'FASSEM'

THE FAST ONE

The Assembler/Disassembler for your CBM 64

Variables and Labels

All variable names with the exception of arrays can now be of any length, but must not contain reserved words (such as TO and FOR) and must not begin with system variable names (such as ST and TI). For example, MESSAGE1$ is now recognised as being different from MESSAGE2$. System variables are still recognised by their first two letters only, to maintain compatability with existing BASIC programs. Any space included in variable names are ignored.

EXAMPLES: CIA1 =56320
CIA2=56576
SID=$D400
MESSAGE$="PRESS ANY KEY TO BEGIN"
KEYBOARD%=PEEK(197)

A new command has been included to make use of variables much easier.

LVAR

FORMAT: LVAR
ABBREVIATION: L **shift** V

ACTION: Lists all current numeric variables (integar and floating point) together with their values, in the order that they were created. This command is very useful when writing machine code programs, as it can be used to examine the values of all labels after assembly.

The Assembler

Assembler source code is written on ordinary BASIC lines, with special commands to initiate and finish assembly. Standard 6510 (6502) mnemonics are used and no spaces are necessary between the assembler mnemonic and the operand. Multi-statement lines are allowed, although it is wise to use only single statement lines as these are much easier to read and alter at a later stage.

PC

A new system variable PC (program counter) controls the location in memory where the next machine code instruction will be placed. When the assembler is loaded, PC is set automatically to 49152 (hex C000), the start of a 4K block of memory just above the BASIC interpreter, which is not affected by BASIC operations. However, PC can be changed at any time and should be set to the start of any piece of machine code before it is assembled.

EXAMPLES: PC=16384
PC=BEGINNING

The location where the next machine code instruction will be assembled can be found by examining the value of PC.

EXAMPLES: PRINTPC
FINISH=PC

PC can also be changed at any time during assembly, by placing the appropriate instruction in the source code. This allows several pieces of machine code to be assembled at different places in memory during the same assembly.

Entering and Exiting the Assembler

When the command ENTER is encountered in a program, control is handed over to the assembler and pass one is initiated. The position of the ENTER command is stored for pass two, as is the current value of PC. During pass one, all label addresses are calculated and the syntax of the machine code instructions is checked. The current value of PC, which is updated continuously is used in place of all unknown labels, so that the addressing modes of instructions in which they occur can be calculated accurately.

When the command EXIT, or the end of the program is encountered during pass one, pass two of the assembler is initiated. During this pass, the source code is actually assembled and stored in memory. When EXIT, or the end of the program is encountered during this pass, control is handed back to the BASIC interpreter. This allows machine code to be mixed with BASIC, although the full power of machine code can only be realised if it is used alone.

The Source Code

As stated earlier, the assembler uses standard 6510 (6502) mnemonics. In the following, XXX represents a three letter assembler mnemonic.

ADDRESSING MODE	FORMAT
ACCUMULATOR	XXX A
IMMEDIATE	XXX operand
ABSOLUTE	XXX operand
ZERO PAGE	XXX operand
INDEXED ABSOLUTE	XXX operand, X or LDA operand, Y
INDEXED ZERO PAGE	XXX operand, X or LDA operand, Y
INDEXED INDIRECT	XXX (operand, X)
INDIRECT INDEXED	XXX (operand), Y
INDIRECT ABSOLUTE	XXX (operand)
RELATIVE	XXX operand or XXX operand (see below)

Accumulator Addressing

Because of the existence of this addressing mode, it is unwise to use the variable A as a label. In some cases, it could be read as part of an assembler instruction.

Branch Instructions

Branch instructions require an offset, which is added to the low byte of the current program counter if the branch is valid. In practice it is tedious to have to calculate the offset, so the assembler calculates it automatically from the current value of PC and the operand, which of course can be a label.

```
EXAMPLES: BNE FINISH
          BCS $COFB
```

The assembler allows an offset to be used if desired. If the branch is forwards, count the number of bytes between the branch instruction and the destination instruction, without including either. If the branch is backwards, count the number of bytes between the destination instruction and the branch instruction, including both, and subtract this from 256 (hex 100). The result is the offset, which should be used as the operand.

```
EXAMPLES: BNE#3
          BEQ#$FO
```

All numeric operands can be any numeric expression, providing it can be reduced to a number in the correct range. The assembler automatically uses zero page addressing in place of absolute addressing wherever possible.

