, POWER 64 will know not to change it into a keyword; you'll get the graphic you intend.

Check over the list of instant keywords in Appendix F, or experiment. You'll find keywords at your fingertips; the longer keywords will be especially convenient.

The Instant Action keywords are activated when you first fire up POWER 64. If by chance you wish to turn them off - perhaps you like to make spelling mistakes - you can easily disconnect this feature. Type in the command:

SEL K-

SEL stands for Select - we'll meet this command again; K stands for Instant Keyboard; and - stands for turn off.

Test your reasoning powers: If SEL K- is the command that turns the Instant Keyword feature off, what command would you use to turn it back on again?

If you answered:

SEL K+

...is the command to make Instant keywords return into being, you're absolutely right, and you may paste a small gold star at the top of this page. Oh, and by the way, if this happens to be somebody else's POWER 64 manual you are reading, and there is no gold star pasted at the top, you'll know either that the rightful owner got this question wrong, or he was out of gold stars at the time. Either way, don't feel too superior; after all, he has a POWER 64 system and you don't. Yet.

Instant keywords are just a small part of Instant keyboard features. Later, you'll learn how to define your own keywords and phrases and call them up with a single key. But that comes later.

5. FIRST POWER 64 COMMANDS: AUTO, DEL, and RENUM.

In this chapter, we'll talk about several commands for helping you with Basic programming.

AUTO gives automatic line numbering, which helps keep track of program line numbers and saves you some typing. DEL stands for Delete; it will help you get rid of a group of unwanted lines. And RENUM is the super deluxe Basic line renumbering package that comes with your POWER 64 system.

AUTO - Automatic Line Numbering.

If you type in the phrase:

AUTO 100,10

...the computer will prompt you with line numbers so that you can input Basic statements; it will start at line 100 and each subsequent prompted line will be 10 higher than the last. You may pick any starting line number you like; the increment (the second value) should not be greater than 255.

You may continue typing Basic program lines and the computer will keep on prompting for new ones. Eventually most programmers finish their coding and don't want to type any more lines. The computer will keep prompting for the next line, however. If you are the cooperative type you may want to be helpful and think up some more lines to type in; but the computer just keeps prompting for more, and eventually even the most helpful of programmers will want to stop somewhere. To stop automatic line prompting, type RETURN. A line number followed by RETURN usually deletes the line in question; but this won't happen with automatic line numbering. If your last prompted line number was 450 and you press RETURN, automatic line prompting will cease but you will not remove line 450 from the program, if it's there.

If you like, you may type in just the starting line number: for example, AUTO 200 will start numbering from 200.. The step size will be whatever you used last, or a value of 10 if you haven't previously set the step.

When you type AUTO without any numbers, a handy thing happens. POWER 64 will automatically find the end of the Basic program and start adding new lines there. For example, if your program went to line 640, commanding AUTO could produce a prompt for new line 650. Very convenient. Try it.

DEL - Delete a Range of Lines.

If you have a group of lines you wish to remove from a Basic program, use the DEL command to delete them. Type the line number range behind the DEL:

DEL 500-600

...removes lines 500 to 600, inclusive. You may use the same type of line range indication as with a LIST command: DEL 900- removes all lines from 900 up, and DEL -125 removes all lines up to 125. If you really must know, DEL - removes all lines; but the command NEW would do the job just as well.

RENUM - Renumber Basic Lines.

This is a deluxe renumbering package that takes up to four (count 'em) parameters. It's easiest to show how they work with examples:

RENUM by itself renumbers the whole program in steps of 10, starting at line 10.

RENUM 5 renumbers the whole program in steps of 5, starting at line 5.

RENUM 5,30 renumbers the whole program in steps of 5, starting at line 30.

RENUM 5,30,100-200 renumbers lines 100 to 200 of the original program into a group of lines in steps of 5, starting at line 30 .. if possible. If the new numbers won't fit properly within the program, a warning message will be issued and renumbering will not take place.

Renumbering will correct all line references within the program to fit the new line numbers. Everything is fixed up nicely.

Using RENUM for Renumbering.

If you have used other Basic renumbering systems, you may be puzzled at first by the seemingly unusual way that POWER 64 asks you to input the numbers: increment first, then starting line number, and finally the range to be renumbered. You'll get used to it quickly and will soon discover that it makes sense. You need put in only the numbers you want; the missing values will be handled naturally and correctly.

A few examples will show how neatly POWER 64 does renumbering jobs.

a) G. Whizz wants his program to be neatly numbered. He types RENUM and the program is renumbered starting at line 10 in steps of 10.

b) L. E. Fant wants her program to be numbered very compactly. She types RENUM 1 and her program now starts at line 1 and goes in steps of 1.

c) P. Nutt is working on his program and thinks he'll need to add new coding at the beginning and lines here and there within the program. He types RENUM 10,200 so that his program starts at 200 and goes in steps of 10.

d) X. Pleative wants to add 30 lines of data statements between lines 360 and 370 of his existing program. He types RENUM 10,900,370- which causes lines from 370 to move up to 900 and beyond. In the meantime, lines below 370 will have their numbers unchanged so that there will be a gap in the line numbers, leaving plenty of room for the new lines.

e) Y. Pout likes her subroutines to start at line 1000 for neatness; at the moment, they start at line 743. She types RENUM 10,1000,743- and it's done.

f) Q. Cumber has coding from 500 to almost 700 which he wants to make neater. There's room for a step size of 5 between lines, so he types RENUM 5,500,500-699 and trims it up. It didn't matter that no line was numbered 699: POWER 64 caught all the lines involved.

6. FINDING AND CHANGING TEXT.

It's easy to find or change parts of your program with the POWER 64 system. For example, suppose you want to find every subroutine call in your program. Just type:

```
/GOSUB/
```

...and the POWER 64 system will list every line containing the command GOSUB. Easy, isn't it?

Changing your program isn't much harder. If you have a variable called T2, and you decide to change it to X2 everywhere in the program, you would type:

```
S/T2/X2/
```

...and the job is done. You'll also get a listing of every line that has been changed. By the way, the S stands for "Substitute".

Either of the above commands can be performed on the entire program, or on a specific range of lines. You could type, for example:

```
/GOSUB/ 100-200
S/T2/X2/ 200-
```

There's much more to be learned about effective use of the find (/) and substitue (S/) commands; but even the above simple uses are very powerful and easy to use. Experiment a little. You'll find nice touches; for example, when a pattern is found more than once in a program line, the line will print only once.

Search Patterns.

Searching for the pattern GOSUB will give you all the occurences of the GOSUB instruction in your program. But if you search for the letter G, you won't get GOSUB - or GOTO, for that matter.

This is because Basic keywords are stored within your computer in an unusual way. Commands like PRINT, GOSUB, and FOR, and other words like RND, STEP and THEN are stored as "tokens" - not as alphabetic characters. When you ask the POWER 64 system to search for GOSUB, it knows to look for the GOSUB token. When you ask to search for the letter G, it will search for G as requested, and will ignore the GOSUB token.

This is quite handy. When you are searching for a variable called G, you really don't want to see all the GOSUBs, GOTOs, GETs, or LOGs just because they happen to be spelled using the letter G. And if you wanted to rename variable G and call it W, you wouldn't appreciate your program GETs being changed to WETs. It won't happen, of course.

Special Search Characters.

A few characters have been set aside for you to allow special, super-powerful searches to take place. Because these characters have special purposes, they are called "meta-characters". You can turn off the meta-characters, if you wish, and then they will behave like ordinary keyboard characters.

The meta-characters used in the POWER 64 system are:

. - to match any character
* - to match any number of characters on the same line
| - to match end of Basic line.

The apostrophe, or single-quote character (') also has a special meaning: it means, "the next character is not a meta-character". We'll give an example of this later.

Let's put these new characters to work. I want to search for things like F1= or F2= or F3=, etc. My search command will be:

/F.=/

...which we can read as F, "anything", "equals".

Here's another example. Suppose I want to find all lines where FOR and NEXT are on the same line, in that order. I would type:

/FOR*NEXT/

It doesn't matter how many characters are between the FOR and the NEXT... POWER 64 will find and report them if they are on the same line. In the same way, you could search for IF ... THEN or any combination.

A somewhat rarer situation: suppose I wanted to find a line ending with a PRINT command. I could type /PRINT]/ to do the search. The end-of-line metacharacter, (]) is more often seen as part of a substitue command, but it does work here.

Suppose I wanted to find all multiplication within a program. I would want to search for the asterisk (*) character .. but that's a meta-character. No problem: we type /'*/ and the apostrophe signals that the asterisk is a search character, not a meta-character. Incidentally, to search for an apostrophe you would type two of them: search for MURPHY'S with /MURPHY''S/.

You'll also find the apostrophe useful if you want to search for a / symbol or for a right square bracket.

We've seen that you can turn off a meta-character by using the apostrophe. If you don't want to use any meta-characters at all - that is, if you want those characters to behave like any other character - you can turn them all off except the apostrophe with the command:

SEL P-

...and, of course, restore them with the SEL P+ command. Beginners might find it easier to disable meta-characters at first.

Single Line Searching or Substitution.

So far, a search or substitution has given us all lines containing the search value. What if we want to look at the lines one at a time?

The trick is in the / symbol at the end of your command. If it's there, you will search all lines; or if you specify a line range you will search all lines within the range. Leave the ending / symbol off, and you'll get the next occurence only of the string you're searching for. For example, the command: /T1/ gives you all lines containing T1, and the command: /T1 gives you the next line only.

In the same way, the command S/DOG/CAT/ changes DOG to CAT everywhere it appears in your program; but the command S/DOG/CAT changes DOG to CAT only in the next line that DOG occurs. It's very neat, by the way: the line is completely processed, so that if there are several DOGs on the line, they will all become CATs.

If you are searching for a line at a time - that is, you don't put the / symbol at the end of your command - POWER 64 will not stop its search when it reaches the end of your Basic program. Instead, it will go back to the beginning and start the search over unless, of course, there's nothing to find.

Here's a very useful feature: if you want to use the same search string as before, you don't need to type it in again! Suppose we are searching for CASH: the first time we type /CASH and POWER 64 reports the first line containing this string. If we want to look for more CASH we just type / and POWER 64 will find it for us. In the same way, we could perform substitutions on one line at a time: the command S/WATER/WINE changes WATER to WINE on the first line that WATER is found; subsequently, the command S//WINE does the job for each following line. Note that we don't need to repeat the search string, WATER.

Single line searching lets you do something very handy: you can change whatever you're searching for part way through the program. For example, you can search your program for the FOR keyword; after you've found it, you can look for the NEXT that matches it. The general procedure is /FOR followed by /NEXT and if the NEXT you find isn't the matching one, you can give / and keep looking. Keep in mind that changing the search string doesn't affect the program location at which you're searching; in the example above, POWER 64 find the FOR command and then searches for NEXT in the lines following.

Advanced Techniques, Curiosities, and No-Nos.

Don't forget the power of the meta-characters. When you combine them, you can do surprising things. Try your hand at seeing how the following work:

```
/DOG]/      ... DOG, only if it is at the end of a line
/DOG*]/     ... DOG and everything following it on the line
/.*/        ... the whole line
```

Using these search strings, we can create:

```
S/DOG*I/KENNEL/ 300 ... replace DOG and everything
                        behind it with KENNEL in line 300
```

Question: can you explain why /./ behaves exactly like a LIST command?

If you want to search for a / symbol, use the apostrophe: e.g., /'// will do the job. It looks puzzling at first glance, but it's not hard to see how it works.

Searching for strings that contain keywords can be tricky. If you are looking for MONEY, you may not find it ... until you realize that ON is a keyword. Similarly, trying to find a WORD can be hard because the keyword OR is in there. The easiest way around this kind of problem is to search for the non-keyword parts: NEY or RD will usually do the job well. If you're a purist, however, you can do the job by breaking up the keywords with an apostrophe: try /MO'NEY/ or /WO'RD/ ... the apostrophes are ignored but they break up the keywords and give you the result you want.

Certain cursor movement symbols can occasionally get confused with certain Basic keywords. Example: searching a program for LOAD will cause the clear-screen cursor character (reverse-heart or reverse-S) to be found. If you must search for some of the troublemakers, do the job a line at a time, so that you can manually sort out the garbage from the trash (or vice versa).

