YOUR COMMODORE

SPEEDY ASSEMBLER

USER MANUAL

Argus Specialist Publications

(Version c/7/FD+FT)

```
PRG-NAME: TYPEWRITER PRG
FILENAME: CURSOR RT'S                                          13. JULY 1987

--------------------------------------------------------------------------------

PRG ENTRY                          49152-49329 C000-C0B1     BYTES FREE: 71
PRINT ASCII                        49400-49466 C0F8-C13A     BYTES FREE: 34
DELETE+CURSLF.                     49500-49596 C15C-C1BC     BYTES FREE: 4
CURSOR RT'S                        49600-49698 C1C0-C222



SCREENMEM  252    $00FC
SCREENCOL  254    $00FE
KEYTEST    49152  $C000
TEXTFILE   166    $00A6
CURSFLAG   114    $0072
COUNT      112    $0070



.C000 4C 48 C0 A9 A0 85 FC A9 04 85 FD A9 A0 85 FE A9 ;LH@) .% )..|) .π)
.C010 D8 85 FF A9 10 85 A6 A9 27 85 A7 A2 04 A0 00 A9 ;X.╓)..&)'.'". .)
.C020 20 91 A6 C8 D0 FB E6 A7 CA D0 F6 38 A5 A7 E9 04 ; .&HP┼╼'JPX8%'┐.
.C030 85 A7 20 44 E5 A9 0E 20 16 E7 A9 07 85 D3 A9 6E ;.' D╼). .|)..S)/
.C040 A0 C0 20 1E AB 20 C0 C1 20 E4 FF F0 FB C9 85 F0 ; @ .+ @A ─╓┐┼I.┐
.C050 11 C9 14 D0 03 4C 5C C1 C9 9D D0 03 4C 5C C1 4C ;.I.P.L£AI.P.L£AL
.C060 F8 C0 20 C3 C1 20 44 E5 A9 8E 20 16 E7 60 2A 2A ;°@ CA D╼). .|-**
.C070 2A 20 53 50 45 45 44 59 20 54 59 50 45 57 52 49 ;* SPEEDY TYPEWRI
.C080 54 45 52 20 2A 2A 2A 0D 0D C0 C0 C0 C0 C0 C0 C0 ;TER ***..@@@@@@@
.C090 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 ;@@@@@@@@@@@@@@@@
.C0A0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 ;@@@@@@@@@@@@@@@@
.C0B0 C0 00 00 10 CA 20 EA C8 CA 10 F7 4C 78 C0 20 C6 ;@...J ┐HJ.oL°@ F
.C0C0 C0 B0 EB 4C 78 C0 8A BA 8E D6 C9 0A AA BD A9 C9 ;@0┘L°@.:.VI.*=)I
.C0D0 48 BD A8 C9 48 60 20 3C C1 88 C8 B9 3C 03 F0 25 ;H=(IH- <A.H9<.┐%
.C0E0 C9 3B F0 21 C9 20 F0 F2 A2 02 20 3E C1 A5 4E D0 ;I;┐!I ┐╼". >A%NP
.C0F0 BD A5 4D A0 00 91 4B E6 C9 1D D0 06 A0 00 B1 A6 ;=%M ..K╼I.P. .1&
.C100 D0 04 A0 00 91 A6 C9 80 90 03 38 E9 40 C9 40 90 ;P. ..&I...8┐@I@.
.C110 03 38 E9 40 A0 00 91 FC A9 06 91 FE E6 FC D0 02 ;.8┐@ ..% )..π╼;P.
.C120 E6 FD E6 FE D0 02 E6 FF E6 A6 D0 02 E6 A7 A0 00 ;╼|╼πP.╼╓╼&P.╼' .
.C130 A9 A0 91 FC A9 06 91 FE 4C 00 C0 C1 A2 00 A9 00 ;) .% )..πL.@A".).
.C140 95 4B 95 4C A4 67 88 C8 B9 3C 03 F0 49 C9 20 F0 ;.K.L$|.H9<.┐II ┐
.C150 F6 B9 3C 03 C9 30 90 3E C9 47 B0 3A A0 A0 C4 FC ;X9<.I0.>IG0:  D%
.C160 D0 06 A0 04 C4 FD F0 52 85 49 C9 9D D0 14 A0 00 ;P. .D|┐R.II.P. .
.C170 B1 A6 C9 80 90 03 38 E9 40 C9 40 90 09 38 E9 40 ;1&I...8┐@I@..8┐@
.C180 D0 04 A0 00 A9 20 91 FC A5 FC D0 02 C6 FD C6 FC ;P. .) .% %P.F|F%
.C190 A5 FE D0 02 C6 FF C6 FE A5 A6 D0 02 C6 A7 C6 A6 ;%πP.F╓Fπ%&P.F'F&
.C1A0 A9 00 85 72 A9 13 85 70 A5 49 C9 9D F0 0C A0 00 ;)..╼)..┐%II.┐. .
.C1B0 A9 20 91 FC 91 A6 A9 06 91 FE 4C 00 C0 A5 4E E5 ;) .% .&)..πL.@%N─
.C1C0 4C C6 C1 4C D9 C1 78 A9 E6 8D 14 03 A9 C1 8D 15 ;LFALYA°)╼...)A..
.C1D0 03 A9 00 85 72 85 70 58 60 78 A9 31 8D 14 03 A9 ;.)..╼.┐X-°)1...)
.C1E0 EA 8D 15 03 58 60 E6 70 A5 70 C9 14 F0 02 D0 30 ;┘...X-╼┐%┐I.┐.P0
.C1F0 A9 00 85 70 A5 72 D0 0E E6 72 A0 00 A9 A0 91 FC ;)..┐%╼P.─╼ .) .%
.C200 A9 06 91 FE D0 1A C6 72 A0 00 B1 A6 C9 80 90 03 ;)..πP.F╼ .1&I...
.C210 38 E9 40 C9 40 90 03 38 E9 40 91 FC A9 06 91 FE ;8┐@I@..8┐@.% )..π
.C220 4C 31 EA 00 00 00 00 00 00 00 00 00 00 00 00 00 ;L1┘..............
.C220 4C 31 EA A5 4D E5 4F A5 4E E5 50 B0 E9 18 60 A2 ;L1┘%M─O%N─P0┐.─"
.C230 04 20 3E C1 A9 00 95 4D 85 4E B9 3C 03 F0 03 20 ;. >A)..M.N9<.┐.
.C240 37 C1 A5 4B 85 4C 20 74 C2 18 A5 4F 69 08 85 4F ;7A%K.L |B.%O┐..O
.C250 A5 50 69 00 85 50 B0 1A 38 A5 4D E5 4F A5 4E E5 ;%P┐..P0.8%M─O%N─
.C260 50 90 0F C6 4B D0 DF A5 4C 85 4B 20 27 C1 C9 03 ;P..FKP┼%L.K 'AI.
.C270 D0 D0 18 60 A9 3A 20 EA C8 A5 4F A4 50 20 D0 C8 ;PP.─): ┐H%O$P PH
.C280 A9 20 20 EA C8 A0 00 B1 4F 20 D6 C8 A9 20 20 EA ;)  ┐H .1O VH)  ┐
.C290 C8 C8 C0 08 90 F1 A9 3B 20 EA C8 A0 00 B1 4F AA ;HH@..●); ┐H .1O*
.C2A0 BD 39 CA F0 05 C9 20 90 01 2C A9 2E 20 16 E7 C8 ;=9J┐.I ..,). .|H
```

INTRODUCTION

SPEEDY ASSEMBLER is a comprehensive machine code development package, which is suitable for both the beginner in machine coding and the experienced programmer, who wants to write large, complicated machine-code programs.

Firstly, SPEEDY ASSEMBLER caters for people who want to write machine code subroutines, possibly to be run in conjunction with BASIC.

For this purpose the screen editor of the program, which is completely independent from the Commodore operating system includes a large number of toolkit routines, such as auto linenumbering, block-deletion and block-copying of lines, a search facility and several advanced list facilities. Furthermore there is a comprehensive facility for converting numbers from decimal into hex and binary and vice versa and a monitor read/write facility.

All listings can be sent to the printer which can either format them as on screen or make use of the 80-column line, and there is a large contingency of disk commands, that are all much easier to use than from the BASIC operating system.

Secondly SPEEDY ASSEMBLER puts special emphasis on catering for the experienced machine code programmer who wants to design large and complicated multi-file programs.

The program includes two special facilities which should make the life of the hardened machine code programmer much easier:

Firstly, the ‘master symbol table’, a second symbol table which allows a pseudo-linkage between different source files. Unlike the ordinary symbol table, it is not built up automatically, instead the user decides with the help of a simple transfer facility which label he wants to be kept in the master symbol table. Once a label is in the master symbol table the assembler takes it into account, just like any other label. This allows the user to declare the main variables of his program only once and any subsequent source file will be assembled without problems. The master symbol table is also very useful for jump-addresses and subroutine calls.

The second advanced Facility of SPEEDY ASSEMBLER is the filecatalog. This gives the user a list of all the source files he has designed: filename, beginning of object code, end of object code and the amount of free bytes in between files. It constitutes a complete record of the program under design which is constantly updated, telling the user with one glance how much code he can add to a certain file before the adjacent file is in danger of being overwritten. Because of the filecatalog the assembler is even able to give a warning error report if this should happen!

Another aim of SPEEDY ASSEMBLER is to give as much flexiblity in overall memory-management as possible. All beginnings and endings of the source file, the symbol table, the master symbol table and the filecatalog are made visible

to the user and there is a facility which allows him to block-move any of these files to a more suitable position at any time with a minimum of bother. In addition to this, all the above files are constantly checked to see if any of them is in danger of overwriting another file or the program itself.

SPEEDY ASSEMBLER gives 'intelligent' error reports that is, there are 26 error messages which tell the user clearly and concisely what error has arisen. When entering an assembly line, most error reports are given immediately and the line is not entered into the source file. In the direct command mode, when a command has been entered incorrectly, an arrow will point to the incorrect character or parameter.

Finally, all effort has been made to save valuable memory: all op-codes mnemonics are tokenized. The source file is as compressed as possible, and even though SPEEDY ASSEMBLER permits labels of up to 9 characters which are all treated as significant, each entry in the symbol table uses only 4 bytes because instead of entering the full label, the assembler enters the address which points at the location where the label is in the source file.

****

To close this introduction a few words about some of the expressions used in this manual:

Whenever something is written in cornered brackets, then it signifies a Commodore-key which should be pressed, e.g.:

[RET], [F3], [Space], [R/S].

The above examples mean: Press the return-key, the function key 3, the spacebar, the run/stop-key.

