REBEL ASSEMBLER EDITOR

- INCLUDES BOTH 64 & 128 VERSIONS!
- SOPHISTICATED SOURCE EDITOR THAT ALLOWS FULL USE OF ALL RESIDENT BASIC COMMANDS, GIVING YOU TOTAL CONTROL OF YOUR COMPUTER!
- USES INDUSTRY STANDARD MOS OP-CODES!
- LIGHTNING FAST ASSEMBLER! (ASSEMBLES A 45K SOURCE TO A 7K OBJECT IN 16 SECS!)
- C-128 80-COL. VERSION HAS ON-SCREEN COMMAND SUMMARY WITH SPLIT SCREEN EDITING!
- C-64 EDITOR INCLUDES PAGE FLIPPING, LINE SCROLLING, SEARCH & REPLACE, ETC.

TABLE OF CONTENTS

REBEL ASSEMBLER/EDITOR
Copyright (c) 1985, Mike Taylor

Be sure to register your software package with us on the enclosed card and let us know your comments. We can be reached at Nu- Age Software, 2311 28th Street No., St.Petersburg, FL 33713.

PUBLISHED BY NU-AGE SOFTWARE

Rebel USER'S GUIDE

Rec'd 3/2/87

Introduction:

The Rebel Assembler/Editor provides powerful features and flexibility with lightning speed. The source code you write on either the C64 or C128 is totally transportable to either computer. The Editors are different, but you can write code for either computer on the other. If that weren't true, the C128 version wouldn't exist.

Since the Editor and Assembler are co-resident **100%** Machine Language programs (on both computers) there is no irritating delay while changing programs. You aren't distracted by the chores of saving and loading and reloading, ad nauseum. You won't need any special combobulators to turn object code into real object code. If you want to assemble to RAM directly, go for it.

If you want to keep your source in memory while assembling to DISK or RAM that's no problem. If you want to load in Supermon or some other "reasonable" monitor and keep your total system then that's OK too.

The brutal speed of the Rebel assembler will entice you to test assemble often, insuring clean and efficient code. Rebel is a powerful RAM based assembler, but plenty of DISK options are provided as well. Rebel creates true disk object files and handles multiple linked source files with ease, even going from one drive to another.

The Rebel editor takes full advantage of Commodore's full screen editing capabilities, plus some helpful additions such as page flipping, scroll control, search and replace routines, and many others.

IMPORTANT NOTE: This documentation is not intended to instruct Machine Language to the beginning ML programmer. Beginners are urged to get a good reference book (such as Butterfield's introductory text) in order to indoctrinate them to machine level programming. REBEL! itself, on the other hand, because of its speed and ease of use, is a good starting point for the beginner, and at the same time, an execellent productivity tool for the seasoned programmer.

might go wrong with any code generated with this program. However, REBEL is quaranteed to work within the limits and following the instructions described in this documentation or any addendum made to it. If it does not, please contact Nu-Age Software at (813) 323-8389 for immediate replacement.

CHAPTER 1: Getting Started

Being a Two Pass Assembler, Rebel will have to examine all source code twice. The first pass creates the Symbol Table. This is where all of the labels that you use in your program are collected and stored. This process is what makes a label based or Symbolic Assembler possible. The Symbol Table is ultimately accessible to the programmer, but more on that later.

NOTE: The C64 and C128 assemblers are essentially identical to the user. Clearly, the actual object code is different, but usage is the same. The directives are identical with one exception. This exception is noted here so that you may assume any reference in the remainder of this text to directives is valid for either computer. The C128 assembler supports an IO directive called :X. This is used to cause a PRG file to be created which contains the results of the :Dump command (later). This file may be loaded or appended as any other source file would be.

Labels may be referenced before they are defined as long as they don't represent Zero-page addresses (0 - 255). Don't worry about why, just remember that Rebel will not be able to assume that you meant Zero-page instead of a "normal" address. The length of your labels is restricted to nineteen characters.

