The ROM Disassembly

The Oric contains a ROM (Read Only Memory) which contains all the machine code programs needed to implement Basic. This large machine code program is permanent and is not erased when the power to the computer is turned off. When the Oric is powered up, this program will run automatically and it enables user programs to be typed in and run.

On the Oric, the routines are divided into two main sections, those that comprise the Basic language, which lies towards the front of the ROM (between #C000 and #ECC3) and those that make up the operating system (which lies between #ECC4 and #FFFF).

The Basic language acts like a large and powerful microprocessor. It can handle real numbers, strings and loops, perform complicated mathematical functions, draw on the screen, generate sounds and make complicated decisions. Such commands cannot be given to the 6502 directly, they have to be broken down into instructions which it can execute. The purpose of the machine code routines in which Basic is written is to break down the statements of a program into instructions that can be run on the 6502.

Real numbers and results of mathematical operations are stored in F1oating Point Accumulators (FPA) as though they were registers used by Basic. Each floating point accumulator consists of 6 bytes of memory, one to hold a signed exponent, four to hold the mantissa and the last to hold the sign of the mantissa (see section 6.2.8). Basic has two FPAs in which mathematical operations such as add and divide are performed. The main and work FPAs each hold an operand at the start of each mathematical operation and the result is left in the main FPA. Other locations in memory are used as temporary storage for the FPAs when complicated expressions are being evaluated.

The same FPAs are used when handling integers and strings but the format of data within each is different.

Many of the other many memory locations used by Basic are used to store information about the program it is running and where all the current variables are.

The other major part of the ROM, the operating system, consists of a series of routines which are used by *Basic* to input and output data. The operating system routines are specific to the Oric and handle such things as input from the keyboard, writing to the screen or printer and loading or saving from the cassette system. It also requires a section of memory to hold all its variables, most of which are in page 2 of memory.

The listing below is that of the Oric Atmos ROM (V1.1) which is an updated version of that used in the Oric 1 (V1.0). The only differences between the two are the correcting of errors and the addition of two new keywords. For example, the original version did not allow the **POKE**ing of hexadecimal numbers into memory and that 13 had to be added to the argument in the **TAB** statement. The two new keywords, **STORE** and **RECALL** allow the saving and loading of arrays from cassette. Their tokens respectively replace those of **INVERSE** and **NORMAL** on the Oric 1 which both give '`SYNTAX ERROR`'.

Note that standard 6502 assembly syntax has been used in which a '$' before a number is used to represent a hexadecimal argument and a '#' is used to represent an immediate argument.

```
C000    4C CC EC    JMP $ECCC       Jump to START BASIC
C003    4C 71 C4    JMP $C471       Jump to RESTART BASIC

C006    72 C9 91 C6 86 E9 D0 E9     JUMP TABLE for each of the
C00E    15 CD 18 CD 11 CA 50 DA     commands, in token order. The
C016    A0 DA DD D9 66 D9 84 DA     table is in two halves, firstly
C01E    A0 DA 54 C8 FC C7 08 C8     for those commands which may
C026    97 CE 3B CA 54 CD 7D D1     start a statement and secondly
C02E    CD CC 88 CD 1B CB E4 C9     for those which may not. Some
C036    BC C9 6F CA 51 C9 C7 C9     tokens do not have start
C03E    11 CA 98 CA CD EB E6 EB     addresses - see Appendix A.
C046    0B EC 20 EC 32 EC B4 FA     The values in the first part of
C04E    CA FA E0 FA 9E FA FB EA     the table are one less than the
C056    FB EA FB EA EF EA EF EA     start address of the routines.
C05E    EF EA EF EA EF EA EF EA     This is because the RTS
C066    EF EA FB EA FB EA 70 C9     instruction is used as an
C06E    C1 CA 57 D9 5A E8 08 E9     indirect jump which
C076    B9 D4 4E D9 AA CB 9F C9     automatically increments the
C07E    47 C7 0C C7 45 CD 45 E9     address by 1.
C086    12 CD ED C6 21 DF BD DF
C0BE    49 DF 21 00 7E D4 A6 D4
C096    B5 D9 FB 02 2E E2 4F E3
C09E    AF DC AA E2 8B E3 92 E3
C0A6    DB E3 3F E4 38 D9 83 D9
C0AE    D4 DD A6 D8 93 D5 D7 D8
C0B6    B5 D8 16 D8 77 DE 0F DF
C0BE    0B DF DA DA 3F DA 45 EC
C0C6    2A D8 56 D8 61 D8 79 24
C0CE    DB 79 0D DB 7B EF DC 7B
C0D6    E6 DD 7F 37 E2 50 E5 D0
C0DE    46 E2 D0 7D 70 E2 5A 3B
C0E6    D0 64 12 D1 45 4E C4 45         ENDE    BASIC KEYWORDS
C0EE    44 49 D4 53 54 4F 52 C5     DITSTORE
C0F6    52 45 43 41 4C CC 54 52     RECALLTR    The last character
C0FE    4F CE 54 52 4F 46 C6 50     ONTROFFP    of a keyword has bit
C106    4F D0 50 4C 4F D4 50 55     OPPLOTPU    7 set.
C10E    4C CC 4C 4F 52 45 D3 44     LLLORESD
C116    4F 4B C5 52 45 50 45 41     OKEREPEA
C11E    D4 55 4E 54 49 CC 46 4F     TUNTILFO
C126    D2 4C 4C 49 53 D4 4C 50     RLLISTLP
C12E    52 49 4E D4 4E 45 58 D4     RINTNEXT
C136    44 41 54 C1 49 4E 50 55     DATAINPU
C13E    D4 44 49 CD 43 4C D3 52     TDIMCLSR
C146    45 41 C4 4C 45 D4 47 4F     EADLETCO
C14E    54 CF 52 55 CE 49 C6 52     TORUNIFR
C156    45 53 54 4F 52 C5 47 4F     ESTOREGO
C15E    53 55 C2 52 45 54 55 52     SUBRETUR
C166    CE 52 45 CD 48 49 4D 45     NREMHIME
C16E    CD 47 52 41 C2 52 45 4C     MGRABREL
C176    45 41 53 C5 54 45 58 D4     EASETEXT
C17E    48 49 52 45 D3 53 48 4F     HIRESSHO
C186    4F D4 45 58 50 4C 4F 44     OTEXPLOD
C18E    C5 5A 41 D0 50 49 4E C7     EZAPPING
C196    53 4F 55 4E C4 4D 55 53     SOUNDMUS
C19E    49 C3 50 4C 41 D9 43 55     ICPLAYCU
C1A6    52 53 45 D4 43 55 52 4D     RSETCURM
C1AE    4F D6 44 52 41 D7 43 49     OVDRAWCI
C1B6    52 43 4C C5 50 41 54 54     RCLEPATT
C1BE    45 52 CE 46 49 4C CC 43     ERNFILLC
C1C6    48 41 D2 50 41 50 45 D2     HARPAPER
C1CE    49 4E CB 53 54 4F D0 4F     INKSTOPO
C1D6    CE 57 41 49 D4 43 4C 4F     NWAITCLO
C1DE    41 C4 43 53 41 56 C5 44     ADCSAVED
C1E6    45 C6 50 4F 4B C5 50 52     EFPOKEPR
```

```
C1EE    49 4E D4 43 4F 4E D4 4C     INTCONTL
C1F6    49 53 D4 43 4C 45 41 D2     ISTCLEAR
C1FE    47 45 D4 43 41 4C CC A1     CETCALL!
C206    4E 45 D7 54 41 42 A8 54     NEWTAB(T
C20E    CF 46 CE 53 50 43 A8 C0     OFNSPC(@
C216    41 55 54 CF 45 4C 53 C5     AUTOELSE
C21E    54 48 45 CE 4E 4F D4 53     THENNOTS
C226    54 45 D0 AB AD AA AF DE     TEP+-*/A
C22E    41 4E C4 4F D2 BE BD BC     ANDOR)=<
C236    53 47 CE 49 4E D4 41 42     SGNINTAB
C23E    D3 55 53 D2 46 52 C5 50     SUSRFREP
C246    4F D3 48 45 58 A4 A6 53     OSHEX$&S
C24E    51 D2 52 4E C4 4C CE 45     QRRNDLNE
C256    58 D0 43 4F D3 53 49 CE     XPCOSSIN
C25E    54 41 CE 41 54 CE 50 45     TANATNPE
C266    45 CB 44 45 45 CB 4C 4F     EKDEEKLO
C26E    C7 4C 45 CE 53 54 52 A4     GLENSTR$
C276    56 41 CC 41 53 C3 43 48     VALASCCH
C27E    52 A4 50 C9 54 52 55 C5     R$PITRUE
C286    46 41 4C 53 C5 4B 45 59     FALSEKEY
C28E    A4 53 43 52 CE 50 4F 49     $5CRNPOI
C296    4E D4 4C 45 46 54 A4 52     NTLEFT$R
C29E    49 47 48 54 A4 4D 49 44     IGHT$MID
C2A6    A4 00 4E 45 58 54 20 57     $ NEXT W
C2AE    49 54 48 4F 55 54 20 46     ITHOUT F
C2B6    4F D2 53 59 4E 54 41 D8     ORSYNTAX
C2BE    52 45 54 55 52 4E 20 57     RETURN W
C2C6    49 54 48 4F 55 54 20 47     ITHOUT G
C2CE    4F 53 55 C2 4F 55 54 20     OSUBOUT
C2D6    4F 46 20 44 41 54 C1 49     OF DATAI
C2DE    4C 4C 45 47 41 4C 20 51     LLEGAL Q
C2E6    55 41 4E 54 49 54 D9 4F     UANTITYO
C2EE    56 45 52 46 4C 4F D7 4F     VERFLOWO
C2F6    55 54 20 4F 46 20 4D 45     UT OF ME
C2FE    4D 4F 52 D9 55 4E 44 45     MORYUNDE
C306    46 27 44 20 53 54 41 54     F'D STAT
C30E    45 4D 45 4E D4 42 41 44     EMENTBAD
C316    20 53 55 42 53 43 52 49      SUBSCRI
C31E    50 D4 52 45 44 49 4D 27     PTREDIM'
C326    44 20 41 52 52 41 D9 44     D ARRAYD
C32E    49 56 49 53 49 4F 4E 20     IVISION
C336    42 59 20 5A 45 52 CF 49     BY ZEROI
C33E    4C 4C 45 47 41 4C 20 44     LLEGAL D
C346    49 52 45 43 D4 44 49 53     IRECTDIS
C34E    50 20 54 59 50 45 20 4D     P TYPE M
C356    49 53 4D 41 54 43 C8 53     ISMATCHS
C35E    54 52 49 4E 47 20 54 4F     TRING TO
C366    4F 20 4C 4F 4E C7 46 4F     O LONGFO
C36E    52 4D 55 4C 41 20 54 4F     RMULA TO
C376    4F 20 43 4F 4D 50 4C 45     O COMPLE
C37E    D8 43 41 4E 27 54 20 43     XCAN'T C
C386    4F 4E 54 49 4E 55 C5 55     ONTINUEU
C38E    4E 44 45 46 27 44 20 46     NDEF'D F
C396    55 4E 43 54 49 4F CE 42     UNCTIONB
C39E    41 44 20 55 4E 54 49 CC     AD UNTIL
C3A6    20 45 52 52 4F 52 00 20      ERROR
C3AE    49 4E 20 00 0D 0A 52 65     IN    Re
C3B6    61 64 79 20 0D 0A 00 0D     ady
C3BE    0A 20 42 52 45 41 4B 00       BREAK
C3C6    BA          TSX             Search for a variable match in
C3C7    E8          INX             FOR-NEXT loop.
C3C8    E8          INX
C3C9    E8          INX             Successive FOR-NEXT loops are
C3CA    E8          INX             pulled off the stack until a
```

ERROR MESSAGES

```
C3CB   BD 01 01    LDA $0101,X     variable match is made.
C3CE   C9 8D       CMP #$8D
C3D0   D0 21       BNE $C3F3       If a match is made then the Z
C3D2   A5 B9       LDA $B9         flag in the status register is
C3D4   D0 0A       BNE $C3E0       set to 1; otherwise it is set
C3D6   BD 02 01    LDA $0102,X     to 0.
C3D9   85 B8       STA $B8
C3DB   BD 03 01    LDA $0103,X
C3DE   85 B9       STA $B9
C3E0   DD 03 01    CMP $0103,X
C3E3   D0 07       BNE $C3EC
C3E5   A5 B8       LDA $B8
C3E7   DD 02 01    CMP $0102,X
C3EA   F0 07       BEQ $C3F3
C3EC   8A          TXA
C3ED   18          CLC
C3EE   69 12       ADC #$12
C3F0   AA          TAX
C3F1   D0 D8       BNE $C3CB
C3F3   60          RTS
C3F4   20 44 C4    JSR $C444       This routine opens up new
C3F7   85 A0       STA $A0         space in memory to store
C3F9   84 A1       STY $A1         new variables.
C3FB   38          SEC             ($CE/$CF points to start of
C3FC   A5 C9       LDA $C9         block, $C9/$CA points to end
C3FE   E5 CE       SBC $CE         of block, $C7/$C8 points to
C400   85 91       STA $91         new end of block).
C402   A8          TAY             X(MSB) and Y (LSB) hold the
C403   A5 CA       LDA $CA         size of block.
C405   E5 CF       SBC $CF
C407   AA          TAX
C408   E8          INX             Branch if block is a whole
C409   98          TYA             number of pages.
C40A   F0 23       BEQ $C42F
C40C   A5 C9       LDA $C9         Point $C9/$CA and $C7/$C8 to
C40E   38          SEC             the bottom of the pages that
C40F   E5 91       SBC $91         they were pointing to.
C411   85 C9       STA $C9
C413   B0 03       BCS $C418
C415   C6 CA       DEC $CA
C417   38          SEC
C418   A5 C7       LDA $C7
C41A   E5 91       SBC $91
C41C   85 C7       STA $C7
C41E   B0 08       BCS $C428
C420   C6 C8       DEC $C8
C422   90 04       BCC $C428
C424   B1 C9       LDA ($C9),Y     Shift whole page up in memory.
C426   91 C7       STA ($C7),Y
C428   88          DEY
C429   D0 F9       BNE $C424
C42B   B1 C9       LDA ($C9),Y     Shift last byte (when Y=0)
C42D   91 C7       STA ($C7),Y
C42F   C6 CA       DEC $CA         Decrement pointer page numbers
C431   C6 C8       DEC $C8
C433   CA          DEX             Continue until all pages have
C434   D0 F2       BNE $C428       been moved.
C436   60          RTS

C437   0A          ASL A           Check for 2 x content of A
C438   69 3E       ADC #$3E        free bytes on stack. C is set
C43A   B0 40       BCS $C47C       at end of routine if enough
C43C   85 91       STA $91         space is free.
C43E   BA          TSX
```

```
C43F    E4 91       CPX $91
C441    90 39       BCC $C47C
C443    60          RTS

C444    C4 A3       CPY $A3         CHECK FOR FREE MEMORY.
C446    90 28       BCC $C470       A (low) & Y (high) hold new
C448    D0 04       BNE $C44E       end of arrays address. Test
C44A    C5 A2       CMP $A2         and branch if above start of
C44C    90 22       BCC $C470       string memory.
C44E    48          PHA
C44F    A2 09       LDX #$09        Save A, Y on stack, also
C451    98          TYA             contents of $CF to $C7
C452    48          PHA             inclusive.
C453    B5 C6       LDA $C6,X
C455    CA          DEX
C456    10 FA       BPL $C452
C458    20 50 D6    JSR $D650       Attempt Garbage collection.
C45B    A2 F7       LDX #$F7
C45D    68          PLA
C45E    95 D0       STA $D0,X       Restore $C7 to $CF from the
C460    E8          INX             stack.
C461    30 FA       BMI $C45D
C463    68          PLA             Restore A, Y from stack.
C464    A8          TAY
C465    68          PLA
C466    C4 A3       CPY $A3         If the end of the block is
C468    90 06       BCC $C470       still above bottom of string
C46A    D0 10       BNE $C47C       space then jump to print "OUT
C46C    C5 A2       CMP $A2         OF MEMORY ERROR"
C46E    B0 0C       BCS $C47C
C470    60          RTS             Normal finish A, Y unaltered.

C471    AD C0 02    LDA $02C0       NMI routine ends up here.
C474    29 FE       AND #$FE        Remove GRAB status and then
C476    8D C0 02    STA $02C0       restart Basic.
C479    4C A8 C4    JMP $C4A8

C47C    A2 4D       LDX #$4D        PRINT ERROR MESSAGES.
C47E    20 2F C8    JSR $C82F       Reset output to screen.
C481    46 2E       LSR $2E         Reset CTRL 0.
C483    20 F0 CB    JSR $CBF0       Move to start of next line.
C486    20 D7 CC    JSR $CCD7       Print "?" on screen.
C489    BD A8 C2    LDA $C2A8,X     Print error message on screen
C48C    48          PHA             until last char which has bit
C48D    29 7F       AND #$7F        7 set. X holds initial offset
C48F    20 D9 CC    JSR $CCD9       into error table at start of
C492    E8          INX             routine.
C493    68          PLA
C494    10 F3       BPL $C489
C496    20 26 C7    JSR $C726       Reset 6502 stack etc.
C499    A9 A6       LDA #$A6        Print "ERROR" after the
C49B    A0 C3       LDY #$C3        message.
C49D    20 B0 CC    JSR $CCB0
C4A0    A4 A9       LDY $A9         If high byte of line number
C4A2    C8          INY             is #FF then the computer is in
C4A3    F0 03       BEQ $C4A8       immediate mode (not program).
C4A5    20 BA E0    JSR $E0BA       Print "IN (line number>"
C4A8    4E 52 02    LSR $0252       RESTART BASIC
C4AB    46 2E       LSR $2E         Clear pending ELSE, CTRL O
C4AD    4E F2 02    LSR $02F2       and LIST/EDIT flags.
C4B0    A9 B2       LDA #$B2
C4B2    A0 C3       LDY #$C3
C4B4    20 1A 00    JSR $001A       Print "Ready"
C4B7    20 2F C8    JSR $C82F       Reset output to screen.
```

```
C4BA   20 92 C5   JSR $C592       Input line from keyboard.
C4BD   86 E9      STX $E9         Save start of line.
C4BF   84 EA      STY $EA
C4C1   20 E2 00   JSR $00E2       Get next non space char.
C4C4   AA         TAX             If end of line, go back to
C4C5   F0 F0      BEQ $C4B7       get another.
C4C7   A2 FF      LDX #$FF        Set immediate mode.
C4C9   86 A9      STX $A9
C4CB   90 06      BCC $C4D3
C4CD   20 FA C5   JSR $C5FA       Tokenise the line.
C4D0   4C 0C C9   JMP $C90C       Execute the line.

C4D3   20 E2 CA   JSR $CAE2       INSERT / DELETE LINE
C4D6   20 FA C5   JSR $C5FA       Get line and tokenise it.
C4D9   84 26      STY $26         Save line length.
C4DB   20 B3 C6   JSR $C6B3       Look for that line in memory.
C4DE   90 44      BCC $C524       If not found skip line delete

C4E0   A0 01      LDY #$01        DELETE LINE.
C4E2   B1 CE      LDA ($CE),Y     Get MSB of end of line.
C4E4   85 92      STA $92
C4E6   A5 9C      LDA $9C         Get LSB of end of Basic.
C4E8   85 91      STA $91
C4EA   A5 CF      LDA $CF         MSB of start of line.
C4EC   85 94      STA $94
C4EE   A5 CE      LDA $CE         The new end of Basic is the
C4F0   88         DEY             old one plus the start address
C4F1   F1 CE      SBC ($CE),Y     of line being deleted minus
C4F3   18         CLC             its end address. Calculation
C4F4   65 9C      ADC $9C         is done for LSB for each
C4F6   85 9C      STA $9C         parameter and then MSB is
C4F8   85 93      STA $93         adjusted accordingly.
C4FA   A5 9D      LDA $9D
C4FC   69 FF      ADC #$FF
C4FE   85 9D      STA $9D
C500   E5 CF      SBC $CF         X holds number of pages of
C502   AA         TAX             memory to be shifted down.
C503   38         SEC
C504   A5 CE      LDA $CE
C506   E5 9C      SBC $9C         Y holds number of bytes to be
C508   A8         TAY             moved as well as whole pages.
C509   B0 03      BCS $C50E
C50B   E8         INX
C50C   C6 94      DEC $94
C50E   18         CLC             Set up 'from' pointer for
C50F   65 91      ADC $91         block.
C511   90 03      BCC $C516
C513   C6 92      DEC $92
C515   18         CLC
C516   B1 91      LDA ($91),Y     Copy rest of page down.
C518   91 93      STA ($93),Y
C51A   C8         INY
C51B   D0 F9      BNE $C516
C51D   E6 92      INC $92         Advance block pointers to the
C51F   E6 94      INC $94         next page.
C521   CA         DEX             Continue until all pages done.
C522   D0 F2      BNE $C516

C524   20 08 C7   JSR $C708       INSERT LINE
C527   20 5F C5   JSR $C55F       Set text pointer and set up
C52A   A5 35      LDA $35         link pointers. If no line to
C52C   F0 89      BEQ $C4B7       insert branch to immediate
C52E   18         CLC             mode.
C52F   A5 9C      LDA $9C         Calculate the number of bytes
```

```
C531   85 C9       STA $C9          to be shifted and by how far
C533   65 26       ADC $26          so that new line can be
C535   85 C7       STA $C7          inserted.
C537   A4 9D       LDY $9D
C539   84 CA       STY $CA
C53B   90 01       BCC $C53E
C53D   C8          INY
C53E   84 C8       STY $C8
C540   20 F4 C3    JSR $C3F4        0pen up space for new line.
C543   A5 A0       LDA $A0          Set end of Basic to end of
C545   A4 A1       LDY $A1          Arrays (end of block).
C547   85 9C       STA $9C
C549   84 9D       STY $9D
C54B   A4 26       LDY $26          Get number of bytes to insert.
C54D   88          DEY
C54E   B9 31 00    LDA $0031,Y      Transfer new line into the
C551   91 CE       STA ($CE),Y      program.
C553   88          DEY
C554   10 F8       BPL $C54E
C556   20 08 C7    JSR $C708        Set text pointer to start.
C559   20 5F C5    JSR $C55F        Set up line link pointers.
C55C   4C B7 C4    JMP $C4B7        Jump to immediate mode

C55F   A5 9A       LDA $9A          SET LINE LINK POINTERS
C561   A4 9B       LDY $9B          Copy start of Basic into a
C563   85 91       STA $91          pointer.
C565   84 92       STY $92
C567   18          CLC
C568   A0 01       LDY #$01
C56A   B1 91       LDA ($91),Y      Test if at end of program.
C56C   F0 1D       BEQ $C58B
C56E   A0 04       LDY #$04
C570   C8          INY
C571   B1 91       LDA ($91),Y      Step through program until
C573   D0 FB       BNE $C570        end of line is reached.