Comments

Anything following a semi-colon in the source code is treated as a comment (similar to a REM statement in a BASIC program). Comments can appear alone on a line or they can follow an instruction.

```
EXAMPLES: 210 LDA  LO(IRQVEC); GET LOW BYTE
          1230; ROUTINE TO GET DIRECTION
          3500 RTS: EXIT ROUTINE
```

NOTE: Reserved keywords in a comment are still tokenized (stored as one byte in the source program). Therefore, abbreviations of keywords will appear as the entire keyword when the program is listed (? will appear as PRINT etc.). This is unfortunate, but necessary because the semi-colon is also used in PRINT statements, where the code following must be tokenized.

Defining Labels

Labels are defined by placing a full stop followed by the label name at the relevant position in the source code. Labels are treated as ordinary floating point variables and are stored in the same way. Each label takes up seven bytes plus the length of the name in memory. It is therefore wise to keep the names fairly short. A sensible length is from four to eight characters. The label name should indicate what the machine code routine is to do, as this makes "de-bugging" the program at a later stage much easier. (See also the section on VARIABLES and LABELS).

```
EXAMPLES: 100 .TRANSFER
          170 .HELP
          1870 .BIGBANG
```

Assembler Directives

BYT

BYT is followed by single byte numbers (or numeric expressions which can be reduced to a single byte) separated by commas. These numbers are stored in memory in the order in which they appear in the source code.

```
EXAMPLES: 130 BYT$FF,$CO,223,ASC ("H"),2
          1120 BYTO,1,2,128,129,130
```

WRD

WRD is followed by double byte numbers (or numeric expressions which can be reduced to two bytes) separated by commas. The numbers are stored in memory in standard 6510 format (the low byte followed by the high byte).

```
EXAMPLES: 210 WRDIRQVEC, NMIVEC, DEEK(792)
          330 WRD$COOO, $COFB, $C11D
```

NUM

NUM is similar to WRD, but the numbers are stored with the high byte first.

```
EXAMPLES: 1710 NUM1000,2000,3000,4000
          2130 NUMSC1,SC2,SC3
```

TXT

TXT is followed by a string expression. The ASCII code of each character in the string is stored in memory.

EXAMPLES: 1200 TXT"PRESS ANY KEY TO BEGIN"
1930 TXTMESSAGE1$+MESSAGE2$

Display of Assembly Code

SETHEX

FORMAT: SETHEX
ABBREVIATION: S **shift** E

This command sets output of the assembler and disassembler to hexadecimal mode. When the assembler is loaded and when RUN/STOP and RESTORE are pressed, output is automatically set to this mode.

SETDEC

FORMAT: SETDEC
ABBREVIATION: SET **shift** D

This command sets output of the assembler and disassembler to decimal mode.

OFF

This assembler directive switches output from the assembler off.

ON

This assembler directive switches output from the assembler on. During assembler pass two, the code is sent to the current output device (see output to the printer and the OUT command). Output is automatically switched on when the ENTER command is executed.

EXAMPLES OF OUTPUT FORMAT

HEXADECIMAL MODE	C000	20	A9	CA	JSR SBRTN1
	C003	A9	20		LDA $20
	C005	20	D2	FF	JSR $FFD2
DECIMAL MODE	49152	32	169	202	JSR SBRTN1
	49155	169	32		LDA $20
	49157	32	210	255	JSR $FFD2

Assembler Error Messages

Pass One Errors

Redef'd Label Error

The same label has been defined twice, or a label has been defined with the same name as an existing floating point variable. This error message will also occur if the same piece of code is assembled twice, without clearing the variables after the first assembly.

Syntax Error

This error can occur in a number of different circumstances. A label name which contains a reserved keyword will generate this error, as will an assembler mnemonic that is not recognised. Although spaces are not usually required between mnemonics and operands, they are necessary in some cases, for example in the instruction STA NAME. Without the space, the TA and N would be tokenized as the function TAN, which of course would not be recognised by the assembler. There are several ways in which this tokenization can occur and if an unexpected syntax error is reported, this may well be what is wrong.

Pass Two Errors

Branch Out of Range Error

This error message occurs when a branch instruction could not branch as far as the destination address. Relative branches are limited to distances of 127 bytes forwards and 128 bytes backwards. If this error occurs, use a different construction for the branch.

For example, replace BEQ LABEL1 with BNE LABEL2
JMP LABEL1
LABEL2

Undef'd Variable Error

this error indicates that either a label or a variable used in an operand has not been defined.

In addition to these errors, all normal BASIC errors concerning numeric operations can occur during pass two.