Let's look over some coding illustrating some things that Rebel recognizes as valid OPERAND data:

```
INDEX = $08          ;zero-page label
POINTER = INDEX+2 ; unused
;
;
* = $C000+500        ;assembly origin
;
START  JSR  SUBPRG
       BCS  ENDPRG   ;we're done!
       LDY  #$FF
YLOOP  INY
       LDA  (INDEX-3),Y  ;get byte
       STA  SPARE
       CLC
       ADC  $C000,X
       AND  49152,X
       EOR  TARGET+15,Y
       STA  ($2B),Y
       BPL  YLOOP
```

```
        STA  VARNAM+13 ;save value
 ENDPRG RTS            ;go away
 SUBPRG LDA  VARNAM+10
        CMP  'Z-64     ;screen code
        ADC  %01000000 ;64 DECIMAL
        RTS
 VARNAM :Reserve #15  ;create 15 byte table
 TARGET :R 20       ;32 DECIMAL
        :WRD SUBPRG-1
        :BYT 1,2,$A,B,'A,'B,>lab.name
 SPARE  = *
```

In the above example we see that Rebel has several types of operands to look at. There are labels, hexadecimal literal values, decimal literals and so on. All of these are acceptable to Rebel. The designation for each operand type is basically standard. The "#$" prefix tells Rebel that you intend the operand to be a Hex immediate value:

```
#$FE means 254 Decimal.
```

The "$" without the "#" means that the operand is a memory location:

```
JSR $FFD2
BASIC = $A000
```

Similarly, a decimal value without the "#" prefix means an address:

```
JSR 49152 ;we are the same
JSR $C000 ;we are the same
```

The "#" without the "$" means that the operand is an immediate decimal value:

```
#254 means 254 Decimal
```

The "'" tells Rebel that you want to use the ASCII equivalent of the character ("Z" in our example) as an immediate value. The "%" prefix means that you want Rebel to convert the operand to a binary literal as represented by the eight "bit" locations:

```
%11110011 = #$F2 = #242
```

Note that some operands are suffixed with "+" followed by a DECIMAL number. This tells Rebel to determine the value of the operand, then add the mini-decimal operand to it. Use the "-" symbol to similarly subtract from operands. The range of these decimal mini-operands is 0-65535.

Comments are preceeded by a semi-colon. A comment can be placed at the end of a line as shown or an entire line may be a comment as long as the first character of the line is a semi-colon.

Rebel tolerates the use of naked labels. Although the practice is not encouraged, it is supported:

```
SUBNAME
          LDA SOMETHING
          RTS
```

In the interest of standardization, the programmer is encouraged to semi clad Labels as follows:

```
SUBNAME = *
```

The * character is a valid operand. The following are examples of legal uses of the * operator:

```
LABNAME = *
LABNAM2 = *+2
;
        BNE *+12
        STA *-15
        STA *+3,X
```

The * operator represents the current assembly address. Note that no attempt has been made to actually alter the current assembly address using this operator. The following illustrate some ILLEGAL uses of this operator:

```
* = *+10
* = *-346
```

Here we see attempts to alter the assembly address. These will seem to work but when dealing with "mainline code" the real effect is chaos. See the :Reserve directive for the valid means of advancing the assembly address in the middle of real code. The * address is acceptably modified only when creating the definition part of your program. The following example illustrates this:

```
100 * = $c600 ;a definition place
110 label1 = *
120 label2 = *+1
130 * = *+100 ;actually $c600+100
140 label3 = *
150 * = $c000 ;actual code
160 begin lda whatever
170 ;don't modify the * anymore
180 ;unless you use :reserve !!
```

CHAPTER 2: DIRECTIVES

You are provided with several options for the output of "formal" assembly listings. You don't have to generate a listing at all, but you could choose to send the listing to the screen, the printer or to the screen and printer simultaneously. These options in no away affect the DISK options. You could decide to have Rebel dump a listing of the Symbol Table along with the regular assembly list. The dump lists each label in your source along with its real-world address. This option may be selected without having invoked either the screen or printer options, but output will default to the screen.

Before proceeding, a couple of rules must be established. All assembler directives are preceeded by a colon (":"). This is to let Rebel know that the data to be dealt with is a directive. Secondly, the "*" (asterisk) is used to declare the beginning address or "Assembly Origin". The assembly origin must be declared before the first line of any source that will be assembled, and must occur only once during the assembly process, with the exceptions noted in Chapter one. If no assembly origin is declared Rebel will default to the C-64 playground of $C000.

The :B(yte) directive is what we'll use when we want to emulate our big brother BASIC in dealing with raw data. The :B directive is the ML programmer's version of the DATA statement. As with BASIC, the data may be either string or numeric. Also as in BASIC, each data element must be seperated by a comma.

