THE MICRO WORKS

MACRO-80C

USER GUIDE

P.O. BOX 111 - DEL MAR, CA 92014 714-942-2400

MACRO-80C

Table of Contents

MACRO - 80C
USER GUIDE

MANUAL AND SOFTWARE
WRITTEN BY
ANDREW PHELPS
THE MICRO WORKS

TEXT EDITOR, ASSEMBLER, AND MONITOR

Owners Guide Manual

A Disk Based Macro Assembler for Radio Shack’s Color Computer

From The Micro Works, Inc Written by Andrew E. Phelps

Copyright Notice:

THE MICRO WORKS, INC. Mailing Address: PO Box 1110 Del Mar, CA 92014

UPS shipping address: 1942 s. El Camino Real Encinitas, CA 92024

Redaction and revision activity done by Pete Willard - end goal; clarity.

INTRODUCTION

This manual describes the Editor, Assembler, and Monitor disk from The Micro Works. The operation of the editor and monitor are described in detail~ the use of the macro assembler is described in another manual which also accompanies your disk.

On the disk you will find several files. The files `ED/BAS`, `AS/BAS`, and `ASP/BAS` are Basic programs which you will run in order to use the editor and assembler. `DCBUG/BIN` is a binary file which you load with the `LOADM` command and run with `EXEC`. The files with names ending in `/TXT` are example programs which you can assemble and edit. The files `AS%/BIN`, `ED%/BIN`, and `HLP%` are used automatically by the assembler and editor.

Do not write (save data) to the supplied disk! If any files are added to the disk, there is a possibility that the directory will be incorrectly written and that the disk will no longer be usable.

SECTION I - THE BASICS

RUNNING THE EDITOR

This is a disk-based screen-oriented text editor. Since it is disk-based, it is run from a disk and used to write text files to a disk and to edit text files which are on the disk. Since it is screen-oriented, there are no line numbers in the text file and editing is done by positioning the cursor in the file by means of the arrow keys.

To run the editor, type `RUN "ED` and press ENTER. This will load and run a small Basic program, which in turn will load the binary file `EDT%/BIN`. You will be asked *"EDIT WHAT FILE?"*. If you are going to be making changes to an existing file, enter the filename, otherwise just hit ENTER and you will begin with an empty file. If you wish to edit a file which is on tape, it must first be transferred to disk. Refer to the section on program `XFER`.

Once you are in the editor you will be in the command mode. There are two numbers at the top of the screen; the one in the upper right corner shows how much free space there is in memory, and the one to the left of it shows how far (in characters) from the top of the file you currently are. Since you are right at the top of the file when you first enter the editor, it shows dashes instead of a number.

To enter text into the file, hit I and you will be in line insert mode. Hit BREAK when you want get Back into command mode. To alter existing lines, hit X and enter the "eXchange" mode. Hit ENTER to return to command mode from this and all other modes. Use D for "Delete" to remove lines. You may move throughout the text file by using the arrow keys, or you may jump to specific points in the file by using Find, Search Symbol, or Jump commands.

When the file has been edited, hit BREAK. You will be asked if you are done editing. Just hit ENTER, and your file will be written out to disk. If you have hit BREAK by accident, enter "NO" and you will return to the editor.

At any time while in the editor, you may press the CLEAR key and this will display (from disk) all your options at that point. The CLEAR key is therefore called the **HELP** key, and there is a section describing its operation below.

If any key is held down for more that half a second, it will start repeating. This is called typamatic or auto-repeat. The speed at which it will repeat depends on which key is being pressed and in which mode.

WHAT YOU CAN EDIT

This editor is designed primarily for the assembly language programmer, and its features tend to reflect this. It can be used, however, to edit any text file, including letters, sequential data files, and even Basic programs saved in ASCII format.

To edit a Basic program, create the file using `SAVE"filename",A` and this will save it in ASCII text file format instead of Basic Token format. When editing the file, it is your responsibility that every line ends up with a line number and that the line numbers are in order. Actually, if the line numbers

are out of order, the lines will be sorted when the file is next loaded by Basic.

The best use of the editor, however, is with assembly language source files. The line oriented features, such as **Find**, are designed with line oriented assembly source in mind. The **Search Symbol** command is specifically for finding your way around an assembly file. The tabs are permanently set for 6809 assembly source files.