You can't add to the end of a line by using] alone as the search string. And be very careful about changing a line; if you succeed in changing the line into nothing at all, Basic will behave rather oddly. So don't type things like S/./ unless you like weird programs. By the same token (no pun intended) don't type in S/ by itself. Un-nice things tend to happen when you do this.

If your Basic program is large, replacement activities can take some time; the whole program may need to be moved up or down to make room or to close up memory space. Be patient. The STOP key will abort the replacement job, of course, if you absolutely can't stand the wait.

7. THE INSTANT ACTION KEYBOARD, PART 2: INSTANT PHRASES.

Earlier we learned how touching a key could produce an entire Basic keyword. Now we'll expand that capability so that a key will generate any text you like.

Let's suppose that there's a phrase that you type in frequently. It could be anything: a Direct command, a group of program statements that you use a lot, or even just a REM statement or a name. Whatever it is, we'll set it up so that it can be called to the screen instantly, with the touch of a key.

All that's needed is to define what we want printed and which key will produce it. We do this with a special type of Basic statement.

Implementing an Instant Phrase.

Let's take a simple example. Suppose you want your name to appear every time you press a shifted N key (N for Name, of course). All you need to do is to type in a special REM (remark) statement at the beginning of your program. It would look like this:

```
1 rem"N=Albert Gonk
```

...and your name, Albert Gonk, will print as soon as you touch the shifted N key. If by any chance your name is not Albert Gonk, you may wish to change the line so that it contains your own name.

This is so easy that we can't resist putting in other lines to help along our program. For example:

```
2 rem"G=gosub 5000     (a favorite subroutine)
3 rem"D=for j=1 to 1000 : next j   (pause 1 second)
4 rem"F=for x=1 to          (start a loop)
```

...and you can use your imagination to think up dozens of others.

The preceding examples show instant phrases that might be used in programs. Don't forget that you may also generate instant phrases for use as direct commands ... or even as screen controls. Some examples:

```
5 rem"S=save "@0:program",8
```

...will allow you to save a program you are working on quickly and easily by typing shifted S followed by Return. You'll find this very handy as you'll never mistype a filename again.

```
6 rem"P=(down)(down)(down)
```

...the Cursor Down key is pressed to type in this line; you'll get the usual reverse-Q character printed. Then whenever you press Shifted P the cursor will zip down three lines from its previous position.

The important thing to keep in mind is that whatever you have placed behind the equals sign will be printed whenever you type the appropriate shifted key. If you have placed cursor movements there, that's what you'll get. If you have placed Basic coding there, or direct commands, or even nonsense, it will be neatly repeated back to you when the right key is pressed. You can even define POWER 64 commands that you think you will need - RENUM for renumbering, for example - and out it they will come when you call for them. For example:

```
7 rem"R=renum 10,100
8 rem"Z=/gosub/
```

Note that the REM lines don't harm the program operation: If you say RUN, Basic will skip over them as it does to all lines that start with Remarks. Eventually when you have finished writing and checking out

your program you'll probably want to remove the instant phrase REM statements. They have served their purpose and won't be needed in the running program.

Experiment. Instant phrases are quite easy to set up; a little playing around will convince you of their usefulness... even if your name isn't Albert Gonk.

Be sure to place your phrase-defining lines - the ones with the REM - at the beginning of your program. That's the only place that they will work. Be careful that you don't put a space after the 'REM' or before the '='.

Challenge for artistic programmers: write an instant phrase which draws a picture on the screen - a square, a face, or a car. Now wander around the screen with the power cursor and with a stroke of the instant key, you can draw the picture many times in many places on the screen. Picasso POWER!

A Special Case.

You may have noted that the foregoing examples have appeared in upper and lower case characters. This is done to make the examples defining shifted keys to be more readable. POWER 64 lets you define any key, including normal keys, shifted keys, control keys, "Commodore" keys and even the special fat keys on the right hand side of the Commodore 64. It is best not to define the normal letters because once you redefine them, they can become rather hard to type. After all, no matter how much you like your name, you can get tired of getting Albert Gonk every time you type an "n". The examples thus show capital letters being defined.

You will want to put your 64 into lower case mode to go over these examples. You can do this by holding down the SHIFT key and the Commodore symbol key at the same time. Doing this again will restore upper case and graphics character mode.

Turning off the Instant Phrase feature.

If you don't want to use instant phrases the easiest thing do do is not to define any of them.

In special situations you might want to disconnect the instant phrase feature after you have defined a few instant phrases. You may do this with:

SEL R-

...and the feature will be disabled until you reinstate it with SEL R+.

Note that instant keywords and instant phrases are independent. Each feature may be turned on or off separately. Keys will drop their instant keyword identities when they are used for instant phrases.

Tricky bits.

Clever readers will have noticed that the quotation mark behind the REM statement allows cursor movements to be included in instant phrases. Smart Aleck readers will point out that a second quotation mark will disable the cursor movmement feature, so that you can't generate an instant phrase like: 1 rem"H=print "(home)" since the second quotation mark turns off the programmed cursor feature. True, but picky.

You and I will probably never need to do it, but there's always the person who wants to have a programmed phrase which ends in one or more space characters. It seems hard to do, since spaces are automatically stripped from the end of the Basic REM line at the time it is entered.

The easy ways are: (I) use the Shifted Space character, which won't be stripped away; or, (II) use a cursor-right character instead. The harder way is to do a little clever work with the Replace feature; I'll leave this as an exercise for the advanced reader.

Here are some examples for you to work with. They may look funny (especially when you list them), but they'll do the job:

```
rem"H=print [RVS]" S
rem"H=print [RVS]" S "[RVS OFF]"hello
rem"H=print [RVS]" S "[RVS OFF]"hello"[RVS]"qqq
```

When you list these examples keep in mind what was mentioned before about the values of SHIFTed characters and Basic keyword tokens.

Calling Instant phrases from INPUT statements.

POWER 64 also provides you the ability to call instant phrases as a response to an INPUT during a program run.

If you want to do this, you must enable the feature with SEL I+.

In most cases, you'll want to use this special input feature as a debugging tool. While you're checking out a program, you can generate "canned" inputs and speed the whole procedure.

It will work in your final program after debugging is complete, of course. If you choose to use instant phrase input in this way remember that the program will need to run on a system equipped with POWER 64.

8. DEBUGGING: WHY, DUMP, and TRACE.

A computer always does exactly what you tell it to do... not what you meant to tell it to do. The mistakes you made are called bugs; getting rid of the mistakes is called debugging. The POWER 64 system has a number of aids to help you with this job.

The most important debugging tool of all is you - and especially your attitude. Most programmers want their programs to be perfect. But some want perfection so badly they turn a blind eye to program faults. Don't be like that. Seek out errors; go hunting for them; lay traps for them. It's better to find problems early, while the program is fresh in your mind, than to have it pointed out to you months - or years - later. While you're doing the debugging, POWER 64 can't do your thinking for you but it can give you a lot of help.

WHY?

The simplest type of program error is signalled to you because the computer stops. The computer says something smug like ?SUBSCRIPT OUT OF BOUNDS or ?REDIM'D ARRAY or the famous ?SYNTAX ERROR. Although there's an attempt to give you some help, it can still be difficult to spot the trouble, especially in a long Basic line with multiple statements.

Type WHY and the POWER 64 system will give you some help. It will list the error line, and indicate where the computer gave up by highlighting that place in the line. Remember that the computer doesn't know what you have in mind: if you say something silly like J="3" it has no way of knowing whether you meant to say J$ or not; but it will know that there is trouble by the time it reaches the first quotation mark, and will stop there.

Try writing a program with a lot of little silly mistakes. Run it, and when the computer stops on an error, type WHY. You'll quickly see how it works.

If you stop a program in mid-run by using the RUN/STOP key, you can ask to see exactly where the program stopped: just ask WHY. POWER 64 doesn't give you what I think is the logical reply to WHY in this case: "Because you pressed the RUN/STOP key, turkey!" Instead, it politely furnishes you with details of exactly where the program stopped. Very courteous. Very handy.

WHY doesn't work on Direct statements, and it doesn't make much sense to say WHY if you didn't get an error message. You'll still be told where Basic was working the last time a program ran; but it won't help. And if you have the clever sort of mind that would clear a program by typing NEW and *then* type WHY, I have only one comment: Why?

DUMP.

When a program has stopped running - normally, abnormally, or via the RUN/STOP key - you may ask to see all its variables by commanding DUMP.

Out will come all the variables, in the order that they were first used. Any variables that haven't been used yet will not be shown. You may slow the listing of variables in the usual way, holding down the CONTROL key. You may stop the variable listing at any time with the RUN/STOP key. Three types of variable will be shown: the usual floating-point varible, such as X; integer variables, such as X%; and string variables, such as X$. In addition to variables, the DUMP command indicates any functions (FN) that have been defined.

Variables are carefully printed so that you can move the cursor up and redefine any variable's value by typing over the display value and pressing RETURN. This can be useful if you have stopped a program with

the RUN/STOP key and plan to continue the run with the CONT command. Sometimes a changed variable value can make it possible to investigate a program's behaviour, or try correcting a problem on the spot.

DUMP prints only variables and function definitions: it does not print arrays.

Careful analysis of your variables is a very useful tool for checking that a program is operating properly. Help unload program bugs ... with a DUMP.

The Versatile TRACE Command.

The Trace feature of POWER 64 lets you watch a Basic program as it is running. It's an excellent way to check out a program: you can watch what the program is doing, and spot where it gets into trouble. The Trace feature is also a marvellous teaching aid; learners can see exactly how instructions are executed.

TRACE is a very rich command; there are many ways you can use it. We'll introduce TRACE with a few examples, but first let's take a quick overview. Trace allows you to:

--step the program through one instruction at a time;
--see which line is executing;
--optionally see a listing of the line;
--if desired, see each variable as it is calculated;
--choose to watch this information in-line, or on a separate strip at the top of the screen.

The Trace feature is engaged by giving the TRACE command, followed by one or more option characters. Tracing is disabled with the command TRACE STOP. The option characters are:

(none) - Full trace: line number, line listing, variable;
↑ - Display trace data at top of screen, rather than in-line;
LIST - Line number and line listing only
- Line number only
STOP - Kill trace feature

Option STOP must be used alone. Option ↑ may be used alone, or with option LIST or #. Now let's try a few simple examples.

Simple Fast Trace.

Load any Basic program into your computer, and then type TRACE ↑#. This is one of the most popular means of quick tracing; it gives line numbers at the top of the screen. Now type RUN, and instantly ... nothing seems to happen.

That's because the Trace feature is waiting for you to ask for the next step. You may choose three types of stepping. They are:

"f7" key: perform one Basic statement and wait;

"f5" key: perform Basic statements as long as the key is held down;

Commodore key: perform Basic statements without Tracing as long as the key is held down.

We want to see our Trace happen slowly, so we'll use the "f7" key. This is the fat key on the bottom right of your Commodore 64. Press it, and you'll see the line number of the first statement of the program appear at the top of the screen. Press it again and a type of "sideways scrolling" takes place: you'll get the line numbers of the first two statements on the screen. These may be the same, if the first line contains more than one statement.

You're on your own now. Try holding down the Space key for a few moments, and watch the line numbers rush by. An INPUT statement will stop Tracing momentarily, of course, to allow you to supply the desired value.

Listing the Line.

Sometimes it is useful to see the line that is being executed. You have another Trace option that permits this: LIST. If you are still Tracing a program, break it with the RUN/STOP key; you may restart it later with CONT after you have changed the Trace options. Now type TRACE LIST ↑ to trace with Listing at the top of the screen. Continue your program or type RUN. Once again, you may use the "f7" key for single stepping. Now, you get a single line number at the top of the screen ... plus the line itself.

If you have lines of coding that contain multiple statements, watch carefully when they execute; you'll see an interesting thing happen. You won't always see the whole line; only the part from where you're executing a command to the end. This means that a line may first list completely as 30 X=5:J=2:M=4. Then, after the X=5 has been executed, it will be listed as 30 J=2:M=4. Finally, it will list one last time as 30 M=4. This is very handy, since you can always see what part of the line is being executed: it's always the first statement listed.