C575   C8          INY
C576   98          TYA              Add length of line to its
C577   65 91       ADC $91          own start address to get
C579   AA          TAX              start address of next line.
C57A   A0 00       LDY #$00
C57C   91 91       STA ($91),Y      Set pointer to next line (low
C57E   A5 92       LDA $92          byte).
C580   69 00       ADC #$00
C582   C8          INY              Set pointer to next line
C583   91 91       STA ($91),Y      (high byte).
C585   86 91       STX $91          Set pointer to start of
C587   85 92       STA $92          following line.
C589   90 DD       BCC $C568        Do next line.
C58B   60          RTS              Exit.
C58C   CA          DEX              "DEL" - go back one char.
C58D   10 05       BPL $C594
C58F   20 F0 CB    JSR $CBF0

C592   A2 00       LDX #$00         INPUT LINE FROM KEYBOARD.
C594   20 E8 C5    JSR $C5E8        X holds char count. Read key.
C597   C9 01       CMP #$01
C599   D0 0D       BNE $C5A8        Branch if key not CTRL A.
C59B   AC 69 02    LDY $0269        Load char from screen, clear
C59E   B1 12       LDA ($12),Y      bit 7. If it is a CTRL char
C5A0   29 7F       AND #$7F         then replace it by a char to
C5A2   C9 20       CMP #$20         move cursor one place to
C5A4   B0 02       BCS $C5A8        right.
C5A6   A9 09       LDA #$09
C5A8   48          PHA              Save character and print it to
```

```
C5A9   20 D9 CC   JSR $CCD9       the screen.
C5AC   68         PLA
C5AD   C9 7F      CMP #$7F        Branch if char is DEL - go
C5AF   F0 DB      BEQ $C58C       back one character.
C5B1   C9 0D      CMP #$0D        If char is RETURN then finish
C5B3   F0 30      BEQ $C5E5       off current input buffer.
C5B5   C9 03      CMP #$03        If CTRL C then set flag, clear
C5B7   F0 28      BEQ $C5E1       line and exit.
C5B9   C9 18      CMP #$18        If CTRL X then print "Q" and
C5BB   F0 0B      BEQ $C5C8       restart the line.
C5BD   C9 20      CMP #$20        Ignore any other control
C5BF   90 D3      BCC $C594       characters.
C5C1   95 35      STA $35,X       Save char in buffer.
C5C3   E8         INX
C5C4   E0 4F      CPX #$4F        If input buffer is full then
C5C6   90 07      BCC $C5CF       print "Q" and start again
C5C8   A9 5C      LDA #$5C        with a new line.
C5CA   20 D9 CC   JSR $CCD9
C5CD   D0 C0      BNE $C58F
C5CF   E0 4C      CPX #$4C
C5D1   90 C1      BCC $C594
C5D3   8A         TXA             If the line is close to max
C5D4   48         PHA             number of chars then give a
C5D5   98         TYA             warning PING.
C5D6   48         PHA
C5D7   20 9F FA   JSR $FA9F       Warning PING.
C5DA   68         PLA
C5DB   A8         TAY
C5DC   68         PLA
C5DD   AA         TAX
C5DE   4C 94 C5   JMP $C594       Go back for next character.
C5E1   E6 17      INC $17         CTRL C pressed, set flag and
C5E3   A2 00      LDX #$00        finish off input buffer.
C5E5   4C EA CB   JMP $CBEA

C5E8   20 3B 02   JSR $023B       READ KEY FROM KEYBOARD.
C5EB   10 FB      BPL $C5E8       Wait until valid key is
C5ED   C9 0F      CMP #$0F        pressed (bit 7 set).
C5EF   D0 08      BNE $C5F9       If key is CTRL O then invert
C5F1   48         PHA             flag.
C5F2   A5 2E      LDA $2E
C5F4   49 FF      EOR #$FF
C5F6   85 2E      STA $2E
C5F8   68         PLA             Return with char in A.
C5F9   60         RTS

C5FA   A6 E9      LDX $E9         TOKENISE LINE.
C5FC   A0 04      LDY #$04        Set initial line counters and
C5FE   84 2A      STY $2A         flag.
C600   B5 00      LDA $00,X       Get character.
C602   C9 20      CMP #$20        If space char then put it in
C604   F0 41      BEQ $C647       line.
C606   85 25      STA $25         Save character.
C608   C9 22      CMP #$22        If character is " then handle
C60A   F0 5F      BEQ $C66B       string in quotes.
C60C   24 2A      BIT $2A         Don't tokenise if in middle of
C60E   70 37      BVS $C647       a 'DATA' statement.
C610   C9 3F      CMP #$3F        If char is '?' then substitute
C612   D0 04      BNE $C618       the 'PRINT' token.
C614   A9 BA      LDA #$BA
C616   D0 2F      BNE $C647
C618   C9 30      CMP #$30        If char is 0-9 or ; or : then
C61A   90 04      BCC $C620       put it in line and go on to
C61C   C9 3C      CMP #$3C        next char.
```

```
C61E    90 27       BCC $C647
C620    84 E0       STY $E0         Save pointer.
C622    A0 00       LDY #$00
C624    84 26       STY $26         Zero Y and reset token number.
C626    A9 E9       LDA #$E9        Set tokenising pointer to
C628    85 18       STA $18         point to byte before start of
C62A    A9 C0       LDA #$C0        keyword list.
C62C    85 19       STA $19
C62E    86 E9       STX $E9         Save pointer.
C630    CA          DEX
C631    E8          INX             Advance input pointer.
C632    E6 18       INC $18         Advance keyword list pointer.
C634    D0 02       BNE $C638
C636    E6 19       INC $19
C638    B5 00       LDA $00,X
C63A    38          SEC
C63B    F1 18       SBC ($18),Y     Test for char match and do
C63D    F0 F2       BEQ $C631       next one if chars matched.
C63F    C9 80       CMP #$80        Test for end of keyword.
C641    D0 2F       BNE $C672       Branch if not end.
C643    05 26       ORA $26         Create token.
C645    A4 E0       LDY $E0         Restore pointer.
C647    E8          INX             Move up pointers and put out
C648    C8          INY             char.
C649    99 30 00    STA $0030,Y
C64C    B9 30 00    LDA $0030,Y     If char is zero, i.e. end of
C64F    F0 39       BEQ $C68A       line then exit.
C651    38          SEC
C652    E9 3A       SBC #$3A        If ":" then clear 'DATA' flag.
C654    F0 04       BEQ $C65A
C656    C9 57       CMP #$57        If 'DATA' token then set flag.
C658    D0 02       BNE $C65C
C65A    85 2A       STA $2A
C65C    38          SEC
C65D    E9 63       SBC #$63        If not 'REM' then loop to get
C65F    D0 9F       BNE $C600       next char.
C661    85 25       STA $25         Transfer chars until same char
C663    B5 00       LDA $00,X       is found again i.e. another "
C665    F0 E0       BEQ $C647       in a string. Or until end of
C667    C5 25       CMP $25         line.
C669    F0 DC       BEQ $C647
C66B    C8          INY
C66C    99 30 00    STA $0030,Y
C66F    E8          INX
C670    D0 F1       BNE $C663
C672    A6 E9       LDX $E9         Token match has failed,
C674    E6 26       INC $26         restore X, increment token no.
C676    B1 18       LDA ($18),Y
C678    08          PHP             Save char of current char.
C679    E6 18       INC $18
C67B    D0 02       BNE $C67F       Move to next one.
C67D    E6 19       INC $19
C67F    28          PLP
C680    10 F4       BPL $C676       If not end, loop to another.
C682    B1 18       LDA ($18),Y     If more tokens left, try
C684    D0 B2       BNE $C638       another match.
C686    B5 00       LDA $00,X       No tokens left so just use
C688    10 BB       BPL $C645       char from line.
C68A    99 32 00    STA $0032,Y
C68D    A9 34       LDA #$34        Point $E9/$EA at start of line
C68F    85 E9       STA $E9         and exit.
C691    60          RTS

C692    20 E2 CA    JSR $CAE2       EDIT Get integer from text.
```

```
C695   20 B3 C6   JSR $C6B3       Look for line number in text.
C698   90 16      BCC $C6B0       Branch if failed.
C69A   6E F2 02   ROR $02F2       Set Edit flag.
C69D   20 6C C7   JSR $C76C       Print line.
C6A0   4E F2 02   LSR $02F2       Clear Edit flag.
C6A3   20 F0 CB   JSR $CBF0       New line.
C6A6   A9 0B      LDA #$0B        Send cursor up one line.
C6A8   20 D9 CC   JSR $CCD9
C6AB   68         PLA
C6AC   68         PLA
C6AD   4C B7 C4   JMP $C4B7       Immediate mode.

C6B0   4C 23 CA   JMP $CA23       Print "UNDEF'D STATEMENT .."

C6B3   A9 00      LDA #$00        LOOK FOR LINE NUMBER.
C6B5   85 1D      STA $1D         Reset line count.
C6B7   85 1E      STA $1E
C6B9   A5 9A      LDA $9A         Get begin Basic.
C6BB   A6 9B      LDX $9B
C6BD   A0 01      LDY #$01
C6BF   85 CE      STA $CE         Set up pointer into line.
C6C1   86 CF      STX $CF
C6C3   B1 CE      LDA ($CE),Y     Exit if end of program (C=0)
C6C5   F0 25      BEQ $C6EC
C6C7   C8         INY             Move pointer to line number
C6C8   C8         INY             (MSB).
C6C9   E6 1D      INC $1D         Increment line count.
C6CB   D0 02      BNE $C6CF
C6CD   E6 1E      INC $1E
C6CF   A5 34      LDA $34         Compare MSB of line number
C6D1   D1 CE      CMP ($CE),Y     with one wanted.
C6D3   90 18      BCC $C6ED       Branch if beyond prog line no.
C6D5   F0 03      BEQ $C6DA       Match in MSB made otherwise
C6D7   88         DEY             try next line.
C6D8   D0 09      BNE $C6E3
C6DA   A5 33      LDA $33         Compare LSB of line number.
C6DC   88         DEY
C6DD   D1 CE      CMP ($CE),Y
C6DF   90 0C      BCC $C6ED       Line number too big.
C6E1   F0 0A      BEQ $C6ED       Line number match made.
C6E3   88         DEY             Get line link bytes into A
C6E4   B1 CE      LDA ($CE),Y     and X. $CE/$CF left pointing
C6E6   AA         TAX             to start of line.
C6E7   88         DEY
C6E8   B1 CE      LDA ($CE),Y
C6EA   B0 D1      BCS $C6BD
C6EC   18         CLC             Line not found exit.
C6ED   60         RTS

C6EE   D0 FD      BNE $C6ED       NEW
C6F0   A9 00      LDA #$00        Set trace to off.
C6F2   4E F4 02   LSR $02F4
C6F5   A8         TAY             Set End of Basic pointer to 2
C6F6   91 9A      STA ($9A),Y     beyond Start Basic and clear
C6F8   C8         INY             the bytes inbetween - empty
C6F9   91 9A      STA ($9A),Y     program.
C6FB   A5 9A      LDA $9A
C6FD   18         CLC
C6FE   69 02      ADC #$02
C700   85 9C      STA $9C
C702   A5 9B      LDA $9B
C704   69 00      ADC #$00
C706   85 9D      STA $9D
C708   20 3A C7   JSR $C73A       Reset program pointer.
```

```
C70B    A9 00       LDA #$00
C70D    D0 2A       BNE $C739       CLEAR
C70F    A5 A6       LDA $A6         Set last string allocated to
C711    A4 A7       LDY $A7         the current value in Himem.
C713    85 A2       STA $A2
C715    84 A3       STY $A3
C717    A5 9C       LDA $9C         Set End Variables pointer and
C719    A4 9D       LDY $9D         End Arrays pointer to value
C71B    85 9E       STA $9E         held in End Basic Pointer.
C71D    84 9F       STY $9F         This deletes all variables and
C71F    85 A0       STA $A0         arrays.
C721    84 A1       STY $A1
C723    20 52 C9    JSR $C952       Reset 'DATA' pointer.
C726    A2 88       LDX #$88
C728    86 85       STX $85
C72A    68          PLA             Place calling routine on top
C72B    A8          TAY             of the stack.
C72C    68          PLA
C72D    A2 FE       LDX #$FE
C72F    9A          TXS
C730    48          PHA
C731    98          TYA
C732    48          PHA
C733    A9 00       LDA #$00        Reset high byte of End of
C735    85 AD       STA $AD         Executed Program pointer.
C737    85 2B       STA $2B
C739    60          RTS

C73A    18          CLC             This routine sets the current
C73B    A5 9A       LDA $9A         program position pointer to
C73D    69 FF       ADC #$FF        the byte before the Start of
C73F    85 E9       STA $E9         Basic.
C741    A5 9B       LDA $9B
C743    69 FF       ADC #$FF
C745    85 EA       STA $EA
C747    60          RTS

C748    08          PHP             LIST
C749    20 E2 CA    JSR $CAE2       Get line number and look for
C74C    20 B3 C6    JSR $C6B3       it in program.
C74F    28          PLP
C750    F0 14       BEQ $C766       If no number, list whole prog
C752    20 E8 00    JSR $00E8       If only one line number then
C755    F0 15       BEQ $C76C       just list it.
C757    C9 CD       CMP #$CD        If no "-" then exit.
C759    D0 92       BNE $C6ED
C75B    20 E2 00    JSR $00E2       Get next non space char.
C75E    F0 06       BEQ $C766
C760    20 E2 CA    JSR $CAE2       Get line number.
C763    F0 07       BEQ $C76C       If end, list between the two
C765    60          RTS             numbers else exit.

C766    A9 FF       LDA #$FF        Set upper line limit to #FFFF
C768    85 33       STA $33
C76A    85 34       STA $34
C76C    A0 01       LDY #$01
C76E    B1 CE       LDA ($CE),Y     Exit if end of program.
C770    F0 4D       BEQ $C7BF
C772    20 62 C9    JSR $C962       Test for CTRL C.
C775    C9 20       CMP #$20        If not space then skip next
C777    D0 0E       BNE $C787       section.
C779    4E DF 02    LSR $02DF       Clear key pressed flag.
C77C    AD DF 02    LDA $02DF       Wait until key is pressed.
C77F    10 FB       BPL $C77C
```

```
C781   20 62 C9    JSR $C962       Test for CTRL C.
C784   4E DF 02    LSR $02DF       C1ear key pressed flag.
C787   C8          INY
C788   B1 CE       LDA ($CE),Y     Get line number into X (LSB)
C78A   AA          TAX             and A (MSB).
C78B   C8          INY
C78C   B1 CE       LDA ($CE),Y
C78E   C5 34       CMP $34         If line number is at limit
C790   D0 04       BNE $C796       then skip test.
C792   E4 33       CPX $33
C794   F0 02       BEQ $C798
C796   B0 27       BCS $C7BF       Exit if over line no. limit.
C798   84 B8       STY $B8         save A and Y.
C79A   48          PHA
C79B   20 F0 CB    JSR $CBF0       Newline.
C79E   68          PLA
C79F   20 C5 E0    JSR $E0C5       Print line number.
C7A2   A9 20       LDA #$20        Get space.
C7A4   A4 B8       LDY $B8
C7A6   29 7F       AND #$7F
C7A8   20 D9 CC    JSR $CCD9       Print character.
C7AB   C8          INY
C7AC   F0 11       BEQ $C7BF       Exit if line too long.
C7AE   B1 CE       LDA ($CE),Y
C7B0   D0 1E       BNE $C7D0       If not end of line print char
C7B2   A8          TAY             or token.
C7B3   B1 CE       LDA ($CE),Y
C7B5   AA          TAX
C7B6   C8          INY
C7B7   B1 CE       LDA ($CE),Y
C7B9   86 CE       STX $CE         Point to next line.
C7BB   85 CF       STA $CF
C7BD   D0 AD       BNE $C76C       Go round and list next line.
C7BF   2C F2 02    BIT $02F2       If 'test return1 flag is set
C7C2   10 01       BPL $C7C5       then exit.
C7C4   60          RTS

C7C5   20 F0 CB    JSR $CBF0       Newline.
C7C8   20 2F C8    JSR $C82F       Reset output to screen.
C7CB   68          PLA             Remove address of calling
C7CC   68          PLA             routine and restart Basic.
C7CD   4C A8 C4    JMP $C4A8

C7D0   10 D6       BPL $C7A8       Print char if not a token.
C7D2   38          SEC
C7D3   E9 7F       SBC #$7F        Get token count into X.
C7D5   AA          TAX
C7D6   84 B8       STY $B8         Save Y.
C7D8   A0 00       LDY #$00        Clear Y.
C7DA   A9 E9       LDA #$E9        Set pointer to point to start
C7DC   85 18       STA $18         1 byte before keyword list.
C7DE   A9 C0       LDA #$C0
C7E0   85 19       STA $19         Increment the pointer at $18/
C7E2   CA          DEX             $19 until correct keyword is
C7E3   F0 0D       BEQ $C7F2       found.
C7E5   E6 18       INC $18
C7E7   D0 02       BNE $C7EB
C7E9   E6 19       INC $19
C7EB   B1 18       LDA ($18),Y
C7ED   10 F6       BPL $C7E5
C7EF   4C E2 C7    JMP $C7E2
C7F2   C8          INY             Get next char.
C7F3   B1 18       LDA ($18),Y
C7F5   30 AD       BMI $C7A4       Another token.
```

```
C7F7   20 D9 CC   JSR $CCD9       Print char.
C7FA   4C F2 C7   JMP $C7F2       Go round again.

C7FD   20 16 C8   JSR $C816       LLIST
C800   4E F2 02   LSR $02F2       Set output to printer, clear
C803   20 E8 00   JSR $00E8       "list return" flag and perform
C806   4C 48 C7   JMP $C748       list.

C809   20 16 C8   JSR $C816       LPRINT
C80C   20 E8 00   JSR $00E8       Set output to printer, perform
C80F   20 AB CB   JSR $CBAB       PRINT and set output back to
C812   20 2F C8   JSR $C82F       screen.
C815   60         RTS

C816   2C F1 02   BIT $02F1       SET OUTPUT TO PRINTER.
C819   30 39      BMI $C854       Exit if printer is on.
C81B   A5 30      LDA $30         Save Basic Screen cursor
C81D   8D 59 02   STA $0259       position.
C820   AD 58 02   LDA $0258       Transfer Printer cursor
C823   85 30      STA $30         position.
C825   38         SEC
C826   6E F1 02   ROR $02F1       Set printer to on.
C829   AD 56 02   LDA $0256       Get printer line width and set
C82C   4C 44 C8   JMP $C844       up linewidth.

C82F   2C F1 02   BIT $02F1       SET OUTPUT TO SCREEN.
C832   10 20      BPL $C854
C834   A5 30      LDA $30         Save Basic printer cursor
C836   8D 58 02   STA $0258       position.
C839   AD 59 02   LDA $0259       Transfer Basic screen
C83C   85 30      STA $30         position.
C83E   4E F1 02   LSR $02F1       Clear printer flag.
C841   AD 57 02   LDA $0257       Transfer screen width to $31
C844   85 31      STA $31         and set content of $32 to the
C846   38         SEC             multiple of 8 that is less
C847   E9 08      SBC #$08        than or equal to content of
C849   B0 FB      BCS $C846       $31.
C84B   49 FF      EOR #$FF
C84D   E9 06      SBC #$06
C84F   18         CLC
C850   65 31      ADC $31
C852   85 32      STA $32
C854   60         RTS

C855   A9 80      LDA #$80        FOR Set 'no integer
C857   85 2B      STA $2B         variables' flag.
C859   20 1C CB   JSR $CB1C       Call 'LET' to assign loop var.
C85C   20 C6 C3   JSR $C3C6       Test & branch if that loop
C85F   D0 05      BNE $C866       doesn't already exist.
C861   8A         TXA             Write over the old loop which
C862   69 0F      ADC #$0F        has the same variable name -
C864   AA         TAX             old loop is lost.
C865   9A         TXS
C866   68         PLA
C867   68         PLA
C868   A9 09      LDA #$09        Check for 18 free bytes of
C86A   20 37 C4   JSR $C437       space on the stack.
C86D   20 4E CA   JSR $CA4E       Find end of statement.
C870   18         CLC
C871   98         TYA             Save end of statement
C872   65 E9      ADC $E9         address on the stack, low
C874   48         PHA             byte first.
C875   A5 EA      LDA $EA
C877   69 00      ADC #$00
```

```
C879   48          PHA
C87A   A5 A9       LDA $A9          Save current line number on
C87C   48          PHA              stack.
C87D   A5 A8       LDA $A8
C87F   48          PHA
C880   A9 C3       LDA #$C3         Search for a 'TO' token, give
C882   20 67 D0    JSR $D067        error if not found.
C885   20 06 CF    JSR $CF06        Check numeric type.
C888   20 03 CF    JSR $CF03        Evaluate expression.
C88B   A5 D5       LDA $D5
C88D   09 7F       ORA #$7F
C88F   25 D1       AND $D1
C891   85 D1       STA $D1
C893   A9 9E       LDA #$9E         Round off the value in the
C895   A0 C8       LDY #$C8         main F1oating Point
C897   85 91       STA $91          Accumulator and then push it
C899   84 92       STY $92          on to the stack.
C89B   4C C0 CF    JMP $CFC0

C89E   A9 81       LDA #$81         Unpack the floating point
C8A0   A0 DC       LDY #$DC         number at $DC81 which is
C8A2   20 7B DE    JSR $DE7B        default STEP size (1).
C8A5   20 E8 00    JSR $00E8        Get next text character.
C8A8   C9 CB       CMP #$CB         Test and branch if next char
C8AA   D0 06       BNE $C8B2        is not a 'STEP' token.
C8AC   20 E2 00    JSR $00E2        Get next text character.
C8AF   20 03 CF    JSR $CF03        Evaluate expression.
C8B2   20 13 DF    JSR $DF13        Get sign of STEP into A.
C8B5   20 B1 CF    JSR $CFB1        Put FPA on stack etc.
C8B8   A5 B9       LDA $B9          Put variable address and FOR
C8BA   48          PHA              token on the stack. Structure
C8BB   A5 B8       LDA $B8          on stack for this loop is now
C8BD   48          PHA              complete.
C8BE   A9 8D       LDA #$8D
C8C0   48          PHA

C8C1   20 62 C9    JSR $C962        EXECUTE NEXT LINE.
C8C4   A5 E9       LDA $E9          Test for CTRL C
C8C6   A4 EA       LDY $EA
C8C8   F0 06       BEQ $C8D0        Immediate mode.
C8CA   85 AC       STA $AC          Save current position in
C8CC   84 AD       STY $AD          program.
C8CE   A0 00       LDY #$00         Branch if next char in program
C8D0   B1 E9       LDA ($E9),Y      is not a null (end of line).
C8D2   D0 5B       BNE $C92F
C8D4   4E 52 02    LSR $0252        Clear pending E1se flag.
C8D7   A0 02       LDY #$02         Test if the address of next
C8D9   B1 E9       LDA ($E9),Y      line is not a null. If it is,
C8DB   18          CLC              then end program.