All the commands for SPEEDY ASSEMBLER are given in the manual in the following way (e.g.):

X1,(filename)

First, the command character itself. (Host command characters in SPEEDY ASSEMBLER are single characters, not words or abbreviations as in BASIC.) Then the parameters. If one or several parameters are put in round brackets, this is to make it clear, that it is something which has to be entered by the user, as opposed to a fixed parameter, which has to be entered exactly as given (e.g."X1"). In the former example "filename" implies the name of a file which has to be entered.

CONTENTS

1. THE EDITOR AND ITS COMMANDS

2. ASSEMBLER RULES AND CONVENTIONS

3. THE ASSEMBLER AND ITS OPTIONS

4. DISK AND TAPE FACILITIES

5. ADVANCED FACILITIES

APPENDIX

1. THE EDITOR AND ITS COMMANDS

1.1. Introduction

The main component of SPEEDY ASSEMBLER is the screen editor. It forms the roof under which all the other elements of the program come together.

The editor allows you to enter lines in assembly language or a "source file", as it is called. In order to make this job as easy as possible, it offers you a whole bag full of toolkit routines, like automatic line numbering, renumbering of the source file, block deleting of lines, block copying, and more.

When you want to save something onto disk or tape or print something out with the printer - it's all done via the editor.

And finally, if you want to assemble your source file, the assembler too is called from the editor.

In the next chapters we will describe in depth all you need to know, to make full use of all those facilities.

All commands to the editor start with single characters (except for the "NEW", "HWARM" and "DELETE" commands, which have to be entered in full, in order to remind you every time, that their effect can be lethal, when used wrongly).

1.2. How to Load SPEEDY ASSEMBLER

SPEEDY ASSEMBLER comes in two versions: a disk version and a tape version.

To load the disk version of SPEEDY ASSEMBLER, enter:

```
LOAD "SP",8,1
```

The program will then load and run automatically.

To load the tape version of SPEEDY ASSEMBLER, enter:

```
LOAD "",1,1
```

(Please, do not forget ",1,1", since this forces the computer to load the program at the correct location! You can't load SPEEDY ASSEMBLER with shifted RUN/STOP!)

After the program has loaded, enter:

```
SYS 22623
```

When the program has been loaded successfully you will be given, underneath the main heading, a printout of the

parameters of the main files of SPEEDY ASSEMBLER. At the beginning you may find all those terms and parameters rather confusing. Don't worry! All will be explained in due time (chapter 5.7.). If, at the beginning, you want only to write machine code subroutines, you can ignore most of this anyhow. Only later, when you want to go into machine code programming in a big way, will all this become relevant.

1.3. How to Return to BASIC B or Q

If at any time you want to return to the BASIC operating system, simply enter "B [RET]" or "Q [RET]".

To re-enter the assembler, type:

```
SYS 22623
```

1.4. SPEEDY ASSEMBLER and BASIC

You may give direct BASIC commands from inside the editor, by prefixing them with a colon. E.g.:

```
:PRINT PEEK (198): PRINT PEEK (199)
```

But if you want to enter BASIC lines, you have to leave the assembler as described above.

1.5. Making a Backup-Copy of SPEEDY ASSEMBLER

It makes good sense to produce a backup-copy of SPEEDY ASSEMBLER for your own use. This allows you also to customize the program to a certain extent. For example, you may prefer certain colours which you would like to have installed automatically every time you load the program. Or you prefer the files at certain addresses.

Here is a list of the variables which you can poke (either with the help of the monitor write facility or from BASIC) before saving your version of SPEEDY ASSEMBLER:

Border Colour:	22667	$588B
Paper Colour:	22672	$5890
Ink Colour:	22677	$5895
Beg. of Source File:	22613	$5855
Beg. of Symbol Table:	22615	$5857
Beg. of Master Symbol T.:	22617	$5859
Beg. of Filecatalog:	22619	$585B
Ramtop:	22621	$585D
Disk/Tape Device Number:	22986	$59CA

After you have POKEd the above variables there is one final POKE which should always be done before you save the program, because it will give the program a cold start after it has been loaded:

```
POKE 40870,3
```

Now save the program from inside the assembler by entering:

←SPEEDY ASSEMBLER,22613-40870

IMPORTANT NOTICE

This backup facility is meant for your benefit as a genuine purchaser of SPEEDY ASSEMBLER. You are not allowed to make copies of the program for other people!

1.6. The Use of the Function Keys for Entering Lines F1-F8

In order to produce readable and well structured assembly listings and to permit labels which consist wholly or partially of op-codes or assembler mnemonics, the editor adheres to a fairly strict system of formatting lines. To understand this, think of each line being divided into five 'fields':

1) the line number field
2) the label field
3) the op-code field
4) the operand field
5) the remarks field

Later we will deal with each of these fields (except, of course, the line number field) in depth. At the moment we are only concerned with the pure mechanics of entering a line.

In order to make entering lines as convenient as possible, the function keys act as tab- and block-delete keys.

Here is a list of the function keys, and what they do inside the editor:

F1 = Jump Forward (to the right) to the next field
F2 = delete forward up to the next field
F3 = Jump backwards (to the left) to the former field
F4 = delete backwards up to the former field
F5 = delete the whole line forwards
F6 = delete the whole line backwards

The last two function keys give you two of the options for assembling a source file:

F7 = assemble source file without listing (quick assembly)
F8 = assemble source file with listing and object code

(For more details about this, see 3.1. which deals with the assembler options.)

It may take you a while to make full use of the function keys when entering lines. But once you have gotten into the habit you will find that it is just as convenient as if you were able to enter lines in any way as some assemblers allow you to. The main reason why SPEEDY ASSEMBLER is rather strict in this respect is so that you can use labels with practically any contents. This is where those other assemblers have to restrict you!

1.7. How to Enter Assembly Lines

Let's say, you want to enter the following line:

```
10 LABEL LDA #1 ;REMARK
```

Proceed as follows:

```
10 [Fl] LABEL [F1] LDA [SPACE or F1] #1 [F1] ;REMARK [RET]
```

Now you want to change this line into:

```
10 LABEL LDA 50000 ;REMARK
```

To do this, bring the cursor onto the beginning of the line and enter as follows:

```
[F1] [F1] [F1] [F2] [F3] 50000 [RET]
```

Please, note: A remark at the end of a line has to be started with a semicolon. (This is to tell the assembler, that everything after the semicolon can be ignored.)

Because a line on the screen can only take 40 characters, there is not much space for remarks on the right hand side, and if you write over the line, you'll get the immediate error report 17 "LINE TOO LONG" because it would make your source file look just as horribly unstructured as most BASIC listings. Instead you are advised, to put the bulk of your remarks into extra lines, that is: line number, semicolon, remark.

Furthermore, you can make your source files even more structured by inserting lines which consist only of the line number and a semicolon. This is very good programming practice!

1.8. Immediate Error Checking

In order to keep the amount of error reports during assembly to a minimum and thus speed up assembly, the editor gives as many error reports as possible immediately after the line has been entered.

If an error report is given, the automatic line numbering facility will be interrupted and the faulty line will not be entered into the source file.

Once you have corrected the error the line will be entered as usual and auto line numbering will continue.

SPEEDY ASSEMBLER gives throughout what might be called "intelligent error reports". That means, instead of presenting you constantly with the infamous "Syntax Error" report a wide variety of specific error reports is given. In appendix I of this manual you'll find a complete list of all the error reports used by SPEEDY ASSEMBLER plus their meanings.

Furthermore, if you give an incorrect or incomplete direct command to the editor, you will not only be presented with the standard error report "← PARDON?", but there will also be above it an arrow pointing at the incorrect or missing character.

1.9. Auto Line Numbering I(n)

Another valuable tool to speed up the entry of assembly lines, is the automatic line numbering facility of the editor.

It works slightly different from most extended BASIC programs for the Commodore 64:

In order to activate it, enter "I(n)", whereby "n" is the number of steps in which you want your source file to be numbered. (E.g., old number + n = new number.)

Unlike some extended BASIC programs, you will not be given a line number immediately, instead the first automatic line number will appear, after you have entered the first line and from then on constantly.

The advantage with this system is that the auto line numbering facility is switched on, as long as you want it to be switched on. Even if it is temporarily interrupted by a faulty line or because you want to do something completely different inbetween entering lines, like converting a number from hex into decimal, once you have entered a new line and pressed return it will continue as from this line.

To switch auto line numbering off, simply enter "I" without a parameter.

If you want to be informed about the value in "I", enter "V" which gives you a listing of all the important variables of SPEEDY ASSEMBLER. At the bottom of the listing you will find the value in "I".

1.10. Renumbering the Source File R(m)-(n),(x),(y)

If you simply want to renumber the whole source file, starting from line 10 and numbering in steps of 10, enter "R[RET]".

Otherwise enter the following four parameters after "R":

R(start of renumbering)-(end of renumbering),(first new line number),(steps of incrementation)

This seems to be rather a lot of parameters but it is very useful if you have entered a large source file and have memorized, where each routine is. Nevertheless, you need to open up some space for new lines. The above facility allows you to renumber only the part of the source file where you want to insert your lines, and keep the rest as it was and as you remember it.

Please note, line numbers in assembly listings have not the same relevance as in BASIC. As a matter of fact, they are mostly there for your benefit and also to some extent for the benefit of the editor, when it has to insert new lines into the middle of the source file. The assembler ignores the line numbers completely. Therefore it is quite possible to have, under certain circumstances, line numbers twice or in a non-consecutive order. If this situation arises, it should always be remedied by renumbering the source file. Otherwise, if you were to change a line, the incorrect line might be changed because the editor gets confused. But, nevertheless, lethal it is not!

If you omit the third and fourth parameter the block of source file, which you have specified with the first and second parameter will be renumbered starting from line 10 and renumbered in steps of 10. If you omit the fourth parameter, the file will be renumbered in steps of 10.

1.11. Block-Deleting Lines DELETE(m)-(n)

If you want to delete just one line, you can do this in the usual way, by typing the linenumber and [RET].

To delete several lines in one go, enter the following:

DELETE(first line to be deleted)-(last line to be deleted)

As already mentioned, this is one of the few commands of the editor, which has to be entered in full, because with this command, you could delete the whole source file in a jiffy and it will be gone for good. There is no way you can recover it afterwards.

Therefore, use this command thoughtfully, if you do not want to destroy hours of your work with one keystroke!

There is one more safeguard with this command: If the two line numbers are equal or the second one is smaller than the first one, no action will be taken by the assembler and error report 19 (“END SMALLER THAN BEGINNING”) is given.

1.12. Block-Copying of Lines C(m)-(n),(x)

In machine code programming there is a lot of repetition. Very often you will find yourself entering the same lines over and over again with only slight differences. For those occasions you will find the block-copying facility of SPEEDY ASSEMBLER very useful indeed.

To use it, enter the following:

C(start of block)-(end of block to be copied),(destination line)

With this command it does not matter, if the destination line already exsists or if there isn’t enough space between it and the next line. Nothing will be overwritten or deleted. The block of lines will simply be copied wholesale in between the destination line and the next line. This might of course result in line numbers being double or in the wrong order.