The :W(ord) directive is used when the programmer needs a mini version of :Byt that provides the two byte equivalent of a Label or address in Lo/Hi format. This is a rather specialized requirement, used for creating special tables or vectored addressing. There may be only one :Word assignment per line.

The :R(eserve) directive provides a means of reserving memory within the bounds of an ML program for things such as variables and tables. The argument may be either a decimal or hex value. This command assumes Hex if the # character is not used to indicate decimal.

The :E(nd) directive tells the assembler to STOP assembling at that point and to ignore all subsequent lines in the source. This is handy for doing partial assemblies (debugging, etc.), but be sure that the assembled part makes no references to labels after the :End directive.

The :S(yntax) directive tells Rebel that you only want to "play" assembler. Rebel will go through all the motions, but no generated object code will be placed into RAM. This command can be used in conjunction with the file handling commands (discussed later) to prevent any "live memory" conflicts.

The :P(rint assembly list) tells Rebel that you want a formal assemblylisting of your efforts directed to the printer on channel 4 of your C-64. The listing generated is standard, that is the labels are offset to the left of the mainline code and that everything is neatly tabbed over to create a pleasing block style list, with compensation for standard 66 line tractor feed paper. The assembler will print 60 lines followed by 6 blank lines for each page.

The :C(loseprint) directive tells Rebel to disable the :P(rint) command until the next occurence of a :P(rint) directive. That means that you have the capability of having an assembly listing of selected parts of your program. You could place a P(rint) command at the beginning of a subroutine and a ;C(loseprint) at the end of that routine. Rebel would detect the ;P command and generate a listing until encountering the ;C command. This process may be repeated as many times as you like.

IF you use the ;P(rint) directive without having your printer turned on, the listing will default to the screen and it will be printed very s-l-o-w. The effect is further complicated by having the ;V(ideo) directive (below) active. The listing to the screen in this circumstance would contain double characters for each character output, one for the ;V direct and one for the ;P command.

The ;N(ewpage) command is used to force the printer to space to the top of the next page. This is used when you want certain things on pages of their own.

The ;V(ideo) directive tells Rebel to place a formal listing of your assembled program to the screen. The screen listing will be formatted as though printing to an 80 column printer, complete with line skipping for paper perforations. You'll find this useful for spotting where page breaks should be etc.

The ;K(illvideo) directive is identical in function to the ;C(loseprint) command. The ;K directive is used to terminate the ;V(ideo) directive. The ;V directive may be used again to re-enable video output.

The ;D(ump symbol table) directive is used to force a listing of the symbol table. This option can be called without having invoked either the ;P or ;V options but the listing will default to the screen. To have Rebel print only a list of your labels you should use the ;D command somewhere in your source and place a ;P(rint) directive on the last line of the source.

Here's an example of the directives as they could appear in a listing.

```
* = $C000       ;set assembly origin
;SYNTAX ONLY    ;this is legal
;VID            ;select screen
;PRINT IT OUT   ;select printer
;DUMP           ;request symbol list
;
BEGIN   LDY #255
YLOOP   INY
        LDA DTABLE,Y
        BEQ ENDJOB
        JSR $FFD2
        BNE YLOOP
ENDJOB  RTS
        ;NEWPAGE

DTABLE ;BYT #13,#$0D,D,#0013,000D
       ;BYT 'A,$12,#234,'Z
       ;BYT %00001011
       ;BY  "ABCDEFGHIJ"
```

```
                   :B    "KLMNOPQRSTUVWXYZ"
                   :BYTE >LABEL1,<LABEL2
                   :BYTE 0
```

Note the free form of the directives used. Rebel only looks at the first character following the colon to determine the directive desired.

Of interest should be the varied forms of the :B statements. The first illustrates some of the ways to express the value thirteen decimal. The default base for any numeric :BYT element is HEX. Note that the decimal thirteens are preceeded by the "#" operator. In the second :B example we see some numeric values mixed with letters preceeded by the "'" operator. This is provided to allow the mixture of single alpha characters and numeric data. Note that the string data is on a seperate line of its own. Rebel expects strings and numeric data to be kept seperate, with the exception of single characters designated by the apostrophe operator. The data entered between quotes in strings will be honored identically as entered. That means that you can use strings just as you do in BASIC. If you include a "CLEAR HOME", for example, the printing of that character will act just as is it does in BASIC. Rebel further expects to have all numeric or "'" data seperated by commas with NO SPACES in between. Rebel will eccentrically take the space to mean "OK, everything from here to end-of-line is a comment or debugging effort". The :b statements containing the < and > characters imply that we can extract either the high or low bytes of addresses using :b. This is a rather specialized requirement, see :word for a more often used version of this type of requirement.