C8DC   D0 03       BNE $C8E1
C8DE   4C 8A C9    JMP $C98A
C8E1   C8          INY
C8E2   B1 E9       LDA ($E9),Y      Load the next line number to
C8E4   85 A8       STA $A8          be executed. This is now the
C8E6   C8          INY              current line number.
C8E7   B1 E9       LDA ($E9),Y
C8E9   85 A9       STA $A9
C8EB   98          TYA
C8EC   65 E9       ADC $E9          Update program position ptr.
C8EE   85 E9       STA $E9
C8F0   90 02       BCC $C8F4
C8F2   E6 EA       INC $EA
C8F4   2C F4 02    BIT $02F4
C8F7   10 13       BPL $C90C        TRACE is off.
```

```
C8F9   48          PHA
C8FA   A9 5B       LDA #$5B
C8FC   20 FB CC    JSR $CCFB       Print '['to screen.
C8FF   A5 A9       LDA $A9
C901   A6 A8       LDX $A8         Print the current line number
C903   20 C5 E0    JSR $E0C5       on the screen.
C906   A9 5D       LDA #$5D
C908   20 FB CC    JSR $CCFB       Print ']' to screen.
C90B   68          PLA
C90C   20 E2 00    JSR $00E2       Step through spaces in program.
C90F   20 15 C9    JSR $C915       Execute statement.
C912   4C C1 C8    JMP $C8C1

C915   F0 49       BEQ $C960       EXECUTE STATEMENT. Exit if end
C917   E9 80       SBC #$80        of line. Branch if not a token
C919   90 11       BCC $C92C       - try an assignment.
C91B   C9 42       CMP #$42        If it is not a statement token
C91D   B0 30       BCS $C94F       then "SYNTAX ERROR"
C91F   0A          ASL A           Get start address of token
C920   A8          TAY             routine and put it on stack.
C921   B9 07 C0    LDA $C007,Y
C924   48          PHA
C925   B9 06 C0    LDA $C006,Y
C928   48          PHA
C929   4C E2 00    JMP $00E2       Clear spaces & enter routine.
C92C   4C 1C CB    JMP $CB1C       Jump to 'LET' routine.
C92F   C9 3A       CMP #$3A        If ":" then do next statement.
C931   F0 C1       BEQ $C8F4
C933   C9 C8       CMP #$C8        If not 'ELSE' token then check
C935   D0 0E       BNE $C945       for ""'
C937   2C 52 02    BIT $0252       If no 'ELSE' pending then give
C93A   10 13       BPL $C94F       "SYNTAX ERROR".
C93C   20 B1 CA    JSR $CAB1       Set text ptr to end of
C93F   4E 52 02    LSR $0252       statement & clear ELSE pending
C942   4C C1 C8    JMP $C8C1       flag. Jump to next line.
C945   C9 27       CMP #$27        Error if character is not a
C947   D0 06       BNE $C94F       "'.
C949   20 99 CA    JSR $CA99       Skip rest of line.
C94C   4C C1 C8    JMP $C8C1       Go back to next line.

C94F   4C 70 D0    JMP $D070       Print "SYNTAX ERROR"

C952   38          SEC             RESTORE
C953   A5 9A       LDA $9A         This routine sets the 'DATA'
C955   E9 01       SBC #$01        pointer to the address 1 byte
C957   A4 9B       LDY $9B         below the Start of Basic.
C959   B0 01       BCS $C95C
C95B   88          DEY
C95C   85 B0       STA $B0
C95E   84 B1       STY $B1
C960   60          RTS
C961   60          RTS

C962   AD DF 02    LDA $02DF       Load next char from keyboard
C965   10 F9       BPL $C960       and test for CTRL C.
C967   29 7F       AND #$7F
C969   A2 08       LDX #$08
C96B   C9 03       CMP #$03
C96D   D0 F2       BNE $C961       Exit if not CTRL C.
C96F   C9 03       CMP #$03        Set C to act like 'STOP'.
C971   B0 01       BCS $C974       STOP
C973   18          CLC             END
C974   D0 43       BNE $C9B9
C976   A5 E9       LDA $E9
```

```
C978   A4 EA       LDY $EA
C97A   F0 0C       BEQ $C988       In immediate mode.
C97C   85 AC       STA $AC         Save current position in the
C97E   84 AD       STY $AD         program.
C980   A5 A8       LDA $A8         Save the current line number.
C982   A4 A9       LDY $A9
C984   85 AA       STA $AA
C986   84 AB       STY $AB
C988   68          PLA             Remove address of calling
C989   68          PLA             routine.
C98A   A9 BD       LDA #$BD        Set up parameters for jumping
C98C   A0 C3       LDY #$C3        back into command mode.
C98E   A2 00       LDX #$00
C990   8E F1 02    STX $02F1       Clear printer flag.
C993   8E DF 02    STX $02DF       Clear input char from keyboard
C996   86 2E       STX $2E         Clear CTRL O flag.
C998   90 03       BCC $C99D       C=1 if "BREAK AT ..." is to be
C99A   4C 9D C4    JMP $C49D       printed before going back to
C99D   4C A8 C4    JMP $C4A8       command mode.
C9A0   D0 17       BNE $C9B9       CONT
C9A2   A2 D7       LDX #$D7
C9A4   A4 AD       LDY $AD         Load the saved current program
C9A6   D0 03       BNE $C9AB       position. Print "CAN'T CONT.."
C9A8   4C 7E C4    JMP $C47E       error if in immediate mode.
C9AB   A5 AC       LDA $AC         Put saved program position
C9AD   85 E9       STA $E9         pointer into current position
C9AF   84 EA       STY $EA         pointer. Do the same for the
C9B1   A5 AA       LDA $AA         line numbers.
C9B3   A4 AB       LDY $AB
C9B5   85 A8       STA $A8
C9B7   84 A9       STY $A9
C9B9   60          RTS             Continue with program.

C9BA   4C 36 D3    JMP $D336       Print "ILLEGAL QUANTITY ERROR"
C9BD   D0 03       BNE $C9C2       RUN If not end of statement
C9BF   4C 08 C7    JMP $C708       then execute from start.

C9C2   20 0F C7    JSR $C70F       Perform 'CLEAR' and then
C9C5   4C DC C9    JMP $C9DC       go to line number.

C9C8   A9 03       LDA #$03        GOSUB
C9CA   20 37 C4    JSR $C437       Test free space left on stack.
C9CD   A5 EA       LDA $EA         Put current position pointer
C9CF   48          PHA             on the stack.
C9D0   A5 E9       LDA $E9
C9D2   48          PHA
C9D3   A5 A9       LDA $A9         Put current line number on the
C9D5   48          PHA             stack.
C9D6   A5 A8       LDA $A8
C9D8   48          PHA
C9D9   A9 9B       LDA #$9B        Put GOSUB token on stack.
C9DB   48          PHA
C9DC   20 E8 00    JSR $00E8       Step through spaces in prog'm.
C9DF   20 E5 C9    JSR $C9E5       Perform 'GOTO'.
C9E2   4C C1 C8    JMP $C8C1       Execute next statement/line.

C9E5   20 53 E8    JSR $E853       GOTO Get +ve integer in $33/
C9E8   20 51 CA    JSR $CA51       $34 & find offset of line end.
C9EB   A5 A9       LDA $A9         If going to a previous line in
C9ED   C5 34       CMP $34         program then search from start
C9EF   B0 0B       BCS $C9FC       of program.
C9F1   98          TYA
C9F2   38          SEC
C9F3   65 E9       ADC $E9         Set A (LSB) and X to point to
```

```
C9F5   A6 EA       LDX $EA          next line.
C9F7   90 07       BCC $CA00
C9F9   E8          INX
C9FA   B0 04       BCS $CA00
C9FC   A5 9A       LDA $9A          Set A (LSB) and X to start of
C9FE   A6 9B       LDX $9B          Basic program.
CA00   20 BD C6    JSR $C6BD        Search for a line.
CA03   90 1E       BCC $CA23        Print error if not found.
CA05   A5 CE       LDA $CE          Set program position to 1
CA07   E9 01       SBC #$01         byte before start of that
CA09   85 E9       STA $E9          line.
CA0B   A5 CF       LDA $CF
CA0D   E9 00       SBC #$00
CA0F   85 EA       STA $EA
CA11   60          RTS              Exit.
CA12   D0 FD       BNE $CA11        POP & RETURN

CA14   A9 FF       LDA #$FF
CA16   85 B9       STA $B9          Set stack to position where
CA18   20 C6 C3    JSR $C3C6        the GOSUB token is expected.
CA1B   9A          TXS
CA1C   C9 9B       CMP #$9B
CA1E   F0 0B       BEQ $CA2B        Branch if GOSUB token found.
CA20   A2 16       LDX #$16         Print "RETURN WITHOUT GOSUB.."
CA22   2C A2 5A    BIT $5AA2        Hides a print "UNDEF'D STAT.."
CA25   4C 7E C4    JMP $C47E        Goto print error message.

CA28   4C 70 D0    JMP $D070        Print "SYNTAX ERROR"

CA2B   68          PLA
CA2C   68          PLA
CA2D   C0 0C       CPY #$0C
CA2F   F0 19       BEQ $CA4A        Token is that of 'POP'.
CA31   85 A8       STA $A8          Returning from GOSUB so
CA33   68          PLA              restore old line number and
CA34   85 A9       STA $A9          program position counter.
CA36   68          PLA
CA37   85 E9       STA $E9
CA39   68          PLA
CA3A   85 EA       STA $EA
CA3C   20 4E CA    JSR $CA4E        DATA Find end of line.
CA3F   98          TYA
CA40   18          CLC              Adjust program position to
CA41   65 E9       ADC $E9          end of the line.
CA43   85 E9       STA $E9
CA45   90 02       BCC $CA49
CA47   E6 EA       INC $EA
CA49   60          RTS

CA4A   68          PLA              Correct stack pointer for POP
CA4B   68          PLA              command.
CA4C   68          PLA
CA4D   60          RTS

CA4E   A2 3A       LDX #$3A         FIND END OF STATEMENT
CA50   2C A2 00    BIT $00A2        FIND END OF LINE
CA53   86 24       STX $24
CA55   A0 00       LDY #$00
CA57   84 25       STY $25          Swap match characters - colon
CA59   A5 25       LDA $25          for end of statement, null for
CA5B   A6 24       LDX $24          end of line.
CA5D   85 24       STA $24
CA5F   86 25       STX $25
CA61   B1 E9       LDA ($E9),Y
```

```
CA63    F0 E4       BEQ $CA49       Exit if end of line.
CA65    C5 25       CMP $25
CA67    F0 E0       BEQ $CA49       Exit if match made.
CA69    C8          INY
CA6A    C9 22       CMP #$22        If " then swap match chars.
CA6C    D0 F3       BNE $CA61       Loop again
CA6E    F0 E9       BEQ $CA59

CA70    20 17 CF    JSR $CF17       IF Evaluate expression.
CA73    20 E8 00    JSR $00E8       Clear spaces in text.
CA76    C9 97       CMP #$97
CA78    F0 05       BEQ $CA7F       Token is that of 'GOTO'.
CA7A    A9 C9       LDA #$C9        Search for 'THEN' token.
CA7C    20 67 D0    JSR $D067
CA7F    A5 D0       LDA $D0
CA81    D0 05       BNE $CA88       Condition is true.
CA83    20 9E CA    JSR $CA9E       Condition is false.
CA86    F0 B7       BEQ $CA3F
CA88    20 E8 00    JSR $00E8       Get next text character.
CA8B    B0 03       BCS $CA90
CA8D    4C E5 C9    JMP $C9E5       Jump to 'GOTO'
CA90    08          PHP
CA91    38          SEC
CA92    6E 52 02    ROR $0252       Set Else pending flag.
CA95    28          PLP
CA96    4C 15 C9    JMP $C915       Execute statement.

CA99    20 51 CA    JSR $CA51       REM Find end of line.
CA9C    F0 A1       BEQ $CA3F       Branch always.
CA9E    A0 00       LDY #$00
CAA0    B1 E9       LDA ($E9),Y     If at end of line no 'THEN's
CAA2    F0 0C       BEQ $CAB0       or 'ELSE's to deal with.
CAA4    C8          INY
CAA5    C9 C9       CMP #$C9        Test for 'THEN' token.
CAA7    F0 F0       BEQ $CA99       'THEN' token found.
CAA9    C9 C8       CMP #$C8        Test for 'ELSE' token.
CAAB    D0 F3       BNE $CAA0       'ELSE' token not found.
CAAD    4C 3F CA    JMP $CA3F       Set program pos'n to line end.
CAB0    60          RTS             Exit.

CAB1    A0 FF       LDY #$FF        Set program position pointer
CAB3    C8          INY             to end of line.
CAB4    B1 E9       LDA ($E9),Y
CAB6    F0 04       BEQ $CABC       Step through program until a
CAB8    C9 3A       CMP #$3A        null or colon is found. Then
CABA    D0 F7       BNE $CAB3       jump to update program
CABC    4C 3F CA    JMP $CA3F       position pointer.

CABF    4C 70 D0    JMP $D070       Print "SYNTAX ERROR".

CAC2    20 C8 D8    JSR $D8C8       ON Get single byte expr'n
CAC5    48          PHA             which returns in X and $D4.
CAC6    C9 9B       CMP #$9B
CAC8    F0 04       BEQ $CACE       Found a 'GOSUB' token.
CACA    C9 97       CMP #$97        Error if character is not a
CACC    D0 F1       BNE $CABF       'GOTO' token.
CACE    C6 D4       DEC $D4         Step through arguments until
CAD0    D0 04       BNE $CAD6       correct line number is found.
CAD2    68          PLA
CAD3    4C 17 C9    JMP $C917       Execute statement.

CAD6    20 E2 00    JSR $00E2       Step through spaces in text.
CAD9    20 E2 CA    JSR $CAE2       Get 2 byte integer from text.
CADC    C9 2C       CMP #$2C
```

```
CADE   F0 EE       BEQ $CACE       Character is a comma.
CAE0   68          PLA             Exit if char was not a comma.
CAE1   60          RTS

CAE2   A2 00       LDX #$00        GET 2 BYTE INTEGER FROM TEXT.
CAE4   86 33       STX $33         Zero result.
CAE6   86 34       STX $34
CAE8   B0 F7       BCS $CAE1       Exit if no more digits.
CAEA   E9 2F       SBC #$2F        Put value of digit into $24.
CAEC   85 24       STA $24
CAEE   A5 34       LDA $34         Transfer MSB to temporary
CAF0   85 91       STA $91         work byte.
CAF2   C9 19       CMP #$19        Syntax error if MSB is over
CAF4   B0 D4       BCS $CACA       25 - result will be too big.
CAF6   A5 33       LDA $33         Multiply original number by
CAF8   0A          ASL A           10, firstly adding itself to
CAF9   26 91       ROL $91         4 times itself to give 5 times
CAFB   0A          ASL A           itself. Then double result.
CAFC   26 91       ROL $91
CAFE   65 33       ADC $33
CB00   85 33       STA $33
CB02   A5 91       LDA $91
CB04   65 34       ADC $34
CB06   85 34       STA $34
CB08   06 33       ASL $33
CB0A   26 34       ROL $34
CB0C   A5 33       LDA $33
CB0E   65 24       ADC $24         Add in next digit.
CB10   85 33       STA $33
CB12   90 02       BCC $CB16
CB14   E6 34       INC $34         Overflow from LSB into MSB.
CB16   20 E2 00    JSR $00E2       Get next non space character.
CB19   4C E8 CA    JMP $CAE8       Jump to do next number.

CB1C   20 88 D1    JSR $D188       LET Get variable.
CB1F   85 B8       STA $B8         Save location.
CB21   84 B9       STY $B9
CB23   A9 D4       LDA #$D4        Give error if "=" is not next
CB25   20 67 D0    JSR $D067       character.
CB28   A5 29       LDA $29         Save integer variable flag.
CB2A   48          PHA
CB2B   A5 28       LDA $28         Save string variable flag.
CB2D   48          PHA
CB2E   20 17 CF    JSR $CF17       Evaluate expression.
CB31   68          PLA
CB32   2A          ROL A           Check type matches.
CB33   20 09 CF    JSR $CF09
CB36   D0 18       BNE $CB50       Do string assignment.
CB38   68          PLA
CB39   10 12       BPL $CB4D       If real do floating pt number.
CB3B   20 F4 DE    JSR $DEF4       Round off main FPA and convert
CB3E   20 A9 D2    JSR $D2A9       to 2 byte signed integer.
CB41   A0 00       LDY #$00
CB43   A5 D3       LDA $D3         Store value into integer
CB45   91 B8       STA ($B8),Y     variable.
CB47   C8          INY
CB48   A5 D4       LDA $D4
CB4A   91 B8       STA ($B8),Y
CB4C   60          RTS
CB4D   4C A9 DE    JMP $DEA9       Pack main FPA.

CB50   68          PLA             String assignment.
CB51   A0 02       LDY #$02        If pointer to strings is
CB53   B1 D3       LDA ($D3),Y     beyond start of string block
```

```
CB55    C5 A3       CMP $A3         then branch to use present
CB57    90 17       BCC $CB70       block of data about string.
CB59    D0 07       BNE $CB62
CB5B    88          DEY
CB5C    B1 D3       LDA ($D3),Y
CB5E    C5 A2       CMP $A2
CB60    90 0E       BCC $CB70
CB62    A4 D4       LDY $D4
CB64    C4 9D       CPY $9D         If string data block is
CB66    90 08       BCC $CB70       beyond end of Basic then it is
CB68    D0 0D       BNE $CB77       in variable block. Copy it
CB6A    A5 D3       LDA $D3         down and set up new string
CB6C    C5 9C       CMP $9C         block.
CB6E    B0 07       BCS $CB77
CB70    A5 D3       LDA $D3
CB72    A4 D4       LDY $D4
CB74    4C 8D CB    JMP $CB8D
CB77    A0 00       LDY #$00
CB79    B1 D3       LDA ($D3),Y     Get string length and set up
CB7B    20 A3 D5    JSR $D5A3       new string & data block.
CB7E    A5 BF       LDA $BF         Copy pointers to present
CB80    A4 C0       LDY $C0         string into $DE/$DF.
CB82    85 DE       STA $DE
CB84    84 DF       STY $DF
CB86    20 A4 D7    JSR $D7A4       Transfer string into position.
CB89    A9 D0       LDA #$D0        Use data block at $00,$01,$02.
CB8B    A0 00       LDY #$00
CB8D    85 BF       STA $BF
CB8F    84 C0       STY $C0
CB91    20 05 D8    JSR $D805       Release from string stack if
CB94    A0 00       LDY #$00        temporary.
CB96    B1 BF       LDA ($BF),Y     Copy data block into variable
CB98    91 B8       STA ($B8),Y     area so that it is now a
CB9A    C8          INY             string pointer.
CB9B    B1 BF       LDA ($BF),Y
CB9D    91 B8       STA ($B8),Y
CB9F    C8          INY
CBA0    B1 BF       LDA ($BF),Y
CBA2    91 B8       STA ($B8),Y
CBA4    60          RTS             Exit.

CBA5    20 B3 CC    JSR $CCB3       Set up string data in main FPA
CBA8    20 E8 00    JSR $00E8       and print string out.

CBAB    F0 43       BEQ $CBF0       PRINT New/line if no data.
CBAD    F0 5C       BEQ $CC0B       Exit if no more data.
CBAF    C9 C2       CMP #$C2
CBB1    F0 7B       BEQ $CC2E       'TAB(' token found.
CBB3    C9 C5       CMP #$C5
CBB5    18          CLC
CBB6    F0 76       BEQ $CC2E       'SPC(' token found.
CBB8    C9 2C       CMP #$2C
CBBA    F0 50       BEQ $CC0C       Comma found.
CBBC    C9 3B       CMP #$3B
CBBE    F0 6B       BEQ $CC2B       Semi-colon found.
CBC0    C9 C6       CMP #$C6
CBC2    D0 03       BNE $CBC7       Character is not an '@'
CBC4    4C 59 CC    JMP $CC59       Set cursor for '@' command.

CBC7    20 17 CF    JSR $CF17       Evaluate expression.
CBCA    24 28       BIT $28
CBCC    30 D7       BMI $CBA5       String flag is set.
CBCE    20 D5 E0    JSR $E0D5       Convert number to string.
CBD1    20 B5 D5    JSR $D5B5       Get string after first ".
```

```
CBD4   A0 00       LDY #$00
CBD6   B1 D3       LDA ($D3),Y
CBD8   18          CLC
CBD9   65 30       ADC $30
CBDB   C5 31       CMP $31          Branch if there will not be
CBDD   90 03       BCC $CBE2        overflow on to next line.
CBDF   20 F0 CB    JSR $CBF0        Newline.
CBE2   20 B3 CC    JSR $CCB3        Print the string.
CBE5   20 D4 CC    JSR $CCD4        Print a space.
CBE8   D0 BE       BNE $CBA8        Branch back for more.
CBEA   A0 00       LDY #$00         Finish off input buffer by
CBEC   94 35       STY $35,X        writing zero to last position.
CBEE   A2 34       LDX #$34

CBF0   A5 30       LDA $30          NEWLINE.
CBF2   48          PHA              Save cursor position.
CBF3   A9 0D       LDA #$0D         Print carnage
CBF5   20 D9 CC    JSR $CCD9
CBF8   68          PLA              Restore cursor position.
CBF9   2C F1 02    BIT $02F1
CBFC   30 04       BMI $CC02        Printer is on.
CBFE   C5 31       CMP $31          Test and branch if cursor is
CC00   F0 09       BEQ $CC0B        at maximum line width.
CC02   A9 00       LDA #$00         Zero the cursor.
CC04   85 30       STA $30
CC06   A9 0A       LDA #$0A         Print a line feed char.
CC08   20 D9 CC    JSR $CCD9
CC0B   60          RTS

CC0C   A5 30       LDA $30
CC0E   2C F1 02    BIT $02F1
CC11   30 04       BMI $CC17        Printer is enabled.
CC13   38          SEC
CC14   ED 53 02    SBC $0253        Subtract content of $253 off
CC17   38          SEC              cursor position and then find
CC18   E9 08       SBC #$08         difference between that and
CC1A   B0 FC       BCS $CC18        the next multiple of 8.
CC1C   49 FF       EOR #$FF
CC1E   69 01       ADC #$01
CC20   AA          TAX
CC21   18          CLC
CC22   65 30       ADC $30          If next multiple of 8 in
CC24   C5 31       CMP $31          cursor position is not off
CC26   90 1F       BCC $CC47        the screen then branch.
CC28   20 F0 CB    JSR $CBF0        Newline.
CC2B   4C 4B CC    JMP $CC4B        Go back for more.

CC2E   08          PHP              Deal with 'TAB(' and 'SPC('.
CC2F   20 C5 D8    JSR $D8C5        Get single byte expression.
CC32   C9 29       CMP #$29
CC34   D0 20       BNE $CC56        ')' not found.
CC36   28          PLP
CC37   90 0E       BCC $CC47        'SPC(' token.