The Big Trace.

Now let's go for the whole thing: line number, Basic line listing, and variable. Stop the current program if necessary. Now enter TRACE ↑ (Trace at top). Start the program again, and use the "f7" key to single step. We see the line number; the line listing; and a variable value.

The variable value is often useful, but there is one case where you might find it a bit confusing. This could be because you may not understand about a special type of computer number called a Boolean value. This is a special number the computer uses to indicate True or False. The value 0 means False and the value -1 means true. Now: during an IF statement the POWER 64 Trace reports this value to you so that you can tell whether or not the IF condition is true. When your program line says IF J=9 .., POWER 64 might display a value of -1. This does not mean that J is equal to minus one; it means that the condition J=9 is True. Similarly, if POWER 64 displayed a value of 0, we know that J=9 is false. If you get the hang of Boolean values, by the way, they can be quite handy in your Basic programs. It's quite permissible to say B=(J=9) within a Basic program. B will be set to 0 or -1, and later you can code IF B THEN... and get the proper action depending on whether or not B represents True or False.

Experiment with the Trace features. You'll find that they are not hard to understand, and become more and more useful as you gain experience.

Tracing In-Line.

So far, we've been Tracing our program with lines at the top of the screen. We don't need to do it that way, of course. If we wish, we may allow the Trace information to go to our normal output location. If the Basic program produces output, Trace information will be mixed in, but that's usually not confusing.

Tracing at the top of the screen is more popular, but there are reasons why it's attractive to do it in-line at times. First, you get more Trace information on the screen: you have the whole screen instead of the top three lines, and new lines don't immediately wipe out the old Trace information. Secondly, you can see how your program lines interrelate with the program output data. Thirdly, you can usually send your output - Trace and all - to the printer and examine the workings of the program at your leisure.

How do you Trace in line? Simple: just don't say ↑ in your options. You would say: TRACE # for line numbers only; TRACE LIST for line listings; and just plain TRACE for everything including variables.

Once again, don't forget to use your "f7" key to single step, your "f5" key to run a continuous Trace, and the Commodore key to run without Tracing.

Try the various Trace modes on a sample program. You'll have no trouble seeing how it all works, once you get used to those kinky variables.

Trace Trivia.

To send Trace information to the printer (along with the Basic program output) you should use the usual command series of OPEN 4,4:CMD4. You may put this in your program coding or type it as a Direct statement after the program has started running (Break the program, enter the command, and CONT to continue). Remember that INPUT, GET, and PRINT# statements will cancel the CMD status, and you may need once again to break the program and do another CMD 4 followed by CONT in order to reinstate everything. It's clumsy, but it's workable. If you choose to Trace at the top of screen (TRACE ↑) the Trace output will never go the the printer.

When coding a Basic program, you're often allowed to say NEXT alone rather than NEXT J or NEXT X. It's good practice to put the variable name in anyway: it can help avoid some subtle programming bugs. And if that doesn't convince you, try this: NEXT J will print the value of variable J for you during a full TRACE; NEXT alone is likely to print nonsense. So how are you going to code your program NEXT statements? With the variable name, I hope.

You may turn the Trace feature on or off during a program run by the simple method of stopping the program, giving the appropriate TRACE command, and then continuing with CONT. It is also possible to have the program itself kick Trace in and out of gear. There's a special SYS command to allow you to do this.

Invoking Trace from a Program.

Finding the address for this SYS command can be tricky, since your version of POWER 64 could be relocated anywhere in memory. Fortunately, the POWER 64 startup program saves the first address of POWER 64 at location 704 in memory as what is known in Black Magic circles as a JuMP vector. Those who have been initiated into the mysterious powers of hex - known more familiarly as machine language - can read the deatails in Appendix D. If not, don't worry; just read on. The important thing for you to know as a mortal is that the address of the first byte in the POWER 64 program can be found with the Basic expression PEEK(705)+256*PEEK(706). The special Trace address is 3 bytes below this.

The command SYSPEEK(705)+256*PEEK(706)-3, - don't forget the comma - within a program will allow you to turn Tracing on or off or change the Trace mode at any time. You would put the Trace options you want behind the comma; a full trace would end with the comma. So 501 SYSPEEK705)+256*PEEK(706)-3,#↑ would turn on line number tracing at the top of the screen when line 501 is executed, and 571

SYSPEEK(705)+256*PEEK(706)-3,STOP will stop the Trace feature when line 571 is executed.

GET statements can be a little tricky to Trace. Provision to cope with this is made in the POWER 64 system. The "f5" key, which is used for fast Tracing, will not normally be seen as input to a GET statement. When the program reaches the GET statement, let go of the "f5" key and type in whatever characters would be appropriate at that point in the program. When you hold down the "f5" key once again, the characters (which have been patiently waiting in the keyboard buffer) will be received by the GET statement.

Summary.

The debugging commands are rich in power and versatility. Get used to the Trace features; they will save you hours of work. Like all POWER 64 features, these are easy and natural once you get used to them.

9. ADVANCED COMMANDS: FIX, PTR, EXEC, SEL, TEST, BACK.

The commands that we will discuss in this chapter are "advanced". These commands solve advanced problems, the type that beginning programmers won't need to worry about.

The FIX command trims up a Basic program, resetting certain essential "pointers" used by Basic. You'll need to FIX a Basic program if you have wrecked these pointers by loading in a Machine Language program. PTR has a similar use.

The EXEC commmand has a variety of uses: the most popular one is to merge two Basic programs together into a single program. When you're writing a program, you'll find it handy for slipping in a few Basic subroutines you previously wrote, saving the trouble of typing them in again. Or you can merge a set of DATA statements into each of a group of related programs so as to implement new information such as a revised price list. (You'll find this can also be done easily by the MERGE command in the MOREPOWER program included on the POWER 64 disk.)

We'll briefly mention the SEL command again here. You've met it during our "instant key" discussions, and you'll meet it again. To keep things neat and to complete our description of POWER 64 commands we'll cover it in this chapter.

The TEST and BACK commands allow you to work with a second Basic program while keeping your main one in memory.
Skip the parts of the chapter you don't need for now. The advanced features will be there when you need them.

FIX and PTR.

Basic usually takes care of itself. The Basic interpreter uses several internal "pointers" to keep track of where things are, and these

pointers usually behave properly as they should.

However, if you LOAD a machine language program to an unusual part of memory, the pointers can get muddled and give you trouble. The biggest problem is with a pointer called "Start-of-Variables"; after a Basic LOAD, it ends up pointing just beyond the last byte loaded. Sometimes that's all right; but other times it will cause the computer to give you instant ?OUT OF MEMORY, and in one horrible case it can cause your program to destroy itself when you say RUN.

Some Machine Language Monsters.

If you work with Machine Language, this is worth looking at more closely. The circumstances that give trouble are usually these: you have a Basic program already in memory, and you wish to load a Machine Language routine to some memory location where it will leave your Basic program untouched. The most popular places for this are: at the top of Basic memory space; and in a Cassette buffer. Whichever location you use, you'll set the Start-of-Variables pointer to an awkward value when you bring the ML (Machine Language) program in with a Basic LOAD.

If your ML program goes to the top of Basic memory space, the pointer will end up looking at the top of memory. This means that there will be no space for variables, and you'll get ?OUT OF MEMORY the moment you try to do anything useful. These are Count Dracula program loads: they have drained away the program's life blood - its variable space.

On the other hand, if your program goes into a cassette buffer, the Start-of-Variables pointer will be set below your Basic program, and your variables will be written down there. Not only is this an unusual place to put variables, but as you write more and more variables they will work their way up and eventually will start storing over the Basic program itself. These are Frankenstein variables, which eventually destroy their creator - the Basic program.

FIX fixes all these problems. It restores the pointers to their normal locations in relationship to the Basic program. Technical tyros will want to know that FIX searches memory for the three zeros that signal the end of Basic, and then parks the pointer behind them. Non-technical duffers may think of FIX as a combined silver bullet, wooden stake, and collection of enraged villagers.

One last comment for advanced users of FIX. In the special case where a combined Basic/ML program has been created with the Machine Language following the Basic, FIX needs to be used carefully. In this case, the Start-of-Variables pointer is set behind the Machine Language coding. FIX will zap this pointer so that it points to the end of the Basic program. This is great for making a quick change in the Basic program (the ML won't move), but remember to put the pointer back before running or saving the program. If you forget, the variables will write all over your Machine Language program: this is known as the dreaded Curse of the Mummy's Tomb bug.

FIX also resets POWER 64 and prints the copyright message. If you find anything is going wrong with POWER 64, try the FIX command. If you reset your 64 by holding down the STOP and RESTORE keys at the same time, you'll find POWER 64 is gone, but typing FIX will bring it back to life.

The PTR command does pretty much what FIX does but it does not reset POWER 64's internal data. You'll find it handy after having loaded a machine language program from disk or tape. You should note, by the way, that on the 64, the only way you can load such a program is by giving a second number on the LOAD command after the device number. This second number is called the secondary address, and if it is not zero, it allows a program to load into memory where it was supposed to, rather than where Basic programs go.

EXEC: Execute From File.

The EXEC command allows another device to take control of the 64. The keyboard normally controls 64's activities; but with EXEC you can re-assign control to some other device.

That's powerful: so powerful, in fact, that it's hard to grasp all the possibilities that the EXEC command opens up. We'll stay with a few simple examples here; even those open up remarkable new possibilities.

The most direct application of the EXEC command is to transfer control of your 64 to a sequential file on disk or tape. The procedure is: first, open the file and then transfer command with EXEC.

The sequential file used for command (we'll call it the command file) may do almost anything that can be done from the keyboard. It might LIST, set a variable, LOAD a program, or even change a line of Basic. The POWER 64 system prints the commands on the screen of your computer so that you can see what's going on. Control will return to the keyboard when the end of the file is reached; you may also press the RUN/STOP key to regain control.

Merging Basic Programs.

It's very useful to be able to merge two programs together. The most common uses of a merge are to add DATA statements to one or several programs, and to include subroutines. There are many other possible uses, however: for example, you can bring in a preset group of instant phrases to use with the POWER 64 system.

Keep in mind that this is a true merge - the new lines interleave with the old, and duplicate line numbers cause the old program line to be completely replaced with the new. It's as if you typed in the new lines at the keyboard to add to or change your program. That's what the EXEC

command is all about: making the file behave as if it were typing at the keyboard.

You should note that if you have loaded in the MOREPOWER package described later in this manual, you can do a merge directly from the disk with the MERGE command. If you want to merge from tape, or don't have MOREPOWER in: the following method should be used.

Here's the general procedure: we write a sequential file containing program lines that we will want to merge into other programs. Once the file is written it may be used over and over again to merge the desired lines into some other program.

Writing the Merge File.

We'll call the sequential file a "merge file". Here's how we write it - the examples show disk files but you can easily adapt it to cassette tape if you wish. To create a merge file we first load the program lines that we will want to merge into the computer. Then we write them out with a statement like:

```
OPEN 1,8,3,"0:MERGFIL,S,W" : CMD 1 : LIST
```

Be sure to type all three Basic commands on a single line. When disk activity stops tidy everything up with:

```
PRINT#1 : CLOSE 1
```

... and our file is written.

There are several variations that you can use when writing the file. If you are using tape you'll type OPEN 1,1,1,"MERGFIL" at the beginning. You can of course choose any file name that is suitable. And finally, you don't need to send the whole program in memory out to the merge

file: you can choose to send a selected portion such as a subroutine, with something like LIST 500-600.

You may have noticed that we have written our merge file without POWER 64's help. We have used standard Basic commands throughout. POWER 64 comes into play when we do the actual merge itself.

Once the merge file is written we may do a program merge right away. Alternatively, we can use the merge file at a later date - perhaps months later.

Merging Programs.

When you are ready to perform the merge, load the second program - the one you want to mix with the merge file program. Now POWER 64 will do the job simply and easily for us. Type:

```
OPEN 1,8,3,"MERGFIL"
```

Now we're ready to go. Just type EXEC 1 and POWER 64 will do the rest. You'll see the lines coming in from the merge file - they will be displayed on the screen. As they are displayed they are also inserted into the program in memory.