Here's an example of how the :K(ill video) and :C(loseprint) directives could be used:

```
      :PRINTHEADER
      * = $C000
      :S        ;syntax only
      :DUMPSYMS
      :CLOSEPRINTER
      :VISUALIZE PRG TEXT

      BEGIN   LDA DATA
              ORA MASK

      :KILLSCREEN ;kill video
      :PRINTSYMS ;enable printer
      :ENDJOB
```

We see here an example of where the programmer began the source file with the :P command (:PRINTHEADER). Rebel would generate an assembly list until encountering the :C directive (:CLOSEPRINTER). The desired effect is to have Rebel print out the initial header information of this source. The actual program code would not be printed, however, output would occur to the screen due to the :V command. At the end of the program body the screen display is disabled and output is directed to the printer for a printed list of the symbol table (:PRINTSYMS). Of interest may be the descriptive function names used here. Since Rebel looks at only the first character of each directive, any descriptor may be appended to them. This could help you to later remember the purpose for each occurence of these commands.

CHAPTER 3: Linked Files

Rebel can assemble more than one source file during a single run. This is provided so that you never have to worry about running out of user memory with large programs.

The first thing that must be done when dealing with multiple files is to have some way of of getting back to the first one. Since Rebel is a Two Pass Assembler, each source file will have to be examined twice. The assembler overcomes this irritation by use of the :O(riginfile) directive. Somewhere in the first file of the family a statement as below must appear containing the name of that same file:

```
     :ORIGFILE WHATEVER
```

Rebel will now know that when pass two rolls around that things began with the file entitled "WHATEVER".

The next step is to name the next file in the series. This is also simply handled. The last line of the file "WHATEVER" should contain the directive :L(inkfile) followed by the name of the next file:

```
     :LINK  VICTIM2
```

When the :L directive is encountered the next file is loaded. It is highly recommended that the :END directive be used to mark the end of the last file. At this point Rebel will look in its notebook for the name of the original file of the series. Since we previously used the :Org directive to name WHATEVER, Rebel will retrieve WHATEVER from disk and start away on pass two. Again it will load VICTIM2 when the :L directive is encountered.

Let's look at the last of the file handling family, the :F(ile) directive. This directive is used when you want to create a file on disk of the object code created during the assembly process. The file created will be the type that you load with ",8,1" appended to the end. The object file created will load to the address specified by the assembly origin. The real value of the :F directive is realized when you want to assemble to an address where something else is already resident (like Rebel!!). It certainly wouldn't do to have the assembler assemble your Penguin Patrol game over the top of itself. Things would quickly get out of hand.

The :File directive can be used in conjunction with the

:Syntax directive to force the assembler to create a real file without actually assembling to memory. If used without the :Syntax directive, the assembler will still create your file as well as writing to RAM. The :File directive may also be used when working with linked files.

IMPORTANT NOTE: The :F statement MUST NOT come before the assembly origin declaration.

Let's review the file family with an example:

(first file)

```
     * = $C000        ;first file only!!
     :ORGFIL WHATEVER,8  ;name of this file
     :FILE OBJ,8      ;file to be created
      BEGIN LDA 'A
      JSR $FFD2
     ;
     :LINK VICTIM2,8 ;next file in series
```

(second file, VICTIM2)

```
     LABEL LDA DATA  ;code etc.
     ;
     ;LDA 'B
          JSR $FFD2
          RTS
     ;
     :END                ;this is the last file
```

Note that the assembly origin is defined ONLY in the first file. Further note that the :Originfile is declared only in the first file. Rebel knows where things are. Don't confuse the issue by redefining the assembly origin. Think of these linked files as being one big file. further note that a device number is included in each filename. This means that two drive users can mix and match source and destination drives.

The result of the two little files would have been assembled to the file OBJ and also to memory at $C000 (49152) because the :Syntax directive was not invoked. No listing would have been generated since :D, :V or :P were not selected .

If you've already begun experimenting you might have noticed that Rebel always displays an asterisk at the beginning of pass two. The asterisk is a flag when working with linked files. It means that the assembler has a source file in its teeth and is actually assembling. The * does not appear until each file is completely loaded. In RAM based assembly, the * doesn't mean a thing.