THE HELP KEY

Whenever you are in the editor, you may press the `CLEAR` key to ask for a display of what may be done while in that mode. This is especially useful while learning to use the editor, but will continue to be useful for seldom-used modes such as **Move** or **Copy**.

The **HELP** function works by reading a portion of file HLP% from disk. If this file is not present, or if there is an error in reading it, the **HELP** key will have no effect. If you do not have room on a certain disk for the `HLP%` file, you may delete it without causing any problems.

If, while in **HELP**, you press **HELP** again, you will get a description of the commands available while in **HELP**. These include `BREAK` (to exit **HELP** mode), `ENTER` (to go to the next page if there is one, or else exit **HELP** mode), and the up and down arrows to see all of the **HELP** pages.

HELP is not available before first entering command mode, or after using `BREAK` to leave command mode, as you are in the Basic program at that time and not in the actual editor.

COMMAND MODE

When you first enter the editor, you are in command mode. The upper left corner of the screen says "EDITOR" to identify this mode. In this mode, there are certain keys which you can press to get into another mode (such as `I` for insert or `D` for delete) or to perform some simple action (such as the arrow keys to move the cursor).

This is the complete list of keys which you may press in command mode, and the effect of each one:

KEY	Function
I	Insert mode
L	Same as I (for "Line Insert")
D	Delete mode
X	eXchange mode
F	Find a symbol or text string
B	Backwards find (ie, search upward)
S	Symbol search (search for symbol at left margin)
P	Jump one Page forward
0	Jump one page backward
C	Find and Change string

KEY	Function
T	Make Two copies of block of text
M	Move block of text
J	Jump to beginning or end
A	Repeat last find or change
← and →	Move cursor left or right
↑ and ↓	Move cursor up or down
SHIFT + ↑	Move cursor to top of screen
SHIFT + ↓	Move cursor to bottom of screen
SHIFT + ←	Move cursor to left margin
SHIFT + →	Move cursor to right end of text on line
ENTER	Move cursor to next line
SPACE	Move cursor right, skipping groups of spaces
BREAK	Exit the editor

As in Basic, SHIFT + 0 is the upper-lowercase toggle. Lowercase mode is indicated by an **L** between the numbers on the top line.

INSERTING TEXT

To add text to a file, use Line Insert mode by typing I from command mode.

Position the cursor to the place where you want the inserted text to start. When starting with a new file, the cursor can be anywhere on the screen, though it is usually at the bottom where it originally appears. It is possible to start inserting in the middle of a line, such as after an assembly-language label, by positioning the cursor on the first character before which you want to insert.

To enter Insert Mode, press I.

At this point, you simply type in text. The following are text keys:

- Letters, Numbers, and Symbols (the gray keys)
- ↑
- SHIFT + ↑ ; which is a back arrow or underline
- ↓ ; Which is used for the end-bracket **]**
- SHIFT + ↓ ; Which is used for the begin-bracket **[**
- CLEAR ; which is the backslash key

There is only one change from the standard Basic key assignments. The end bracket ("]") has been moved to the down arrow key. This was done to simplify things, because now the up and down arrow keys are text keys (whether shifted or

not) and the right and left arrow keys are control keys (whether shifted or not). And the brackets (which are used a lot in assembly language programming) are neatly on one key (the down arrow) where they can be easily remembered.

In Insert mode, the following may be entered:

- Text Keys ; (see above) These get entered into the file
- ← ; Backspace and Delete one character
- SHIFT + ← ; Delete the current line
- → ; Tab to column 8 or 14 (for mnemonic or operand entry)
- SHIFT + → ; Has no effect
- ENTER ; Go to the next line
- BREAK ; Exit this mode
- CLEAR ; Help

The SHIFT + 0 key combination will set or clear lowercase mode as always.

TAB stops are not designed to be set by the user. They are set by the editor to be 8 or 14 columns. These are presets fpr entering assembly language mnemonics or operands. They are not entirely necessary as the Macro Works Assembler will automatically insert a tab at the correct column. TABS in the source file will take up space in the file, but will not be displayed. If you want them, they are available

The BREAK will exit Insert mode and return you to command mode. In most other modes, the ENTER key finish the operation but in Insert mode, it will finish each line. This may seem arbitrary, but it is done to make it easier to perform data entry the way commands are entered into a computer program.

Lines longer than 32 columns may be entered simply by continuing to type after the end of the line. A black colored block will appar at the start of the next line to indicate a con±inuation.