Everything happens quite fast and the merge will be complete very quickly. There may be an error statement at the end of the merge; don't worry about it. For those people who will worry about it anyway, here's a hint: the error notice is caused because the word "READY." is on the file, which causes the computer to try to execute a READ statement. Which proves the old proverb: computers are dumber than people but smarter than programmers.

If anything went wrong with the merge you should be neat and tidy up with CLOSE 1 which puts the merge file to sleep until you need it again.

The computer and the disk unit will appreciate your efforts to keep the files clean and orderly.

A Small Step for a Commodore 64...

You may not have noticed, but an important new concept has been introduced here. We've broken the barrier between programs and data files. Now we can change programs to data and analyze them ... or invent a program to write data files which in turn are changed into programs. Have you ever heard of programs that write other programs before? How do you feel about your computer writing its own programs and making you obsolete?

The idea of allocating control away from the keyboard is also profound. It's like being able to put the computer on autopilot.

These are advanced ideas. Don't worry about computers taking over, however. If you hear something tiptoing around your house in the middle of the night it's probably the cat or a burglar. It isn't your Commodore computer. Unless you have a very long extension cord.

Pitfalls and Pratfalls.

When you write a MERGE file, be sure that the program lines don't LIST in such a way that they are over 80 characters long. If necessary LIST the program on the screen first and make sure it's OK. Change any lines that are too long into two program lines.

Lines that are too long produce exactly the same effect as if they were typed in at the keyboard. You know what happens when you type in over 80 characters before you press RETURN. If you don't know, try it. It doesn't work right; that's what happens.

If you are using some of the advanced features of the EXEC command remember that the sequential file may only give the same instructions that are allowed on the keyboard. This means that you can't include a direct INPUT, GET or DIM on the file... just as you can't type these statements in at the keyboard.

It may be stating the obvious, but don't try to create a program bigger than the memory space you have available. It won't work.

SEL: Select Instant Option.

We've met this command before. This is a good time to summarize its features.

SELK- turns instant keywords off; SELK+ turns them on. Instant keywords were discussed in section 4. Instant keywords are normally on until you turn them off.

SELR+ turns "REM" instant phrases and instant subroutines on; SELR- turns them off. Instant phrases were discussed in section 7; instant subroutines will be dealt with in section 10. Instant keywords, phrases and subroutines are normally on until you turn them off.

SELP+ enables the metacharacters for pattern matching; SELP- disables them. If you've forgotten about metacharacters, go back and read section 6. "Metacharacter" is a good word to have in your vocabulary. It sounds impressive when you slip it into conversation; for example, you might casually say, "The other day I metacharacter who didn't even have a POWER 64 system on his Commodore 64... ".

SELI+ turns on the instant phrase feature during the running of a Basic program, so that you can use them during an INPUT response. This can be very handy for giving "canned" input during program testing. SELI- disables this feature.

TEST: Switch to Alternate Program

The TEST command allows you to temporarily suspend work on the Basic program in memory to work on another one. No saving or loading from disk or tape is required, and when you are finished with the new program you can go back to the old one.

TEST works by telling Basic to use a different area of memory than it normally does. When you issue it, it finds the end of the program you are working with and sets up a new area immediately after it in memory. If you know a bit about how Basic uses memory, it might help you to know that the new area is in the memory that was formerly occupied by the array variables of Basic.

Let's say you're working with a Basic program and you decide you want to test something out with another Basic program. Just type TEST.

Basic will now print READY, as usual. Now, since you want to try something new, type the NEW command to make sure the new Basic work area you have created is empty. Under special circumstances described below, you will not want to type NEW, but in just about every other case you will, so type it. Don't worry about your old program, it is safe, and you will see it again.

You now have a whole new area to write a program in. You can type in lines, edit them and run them. You can also LOAD a Basic program off the disk or tape, RUN it, play with it and SAVE it. You must be careful, however, that this program is pure Basic, and doesn't have anything in it that might be affected by the different memory it is running in. Most notably, it can't be a program with any machine language, such as the program that loads POWER 64 itself. Pure Basic programs are free spirits, however, and don't care much where you put them, so you should have no trouble working with them.

You may have loaded in a program just to look at it, or created a special routine to draw a picture. Whatever it is: don't worry, your other program is safe. You can restore it by using the BACK command described below.

BACK: Restore old Program.

As the name suggests, the BACK command takes you back to the old program you had in memory. This is the program located at the normal place in memory for Basic programs, which is the memory starting at location 2048. Type BACK and your old program will appear again after you have done a TEST. Its variables and arrays will be gone however, so you should not have left anything valuable in them.

Once you go BACK, you can not normally return to any program you set up while in TEST mode. Be sure to save your program before you type BACK if it is valuable to you.

Using TEST and BACK with PAL.

There is a special product called PAL (Personal Assembly Language) which is a machine language assembler for the Commodore 64 computer. It helps you to write machine language programs once you have become a bit more sophisticated in the use of your computer. It is available from Pro-Line Software Ltd. through your local Commodore Dealer, and can also available combined in a special package with POWER 64.

The TEST and BACK commands are especially useful for testing combined Basic and Machine Language programs produced with PAL. See the section of the PAL manual on the ".BAS" pseudo-opcode for details. When TESTing a PAL produced program of this nature, you should not type NEW after using the TEST command.

10. THE INSTANT ACTION KEYBOARD, PART 3: INSTANT SUBROUTINES.

We saw in section four how a single key could call up an entire Basic keyword. In section seven we found out how to have a single key translated to a phrase of our own choice. Now we'll look at the third aspect of instant action - how to make a single key run an entire subroutine for us.

Note, that once again, we're going to use lower case for this example so that it is easier to read. You can switch back and forth between lower case and graphics on your 64 by holding down a SHIFT key and the COMMODORE symbol key at the same time.

Suppose we have a subroutine starting at line 8000 that we wish to run whenever we press the Shifted S character. We would set this up with:

```
1 rem"S<8000
```

Note that the format is similar to that of instant phrases. The equals sign has been replaced with a left-arrow character, and the phrase itself has been replaced with the line number of the subroutine.

With the above coding in place the instant subroutine is ready to go. The moment we touch the shifted S key, the subroutine at Basic line 8000 is running. The subroutine may contain any legal Basic code; it must end with a RETURN statement, of course.

We may disable the instant subroutine feature with same SEL R- command as for instant phrases. SEL R+ reconnects the feature, of course, and causes instant subroutines and instant phrases to come into action again.

What to use it for.

The uses are almost limitless. You'll want to strike a balance between your use of instant phrases and instant subroutines. If a job can be done with a direct command to Basic, it will probably be easier to call it up from an instant phrase, striking the RETURN key after the phrase appears on the screen.

When the job gets too big for a direct Basic statement - or if it needs commands like GET or INPUT which aren't allowed within direct statements - you'll want to go to instant subroutines.

Here's a list of just a few things you can do using instant subroutines:

--print arrays; search arrays for the biggest value, or a certain desired value;
--print the time of day or elapsed time on the screen;
--INPUT or GET a value from a file, for testing purposes;
--perform hexadecimal to decimal conversions, or vice versa;
--do disk operations.

Quick tricks.

There are one or two "cute" techniques that come in handy with instant subroutines. They are probably too clever for use in normal Basic programming, but can make instant subroutines even more convenient.

They have to do with opening files for input. Normally, you might want to input from devices 1 or 2 (tape) or device 8 (disk). It is possible, however, to open device 0 (the keyboard) or device 3 (the screen).

It seems curious that we would want to open keyboard or screen as an input device. After all, the INPUT statement takes input from the

keyboard/screen. Why use the peculiar OPEN to name the keyboard or screen as a device? There are a few possible reasons, chief among which are: to eliminate the prompting question mark; and to eliminate a program halt if RETURN alone is pressed.

Opening device 0 (keyboard) for input produces: no prompt; no halt if RETURN alone is pressed; and no move to the following line after INPUT. It can be a handy way to link input data to a command generated with instant subroutines. You'll see an example later in this section.

Opening device 3 (screen) for input causes a peculiar thing to happen during the INPUT statement itself. There is no prompt, and the input is taken instantly from the screen - the user is not invited to type. This means that the desired data must be on the screen before the INPUT command is given. It's possible to do some very clever things by positioning the cursor over the beginning of a number, invoking INPUT on a file set to device number 3 ... and the number will be picked from the screen for you.

Device 3 can be somewhat dangerous to use, depending on the system you have. If you try to input from an empty screen, you could be in big trouble. Before doing too many clever things, experiment; get a feel for how it works.

Some disk utilities.

If you have loaded the MOREPOWER package in your machine with its DISK command and pre-defined keys, you'll find it useful to have some disk routines tied to instant keys. If you do have the package in, these should serve as handy examples. Here are a few for you.

Disk status: to get the disk status or error code, the following subroutine will come in handy. You'll probably want to link it to a shifted S for status.

```
7000 REM GET DISK STATUS
7010 OPEN 100,8,15
7020 INPUT#100,E,E$,E1,E2
7030 CLOSE 100
7040 PRINT E;E$,E1;E2
7050 RETURN
```

You can see that the coding is quite straightforward. There is a potential problem that you'll need to watch, however. In opening and closing the disk command channel (secondary address 15), all active files on the disk unit are automatically closed. So once you call this subroutine in its present form, you won't be able to continue reading or writing any files that were underway at the time.

We use logical file number 100 in the hopes that we don't get mixed up with other files which may already by open. And you should confirm that variables E, E$, E1 and E2 don't conflict with any other variables in your program.

Getting a disk catalogue: the following subroutine will get the directory from drive zero of the disk and print it. You might want to link it to the Shifted C key as an instant subroutine.

```
8000 OPEN 100,8,0,"$0"
8010 GET#100,A$,B$
8020 GET#100,A$,B$
8030 IF ST<>0 GOTO 8100
8040 GET#100,A$,B$
8050 N$=CHR$(0)
8060 PRINT ASC(A$+N$)+ASC(B$+N$)*256;
```

```
8070 GET#100,A$
8080 IF A$="" THEN PRINT:GOTO 8020
8090 PRINT A$; : GOTO 8070
8100 CLOSE 100 : RETURN
```

Once again, we have chosen logical file 100 for our input, in the hope that it desn't conflict with anything else. Line 8000 opens the directory file; we could substitute "$1" if we wanted to read the drive 1 directory. Line 8010 throws away the first two bytes from the directory: they will not be needed.

Lines 8020 to 8090 input a line of the directory. The first two bytes are thrown away (line 8020); the next two bytes are the number of blocks in binary, and we must do a conversion to decimal before we print (line 8060). The rest of the directory line is pure ASCII, and we may simply read it and print it (lines 8070 to 8090. In the meantime, line 8080 spots the end of a directory line and takes us back to 8020 where we deal with the next directory entry.

Care should be taken as always to ensure that variable names do not conflict with those in the rest of your program.

Sending a command to disk: You might choose a Shifted D character to invoke this procedure.

```
9000 OPEN 100,0 : PRINT "DISK:";
9010 INPUT#100,DI$ : PRINT
9020 CLOSE 100
9030 OPEN 100,8,15
9040 PRINT#100,DI$
9050 CLOSE100
9060 RETURN
```

We use the trick of opening the keyboard (device 0) as a file. A prompt of DISK: is printed, and then we input the desired command string. Out it goes to the disk, and the job's done.

Remember that opening and closing the command channel on disk (secondary address 15) causes all other files on the disk unit to be closed.

Disk Wrap-Up.

The above sample routines will be useful if you don't have the newer Basic which contains disk commands. They work on all Commdore machines, of course. If you have the MOREPOWER package in, you might still like to try them as an exercise; normally, you'd use commands such as DISK, ERR and LIST"$" to do the functions you want.

Summary.

You can call in any of a whole galaxy of subroutines with the touch of a key. It's handy and it's powerful.

There's really no difference between a subroutine and a program. If you like, you may use the Instant subroutine system to select any one of a group of programs as desired. So long as you have room for them all they can co-exist peacefully, sharing the same memory space and using common variables. You might select a read routine to bring in data from a file; then an input routine to add to the data; then a sort routine to sort it all; then a write routine to put the new data to the disk or to a printer ... the possibilities are endless.