<u>CHAPTER 4: Source files and the Editor</u>

This chapter deals with the C64 Editor. See the section on the C128 for that computer.

Source files are created and maintained like BASIC program files. That means that you will be loading and saving these files using the BASIC ROM routines. You will also be entering them just like BASIC programs. Although the source files are input by the BASIC file editor they are not crunched. They are input into memory exactly as they appear on the screen, with the exception of the line numbers, they are crunched into the usual two byte binary values. This almost precludes the use of REBEL for entering BASIC programs, however, pre-written BASIC programs can be loaded and RUN as usual. You'll further find that **BASIC is still active in the DIRECT MODE.**

There is a way to use Rebel to enter BASIC source code. Any line number that is preceeded by zero will be entered as real BASIC. The C128 version Editor has a facility for disabling/enabling the non-crunching routines altogether. See the section on the C128 Editor for more details.

NOTE: Rebel does <u>not</u> support multiple statements per line.

IMPORTANT: Your source files must be entered via the Rebel editor. Since BASIC crunches input lines, certain opcodes and any label you use that contains a crunchable BASIC word will be crunched as though they were BASIC keywords. Rebel will not be able to recognize some of these mutilated strings as valid assembler data.

Since BASIC is used to input lines it follows that BASIC is used to SAVE, LOAD and LIST these lines as well. As you well know the C-64 lists lines faster than any mere mortal can read them. This problem is overcome by a wedge. This wedge lets you list data and have it freeze when it hits the bottom of the screen page. The depression of LOGO and F1 at the same time will activate the freeze mode. The **left arrow key** (upper left of keyboard) will cause the frozen screen to be cleared and the listing will continue again from the top of the screen to the bottom and freeze again. This allows you to view your lines by page. The depression any other key except STOP, will allow lines to be scrolled up one at a time. If the stop key is pressed while

the screen is frozen the LIST will end with the cursor at the end of the last line. The familiar "BREAK" message will not appear. When the freeze mode is active the screen background and border colors will be automatically made different. When disabled by the depression of the CTRL and F1 keys the border will switch to match the background.

When the freezer is disabled, another convenience is enabled. Bi-directional scrolling. The cursor up/down keys now become vehicles for automatic up or down list scrolling when the cursor is either on the top or bottom line of the screen.

NOTE: NEVER attempt to use scrolling when doing a regular LIST or when accessing DISK. The automatic list-scroller will create havoc if this is attempted!!

An alternate screen page is provided (C64 only, see C128 Editor). The depression of the SHIFT key and F7 together will cause the C-64 to flip to the other page. The alternate page is automatically cleared before the very first usage. The alternate page may be used anytime. Both pages are useable not only for viewing but for text editing as well.

You can change the background color of the screen by pressing LOGO and F7 together. For each depression the background color will be incremented by one.

The F3/F4 function key causes a tab/rvs-tab. F3 is a forward tab and F4 is a reverse tab. You'll find these useful when the special lister is active, or when working with Supermon.

The next group of extensions are used for actual data manipulation and related tasks. Rebel provides a powerful group of commands that make text manipulation a breeze. You can renumber, delete, move lines around, search and replace, insert lines, append files, selectively save parts of your work file and more. Let's move on and look at the commands available.

The **A** (append) command is used to append a file from disk to the work in memory. When coupled with the S command (described later) this command provides for the easy writing of new programs from existing and proven ones.

The ability to search and replace is provided by the **B** (buffer) and **H** (hunt) commnds. The Buffer command is activated by typing the "B" character and pressing RETURN. As with all Rebel commands, only the first character of a word is looked at. The computer will respond with the familiar question mark and flashing cursor associated with inputs. Simply type the data string you want to find (up to 20 characters) and hit RETURN. Nothing will happen other than an "ok" prompt. What you have done is placed the target string into a buffer. This string will stay there under most conditions. That means you can input lines, assemble or anything and still have that data intact in the search buffer after an actual search.

The **H**(unt) command is used to tell the editor to search your entire source file for a literal match to the characters in the search text buffer. Just type the "H" character and press RETURN. The editor will search your file and list any lines containing a match. The lines displayed are allowed to be edited as usual.

Next up is the **C** (change) command. We'll use this for our global search/replace command. The command is self documenting. When invoked the editor will prompt you along.