CC39   8A          TXA              'TAB(' token.
CC3A   C5 31       CMP $31
CC3C   90 03       BCC $CC41        If TAB will go off screen then
CC3E   4C 36 D3    JMP $D336        print "ILLEGAL QUANTITY ERROR"
CC41   38          SEC
CC42   E5 30       SBC $30          Branch if TAB column is before
CC44   90 05       BCC $CC4B        current cursor column.
CC46   AA          TAX
CC47   E8          INX              Print spaces to get cursor in
CC48   CA          DEX              correct column for next chars
CC49   D0 06       BNE $CC51        to be printed.
```

```
CC4B   20 E2 00   JSR $00E2       Clear spaces in text.
CC4E   4C AD CB   JMP $CBAD       Jump back to print more.
CC51   20 D4 CC   JSR $CCD4       Print space.
CC54   D0 F2      BNE $CC48       Jump back for more.
CC56   4C 70 D0   JMP $D070       Print "SYNTAX ERROR"

CC59   2C F1 02   BIT $02F1       SET CURSOR FOR '@'.
CC5C   30 F8      BMI $CC56       Printer is on.
CC5E   AE 1F 02   LDX $021F
CC61   F0 03      BEQ $CC66       In text mode.
CC63   4C F7 EA   JMP $EAF7       Print "DISP TYPE MISMATCH E.."

CC66   20 C5 D8   JSR $D8C5       Get single byte expression.
CC69   E0 28      CPX #$28        Print "ILLEGAL QUANTITY ERROR"
CC6B   B0 40      BCS $CCAD       if going off screen.
CC6D   86 0C      STX $0C
CC6F   20 65 D0   JSR $D065       Test for comma.
CC72   20 C8 D8   JSR $D8C8       Get single byte expression.
CC75   E8         INX
CC76   E0 1C      CPX #$1C        Give error if cursor will be
CC78   B0 33      BCS $CCAD       off bottom of screen.
CC7A   AD 6A 02   LDA $026A
CC7D   48         PHA             Temporarily disable cursor.
CC7E   29 FE      AND #$FE
CC80   8D 6A 02   STA $026A
CC83   A9 00      LDA #$00        Turn cursor off.
CC85   20 01 F8   JSR $F801
CC88   A5 0C      LDA $0C         Put new cursor column and rows
CC8A   8D 69 02   STA $0269       into the locations used by the
CC8D   8A         TXA             operating system.
CC8E   8D 68 02   STA $0268
CC91   20 0C DA   JSR $DA0C       Calculate screen row address.
CC94   A5 1F      LDA $1F         Put start of current row
CC96   A4 20      LDY $20         address into correct pointer.
CC98   85 12      STA $12
CC9A   84 13      STY $13
CC9C   68         PLA             Restore cursor flag.
CC9D   8D 6A 02   STA $026A
CCA0   A9 01      LDA #$01        Turn cursor back on.
CCA2   20 01 F8   JSR $F801
CCA5   A9 3B      LDA #$3B
CCA7   20 67 D0   JSR $D067       Test for a ";" in text.
CCAA   4C AD CB   JMP $CBAD       Jump back for more.

CCAD   4C C2 D8   JMP $D8C2       Print "ILLEGAL QUANTITY...".

CCB0   20 B5 D5   JSR $D5B5       PRINT OUT STRING AFTER ".
CCB3   20 D0 D7   JSR $D7D0       Get string after " and set up
CCB6   AA         TAX             string in main FPA.
CCB7   A0 00      LDY #$00
CCB9   E8         INX             Print out the number of chars
CCBA   CA         DEX             held in X using the pointer at
CCBB   F0 10      BEQ $CCCD       $91/$92 to load in string from
CCBD   B1 91      LDA ($91),Y     memory.
CCBF   20 D9 CC   JSR $CCD9
CCC2   C8         INY
CCC3   C9 0D      CMP #$0D
CCC5   D0 F3      BNE $CCBA
CCC7   20 0B CC   JSR $CC0B
CCCA   4C BA CC   JMP $CCBA
CCCD   60         RTS

CCCE   A9 0C      LDA #$0C        CLS Load A with CTRL L.
CCD0   2C A9 11   BIT $11A9       BIT instructions are used to
```

```
CCD3   2C A9 20    BIT $20A9       hide the loading of A with
CCD6   2C A9 3F    BIT $3FA9       different values
CCD9   24 2E       BIT $2E         If CTRL O flag is set then set
CCDB   30 33       BMI $CD10       flags and exit.
CCDD   48          PHA             Save char to be printed.
CCDE   C9 20       CMP #$20        If control character do not
CCE0   90 0B       BCC $CCED       check cursor position.
CCE2   A5 30       LDA $30         Compare cursor position with
CCE4   C5 31       CMP $31         line width.
CCE6   D0 03       BNE $CCEB
CCE8   20 F0 CB    JSR $CBF0       If past end, print Newline.
CCEB   E6 30       INC $30         Advance cursor column.
CCED   68          PLA
CCEE   2C F1 02    BIT $02F1
CCF1   10 08       BPL $CCFB       Printer is off.
CCF3   48          PHA
CCF4   20 3E 02    JSR $023E       Send byte to printer.
CCF7   68          PLA
CCF8   29 FF       AND #$FF        Set flags and exit.
CCFA   60          RTS

CCFB   86 27       STX $27         Save X register.
CCFD   AA          TAX
CCFE   20 7C F7    JSR $F77C       Print character to screen.
CD01   C9 20       CMP #$20
CD03   90 04       BCC $CD09       Control character.
CD05   C9 7F       CMP #$7F
CD07   D0 05       BNE $CD0E       Character is not DEL.
CD09   AE 69 02    LDX $0269
CD0C   86 30       STX $30         Update Basic's cursor column.
CD0E   A6 27       LDX $27
CD10   29 FF       AND #$FF
CD12   60          RTS

CD13   6C F5 02    JMP ($02F5)     ! Command.

CD16   A9 80       LDA #$80        TRON The BIT instruction is
CD18   2C A9 00    BIT $00A9       used to hide an entry point.
CD1B   8D F4 02    STA $02F4       Set the TRACE flag to content
CD1E   60          RTS             of accumulator.

CD1F   A5 2C       LDA $2C         Part of READ command.
CD21   F0 13       BEQ $CD36       Branch if REDO FROM START.
CD23   30 04       BMI $CD29
CD25   A0 FF       LDY #$FF
CD27   D0 04       BNE $CD2D
CD29   A5 AE       LDA $AE
CD2B   A4 AF       LDY $AF
CD2D   85 A8       STA $A8
CD2F   84 A9       STY $A9
CD31   A2 A8       LDX #$A8
CD33   4C 7E C4    JMP $C47E       "TYPE MISMATCH ERROR".

CD36   A9 85       LDA #$85
CD38   A0 CE       LDY #$CE
CD3A   20 B0 CC    JSR $CCB0       Print out string after ".
CD3D   A5 AC       LDA $AC         Restore program position
CD3F   A4 AD       LDY $AD         pointer.
CD41   85 E9       STA $E9
CD43   84 EA       STY $EA
CD45   60          RTS

CD46   20 D2 D4    JSR $D4D2       GET Check for ILLEGAL DIRECT
CD49   A2 36       LDX #$36        error.
```

```
CD4B    A0 00       LDY #$00
CD4D    84 36       STY $36
CD4F    A9 40       LDA #$40
CD51    20 8F CD    JSR $CD8F       Get input by using READ
CD54    60          RTS             command.

CD55    46 2E       LSR $2E         INPUT Turn off CTRL O flag.
CD57    C9 22       CMP #$22
CD59    D0 0B       BNE $CD66       Double quote is not present.
CD5B    20 25 D0    JSR $D025       Get string after " and update
CD5E    A9 3B       LDA #$3B        position pointer.
CD60    20 67 D0    JSR $D067       Check for ;
CD63    20 B3 CC    JSR $CCB3       Print out string after "
CD66    20 D2 D4    JSR $D4D2       Check for ILLEGAL DIRECT error
CD69    A9 2C       LDA #$2C
CD6B    85 34       STA $34
CD6D    A9 00       LDA #$00
CD6F    85 17       STA $17         Reset CTRL C flag.
CD71    20 80 CD    JSR $CD80       Print ? and input line from KB
CD74    A5 35       LDA $35
CD76    D0 16       BNE $CD8E
CD78    A5 17       LDA $17
CD7A    F0 F1       BEQ $CD6D       CTRL C flag is still off.
CD7C    18          CLC
CD7D    4C 80 C9    JMP $C980       Sort out CTRL C.

CD80    20 D7 CC    JSR $CCD7       Print ?.
CD83    20 D4 CC    JSR $CCD4       Print a space.
CD86    4C 92 C5    JMP $C592       Input line from keyboard.

CD89    A6 B0       LDX $B0         READ
CD8B    A4 B1       LDY $B1
CD8D    A9 98       LDA #$98        Clear REDO FROM START flag.
CD8F    85 2C       STA $2C
CD91    86 B2       STX $B2
CD93    84 B3       STY $B3
CD95    20 88 D1    JSR $D188       Get variable from text.
CD98    85 B8       STA $B8         Save address of pointer.
CD9A    84 B9       STY $B9
CD9C    A5 E9       LDA $E9
CD9E    A4 EA       LDY $EA         Copy program position pointer.
CDA0    85 BA       STA $BA
CDA2    84 BB       STY $BB
CDA4    A6 B2       LDX $B2         Copy DATA pointer.
CDA6    A4 B3       LDY $B3
CDA8    86 E9       STX $E9
CDAA    84 EA       STY $EA
CDAC    20 E8 00    JSR $00E8       Get next non space character.
CDAF    D0 1D       BNE $CDCE       Branch if not end of line.
CDB1    24 2C       BIT $2C
CDB3    50 0D       BVC $CDC2
CDB5    20 78 EB    JSR $EB78       Read next key from keyboard.
CDB8    10 FB       BPL $CDB5       Wait until key is valid.
CDBA    85 35       STA $35
CDBC    A2 34       LDX #$34
CDBE    A0 00       LDY #$00
CDC0    F0 08       BEQ $CDCA
CDC2    30 71       BMI $CE35
CDC4    20 D7 CC    JSR $CCD7       Print ?
CDC7    20 80 CD    JSR $CD80       Print ? and input line from KB
CDCA    86 E9       STX $E9
CDCC    84 EA       STY $EA         Set position of input.
CDCE    20 E2 00    JSR $00E2       Get next char from text.
CDD1    24 28       BIT $28
```

```
CDD3   10 31       BPL $CE06       Variable is not string type.
CDD5   24 2C       BIT $2C
CDD7   50 09       BVC $CDE2
CDD9   E8          INX
CDDA   86 E9       STX $E9
CDDC   A9 00       LDA #$00
CDDE   85 24       STA $24
CDE0   F0 0C       BEQ $CDEE
CDE2   85 24       STA $24
CDE4   C9 22       CMP #$22
CDE6   F0 07       BEQ $CDEF
CDE8   A9 3A       LDA #$3A
CDEA   85 24       STA $24
CDEC   A9 2C       LDA #$2C
CDEE   18          CLC
CDEF   85 25       STA $25
CDF1   A5 E9       LDA $E9
CDF3   A4 EA       LDY $EA
CDF5   69 00       ADC #$00
CDF7   90 01       BCC $CDFA
CDF9   C8          INY
CDFA   20 BB D5    JSR $D5BB       Get string after "
CDFD   20 0D D9    JSR $D90D       Set program ptr to content of
CE00   20 51 CB    JSR $CB51       $E0/$E1 and assign string.
CE03   4C 0E CE    JMP $CE0E

CE06   20 E7 DF    JSR $DFE7       Get number.
CE09   A5 29       LDA $29         Load integer variable flag.
CE0B   20 39 CB    JSR $CB39       Assign integer.
CE0E   20 E8 00    JSR $00E8       Get next char from text.
CE11   F0 07       BEQ $CE1A       End of line reached.
CE13   C9 2C       CMP #$2C
CE15   F0 03       BEQ $CE1A       Character is a comma.
CE17   4C 1F CD    JMP $CD1F
CE1A   A5 E9       LDA $E9         Copy program position into
CE1C   A4 EA       LDY $EA         data pointer.
CE1E   85 B2       STA $B2
CE20   84 B3       STY $B3
CE22   A5 BA       LDA $BA         Copy temporary pointer into
CE24   A4 BB       LDY $BB         program position.
CE26   85 E9       STA $E9
CE28   84 EA       STY $EA
CE2A   20 E8 00    JSR $00E8       Get next character.
CE2D   F0 2C       BEQ $CE5B       End of line reached.
CE2F   20 65 D0    JSR $D065       Test for comma.
CE32   4C 95 CD    JMP $CD95       Get next variable.

CE35   20 4E CA    JSR $CA4E       Find end of statement.
CE38   C8          INY
CE39   AA          TAX
CE3A   D0 12       BNE $CE4E
CE3C   A2 2A       LDX #$2A
CE3E   C8          INY
CE3F   B1 E9       LDA ($E9),Y     Give "TYPE MISMATCH ERROR" if
CE41   F0 69       BEQ $CEAC       run out of program.
CE43   C8          INY
CE44   B1 E9       LDA ($E9),Y     Copy line number to temporary
CE46   85 AE       STA $AE         pointer.
CE48   C8          INY
CE49   B1 E9       LDA ($E9),Y
CE4B   C8          INY
CE4C   85 AF       STA $AF
CE4E   B1 E9       LDA ($E9),Y
CE50   AA          TAX
```

```
CE51   20 3F CA    JSR $CA3F       Add X to content of $E9/$EA.
CE54   E0 91       CPX #$91
CE56   D0 DD       BNE $CE35
CE58   4C CE CD    JMP $CDCE       Jump back to do more.
CE5B   A5 B2       LDA $B2
CE5D   A4 B3       LDY $B3
CE5F   A6 2C       LDX $2C
CE61   10 03       BPL $CE66       REDO FROM START flag is set.
CE63   4C 5C C9    JMP $C95C       Exit and update DATA pointer.
CE66   A0 00       LDY #$00
CE68   B1 B2       LDA ($B2),Y
CE6A   F0 07       BEQ $CE73       No extra data.
CE6C   A9 74       LDA #$74
CE6E   A0 CE       LDY #$CE
CE70   4C B0 CC    JMP $CCB0       Print "EXTRA IGNORED".
CE73   60          RTS

CE74   3F 45 58 54 52 41 20 49     ?EXTRA I
CE7C   47 4E 4F 52 45 44 0D 0A     GNORED
CE84   00 3F 52 45 44 4F 20 46      ?REDO F
CE8C   52 4F 4D 20 53 54 41 52     ROM STAR
CE94   54 0D 0A 00                 T

CE98   D0 04       BNE $CE9E       NEXT more input after token.
CE9A   A0 00       LDY #$00
CE9C   F0 03       BEQ $CEA1       No variable name given.
CE9E   20 88 D1    JSR $D188       Get variable from text.
CEA1   85 B8       STA $B8         Save pointer to variable.
CEA3   84 B9       STY $B9
CEA5   20 C6 C3    JSR $C3C6       Search for that var. on stack.
CEA8   F0 04       BEQ $CEAE       Variable found.
CEAA   A2 00       LDX #$00        Print "TYPE MISMATCH ERROR".
CEAC   F0 66       BEQ $CF14
CEAE   9A          TXS
CEAF   8A          TXA
CEB0   18          CLC
CEB1   69 04       ADC #$04
CEB3   48          PHA
CEB4   69 06       ADC #$06
CEB6   85 93       STA $93
CEB8   68          PLA
CEB9   A0 01       LDY #$01
CEBB   20 7B DE    JSR $DE7B       Unpack floating point number.
CEBE   BA          TSX
CEBF   BD 09 01    LDA $0109,X     Take sign byte off stack and
CEC2   85 D5       STA $D5         put it in FPA sign byte.
CEC4   A5 B8       LDA $B8
CEC6   A4 B9       LDY $B9
CEC8   20 22 DB    JSR $DB22       Add in STEP value.
CECB   20 A9 DE    JSR $DEA9       Pack main FPA and put it in
CECE   A0 01       LDY #$01        memory.
CED0   20 4E DF    JSR $DF4E       Compare main FPA with number
CED3   BA          TSX             pointed to by Y (MSB) and A.
CED4   38          SEC
CED5   FD 09 01    SBC $0109,X
CED8   F0 17       BEQ $CEF1       Exit current FOR-NEXT loop.
CEDA   BD 0F 01    LDA $010F,X     Take line number and program
CEDD   85 A8       STA $A8         position off stack so that
CEDF   BD 10 01    LDA $0110,X     program can go back to just
CEE2   85 A9       STA $A9         after the FOR statement.
CEE4   BD 12 01    LDA $0112,X
CEE7   85 E9       STA $E9
CEE9   BD 11 01    LDA $0111,X
CEEC   85 EA       STA $EA
```

```
CEEE   4C C1 C8   JMP $C8C1       Goto next statement.
CEF1   8A         TXA             Adjust stack pointer to having
CEF2   69 11      ADC #$11        one less loop.
CEF4   AA         TAX
CEF5   9A         TXS
CEF6   20 E8 00   JSR $00E8       Get next char from program.
CEF9   C9 2C      CMP #$2C        Execute next statement if
CEFB   D0 F1      BNE $CEEE       character is not a comma.
CEFD   20 E2 00   JSR $00E2       Get next char.
CF00   20 9E CE   JSR $CE9E       Go round loop again.

CF03   20 17 CF   JSR $CF17       GET NUMERIC EXPRESSION.
CF06   18         CLC             Evaluate expression.
CF07   24 38      BIT $38         Hides a SEC instruction.
CF09   24 28      BIT $28
CF0B   30 03      BMI $CF10       Expression is string type.
CF0D   B0 03      BCS $CF12
CF0F   60         RTS

CF10   B0 FD      BCS $CF0F       String type allowed if C=1.
CF12   A2 A8      LDX #$A8
CF14   4C 7E C4   JMP $C47E       Print "TYPE MISMATCH ERROR".

CF17   A6 E9      LDX $E9         EVALUATE EXPRESSION.
CF19   D0 02      BNE $CF1D
CF1B   C6 EA      DEC $EA         Decrement text pointer.
CF1D   C6 E9      DEC $E9
CF1F   A2 00      LDX #$00
CF21   24 48      BIT $48         Hides a PHA instruction.
CF23   8A         TXA
CF24   48         PHA
CF25   A9 01      LDA #$01        Check for 2 free bytes on
CF27   20 37 C4   JSR $C437       stack.
CF2A   20 00 D0   JSR $D000       Get item.
CF2D   A9 00      LDA #$00        Clear relational operator bit
CF2F   85 BC      STA $BC         mark.
CF31   20 E8 00   JSR $00E8       Get next character.
CF34   38         SEC
CF35   E9 D3      SBC #$D3        Token is in list before that
CF37   90 17      BCC $CF50       of >, < or =.
CF39   C9 03      CMP #$03        Token is in list after that
CF3B   B0 13      BCS $CF50       of >, < or =.
CF3D   C9 01      CMP #$01        Form comparator bit mask.
CF3F   2A         ROL A           001 for >
CF40   49 01      EOR #$01        010 for =
CF42   45 BC      EOR $BC         100 for <
CF44   C5 BC      CMP $BC         Error if one of these tokens
CF46   90 61      BCC $CFA9       has appeared twice in a row.
CF48   85 BC      STA $BC
CF4A   20 E2 00   JSR $00E2       Get next character.
CF4D   4C 34 CF   JMP $CF34       Test next char for <, > or =.
CF50   A6 BC      LDX $BC
CF52   D0 2C      BNE $CF80       Relational Operator.
CF54   B0 7F      BCS $CFD5       If not binary operator, finish
CF56   69 07      ADC #$07        expression.
CF58   90 7B      BCC $CFD5
CF5A   65 28      ADC $28         Add string flag plus carry.
CF5C   D0 03      BNE $CF61       Jump to concatenate strings if
CF5E   4C 67 D7   JMP $D767       operator was a "+".
CF61   69 FF      ADC #$FF        Multiply operator by 3 and put
CF63   85 91      STA $91         value into Y.
CF65   0A         ASL A
CF66   65 91      ADC $91
CF68   A8         TAY
```

```
CF69   68          PLA             If old operator priority was
CF6A   D9 CC C0    CMP $C0CC,Y     greater or equal, then exit
CF6D   B0 6B       BCS $CFDA       this level.
CF6F   20 06 CF    JSR $CF06       Check numeric type.
CF72   48          PHA             Save operator priority.
CF73   20 99 CF    JSR $CF99       Perform higher priority oper'n
CF76   68          PLA             Restore old operator priority.
CF77   A4 BA       LDY $BA         Branch if not end of
CF79   10 17       BPL $CF92       expression.
CF7B   AA          TAX             Exit if no operator pending on
CF7C   F0 5A       BEQ $CFD8       stack.
CF7E   D0 63       BNE $CFE3       Pull work FPA and exit.
CF80   46 28       LSR $28         Set C if string type.
CF82   8A          TXA             Get mask, bottom bit set if
CF83   2A          ROL A           string.
CF84   A6 E9       LDX $E9         Decrement text pointer.
CF86   D0 02       BNE $CF8A
CF88   C6 EA       DEC $EA
CF8A   C6 E9       DEC $E9
CF8C   A0 1B       LDY #$1B        Operator code.
CF8E   85 BC       STA $BC         Save relation mask.
CF90   D0 D7       BNE $CF69       Branch for another operator.
CF92   D9 CC C0    CMP $C0CC,Y     If next operator is of lower
CF95   B0 4C       BCS $CFE3       priority then exit.
CF97   90 D9       BCC $CF72       Get next operator.
CF99   B9 CE C0    LDA $C0CE,Y     Push operator action address
CF9C   48          PHA             on to the stack.
CF9D   B9 CD C0    LDA $C0CD,Y
CFA0   48          PHA
CFA1   20 AC CF    JSR $CFAC       Set up and perform operation.
CFA4   A5 BC       LDA $BC
CFA6   4C 22 CF    JMP $CF22       Get operator code & loop again

CFA9   4C 70 D0    JMP $D070       PRINT "SYNTAX ERROR".

CFAC   A5 D5       LDA $D5         SET UP AND PERFORM OPERATION.
CFAE   BE CC C0    LDX $C0CC,Y     Get sign of FPA and put
CFB1   A8          TAY             operator priority into Y.
CFB2   68          PLA
CFB3   85 91       STA $91         Set up action address.
CFB5   68          PLA
CFB6   85 92       STA $92
CFB8   E6 91       INC $91         Increment address.
CFBA   D0 02       BNE $CFBE
CFBC   E6 92       INC $92
CFBE   98          TYA
CFBF   48          PHA             Push sign of main FPA.
CFC0   20 F4 DE    JSR $DEF4       Round off main FPA
CFC3   A5 D4       LDA $D4
CFC5   48          PHA             Push main FPA on to stack
CFC6   A5 D3       LDA $D3
CFC8   48          PHA
CFC9   A5 D2       LDA $D2
CFCB   48          PHA
CFCC   A5 D1       LDA $D1
CFCE   48          PHA
CFCF   A5 D0       LDA $D0
CFD1   48          PHA
CFD2   6C 91 00    JMP ($0091)     Perform operation.