Practice a little. You can make some very powerful systems once you get the knack of Instant action subroutines.

11. ADVANCED POWER 64 TECHNIQUES.

It probably goes without saying that you can and should combine various features of the POWER 64 system to work together.

Use instant phrases to call up other POWER 64 features such as renumbering. During development of a program, it's a good idea to SAVE at frequent intervals; you may use instant keywords or instant phrases to do this.

The remainder of this chapter will deal in a conceptual way with advanced features. Most of these are machine langage oriented, so that you may not be able to tap these capabilities if you don't have expertise in that area. Read through anyway: it's always useful to know what's in there so that you know where the system can be taken in the future.

Make your own keywords.

POWER 64 has its own table that tells it what to do when a key is pressed. For example, when you type a shifted L, this table tells POWER 64 to print the LIST keyword on the screen. When you hit the RUN key, this table says to print RUN on the screen and also act like a RETURN key had been hit.

POWER 64 also allows you to define a table just like the one it uses, and if you do, it will even search your table before its own, so that you can have complete control over what is done on your computer when any key is hit.

With this table, you can make any key print any string up to 255 characters in length, and you can even have a RETURN at the end of the string so it is executed by the 64 right away. You can also easily define any key to print any keyword in the keyword table of the 64 by

giving its token number.

Finally, you can define any key to call any machine language subroutine, giving you the full POWER to do whatever your heart desires.

You'll find precise details on how the instant keyword table works in Appendix D.

Adding POWER 64 commands.

POWER 64 finds its various commands, such as AUTO, RENUM, or DUMP from an internal table of command words. If you'd like to add features of your own, POWER 64 has made special provision for this.

Before checking its own internal command word table, POWER 64 looks to see if you want to perform your own tests. There's a location where you may place an action address if you wish; the POWER 64 system checks this location, and if the address is valid, you'll get control.

The address you store is carefully checked for correctness; the method used is a checksum test. You must get your address right for POWER 64 to recognize you. Once it does, however, you will take over the action until you return control to the POWER 64 system.

Note that you get first shot at checking the command; this means that you can override an existing POWER 64 command if you wish. Keep in mind that if you don't find any of the commands you recognize, you must pass control along to POWER 64 to allow it to check for its own commands. If POWER 64 in turn doesn't find anything it recognizes it will return control to the Basic interpreter; the command may be standard Basic, or it might be a syntax error. Basic has the last look; it will either perform the action or report the error.

So you could create new commands that might display array values, perform disk activities, delete REM statements, or do any other tasks you deem useful. POWER 64 can't do everything that you might imagine; but it gives you the hooks to add on such features.

You'll find a detailed description of how to link your own commands into the POWER 64 system in Appendix D. In addition, you'll find the source for a PAL assembler program that does what you need to add commands exactly the way the MOREPOWER package does it.

12. SUMMARY.

This User Guide has tried to take you through the features of the POWER 64 system one simple step at a time. The method has been to encourage you to get your hands on the machine. If you've gone through some parts without actually touching the computer, make a point of going back and trying them soon. You'll find that many of the features feel natural.

If you feel overwhelmed by the rich variety of features that POWER 64 offers, you might like to slow down and take things one step at a time. Even if you use only a quarter of the capabilities of POWER 64, you'll still be gaining many benefits; and you can add the extra features to your repertoire as you need them.

Remember that the instant action keyboard features can be turned off (with a SEL command) if you don't want them or if they get in the way. And you may shut down the whole POWER 64 system by using the OFF command.

Good luck with your new system. You'll write better programs faster and more easily. But keep in mind that POWER 64 isn't a substitute for brains. Your programs will be good because you make them good. Don't be modest and give POWER 64 all the credit. Just be glad it's there.

APPENDIX A: The MOREPOWER Package of extra POWER 64 commands.

How It Works.

POWER 64 contains plenty of added features for your Commodore 64, and after reading the POWER 64 manual, you may think it has more than enough. If you have a disk unit, though, you'll find out very soon that there are a lot of things about disk operation that could be more convenient. MOREPOWER to the rescue!

MOREPOWER contains a whole bevy of additional handy commands that have been placed onto POWER 64 by means of the built in hooks left in POWER 64 for this purpose. It is just one of several packages of extra goodies for POWER 64 that you should see around in the near future.

MOREPOWER contains many fancy disk commands, and it also contains stuff for defining keys for even more instant phrases.

How to Load MOREPOWER.

Most people want to have MOREPOWER with POWER 64 all the time if they have a disk system. This is easy to do. As you will recall, to engange POWER 64 you simply load it off the disk and type RUN. What the main POWER 64 manual didn't tell you is that you can select certain options by entering certain characters after the RUN command.

For example, if you type "RUN:1" you will find that both POWER 64 and MOREPOWER will load in together and be set up properly. You will know this because the disk will still be loading after you see the POWER 64 copyright message appear on the screen.

If you type "RUN:2", you won't get MOREPOWER, but you will find that your copy of POWER 64 will load up into a special area of high memory that your Commodore 64 has. This will mean that POWER 64 will take up none of the memory Basic uses, which could be important if your programs use up a lot of memory. The place POWER 64 loads is just below address "$D000" in hexadecimal. You should be careful when loading POWER 64 there because only one program the size of POWER 64 can exist in that memory.

If you add 1 and 2 together you might figure out what "RUN:3" does. You guessed it: this will provide you with POWER 64 and MOREPOWER together, and will put POWER 64 in the special high area of memory you were just told about. If you know no other program wants that memory, this is the option for you.

If you don't load MOREPOWER in at the start, you can still get it after loading in POWER 64. With POWER 64 in place, just type: LOAD"MOREPOWER",8 and when it has loaded in, type RUN, just like you do for POWER 64. It will then be linked in and you'll have even MOREPOWER at your fingertips.

The MOREPOWER Commands

All MOREPOWER disk commands ignore all data 19 characters after the start of the file name. This allows all MOREPOWER disk commands to be issued by getting a disk directory on the screen, moving the cursor up to the desired file and typing in the MOREPOWER disk command in front of that file name. With MOREPOWER, you need never type another file name for loading if you don't want to.

DEVICE

The MOREPOWER disk commands all take a filename with an optional device address and secondary address just like normal Commodore 64 I/O commands. The difference is that if you do not provide a device, MOREPOWER commands will assume the you want the default given by the DEVICE command. If you don't use the DEVICE command, the default device will be 8, which is normally your disk drive.

To make the default be device 9, for example, just type the command: DEVICE 9 and hit RETURN.

Commodore 64 commands default to device 1, which is the tape drive, which is generally a bad idea if you have a disk. You can get this, however, by typing DEVICE 1.

DISK

The DISK command lets you send a string to the disk command channel. Typing:

```
DISK"Command"
```

is the same as:

```
OPEN 1,8,15:PRINT#1,"Command"
```

... and you don't have to deal with opening files and other problems. You can use this to initialize disks, scratch files, copy files and many other purposes. Disk commands are listed in your Commodore disk manual.

ERR

This command prints the disk error channel. Issue it if the disk error light is on. It takes no arguments. You can also get the error channel printed by hitting the "f4" key on the right side of your keyboard.

LIST

What's so special about LIST - everybody knows it's a standard command in the 64, right? Well, that's true, but MOREPOWER adds an extra dimension to this command by giving it the ability to list and let you look at disk files without disturbing the program in memory.

The secret is to type a filename, in quotes, after the LIST command. For example, if you have a program called "B PROG" on the disk, you can type:

LIST"B PROG"

and you will see the program, assuming it's in Basic, appear on the screen just like you loaded it and typed LIST. You'll find this LIST has some extra features, too. For example, at any point you can hit any key and that will pause the listing so you can look more carefully at it. Hitting any other key will start it going again. True, if you hit the RUN/STOP key, the listing will stop, and you'll have to start it again, but that's probably what you wanted, or else why did you hit the RUN/STOP key?

You will find yourself using this command a lot, in particular if you are programming a large project that requires several files. You can use it to answer questions like: "What did I call that array?" and "How did that program use the GET statement?" without having to save your program and load another one. You'll find this a big time saver.

You might remember now, if you have used the disk a lot, that the disk directory loads in just like a Basic program does. If that's the case, you should be able to LIST it just like any other Basic program. You sure can, and it turns out to be one of the best features of the new LIST command. Just say:

LIST"$"

and you will see your disk directory. You may also be interested in knowing that key "f2" has exactly this effect as well!

MERGE

As noted in the main POWER 64 manual, you can use the EXEC command to do all kinds of amazing tricks, including a program merge. MOREPOWER has a command to make that merge process a bit more convenient if you have a disk. If you have one program in memory and want to merge another one in with it, just as described in the section on EXEC, just type:

MERGE"PROGRAM"

In this case, the PROGRAM is a standard loadable program file and you need perform no special conversion on it. You'll find the operation of MERGE to be quite similar to that of EXEC, and you can hit the RUN/STOP key to halt it, too. It should run a little faster than an EXEC while it's at it. The same warning that you were given in the EXEC section about not trying this without enough memory goes double here!

RUN

This is yet another built-in command of the 64 which has been enhanced by MOREPOWER. Just like LIST, RUN can now be given the name of a file in quotes. It will load that file into memory and RUN it. In fact, it's just the same as typing LOAD with the file you want and then typing RUN.

Since all MOREPOWER disk commands will work off a disk directory, you can pick up a handy trick from this. With POWER 64, the RUN key produces a RUN and a RETURN. You can use this to save a lot of typing when loading programs. For example, say you have (let's be honest) your GAMES disk in the drive. You can hit key "f2" and a disk directory will appear on the screen. Simply move the cursor up to the game you want, and hit the RUN key. With no more typing, your program will be brought into memory and run - just like that!

There's another, very interesting way of loading a program into memory and running it that will be described later.

LOAD

Yet another altered command, the LOAD command has been modified simply to use the MOREPOWER disk command rules. This means you don't have to type ",8" every time you use this command on disk, and you can also use it in a disk directory.

While POWER 64 defines key "f1" to be LOAD, MOREPOWER defines key "f8" to be LOAD followed by a RETURN. This means you can play the same trick described for RUN when using LOAD. Get a directory on the screen, move the cursor to the program you want, and hit the "f8" key. That's all you have to do and your program will be on its way into memory.

START

START is a command that tells you the location where a disk program file is supposed to load. It gives it in decimal and hexadecimal, and is handing in dealing with machine language programs. When dealing with a Basic program, it doesn't matter where it is supposed to load.

SIZE

This command will tell you the size of a disk file in bytes, giving the value in both decimal and hexadecimal. If you don't give it a file, it will tell you the size of the current program in memory. Unlike the FRE function of Basic, this command tells you how much memory you are using, rather than how much you have left, and doesn't deal with memory taken for variables and strings.

TEXT

TEXT is a command very much like the modified LIST command except that it takes sequential files and lists them rather than Basic files. You might know these as data files, and they show up with an SEQ beside their names in a disk directory. Otherwise, TEXT works just like LIST.

Subsystem Loading

We now come to a very special feature of MOREPOWER - one that lets you define your own commands for the 64 very easily. Let's say you have a Basic program on the disk that does something - anything. It might print your name over and over or compact another Basic program. You might be sitting working with one Basic program and find the need to quickly load in and run another. Your first program you might call a "system" and the second program would be a "subsystem"

One way you might do this would be to use the TEST command, load in the new program, RUN it and use the BACK command when it is finished. MOREPOWER allows you do do all this in one stroke, and even from a directory listing, no less.

All you need do is type the filename of the Basic program you want in quotes. No command, nothing else - just start the line with a quote. MOREPOWER will save away the important information about your original

program and create a new area of memory for the program to load into. The new program will then be loaded into memory and run, in a similar fashion to that of the modified RUN command.

When the program is finished, however, your old program will be immediately restored - not just the text, but all the variables and arrays as well. (Strings may not survive but all numbers will.) This happens when the subsystem finishes for any reason, which includes an error or your hitting the RUN/STOP key. It will be as though the subsystem program were never there. Remember, though that the subsystem program should follow all the rules that a program you can load in under the TEST command must follow - ie. it can't contain machine language or any absolute memory references to within it.

This may seem like a bit of magic, and in some ways it is for it provides you with the POWER to program your 64 with all kinds of handy commands even if all you know is a little Basic.