Next is the **D**(elete) command. This is used to delete lines. To activate it simply type "D" and press RETURN. The question mark prompt will appear. The editor is waiting for you type the line number of the first line you want to erase. Type the number and press return. Another question mark prompt will appear. Type the number of the last line you want to delete. The editor will trot off and erase these lines. The purpose for having two prompts for the line numbers, for that matter having any prompts at all is for the sake of security. If you've ever used a delete command that expects all data to be entered along with the command then you've experienced the agony and despair of having errantly entered the numbers of lines you didn't really want to delete. Delete gives you two chances to change your mind. If either line number entered is not an exact match of an existing line number the message "UNDEF'D STATEMENT ERROR" is displayed and the delete routine aborts. I like to use the number 99999 since there is no chance of ever having a line number that big.

The F (file list) command is used when you want to list a file from disk. This means that you can view another source file without having to LOAD it. The F command may be used to list sequential files as well. Just tack on ,s,r to the end of the filename. The directory of a disk may be similarly viewed by simply entering $ as the filename.

The I (insert) command is used to insert blank lines into your source file in memory. The lines will contain only a line number and a semi-colon. This is provided as an alternative to an AUTO command, not present in Rebel. AUTO commands are irritating because they tend to eat lines.

The M (memory free) command is used to simply report the amount of source memory remaining.

The next command is the R command. This command will cause the editor to instantly renumber your file. The first line number will be 100 and each successive line will be 10 higher.

The S (selective line save) is used when you want to save a selected part of your source program as a seperate file. Just enter the starting line number when prompted, the ending line number when prompted and finally the filename you want to use.

The T (xfer lines) command is used to move lines of source around within the source file in memory. I find this most useful for attempts at structuring my programs. The utility will prompt you for the beginning line of the source area, the ending line and finally the destination. The destination line must already exist. That isn't to say that you must create a dummy line for this to have any place to go. The transferred lines will insert themselves in between the destination line and the line right before it. Think of this as an ambitious insert. The source file will understandably be renumbered before this routine exits.

The X command is used to "unnew" after a BASIC NEW. This is most useful after a reset, or inadvertant real NEW.

The next command is the * command. This is how you invoke the assembler. When you're ready to assemble something just type the "*" character and press return. The assembler will attack your source and return control to the editor.

The # and $ commands are used to invoke the decimal to hex and hex to decimal conversion routines. If you enter the # command the computer will respond with the question mark prompt. At this point you should enter a decimal number. The computer will report back the hex equivalent of that value. The $ command expects a hex value. The hex value should contain four digits, i.e., the hex value $FF should be entered as 00FF. The computer will return the decimal value #255 for this particular entry.

The ' command is used when you need to know the HEX or Decimal equivalent of a charcter. The format is 'A, where A is any character that can be displayed in direct mode. The editor will report back the HEX and Decimal equivalents of that ASCII character.

CHAPTER 4.1: C128 Editor

The C128 Editor is designed with a heavy predjudice toward the 80-column screen. That isn't to say that you can't use the 40-column mode. It's just that, well, you'd be better off using the C64 Editor for convenience sake. All of the following direct commands are supported but the handy windowing stuff is not supported. Clearly, one would be happier utilizing the more powerful C64 version 40 column Editor that supports page flipping, scrolling etc.

It is assumed that the user reading this text is either preparing to use the 80 column Editor or soon will be. Before proceeding I should point out that the FAST mode of the C128 makes things nicer to work with.

The C128 Editor contains most of the direct commands found in the C64 version. The #,$,' and Delete commands are gone. These functions are duplicated in either BASIC or the MONITOR. All of the other direct commands are present. Some of the names have changed, but they're all still there.

Since most of the direct commands of the C128 Editor are repeats of the C64 version, we'll concentrate on those that are either different, or new.

First up is the E (edit basic) command. This is used to disable the "no-crunch" feature of the fundamental Rebel Editor. In short, you can write basic if this option is used, but don't attempt adding Rebel lines.

The F (find) command is used to search the source for all occurrences of the Buffer. This is identical to the H (hunt) command in the C64 version.

The H (help or restart) command is used to get back to Rebel "proper". This is useful when you're not sure what mode you're in or when you have to apply the dreaded RUN/STOP RESTORE key combination.

The L (lister on) command is used to enable the LIST format function. All labels are displayed in white and all "mainline code" is displayed in black and everything is neatly displayed in "block" format.