CFD5   A0 FF       LDY #$FF        End of expression indicator.
CFD7   68          PLA             If no operators pending then
CFD8   F0 23       BEQ $CFFD       exit.
CFDA   C9 64       CMP #$64        If not relational operator
CFDC   F0 03       BEQ $CFE1       check for numeric type.
```

```
CFDE   20 06 CF   JSR $CF06
CFE1   84 BA      STY $BA         Save operator code.
CFE3   68         PLA             Pull operator code and shift
CFE4   4A         LSR A           it before putting into $2D.
CFE5   85 2D      STA $2D
CFE7   68         PLA             Restore work floating point
CFE8   85 D8      STA $D8         accumulator from stack.
CFEA   68         PLA
CFEB   85 D9      STA $D9
CFED   68         PLA
CFEE   85 DA      STA $DA
CFF0   68         PLA
CFF1   85 DB      STA $DB
CFF3   68         PLA
CFF4   85 DC      STA $DC
CFF6   68         PLA
CFF7   85 DD      STA $DD
CFF9   45 D5      EOR $D5
CFFB   85 DE      STA $DE         Set sign difference flag.
CFFD   A5 D0      LDA $D0
CFFF   60         RTS             Exit.

D000   A9 00      LDA #$00        GET ITEM.
D002   85 28      STA $28         Clear string type flag.
D004   20 E2 00   JSR $00E2       Get next character.
D007   B0 03      BCS $D00C
D009   4C E7 DF   JMP $DFE7       If digit then get number.
D00C   20 16 D2   JSR $D216       If "A-Z" then get value from
D00F   B0 6B      BCS $D07C       variable.
D011   C9 2E      CMP #$2E        If "." or "#" then get number.
D013   F0 F4      BEQ $D009
D015   C9 23      CMP #$23
D017   F0 F0      BEQ $D009
D019   C9 CD      CMP #$CD        If "-" then handle unary minus
D01B   F0 58      BEQ $D075       number.
D01D   C9 CC      CMP #$CC        If "+" token then ignore it.
D01F   F0 E3      BEQ $D004
D021   C9 22      CMP #$22        If not " then skip string bit.
D023   D0 0F      BNE $D034
D025   A5 E9      LDA $E9         Get text pointer + 1 into A
D027   A4 EA      LDY $EA         and Y.
D029   69 00      ADC #$00
D02B   90 01      BCC $D02E
D02D   C8         INY
D02E   20 B5 D5   JSR $D5B5       Get string after ".
D031   4C 0D D9   JMP $D90D       Update text pointer and exit.
D034   C9 CA      CMP #$CA        If "NOT" token then use
D036   D0 13      BNE $D04B       operator at $18 and go round
D038   A0 18      LDY #$18        again.
D03A   D0 3B      BNE $D077

D03C   20 A9 D2   JSR $D2A9       NOT Convert main FPA to
D03F   A5 D4      LDA $D4         signed integer.
D041   49 FF      EOR #$FF        Invert LSB into Y.
D043   A8         TAY
D044   A5 D3      LDA $D3         Invert MSB into A.
D046   49 FF      EOR #$FF
D048   4C 99 D4   JMP $D499       Convert to main FPA and exit.
D04B   C9 C4      CMP #$C4        If "FN" token then goto FN
D04D   D0 03      BNE $D052       call address.
D04F   4C 22 D5   JMP $D522
D052   C9 D6      CMP #$D6        If function token is >= than
D054   90 03      BCC $D059       #D6 then deal with function.
D056   4C A0 D0   JMP $D0A0
```

```
D059   20 62 D0   JSR $D062       GET EXPRESSION IN (). Check
D05C   20 17 CF   JSR $CF17       ")" and evaluate expression.
D05F   A9 29      LDA #$29        Check for ")"
D061   2C A9 28   BIT $28A9       Check for "(" - hidden in BIT.
D064   2C A9 2C   BIT $2CA9       Check for "," - hidden in BIT.
D067   A0 00      LDY #$00        Check for char in A.
D069   D1 E9      CMP ($E9),Y
D06B   D0 03      BNE $D070       "SYNTAX ERROR" if not present.
D06D   4C E2 00   JMP $00E2       Get next character.
D070   A2 10      LDX #$10        Print "SYNTAX ERROR".
D072   4C 7E C4   JMP $C47E
D075   A0 15      LDY #$15        Unary minus operator.
D077   68         PLA             Pull return address and jump
D078   68         PLA             to next operator.
D079   4C 73 CF   JMP $CF73

D07C   20 88 D1   JSR $D188       GET VALUE FROM VARIABLE. Get
D07F   85 D3      STA $D3         variable and set up pointer.
D081   84 D4      STY $D4
D083   A6 28      LDX $28         If a string then clear the
D085   F0 05      BEQ $D08C       Founding byte and exit.
D087   A2 00      LDX #$00
D089   86 DF      STX $DF
D08B   60         RTS

D08C   A6 29      LDX $29         If a real number then get
D08E   10 0D      BPL $D09D       value.
D090   A0 00      LDY #$00
D092   B1 D3      LDA ($D3),Y     Get MSB of integer into X.
D094   AA         TAX
D095   C8         INY
D096   B1 D3      LDA ($D3),Y     Get LSB of integer into Y.
D098   A8         TAY
D099   8A         TXA             Put MSB in A.
D09A   4C 99 D4   JMP $D499       Convert A/Y to FPA & exit.
D09D   4C 7B DE   JMP $DE7B       Unpack number into FPA.
D0A0   0A         ASL A           Double token and save it on
D0A1   48         PHA             stack.
D0A2   AA         TAX
D0A3   20 E2 00   JSR $00E2       Get character.
D0A6   E0 DB      CPX #$DB        If token is CHR$ or less then
D0A8   90 24      BCC $D0CE       handle single argument.
D0AA   E0 E7      CPX #$E7        If token is POINT or less then
D0AC   90 23      BCC $D0D1       no argument is needed.
D0AE   20 62 D0   JSR $D062       Check for "(".
D0B1   20 17 CF   JSR $CF17       Evaluate expression.
D0B4   20 65 D0   JSR $D065       Check for ",".
D0B7   20 08 CF   JSR $CF08       Check for string type.
D0BA   68         PLA
D0BB   AA         TAX             Save table offset in X.
D0BC   A5 D4      LDA $D4
D0BE   48         PHA             Push pointer to string block
D0BF   A5 D3      LDA $D3         on stack.
D0C1   48         PHA
D0C2   8A         TXA             Push table offset on stack.
D0C3   48         PHA
D0C4   20 C8 D8   JSR $D8C8       Get 1 byte expression into X.
D0C7   68         PLA
D0C8   A8         TAY             Push expression byte.
D0C9   8A         TXA
D0CA   48         PHA
D0CB   4C D3 D0   JMP $D0D3       Set up and execute function.
```

```
D0CE   20 59 D0   JSR $D059       Get expression in brackets.
D0D1   68         PLA            Get table offset into Y.
D0D2   A8         TAY
D0D3   B9 DE BF   LDA $BFDE,Y    Set up action address.
D0D6   85 C4      STA $C4
D0D8   B9 DF BF   LDA $BFDF,Y
D0DB   85 C5      STA $C5
D0DD   20 C3 00   JSR $00C3      Execute function. Check for
D0E0   4C 06 CF   JMP $CF06      numeric types and exit.

D0E3   A0 FF      LDY #$FF       Routine for OR and routine
D0E5   2C A0 00   BIT $00A0      for AND (hidden by BIT).
D0E8   84 26      STY $26        Initialise $26.
D0EA   20 A9 D2   JSR $D2A9      Convert FPA to signed integer.
D0ED   A5 D3      LDA $D3        Transfer integer to $24/$25
D0EF   45 26      EOR $26        inverting as well if using the
D0F1   85 24      STA $24        OR operator.
D0F3   A5 D4      LDA $D4
D0F5   45 26      EOR $26
D0F7   85 25      STA $25
D0F9   20 D5 DE   JSR $DED5      Copy work FPA into main FPA.
D0FC   20 A9 D2   JSR $D2A9      Convert to signed integer.
D0FF   A5 D4      LDA $D4        Get result into A and Y and
D101   45 26      EOR $26        AND it with the other integer.
D103   25 25      AND $25        If OR is being used then
D105   45 26      EOR $26        invert A/Y before and after
D107   A8         TAY            the ANDing.
D108   A5 D3      LDA $D3
D10A   45 26      EOR $26
D10C   25 24      AND $24
D10E   45 26      EOR $26
D110   4C 99 D4   JMP $D499      Convert result to FPA & exit.

D113   20 09 CF   JSR $CF09      RELATIONAL OPERATORS >, =, <
D116   B0 13      BCS $D12B      Check type & branch if string.
D118   A5 DD      LDA $DD        Put work FPA in packed format.
D11A   09 7F      ORA #$7F
D11C   25 D9      AND $D9
D11E   85 D9      STA $D9
D120   A9 D8      LDA #$D8       Set Y (MSB) and A to point to
D122   A0 00      LDY #$00       work FPA.
D124   20 4C DF   JSR $DF4C      Compare main and work FPAs.
D127   AA         TAX            Save result in X and skip
D128   4C 5E D1   JMP $D15E      over string section.

D12B   A9 00      LDA #$00       Clear string flag.
D12D   85 28      STA $28
D12F   C6 BC      DEC $BC        Adjust relational flags.
D131   20 D0 D7   JSR $D7D0      Set up string.
D134   85 D0      STA $D0        Store block in main FPA.
D136   86 D1      STX $D1
D138   84 D2      STY $D2
D13A   A5 DB      LDA $DB        Get pointer to first string in
D13C   A4 DC      LDY $DC        work FPA.
D13E   20 D4 D7   JSR $D7D4      Set up string.
D141   86 DB      STX $DB        Store block in work FPA and X.
D143   84 DC      STY $DC
D145   AA         TAX
D146   38         SEC
D147   E5 D0      SBC $D0        Set up length of shorter
D149   F0 08      BEQ $D153      string in X.
D14B   A9 01      LDA #$01       A=0 if strings are same length
D14D   90 04      BCC $D153      A=1 if string pointed to by
D14F   A6 D0      LDX $D0        main FPA is longer otherwise
```

```
D151   A9 FF       LDA #$FF        A=#FF.
D153   85 D5       STA $D5         Save length difference flag.
D155   A0 FF       LDY #$FF        Set Y and loop counter.
D157   E8          INX
D158   C8          INY
D159   CA          DEX
D15A   D0 07       BNE $D163       Search through the strings
D15C   A6 D5       LDX $D5         comparing each of the
D15E   30 0F       BMI $D16F       characters until one string
D160   18          CLC             has ended.
D161   90 0C       BCC $D16F
D163   B1 DB       LDA ($DB),Y
D165   D1 D1       CMP ($D1),Y
D167   F0 EF       BEQ $D158       Characters match.
D169   A2 FF       LDX #$FF
D16B   B0 02       BCS $D16F       Set X to flag difference.
D16D   A2 01       LDX #$01
D16F   E8          INX             Form comparison result bit.
D170   8A          TXA
D171   2A          ROL A
D172   25 2D       AND $2D         Mask with relational operator
D174   F0 02       BEQ $D178       mask and branch if false.
D176   A9 FF       LDA #$FF        Set FPA according to content
D178   4C 24 DF    JMP $DF24       of A and exit.

D17B   20 65 D0    JSR $D065       Check for ","
D17E   AA          TAX             DIM
D17F   20 8D D1    JSR $D18D       Handle array dimensioning.
D182   20 E8 00    JSR $00E8       Get next character.
D185   D0 F4       BNE $D17B       Loop until end of statement.
D187   60          RTS

D188   A2 00       LDX #$00        GET VARIABLE FROM TEXT.
D18A   20 E8 00    JSR $00E8
D18D   86 27       STX $27
D18F   85 B4       STA $B4         Put first char in $64.
D191   20 E8 00    JSR $00E8
D194   20 16 D2    JSR $D216       Give error if not a letter.
D197   B0 03       BCS $D19C
D199   4C 70 D0    JMP $D070       Print "SYNTAX ERROR".
D19C   A2 00       LDX #$00
D19E   86 28       STX $28         Clear type flags.
D1A0   86 29       STX $29
D1A2   20 E2 00    JSR $00E2       Next character.
D1A5   90 05       BCC $D1AC
D1A7   20 16 D2    JSR $D216       Check that it is a letter.
D1AA   90 0B       BCC $D1B7       Character not in range A-Z.
D1AC   AA          TAX             Save second char.
D1AD   20 E2 00    JSR $00E2       Loop until not 0-9 or A-Z.
D1B0   90 FB       BCC $D1AD
D1B2   20 16 D2    JSR $D216
D1B5   B0 F6       BCS $D1AD
D1B7   C9 24       CMP #$24        Test for string indicator.
D1B9   D0 06       BNE $D1C1       Character is not a $.
D1BB   A9 FF       LDA #$FF
D1BD   85 28       STA $28         Set string type.
D1BF   D0 10       BNE $D1D1
D1C1   C9 25       CMP #$25        Test for integer indicator.
D1C3   D0 13       BNE $D1D8       Character is not a %.
D1C5   A5 2B       LDA $2B
D1C7   30 D0       BMI $D199
D1C9   A9 80       LDA #$80
D1CB   85 29       STA $29         Set integer flag.
D1CD   05 B4       ORA $B4
```

```
D1CF   85 B4       STA $B4
D1D1   8A          TXA             Set top bits of name according
D1D2   09 80       ORA #$80        to type of variable.
D1D4   AA          TAX
D1D5   20 E2 00    JSR $00E2       Next character.
D1D8   86 B5       STX $B5
D1DA   38          SEC
D1DB   05 2B       ORA $2B
D1DD   E9 28       SBC #$28        Handle an array if following
D1DF   D0 03       BNE $D1E4       char is "(" with number.
D1E1   4C BB D2    JMP $D2BB

D1E4   24 2B       BIT $2B         If STORE / RECALL flag bit is
D1E6   70 F9       BVS $D1E1       set then handle an array.
D1E8   A9 00       LDA #$00
D1EA   85 2B       STA $2B         Clear variable flag.
D1EC   A5 9C       LDA $9C         X and A set to end Basic.
D1EE   A6 9D       LDX $9D
D1F0   A0 00       LDY #$00
D1F2   86 CF       STX $CF
D1F4   85 CE       STA $CE         If end variables reached then
D1F6   E4 9F       CPX $9F         create new one if necessary.
D1F8   D0 04       BNE $D1FE
D1FA   C5 9E       CMP $9E
D1FC   F0 24       BEQ $D222
D1FE   A5 B4       LDA $B4
D200   D1 CE       CMP ($CE),Y
D202   D0 08       BNE $D20C
D204   A5 B5       LDA $B5
D206   C8          INY             If variable is found then set
D207   D1 CE       CMP ($CE),Y     pointer and exit.
D209   F0 6C       BEQ $D277
D20B   88          DEY
D20C   18          CLC
D20D   A5 CE       LDA $CE
D20F   69 07       ADC #$07        Otherwise add 7 to pointer and
D211   90 E1       BCC $D1F4       go round to search again.
D213   E8          INX
D214   D0 DC       BNE $D1F2
D216   C9 41       CMP #$41        Set C if char in A is in the
D218   90 07       BCC $D221       Ascii range A - Z.
D21A   E9 5B       SBC #$5B
D21C   38          SEC
D21D   E9 A5       SBC #$A5
D21F   B0 00       BCS $D221
D221   60          RTS

D222   68          PLA             Routine jumps here if variable
D223   48          PHA             is not found. A and Y are set
D224   C9 7E       CMP #$7E        to point to $E207 if a value
D226   D0 0D       BNE $D235       is needed (which will be 0).
D228   BA          TSX             If a new variable is to be
D229   BD 02 01    LDA $0102,X     created then execute routine
D22C   C9 D0       CMP #$D0        below.
D22E   D0 05       BNE $D235
D230   A9 07       LDA #$07
D232   A0 E2       LDY #$E2
D234   60          RTS

D235   A5 9E       LDA $9E         Routine to open up space for a
D237   A4 9F       LDY $9F         variable.
D239   85 CE       STA $CE         Copy end of variables pointer.
D23B   84 CF       STY $CF
D23D   A5 A0       LDA $A0
```

```
D23F   A4 A1       LDY $A1
D241   85 C9       STA $C9          Copy end of Arrays pointer.
D243   84 CA       STY $CA
D245   18          CLC              Add 7 to copy of end of Arrays
D246   69 07       ADC #$07         pointer so that a new variable
D248   90 01       BCC $D24B        can be inserted in variable
D24A   C8          INY              block.
D24B   85 C7       STA $C7
D24D   84 C8       STY $C8
D24F   20 F4 C3    JSR $C3F4        Shift up arrays.
D252   A5 C7       LDA $C7
D254   A4 C8       LDY $C8
D256   C8          INY
D257   85 9E       STA $9E          Update end of Variables
D259   84 9F       STY $9F          pointer.
D25B   A0 00       LDY #$00
D25D   A5 B4       LDA $B4          Copy across name of new
D25F   91 CE       STA ($CE),Y      variable.
D261   C8          INY
D262   A5 B5       LDA $B5
D264   91 CE       STA ($CE),Y
D266   A9 00       LDA #$00         Set value of variable to zero.
D268   C8          INY
D269   91 CE       STA ($CE),Y
D26B   C8          INY
D26C   91 CE       STA ($CE),Y
D26E   C8          INY
D26F   91 CE       STA ($CE),Y
D271   C8          INY
D272   91 CE       STA ($CE),Y
D274   C8          INY
D275   91 CE       STA ($CE),Y
D277   A5 CE       LDA $CE
D279   18          CLC
D27A   69 02       ADC #$02         Set $66 and $B7 to point to
D27C   A4 CF       LDY $CF          value of variable (2 beyond
D27E   90 01       BCC $D281        its name).
D280   C8          INY
D281   85 B6       STA $B6
D283   84 B7       STY $B7
D285   60          RTS

D286   A5 26       LDA $26          Set $C7 and $C8 to point to
D288   0A          ASL A            start of array cells.
D289   69 05       ADC #$05
D28B   65 CE       ADC $CE
D28D   A4 CF       LDY $CF
D28F   90 01       BCC $D292
D291   C8          INY
D292   85 C7       STA $C7
D294   84 C8       STY $C8
D296   60          RTS

D297   90 80 00 00 00              F1oating point value of -32768

D29C   20 E2 00    JSR $00E2        Next character.
D29F   20 17 CF    JSR $CF17        Get numeric expression.
D2A2   20 06 CF    JSR $CF06        Check that it is numeric.
D2A5   A5 D5       LDA $D5
D2A7   30 0D       BMI $D2B6        Error if negative subscript.

D2A9   A5 D0       LDA $D0          MAIN FPA INTO SIGNED INTEGER.
D2AB   C9 90       CMP #$90         Number is less than 32768 in
D2AD   90 09       BCC $D2B8        magnitude.
```

```
D2AF   A9 97       LDA #$97        Compare number with -32768
D2B1   A0 D2       LDY #$D2        held at $0297.
D2B3   20 4C DF    JSR $DF4C
D2B6   D0 7E       BNE $D336       Error if not equal.
D2B8   4C 8C DF    JMP $DF8C       Convert to integer and exit.

D2BB   A5 2B       LDA $2B         HANDLE ARRAY. If STORE/RECALL
D2BD   D0 47       BNE $D306       then skip handling subscripts.
D2BF   A5 27       LDA $27
D2C1   05 29       ORA $29         Save flag bytes.
D2C3   48          PHA
D2C4   A5 28       LDA $28
D2C6   48          PHA
D2C7   A0 00       LDY #$00        Set initial count of
D2C9   98          TYA             subscripts and save it on
D2CA   48          PHA             stack.
D2CB   A5 B5       LDA $B5         Save address of last variable
D2CD   48          PHA             accessed.
D2CE   A5 B4       LDA $B4
D2D0   48          PHA
D2D1   20 9C D2    JSR $D29C       Get subscript.
D2D4   68          PLA
D2D5   85 B4       STA $B4         Restore address of last
D2D7   68          PLA             variable accessed.
D2D8   85 B5       STA $B5
D2DA   68          PLA             Restore subscript counter.
D2DB   A8          TAY
D2DC   BA          TSX
D2DD   BD 02 01    LDA $0102,X     Copy flags on to top of the
D2E0   48          PHA             stack.
D2E1   BD 01 01    LDA $0101,X
D2E4   48          PHA
D2E5   A5 D3       LDA $D3         Put on the size of new
D2E7   9D 02 01    STA $0102,X     dimension underneath them.
D2EA   A5 D4       LDA $D4
D2EC   9D 01 01    STA $0101,X
D2EF   C8          INY             Increment subscript number.
D2F0   20 E8 00    JSR $00E8       Get next character.
D2F3   C9 2C       CMP #$2C        Branch if it is a comma.
D2F5   F0 D2       BEQ $D2C9
D2F7   84 26       STY $26         Save subscript number.
D2F9   20 5F D0    JSR $D05F       Test for a ")".
D2FC   68          PLA
D2FD   85 28       STA $28         Restore variable type flags.
D2FF   68          PLA
D300   85 29       STA $29
D302   29 7F       AND #$7F        Set Dimension flag, 0=not dim.
D304   85 27       STA $27
D306   A6 9E       LDX $9E
D308   A5 9F       LDA $9F         Set pointer to next array.
D30A   86 CE       STX $CE
D30C   85 CF       STA $CF
D30E   C5 A1       CMP $A1
D310   D0 04       BNE $D316       Address of next array is same
D312   E4 A0       CPX $A0         as that of end arrays.
D314   F0 3F       BEQ $D355
D316   A0 00       LDY #$00
D318   B1 CE       LDA ($CE),Y
D31A   C8          INY
D31B   C5 B4       CMP $B4         Branch to $0336 if the next
D31D   D0 06       BNE $D325       array pointed to has same name
D31F   A5 B5       LDA $B5         as the array being put into
D321   D1 CE       CMP ($CE),Y     memory.
D323   F0 16       BEQ $D33B
```

```
D325   C8          INY
D326   B1 CE       LDA ($CE),Y     Add offset to pointer.
D328   18          CLC
D329   65 CE       ADC $CE
D32B   AA          TAX
D32C   C8          INY
D32D   B1 CE       LDA ($CE),Y
D32F   65 CF       ADC $CF
D331   90 D7       BCC $D30A       Next array if all is okay.
D333   A2 6B       LDX #$6B        Set X - "BAD SUBSCRIPT ERROR"
D335   2C A2 35    BIT $35A2       Set X - "ILLEGAL QTY ERROR"
D338   4C 7E C4    JMP $C47E       Print error message.

D33B   A2 78       LDX #$78        Set X - "REDIM'D ARRAY ERROR"
D33D   A5 27       LDA $27
D33F   D0 F7       BNE $D338       Print error.