If you know more than a little Basic, and have a machine language program on the disk, you can load it in like a subsystem, too. Placing an exclamation point in front of the file name in quotes will cause MOREPOWER to load the given file into memory at the address it says it belongs, and also to start it running at that address. This can save you figuring out a funny SYS number for the average machine language program.

You should watch out when loading machine language programs like this. Many such programs are not designed to run from their first address, and if you use this command your machine could well crash! A tried and true command should give you no trouble of course, but don't just go typing random names after an exclamation point, or you'll be sorry.

UNDO

This command will UNDO the effects of a NEW if you issue it before any command that uses memory. This includes anything starting with a line number or anything that assigns to a variable. If you haven't done any of these operations, and you suddenly realize you did not want to type NEW, UNDO can save your day.

HEX

The HEX command is very handy for people working with the hexadecimal address system. If you follow the HEX command with a Basic expression, it will print out the value of the expression (usually a number) in hexadecimal. On the other hand, if you follow the HEX command with a dollar sign and a hexadecimal number, it will give you the decimal value of that hexadecimal number. A good way to handle all that annoying conversion work when using PEEKs, POKEs and SYSes.

KEY

The KEY command lets you define a whole new class of instant phrases and subroutines. As you will recall, POWER 64 allows you to define instant phrases and subroutines by using REM statements at the start of the program. These are very useful, in particular for strings relevant to the program being worked upon. The KEY command lets you define instant phrases and subroutines that don't change from program to program, and stay until you turn POWER 64 off on your machine.

The secret is the POWER 64 key table, similar to the one used to map a shifted L into the LIST command. When you start MOREPOWER up, it creates a table for you to put your own key definitions into, and the KEY command lets you do this. See the section on machine language interfaces for more details on how this is done.

Say you want to define a key to print a string. You might try:

```
10 rem"N=Albert Gonk"
```

but you can also try:

```
key "N", "Albert Gonk"
```

This will define the shifted N key as long as there is not a REM style instant phrase superceding it. It also means that you can't get NEXT from the shifted N key like you used to. Keys defined in this way can be turned on (if they are not already) by SEL K+ and off with SEL K-.

Just typing:

```
key "N"
```

will get rid of any definition you gave for the shifted N key. Of course, you can define any key, not just shifted N.

You will be happy to know that the last character in the string you give is not printed on the screen, but actually replaces the character you typed. This means that if the last character is a RETURN, it will seem as though you hit RETURN on the line the cursor is on. Try:

```
key "N", "?3+5"+chr$(13)
```

and then hit the shifted N key. Don't forget to have SEL K+ on at the time. Buy the way, you can't have a chr$(0) in your string, nor define that character.

You can also have any key set to call a machine language subroutine. Say you have a machine language routine at address 30000 in memory. (If you don't know any machine language, don't worry about this.) You can then say:

```
key "N" sys 30000
```

and your routine will be called each time you hit the shifted N key. If, when the routine exits, the carry is clear, POWER 64 will do nothing unusual when it is finished. If the carry is set, however, POWER 64 will treat the character in the accumulator like it was typed at the keyboard, so you can play the same trick with the RETURN.

Built In Keys

When you use MOREPOWER, the four shifted function keys on the right of the keyboard get definitions, which, incidentally, you can change with the KEY command. Most of these have been noted above, but we'll list them here to be complete.

The "f2" key gives a disk directory.
The "f4" key prints the disk error channel.
The "f8" key prints the LOAD keyword and a RETURN

The "f6" key puts you into AUTO mode in a very special way. To use it, place the cursor on a line on the screen that contains a Basic line number, generally a listing of that line. If you hit the "f6" key, MOREPOWER will read the line number off the screen and move the cursor to the bottom of the screen. There it will put you in AUTO line number mode at one plus the line number read off the screen, with a line step of one.

What this does is allow you to easily add lines after a given line (assuming there is room for them) with the touch of one key.

Future Additions

MOREPOWER is designed to link onto POWER 64 and to allow other packages to link in before or after it in a chain. Expect to see even MOREPOWER commands and defined keys in the future, both from the author of POWER 64 and other users of it.

APPENDIX B: TROUBLESHOOTING POWER 64

In the unlikely event that you cannot seem to load POWER 64 from the disk, check out a few things before panic sets in. Relax a little, there are thousands of copies of POWER 64 in daily use and they all work beautifully. The trouble is probably something pretty obvious.

1. Check that the power plugs are all securely in the wall socket.
2. Check that the connecting cables are all plugged into the correct places.
3. Are you trying to load the right disk?
4. Does any other disk load properly?
5. If it does, read through your POWER 64 loading instructions step by step and try again.
6. If POWER 64 still won't load, contact your dealer.

APPENDIX C: FORMAL DESCRIPTION OF POWER 64 COMMANDS.

by: Brad Templeton

Optional information is shown in parentheses. For example: AUTO (line-number(,increment)) means that you may type AUTO or AUTO 100 or AUTO 100,10.

The keywords are given in upper case. Lower case phrases describe the type of information to be supplied. "Line-range" indicates the same type of numbers as with the Basic LIST command: 300-400, 500-, -900, and - are all valid line ranges.

AUTO (line-number (,increment))

The AUTO command puts the 64 into automatic line numbering entry mode. Each time RETURN is pressed, the machine will prompt with the next line number. This allows easy entry of Basic text. If a line-number is provided, the 64 will begin by prompting with this number. Each subsequent number will be a value of "increment" higher than the previous line. The increment value may range from 1 to 255. If no increment is provided, the last known increment will be used; its initial value is 10.

If no arguments are supplied, POWER 64 will search through the current Basic program for the last line number. It will then prompt with a number which is the increment value higher than this. This feature can be used to append lines to a program.

During entry, pressing RETURN will cause the next line prompt to be given. If RETURN is pressed immediately after a line is prompted (i.e., before any other key), auto line numbering will be turned off. The

RETURN character will be changed to a Shifted RETURN so that the line whose number was displayed will not be deleted.

DEL line-range

This command deletes a block of lines from the current Basic program. Giving a null line-range produces an error rather than deleting the entire program.

DUMP

This command causes a list of all 64 variables and defined functions to be listed. Variables will be printed with the following format:

"variable-name" = "value"

The user may change the value and press RETURN to cause the variable to be changed.

Functions which have been defined by executing a DEF FN... statement will be shown.

FIX

The FIX command re-initializes the POWER 64 package. Some internal pointers are restored. The end of the current Basic program is found and internal Basic pointers are set to match this location. Using FIX will destroy all Basic variables.

FIX is valuable if a bad program or a non-standard LOAD has upset some of the internal pointers used by POWER 64 or Basic, or if the second cassette buffer has been used for tape or other operations.

PTR

PTR resets Basic pointers like FIX but does not reset POWER 64 itself. It can be used after a non-Basic load.

OFF

The OFF command disables the POWER 64 package, restoring normal 64 operation. Notably, the 64 Basic link vectors in three page are restored to the states they had before POWER 64 was called. Once you have turned POWER 64 OFF you can re-enable it with a command of SYS 704.

RENUM

The renumber command will resequence line numbers in a program. Line numbers will be changed according to specified rules. All line number references in GOTO, GOSUB, IF .. THEN, IF .. GOTO, ON .. GOTO, ON .. GOSUB and RUN statements will be changed. To support the instant subroutine feature, line numbers appearing after the left-arrow character will also be changed.

Renumber allows any portion of the program to be renumbered to any line range, provided no overlap occurs as a result. The line range, if provided, specifies what section of the program is to be renumbered; if not provided, the whole program is renumbered. The first line in the specified range will be renumbered to the value given in "new-start". Each line within the range will be renumbered, with each line higher than the last by the increment value. An error notice is given if the renumbered line range conflicts with the sequence of other lines outside the range.

During renumbering, parts of the program must be moved to make way for larger line numbers. In the unlikely case that such expansion causes memory to be filled, an OUT OF MEMORY error will result, and the Basic

program may be irretrievably damaged. The user should try to avoid this problem by refraining from renumbering programs which are very close the the memory limit of his machine.

"New-start" designates the line number to be given to the first renumbered line. If it is left out, the increment value will be used.

The increment value gives the spacing between renumbered lines. If it is left out, a value of ten will be used. Thus a REN command with no arguments will renumber the entire program by increments of ten, starting at line ten.

SEL option-string (additional option-strings)

The SEL command allows setting or disabling of user features. Up to four features can be modified:

K - keyword expansion; enables expansion of keywords from shifted keys;
R - macro expansion; enables instant phrases and instant subroutines;
P - metacharacters; enables special characters for search/replace;
I - input; enables instant keys during program runs.

The sign may be + to enable the selected feature, or - to disable the feature. Normally, features P and R are enabled at startup and features K and I are disabled.

Several option-strings can be specified within a SEL command: for example, SEL K- R+ is acceptable.

TRACE (options) (↑)

TRACE enables trace features. The option ↑ indicates that trace output is to be written to a special group of inverted lines at the top of the screen. If it is not provided, trace output goes out normal vectors, and thus may be redirected by means of a CMD. TRACE STOP disables the trace feature.

Full trace is the default. This implies output of the listed line and the result of the operation for every line that is executed. Option LIST generates the line listing only, while option # causes line numbers only to be output. Lines are listed starting with the currently executing statement. If the first part of a line has already been executed, it will not be displayed. Line numbers are output in one of two formats when displayed alone; they are printed separated by colons (:) without carriage returns. In T mode, the last ten line numbers to be executed are scrolled in the top 80 positions of screen memory.

Variable output is provided beneath the listing of the executed line. Results are preceded by an equals sign (=). String results will be preceded by a quote, and will be truncated to screen width. These values are obtained by examining 64 internal memory locations at the termination of a statement. It is not always possible to have a meaningful value available. NEXT statements without a variable may generate an incorrect value. Only one result can be output per line, even though some statements compute several values during execution. (POKE, PRINT)

Tracing is controlled by three keys on the keyboard. If the "f5" key is held down, tracing will continue at full speed. Pressing the equals "f7" key once will cause one line to be executed; this is known as single-stepping. Holding down the Commodore symbol key will cause the program to execute normally (no tracing) while the key is held down. If the program is in a GET loop and needs input, you may type in your

Input, and then hold down the "f5" key or Commodore key until it has been read.

Tracing is turned of via the TRACE STOP command. Tracing remains in effect with its current options until changed or turned off by command or by the program. Tracing can be enabled, disabled or changed within a program by means of a SYS command. A vector in three page provides this feature. The user may code a program command of the form:

```
TR = PEEK(705)+256*PEEK(706)-3
SYS TR,["traceletters"]
```

to change trace operation. SYS TR,STOP will disable tracing, while SYS TR,#↑ will enable number tracing at the top of the screen. A simple SYS TR, will enable full tracing with normal output.

WHY

The WHY command is a debugging aid. It may be executed immediately after program execution is halted by an error or intentional means. If done at this point, the WHY command will list the line that was being executed at the time of the interruption with the block where the Basic memory scan pointer was printed in reverse field. This only indicates where the Basic scanner stopped, which is not always a perfect indication of the source of an error. If other commands are executed before the WHY command is done, the information it needs may be destroyed, in which case it will list nothing.

EXEC filenumber

The EXEC execute command executes lines from a Commdore 64 sequential file (disk or tape) as though they were typed from the keyboard. Each line is read from the designated file and placed in the Basic input buffer. Line-feed characters are ignored. Lines are printed on the

screen as they are received. Once the line is read in, control is passed to the 64 so that it can be processed. At end of file, the file is closed, and control is returned to the user.

To use EXEC, first open the file to be executed in the normal way (an OPEN statement). The file number used in the OPEN statement is then passed to EXEC on its command line; a variable or expression may be used to define the file number if desired. The file may contain a listing of a program in which case that program will be merged with the current one. A RUN statement may also be included on the file; this would start the current program. Operation of EXEC can be halted with the STOP key.

/ (pattern) (/linerange)

The search command allows the user to scan through Basic program text for a desired string. The search argument is provided as a "pattern", which will be searched for in one of two ways:

A) All occurences in the specified line range will be printed if the "/ linerange" sequence follows the command, or

B) The next occurence of the pattern will be printed.