The M (menu) command re-draws the screen boxes without changing anything else. This is useful for utilitizing the split windowing features while editing BASIC, for example.

The O (lister off) command is used to disable the LIST formatter. You'll want this when editing BASIC, or for whatever reason.

The C128 80-column Editor uses two 39-column screens at the same time for text editing and LISTing. This is accomplished through the use of two WINDOWS. These WINDOWs are totally independent screens as far as the EDITOR is concerned. That is, anything you do in the left WINDOW doesn't affect anything in the right WINDOW. All line links are maintained seperately so that you can edit text in either side without scrambling the lines.

To go from one WINDOW to the other you will be using two grey cursor keys at the top of the keyboard. The cursor-left key toggles from one window to the next. The cursor-down key moves the cursor to the bottom of the current window. These keys should only be used when not outputting to the screen. That is, if you're LISTing something don't use these keys. Instead, use the "w" key. This causes the EDITOR to toggle the WINDOW. The left-arrow key, found next to the "1" key is used to clear the screen when LISTING and the LISTER is active. The LISTing will continue down from the top of the current WINDOW and freeze again at the bottom.

The C128 Function keys are unused by REBEL. Feel free to use them for whatever you like.

Chapter 5: C64 Assembler Error Handling

The Rebel Assembler has built in error trapping routines for things like mispelled opcodes, undefined labels, duplicate labels, illegal addresses and so on. The error messages displayed are taken from the table of error messages in ROM. The following list shows all of the possible messages Rebel could display during actual assembly:

?OUT OF DATA ERROR
?SYNTAX ERROR
?ILLEGAL QUANTITY ERROR
?UNDEF'D STATEMENT ERROR
?VERIFY ERROR
?CAN'T CONTINUE ERROR
?FILE OPEN ERROR
?FILE NOT FOUND ERROR

When Rebel fires off an error message you will usually be able to immediately determine what has gone wrong. The offending line will be displayed followed immediately by the line number on the next screen line. The error message will be displayed on the next screen line after that. Again, most errors will be self explanatory. Let's start with the "?ILLEGAL QUANTITY ERROR" message.

Rebel will generate the quantity error whenever it finds that an operand of any type is out of range for its associated opcode. This error message is also used to identify illegal relative branches as well. The reasoning here is that the 6510 may only execute branches within a limited range. A reference to an address past the limit results in a bad operand quantity. Rebel will also generate this error if you try to use immediate addressing with operands greater than #255. The quantity error will also appear if you enter a bad decimal number, like #12F for example.

The "?SYNTAX ERROR" message is generated when you do things like mis-spell opcodes. You should have no trouble with this one, most of the time. A snag develops if you attempt to use the single letter "A" as an operand. The assembler takes this to mean accumulator mode addressing. Look at the following examples:

```
     LDA A ; this won't work
     ASL A ; this is ok
     ROR A ; so is this
```

Clearly, the ROR and ASL instructions support accumulator mode addressing where LDA does not.

The "UNDEF'D STATEMENT ERROR" pops up when you attempt to reference a non-existant label. This corresponds to the error generated when you use the GOTO command in BASIC and reference a line number that doesn't exist. The most common circumstance for this error is the mispelling of the operand. The UNDEF'D will also occur if you try to use one the reserved 6502 mnemonics as a label:

```
     LDA  EOR #15 ;this won't work!
     LDZ  EOR #15 ;that's better,..
```

The first line of the above example would cause Rebel to believe that LDA was the actual opcode and EOR the operand. A search of the table containing your labels would not turn up "EOR", hence the error message.

The "?OUT OF DATA ERROR" will surface when dealing with :Byt statements. The assembler expects all data elements to be seperated by commas, but by only one comma. Double commas will force this error. Also, if the last element of a :B statement is followed by a comma or the first element is prefixed by a comma, the error will be generated. Simply put, all numeric :B elements must be seperated by commas and don't use leading or trailing commas. The following illustrates some possible errors with the :B statement:

```
TABLE :BYT ,1,2,3,4
      :B   1,2,3,4,
      :B   1,2,3,,4
```

The errors in sequence are (1) leading comma, (2) trailing comma and (3), double comma. One final trap with the :B statements concerns the use of comments and spaces. Rebel doesn't allow comments on numeric :B lines. further, the elements must not be seperated by spaces.