To remedy this, simply use the renumbering facility (see 1.9).

1.13. Listing the Source File L(m)-(n)

If you know how to list a BASIC textfile (and we would be very surprised, if you didn’t), you will have no difficulties with the “L”-command in SPEEDY ASSEMBLER.

It works in a similar way as "LIST". There are only two exceptions: If you want the source file to be listed from a specified line till the end, you can omit the "-"-sign. On the other hand, if you want to list only one line, you have to specify this line with two parameters. E.g.:

L250-250

But there is a more convenient way of listing just one line. See the next chapter!

Here is the "L"-command in full:

L(first line)-(last line to be listed)

1.14. lnput-List Z(m)

This allows you to list your source file line by line and make alterations to it straight away. Simply enter:

Z(start of input-list)

The line, which you have chosen, will be listed and the cursor will be positioned at the beginning of the label field. You can now make alterations to the line or list the next line, by pressing [RET].

Another advantage of this kind of listing facility is that it does not stop when you come to the bottom of the screen. In fact, if you use it in combination with the facility described in the next chapter, you are able to scroll the source file backwards and forwards.

Please note, for operational reasons the "Z"-facility turns off the automatic line numbering facility, if it has been turned on before. It also turns the printer off, in case it has been turned on. You can interrupt Z-list at any time by simply using cursor down to leave the line you are on. But, for reasons of convenience, z-list, like auto line numbering, will still be on, which means, you may return to the line you left or any other line on the screen, and, once you press enter, the next line will be listed.

Z-list is turned off, if you use ordinary list or start auto line numbering. As long as it is turned on you will find a "Z" at the bottom line of the variable listing to remind you.

1.15. Scrolling the Source File Into the Screen From the Top [CURSOR UP]

If there is an assembly line in the top column of the screen, you can scroll the beginning of the source file back into the screen by using the cursor up key.

Once the first line of the source file has been reached or if there isn't an assembly line in the top line, the facility is automatically disabled.

1.16. Ordinary Listing Interrupts [Space], [RET], [R/S]

You can interrupt a listing temporarily at any time by pressing the spacebar. It will then stop and resume only, when you press the return-key.

To abandon a listing permanently - either in midstream or after you have interrupted it temporarily - press the run/stop-key.

As in BASIC listings, you can use the CTRL-key to slow your listing down.

But SPEEDY ASSEMBLER offers you even more control over your listings:

1.17. Automatic Listing Interrupts K(n)

By entering "K" and then a number in the range between 1 and 255, you can determine the number of lines you want to be listed, until the listing stops automatically.

Let's say, you enter "K10".

Now you start your listing in the normal way, with the "L" command.

The result will be, that the first 10 lines will be listed and then the listing stops.

Press [RET], and it continues for another 10 lines. Press [RET] again, and the next 10 lines will be displayed, and so on.

The point about this facility, is, that it gives you much more controlled listings, than the hit-and-miss-affair with the spacebar.

To turn the "K"-facility off, simply enter "K" without a parameter.

To inform yourself, what state the "K"-facility is in at any time, list the system variables, by entering "V".

Finally, when "K" has been switched on, the interrupt facilities, described in the last chapter are still working just the same.

1.18. Finding a String F,(search argument)

SPEEDY ASSEMBLER includes a search facility, which allows you to search a source file for a given string.

For example:

```
F,LABEL
```

would search the source file from beginning to end, and, every time it finds a line, containing the string 'LABEL', would list that line.

There are no rules or restrictions for the search argument; it can be of any length, containing numbers, letters, punctuation marks, spaces, etc.

This facility is useful to find out quickly if a certain variable is contained in a source file, how often and where. It is also useful for making a large amount of similar changes without having to list the whole of the source file.

Please note, at times you can get quite a long list of lines which contain the search argument. In order to make this more controllable, the ordinary and automatic listing interrupts (see 1.16 and 1.17.) work for this facility in the same way as for all other listings!

1.19. SPEEDY ASSEMBLER and Your Commodore Printer P

To send output from SPEEDY ASSEMBLER to the printer instead to the screen is simplicity itself. Just enter "P" without a parameter, and the next output will be sent to the printer. Don't forget to have the printer ready and switched on, before you give the next command!

This works for all the facilities of SPEEDY ASSEMBLER, be it the listing of a source file, an assembler listing, the listing of the symbol table, etc.

And, unlike other such facilities in similar programs, only the things you want to be printed, will be printed. That is, no error reports, ready-messages or other unwanted bits of text will be sent to the printer!

Afterwards the printer will be turned off automatically, that is, the output of the next command you give, will again be sent to the screen.

Remember therefore whenever you want something sent to the printer, enter "P" first!

All the listing interrupts (see 1.16 and 1.17) work with the printer in just the same way as on screen and if a listing is interrupted temporarily and then restarted again, the output will still go to the printer. Only when you abandon it permanently will the printer be turned off.

1.20. Printing a Header P+ G

If you enter "P+" instead of simple "P", a header consisting of the program, name the current filename, the date and a separating line will be added to the listing you want to be printed out.

At the beginning the default date is "NO DATE". To enter the present date, enter

G,31. September 1987

for example. The maximum length is 18 characters, and once you have entered a date it will not be changed back to default, even by the "NEW"-command (see 5.2.).

1.21. Sending Text Directly to the Printer P,(text)

The “P”-command allows you also to send text directly to the printer. In other words, you get a sort of immediate screen-dump.

The syntax for this is:

P,(text to be sent to the printer)

This facility is useful, if you want to add some additional remarks or headings to an assembler listing. To format this text, before you send it to the printer, proceed in the following way:

Clear the screen.

Now type in the text, you want to be sent to the printer, preceding every line you want to be printed as one line with “P,”. But, when moving the cursor down, do not press the RETURN key, use the cursor down key or shifted return instead!

If you are satisfied with what you have on the screen (remembering, that things will look different on paper than on screen, because the printer gives you 80 columns, instead of the mere 40 columns on screen!), get the printer ready and home the cursor back to the top left hand corner of the screen.

Now move the cursor down again, pressing [RET] every time.

Finally, “P,” without any text sends a carriage return to the printer, that is, the paper in the printer is advanced by one line, without anything being printed.

Please note: Do not use quotation marks, since any text in quotation marks is ignored by the printer! Use “‘” instead.

1.22. Monitor Read M(m)-(n)

The editor of SPEEDY ASSEMBLER includes a monitor which allows you to tabulate memory in hex and ASCII and to write directly into memory.

M(start of memory)-(end of memory)

will tabulate a block of memory in lines of 8 bytes in hex and after a semicolon, the same 8 memory-bytes in ASCII-characters.

You may enter the parameters after "M" in decimal as well as in hex.

If you enter "M" with only one parameter, only one line of memory will be tabulated.

If you enter "M" with one parameter and the "-" sign, memory will be tabulated as from that location onwards.

You can interrupt the listing temporarily and permanently in the same way as you would interrupt the listing of a source file.

If you turn the printer on, before entering the "M" command, the listing will go to the printer instead of the screen. The printer tabulates 16 bytes of memory in one line, instead of the 8 bytes on screen.

1.23. Monitor write .($m)

You can modify memory in either one of three ways:

Firstly, by typing in a line like this;

.C000 01 FF 0A

This would modify the first three bytes of User RAM.

Secondly, by tabulating some memory with monitor read (see above) and then modifying the bytes.

Thirdly, you can enter a string of ASCII, like this:

.C000 ;THIS IS AN EXAMPLE.

To make this work, the space and the semicolon in between the location address and text must not be omitted! (If in doubt, use Fl. This will carry you to the correct position.)

If there are further characters on the line which you don't want to be written into memory, use quotation marks as end marker, like this:

.$C000 ; "

The above example would write four spaces ($20) into the beginning of User RAM.

Lastly, with monitor write the location must always be given in hex!

Please note: Since the tape buffer ($033C and upwards) is used by the monitor as workspace, the first few bytes of

it will not give you correct read or write results. If you want to use these locations nevertheless, use, for example, “POKE 828” or “PRINT PEEK (828)” respectively.

1.24. Converting Numbers # $ %

One of the aims of SPEEDY ASSEMBLER is to give you as much assistance as possible in your task of writing a machine code program. Since you will have to do a lot of converting numbers from decimal into hex or from decimal into binary and so on, you might find the following facility useful:

If you enter, let’s say, the decimal number 50000 prefixed with “#”, you will get the following printout which converts the number into low byte decimal, high byte decimal, hex-value, ASCII-code of low byte and 16 bit binary number:

```
<80 >195 $C350 “P” %11000011 01010000
```

In the same manner you can convert hex and binary (8-bit binary only!) into the different notations. You may also enter an ASCII character, prefixing it, like decimal, with the “#” sign, and it will be converted.

1.25. Arithmetic with Different Notations

The conversions routine permits you also to add, subtract, multiply and divide numbers in different notations.

Let’s say, you are writing a graphics routine which includes a table of user designed characters starting at User RAM (dec. 49152). Now you want to find the base address of the re-designed ASCII-character “C”. To do this, enter the following line:

```
#C-32*8+$C000
```

This will give you the result:

```
#49432 <24 >193 $C018 %11000001 00011000
```

Please note, that this facility is not meant to be used as a fully-fledged calculator. No arithmetical priorities are adhered to: if a number becomes negative it will still be treated as positive, and in divisions the remainders are ignored.

2. ASSEMBLER RULES AND CONVENTIONS

2.1. Introduction

For the assembler to do its job, certain rules have to be observed. If you already know how to use assembly language, you will have no difficulties using SPEEDY ASSEMBLER.

But assembly language on its own would make a rather poor assembler. In order for the assembler to become a really useful programming tool, there has to be more.

These extras are called "pseudo instructions" or "assembler instructions". "Pseudo" means that they are not genuine assembly instructions which are translated into machine code for the 6510 microprocessor. Instead they tell the assembler to do certain things, like writing some data into memory at a given location or formatting the assembler listing in a certain way.

In the following chapters we will deal mainly with this kind of instruction.

But first we will look at the use of labels and related subjects.

2.2 LABELS

SPEEDY ASSEMBLER permits labels of 9 characters in length, making "meaningful" labels possible.

Unlike some other assemblers, all 9 characters are significant, that is, are taken into account when comparing one label with another one, each label uses only 4 bytes of space in the symbol table. This is made possible, because SPEEDY ASSEMBLER does not store the label itself in the symbol table, but the address, which points at the position of the label in the source file.

To declare a label used like a variable, use the assembler instruction "EQU", like this:

```
LABEL     EQU 50000
```

A label, which is being used as a Jump or branch destination, has to be declared at one point of the source file on the left hand side.

Please note, all labels on the left hand side have to be "unique", that is, they have to differ from each other by at least one character. Otherwise the assembler will give the error report 9 "LEFT HAND LABEL ALREADY USED".