The Disassembler

DISASSEMBLE

FORMAT: DISASSEMBLEstart,end
ABBREVIATION: DIS **shift** A

this command disassembles the area of memory from the start address to the end address. If the comma is included and the end address is omitted, the disassembly will not stop until the RUN/STOP key is pressed. If both the comma and the end address are omitted, only the start address will be disassembled. Disassembly can be paused by pressing the space bar and re-started in the same way. While the pause is on, pressing return will insert blank lines. Pressing the H key switches output to hexadecimal mode and the D key switches output to decimal mode. Disassembly can be stopped at any time by pressing the RUN/STOP key. See also the SETHEX, SETDEC and OUT commands.

Disassembly Format

Address Byte 1, Byte 2, Byte 3, Assembler Code, ASCII Codes Screen Codes.

Dots are substituted for those ASCII characters that would cause a messy screen display. The screen codes are the characters that would result if BYTE1, BYTE2 and BYTE3 were poked to the Commodore 64 screen.

EXAMINE

FORMAT: EXAMINEstart,end
ABBREVIATION: EXA **shift** M

This command lists the contents of the memory locations from the start address to the end address, giving them in decimal, hexadecimal and binary. If the comma is included and the end address is omitted, listing will continue until the RUN/STOP key is pressed. If both the comma and the end address are omitted, only the contents of the start address will be listed. The controls are the same as those for disassembly. See also the SETHEX, SETDEC and OUT commands.

Examine Format

Address Decimal, Hexadecimal, Binary, ASCII Code, Screen Code.

Where the ASCII code is a control character, the effect of this character is printed in reverse field.

BLACK	Forground colour Black
WHITE	Forground colour White
RED	Forground colour Red
CYAN	Forground colour Cyan
PURPLE	Forground colour Purple
GREEN	Forground colour Green
BLUE	Forground colour Blue
YELLOW	Forground colour Yellow
ORANGE	Forground colour Orange
BROWN	Forground colour Brown
L. RED	Forground colour Light Red
D. GREY	Forground colour Dark Grey
M. GREY	Forground colour Medium Grey
L. GREEN	Forground colour Light Green
L. BLUE	Forground colour Light Blue
L. GREY	Forground colour Light Grey
L-CASE	Switch to lower case
U-CASE	Switch to upper case
C/S ON	Enable commodore/shift
C/S OFF	Disable commodore/shift
SPACE	Space
CRSR-L	Move cursor left
CRSR-R	Move cursor right
CRSR-U	Move cursor up
CRSR-D	Move cursor down
RVS ON	Turn reverse field on
RVS OFF	Turn Reverse field off
CLR	Clear screen
HOME	Home cursor
INSERT	Insert space
DELETE	Delete character
FUNC 1	Function key 1
FUNC 2	Function key 2
FUNC 3	Function key 3
FUNC 4	Function key 4
FUNC 5	Function key 5
FUNC 6	Function key 6
FUNC 7	Function key 7
FUNC 8	Function key 8
RETURN	Carriage return
S/RTN	Shift return

OUTPUT TO DEVICES OTHER THAN THE SCREEN

Out

FORMAT: OUTfile, string
ABBREVIATION: O **shift** U
ACTION: This command has exactly the same effect as the CMD command in commodore BASIC (which is still available), with the exception that output is sent to the screen as well as to the file. This is necessary because of the way in which the assembler and disassembler format their output. CMD will NOT work with the assembler and disassembler.
EXAMPLES: OUT4
OUTF
OUT4,MESSAGE$
To send a disassembly to a printer, merely open a file to the printer, use the OUT command and instruct the computer to disassemble the code.
EXAMPLES: OPEN4,4:OUT4:DISASSEMBLE$C1AD,$C20A:PRINT 4:CLOSE4
OPEN4:EXAMINE49200,49317:PRINT 4:CLOSE4
NOTE: The screen codes will not be sent to the printer.

EXTENSIONS TO BASIC

Disk Commands

In all DISK COMMANDS, device and drive numbers are optional. If they are left out, the computer assumes that the device number is eight and the drive number is zero. the drive number should be omitted if you are using a single drive.

DIR

FORMAT: DIRdevice, drive
ABBREVIATION: NONE

ACTION: Lists the directory of a disk.

EXAMPLES: DIR
DIRA
DIRA%, B%
DIR9, 1

STATUS

FORMAT: STATUSdevice
ABREVIATION: ST **shift** A

ACTION: Reads and displays the disk error channel.