ALTERING TEXT

To modify text on the screen, use the X command. The D command is used to delete lines or groups of lines. Global modifications, (repetative changes) are done with the C command. The X command is described here, and the D command follows. The C command is described in the section on "Global Changes".

To alter some text on the screen, place the cursor there and hit X. Any text typed in this mode will simply be written over whatever is on the screen. The definition of the text keys is the same as given in the Insert section above. The control keys operate as follows:

- ← ; Move the cursor left one character without altering text

- → ; Move the cursor right one character without altering text
- SHIFT + ← ; Delete the character to the left of the cursor, move text to the right
- SHIFT + → ; Insert a space at the cursor position, move text to the right
- ENTER ; Back to Command mode
- BREAK ; UNDO. Restore text what is was prior to the X command
- CLEAR ; Help

To delete a line or lines, place the cursor at the beginning of the text to be deleted and type D. While in delete mode, pressing the ↓ key will delete a line, and holding it down will delete a series of lines. Use ENTER to get back to command mode, or BREAK to recover lines which were accidentally deleted.

Besides the **Help** key, the only other control key which has effect in delete mode is the → key. This will delete the next 32 characters, and so can be used to delete part of one line and part of the next.

SEARCHING AND FINDING

Positioning the cursor within the file is done both by moving it with the arrow keys and by jumping around with the F, B, S and J commands. Typing S allows you to jump to any symbol definition in an assembly language program. F and B search forward and backward for some word or text string: J allows you to jump to the beginning or end of the file.

To find the line on which a label (say for example **LABEL1**) is defined, first type S. The cursor will jump to the top line of the screen. Now type **LABEL1** and hit ENTER. You will now be on that line. If there is no **LABEL1** in the first column, you will end up at the end of the file.

If you ask for **LABEL1**, a line starting with **LABEL2** will not be found but a line starting with **LABEL1**= will be. This is because the symbol name must end with a non-alphanumeric character.

The Find command is used by typing F and then entering a word to be found. The cursor moves to the top line for entering this word. The cursor will then be placed at the beginning of the line which contains that word. The editor searches for the first occurance of that word which is after the line which contained the cursor when F was hit.

In the usual form of the command, the "word" to be found is any string which is bounded by non-alphanumeric characters. What that means is this: If you tell the editor to find **NERF** it will not find **NERFOID** and it will not find **99NERF** but will keep going until it finds **NERF**. This is particularly useful when looking for an assembly language symbol.

If you wish to defeat this feature, and find all occurances of **NERF** even if it is in **NERFOID**, etc., tell it to find **"NERF**. This is called literal mode. There is no ending quote mark entered: there is just the initial quote (which is SHIFT + 2) at the beginning of the line.

- F ; Find
- B ; "backward find", to search toward the start of the file instead of toward the end
- J ; "Jump". Another letter is typed to specify where to jump
- J + B ; will jump to the beginning of the file
- J + E ; will jump to the end
- JF ; will jump to the beginning and then go into **Find** mode
- JC ; will jump to the beginning and then go into **Change** mode, but that's another story

After the line is found, hit A to find the next occurance of the same word. A stands for again and will repeat the last find or change. Pressing A repeatedly will find all lines which contain a certain string.

GLOBAL CHANGES

Suppose you want to change every occurance of the word **RONG** to **WRONG**. Start at the top of the file, and type C for change. The cursor will jump to the top line. Now type **RONG** and hit ENTER. The cursor will stay on the top line, and you type **WRONG** and hit ENTER.

This will cause the editor to search forward through the file until it encounters the word **RONG**. It will change this to **WRONG**, and then stop with the cursor on that line. This gives you a chance to look at it if you like, and make sure that the right thing is occurring.

Now hit A. This is the **again** command. The editor once more searches for the next **RONG** and changes it. Hit A again and it will do it again. Hold down the A key, and the typamatic will cause it to continuously get A entries and the editor will keep searching and changing as long as there are more **RONGs** in the file. When you get to the end, you will see the screen stop at the bottom of the file.

The rules for searching are the same as those for the **FIND** command. If you try to change **RONG**, then the word **RONGER** will not be changed. Likewise, ,the word **WRONG** would not be changed to **WWRONG**. If you wish to change some literal string, however, regardless of whether or not it is set off by non-alphanumeric characters, then start the first string with a begin quote **"**. Change **"RONG** to **WRONG** will also change **RONGER** to **WRONGER** as in the example above.