Like BASIC variables, the first character of a label has to be a letter. Apart from numbers and letters the following characters are permitted inside labels: "!&'?.:@[]#=".

You may use labels, which consist wholly or partially of op-code mnemonics or pseudo instructions. This is possible because of the strict way in which an assembly line for SPEEDY ASSEMBLER has to be formatted. (see also 1.6.)

2.3. Listing the Symbol Table S

You may list the symbol table after a source file has been assembled by entering

S[RET].

With each label you will be given the label value in decimal and in hex.

All the listing facilities of SPEEDY ASSEMBLER apply, including sending the output to the printer.

2.4. The Master Symbol Table S1, T, T1

This is one of the advanced facilities of SPEEDY ASSEMBLER which assists you in writing large and complicated machine code routines.

The master symbol table allows labels to be used as pseudo links between source files.

Unlike the ordinary symbol table, the master symbol table is not built up automatically. Instead, you decide, which labels you want in the master symbol table.

Here are the mechanics of it:

First, assemble your source file in the normal way.

Then enter "T". Now you will be prompted with a listing of the ordinary symbol table. With each label you will be asked, if you would like to transfer this particular label into the master symbol table.

If yes, press "Y", if not, press "N". ([R/S] terminates the process.)

After you have gone like this through the symbol table, assemble the source file once more and list the ordinary symbol table. As you will see, all the labels you have selected for the master symbol table will not have been re-entered into the ordinary symbol table.

They are of course listed in the master symbol table, as you can very quickly find out, by pressing "S1" which lists the master symbol table.

In fact, the assembler treats the master symbol table in the same way as the ordinary symbol table: If it can't find a label in the ordinary symbol table, it searches the master symbol table.

If you now start a new source file or load an old one from disk or tape, then the master symbol table will still be in situ. In fact, at the end of a work session, you should save the master symbol table onto disk or tape, like a source file or an object code file.

All this becomes useful when you write a large machine code program with several source files. Then you need to declare the main variables of the program only in one source file, and, via the master symbol table, they will be found by the assembler every time.

This system becomes even more useful, if you want to jump from a routine in one source file to a routine in another source file which at present is not in the assembler.

With other assemblers, the only way you can do this, is by making a note of the address where you want to jump to. With the master symbol table of SPEEDY ASSEMBLER you can instead simply transfer the label, which points at the routine, from the ordinary symbol table into the master symbol table. Then you can forget all about it. The rest will be done by the assembler.

The result: No long lists of jump-addresses which constantly have to be updated!

To close, one more mechanical point: The "T" command allows you also to erase unwanted labels from the master symbol table. This is a matter of memory-economics, because the master symbol table takes up much more space than the ordinary symbol table, and with a 64k machine you want to be as careful with the available memory, as you can.

It also becomes necessary to erase, at least temporarily, a label from the master symbol table, if you want to change the value of that label. SPEEDY ASSEMBLER does not permit label values in the master symbol table to be altered directly! If you want to change a variable, you have to erase the label from the master symbol table, return to the original source file, where the label has been declared, change the particular line, re-assemble the source file and transfer the label again into the master symbol table.

You might find all this a bit awkward. But it is the best way to prevent a mix-up of variables, which is one of the major trouble sources in writing large multi-file programs!

To erase one or several labels from the master symbol table, enter "T1", and you will be presented with similar prompts as in the transfer a label routine. Only, this time you decide if you want a label to be erased from the master symbol table or not.

All labels, which you have erased from the master symbol table, will of course reappear in the ordinary symbol table once you have re-assembled the source file.

2.5. Op-Codes

SPEEDY ASSEMBLER adheres completely to the conventions of 6510 assembly language. All op-code mnemonics of 6510 assembly will therefore be recognized by the assembler and properly translated into machine code.

2.6. Assembler Instructions

All assemblers worth their salt use pseudo instructions to give added facilities.

SPEEDY ASSEMBLER uses pseudo instructions throughout which fit into the same space as the op-code mnemonics. That is, all pseudo instructions are three letters in length.

There are mainly two categories of assembler instructions used by SPEEDY ASSEMBLER:

Firstly those, which have a direct influence upon the object code output of the assembler.

Secondly, there is a less important category of assembler instructions, which give additional formatting to the assembler listing produced by the assembler.

In the following paragraphs we will deal with each assembler instruction in depth.

2.7. “ORG”

This stands short for “origin” and is followed by a location address in decimal or hex in the range between 0 and 65535.

“ORG” tells the assembler, the start-location of the object code.

The syntax for “ORG” is as follows (e.g.):

```
10              ORG 49152
```

In the above example, the object code will be written to the beginning of User RAM.

You can have several origins in one source file, which means, that the location of the object code will be switched with every “ORG” to the address given after the instruction.

(Please note: When using several ORGs in one source file the filecatalog will give the first ORG as the beginning of the object code and the last location of the source file as the end, even if you use a second or third ORG which is smaller than the first. For this reason it is good practice to use only one ORG in one source file when writing large multi-file programs. If, on the other hand, you only want to write a one file machine code routine, you can ignore the above advice, because in this case the information in the filecatalog will be of little interest to you. - For more about the filecatalog see 5.6.)

2.8. "DIS"

This stands for "displacement" and allows you to assemble object code to one address - the one given after "DIS" - while it is designed to be run later at the address given by "ORG".

The use of "DIS" is best explained with an example:

Say, you want to assemble a source file to the location $6000 (dec. 24576). This is of course well within the space occupied by SPEEDY ASSEMBLER and would therefore result in the error report 21: "'SPEEDY ASSEMBLER' IN DANGER!"

The only way to solve this problem is by getting the assembler to write the object code to a safe place, let's say, $C000, and then saving the object code in such a way, so that it will be loaded to the start address $6000.

To do this, start your source file like this:

```
10          ORG $6000     ;correct ORG
20          DIS $C000     ;temporary ORG
```

If you assemble this source file, you'll see, that the locations shown by the assembler listing, start with $6000. But the object code will have been written to the start of user ram. (If you choose an assembler listing, displacement is signified with a "<" in front of the location.)

Finally, save the object code with the special disk and tape command which is described in chapter 4.6. In our example:

```
↑SOURCE FILE,$C000-$C050,$6000
```

When you re-load the object code it will be loaded to the correct execution location $6000, that is, you can execute the routine by entering "SYS 24576".

2.5. "ENT" E

This means "entry", and is part of a facility, which allows you to execute the machine code routine, which you have written from within the editor, with the simple "E" command.

"ENT" has no parameters after it.

When the assembler encounters "ENT" in the source file, it will save the following location address as the start address of the machine code routine. (In BASIC you would enter that address, together with the "SYS" command.)

Later, when you use the "E" command, the above address will be recovered by the editor, and your machine code routine will be executed from that address.

(Incidentally, you can only use the "E" command, if you have beforehand assembled a routine which did contain the "ENT" instruction.)

In theory, you may give as many "ENT" instructions as you like in one source file. But in practice this is not a good idea, because one "ENT" instruction will overwrite the former one and before you know it, the routine will be run from a different point than it is supposed to. This can be lethal!

2.10. “EQU”

This stands for “equals” and is used to declare a label by assigning a value to it.

We have dealt with the syntax of this instruction already in the chapter about labels (see 2.2.).

2.11. “BYT”

This assembler instruction allows you to give a number of data-bytes, which the assembler will write into memory at the current location address.

As the mnemonic suggests, you may only give “bytes”, that is, numbers in the range between 0 and 255. But you may give several of them in one line, as long as you separate each one of them with a comma.

You may give each byte in decimal, hex or ASCII, like this:

```
100           BYT 13,$0D,”EXAMPLE”,0
```

You may also give a byte in the form of an 8-bit binary number, like this:

```
110           BYT %10001011
```

But the rule here is, that you may give only one binary number in one line and no other numbers or ASCII characters!

2.12. “COM”

This does the same as “BYT”, but any ASCII-codes will be translated by the assembler into the screen values of the Commodore 64. (See page 132 of your C64 manual.)

2.13. “WOR”

This assembler instruction is short for “word”, meaning a 16-bit address.

Why the extra instruction?

“BYT” is used to give simple data-bytes, while “WOR” is used to define a 16-bit location address, which may form part of a jump table. This is why you can also use labels with “WOR”, something, you can’t do with “BYT”.

2.14. “RES”

This assembler instruction stands for “reserve”, and gets the assembler to reserve an area of memory, starting with the current location address. The length is determined by the

number you enter after the instruction. E.g.

RES 100

would reserve a space of 100 bytes. Let's say, the current location address is 50000. This means, that the next instruction will be assembled to the location 50100.

2.15. "END"

This instruction turns assembly on or off. That is, if the assembler meets up with "END" it will ignore the following source file completely until it finds another "END" instruction, in which case it will commence assembly.

"END" is therefore useful if you want part of a source file not to be assembled, but also haven't made up your mind yet, whether to delete that part or not.

2.16. "LIS"

The following assembler instructions are all formatting instructions, that is, they have no influence on the object code produced.

The first one is short for "list", and turns the assembler listing on or off, from inside the assembler.

Let's say you have chosen the assembler command "A(no parameter)" (see 3.1.). This will mean that there will be no assembler listing. Except, if the assembler encounters the "LIS" instruction. In this case it will start producing an assembler listing nevertheless. If there should be another "LIS" instruction in the source file then it will turn the assembler listing off again.

The advantage of all this is, that you can have a selective assembler listing of only part of the source file.

2.17. "PRI"

This assembler instruction will send a carriage return (empty line) to the printer, but only if the printer has been turned on beforehand with the "P" command. The instruction has no effect upon the screen output.

"PRI" without a parameter will send one carriage return to the printer. But you can have several, by entering a number after the instruction. This will then, of course, be the number of carriage returns you will get.

2.18. "NTA"

This is short for "notation" and changes the notation, in which the assembler listing is put out. That is if the location addresses and the object code have before been expressed in hex, they will after "NTA", be expressed in decimal, and vice versa.

2.19. The Operand

SPEEDY ASSEMBLER adheres to all usual conventions of expressing the operand. There are only minor differences, again for the convenience of the user.

In general, you can express an operand in either of six ways:

1) in the form of a decimal number
2) in the form of a hex number (prefixed with "$")
3) in the form of an 8-bit binary number (prefixed with "%")
4) in the form of a character to be converted into the equivalent ASCII-code (prefixed with "')
5) in the form of a character to be converted into a C64-screen code (prefixed with a quotation mark)
6) in the form of a label

If the operand is given in form of a character or a binary number, the "#" sign, which normally has to be used for immediate addressing, can and should be omitted. This is because these notations are used for immediate addressing only.

Binary numbers may only be given in the form of 8-bit binary. Everything else wouldn't make much sense, because who wants to express an address in 16-bit binary? This was only done in ancient days as computers were invented and then it was the only way to enter anything into the computer... On the other hand, 8-bit binary numbers can be extremely useful in graphic routines, where you have to do a lot of bit-operations.