EXAMPLES: STATUS
STATUS10
STATUSA

COPY

FORMAT: 1 COPY"newname=oldname",device,drive
2 COPY"newfile=oldfile,oldfile2,oldfile3,
oldfile 4",device,drive

ABBREVIATION: CO **shift** P

ACTION: 1.Makes a copy of a program or file on a disk.
2.Combines from two to four sequential files ACTION: on a disk.

EXAMPLES: COPY"BACKUP=ORIGINAL"
COPY"MASTERFILE=NAMES,ADDRESSESS,PHONE NUMBERS",9
COPY"GAME=OBJECT CODE",8,1

DISK

FORMAT: DISK "name, ID",device,drive
ABBREVIATION: DI **shift** S

ACTION: Formats a disk. The ID is a two character code wihch should be different for each disk. This command can also be used to rename and clear the directory of a disk which has already been formatted. To do this , the ID code must be omitted.

EXAMPLES: DISK"OBJECT CODE,OC"
DISK"GAMES DISK",9

INIT

FORMAT: INITdevice,drive
ABBREVIATION: IN **shift** I

ACTION: Initialises the disk drive, copying the Block Availability Map (BAM) from the disk into drive memory. This command must be used when two disks with the same ID code are exchanged.

EXAMPLES: INIT
INITA,B
INIT9

RENAME

FORMAT: RENAME 'newname=oldname',device,drive
ABBREVIATION: RE **shift** N

ACTION: Renames a file on the disk.

EXAMPLES: RENAME"OBJECT CODE=GAME OBJ"
RENAME"NAMES=FILE1",9
RENAMEN$+"="+O$,10,1

SCRATCH

FORMAT: SCRATCH 'name',device,drive
ABBREVIATION: S **shift** C

ACTION: Erases a file from the disk.

EXAMPLES: SCRATCH"SOURCE CODE"
SCRATCHA$,9

TIDY

FORMAT: TIDYdevice,drive
ABBREVIATION: T **shift** I

ACTION: Validates a disk. This reorganises the disk, creating a new BAM. This command should never be used on a disk which contains random files.

EXAMPLES: TIDY
TIDY8,1
TIDYA%

PROGRAMMING COMMANDS

Programming Commands

APPEND

FORMAT: APPEND "name",device
ABBREVIATION: A **shift** P

ACTION: Appends a program on tape or disk to the one currently in memory. The appended program will appear at the end of the original program when it is listed. This command generates a SYNTAX ERROR if used in a program.

EXAMPLES: APPEND "PROG2"
APPENDA$,8

AUTO

FORMAT: AUTOstart, increment
ABBREVIATION: A **shift** U

ACTION: Generates line numbers automatically. The start and increment are optional and if left out, the computer assumes that both are ten. If the current line number is changed using the cursor, the increment will be added to the new number to give the next line. To return to normal operation, either enter a blank line (which will not be deleted) or enter an immediate command.

EXAMPLES: AUTO
AUTO100,1
AUTO1000

DELETE

FORMAT: DELETEfirstline-lastline
ABBREVIATION: DE **shift** L

ACTION: Deletes the specified range of program lines.

EXAMPLES: DELETE10-350
DELETE-400
DELETE500-

FIND

FORMAT: FINDstring
ABBREVIATION: F **shift** I

ACTION: Searches a program for occurences of a string and lists all lines in which it is found.

EXAMPLES: FINDPRINT
FINDJSR $FFDB

OLD

FORMAT: OLD
ABBREVIATION: O **shift** L

ACTION: Recovers a program when it has been NEWed or a cold reset has been performed. This command will not work it new program lines have been entered or variables have been defined.

RENUMBER

FORMAT: RENUMBERstart,increment
ABBREVIATION: REN **shift** U

ACTION: Renumbers a program, changing all RUN, GOTO, GOSUB, ON...GOTO, ON...GOSUB, THEN and RESTORE commands. The start and increment are optional and if left out, the computer assumes that both are ten.

EXAMPLES: RENUMBER
RENUMBER100,1
RENUMBER1000

RKILL

FORMAT: RKILL
ABBREVIATION: R **shift** K

ACTION: Removes all REM statements on multi-statement lines and the comments following REM statements if they occur alone on a line, so that GOTO and GOSUB statements refering to such lines still work.

Memory Handling Commands

BLOAD

FORMAT:BLOAD"name",start location,device
ABBREVIATION: B **shift** L

ACTION: Loads a program from tape or disk at the start location. This command will not work if the program was saved on tape with a secondary address of one. Instead, the program would load at the location from which it was saved.