A current-line pointer is maintained by POWER 64 and is used by the scrolling feature, the AUTO command and the Search and Replace commands. If the next occurence of a pattern is desired, the search will proceed forward, starting after the current line, wrapping around the end of the program to the beginning and continuing until the current line is reached. This feature allows the POWER 64 user to search through a program, listing occurences of a given pattern one at a time.

A pattern is used to specify the string to be searched for. Normally it is the Basic text that the user desires; for example, to find all occurences of NEXTX in a Basic program, one would type:

/NEXTX/

The pattern may also contain what are referred to as "meta-characters", which are pattern-matching characters with special meanings. The dot character (.) will match any character or token in Basic text. For example, /A.B/ would find all cases of strings such as AAB ABB ACB A$B etc. The dot will also match any internal Basic token, since the search is done byte by byte. Thus the above would also match cases of ATOB, ATHENB etc.

As the dot matches any arbitrary character or token, the star (*) will match any arbitrary string on a given line. For example, the pattern FOR*NEXT would find all cases of a FOR which is followed by a NEXT on the same line. Similarly FOR*TO*STEP would find all FOR statements with STEP modifiers, no matter what variables or other conditions are in the statement. The star matches the shortest possible arbitrary string, so to match the beginning of a line, it is necessary to use dot star (.*), which will match any character (the first) followed by an arbitrary string.

The end of the line (not the last character) may also be matched with the close bracket (]) character. This is generally of use during substitutions (see Replace command) but can be used in searches. The pattern NEXTX] will find all lines ending in NEXTX. The *] pattern will match the remainder of a given line. (It is assumed some characters preceed this pattern, since *] alone will match all lines.)

The pattern argument of the Search command is optional. Should it be left out, the pattern given in the last Search or Replace command will be used. This allows convenient searching for further occurences of an

already typed pattern. For example, just typing the '/' command alone on a line will cause a search for the next occurence of the last pattern used. This allows the user to go through occurences of a pattern, one by one, with just a two keystroke (/,RETURN) sequence being required to list the next occurence.

Replace command

S/ (search pattern) / (replacementstring) (/ "linerange")

The replace command will take the given pattern and replace it with the replacement string at all occurences in the given line range. If the terminating / symbol and line range are omitted, it will work on the same line by line basis as Search. As in Search, the pattern may be omitted, which will cause the last pattern to be used. The escape character may be used in the replacement string to avoid tokenization and to include a / symbol.

Replace operation is very similar to Search operation. Each line changed is listed on the screen after it is changed. Replace action over a line range can be aborted with the STOP key.

INSTANT ACTION FEATURES

POWER 64 provides many instant action features. They are detailed below

Basic Program Scrolling

POWER 64 provides the ability to scroll back and forth through a Basic program in full screen editor style. To do this, a special interpretation is given to cursor up charactrers typed on the top line of the screen and cursor down characters on the bottom. When cursor down is typed on the bottom, POWER 64 reads screen memory to find the line number of the last line on the screen. To do this, it looks at the first two characters of every logical screen line, going upwards from the bottom. If it finds a number starting in such a place, it reads it using standard Basic number reading routines. It then finds the next line in the Basic program and prints it at the bottom.

Cursor up characters have a similar effect. When one is typed, POWER 64 searches forward from the first character on the screen for a line number. If it finds one, it searches the program for the previous line and lists it at the top of the screen. The text on the screen is moved down one or two lines to accomodate for the new line being listed. There may be some problem with lines longer than 80 characters, but POWER 64 handles most contingencies. Blank lines are rarely left on the screen, and lines longer than 40 characters are not left truncated on 40 column machines.

It should be noted that since scrolling from the bottom searches the first two characters of each line, user typed lines will be found as well as lines listed in the conventional manner. It is possible with POWER 64 to type a number at the bottom of the screen, cursor down and see the lines following it. This is usually quicker than a LIST command for the same lines.

REM MACROS (Instant Phrases and Subroutines)

POWER 64 provides the facility to redefine keys by means of special REM statements. A key can be defined to print a string or call a Basic subroutine. When any key is pressed, a search is done through statements at the beginning of the program. The search is terminated if a definition for the key pressed is found, or if some statement that is not a key defining statement is found. A defining statement which causes a string to be printed is of the form:

```
10rem"P=Power
```

which will cause the shifted P to print the word Power on the screen. The quote allows graphic characters and cursor movements to be included in the string, which means that graphic pictures can be generated with a single key.

If a definition statement is of the form

```
12REM"T<100
100?"the time is ";TI$:RETURN
```

it will define the shifted T so that it calls the subroutine at line 100, which in this case prints out the time of day variable. Uses for this feature are unlimited, and are detailed in the tutorial section of this manual.

REM macros can be disabled by means of the SELR- command, and re-enabled by means of the SELR+ command.

Instant Keywords

When instant keywords have been enable by SELK+, pressing a shifted key will cause a keyword to be printed on the screen. This will not happen if a REM macro is defined for that key. The keywords are associated in a loose manner with the letters that call them. Refer to Appendix F for details of the mappings. The keyword expansion feature can be enabled and disabled with SELK+ and SELK- respectively. The keyword is printed by a POWER 64 routine called PRKWRD. See the machine language interface section for details on that.

APPENDIX D: Machine language interfaces.

General information.

POWER 64 must co-exist with the internal 64 operating system. It interacts with and makes heavy use of built-in ROM routines. The methods used lead to some considerations that machine level programmers should know about. POWER 64 provides facilities for expansion and compatibility with other packages; this is documented here.

The Commodore 64 ROM Basic provides special vectors to allow programs like POWER 64 to alter the characteristics of Basic operation. POWER 64 makes use of four of these vectors in normal operation. Realizing that other programs may wish to make use of these vectors as well, POWER 64 does not destroy them. Rather, before altering them it saves the old contents away, and where it would use the routines these vectors point to, it uses the saved values. It should thus be possible to use POWER 64 with other programs that make use of these vectors, as long as POWER 64 is called into being after these programs are, or if these programs are as courteous as POWER 64 in their use of the vectors.

The four vectors used are known as the character INPUT vector, the character OUTPUT vector, the WARMSTART vector and the EXECUTE vector. All others are left untouched.

POWER 64 adds commands by intercepting the EXECUTE vector, which is passed through efore every statement is executed. The WARMSTART vector keeps POWER 64 going and helps the WHY command. The instant action features are all provided by means of the INPUT vector.

The Keyboard Map Table

Several of the instant action features of POWER 64 are controlled by the Keyboard Map Table. Whenever a key is pressed, POWER 64 searches this table to see if it has an entry for what to do with this key, provided keyword expansion is enabled. There is a table inside POWER 64 that maps the instant keywords that come with the system, but POWER 64 allows for the user to add his own table that will be scanned before the one inside POWER 64. To add your own table, you must alter a special vector in POWER 64's page two memory block. It is called USRDEX. To have your own table, place the address of it in the two byte vector USRDEX. Your table can be up to 255 bytes long. To tell POWER 64 you have a table for it to scan, you must put the correct value in a check byte called USRDEXCK. To calculate this byte, add the two bytes of your new USRDEX vector together and exclusive or the sum with hex value $A5. Store this result in USRDEXCK and POWER 64 will scan your table.

Table entries are two or four bytes long. The first byte in any entry is the character being defined. If this byte is zero, this signals the end of the table. Otherwise it defines a character. The second byte says what to do with the character. If it is zero, that means to stop the scan immediately and do nothing with this character. Through this, you can stop expansion to keywords done in the second table.

If the second byte is negative, (high bit set) it is taken to be a Basic token number, and the keyword cooresponding to that token will be printed. If it is a function token, an open parenthesis will be printed after the keyword.

If the second byte is one, the next two bytes in the table are considered a subroutine address. The subroutine at that address will be called. If it exits with carry set, the character in the accumulator will be passed back to POWER 64 to be treated as though it had been typed. If carry is clear, the accumulator will be ignored.

If the byte is a two, the next two bytes are treated as a pointer to a string. This string will be printed up to the first zero byte. The last character in the string before the zero byte will be treated as though it were typed, so if it is a RETURN character you can get this key to cause action.

All other second bytes are undefined.

MOREPOWER creates its own Keyboard Map Table, and lets you work in it with the KEY command. If a table already exists, it will work with that one instead.

POWER 64 uses memory. Zero page storage is used only temporarily; permanent storage is kept in page two of 64 memory. In general, the information there is not overy sensitive, so they may be overwritten without crashing POWER 64. Overwriting these bytes will, however, cause some options and default parameters to change in unpredictable ways. This storage area is from $2C0 to $2E0, and the locations themselves that are of interest are given in the memory map below.

Adding commands to POWER 64.

POWER 64 provides a facility for adding extra commands to the thirteen already provided. This is done through a vector in the page two area. POWER 64's control of the command scan takes place after the line has been tokenized. This means that if you wish to recognize additional commands, you will have to do so with a tokenized line. (Watch out here! For example, if the command contains the letters "TO", you will have to check for $A4 instead of a "T" and a "O".) There is a three byte vector consisting of the byte UCVCHK ($02D2) and the indirect jump vector UCOMVEC ($02D3). Before command scanning is done, POWER 64 checks to see if a valid vector is present for added commands. To do this it adds (carry clear) the two bytes in UCOMVEC. The result is then

exclusive or'd with the value $A5. (binary 10100101) This is compared with the check byte UCVCHK. If they match, an indirect jump is done through UCOMVEC to the user routine waiting there. Note that this jump is done before normal POWER 64 command scanning occurs, so the user may redefine POWER 64 commands.

The user routine may scan for commands in the Basic Input Buffer ($200) in any way it wishes. If it does NOT find any command it wishes to interpret, it should return control to POWER 64 and Basic. To do this, a JMP should be made through the vector NORMCOM. If the user does find a command, POWER 64 will help him call it. First, the user routine should alter the Basic execution pointer ($7A-$7B) to point to the last byte in the text of the recognized command in the Basic input buffer. Then, if ADDR is the address of the routine to execute the command, the user should load the low byte of ADDR-1 into the Y register and the high byte of ADDR-1 into the X register. A JMP should then be made to ENTCMD. This will start the user command executing in the normal Basic-POWER environment. In this environment, Basic and POWER 64 routines may be called to scan the command line and evaluate expressions. CHRGET may be used to get subsequent bytes, and a simple RTS will terminate the command and allow Basic to pick up as usual. When the command is invoked, the first byte following the command on the command line will be in the accumulator, as is usual for a Basic statement.

The vectors ENTCMD and NORMCOM noted above move about in memory depending on where POWER 64 is located. To call them, you will have to build indirect jump vectors for them. This is explained in the section of POWER 64 routine addresses, and the example program for adding commands to POWER 64 that comes with this document shows how to do this.

When adding new commands, it is a good idea to allow them to 'chain'. This means that several new commands can be added independently. This can be done in the following way:

When new commands are being added, it will be necessary to alter UCOMVEC and UCVCHK. Before this is done, the user code should check to see if they already contain valid entries (ie. somebody else is already there). If this is the case, the user should save the old vector in a private place. When the time comes to JMP to NORMCOM if a command is not found, it should instead jump to this vector. The user can also make it easy by always jumping through this vector, and putting NORMCOM there if there is nothing valid in UCOMVEC. Naturally, the user may wish to put check bytes on the vectors he saves away.

The sample program to do all this is on your POWER 64 program disk under the name ADDCOMS.PAL. This program adds one sample command to POWER 64 called NAME, which simply prints a string.

POWER Entry Point Vectors

There are several useful vectors at the front of POWER 64. Already documented in the main manual is USRINV which allows the user to control tracing from within a program.

The difficulty is using these vectors is that they move about in memory as POWER 64 is relocated. In order to call these vectors, which are found at the start of POWER 64 memory, you must create indirect JMP vectors based on your calculations of where they are.

At location 704 decimal or $2C0 hexadecimal, POWER 64 stores a 3 byte JMP instruction to start it up again after an OFF. The address in this JMP is just after the entry point vectors. You can calculate your own JMPs for POWER 64 vectors by taking the value at $2C1-$2C2 and subtracting the amount shown below from it. In the listings below, the value at $2C1-2 will be represented by the symbol "POWER".