You may also give the operand in form of an arithmetical expression. The following examples are all permitted and will be calculated by the assembler:

```
LABEL+1-2
$FF/8*$02+1
```

The only thing you have to be aware of is, that this kind of operand arithmetic is done strictly from left to right. That is, no arithmetical priorities are adhered to, as in BASIC, where, for example, multiplication is always done before addition or subtraction.

Furthermore, if a number should become negative, it will still be treated as positive, and remainders of division will just be ignored.

2.20. The Use of "<" and ">"

In assembly language the "<" and ">" signs are frequently used in order to denounce the low byte and the high byte of a location address.

SPEEDY ASSEMBLER uses this for two occasions:

Firstly, in absolute addressing, in order to make it visible, that the data has to be loaded or retrieved either from the low byte or from the high byte of a given memory location.

Secondly, in immediate addressing, in order to make it clear, that a register or a memory location has to be loaded with either the low byte or the high byte of a given address itself .

The first occasion is by many assemblers expressed like this:

```
10              LDA SOURCE
20              STA DESTIN
30              LDA SOURCE+1
40              STA DESTIN+1
```

You may use the above example with SPEEDY ASSEMBLER too, but SPEEDY ASSEMBLER also offers you an alternative, which you may find makes the state of affair more visible:

```
10              LDA <SOURCE
20              STA <DESTIN
30              LDA >SOURCE
40              STA >DESTIN
```

In lines 10 and 20 of the above example, you may, of course, quite happily omit the "<" signs, because they are only there for visual clarification. But in lines 30 and 40, you have to use either ">" or "+1"!

In immediate addressing the "<" and ">" sign are even more useful. Look at the following example:

```
10              LDA #<SOURCE
20              LDY #>SOURCE
```

In line 10 the accumulator is loaded with the low byte of the address "SOURCE".

But let's be clear about this: The accumulator is loaded, not with the data contained in "SOURCE", but with the low byte of the address itself, and that means:

```
SOURCE-((SOURCE/256)*256)
```

In line 20 the Y-register is loaded with the high byte of "SOURCE". That means:

```
SOURCE/256
```

Another useful way of expressing things, you might agree, but under no circumstances must you forget the "#"-sign in front of the expression! Otherwise it will be treated by the assembler like the two former examples, and that would give of course a completely different result!

3. THE ASSEMBLER AND ITS OPTIONS

3.1. How to Call the assembler A(n)

To call the assembler, simply enter "A(n)", where n is a number between 1 and 4, and represents the various assembler options.

Here is a full list of the assembler options:

A = Assemble source file, produce assembler listing in hex, and write the object code into memory
A1 = Assemble source file, produce an assembler listing in hex, but do not write the object code into memory
A2 = Assemble source file, write object code into memory, but produce no listing (quick assembly)
A3 = Like A, but produce the assembler listing in decimal
A4 = Like A1, but produce the assembler listing in decimal
F7 = Alternative to entering A2 (quick assembly)
F8 = Alternative to entering A

As you can see the last two options are merely duplications, which allow you to use the convenient function keys.

3.2. How the Assembler works

SPEEDY ASSEMBLER is a two-pass assembler, that means, it assembles a source file into machine code in two goes.

In the first pass it looks through the listing and makes sure, that everything can be assembled properly into machine code.

Since most errors in the first pass, are not lethal, the assembler does not stop the moment it has found an error. Instead it gives the error report plus a reprint of the faulty line and continues until the first pass is completed. Then it tells you the number of errors made and abandons the assembly.

But error checking isn't the only reason for the first pass. There is a much more important one:

In the first pass the assembler builds up the symbol

table (see 2.2.) and thus prepares the calculation of the relative jumps in the second pass. If a relative jump is forward, it would be very difficult to do this in one pass.

If you get to the end of the first pass the caption in reverse:

* FIRST PASS FINISHED: 0 ERRORS *

then you will know that the first pass has been successfully concluded, and that means, that the assembler starts straight away with the second pass.

The second pass will be completed, when you get the reverse caption:

* SECOND PASS FINISHED - OK *

Finally, if you have given anywhere in the textfile the "ENT"-instruction, then the assembler will finish with a caption like this:

CODE RUNS AT 49152

3.3. The Assembler Listing

If you choose the relevant options, the assembler will give you an assembler listing which consists of a listing of the source file, as you have created it, the location addresses, where the object code has been written to, and the object code itself.

Because the screen output is limited to 40 characters per line, it looks very cramped to squeeze all this information onto one line. Therefore, SPEEDY ASSEMBLER formats the assembler listing into two lines for each line of source file:

The first line, which is printed normal, gives you the line number, the location address and then the object code itself. According to your choice, location address and object code will be given either in hex or in decimal.

The second line, which is printed in reverse, gives you a reprint of your assembly line, including remarks and all.

Please note that if you have given a full line of bytes or a full line of ASCII-text, a few bytes of op-code will be omitted from the listing. This is again because of shortage of space. If you compare the location addresses or even check up on this with the monitor, you will find that everything has been assembled correctly!

If you want to study an assembler listing thoroughly, you are advised to have it sent to the printer, because this way it is formatted much more clearly.

For a start, on paper the information, which on the screen is given in two lines, will be given in one line. Furthermore, if there are two succeeding lines of remark, they will be combined into one line with one line number being omitted.

You might even want to format some assembler listings even more to your liking by using the "PRI"-instruction, which gives you empty lines on the printer (see 2.17.) or send some headers or footers directly to the printer,

by using the "P+"-command (see 1.20.) or the "P,"-command (see 1.21.).

3.4. The Production of Object Code

SPEEDY ASSEMBLER gives you the option, where the object code is not written into memory.

This is useful if you want to see how a particular source file assembles but don't want the object code to overwrite other code. Because of this you'll always get an assembler listing with this option.

3.5. The Error Report 26: "OBJECT CODE MAYBE OVERWRITTEN"

This error report is special in that it reports an error condition which is of no importance to the assembler. It is only given for your benefit, if the adjacent object code, as noted down in the filecatalog, is in danger of being overwritten by the present routine.

Since this error is not lethal, you will be presented with three options:

1) continue assembly as usual
2) continue assembly, but no object code
3) abandon assembly

Choose the option you require by simply entering the relevant number.

4. DISK AND TAPE FACILITIES

4.1. Introduction

In order to make the use of the disk drive and the cassette recorder as easy and convenient as possible, SPEEDY ASSEMBLER includes a large number of disk and tape facilities which are all much easier to use than from the BASIC operating system.

SPEEDY ASSEMBLER comes in two versions: a disk version and a tape version.

You can't use the disk version for tape operations by changing the device number. This is not possible for operational reasons. But you can use the tape version for disk operations by changing the device number into, let's say, 8.

So if you upgrade from tape to disk you will have no problems upgrading SPEEDY ASSEMBLER too. There is only one limitation: When loading a file from disk the memory is constantly checked and the loading is aborted if the file is in danger of overwriting another file. This is not done with the tape version, nor will it be done if you upgrade the tape version to disk!

There are basically four main categories of disk commands and three main categories of tape commands, used by SPEEDY ASSEMBLER, each with its uniform command character or prefix:

1) "@" = Disk-command (in BASIC: OPEN 15,8,15)
2) "←" = Save Ram onto disk or tape
3) "↑" = Load Ram from disk or tape
4) "/" = Merge Source file from disk or tape

The first category does of course not work with tape!

All filenames can and should be given without quotation marks, and there is no need to state a device number or a logical file number. (In fact, if not changed, the disk version of SPEEDY ASSEMBLER uses "8" as device-number and the tape version uses "1". In the next chapter you will find out, how you can change the device number if required.)

As already said the disk version (but not the tape version!) of SPEEDY ASSEMBLER uses a special routine for loading files. This has the advantage that, if a file to be loaded is in danger of overwriting an adjacent file, you'll get an error-report 21-26 and loading is aborted. To remedy this move the beginning of the file to a different position where there is enough space for the whole file to be loaded (see 5.8.).

Please note all files are loaded to the start address given in the system variables as listed with the "V"-facility (see 5.7.). This means that you can load any file to a specific address of your choice by changing the start address beforehand (see 5.8.).

4.2. Changing the Device Number &d

If you want to change the device number temporarily, enter:

&d[RET]

where "d" is the device number you require. If you list the system variables with "V", the new device number will be declared in the bottom line.

If you want to upgrade permanently from tape to disk or are using always a disk drive with a different device number than 8 you can change the default device number of SPEEDY ASSEMBLER permanently by poking 22986 with the device number you require and then making a backup of this version of SPEEDY ASSEMBLER (see also 1.5.). (In order for the new device number to come into effect, you have to "NEW" the assembler (see 5.2.), after you have poked the number in!)

IMPORTANT NOTICE: If your disk drive uses a different device number than 8, you will not be able to use any of the disk facilities of SPEEDY ASSEMBLER before you have changed the device number in the above manner! And if you already had trouble using any of the disk facilities, check up on this. It might very well be the source of the trouble you are having!

4.3. Disk-Commands @

The following commands do not work with the tape version of SPEEDY ASSEMBLER (device number 1), but they will work if you upgrade the tape version to disk by changing the device number.

By using "@" you can give any of the commands to the disk drive, which you would give in BASIC by opening channel 15,8,15, and list the directory of a disk.

Here is a list of the disk-commands you can give with "@" (for more explanations, see your disk drive manual!):

```
@    = read disk drive error channel
@$   = print disk directory
@S:  = scratch file
@R:  = rename file
@N:  = format diskette
@I   = initialize drive
@V   = validate diskette
```

When using "@$" to list the disk directory, you can interrupt the listing temporarily or permanently in the same way as with all listings (see 1.16.). The K-facility works too (see 1.17)!

4.4. Saving a Source File ←(filename)

To save the source file you have created onto disk or tape, enter:

←(FILENAME)

This command not only saves the source file, but also the parameters of the file, so that the file will be automatically initialized when reloaded from disk.

4.5. Saving RAM ←(FILENAME),(m)-(n)

This command allows you to save a block of Ram onto disk or tape. The full command is:

←(FILENAME),(beginning of block)-(end of block)

You may give the parameters in hex as well as in decimal, and the end address is inclusive - that is, the end address you give will be the last byte saved. (Similar commands in many other programs are exclusive!)

4.6. Saving and Relocating RAM ←(F1LENAME),(m)-(n),(r)

This works in the same way as the above command, except that after the RAM has been saved, the start address will be changed into the address given at the end of the command. That is, when reloading this file, it will be loaded to a different address than the one it was saved from.