EXAMPLES: BLOAD"OBJECT CODE",$COOO,8
BLOADA$,S

BSAVE

FORMAT: BSAVE"name",start location,end location+1,device,secondary address
ABBREVIATION: B **shift** S

ACTION: Saves the block of memory from the start location to the end location. The secondary address only applies if saving to tape. A secondary address of one means that the program will always load at the same location in memory. A secondary address of two causes an end-of-tape marker to follow the program and a secondary address of three combines both these functions.

EXAMPLES: BSAVE"OBJECT CODE",$COOO,$DOOO,8
BSAVEA$,S,E+1
BSAVE"O:GAME OBJ",16384,32768,10

BVERIFY

FORMAT: BVERIFY"name",device
ABBREVIATION: B **shift** V

ACTION: Compares the program on tape or disk with the contents of memory starting at the location from which it was saved. If using tape, the name and device can be ommitted. If no name is given, the first program found will be verified.

EXAMPLES: BVERIFY"OBJECT CODE",8
BVERIFY
BVERIFY"O:GAME OBJ",10

CLEAR

FORMAT: CLEARlocation
ABBREVIATION: CL **shift** E

ACTION: Sets the top of memory and clears all variables and labels. Since the KERNAL MEMTOP routine is called, no RS232 channels should be opened before using this command.

EXAMPLES: CLEAR$4000
CLEARA

DOKE

FORMAT: DOKElocation,number
ABBREVIATION:D **shift** O

ACTION: Stores a two-byte number in memory at a given location in standard 6510 format, i.e., the low byte of the number followed by the high byte.

EXAMPLES: DOKE$4000,$FFFF
DOKEA,B
DOKE788,49152

FILL

FORMAT: FILLstart,end,number
ABBREVIATION: FI **shift** L

ACTION: Fills the memory from the start location to the end location with the given number.

EXAMPLES: FILL$4000,$5000,0
FILLS,E,N
FILL8192,16384,255

MOVE

FORMAT: MOVEstart,end,destination
ABBREVIATION: M **shift** O

ACTION: Moves the given block of memory to the destination location.

EXAMPLES: MOVE$4000,$5000,$6000
MOVES,E,D
MOVE14*4096,65535,16384

Functions

DEEK

FORMAT: DEEK(location)
ABBREVIATION: DE **shift** E

ACTION: Returns an integer in the range 0 to 65535 (hex 0 to FFFF) which is read from two consecutive memory locations, the first of which contains the low byte of the number and the second the high byte.

EXAMPLES: PRINTDEEK($FFFC)
IFDEEK(I)=5000THENPRINTI

HI

FORMAT: HI(number)
ABBREVIATION: NONE

ACTION: Returns the high byte of a number in the range 0 to 65535 (hex 0 to FFFF).

EXAMPLES: PRINTHI(59953)
H=HI(A)

LO

FORMAT: LO(number)
ABBREVIATION: NONE

ACTION: Returns the low byte of a number in the range 0 to 65535 (hex 0 to FFFF)

EXAMPLES: PRINTLO (59953)
L=LO(A)

Special Functions

Bin

FORMAT: PRINTBINnumber
ABBREVIATION: B **shift** I

ACTION: Prints the Binary Form of a number in the Range 0 to 65535 (hex 0 to FFFF)

EXAMPLES: PRINT BIN 234D
EXAMPLES: PRINT BINA+B*256

HEX

FORMAT: Print Hex Number
ABBREVIATION: H shift E

ACTION: Prints the hexadecimal form of a number in the range 0 to 65535 (hex 0 to FFFF).

EXAMPLES: PRINTHEX%1110011000
PRINTHEXA/256+B

$

ACTION: The $ symbol precedes a hexadecimal number in the range 0 to FFFF. Hexadecimal numbers are evaluated much faster than decimal number and are also much faster than decimal numbers and are also much more convenient when dealing with memory, so it is wise to use them wherever possible.

EXAMPLES: A=$AB9F
PRINTBIN$2FBC
POKEA,$80

%

ACTION: The % symbol precedes a binary number in the range 0 to 1111111111111111. Binary numbers are useful when designing characters and when it is important to affect only certain bits of a memory location. As with hexadecimal numbers, binary numbers are evaluated much faster than equivalent decimal numbers.

EXAMPLES: PRINT%1101101
POKECH,%10011001
POKE53270,PEEK(53270)AND%11001111

Extended Commands

ASC

This command now returns zero when it is used with a null string.