D341   A5 2B       LDA $2B         If STORE / RECALL flag was set
D343   F0 02       BEQ $D347       then exit with C=1.
D345   38          SEC
D346   60          RTS

D347   20 86 D2    JSR $D286       Set up start of Array cells.
D34A   A5 26       LDA $26         Get number of subscripts.
D34C   A0 04       LDY #$04
D34E   D1 CE       CMP ($CE),Y     If not same number as
D350   D0 E1       BNE $D333       dimensioned then give error.
D352   4C EB D3    JMP $D3EB

D355   A5 2B       LDA $2B         End up here if array not found
D357   F0 08       BEQ $D361       STORE/RECALL flag is not set.
D359   20 3D E9    JSR $E93D       Reset cassette status and give
D35C   A2 2A       LDX #$2A        "OUT OF DATA" error.
D35E   4C 7E C4    JMP $C47E

D361   20 86 D2    JSR $D286       DIMENSION AN ARRAY. Set up
D364   20 44 C4    JSR $C444       start of array cells and check
D367   A9 00       LDA #$00        for enough memory for header.
D369   A8          TAY             Set MSB of correct array size.
D36A   85 E1       STA $E1
D36C   A2 05       LDX #$05        Try an element size of 5.
D36E   A5 B4       LDA $B4
D370   91 CE       STA ($CE),Y     Transfer first letter of name.
D372   10 01       BPL $D375       Decrement size to 4 if integer
D374   CA          DEX             array.
D375   C8          INY
D376   A5 B5       LDA $B5
D378   91 CE       STA ($CE),Y     Transfer second letter of name
D37A   10 02       BPL $D37E       and decrement element size to
D37C   CA          DEX             3 for strings and 2 for
D37D   CA          DEX             integers.
D37E   86 E0       STX $E0
D380   A5 26       LDA $26
D382   C8          INY
D383   C8          INY             Transfer number of subscripts
D384   C8          INY             into array header.
D385   91 CE       STA ($CE),Y
D387   A2 0B       LDX #$0B        Set default dimension size to
D389   A9 00       LDA #$00        11 (0 to 10).
D38B   24 27       BIT $27         DIM flag is clear.
D38D   50 08       BVC $D397
D38F   68          PLA             Load A and X with dimension
D390   18          CLC             size.
D391   69 01       ADC #$01
D393   AA          TAX
```

```
D394   68          PLA
D395   69 00       ADC #$00
D397   C8          INY
D398   91 CE       STA ($CE),Y     Put MSB of dimension into
D39A   C8          INY             array.
D39B   8A          TXA             Put LSB of dimension into
D39C   91 CE       STA ($CE),Y     array.
D39E   20 4D D4    JSR $D44D       Multiply element size by that
D3A1   86 E0       STX $E0         of dimension.
D3A3   85 E1       STA $E1         Save size.
D3A5   A4 91       LDY $91         Restore offset into header.
D3A7   C6 26       DEC $26         Decrement no of dim's left to
D3A9   D0 DC       BNE $D387       do. Branch if not all done.
D3AB   65 C8       ADC $C8
D3AD   B0 5D       BCS $D40C       Add array size to start
D3AF   85 C8       STA $C8         address of start of array.
D3B1   A8          TAY             Give "OUT OF MEMORY ERROR" if
D3B2   8A          TXA             too large.
D3B3   65 C7       ADC $C7
D3B5   90 03       BCC $D3BA
D3B7   C8          INY
D3B8   F0 52       BEQ $D40C
D3BA   20 44 C4    JSR $C444       Check sufficient memory.
D3BD   85 A0       STA $A0         Save top of arrays.
D3BF   84 A1       STY $A1
D3C1   A9 00       LDA #$00
D3C3   E6 E1       INC $E1         Set number of whole/part pages
D3C5   A4 E0       LDY $E0         that must be initialised to 0.
D3C7   F0 05       BEQ $D3CE       Whole number of pages to do.
D3C9   88          DEY
D3CA   91 C7       STA ($C7),Y     Clear rest of page.
D3CC   D0 FB       BNE $D3C9
D3CE   C6 C8       DEC $C8         Decrement pointers and page
D3D0   C6 E1       DEC $E1         count.
D3D2   D0 F5       BNE $D3C9       More to do.
D3D4   E6 C8       INC $C8         Pointer back to start.
D3D6   38          SEC
D3D7   A5 A0       LDA $A0         Get total array size, LSB.
D3D9   E5 CE       SBC $CE
D3DB   A0 02       LDY #$02
D3DD   91 CE       STA ($CE),Y     Save LSB of size in header.
D3DF   A5 A1       LDA $A1
D3E1   C8          INY
D3E2   E5 CF       SBC $CF         Get total array size, MSB and
D3E4   91 CE       STA ($CE),Y     save it in array header.
D3E6   A5 27       LDA $27         If DIM flag set then exit.
D3E8   D0 62       BNE $D44C
D3EA   C8          INY
D3EB   B1 CE       LDA ($CE),Y     GET ARRAY ELEMENT. Get
D3ED   85 26       STA $26         number of dimensions into $26
D3EF   A9 00       LDA #$00
D3F1   85 E0       STA $E0         Set LSB of cell number to 0.
D3F3   85 E1       STA $E1         Set MSB of cell number to 0.
D3F5   C8          INY             Point at first dimension size.
D3F6   68          PLA
D3F7   AA          TAX             Get LSB of required subscript.
D3F8   85 D3       STA $D3
D3FA   68          PLA             Get MSB of required subscript.
D3FB   85 D4       STA $D4
D3FD   D1 CE       CMP ($CE),Y     If bigger than dimensioned
D3FF   90 0E       BCC $D40F       then give "BAD SUBSCRIPT
D401   D0 06       BNE $D409       ERROR"
D403   C8          INY
D404   8A          TXA
```

```
D405   D1 CE       CMP ($CE),Y     Check LSB of subscript.
D407   90 07       BCC $D410       Continue if okay.
D409   4C 33 D3    JMP $D333       Print "BAD SUBCRIPT ERROR".
D40C   4C 7C C4    JMP $C47C       Print "OUT OF MEMORY ERROR".
D40F   C8          INY             Point Y at LSB of subscript.
D410   A5 E1       LDA $E1         If cell number so far is zero
D412   05 E0       ORA $E0         then skip the multiply.
D414   18          CLC
D415   F0 0A       BEQ $D421
D417   20 4D D4    JSR $D44D       Multiply cell number by
D41A   8A          TXA             dimension size.
D41B   65 D3       ADC $D3         Add subscript into cell
D41D   AA          TAX             number.
D41E   98          TYA
D41F   A4 91       LDY $91         Get offset into header.
D421   65 D4       ADC $D4         Add subscript to cell number
D423   86 E0       STX $E0         LSB and set new cell number.
D425   C6 26       DEC $26         Decrement and loop if more
D427   D0 CA       BNE $D3F3       dimensions to do.
D429   85 E1       STA $E1         Set final cell number MSB.
D42B   A2 05       LDX #$05        Try element size of 5.
D42D   A5 B4       LDA $B4
D42F   10 01       BPL $D432       If integer type then set size
D431   CA          DEX             to 4.
D432   A5 B5       LDA $B5
D434   10 02       BPL $D438       If string type then decrement
D436   CA          DEX             size to 3 and also set integer
D437   CA          DEX             size to 2.
D438   86 97       STX $97
D43A   A9 00       LDA #$00        Multiply final cell number by
D43C   20 56 D4    JSR $D456       element size.
D43F   8A          TXA
D440   65 C7       ADC $C7         Add start of cell's address to
D442   85 B6       STA $B6         cell offset.
D444   98          TYA
D445   65 C8       ADC $C8
D447   85 B7       STA $B7         Set A and Y to point to the
D449   A8          TAY             cell and then exit.
D44A   A5 B6       LDA $B6
D44C   60          RTS

D44D   84 91       STY $91         MULTIPLY $E0/$E1 BY DIMENSION
D44F   B1 CE       LDA ($CE),Y     SIZE. Save offset into header.
D451   85 97       STA $97         Transfer dimension size into
D453   88          DEY             $97 and $98.
D454   B1 CE       LDA ($CE),Y
D456   85 98       STA $98
D458   A9 10       LDA #$10        Set loop counter to 16 bits.
D45A   85 CC       STA $CC
D45C   A2 00       LDX #$00        Set result to 0.
D45E   A0 00       LDY #$00
D460   8A          TXA
D461   0A          ASL A           Shift result up 1 bit.
D462   AA          TAX
D463   98          TYA
D464   2A          ROL A
D465   A8          TAY
D466   B0 A4       BCS $D40C       Error if overflow.
D468   06 E0       ASL $E0         Shift current size up 1 bit.
D46A   26 E1       ROL $E1
D46C   90 0B       BCC $D479       If 0 shifted out, then skip
D46E   18          CLC             the addition.
D46F   8A          TXA
D470   65 97       ADC $97         Add dimension size to current
```

```
D472   AA          TAX             size.
D473   98          TYA
D474   65 98       ADC $98
D476   A8          TAY
D477   B0 93       BCS $D40C       Error if overflow.
D479   C6 CC       DEC $CC         Decrement loop counter.
D47B   D0 E3       BNE $D460       More loops to execute.
D47D   60          RTS

D47E   A5 28       LDA $28         FRE
D480   F0 03       BEQ $D485       If string then set up string
D482   20 D0 D7    JSR $D7D0       in main FPA.
D485   20 50 D6    JSR $D650       Attempt Garbage collection.
D488   38          SEC             Calculate LSB of Top of
D489   A5 A2       LDA $A2         Strings - Bottom of Strings.
D48B   E5 A0       SBC $A0
D48D   A8          TAY             Transfer LSB to Y.
D48E   A5 A3       LDA $A3         Calculate MSB.
D490   E5 A1       SBC $A1
D492   A2 00       LDX #$00
D494   86 28       STX $28         Clear string flag.
D496   4C 40 DF    JMP $DF40       Convert to main EPA and exit.

D499   A2 00       LDX #$00        Put signed integer from A
D49B   86 28       STX $28         (MSB) and Y into main EPA.
D49D   85 D1       STA $D1         Put A and Y into mantissa.
D49F   84 D2       STY $D2
D4A1   A2 90       LDX #$90
D4A3   4C 2C DF    JMP $DF2C       Normalise and exit.

D4A6   20 CB D8    JSR $D8CB       POS
D4A9   8A          TXA             Get single byte expression in
D4AA   F0 08       BEQ $D4B4       A and branch if zero.
D4AC   AC 58 02    LDY $0258       Get column number used for
D4AF   2C F1 02    BIT $02F1       printer.
D4B2   10 02       BPL $D4B6       Printer is off.
D4B4   A4 30       LDY $30         Load screen cursor column.
D4B6   A9 00       LDA #$00        Branch to convert integer to
D4B8   F0 DF       BEQ $D499       floating point number.

D4BA   C9 D9       CMP #$D9        DEF
D4BC   D0 21       BNE $D4DF       Token is not that of USR.
D4BE   20 E2 00    JSR $00E2       Get next character.
D4C1   A9 D4       LDA #$D4
D4C3   20 67 D0    JSR $D067       Test for "=" token.
D4C6   20 53 E8    JSR $E853       Get +ve integer into $33/$34.
D4C9   A5 33       LDA $33         Transfer jump address to jump
D4CB   A4 34       LDY $34         location.
D4CD   85 22       STA $22
D4CF   84 23       STY $23
D4D1   60          RTS             Exit.

D4D2   A6 A9       LDX $A9         CHECK FOR ILLEGAL DIRECT ERROR
D4D4   E8          INX
D4D5   D0 FA       BNE $D4D1       Not in immediate mode.
D4D7   A2 95       LDX #$95        Set X - "ILLEGAL DIRECT ERROR"
D4D9   2C A2 E5    BIT $E5A2       Set X - "UNDEF'D FUNCTION E.."
D4DC   4C 7E C4    JMP $C47E       Print error message.
D4DF   20 0D D5    JSR $D50D       Check FN and get name.
D4E2   20 D2 D4    JSR $D4D2       Check illegal direct error.
D4E5   20 62 D0    JSR $D062       Check for "(".
D4E8   A9 80       LDA #$80        Set no integer flag.
D4EA   85 2B       STA $2B
D4EC   20 88 D1    JSR $D188       Get variable.
```

```
D4EF   20 06 CF   JSR $CF06       Check that it is numeric.
D4F2   20 5F D0   JSR $D05F       Check for ")".
D4F5   A9 D4      LDA #$D4
D4F7   20 67 D0   JSR $D067       Check for "=" token.
D4FA   48         PHA
D4FB   A5 B7      LDA $B7         Save variable address.
D4FD   48         PHA
D4FE   A5 B6      LDA $B6
D500   48         PHA
D501   A5 EA      LDA $EA
D503   48         PHA             Save text pointer.
D504   A5 E9      LDA $E9
D506   48         PHA
D507   20 3C CA   JSR $CA3C       Skip rest of statement.
D50A   4C 7D D5   JMP $D57D       Set up FN description & exit.
D50D   A9 C4      LDA #$C4        Check for "FN" token.
D50F   20 67 D0   JSR $D067
D512   09 80      ORA #$80        set top bit of first char.
D514   A2 80      LDX #$80
D516   86 2B      STX $2B         Set no integer flag.
D518   20 8F D1   JSR $D18F       Get variable.
D51B   85 BD      STA $BD         Save pointer to variable.
D51D   84 BE      STY $BE
D51F   4C 06 CF   JMP $CF06       Check numeric type and exit.
D522   20 0D D5   JSR $D50D       FN Check FN and get name.
D525   A5 BE      LDA $BE         Save pointer to FN descriptor.
D527   48         PHA
D528   A5 BD      LDA $BD
D52A   48         PHA
D52B   20 59 D0   JSR $D059       Get expression in brackets.
D52E   20 06 CF   JSR $CF06       Check numeric type.
D531   68         PLA             Restore pointer to FN
D532   85 BD      STA $BD         descriptor.
D534   68         PLA
D535   85 BE      STA $BE
D537   A0 02      LDY #$02
D539   B1 BD      LDA ($BD),Y     Get parameter location LSB.
D53B   85 B6      STA $B6         Save it.
D53D   AA         TAX
D53E   C8         INY
D53F   B1 BD      LDA ($BD),Y     Get parameter location MSB.
D541   F0 97      BEQ $D4DA       Error if zero.
D543   85 B7      STA $B7         Save result.
D545   C8         INY
D546   B1 B6      LDA ($B6),Y     Save parameter value on stack.
D548   48         PHA
D549   88         DEY
D54A   10 FA      BPL $D546
D54C   A4 B7      LDY $B7         Pack FPA into parameter.
D54E   20 AD DE   JSR $DEAD
D551   A5 EA      LDA $EA
D553   48         PHA             Save text position.
D554   A5 E9      LDA $E9
D556   48         PHA
D557   B1 BD      LDA ($BD),Y     Set text pointer to start of
D559   85 E9      STA $E9         FN definition.
D55B   C8         INY
D55C   B1 BD      LDA ($BD),Y
D55E   85 EA      STA $EA
D560   A5 B7      LDA $B7         Save parameter location.
D562   48         PHA
D563   A5 B6      LDA $B6
D565   48         PHA
D566   20 03 CF   JSR $CF03       Get numeric expression.
```

```
D569   68          PLA             Restore parameter location.
D56A   85 BD       STA $BD
D56C   68          PLA
D56D   85 BE       STA $BE
D56F   20 E8 00    JSR $00E8       Next character.
D572   F0 03       BEQ $D577       End of line.
D574   4C 70 D0    JMP $D070       Print "SYNTAX ERROR".
D577   68          PLA             Restore text position.
D578   85 E9       STA $E9
D57A   68          PLA
D57B   85 EA       STA $EA
D57D   A0 00       LDY #$00
D57F   68          PLA             This section is used for two
D580   91 BD       STA ($BD),Y     purposes. One is to restore
D582   68          PLA             the FN descriptor block and
D583   C8          INY             the other is to restore a
D584   91 BD       STA ($BD),Y     variable value into the work
D586   68          PLA             floating point accumulator.
D587   C8          INY
D588   91 BD       STA ($BD),Y
D58A   68          PLA
D58B   C8          INY
D58C   91 BD       STA ($BD),Y
D58E   68          PLA
D58F   C8          INY
D590   91 BD       STA ($BD),Y
D592   60          RTS

D593   20 06 CF    JSR $CF06       STR$ Check numeric type.
D596   A0 OO       LDY #$00
D598   20 D7 E0    JSR $E0D7       Convert to string.
D59B   68          PLA
D59C   68          PLA             Restore return address.
D59D   A9 FF       LDA #$FF
D59F   A0 00       LDY #$00        Set up string and then exit.
D5A1   F0 12       BEQ $D5B5
D5A3   A6 D3       LDX $D3         SET UP MEM STRING SLOT & BLOCK
D5A5   A4 D4       LDY $D4         Copy pointer to string.
D5A7   86 BF       STX $BF
D5A9   84 C0       STY $C0
D5AB   20 1E D6    JSR $D61E       Get slot in memory or string.
D5AE   86 D1       STX $D1         Store data in block, LSB and
D5B0   84 D2       STY $D2         MSB of pointer to string and
D5B2   85 D0       STA $D0         then its length.
D5B4   60          RTS

D5B5   A2 22       LDX #$22        GET STRING AFTER ".
D5B7   86 24       STX $24         Set $24 and $25 to ASCII value
D5B9   86 25       STX $25         of double quote.
D5BB   85 DE       STA $DE         Set up pointer to start of
D5BD   84 DF       STY $DF         string.
D5BF   85 D1       STA $D1         Set up pointer to start of
D5C1   84 D2       STY $D2         string.
D5C3   A0 FF       LDY #$FF        Set initial loop value.
D5C5   C8          INY
D5C6   B1 DE       LDA ($DE),Y     Loop until null found.
D5C8   F0 0C       BEQ $D5D6
D5CA   C5 24       CMP $24         Or there is a match with
D5CC   F0 04       BEQ $D5D2       content of $24.
D5CE   C5 25       CMP $25
D5D0   D0 F3       BNE $D5C5
D5D2   C9 22       CMP #$22        Set C if terminated by a ".
D5D4   F0 01       BEQ $D5D7
D5D6   18          CLC
```

```
D5D7    84 D0       STY $D0         Save string length.
D5D9    98          TYA
D5DA    65 DE       ADC $DE         Calculate end address of
D5DC    85 E0       STA $E0         string in $E0/$E1.
D5DE    A6 DF       LDX $DF
D5E0    90 01       BCC $D5E3
D5E2    E8          INX
D5E3    86 E1       STX $E1
D5E5    A5 DF       LDA $DF         If string is not in page 0
D5E7    D0 0B       BNE $D5F4       then push block on stack.
D5E9    98          TYA
D5EA    20 A3 D5    JSR $D5A3       Set up new slot and block.
D5ED    A6 DE       LDX $DE         Get start of string.
D5EF    A4 DF       LDY $DF
D5F1    20 B2 D7    JSR $D7B2       Transfer string to new slot.
D5F4    A6 85       LDX $85         Routine to push string block
D5F6    E0 91       CPX #$91        on string stack.
D5F8    D0 05       BNE $D5FF
D5FA    A2 C4       LDX #$C4        If stack full then print
D5FC    4C 7E C4    JMP $C47E       "FORMULA TOO COMPLEX".
D5FF    A5 D0       LDA $D0         Transfer string block on to
D601    95 00       STA $00,X       string stack (between $88 and
D603    A5 D1       LDA $D1         $90 inclusive).
D605    95 01       STA $01,X
D607    A5 D2       LDA $D2
D609    95 02       STA $02,X
D60B    A0 OO       LDY #$00
D60D    86 D3       STX $D3         Set $D3/$D4 to point to it.
D60F    84 D4       STY $D4
D611    84 DF       STY $DF         Clear rounding byte.
D613    88          DEY
D614    84 28       STY $28         Set string type flag.
D616    86 86       STX $86         Set address of string block.
D618    E8          INX             Set string stack pointer to
D619    E8          INX             next available space.
D61A    E8          INX
D61B    86 85       STX $85         Save string stack pointer.
D61D    60          RTS

D61E    46 2A       LSR $2A         Routine to get slot for string
D620    48          PHA
D621    49 FF       EOR #$FF
D623    38          SEC             Set A, Y to bottom of string
D624    65 A2       ADC $A2         area - length of string.
D626    A4 A3       LDY $A3
D628    B0 01       BCS $D62B
D62A    88          DEY
D62B    C4 A1       CPY $A1
D62D    90 11       BCC $D640       Attempt garbage collection if
D62F    D0 04       BNE $D635       start of string would be below
D631    C5 A0       CMP $A0         end of Arrays.
D633    90 0B       BCC $D640
D635    85 A2       STA $A2         Set new bottom of strings
D637    84 A3       STY $A3         pointer.
D639    85 A4       STA $A4         Set address for string to be
D63B    84 A5       STY $A5         inserted.
D63D    AA          TAX             Save LSB of address in X.
D63E    68          PLA             Restore string length.
D63F    60          RTS

D640    A2 4D       LDX #$4D        Prepare error message pointer.
D642    A5 2A       LDA $2A         Print "OUT OF MEMORY ERROR" if
D644    30 B6       BMI $D5FC       garbage collection already tried.
D646    20 50 D6    JSR $D650       Garbage collection.
```

```
D649    A9 80       LDA #$80        Set flag to indicate garbage
D64B    85 2A       STA $2A         collection has been done.
D64D    68          PLA
D64E    D0 D0       BNE $D620       Try again.

D650    A6 A6       LDX $A6         GARBAGE COLLECTION.
D652    A5 A7       LDA $A7         Update last string allocated,
D654    86 A2       STX $A2         initially set to HIMEM.
D656    85 A3       STA $A3
D658    A0 00       LDY #$00        Clear pointer.
D65A    84 BE       STY $BE
D65C    84 BD       STY $BD
D65E    A5 A0       LDA $A0         Copy end of Arrays pointer.
D660    A6 A1       LDX $A1
D662    85 CE       STA $CE
D664    86 CF       STX $CF
D666    A9 88       LDA #$88
D668    A2 00       LDX #$00
D66A    85 91       STA $91         Pointer set to string stack
D66C    86 92       STX $92         base.
D66E    C5 85       CMP $85         Set $91,$92 to point to non -
D670    F0 05       BEQ $D677       collected string at top of
D672    20 F1 D6    JSR $D6F1       stack.
D675    F0 F7       BEQ $D66E       Branch always.
D677    A9 07       LDA #$07        Set string variable size.
D679    85 C2       STA $C2
D67B    A5 9C       LDA $9C         Copy End Basic and set current
D67D    A6 9D       LDX $9D         variable position in A,X.
D67F    85 91       STA $91
D681    86 92       STX $92
D683    E4 9F       CPX $9F         Compare end variable with
D685    D0 04       BNE $D68B       current variable position.
D687    C5 9E       CMP $9E
D689    F0 05       BEQ $D690       Pointers are equal.
D68B    20 E7 D6    JSR $D6E7       Set string pointer to next non
D68E    F0 F3       BEQ $D683       collected variable
D690    85 C7       STA $C7         Set current variable position.