Here is the command in full:

←(FILENAME),(beginning of RAM or saving address)-(end of RAM),(new beginning or loading address)

You may want to use this command after you have used the "DIS"-assembler instruction (see 2.8.). In this case the command would look like this:

←(FILENAME),(address after DIS)-(address after DIS + length of object code file),(address after ORG)

4.7. Loading a Source File ↑(FILENAME)

This is the opposite to the save a source file command as described in 4.4., that is, a formerly saved source file will be re-loaded and re-initialized so that it can be listed, changed and assembled as usual.

4.8. Loading RAM ↑(FILENAME),

This is the opposite to the command described in 4.5.

The important part of this command is the comma after the filename. Do not Forget this comma or the File will be treated as a source file, which could mean that you loose the source file which you have in memory at the moment!

4.9. Merging a Source file /(FILENAME),(n)

This allows you to merge a source file from disk or tape into the existing source file, after the line given at the end of the command.

This is how this command is executed:

Firstly, the source file on disk or tape is appended to the end of the source file in memory.

Secondly, space for merging the block into the source file is made, by moving the top of the source file and the appended source file up.

Finally the block is copied into the space created, in the same way as when you use the block-copy of lines command as describe in 1.12.

4.10. Specialized Disk and Tape Facilities

Together with its advanced facilities to create large and complicated machine code programs, SPEEDY ASSEMBLER offers a number of specialized disk and tape commands which make saving and loading files even more easier and convenient.

In order to make this possible, the program uses and recognizes a number of prefixes which are put in front of the filename and make it clear to the system if a file is a source file, an object code file, a master symbol table file or a filecatalog.

Here are the prefixes used by SPEEDY ASSEMBLER:

S) = source file
C) = object code file
M) = master symbol table
G) = filecatalog

When saving any of the above files - in the abbreviated version, as will be described in the next chapter - the correct prefix is put automatically in front of the filename. But please note, that this means that the maximum length of a filename can only be 14 characters, instead of the usual 16 characters! (See also: “Advanced Facilities”, 5.4. and 5.5.)

4.11. Abbreviated Savings Commands

In the above context the following abbreviated disk and tape commands maybe used:

←,S = save current source file under filename given by "Current File:"
←,C = save object code from first address given by filecatlog to last address given by filecatlog, under filename given by "Programname:"
←,M = save master symbol table under filename given by "Programname:"
←,G = save filecatalog under filename given by "Programname:"

Finally, there are two more abbreviated commands:

←,A = save everything.
←,B = save everything, except the object code file.

The above two commands execute all the above commands in one sequence.

4.12. Loading Prefixed Files

All files which have been saved in the above manner can be loaded by using the "↑"-command. (If you are a tape user, see also 4.13.!)

At the beginning of a work session, first load the filecatalog by entering:

↑G)Programname

This loads the filecatalog, initializes the program name (see 5.1.) and sets the system up for a new work session on a particular program.

Next load the master symbol table by entering:

↑M)*

If you have enough space to have the object code resident in memory, load it now by entering:

↑C)*

Finally, list the filecatalog ("X[RET]") and place the cursor over the source file which you want to be loaded first. Then press [RET].

The source file you have chosen will be loaded and initialized as the current file.

Disk users note: You can use the "↑C)*" and "↑M)*" commands successfully only if you have saved the files for one program onto one disk only! If you have several different programs on one disk, it will result with the first file with that prefix being loaded (see your disk drive manual on "wild card loading"). Therefore, our advice is: Use one disk for the files of one program!

4.13. A Special Note for Tape Users

The whole concept of SPEEDY ASSEMBLER has been designed around the disk drive. While it is not our job to push hardware, we like nevertheless to advise anybody who wants to go into machine code programming in a big way, to consider seriously upgrading to disk.

But if you are a tape user and want to try your hand at writing a large multi-file machine code program, consider the following advice:

Firstly, save one source file onto one cassette. This might be a lot of C10 or C15 cassettes for a lot of source files, but it saves quite a bit of work in terms of having to note down digits and re-saving source files if one might overwrite another one.

Secondly, use one cassette to save the filecatalog, the master symbol table, the object code, and the first source file by using the "←,a"-command. If there are any changes in any of those files you can then simply re-save the whole lot with the same command!

Thirdly, be aware that when you are loading any file which has been saved with the abbreviated saving command, you have to type in the full filename, including the prefix! This is necessary because SPEEDY ASSEMBLER needs to know what type of file to initialize. E.g.:

G)DEMONSTRATION or M)DEMONSTRATION

But if you want to load a file without a prefix you may simply enter "↑" or "↑,", and the next file on tape will be loaded in

5. ADVANCED FACILITIES

5.1. Introduction

In order to assist the programmer who wants to create a large machine code program and to be as flexible as possible, SPEEDY ASSEMBLER has a number of “advanced” facilities, which will be described in the following chapters in detail.

5.2. NEWing the Program NEW

When you enter:

NEW [RET]

the whole program is given a “cold start”, that is, all the variables are reset to default (except the date) and all files are cleared. In short, the program is restarted as if it has just been loaded from disk (see 1.2. and following).

To use this command safely, make sure that you have saved everything you want to use again!

5.3. Half-NEWing the Program HWARM

This command should be used every time you want to start a new source file in a chain of source files and after you have saved the old source file.

It results in the old source file being newed and the source file variables being set to default. Furthermore, a new line is added to the filecatalog under the default file name “No name”. this becomes then the “current file” and you can give it the filename you want, by using the “X2” command (see 5.5.).

The master symbol table and the rest of the filecatalog are obviously left unchanged by this start.

5.4. Changing the_Programname X1,(programname)

The “programname” is the overall name you would give to a large machine code program, like “Supergame”, “Basic Utility”, or “SPEEDY ASSEMBLER”.

After having loaded SPEEDY ASSEMBLER the default program

name is “No Name”, as you can see from the list of the main variables which is printed out every time the program is loaded afresh.

You can change this name at any given time by entering:

X1,(programname)

Apart from the visual impact, the programname is used as the filename for saving the filecatalog, the object code file and the master symbol table. This means, of course, that the appropriate prefix will be put in front of the programname, and because of this the programname may not be longer than 14 characters.

Please note, if you change the programname after you have already saved the filecatalog or the master symbol table or an object code file under the former name, you will have to rename all those files by using the rename disk command.

5.5. Changing the Current Filename X2,(filename)

The current filename is the name of the sourcefile which you are creating at the moment or have created. That is, when using the “←,S” command, the source file will be saved under this filename.

Again, at the beginning, the default filename is “No Name”. To change this name enter:

X2,(filename)

The maximum length is again 14 characters, and you can check up on the filename having been assigned properly by listing the system variables (“V [RET]”) or the filecatalog (“X [RET]”).

5.6. The Filecatalog X X3

This is another important facility if you are creating a large multi-file machine code program.

Each entry of the filecatalog contains the following information:

1) the sourcefile loading prefixes for easy loading of the source file
2) the filename of the respective source file
3) the beginning of the object code as produced by that source file in decimal
4) the end of the object code as produced by that source file in decimal
5) the beginning of the object code in hex
6) the end of the object code in hex
7) the number of bytes free inbetween the end of one object code file and the beginning of the next one

In other words, the filecatalog informs you about all the source files you have written so far and how much space in memory the object code, created by each source file, takes up.

The filecatalog shows you also with one glance how much space there is still inbetween two source files, that is, how

much you can add to a source file until the object code produced will overwrite existing object code. The filecatalog also makes it possible for the assembler to give you the error report 26: "OBJECT CODE MAY BE OVERWRITTEN" (see 3.5.).

To list the filecatalog, enter:

X [RET]

For this, all the usual listing facilities apply, and you can send the listing to the printer too.

The listing of the filecatalog permits you to also load any source file from disk or tape in a very easy manner: Simply place the cursor over the source file you require and press [RET].

From time to time you will want to erase a file from the filecatalog. Do this by entering "X3". You will then be prompted with one file after the other and can decide if you want it to be erased or not.

5.7. Listing the Main Variables of SPEEDY ASSEMBLER V

One of the major aims of SPEEDY ASSEMBLER is maximum flexibility for you, the user.

In order for this to work, firstly, all the main variables of the program are being made visible to you. Secondly, you can change them in an easy manner, whenever you like.

Whenever you load SPEEDY ASSEMBLER anew, you get a listing of all the main variables of the program.

At the beginning you need not concern yourself too much with all of that. SPEEDY ASSEMBLER will work quite happily without you knowing the exact meaning of each variable.

But later on it will become a useful tool, especially if you want to write a large machine code program consisting of several source files. Then you will have to do some judicious memory planning and, as things go, it will not always work out as you planned it at the beginning.

This is where the flexibility of SPEEDY ASSEMBLER comes into its own: Instead of spending hours of peeking and poking and trying to fit things in somehow, you can reorganize things with a minimum of fuss.

Every component of SPEEDY ASSEMBLER, that is, the source file, the symbol table space, the master symbol table space and the filecatalog, can be relocated easily (see the next chapter).

But first, of course, you want to know where everything is at any given time and how much space each item is taking up.

You get this information by typing "V". The listing you will be presented with gives you the following information:

1) the name of the present program (see 5.4.)
2) the name of the present file (see 5.5.)
3) the date (see 1.20.)

4) the beginning of the current source file
5) the end of the current source file
6) the beginning of the ordinary symbol table
7) the end of the ordinary symbol table
8) the beginning of the master symbol table
9) the end of the master symbol table
10) the beginning of the filecatalog
11) the end of the filecatalog
12) ramtop (the ceiling set onto the BASIC system)
13) the present value in "I" (auto line numbering) (see 1.9.)
14) the present value in "K" (auto listing interrupts, see 1.17.)
15) the default device number (see 1.5. and 4.2.)
16) "P" or "P+", if the printer is on (see 1.19. and 1.20.)
17) "Z", if input-list is on (see 1.14.)

5.8. Changing and Block-Moving the Major Components of SPEEDY ASSEMBLER V1

If you want to change, let's say, the position of the source file, enter "V1".

Now you will be prompted with the beginnings of all the files listed by "V". (You don't need to change the endings because they will automatically be adjusted in relation to the beginnings).

If you don't want to change any particular item, simply press [RET]. If you want to exit prematurely, use cursor down and press [RET].

If you want to change the beginning of a file, type in the new address where you want it to go and then press [RET].

Now two things will happen: Firstly, the whole file will be moved to the new address you have given. Secondly, all the variables concerning this file will be re-initiated at the new position.

You can see some of the effects by listing the variables again.

(Incidentally, when starting off, the beginning of the source file is set by default to 3500. This is some 1000 bytes into BASIC-RAM. The reasoning behind this is that it allows you to enter a few BASIC lines if you want to test your machine code routine.)