To make several changes in one area of the text file, start the cursor just above the area in which the changes are to be made. The change command always searches forward through the file. If you really do wish to change the entire file, type J + C (jump change) and the editor will jump to the beginning before making the first change.

THE WILD CARD CHARACTER

When using the **Find** or **Change** commands, it is possible to specify a character which will match any character in the string being searched. This "Wild Card" character is obtained by hitting the → while entering the string to be searched. A question mark will show on the screen, but is shown in reverse video to distinguish it from a real question mark. In the example below it is shown as a question mark, but don't let that fool you.

Suppose you want to find the next `FCB` or `FDB`. Type F and then enter the string "F?B" to find any three letter word starting with an **F** and ending with a **B**. Another example: To change all of the words **NERF1**, **NERF2**, etc. into **NERF**, use **Change** to change **NERF?** into **NERF**.

MOVING AND COPYING

The **M** command is used to move a block of text to another position in the file. The **T** command makes two copies of a block of text. If two copies of some text are heeded in separate places in the file, use T to duplicate it and then M to move the second copy to where it is needed.

When the text to be moved can fit on the screen, position it so that the first line to be moved is the top line on the screen. Now move the cursor to the line below the last line, to be moved.

Now hit M. In Move mode, the up and down arrows move the block. The down arrow moves the block down~ it does this by leaving the block fixed on the screen and scrolling the rest of the file up past it. Similarly the up arrow key is used to move the block up through the file.

If the block to be moved won't fit on the screen, don't despair. It's almost as simple. Once again, start with the first line to be moved on the top line of the screen. Hit M. Now use shift down arrow to move the text up until the cursor is on the line below the last line to be moved. Now you can use up and down arrow (without the shi£t) to move the block around as before.

Hit ENTER when the block has been moved to its destination.

To make two copies of a block of text, position the first line to be copied on the top line of the screen. Hit T. Now use the down arrow key to move the block to be copied off the top of the screen. Each line which leaves the screen will be stored in each of two blocks of text. Hit ENTER when all lines have been copied.

SAMPLE SESSION

In this section, we will run step-by-step through the creation of a small program.

First, we will run the editor. Type `RUN"ED` and hit ENTER. If you get an error, type `DIR` and make sure that the file `ED/BAS` and `ED%/BIN` are there.

The editor will ask *"FILE NAME? "* Just hit ENTER. You should now be in command mode with the screen blank except for the top line. Hit the I key. You are now in Line **Insert** mode. At this point you can type in data.

For 'this example, type a space and then type:

```
LDA #'!
```

Now hit ENTER. You are ready for the next line. Type a space and then type:

```
JMP $A282
```

Hit ENTER again. You now have your whole program entered. If you made a mistake, use the ← key to back up to it and correct it. When it is correct, hit BREAK to get back to command mode.

At this point, you may want to practice positioning the cursor at various places on these lines. Use the ↑, ↓, ←, and → keys, and also try holding SHIFT while you use these keys. ENTER and the SPACEBAR will also move the cursor around.

Try positioning the cursor over the letter "L". Now hit X to go into exchange mode. Hit some other letter, like "Q", so that the line reads `QDA #' !`. Now hit ← once, and hit L, so that the correct line is back again. Hit ENTER to get back to command mode.

Hit BREAK. This exits the editor. You will be asked *DONE?*. If you had hit BREAK by accident, you would reply **NO** and be back in the editor. Since you are done, just hit ENTER. You will be asked for a filename, since you didn't give one before, so type **MYFILE** and hit ENTER. After a few seconds of disk access you will be back to the Basic system prompt of **OK**.

Congratulations! You now have a text file. You may list it to the screen using a program such as FILELIST (supplied with the editor), or assemble it with an assembler. You may edit it again by typing `RUN"ED` and, when asked for a filename, reply **MYFILE**. The full name of this file is **MYFILE/TXT**, and this is how you would write it when using a command such as **KILL** or **COPY**, but the **/TXT** is not needed when you are using the editor.

WHEN MEMORY IS FULL

The number in the upper right corner of the display indicates the number of free bytes of memory. When this number approaches zero, some lines of text should be deleted. If this number becomes zero, you will not be allowed to enter certain modes such as **Line Insert** or **eXchange**.

Do not attempt to operate the editor extensively when memory is very close to being full. It is possible, once in a mode such as. eXchange, to make changes which will fill memory and cause errors in the source file. If memory is near full, your program or text should be split into two files and edited separately. Duplicate the file using Basic's COPY command and then use the editor to delete half of each file. The MACRO-80c assembler has the ability to read and assemble a source program which has been split into two or more files.