D692    86 C8       STX $C8
D694    A9 03       LDA #$03        Set element size for string
D696    85 C2       STA $C2         arrays.
D698    A5 C7       LDA $C7         Compare end of Arrays with
D69A    A6 C8       LDX $C8         current pointer.
D69C    E4 A1       CPX $A1
D69E    D0 07       BNE $D6A7
D6A0    C5 A0       CMP $A0
D6A2    D0 03       BNE $D6A7
D6A4    4C 30 D7    JMP $D730

D6A7    85 91       STA $91         Save pointer and find next
D6A9    86 92       STX $92         array.
D6AB    A0 00       LDY #$00
D6AD    B1 91       LDA ($91),Y     Skip through array header
D6AF    AA          TAX             saving array type on the way.
D6B0    C8          INY
D6B1    B1 91       LDA ($91),Y
D6B3    08          PHP
D6B4    C8          INY
D6B5    B1 91       LDA ($91),Y     Add LSB of offset in array
D6B7    65 C7       ADC $C7         header to point to next one.
D6B9    85 C7       STA $C7
D6BB    C8          INY
D6BC    B1 91       LDA ($91),Y     Add MSB of offset in array
D6BE    65 C8       ADC $C8         header to point to next one.
D6C0    85 C8       STA $C8
```

```
D6C2    28          PLP
D6C3    10 D3       BPL $D698       Test bit 7 of each of the
D6C5    8A          TXA             array name letters and branch
D6C6    30 D0       BMI $D698       back if array is not string
D6C8    C8          INY             type.
D6C9    B1 91       LDA ($91),Y
D6CB    A0 00       LDY #$00        Advance the pointer beyond
D6CD    0A          ASL A           the array header and
D6CE    69 05       ADC #$05        dimension specifiers to the
D6D0    65 91       ADC $91         first string array element.
D6D2    85 91       STA $91
D6D4    90 02       BCC $D6D8
D6D6    E6 92       INC $92
D6D8    A6 92       LDX $92
D6DA    E4 C8       CPX $C8         Go through the elements of the
D6DC    D0 04       BNE $D6E2       array until top one found.
D6DE    C5 C7       CMP $C7
D6E0    F0 BA       BEQ $D69C
D6E2    20 F1 D6    JSR $D6F1
D6E5    F0 F3       BEQ $D6DA       Branch always.
D6E7    B1 91       LDA ($91),Y     Test if variable is a string.
D6E9    30 35       BMI $D720       If it is then test whether it
D6EB    C8          INY             has been collected or not.
D6EC    B1 91       LDA ($91),Y
D6EE    10 30       BPL $D720
D6F0    C8          INY
D6F1    B1 91       LDA ($91),Y
D6F3    F0 2B       BEQ $D720       String is null.
D6F5    C8          INY
D6F6    B1 91       LDA ($91),Y
D6F8    AA          TAX
D6F9    C8          INY
D6FA    B1 91       LDA ($91),Y     Branch if string address is
D6FC    C5 A3       CMP $A3         above current bottom of
D6FE    90 06       BCC $D706       strings pointer. Test MSB
D700    D0 1E       BNE $D720       first and then LSB if the MSBs
D702    E4 A2       CPX $A2         are equal.
D704    B0 1A       BCS $D720
D706    C5 CF       CMP $CF         Branch if string address is
D708    90 16       BCC $D720       below end of arrays, i.e. it
D70A    D0 04       BNE $D710       is a string constant in a
D70C    E4 CE       CPX $CE         program.
D70E    90 10       BCC $D720
D710    86 CE       STX $CE         Save pointer to string ready
D712    85 CF       STA $CF         for transfer.
D714    A5 91       LDA $91         Save current string pointer.
D716    A6 92       LDX $92
D718    85 BD       STA $BD
D71A    86 BE       STX $BE
D71C    A5 C2       LDA $C2         Copy string block size.
D71E    85 C4       STA $C4
D720    A5 C2       LDA $C2         Set $91,$92 to point to next
D722    18          CLC             variable.
D723    65 91       ADC $91
D725    85 91       STA $91
D727    90 02       BCC $D72B
D729    E6 92       INC $92
D72B    A6 92       LDX $92
D72D    A0 00       LDY #$00        Set Z and return with A,X
D72F    60          RTS             holding current position.

D730    A5 BE       LDA $BE         COPY ACROSS STRING.
D732    05 BD       ORA $BD
D734    F0 F5       BEQ $D72B
```

```
D736    A5 C4       LDA $C4
D738    29 04       AND #$04
D73A    4A          LSR A
D73B    A8          TAY
D73C    85 C4       STA $C4         Calculate end address of the
D73E    B1 BD       LDA ($BD),Y     string and set pointers for
D740    65 CE       ADC $CE         block transfer.
D742    85 C9       STA $C9
D744    A5 CF       LDA $CF         $CE,$CF - start of data.
D746    69 00       ADC #$00        $C9,$CA - end of data.
D748    85 CA       STA $CA         $C7,$C8 - new end of data.
D74A    A5 A2       LDA $A2
D74C    A6 A3       LDX $A3
D74E    85 C7       STA $C7
D750    86 C8       STX $C8
D752    20 FB C3    JSR $C3FB       Block transfer to copy across
D755    A4 C4       LDY $C4         string.
D757    C8          INY
D758    A5 C7       LDA $C7
D75A    91 BD       STA ($BD),Y
D75C    AA          TAX             Transfer pointer into
D75D    E6 C8       INC $C8         memory.
D75F    A5 C8       LDA $C8
D761    C8          INY
D762    91 BD       STA ($BD),Y
D764    4C 54 D6    JMP $D654

D767    A5 D4       LDA $D4         STRING CONCATENATION.
D769    48          PHA
D76A    A5 D3       LDA $D3         Save pointer to string.
D76C    48          PHA
D76D    20 00 D0    JSR $D000       Get item.
D770    20 08 CF    JSR $CF08       Check it is string type.
D773    68          PLA
D774    85 DE       STA $DE         Restore pointer to first
D776    68          PLA             string.
D777    85 DF       STA $DF
D779    A0 00       LDY #$00
D77B    B1 DE       LDA ($DE),Y     Add string lengths.
D77D    18          CLC
D77E    71 D3       ADC ($D3),Y
D780    90 05       BCC $D787
D782    A2 B5       LDX #$B5        Give error if strings are too
D784    4C 7E C4    JMP $C47E       long.

D787    20 A3 D5    JSR $D5A3       Set up slot for new string.
D78A    20 A4 D7    JSR $D7A4       Transfer first string into
D78D    A5 BF       LDA $BF         slot.
D78F    A4 C0       LDY $C0
D791    20 D4 D7    JSR $D7D4       Set up string, releasing if
D794    20 B6 D7    JSR $D7B6       necessary and transfer.
D797    A5 DE       LDA $DE
D799    A4 DF       LDY $DF
D79B    20 D4 D7    JSR $D7D4       Set up string, releasing if
D79E    20 F4 D5    JSR $D5F4       necessary & push string block
D7A1    4C 31 CF    JMP $CF31       on stack. Go back for more.

D7A4    A0 00       LDY #$00        This routine transfers the
D7A6    B1 DE       LDA ($DE),Y     block pointed to by $DE into
D7A8    48          PHA             slot.
D7A9    C8          INY
D7AA    B1 DE       LDA ($DE),Y
D7AC    AA          TAX             X holds LSB and Y holds MSB of
D7AD    C8          INY             pointer.
```

```
D7AE   B1 DE      LDA ($DE),Y
D7B0   A8         TAY
D7B1   68         PLA             Restore length.
D7B2   86 91      STX $91         Set up pointer to string.
D7B4   84 92      STY $92
D7B6   A8         TAY             Y holds length and skip
D7B7   F0 0A      BEQ $D7C3       transfer if null.
D7B9   48         PHA
D7BA   88         DEY
D7BB   B1 91      LDA ($91),Y     Transfer the characters of the
D7BD   91 A4      STA ($A4),Y     string.
D7BF   98         TYA
D7C0   D0 F8      BNE $D7BA
D7C2   68         PLA             restore length.
D7C3   18         CLC
D7C4   65 A4      ADC $A4
D7C6   85 A4      STA $A4         Add length to content of $A4,
D7C8   90 02      BCC $D7CC       $A5 ready for next string.
D7CA   E6 A5      INC $A5
D7CC   60         RTS

D7CD   20 08 CF   JSR $CF08       Check string type.
D7D0   A5 D3      LDA $D3
D7D2   A4 D4      LDY $D4         Set pointer to string block.
D7D4   85 91      STA $91
D7D6   84 92      STY $92
D7D8   20 05 D8   JSR $D805       Release string stack.
D7DB   08         PHP
D7DC   A0 00      LDY #$00
D7DE   B1 91      LDA ($91),Y     Save length.
D7E0   48         PHA
D7E1   C8         INY
D7E2   B1 91      LDA ($91),Y     Get LSB of pointer into X.
D7E4   AA         TAX
D7E5   C8         INY
D7E6   B1 91      LDA ($91),Y     Get MSB of pointer into Y.
D7E8   A8         TAY
D7E9   68         PLA
D7EA   28         PLP             If not from string stack then
D7EB   D0 13      BNE $D800       set pointer and exit.
D7ED   C4 A3      CPY $A3         If not bottom of strings then
D7EF   D0 0F      BNE $D800       set pointer and exit.
D7F1   E4 A2      CPX $A2
D7F3   D0 0B      BNE $D800
D7F5   48         PHA             Save length.
D7F6   18         CLC
D7F7   65 A2      ADC $A2         Move up bottom of strings to
D7F9   85 A2      STA $A2         remove temporary string.
D7FB   90 02      BCC $D7FF
D7FD   E6 A3      INC $A3
D7FF   68         PLA             Restore length.
D800   86 91      STX $91         Set pointer to string and
D802   84 92      STY $92         exit.
D804   60         RTS

D805   C4 87      CPY $87         Release string stack item if
D807   D0 0C      BNE $D815       necessary.
D809   C5 86      CMP $86
D80B   D0 08      BNE $D815
D80D   85 85      STA $85
D80F   E9 03      SBC #$03
D811   85 86      STA $86
D813   A0 00      LDY #$00        Z set if released, clear if
D815   60         RTS             not.
```

```
D816   20 CB D8   JSR $D8CB       CHR$ Get single byte numeric
D819   8A         TXA            expression, save it on stack.
D81A   48         PHA
D81B   A9 01      LDA #$01
D81D   20 AB D5   JSR $D5AB      Get slot for string & save
D820   68         PLA            pointer. Restore expression.
D821   A0 00      LDY #$00
D823   91 D1      STA ($D1),Y    Save expression in memory.
D825   68         PLA            Remove address of calling
D826   68         PLA            routine.
D827   4C F4 D5   JMP $D5F4      Push string block on stack.

D82A   20 8B D8   JSR $D88B      LEFT$ Set up argument.
D82D   D1 BF      CMP ($BF),Y
D82F   98         TYA            Clear A. If length of string
D830   90 04      BCC $D836      is less than slice size then
D832   B1 BF      LDA ($BF),Y    set slice size to string
D834   AA         TAX            length.
D835   98         TYA            Save value to be added to
D836   48         PHA            string pointer after setting
D837   8A         TXA            up string.
D838   48         PHA
D839   20 AB D5   JSR $D5AB      Get slot for string and save
D83C   A5 BF      LDA $BF        block.
D83E   A4 C0      LDY $C0        Set A, Y
D840   20 D4 D7   JSR $D7D4      Set up string, releasing if
D843   68         PLA            necessary.
D844   A8         TAY
D845   68         PLA
D846   18         CLC
D847   65 91      ADC $91        Add to the string pointer the
D849   85 91      STA $91        size of new string.
D84B   90 02      BCC $D84F
D84D   E6 92      INC $92
D84F   98         TYA
D850   20 B6 D7   JSR $D7B6      Transfer string ptr to $A4,$A5
D853   4C F4 D5   JMP $D5F4      Push $ block on $ stack.

D856   20 8B D8   JSR $D88B      RIGHT$
D859   18         CLC            Subtract the slice size from
D85A   F1 BF      SBC ($BF),Y    the length of the string.
D85C   49 FF      EOR #$FF
D85E   4C 30 D8   JMP $D830      Rest is same as LEFT$.

D861   A9 FF      LDA #$FF       MID$
D863   85 D4      STA $D4
D865   20 E8 00   JSR $00E8      Get next char.
D868   C9 29      CMP #$29
D86A   F0 06      BEQ $D872      Found a ")".
D86C   20 65 D0   JSR $D065      Test for comma.
D86F   20 C8 D8   JSR $D8C8      Get 1 byte numeric expression.
D872   20 8B D8   JSR $D88B      Set up arguments.
D875   F0 4B      BEQ $D8C2      Error in value.
D877   CA         DEX
D878   8A         TXA
D879   48         PHA
D87A   18         CLC
D87B   A2 00      LDX #$00       Chop the string as in RIGHT$
D87D   F1 BF      SBC ($BF),Y    and then branch to LEFT$
D87F   B0 B6      BCS $D837      routine to chop the left side
D881   49 FF      EOR #$FF       of the string.
D883   C5 D4      CMP $D4
D885   90 B1      BCC $D838
```

```
D887   A5 D4       LDA $D4
D889   B0 AD       BCS $D838       Rest same as LEFT$.
D88B   20 5F D0    JSR $D05F       Check for "(".
D88E   68          PLA             Save call address.
D88F   A8          TAY
D890   68          PLA
D891   85 C4       STA $C4
D893   68          PLA             Remove call address from
D894   68          PLA             expression evaluator.
D895   68          PLA             Put magnitude of string slice
D896   AA          TAX             into X.
D897   68          PLA             Pull and store pointer to
D898   85 BF       STA $BF         string.
D89A   68          PLA
D89B   85 C0       STA $C0
D89D   A5 C4       LDA $C4         Restore call address.
D89F   48          PHA
D8A0   98          TYA
D8A1   48          PHA
D8A2   A0 00       LDY #$00        Clear Y.
D8A4   8A          TXA             Set status register to size
D8A5   60          RTS             of string slice.

D8A6   20 AC D8    JSR $D8AC       LEN Do check and convert
D8A9   4C B6 D4    JMP $D4B6       length to floating point no.
D8AC   20 CD D7    JSR $D7CD       Check string type.
D8AF   A2 00       LDX #$00        Clear string flag.
D8B1   86 28       STX $28
D8B3   A8          TAY
D8B4   60          RTS

D8B5   20 AC D8    JSR $D8AC       ASC Get string.
D8B8   F0 08       BEQ $D8C2       Error if empty.
D8BA   A0 00       LDY #$00
D8BC   B1 91       LDA ($91),Y     Get first character of string.
D8BE   A8          TAY             Put into Y.
D8BF   4C B6 D4    JMP $D4B6       Convert code to FPA and exit.

D8C2   4C 36 D3    JMP $D336       Print "ILLEGAL QUANTITY ERROR"

D8C5   20 E2 00    JSR $00E2       GET SINGLE BYTE EXPRESSION.
D8C8   20 03 CF    JSR $CF03       Get next char and evaluate
D8CB   20 A2 D2    JSR $D2A2       expression, convert to +ve
D8CE   A6 D3       LDX $D3         integer.
D8D0   D0 F0       BNE $D8C2       Error if too large.
D8D2   A6 D4       LDX $D4         Exit with byte in X.
D8D4   4C E8 00    JMP $00E8
D8D7   20 AC D8    JSR $D8AC       VAL Set up string.
D8DA   D0 03       BNE $D8DF
D8DC   4C B2 DB    JMP $DBB2       If empty then use 0.
D8DF   A6 E9       LDX $E9
D8E1   A4 EA       LDY $EA
D8E3   86 E0       STX $E0         Copy text pointer.
D8E5   84 E1       STY $E1
D8E7   A6 91       LDX $91         Copy content of $91,$92 into
D8E9   86 E9       STX $E9         $E9,$EA.
D8EB   18          CLC
D8EC   65 91       ADC $91         Add A to $91,$92 and place
D8EE   85 93       STA $93         result in $93,$94.
D8F0   A6 92       LDX $92
D8F2   86 EA       STX $EA
D8F4   90 01       BCC $D8F7
D8F6   E8          INX
D8F7   86 94       STX $94
```

```
D8F9    A0 00       LDY #$00
D8FB    B1 93       LDA ($93),Y     Get character from string.
D8FD    48          PHA
D8FE    A9 00       LDA #$00
D900    91 93       STA ($93),Y
D902    20 E8 00    JSR $00E8       Get next char.
D905    20 E7 DF    JSR $DFE7       Get number.
D908    68          PLA
D909    A0 00       LDY #$00
D90B    91 93       STA ($93),Y
D90D    A6 E0       LDX $E0         Restore text pointer.
D90F    A4 E1       LDY $E1
D911    86 E9       STX $E9
D913    84 EA       STY $EA
D915    60          RTS

D916    20 03 CF    JSR $CF03       Evaluate expression and
D919    20 22 D9    JSR $D922       convert to integer.
D91C    20 65 D0    JSR $D065       Check for comma and get single
D91F    4C C8 D8    JMP $D8C8       byte numeric expression.

D922    A5 D5       LDA $D5         CONVERT MAIN FPA TO INTEGER.
D924    30 9C       BMI $D8C2       Error if negative number.
D926    A5 D0       LDA $D0
D928    C9 91       CMP #$91        Error if number over 32768.
D92A    B0 96       BCS $D8C2
D92C    20 8C DF    JSR $DF8C       Convert main FPA to integer.
D92F    A5 D3       LDA $D3         Put result in $33,$34.
D931    A4 D4       LDY $D4
D933    84 33       STY $33
D935    85 34       STA $34
D937    60          RTS             Exit.

D938    A5 34       LDA $34         PEEK
D93A    48          PHA             Save $33,$34 on stack.
D93B    A5 33       LDA $33
D93D    48          PHA
D93E    20 22 D9    JSR $D922       Convert main FPA to integer.
D941    A0 00       LDY #$00
D943    B1 33       LDA ($33),Y     Load byte from memory.
D945    A8          TAY             Transfer to Y.
D946    68          PLA             Restore $33,$34.
D947    85 33       STA $33
D949    68          PLA
D94A    85 34       STA $34
D94C    4C B6 D4    JMP $D4B6       Convert Y to FPA and exit.

D94F    20 16 D9    JSR $D916       POKE Get expression.
D952    8A          TXA
D953    A0 00       LDY #$00
D955    91 33       STA ($33),Y     Store byte in memory.
D957    60          RTS             Exit

D958    20 03 CF    JSR $CF03       WAIT Evaluate expression
D95B    20 22 D9    JSR $D922       and convert to integer.
D95E    A4 33       LDY $33         Load value into X,Y.
D960    A6 34       LDX $34
D962    A9 02       LDA #$02        Set spare counter and wait
D964    4C C9 EE    JMP $EEC9       until count down to zero.

D967    20 53 E8    JSR $E853       DOKE
D96A    A5 33       LDA $33         Get integer argument and save
D96C    A4 34       LDY $34         it in $1D,$1E.
D96E    85 1D       STA $1D
```

```
D970    84 1E       STY $1E
D972    20 65 D0    JSR $D065       Test for comma.
D975    20 53 E8    JSR $E853       Get integer argument.
D978    A0 01       LDY #$01
D97A    B9 33 00    LDA $0033,Y
D97D    91 1D       STA ($1D),Y     Put value into memory.
D97F    88          DEY
D980    10 F8       BPL $D97A
D982    60          RTS             Exit.

D983    20 22 D9    JSR $D922       DEEK Convert main FPA into
D986    A0 01       LDY #$01        integer.
D988    B1 33       LDA ($33),Y     Get bytes from memory into A
D98A    48          PHA             and Y.
D98B    88          DEY
D98C    B1 33       LDA ($33),Y
D98E    A8          TAY
D98F    68          PLA             Convert A, Y into floating
D990    4C 40 DF    JMP $DF40       point number and exit.

D993    48          PHA             CONVERT BYTE TO 2 HEX DIGITS.
D994    4A          LSR A
D995    4A          LSR A           Shift left hand nibble into
D996    4A          LSR A           right hand nibble.
D997    4A          LSR A
D998    20 9C D9    JSR $D99C       Convert L.H. nibble to char.
D99B    68          PLA             Restore original byte.
D99C    29 0F       AND #$0F        Isolate R.H nibble.
D99E    09 30       ORA #$30        Convert to ASCII.
D9A0    C9 3A       CMP #$3A
D9A2    90 02       BCC $D9A6
D9A4    69 06       ADC #$06
D9A6    C9 30       CMP #$30
D9A8    D0 04       BNE $D9AE       Digit is non zero.
D9AA    A4 2F       LDY $2F
D9AC    F0 06       BEQ $D9B4       Exit if char is leading zero.
D9AE    85 2F       STA $2F
D9B0    9D 00 01    STA $0100,X     Put char in bottom of stack
D9B3    E8          INX             page. Advance pointer.
D9B4    60          RTS

D9B5    20 22 D9    JSR $D922       HEX$ Convert FPA to positive
D9B8    A2 00       LDX #$00        integer.
D9BA    86 2F       STX $2F         Set leading zero flag.
D9BC    A9 23       LDA #$23        Set # at front of number to
D9BE    85 FF       STA $FF         indicate hexadecimal.
D9C0    A5 34       LDA $34         Convert upper byte to 2 hex
D9C2    20 93 D9    JSR $D993       digits.
D9C5    A5 33       LDA $33         Convert lower byte to 2 hex
D9C7    20 93 D9    JSR $D993       digits.
D9CA    8A          TXA
D9CB    D0 06       BNE $D9D3       Number is non zero.
D9CD    A9 30       LDA #$30        Put in single 0 for zero
D9CF    9D 00 01    STA $0100,X     numbers.
D9D2    E8          INX             Advance pointer.
D9D3    A9 00       LDA #$00        Put null at end of string.
D9D5    9D 00 01    STA $0100,X
D9D8    4C 9B D5    JMP $D59B       Point to string and exit.

D9DB    4C 70 D0    JMP $D070       Print "SYNTAX ERROR".

D9DE    20 21 EC    JSR $EC21       LORES Set screen to text.
D9E1    20 C8 D8    JSR $D8C8       Get single byte expression.
D9E4    8A          TXA
```

```
D9E5   F0 06      BEQ $D9ED       In LORES 0.
D9E7   CA         DEX
D9E8   D0 F1      BNE $D9DB       Error if not LORES 1.
D9EA   A9 09      LDA #$09
D9EC   2C A9 08   BIT $08A9       Hides LDA #$08.
D9EF   A2 10      LDX #$10        Set paper to black in temp
D9F1   8E F8 02   STX $02F8       location.
D9F4   A2 1B      LDX #$1B        Set row counter.
D9F6   48         PHA
D9F7   8A         TXA             Calculate start address of Xth
D9F8   20 0C DA   JSR $DA0C       row of screen.
D9FB   AD F8 02   LDA $02F8
D9FE   A0 27      LDY #$27
DA00   91 1F      STA ($1F),Y     Write paper colour on every
DA02   88         DEY             column of row except first.