The "VERIFY ERROR" pops up when Rebel discovers that you have used a label twice. If an entry is discovered that matches the newcomer, the "VERIFY" message is generated followed by the offending line and linenumber of the latest attempt to define that label.

The "CAN'T CONTINUE ERROR" is generated when the stop key is pressed during the assembly process. This really isn't an error as the stop key has been purposely left functional. If you are using linked files or are writing an output file these files will be closed. Clearly, the incomplete output file should be scratched.

Rebel will not generate a warning at the end of assembly if you attempt to use ZPAGE,Y indexing. This mode is not supported by your microprocessor. However, it is perfectly legal to use Y as an index on any absolute address. Consider the following:

```
C000 ?? ??      STA $02,Y
C002 99 02 00   STA $0002,Y
```

We see here that the only true difference between the illegal ZPAGE,Y and legal ABSOLUTE,Y modes are the number of bytes required for the operand. Rebel will spot any attempt at ZPAGE,Y and and assemble as though the intent were ABSOLUTE,Y. Assembly will continue as normal, but the object code will be adjusted to be STA $0002,Y.

The "?FILE OPEN ERROR" will be issued if you attempt to create an object file to disk when a file of the same name already exists.

The "?FILE NOT FOUND ERROR" will pop up if you attempt to :Link a non-existant source file.

The "Errors That Drive You Crazy" Section:

There are some errors that might be committed that Rebel will not spot. These errors are typically ones where there is no obvious syntax error or the programmer has chosen the use of an addressing mode that doesn't do what he thinks it does. You may know these errors as "logic flow errors."

A situation that could pop up from time to time concerns the use of the single character "A" as a variable or label name. Rebel reserves the single letter A as a flag to mean Accumulator mode addressing. Consider the following:

```
        X = $02
        Y = $04
        A = $0888
        ;
        BEGIN LDA X ;no problem here
              ADC Y ;still ok
              ROR X ;still ok
              ASL Y ;no probelm
              LDA A ;SYNTAX ERROR
              ROR A ;nasty!!
```

Note the comment following LDA A. Rebel knows that this is an invalid use of the accumulator mode. Where things get nasty is when you attempt to use a legal accumulator mode opcode. The programmer thinks that the variable A is being ROR'd in this example. This is not what the assembler will think! Rebel will take the statement ROR A to mean ROR " the accumulator". Assembly will continue on as if nothing in the world were wrong. Clearly, this could be a nasty, nasty bug to find.

My favorite (?) exasperation involves the use of linked files that are, well, improperly linked. It is

REBEL ASSEMBLER/EDITOR QUICK REFERENCE CARD

C64 Assembler Command Summary

!byte
!closeprinter
!dump labels
!end
!file name,dev.#
!killvideo
!link filename,dev.#
!newpage
!originfile name,dev.#
!print (*device 4 only*)
!reserve
!syntax
!video
!wrd

C128 Assembler Command Summary

!byte
!closeprinter
!dump labels
!end
!file name,dev.#
!killvideo
!link filename,dev.#
!newpage
!originfile name,dev.#
!print (*device 4 only*)
!reserve
!syntax
!video
!wrd
!xfile name,dev.#

C64 Editor Command Summary

Append
Buffer
Change
Delete

File list (*device 8 only*)
Hunt
Insert

Memory remaining

Renumber
Selective SAVE
Transer lines
Xnew (unnew)

```
LOGO  f1  freezer on
      f3  lister  on
      f5  plotter on
      f7  background color
CTRL  f1  freezer off
      f3  lister  off
      f5  plotter off
      left-arrow key cursor bothome
SHFT  f3  rvs-tab
      f7  flip page
----  f3  tab forward
  restore reset editor
  crsr up  scroll if freezer off
  crsr dn  scroll if freezer off
  left arrow key continue list from
       home when frozen
$   convert from HEX
#   convert from DEC
'char convert ASCII char.
```

C128 Editor Command Summary

Append
Buffer
Change
Disk list (*device 8 only*)
Edit BASIC
Find
Help (lister on, redraw frame)
Insert
Lister/freezer on
Menu redraw
Off (lister/freezer)
Renumber
Selective save
Transfer lines
Xnew (unnew)

Grey cursor keys:

```
up     background color
down   cursor bothome
left   switch to other side
right  cursor right
```

when freezer is on and a LIST is is actually frozen:

left arrow key continues LIST from the top of the page.

w key continues list from top of the *other* page.

*** = start assembly**

N O T E S