TEXT PROCESSING FROM BASIC

When faced with a really major repetitive change to a text file, don't overlook the possibility of doing it with a Basic program. Here is a hypothetical example of such a problem and a Basic program to make the changes.

Suppose you are writing an assembly language program which contains a large data table. This table might look something like this:

```
TABLE1  EQU *
        FCC "NERF"
        FDB HITHER
        FCC "NERBLE"
        FDB THITHR
        FCC "STUFF"
        FCB YON
```

And so forth. Now, suppose you want to make this program into Position Independent Code. (If you don't know what that is, don't worry about it.) This means that you have to change all the FDB statements, like so: FDB HITHER-*

Assuming that this is a huge table, this would become very difficul to just editing the text in a normal fashion. However, it is not hard to construct a Basic program just for this purpose. Example:

```
1000 OPEN"I",#1,"FILEl/TXT"
1010 OPEN"O",#2,"FILE2/TXT"
1020 LINE INPUT #l,A$
1030 PRINT #2,A$
1040 IF A$<>"TABLE1 EQU *" THEN 1020
1050 LINE INPUT #l,A$
1060 IF INSTR(A$," FDB ")<>0 THEN A$=A$+"-*"
1070 PRINT #2,A$
1080 IF NOT EOF(l) THEN 1050
1090 CLOSE
1100 END
```

THE DCBUG MONITOR

THE USES OF DCBUG

DCBUG is a monitor program provided on disk to assist in debugging and experimenting on the Radio Shack Color Computer. It provides commands for examining and altering memory in hexadecimal, setting breakpoints, converting between hex and decimal, setting and moving blocks of memory, etc.

You do not need to use DCBUG to run an assembly-language program. If you are familiar with assembly language and not very familiar with machine language (hexadecimal instructions), then you will be tempted to ignore DCBUG altogether. Actually, using DCBUG to examine and modify your programs is a good way to become familiar with machine language, and once you are, you will find it an invaluable aid in debugging.

DCBUG is written in position Independent Code so that it may be loaded and run anywhere in memory. It normally loads at $0E00, but may be offset-loaded when debugging a program which is at $0E00. It is a little more than IK long (a little more than $0400 long).

For those of you who are interested, DCBUG has been written so as to be ROM-able (as it is not self-modifying). It is reentrant, as all of its variables are stored on the stack. The only data stored in absolute memory is the breakpoint address and data stored in $00FD through $00FF, which are locations not used by Basic.

RUNNING DCBUG

Type `LOADM"DCBUG"` to load the program off disk. Now type `EXEC` to execute it. DCBUG will display a prompt to the screen, at which point you may type in any of the commands.

Suppose you have a program already loaded at $0E00 which is something less than $0500 bytes long. Type `LOADM "DCBUG",&H0500` and this will load DCBUG $500 bytes past where it would otherwise load, and thus beyond the end of the other program. Now type `EXEC` as usual to run DCBUG.

Many of the commands require one or more parameters. Do not backspace if you make a mistake: just hit `BREAK` and start again. DCBUG will display spaces between the parameters and will return to the command prompt if a parameter is entered in the wrong format.

In the commands listed below, parameters are shown as they appear on the screen. When you type them in, do not enter the spaces: you will see them appear as you type the numbers. You must type in all four digits on each hexadecimal number: all numbers are hexadeciumal except in the convert-to-hexadecimal command.

Summary of Commands

Command	Excample	Description
M address	M 2000	Examine or chanbe memory starting at address $2000
G address	G	Move the address in the PC (Program Counter register) See also section on "Returning to Basic"
I range	I 2000 20FF 3F	Insert the byte $3F into all locations $2000 through $20FF
T range	T 2000 20FF 3000	Transfer contents of all locations $2000 through $20FF to locations $3000 through $30FF
J address	J 2000	Jump to subroutine at address $2000. All registers except S & PC loaded from the register list shown by the R command
R	R	Display register list saved on stack.
C reg	C A	Change register All registers in the register list. This puts you into memory examine/change at the point where the registers are saved on the stack.
$ data	$ 12AB	Convert hexadecimal data to decimal
. data	. 1000	Convert decimal data to hexadecimal note: hit `ENTER` after number
U unit	U FE	Set output unit. FE = Printer
*	*	Reset computer

The "M" Command

This is the most commonly used command. With it you can display or print the contents of memory, alter memory, configure I/O ports, modify programs being debugged, figure out how Basic works, etc.