APPENDIX

I. SPEEDY ASSEMBLER Error Reports

→1 SYNTAX ERROR

Immediate Error-Check: Label or op-code in wrong field

→2 INVALID LABEL

Immediate Error-Check: Label starts with number; label contains reserved character; label longer than 9 characters

→3 INVALID OP-CODE

Immediate Error-Check: Op-code or pseudo instruction unknown

→4 INVALID OPERAND

Immediate Error-Check: Operand not allowed

→5 OPERAND MISSING

Immediate Error-Check: Op-code or pseudo instruction should have an operand

→6 OP-CODE HAS NO OPERAND

Immediate Error-Check: Op-code or pseudo instruction should have no operand

→7 NO IMMEDIATE ADDRESSING

Immediate Error-Check: Op-code does not support immediate addressing

→8 OPERAND TOO BIG

Immediate Error-Check and First Pass: Operand should be of zero-page size

→9 INCORRECT INDIRECT ADDRESSING

Immediate Error-Check: Incorrect construction of (INDIRECT),Y or (INDIRECT,X)

→10 INCORRECT INDEXED ADDRESSING

Immediate Error-Check: Incorrect construction of ABSOLUTE,X or ABSOLUTE,Y

→11 QUOTATION MARK MISSING

Immediate Error-Check: after "BYT" and "COM"; final quotation mark missing at the end of ASCII-string

→12 SEPARATION MARK MISSING

Immediate Error-Check:
1) Rest of line might be remark. but not sure because semicolon missing
2) After "BYT" and "COM": Separating comma missing

→13 LABEL NOT INTRODUCED

Second Pass: Label used which has not been introduced, neither in source file nor master symbol table

→14 LEFT HAND LABEL ALREADY USED

First Pass: Label on left hand side has been re-defined

→15 RE-DEFINED MASTER-LABEL

Second Pass: Label in master Symbol table has been re-used with different value

→16 RELATIVE JUMP TOO BIG

Second Pass: Relative forward or backward branch out of range

→17 LINE TOO LONG

Immediate Error-Check: Line entered longer than 40 characters

→18 NO FILE

Direct Entry: File Empty

→19 END SMALLER THAN BEGINNING

Direct Entry: Renumber, block-copy, block-delete: beginning of block larger than end

→20 INTEGER OUT OF RANGE

Direct Entry: Block-copy and block-delete: line number too big

→21 SPEEDY ASSEMBLER IN DANGER!

File Entry: Program in danger of being overwritten

→22 SOURCE FILE IN DANGER!

File Entry: Source File in danger of being overwritten

→23 SYMBOL TABLE IN DANGER!

File Entry: Symbol Table in danger of being overwritten

→24 MASTER SYMBOL TABLE IN DANGER!

File Entry: Master Symbol Table in danger of being overwritten

→25 FILECATALOG IN DANGER!

File Entry: Filecatalog in danger of being overwritten

→26 OBJECT CODE MAYBE OVERWRITTEN

First Pass: Object code is in danger of overwriting adjacent object code (see 3.5.)

II. SPEEDY ASSEMBLER Commands

A	Assemble in hex with listing	O/K/P/P+	3.1.
A1	Assemble in hex with no obj.-code	O/K/P/P+	3.1.
A2	Quick assembly (no listing)		3.1.
A3	assemble in decimal with listing	O/K/P/P+	3.1.
A4	Assemble in decimal with no obj.-code	O/K/P/P+	3.1.
B	Exit to BASIC		1.3.
C	Block-copy Lines		1.12.
DELETE	Block-delete Lines		1.11.
E	Execute code		2.9.
F	Search for String	O/K	1.18.
G	Enter date		1.20.
HWARM	Half warm start		5.3.
I	Automatic line numbering		1.9.
K	Automatic listing interrupts		1.17.
L	List source file	O/K/P/P+	1.13.
M	Monitor read	O/K/P/P+	1.22.
NEW	Cold start		5.2.
P	Printer on		1.19.
P+	Printer on + header		1.20.
Q	Exit to BASIC		1.3.
R	Renumber source file		1.10.
S	List symbol table	O/K/P/P+	2.3.
S1	List master symbol table	O/K/P/P+	2.4.
T	Transfer label		2.4.
T1	Erase label from master symbol table		2.4.
V	Print main prg-variables		5.7.
V1	Change main prg-variables		5.8.
X	List filecatalog	O/K/P/P+	5.6.
X1	Change program name		5.4.
X2	Change current filename		5.5.
X3	Erase file from catalog		5.6.
Z	Input-List		1.14.
&	Change device number		4.2.
.	Monitor write		1.23.
#	Decimal conversion		1.24.
$	Hex conversion		1.24.
%	Binary conversion		1.24.
@	Execute disk command		4.3.
@$	List disk directory	O/K	4.3.
←	Save file		4.4.
↑	Load file		4.7.
/	Merge file from disk/tape		4.9.
[F1]	Tabulator right to next field		1.6.
[F2]	Delete right to next field		1.6.
[F3]	Tabulator left to former field		1.6.
[F4]	Delete left to former field		1.6.
[F5]	Delete whole line to the right		1.6.
[F6]	Delete whole line to the left		1.6.
[F7]	Quick Assembly (no listing)		3.1.
[F8]	Assemble in hex with listing	O/K/P/P+	3.1.

O = Ordinary listing interrupts work with this command
K = Automatic listing interrupts work with this command
P = Listing can be sent to printer
P+ = Listing can be sent to printer together with header

III. Using the Demonstration Program

Both the disk and the tape version of SPEEDY ASSEMBLER come with a demonstration program, called "TYPEWRITER PRG".

This is a very simple set of routines which allow you to enter some text in 40-column business mode, delete characters, and move the cursor left or right.

It is a very similar routine as currently developed in the "YOUR COMMODORE" series of articles, called "Byting Into the 6510". But the main purpose here is to demonstrate the workings of SPEEDY ASSEMBLER and especially the multi-file capacities of the program.

If you are a disk user load the program in by entering first;

```
↑G)TYPEWRITER PRG
```

(Remember, that you always have to type in the full filename when loading the filecatalog! Otherwise the programname will not be initialized properly.)

Afterwards enter "V" to list the system variables. This shows you that the programname has been initialized and also the name of the first source file (in this case "PRG ENTRY").

Next load the master symbol table, simply like this:

```
↑M)*
```

The object code, that is, the assembled code of the program proper, can be loaded just as easy:

```
↑C)*
```

and, if you like, you can give the program a first run at this stage, by entering ": SYS 49155". (You have to prefix this command with a colon, because you are giving a direct BASIC command from inside the assembler.)

Finally, list the filecatalog, by entering "X", and place the cursor on whichever source file you want to load first. Then press return.

If you've got the tape version of SPEEDY ASSEMBLER, you can load and initialize the files in a similar way, but remember, you have to type all prefixes and filenames out .IN FULL and you have to do it in the exact sequence as listed above, because this is the way in which the files have been saved onto tape!

Therefore, type:

```
↑G)TYPEWRITER PRG
↑M)TYPEWRITER PRG
↑C)TYPEWRlTER PRG
```

each time after one file has been loaded.

The first source file ("PRG ENTRY") you can load equally conveniently, by listing the filecatalog, placing the cursor over the first file, and pressing return.

The source files are recorded in the sequence as they are listed in the filecatalog, that is:

```
↑S)PRG ENTRY
↑S)PRINT ASCII
↑S)DELETE+CURS.LF
↑S)CURSOR RT'S
```

You can run the program by giving the direct BASIC command:

```
:SYS 49155
```

from inside the assembler. Or you can run it by assembling the first source file ("PRG ENTRY") and then simply entering "E"

It is a good idea to have the demonstration program loaded in the computer when studying this manual. This way you can try out a lot of things straight away without having too much to type in.

For example, list the source file of "PRG ENTRY" with the Z-facility (see 1.14.) to see how this works. When some of the lines have been scrolled out of the screen, recover them by using cursor-up (see 1.15.).

All this gives you a practical demonstration of what has been described in this manual.

We also hope that the demonstration program will give you a much clearer idea of the multi-file capabilities of SPEEDY ASSEMBLER.

For example, the label "SCREENMEM", which you can find in the master symbol table (enter "S1") is used in all the files, and yet, it has only been declared once in "PRG ENTRY"!

Maybe there are things you would like to add to the routines. In this case the filecatalog (enter "X") shows you clearly how much more code you can add to each routine until it would be in danger to overwrite the one next to it.

Lastly, you might also like to send some of the lists to the printer to see how they are formatted on paper.

PRG-NAME: TYPEWRITER PRG
FILENAME: PRG ENTRY 13. JULY 1987

```
10              ORG 49152
20    ;
30    SCREENMEM EQU 252
40    SCREENCOL EQU 254
50    TEXTFILE  EQU $A6
60    ;
70    GETIN     EQU $FFE4
80    ;
90    PRINTASCI EQU 49400
100   DELE:CRLF EQU 49500
110   CURSORON  EQU 49600
120   CURSOROFF EQU 49603
130   ;
140   ;
150   ;JUMP-VECTOR TO KEYTEST
160   ;
170   KEYTEST   JMP KEY
180   ;
190   ;
200   ;PROGRAM ENTRY
210   ;
220             ENT
230   ;
240   ;
250   ;   *** PROGRAM INITIATION  ***
260   ;
270   ;INITIATE SCREEN AND COLOUR VARS.
280   ;
290             LDA #<1024+160
300             STA <SCREENMEM
310             LDA #>1024+160
320             STA >SCREENMEM
330   ;
340             LDA #<55296+160
350             STA <SCREENCOL
360             LDA #>55296+160
370             STA >SCREENCOL
380   ;
390   ;INITIATE TEXTFILE
400   ;
410             LDA #<10000
420             STA <TEXTFILE
430             LDA #>10000
440             STA >TEXTFILE
450   ;
460   ;FILL TEXTFILE WITH SPACES
470   ;
480             LDX #4
490             LDY #0
500             LDA #32
510   FILLOOP   STA (TEXTFILE),Y
520             INY
530             BNE FILLOOP
540             INC >TEXTFILE
550             DEX
560             BNE FILLOOP
570   ;
580   ;RESET TO BEGINNING OF TEXTFILE
590   ;
600             SEC
610             LDA >TEXTFILE
620             SBC #4
630             STA >TEXTFILE
640   ;
650   ;CLEAR THE SCREEN.
660   ;
670             JSR $E544
680   ;
690   ;SWITCH COMPUTER INTO
700   ;BUSINESS MODE.
710   ;
720             LDA #14
730             JSR $E716
740   ;
750   ;PRINT HEADER AT TAB 7
```

```
760   ;
770             LDA #7
780             STA $D3 ;CURSOR POSIT.
790   ;
800             LDA #<HEADER
810             LDY #>HEADER
820             JSR $AB1E
830   ;
840   ;TURN INTERRUPT DRIVEN CURSOR ON
850   ;
860             JSR CURSORON
870   ;
880   ;
890   ;
900   ;   ***  KEYTEST ROUTINE  ***
910   ;
920   ;WAIT FOR KEYPRESS AND JUMP TO
930   ;APPROPRIATE SUBROUTINE
940   ;
950   KEY       JSR GETIN
960             BEQ KEY
970   ;
980   ;F1 = EXIT TO BASIC
990   ;
1000            CMP #133
1010            BEQ EXIT
1020  ;
1030  ;20 = DELETION ROUTINE
1040  ;
1050            CMP #20
1060            BNE CONTINUE
1070            JMP DELE:CRLF
1080  ;
1090  ;157 = CURSOR LEFT ROUTINE
1100  ;
1110  CONTINUE  CMP #157
1120            BNE CONTINUE1
1130            JMP DELE:CRLF
1140  ;
1150  ;PRINT ASCII
1160  ;
1170  CONTINUE1 JMP PRINTASCI
1180  ;
1190  ;
1200  ;
1210  ;  ***  MAIN PROGRAM EXIT  ***
1220  ;
1230  ;TURN CURSOR OFF
1240  ;
1250  EXIT      JSR CURSOROFF
1260  ;
1270  ;CLEAR SCREEN
1280  ;
1290            JSR $E544
1300  ;
1310  ;SWITCH BACK TO GRAPHICS MODE
1320  ;
1330            LDA #142
1340            JSR $E716
1350  ;
1360  ;RETURN TO BASIC
1370  ;
1380            RTS
1390  ;
1400  ;
1410  ;
1420  ;  ***  TEXT FOR PRINTING  ***
1430  ;
1440  HEADER    BYT "*** SPEEDY TYPEWR"
1450            BYT "ITER ***",13,13
1460            BYT "-----------------"
1470            BYT "-----------------"
1480            BYT "------",0
```