EXAMPLES: PRINTASC (" ")

FRE

This command now returns the correct value for the amount of free memory, even if it exceeds 32K. It also no longer requires a 'dummy value'.

EXAMPLES: PRINTFRE
X=FRE

LIST

This command now lists only line 0 when the instruction LISTO is executed.

POS

This command no longer requires a 'dummy value'.

EXAMPLES: PRINTPOS
P=POS

RESTORE

This command can now be used to RESTORE the data pointer to a given line.

EXAMPLES: RESTORE
RESTORE2000

NOTE: The INPUT, READ and VAL commands have been extended so that they will now accept hexadecimal and binary numbers.

THE CHARACTER EDITOR

Operation

The character editor can be used to define two different character sets at the same time.

Set 1 - 12288 to 14335, $3000 to $37FF
Set 2 - 14336 to 16383, $3800 to $3FFF

Loading and saving of character sets is performed while the editor is NOT running, using the BLOAD and BSAVE commands. This allows relocation of the characters in memory. One or both sets can be saved at the same time.

Loading

BLOAD"data name",address,device

The address is the start location of the character set in memory (see the BLOAD command). The device number can be omitted if using tape.

EXAMPLES

BLOAD"SET 1", 12288,8
BLOAD"SET 2", $3800
BLOAD"BOTH SETS", $3000,9

Saving

BSAVE"data name",start,end,device

The start number is the start address of the character set in memory and the end number is the finish address plus 1 (see the BSAVE command). The device number can be omitted if using tape.

EXAMPLES

BSAVE"SET 1", 12288, 14336,8
BSAVE"SET 2", $3800, $4000
BSAVE"BOTH SETS", $3000, $4000,9

Of course, the character sets need not be loaded in the specified area of memory once they have been defined. By changing the address in the BLOAD command, a character set can be loaded almost anywhere in memory.

The Editor Itself

Before running the character editor, any previously defined characters should be loaded as described above.

To break out of the editor to load or save a character set, hold down RUN/STOP and press RESTORE. The editor and character sets will still be in memory, totally unaffected. In fact, because the editor sets the top of memory to 12288 ($3000) when it is run, the character sets are virtually impossible to corrupt accidentally.

Brief instructions describing the functions of various keys are included in the editor, but for convenience there follows a list of all the keys and their functions:

KEY	FUNCTION
Q	Plot Pixel
W	Unplot Pixel
1	Change background colour of upper portion of screen
2	Change background colour of lower portion of screen
3	Change foreground colour of upper portion of screen
4	Change foreground colour of lower portion of screen
5	Change border colour
6	Change multicolour 1/extended colour 1
7	Change multicolour 2/extended colour 2
8	Change extended colour 3
F6	Transfer ROM character set to RAM (UPPER or LOWER case)
F8	Clear character set (use carefully!)
M	Switch to multicolour mode
E	Switch to extended colour mode
I	Invert character
+	Rotate character +90 degrees
—	Rotate character —90 degrees
L	Roll character left one pixel
R	Roll character right one pixel
U	Roll character up one pixel
D	Roll character down one pixel
F1	Get character from set (RETURN to exit)
F3	Choose new destination character (RETURN to exit)
F5	Switch to character set 1
F7	Switch to character set 2

In addition to these functions, the cursor control keys operate as usual, as do HOME and CLR (which clears the character). In addition, all keys have an auto-repeat.

How the Program Works

The character editor, besides being a useful and versatile utility, also shows how many of the extra commands are used.

There follows a description of the functions of some of the major parts of the program, for reference when learning how the assembler and the extra commands work:

LINE 10

Sets top of memory to 12288 ($3000) thus protecting the character sets.

LINES 20-40

Define variables for the assembler source code.

LINE 50

Tells the assembler to begin assembly at address 49152 ($C000) - the start of a free 4K buffer area above BASIC.

LINES 60-70

Enter assembler mode and switch output off.

LINES 80-3550

This is the assembler source code (see the section on the assembler). The routine labelled IRQ handles the split screen graphics used in the character editor. All of the routines have been given fairly explanatory labels (a wise practise!) and it should be relatively simple to determine which routines handle which functions.

LINE 3560

Exits assembler mode, returning to normal operation.

LINES 3570-5510

These lines contain only BASIC commands and all subroutines carry a REM statement explaining their particular function. Notice the use of explanatory variable names which have names more than two letters long. Try breaking out of the editor while it is running, then use the LVAR command to see the values of all currently defined numeric variables and labels - this command can be extremely useful for 'de-bugging'.