The contents of memory are written to the screen in eight columns of data with a column of addresses down the left side of the screen. Use the arrow keys to move the cursor throughout memory. When you type in a hex number, that number is written into memory. Memory is then read back to verify that there is memory at that location. If a different number is read back, then the display is inverted at that number. (Try writing to location A000 and you'll see what we mean.)

Hit an `ENTER` to leave this mode. To jump to a new address, just type `M` and the address whether or not you were already in a memory display.

SETTING BREAKPOINTS

If an `SWI` instruction is encountered, control is transferred back to DCBUG. The registers are saved on the stack and printed on the screen. If you type `G`, execution of the interrupted program is resumed at the byte following the `SWI`. The `SWI` instructions may be assembled into your program for debugging.

`SWI` instructions may also be inserted into an object program with the **B** command. Type `B` and an address in RAM, and the contents of that byte will be replaced by an `SWI`. The old contents are saved in location $OOFD, and the address is stored in $00FE and $00FF. When an `SWI` is encountered, if it is at that location saved in $00FE then the old contents of that byte is restored and the program counter decremented so that it points to that byte.

Since a breakpoint is removed when it is encountered, you must use the **B** command again if you wish to stop at that point again. However, if you use the **B** command immediately on the very same address, you will encounter the breakpoint again before any instructions are executed. If you enter a new breakpoint before a previous one is encountered, then the old byte is restored. If, however, DCBUG is reentered at its beginning (such as from Basic) any old breakpoint information is lost.

SENDING THE OUTPUT TO ANOTHER UNIT

The output of DCBUG may be sent to the printer or saved on disk. This is done with the **U** command, which allows you to enter an output unit number in hex. For example, type `U` `FE` and all output will be sent to the printer as well as the screen. Type `U` `00` to turn off the printer. If you open an output file to disk using the Basic command `OPEN"O",#1,"name"` then you can enter DCBUG and type `U` `01` to save all output to the disk file. When you get back to Basic type CLOSE to close the file.

RETURNING TO BASIC

When you hit `G`, you will return to the program which called DCBUG. Generally, DCBUG is called from Basic (with the EXEC command) and the **G** command will return you to Basic. If, however, you call DCBUG from your own program by means of an `SWI` or a breakpoint, then `G` will return to that

program. This is the nature of reentrant programs. Use the **R** command to see where you are going to return to. When in doubt, use the `*` command to do a soft reset and return to Basic that way.

FILELIST AND XFER

These two short programs are written in Basic and are included to make it easier to copy files between disk, tape, printer, and screen.

FILELIST

Type `RUN"FILELIST"` and you will be asked what file to list. You may leave off the `/TXT` extension. The file will be listed to the screen.

XFER

Type `RUN"XFER"` and you will be asked whether to transfer from disk or tape, and whether to transfer to disk, tape, or printer. This is useful for getting printed listings of text files, or for copying files between cassette and disk. You may use this program to copy text files written by the SDS80C Rompack-based editor.

NOTES FOR OWNERS OF SDS80C

If you are familiar with the text editor in the SDS80C from The Micro Works, you will note many similarities with this editor. Here is a summary of the differences:

Lines may be longer than 32 characters

If you continue typing while in line insert mode, a black block will signify a continuation line.

The end bracket character"]" has been moved from shift right arrow to down arrow

This makes the up and down arrows purely text keys while in Line Insert or eXchange, and the left and right arrows purely control keys.

The spacebar no longer is a tab key when in line insert

The right arrow is a tab key if you really want one, but should not be needed for assembly language development.

In eXchange mode, the right arrow key moves the cursor right, and shift right arrow is used to make room

This is easier to remember and more symmetrical with the left arrow key.

Help

The Help key has been added.

Insert and Insert Line

The I key will enter Line Insert mode, as well as the L key.

The minus key is not used

Use "0" instead of "-P", and "JC" instead of "-C". Use "B" instead of "-F".

Jump

The "S" command has been added to jump to a label.

Typeamatic Speed

The typamatic (auto repeat) has been slowed down for certain modes.

Wild Card

The wild card character in find and change now displays a"?" instead of a bracket.

Carriage Return

There is no character for matching a carriage return in find and change modes.