USRINV (POWER - 3) is, as documented before, the place to call from Basic to change TRACE parameters.

NORMCOM (POWER - 6) is the location to jump to after scanning the command line when adding extra commands to POWER 64 if you decide that you haven't found one of your own commands.

ENTCMD (POWER - 9) is the jump location to call your own extra command once you have found it. It's use, with NORMCOM is documente above.

POWER 64 has the ability to make substitutions in Basic text. This is provided as a subroutine for the more adventurous programmer. Before REPL (POWER - 12) is called, the following should be set up:

POINTR ($7A)	- pointer to first byte beyond string to be replaced
LINPTR ($5F)	- pointer to start of string to be replaced
NEWLEN ($9B)	- length of replacement string
NEWBUF ($120)	- replacement string starts here

REPL fixes all internal Basic chain and end pointers. An OUT OF MEMORY ERROR can result if there is not enough space left.

JSR DORANG (POWER - 15) will read in a line range from the Basic input stream. This is handy if you are defining new commands. Once called, a pointer to the first line in the range will be in LINPTR ($5F) and the number of the last line will be in INTADR ($14).

A JSR WRP (POWER - 18) will correct the Basic link chain and alter the end of Basic pointers in zero page to point to the real end.

JSR DELEN2 (POWER - 21) is the memory mover (downwards) used by REPL. The low address (start of destination block) is kept in LINPTR ($5F) and the start of the moved block should be in TP1 ($58). Memory up to the end of Basic text is moved ($2D).

The routine INGOCALL (POWER - 24) can be used to call Basic subroutines from machine language programs. LINPTR ($5F) should contain a pointer to the first line in the subroutine. This would be set up by a call to the Basic routine FINDLINE ($A613) with the line number in INTADR ($14). The Basic subroutine will be called, and when it RETURNs, it will come back to your machine language code. This is tricky, however, as the Basic routine could have side effects that could cause havoc for your program.

Routine LSTLIN (POWER - 27) lists a program line through POWER 64's output subroutine. This normally points to the normal 64 output routine. A pointer to the start of the line should be in LINPTR ($5F). This pointer should point at the link bytes preceding the line.

PRLNO (POWER - 30) is a handy routine to print the line number found in LNOLO ($3B) out the standard input vector.

PRKWRD will print a keyword from the POWER 64 keyword table. The index number of the keyword in the table should be in the X register when the routine is called. The keyword table format is the same as that used by the 64. If the X register contains $FF, the symbol PI will be printed. Keyword numbers range from $80-$FE. This routine is special, in that it is not in the normal vector area. It's address can be found at location KEYWLINK ($2CB), and a JMP indirect through there will transfer to it. Note that programs that add extra keywords to the 64 should change this vector to a routine that understands the new tokens if they want POWER 64 to deal with them.

POWER Memory Map

The only permanent storage used by POWER 64 is in page 2. Locations of interest or use to you are given below.

CALLVEC ($2C0-3) - Three byte vector giving startup address for POWER 64. Also of use for locating POWER 64 routine vectors.

WARMLSAV ($2C3-4) - Storage for old WARMSTART link vector.

RUNLSAV ($2C5-6) - Storage for old EXECUTE link vector.

INLSAV ($2C7-8) - Storage for old INPUT link vector.

KEYWLINK ($2CB-C) - Special vector that POWER 64 jumps through to print keywords. Should be altered to new routine by any program that adds keyword tokens to 64 Basic.

TEMSA ($2CE) - This location is used by the EXEC command to store the secondary address of the file being read. It is restored after every line pulled in by the EXEC command, since it is possible that EXEC lines will close files.

TEMDEV ($2CF) - This location stores the device of the file used in the EXEC command.

OUTLSAV ($2C9-A) - These locations are where the 64's OUTPUT vector is saved when POWER 64 changes it.

AUTFLAG ($2E0) - This flag indicates whether AUTO line numbering mode is in effect. It is nonzero if yes, zero otherwise.

UCOMVEC ($2D3-4) - This is another indirect JMP vector. It is used for adding commands to POWER 64, and a detailed description of this is given in the section on that topic.

UCVCHK ($2D2) - This is the check byte for UCOMVEC. See the section on adding commands to POWER 64 for details.

WHYCNT ($2D5) - This is a special counter used by the WHY command to indicate how far into the line Basic was when it stopped.

WHYPTR ($2D0-1) - This is the actual pointer into the Basic text where Basic stopped scanning, used by WHY.

REMFLAG ($2D6) - This flag indicates whether REM macro expansion is on. Negative for on, zero for off.

USRDEX ($2D9-A) - This points to the Keyboard Map Table if the user sets one up.
USRDEXCK ($2D8) - Check byte for USRDEX.

OPTFLAG ($2DB) - General option flag for K and P options of SEL. Keyword expansion is ON when this is negative, OFF when positive. Metacharacters are not given special meanings if bit 6 is on, they are if it is off.

AUTINC ($2DC) - Current AUTO increment.

CURSOR ($2DD) - Output cursor for reversed trace output.

TFLAG ($2DE) - Flag that controls trace mode. Zero for Trace off. Odd if tracing is to be at top (bit 0=1). Bit 6 on for a Numbers trace, bit 5 on for a Listing trace, and bit 7 on for a full trace.

INPUTFLG ($2D7) - Flag for SEL I option.

APPENDIX E: Anomalies of the POWER 64 system.

There are certain things that happen with the POWER 64 system that seem odd. They are not errors or oversights - just odd things that happen under certain circumstances.

Many of these happenings are a result of the 64's internal workings. POWER 64 taps into the 64's mechanisms - and some of these aren't able to react to every possible combination of program and command.

Don't get upset if some of the occurences listed below happen to you. They are normal, and don't mean that POWER 64 is malfunctioning.

Search oddities

Keywords such as LOAD or REM can occasionally be confused with cursor movements such as CLEAR and HOME. They are stored in a similar manner within the computer, and POWER 64 sometimes has trouble telling them apart.

Instant features lost on RESET

You will find your POWER 64 instant features gone if you reset the machine with the STOP and RESTORE keys. Use FIX to get them back.

Scrolling anomaly

If a Basic line exceeds 80 characters in length, the listed line will "spill over" into the next line. If the overrun happens to be a number, POWER 64 will confuse it with a legitimate line number and will not scroll properly to the next line.

If the cursor is not at the left of the screen when scrolling takes place, the first line printed will not start at the beginning of the line.

Also note that if POWER 64 finds a number on the screen that is not a valid line number, scrolling the screen with the cursor-up or cursor-down key will cause the computer to print "SYNTAX ERROR." Not to worry, once this illegal number (which POWER 64 perceives as a line number) is gone, the error will go away, and POWER 64 will scroll your program from the beginning.

Conflicts with programs

Programs which use the page two area of memory that POWER 64 uses for storage may conflict with the POWER 64 system.

Programs or systems making use of the same link vectors to create new Basic commands are likely to conflict with POWER 64 unless they are called before POWER 64 or are as courteous as POWER 64 in using these links. If possible, the POWER 64 "hooks" should be used to allow the programs to exist together.

Programs with machine language portions should not be used with POWER 64 unless care is taken to ensure that pointers or programs are not harmed. POWER 64 is intended primarily for Basic support: it may not know of special requirements of the machine language systems. Experienced programmers may use PTR to good advantage, restoring vital pointers as necessary before running or saving.

Renumbering

If the Basic program should contain a line number reference of 64000 or greater, renumbering will not take place correctly and the machine will report ?SYNTAX ERROR. The Basic program will likely be harmed and the

user should not attempt to continue using it. This should be a rare occurence, since you cannot have Basic lines numbered 64000 or over; it would be most unusual to code GOTO 64000.

On rare occasions, renumbering a program to higher line numbers might cause it to require more memory than is available in the system. If this should happen, an ?OUT OF MEMORY error will be generated, and the program will be irretrievably damaged.

APPENDIX F: INSTANT KEYWORD TABLE

This table shows what instant keywords you get from various keys when keyword expansion is enabled with POWER 64.

Keys are listed below with what keyword they produce. To get a keyword, you must press the given letter along with either the Shift (SH), Control (CTL) or Commodore symbol (CBM) key.

Key	Keyword
SH A	ASC(
SH C	CLOSE
SH F	FOR
SH G	GOSUB
CBM G	GOTO
SH I	INPUT
SH L	LIST
CBM L	LEFT$(
SH N	NEXT
SH O	OPEN
SH P	PEEK(
CBM P	PRINT#
CTL P	POKE
SH R	RETURN
CBM R	RIGHT$(
SH S	SAVE
SH T	THEN
"f1"	LOAD
RUN	RUN<return>

APPENDIX G: A Sample Basic Program.

Here's a sample program you can use to try out your skills on the POWER 64 system. It is a simple program to find prime numbers within a given range.

```
100 print"prime number finder"
110 print"find primes from";
120 gosub 330
130 f=v
140 print"up to";
150 gosub 330
160 t=v
170 f1=f
180 if t<2 goto 300
190 if f<=2 then j=2:gosub 370
200 if t<3 goto 300
210 if f<=3 then j=3:gosub 370:f1=4
220 f2=int(f1/2)*2+1
230 if f2>t goto300
240 for j=f2 to t step 2
250 for k=3 to sqr(j)+.01
260 x=j/k
270 if x=int(x) goto 290
280 next k:gosub 370
290 next j
300 print:print n;"primes found from";f;"to";t
310 stop
320 print" - try another number";
330 input v
340 if v>1e9 then print"too high";:goto 320
```

```
350 if v<f then print"too low";:goto 320
360 return
370 print j;
380 n=n+1
390 return
```

Here are a few exercises you might like to try with this program:

1. Enter the program using Auto Line numbering. If you like, use Instant Keywords.

2. The program is too big to fit on the screen. Scroll up and down and look at the various parts of it.

3. Run the program and dump the variables with DUMP.

4. Renumber the program so that it starts at line 10 and goes in steps of 10.

5. Now find the new line numbers for the two subroutines. Searching for GOSUB should do the trick.

6. Renumber the subroutines only so that they start at line 500.

7. Change variable f2 to m2.

8. Change SQR to SQT. This will cause an error, of course. Run the program, and when it stops, ask: WHY? Then fix the problem.

9. Set TRACE ON and run the program. Don't forget to use the "f5" or "f7" keys to step through the instructions. Watch it all work.

10. What do you think variable f2 does? Find all occurences of the variable and try to figure out why it is used that way.

11. Try your own tests, changes, or whatever.

WARRANTY DISCLAIMER

All Pro-Line Software Ltd. computer programs, whether supplied on magnetic media such as but not limited to cassette tapes or diskettes or in any other form, any associated devices and any instructions or manuals, are sold on an "as is" basis without a warranty of any kind expressed or implied. Pro-Line Software Ltd. does not warrant the fitness of any program for any purpose nor does Pro-Line Software Ltd. warrant the accuracy, quality or freedom from errors of any programs, devices, instructions or manuals.

Pro-Line Software Ltd., their distributors, agents and retailers assume no responsibility for any consequential, incidental or other liability arising from the use, the inability to use or the attempted use of any program nor for any lost profit or any lost savings however incurred.

* * * * * * *

PLEASE PROVIDE THE FOLLOWING INFORMATION

№ 14825

NAME: _____

ADDRESS: _____

CITY: _____

STATE/PROV.: _____

ZIP CODE: _____

PHONE: _____

PLEASE CIRCLE ONE

Power 64

CHEQUE, MASTER CARD, VISA, AMEX

CARD NUMBER: _____

EXPIRY DATE: _____ SIGNATURE: _____

WE'LL BACK YOU UP!

ONE BACK-UP COPY OF THIS PROGRAM IS AVAILABLE BY RETURNING THIS COUPON AND A CHEQUE, MASTER CARD, VISA OR AMERICAN EXPRESS NUMBER FOR $20.00 TO:

PRO LINE SOFTWARE LTD.
755 THE QUEENSWAY EAST, UNIT 8
MISSISSAUGA, ONTARIO CANADA
L4Y 4C5

PLEASE SEE OVER

made in Canada by
PRO-LINE SOFTWARE LTD. MISSISSAUGA, ONTARIO