PRG-NAME: TYPEWRITER PRG
FILENAME: PRINT ASCII 13. JULY 1987

```
10              ORG 49400
20    ;
30    ;*******************************
40    ;
50    ;THIS ROUTINE PRINTS A CHARACTER
60    ;ONTO THE SCREEN AND ENTERS IT
70    ;INTO THE TEXTFILE. IF CRSR-RIGHT
80    ;(ASCII 29) HAS BEEN PRESSED, THE
90    ;THE CHARACTER IN THE TEXTFILE IS
100   ;RECOVERED TO MOVE THE CURSOR.
110   ;
120   ;*******************************
130   ;
140   ;IF CURSOR RIGHT, RESTORE
150   ;CHARACTER FROM TEXTFILE
160   ;
170             CMP #29
180             BNE CONTINUE
190   ;
200             LDY #0
210             LDA (TEXTFILE),Y
220             BNE CURSRIGHT
230   ;
240   ;
250   ;
260   ;ENTER CHARACTER INTO TEXTFILE
270   ;
280   CONTINUE  LDY #0
290             STA (TEXTFILE),Y
300   ;
310   ;CONVERT ASCII INTO SCREEN CODE.
320   ;
330   CURSRIGHT CMP #128
340             BCC SKIP
350             SEC          ;UPPER CASE
360             SBC #64
370   SKIP      CMP #64      ;LOWER CASE
380             BCC PRINT
390             SEC
400             SBC #64
410   ;
420   ;PRINT CHARACTER ONTO SCREEN.
430   ;
440   PRINT     LDY #0
450             STA (SCREENMEM),Y
460             LDA #6
470             STA (SCREENCOL),Y
480   ;
490   ;ADVANCE TO NEXT PRINT POSITION.
500   ;
510             INC <SCREENMEM
520             BNE NOHIGH
530             INC >SCREENMEM
540   ;
550   NOHIGH    INC <SCREENCOL
560             BNE NOHIGH1
570             INC >SCREENCOL
580   ;
590   NOHIGH1   INC <TEXTFILE
600             BNE NOHIGH2
610             INC >TEXTFILE
620   ;
630   ;PRINT CURSOR
640   ;
650   NOHIGH2   LDY #0
660             LDA #160
670             STA (SCREENMEM),Y
680             LDA #6
690             STA (SCREENCOL),Y
700   ;
710   ;
720   ;
730   ;MAIN EXIT: RETURN TO KEYTEST
740   ;
750             JMP KEYTEST
```

PRG-NAME: TYPEWRITER PRG
FILENAME: CURSOR RT'S 13. JULY 1987

```
10              ORG 49600
20    ;
30    ;*********************************
40    ;
50    ;     THIS FILE CONSISTS OF TWO
60    ;             ROUTINES:
70    ;
80    ;RT 1 TURNS THE INTERRUPT, WHICH
90    ;DRIVES THE CURSOR, ON OR OFF.
100   ;
110   ;RT 2 DRIVES THE CURSOR WHENEVER
120   ;THE INTERRUPT IS CALLED.
130   ;
140   ;*********************************
150   ;
160   IRQVECTOR EQU $0314
170   NORMALIRQ EQU $EA31
180   COUNT     EQU $70
190   CURSFLAG  EQU $72
200   ;
210   ;
220             JMP CURSORON
230             JMP CURSOROFF
240   ;
250   ;
260   ;
270   ;     ***  TURN CURSOR ON  ***
280   ;
290   CURSORON  SEI
300             LDA #<FLASHCURS
310             STA <IRQVECTOR
320             LDA #>FLASHCURS
330             STA >IRQVECTOR
340   ;
350             LDA #0
360             STA CURSFLAG
370             STA COUNT
380   ;
390             CLI
400             RTS
410   ;
420   ;
430   ;
440   ;
450   ;     ***  TURN CURSOR OFF  ***
460   ;
470   CURSOROFF SEI
480             LDA #<NORMALIRQ
490             STA <IRQVECTOR
500             LDA #>NORMALIRQ
510             STA >IRQVECTOR
520             CLI
530             RTS
540   ;
550   ;
560   ;
570   ;*********************************
580   ;*********************************
590   ;
600   ; ***  FLASH CURSOR ROUTINE  ***
610   ;
620   ; (THIS ROUTINE IS CALLED EVERY
630   ;       60TH OF A SECOND.)
640   ;
650   ;UPDATE COUNTER. IF 20 HAS BEEN
660   ;REACHED, CHANGE CURSOR. ELSE
670   ;EXIT STRAIGHT AWAY.
680   ;
690   FLASHCURS INC COUNT
700             LDA COUNT
710             CMP #20
720             BEQ CHANGE
730             BNE EXIT
740   ;
750   ;RESET COUNTER AND TEST CURSFLAG.
```

```
760   ;IF CLEAR, PRINT CURSOR. IF SET,
770   ;PRINT LETTER UNDER CURSOR.
780   ;
790   CHANGE    LDA #0
800             STA COUNT
810   ;
820             LDA CURSFLAG
830             BNE PRINTCHAR
840   ;
850             INC CURSFLAG
860             LDY #0
870             LDA #160
880             STA (SCREENMEM),Y
890             LDA #6
900             STA (SCREENCOL),Y
910             BNE EXIT
920   ;
930   PRINTCHAR DEC CURSFLAG
940             LDY #0
950             LDA (TEXTFILE),Y
960   ;
970   ;CONVERT ASCII INTO SCREEN CODE.
980   ;
990             CMP #128
1000            BCC SKIP
1010            SEC          ;UPPER CASE
1020            SBC #64
1030  SKIP      CMP #64      ;LOWER CASE
1040            BCC PRINT
1050            SEC
1060            SBC #64
1070  ;
1080  PRINT     STA (SCREENMEM),Y
1090            LDA #6
1100            STA (SCREENCOL),Y
1110  ;
1120  ;
1130  ;
1140  EXIT      JMP NORMALIRQ


PRG-NAME: TYPEWRITER PRG
FILENAME: DELETE+CURS.LF                          13. JULY 1987

----------------------------------------------------------------

10              ORG 49500
20    ;
30    ;*******************************
40    ;
50    ;THIS ROUTINE BACKSPACES BY ONE.
60    ;IF THE DEL-KEY (ASCII 20) HAS
70    ;BEEN PRESSED, THE CURSOR IS
80    ;MOVED BACK BY ONE AND THE FORMER
90    ;CHARACTER IS DELETED. IF
100   ;CRSR-LEFT (ASCII 157) HAS BEEN
110   ;PRESSED, THE CURSOR IS ONLY
120   ;MOVED ONE POSITION TO THE LEFT.
130   ;
140   ;*******************************
150   ;
160   TEMPSTORE EQU $49
170   ;
180   ;
190   ;
200   ;IF BEG. OF SCREEN, EXIT.
210   ;
220   DELETE    LDY #<1024+160
230             CPY <SCREENMEM
240             BNE DELETE1
250             LDY #>1024+160
260             CPY >SCREENMEM
270             BEQ EXIT
280   ;
290   ;
300   ;IF CURSOR LEFT, RECOVER
310   ;CHARACTER FROM TEXTFILE
320   ;
330   DELETE1   STA TEMPSTORE
```

```
340             CMP #157
350             BNE DELETE2
360   ;
370             LDY #0
380             LDA (TEXTFILE),Y
390   ;
400   ;CONVERT ASCII INTO SCREEN CODE.
410   ;
420             CMP #28
430             BCC SKIP
440             SEC          ;UPPER CASE
450             SBC #64
460   SKIP      CMP #64      ;LOWER CASE
470             BCC PRINT
480             SEC
490             SBC #64
500   ;
510             BNE PRINT
520   ;
530   ;
540   ;
550   ;ERASE CURSOR
560   ;
570   DELETE2   LDY #0
580             LDA #32
590   PRINT     STA (SCREENMEM),Y
600   ;
610   ;BACKSPACE ONE POSITION
620   ;
630   CURSLEFT  LDA <SCREENMEM
640             BNE NOHIGH1
650             DEC >SCREENMEM
660   NOHIGH1   DEC <SCREENMEM
670   ;
680             LDA <SCREENCOL
690             BNE NOHIGH2
700             DEC >SCREENCOL
710   NOHIGH2   DEC <SCREENCOL
720   ;
730             LDA <TEXTFILE
740             BNE NOHIGH3
750             DEC >TEXTFILE
760   NOHIGH3   DEC <TEXTFILE
770   ;
780   ;ENSURE THAT CURSOR IS PRINTED
790   ;
800             LDA #0
810             STA CURSFLAG
820             LDA #19
830             STA COUNT
840   ;
850   ;IF CURSOR LEFT, EXIT
860   ;
870             LDA TEMPSTORE
880             CMP #157
890             BEQ EXIT
900   ;
910   ;IF DELETE, PRINT SPACE OVER
920   ;FORMER CHARACTER
930   ;
940             LDY #0
950             LDA #32          ;SPACE
960             STA (SCREENMEM),Y
970             STA (TEXTFILE),Y
980             LDA #6           ;BLUE
990             STA (SCREENCOL),Y
1000  ;
1010  ;
1020  ;
1030  ;EXIT: RETURN TO KEYTEST
1040  ;
1050  EXIT      JMP KEYTEST
```