DA03   D0 FB      BNE $DA00
DA05   68         PLA
DA06   91 1F      STA ($1F),Y     Write char set type for row.
DA08   CA         DEX             Repeat until all rows are done
DA09   D0 EB      BNE $D9F6       except status line.
DA0B   60         RTS

DA0C   20 31 F7   JSR $F731       CALCULATE START ADDRESS OF Nth
DA0F   84 20      STY $20         ROW ON SCREEN.
DA11   18         CLC             Multiply A by 40 with Y
DA12   69 80      ADC #$80        holding overflow beyond 8 bits
DA14   48         PHA
DA15   85 1F      STA $1F         Add in start address of screen
DA17   A9 BB      LDA #$BB        and put result in $1F,$20.
DA19   65 20      ADC $20
DA1B   85 20      STA $20
DA1D   68         PLA
DA1E   60         RTS

DA1F   4C C2 D8   JMP $D8C2       Print "ILLEGAL QUANTITY ERR.".

DA22   20 F6 DA   JSR $DAF6       Test for text screen.
DA25   20 C8 D8   JSR $D8C8       Get single byte expression.
DA28   E0 28      CPX #$28        Error if column number is too
DA2A   B0 F3      BCS $DA1F       large.
DA2C   8E F8 02   STX $02F8       Save result.
DA2F   20 65 D0   JSR $D065       Test for comma.
DA32   20 C8 D8   JSR $D8C8       Get single byte expression.
DA35   E0 1B      CPX #$1B        Error if row number is too
DA37   B0 E6      BCS $DA1F       large.
DA39   E8         INX             Increment row number.
DA3A   8A         TXA             Transfer to A and calculate
DA3B   20 0C DA   JSR $DA0C       address of start of that row.
DA3E   60         RTS             Exit.

DA3F   20 62 D0   JSR $D062       SCRN Test for "(".
DA42   20 22 DA   JSR $DA22       Get X, Y co-ordinates.
DA45   20 5F D0   JSR $D05F       Test for comma.
DA48   AC F8 02   LDY $02F8
DA4B   B1 1F      LDA ($1F),Y     Get character from screen
DA4D   A8         TAY             Transfer result to floating
DA4E   4C B6 D4   JMP $D4B6       point in main FPA.

DA51   20 22 DA   JSR $DA22       PLOT Get X, Y co-ordinates.
DA54   20 65 D0   JSR $D065       Test for comma.
DA57   20 17 CF   JSR $CF17       Evaluate expression.
DA5A   24 28      BIT $28
DA5C   10 1D      BPL $DA7B       Expression not string type.
DA5E   20 D0 D7   JSR $D7D0       Set up string in FPA.
```

```
DA61    AA          TAX
DA62    18          CLC
DA63    AD F8 02    LDA $02F8       Calculate start address for
DA66    65 1F       ADC $1F         writing string to screen.
DA68    90 02       BCC $DA6C
DA6A    E6 20       INC $20
DA6C    85 1F       STA $1F
DA6E    A0 OO       LDY #$00
DA70    E8          INX
DA71    CA          DEX
DA72    F0 10       BEQ $DA84       String plotted.
DA74    B1 91       LDA ($91),Y     Write each element to screen.
DA76    91 1F       STA ($1F),Y
DA78    C8          INY
DA79    D0 F6       BNE $DA71       More to be done.
DA7B    20 CB D8    JSR $D8CB       Get single byte expression.
DA7E    8A          TXA
DA7F    AC F8 02    LDY $02F8
DA82    91 1F       STA ($1F),Y     Print it to screen.
DA84    60          RTS             Exit.

DA85    D0 17       BNE $DA9E
DA87    A9 03       LDA #$03        Check for 6 free bytes on the
DA89    20 37 C4    JSR $C437       stack.
DA8C    A5 EA       LDA $EA         Save the program position, the
DA8E    48          PHA             current line number and the
DA8F    A5 E9       LDA $E9         REPEAT token on the stack for
DA91    48          PHA             next loop.
DA92    A5 A9       LDA $A9
DA94    48          PHA
DA95    A5 A8       LDA $A8
DA97    48          PHA
DA98    A9 8B       LDA #$8B
DA9A    48          PHA
DA9B    4C C1 C8    JMP $C8C1

DA9E    4C 70 D0    JMP $D070       Print "SYNTAX ERROR".

DAA1    A9 FF       LDA #$FF        PULL / UNTIL
DAA3    85 B9       STA $B9
DAA5    20 C6 C3    JSR $C3C6       Pull data off stack.
DAA8    9A          TXS
DAA9    C9 8B       CMP #$8B
DAAB    F0 05       BEQ $DAB2       REPEAT token found.
DAAD    A2 F5       LDX #$F5        Print "BAD UNTIL ERROR" if
DAAF    4C 7E C4    JMP $C47E       token not found.
DAB2    C0 10       CPY #$10
DAB4    D0 05       BNE $DABB       PULL token not found.
DAB6    84 D0       STY $D0
DAB8    98          TYA
DAB9    D0 06       BNE $DAC1
DABB    20 E8 00    JSR $00E8       Get next char.
DABE    20 17 CF    JSR $CF17       Evaluate expression.
DAC1    68          PLA
DAC2    A5 D0       LDA $D0         Go back to start of loop if
DAC4    F0 05       BEQ $DACB       condition still false.
DAC6    68          PLA             Pull loop data off stack and
DAC7    68          PLA             forget it. Used to exit the
DAC8    68          PLA             loop.
DAC9    68          PLA
DACA    60          RTS

DACB    68          PLA             Pull old program line number
DACC    85 A8       STA $A8         and position in text from the
```

```
DACE    68          PLA             stack.
DACF    85 A9       STA $A9
DAD1    68          PLA
DAD2    85 E9       STA $E9
DAD4    68          PLA
DAD5    85 EA       STA $EA
DAD7    4C 8C DA    JMP $DA8C

DADA    20 78 EB    JSR $EB78       KEY$ Get next char from
DADD    08          PHP             keyboard.
DADE    48          PHA
DADF    10 03       BPL $DAE4       Key is not valid.
DAE1    A9 01       LDA #$01        Set string length for 1 key.
DAE3    2C A9 00    BIT $00A9       Hides string length for 0 keys
DAE6    20 AB D5    JSR $D5AB       Get slot for string.
DAE9    68          PLA
DAEA    28          PLP
DAEB    10 04       BPL $DAF1       No valid key was obtained.
DAED    A0 00       LDY #$00
DAEF    91 D1       STA ($D1),Y     Save key in string.
DAF1    68          PLA
DAF2    68          PLA             Push string block on to the
DAF3    4C F4 D5    JMP $D5F4       string stack.

DAF6    AD C0 02    LDA $02C0       TEST FOR TEXT SCREEN.
DAF9    29 01       AND #$01
DAFB    F0 05       BEQ $DB02       Print "DISP TYPE MISMATCH
DAFD    A2 A3       LDX #$A3        ERROR" if wrong screen mode.
DAFF    4C 7E C4    JMP $C47E
DB02    60          RTS
DB03    60          RTS

DB04    A9 05       LDA #$05        Set A, Y to point to floating
DB06    A0 E2       LDY #$E2        point value for 0.5 and jump
DB08    4C 22 DB    JMP $DB22       to "+" routine.

DB0B    20 51 DD    JSR $DD51       Unpack work FPA.
DB0E    A5 D5       LDA $D5         - OPERATOR
DB10    49 FF       EOR #$FF        Invert sign of main FPA.
DB12    85 D5       STA $D5
DB14    45 DD       EOR $DD
DB16    85 DE       STA $DE         Set sign difference flag.
DB18    A5 D0       LDA $D0
DB1A    4C 25 DB    JMP $DB25       Jump to addition routine.

DB1D    20 54 DC    JSR $DC54       Shift number by required amount
DB20    90 3C       BCC $DB5E       and continue with same sign.
DB22    20 51 DD    JSR $DD51       Unpack work FPA.
DB25    D0 03       BNE $DB2A       + OPERATOR.
DB27    4C D5 DE    JMP $DED5       If main FPA is 0 then copy
DB2A    A6 DF       LDX $DF         work FPA into main FPA.
DB2C    86 C5       STX $C5         Save rounding byte.
DB2E    A2 D8       LDX #$D8        Point to work FPA.
DB30    A5 D8       LDA $D8         Get exponent of work FPA.
DB32    A8          TAY             If zero then result is in
DB33    F0 CE       BEQ $DB03       main FPA, so exit.
DB35    38          SEC
DB36    E5 D0       SBC $D0         Get difference in exponents.
DB38    F0 24       BEQ $DB5E       If equal then skip shift.
DB3A    90 12       BCC $DB4E       If main FPA > work FPA then
DB3C    84 D0       STY $D0         skip change. Put work exponent
DB3E    A4 DD       LDY $DD         in main FPA. Put sign of
DB40    84 D5       STY $D5         work FPA into main FPA.
DB42    49 FF       EOR #$FF        Negate difference.
```

```
DB44    69 00       ADC #$00
DB46    A0 00       LDY #$00        Clear rounding byte.
DB48    84 C5       STY $C5
DB4A    A2 D0       LDX #$D0        Point to main FPA and skip
DB4C    D0 04       BNE $DB52       zeroing of rounding byte.
DB4E    A0 00       LDY #$00
DB50    84 DF       STY $DF         Clear rounding byte.
DB52    C9 F9       CMP #$F9        If more than 8 different then
DB54    30 C7       BMI $DB1D       shift the blocks.
DB56    A8          TAY             Transfer bit count to Y and
DB57    A5 DF       LDA $DF         rounding byte to A.
DB59    56 01       LSR $01,X       Shift mantissa by required
DB5B    20 6B DC    JSR $DC6B       number of bits.
DB5E    24 DE       BIT $DE         If signs are same then add
DB60    10 57       BPL $DBB9       mantissa and exit.
DB62    A0 D0       LDY #$D0        X points to smaller and Y
DB64    E0 D8       CPX #$D8        to larger FPA.
DB66    F0 02       BEQ $DB6A
DB68    A0 D8       LDY #$D8
DB6A    38          SEC             Negate rounding byte.
DB6B    49 FF       EOR #$FF        Add other rounding byte to
DB6D    65 C5       ADC $C5         get new one.
DB6F    85 DF       STA $DF
DB71    B9 04 00    LDA $0004,Y
DB74    F5 04       SBC $04,X       Subtract LSBs of mantissas.
DB76    85 D4       STA $D4
DB78    B9 03 00    LDA $0003,Y     Subtract next LSBs.
DB7B    F5 03       SBC $03,X
DB7D    85 D3       STA $D3
DB7F    B9 02 00    LDA $0002,Y     Subtract next LSBs.
DB82    F5 02       SBC $02,X
DB84    85 D2       STA $D2
DB86    B9 01 00    LDA $0001,Y     Subtract MSBs of mantissas.
DB89    F5 01       SBC $01,X
DB8B    85 D1       STA $D1
DB8D    B0 03       BCS $DB92       If carry clear then negate it.
DB8F    20 02 DC    JSR $DC02

DB92    A0 00       LDY #$00        NORMALISE MAIN FPA.
DB94    98          TYA             Set shift count to 0.
DB95    18          CLC
DB96    A6 D1       LDX $D1         If top byte empty then shift
DB98    D0 4A       BNE $DBE4       bits.
DB9A    A6 D2       LDX $D2
DB9C    86 D1       STX $D1         Shift by whole byte.
DB9E    A6 D3       LDX $D3
DBA0    86 D2       STX $D2
DBA2    A6 D4       LDX $D4
DBA4    86 D3       STX $D3
DBA6    A6 DF       LDX $DF
DBA8    86 D4       STX $D4
DBAA    84 DF       STY $DF
DBAC    69 08       ADC #$08        Update count.
DBAE    C9 28       CMP #$28        If underflow then set to zero
DBB0    D0 E4       BNE $DB96       else go round again.
DBB2    A9 00       LDA #$00        Set main FPA to 0.
DBB4    85 D0       STA $D0
DBB6    85 D5       STA $D5
DBB8    60          RTS             Exit.

DBB9    65 C5       ADC $C5         ADD MANTISSAS.
DBBB    85 DF       STA $DF
DBBD    A5 D4       LDA $D4         Add the Founding bytes and
DBBF    65 DC       ADC $DC         then each of the bytes in the
```

```
DBC1    85 D4       STA $D4         mantissa in ascending order
DBC3    A5 D3       LDA $D3         of significance.
DBC5    65 DB       ADC $DB
DBC7    85 D3       STA $D3
DBC9    A5 D2       LDA $D2
DBCB    65 DA       ADC $DA
DBCD    85 D2       STA $D2
DBCF    A5 D1       LDA $D1
DBD1    65 D9       ADC $D9
DBD3    85 D1       STA $D1
DBD5    4C F1 DB    JMP $DBF1       Shift if necessary and exit.

DBD8    69 01       ADC #$01        Put main FPA into standard
DBDA    06 DF       ASL $DF         form by shifting bits until
DBDC    26 D4       ROL $D4         top one is set.
DBDE    26 D3       ROL $D3
DBE0    26 D2       ROL $D2
DBE2    26 D1       ROL $D1
DBE4    10 F2       BPL $DBD8
DBE6    38          SEC             If underflow then zero the
DBE7    E5 D0       SBC $D0         number.
DBE9    B0 C7       BCS $DBB2
DBEB    49 FF       EOR #$FF        Negate A to get the new
DBED    69 01       ADC #$01        exponent.
DBEF    85 D0       STA $D0
DBF1    90 0E       BCC $DC01       Exit if okay.
DBF3    E6 D0       INC $D0         Increment exponent and shift
DBF5    F0 42       BEQ $DC39       down mantissa by 1 bit.
DBF7    66 D1       ROR $D1
DBF9    66 D2       ROR $D2
DBFB    66 D3       ROR $D3
DBFD    66 D4       ROR $D4
DBFF    66 DF       ROR $DF
DC01    60          RTS

DC02    A5 D5       LDA $D5         Negate the content of the
DC04    49 FF       EOR #$FF        main FPA.
DC06    85 D5       STA $D5
DC08    A5 D1       LDA $D1         Achieved by finding 2's
DC0A    49 FF       EOR #$FF        complement value of mantissa
DC0C    85 D1       STA $D1         and inverting sign bit.
DC0E    A5 D2       LDA $D2
DC10    49 FF       EOR #$FF
DC12    85 D2       STA $D2
DC14    A5 D3       LDA $D3
DC16    49 FF       EOR #$FF
DC18    85 D3       STA $D3
DC1A    A5 D4       LDA $D4
DC1C    49 FF       EOR #$FF
DC1E    85 D4       STA $D4
DC20    A5 DF       LDA $DF
DC22    49 FF       EOR #$FF
DC24    85 DF       STA $DF
DC26    E6 DF       INC $DF         Increment rounding byte and
DC28    D0 0E       BNE $DC38       exit if no carry.
DC2A    E6 D4       INC $D4         Increment mantissa of main FPA
DC2C    D0 0A       BNE $DC38       branching at each stage if
DC2E    E6 D3       INC $D3         no carry from one byte to
DC30    D0 06       BNE $DC38       next.
DC32    E6 D2       INC $D2
DC34    D0 02       BNE $DC38
DC36    E6 D1       INC $D1
DC38    60          RTS
```

```
DC39    A2 45       LDX #$45        Print "OVERFLOW ERROR".
DC3B    4C 7E C4    JMP $C47E
DC3E    A2 94       LDX #$94        Shift mantissa & keep sign.
DC40    B4 04       LDY $04,X
DC42    84 DF       STY $DF         Copy LSB into rounding byte.
DC44    B4 03       LDY $03,X
DC46    94 04       STY $04,X       Copy each of the other bytes
DC48    B4 02       LDY $02,X       of the mantissa into the one
DC4A    94 03       STY $03,X       lower down.
DC4C    B4 01       LDY $01,X
DC4E    94 02       STY $02,X
DC50    A4 D7       LDY $D7
DC52    94 01       STY $01,X
DC54    69 08       ADC #$08        If more than 7 bits of shift
DC56    30 E8       BMI $DC40       still required then go round
DC58    F0 E6       BEQ $DC40       again.
DC5A    E9 08       SBC #$08        Re-adjust counter.
DC5C    A8          TAY
DC5D    A5 DF       LDA $DF
DC5F    B0 14       BCS $DC75
DC61    16 01       ASL $01,X       Shift each bit of the mantissa
DC63    90 02       BCC $DC67       one bit to the right. Keep the
DC65    F6 01       INC $01,X       sign in the top bit of the
DC67    76 01       ROR $01,X       most significant byte of the
DC69    76 01       ROR $01,X       mantissa.
DC6B    76 02       ROR $02,X
DC6D    76 03       ROR $03,X
DC6F    76 04       ROR $04,X
DC71    6A          ROR A
DC72    C8          INY             Repeat until correct number of
DC73    D0 EC       BNE $DC61       bit shifts have taken place.
DC75    18          CLC
DC76    60          RTS             Exit.

DC77    82 13 5D 8D DE              LN(10) F1oating point numbers for use
DC7C    82 49 0F DA 9E              PI     by numeric functions.
DC81    81 00 00 00 00              1.0
DC86    03

DC87    7F 5E 56 CB 79              Data for the Log series.
DC8C    80 13 9B 0B 64
DC91    80 76 38 93 16
DC96    82 38 AA 3B 20
DC9B    80 35 04 F3 34              SQR(.5) More floating point numbers.
DCA0    81 35 04 F3 34              SQR(2)
DCA5    80 80 00 00 00              -0.5
DCAA    80 31 72 17 F8              LN(2)

DCAF    20 13 DF    JSR $DF13       LN Test main FPA.
DCB2    F0 02       BEQ $DCB6       Give error if zero.
DCB4    10 03       BPL $DCB9       Give error if negative.
DCB6    4C 36 D3    JMP $D336       Print "ILLEGAL QUANTITY ERR.".
DCB9    A5 D0       LDA $D0
DCBB    E9 7F       SBC #$7F
DCBD    48          PHA             Save signed binary exponent.
DCBE    A9 80       LDA #$80        Set exponent to +0.
DCC0    85 D0       STA $D0
DCC2    A9 9B       LDA #$9B
DCC4    A0 DC       LDY #$DC
DCC6    20 22 DB    JSR $DB22       Add SQR(.5) to number.
DCC9    A9 A0       LDA #$A0
DCCB    A0 DC       LDY #$DC
DCCD    20 E4 DD    JSR $DDE4       Divide number into SQR(2).
DCD0    A9 81       LDA #$81
```

```
DCD2    A0 DC       LDY #$DC
DCD4    20 0B DB    JSR $DB0B       Subtract from 1.
DCD7    A9 86       LDA #$86
DCD9    A0 DC       LDY #$DC
DCDB    20 FD E2    JSR $E2FD       Evaluate LN series.
DCDE    A9 A5       LDA #$A5
DCE0    A0 DC       LDY #$DC
DCE2    20 22 DB    JSR $DB22       Add -0.5.
DCE5    68          PLA             Get exponent.
DCE6    20 76 E0    JSR $E076       Add A to main FPA.
DCE9    A9 AA       LDA #$AA
DCEB    A0 DC       LDY #$DC        Point A, Y to LN(2).
DCED    20 51 DD    JSR $DD51       Unpack work FPA.
DCF0    D0 03       BNE $DCF5       * OPERATOR.
DCF2    4C 50 DD    JMP $DD50       Exit if work FPA is zero.
DCF5    20 7C DD    JSR $DD7C       Check & set up exponents.
DCF8    A9 00       LDA #$00        Clear work area.
DCFA    85 95       STA $95
DCFC    85 96       STA $96
DCFE    85 97       STA $97
DD00    85 98       STA $98
DD02    A5 DF       LDA $DF         Multiply using rounding byte.
DD04    20 1E DD    JSR $DD1E
DD07    A5 D4       LDA $D4         Multiply using LSB of
DD09    20 1E DD    JSR $DD1E       mantissa.
DD0C    A5 D3       LDA $D3         Multiply using next LSB of
DD0E    20 1E DD    JSR $DD1E       mantissa.
DD11    A5 D2       LDA $D2         Multiply using next LSB of
DD13    20 1E DD    JSR $DD1E       mantissa.
DD16    A5 D1       LDA $D1         Multiply using MSB of the
DD18    20 23 DD    JSR $DD23       mantissa.
DD1B    4C 64 DE    JMP $DE64       Transfer to main FPA & exit.
DD1E    D0 03       BNE $DD23       If byte is zero then shift up
DD20    4C 3E DC    JMP $DC3E       work area by 8 bits.

DD23    4A          LSR A           Set dummy bit to keep shifting
DD24    09 80       ORA #$80
DD26    A8          TAY             Save control byte.
DD27    90 19       BCC $DD42
DD29    18          CLC
DD2A    A5 98       LDA $98
DD2C    65 DC       ADC $DC         Add LSBs.
DD2E    85 98       STA $98
DD30    A5 97       LDA $97         Do next LSBs.
DD32    65 DB       ADC $DB
DD34    85 97       STA $97
DD36    A5 96       LDA $96         Do next LSBs.
DD38    65 DA       ADC $DA
DD3A    85 96       STA $96
DD3C    A5 95       LDA $95         Do MSBs.
DD3E    65 D9       ADC $D9
DD40    85 95       STA $95
DD42    66 95       ROR $95         Shift work area down by 1 bit.
DD44    66 96       ROR $96
DD46    66 97       ROR $97
DD48    66 98       ROR $98
DD4A    66 DF       ROR $DF
DD4C    98          TYA             Restore control byte and shift
DD4D    4A          LSR A           out 1 bit.
DD4E    D0 D6       BNE $DD26       Loop again if not finished.
DD50    60          RTS

DD51    85 91       STA $91         UNPACK WORK FPA FROM MEMORY.
DD53    84 92       STY $92         Save pointer to variable.
```

```
DD55    A0 04       LDY #$04
DD57    B1 91       LDA ($91),Y     Set up LSB.
DD59    85 DC       STA $DC
DD5B    88          DEY
DD5C    B1 91       LDA ($91),Y     Set up next LSB.
DD5E    85 DB       STA $DB
DD60    88          DEY
DD61    B1 91       LDA ($91),Y     Set up next LSB.
DD63    85 DA       STA $DA
DD65    88          DEY
DD66    B1 91       LDA ($91),Y     Set up sign byte.
DD68    85 DD       STA $DD
DD6A    45 D5       EOR $D5
DD6C    85 DE       STA $DE         Set sign difference byte.
DD6E    A5 DD       LDA $DD
DD70    09 80       ORA #$80
DD72    85 D9       STA $D9         Set up MSB.
DD74    88          DEY
DD75    B1 91       LDA ($91),Y     Set up exponent.
DD77    85 D8       STA $D8
DD79    A5 D0       LDA $D0         Get exponent of main FPA and
DD7B    60          RTS             exit.
DD7C    A5 D8       LDA $D8         Check & set exponents for
DD7E    F0 1F       BEQ $DD9F       If work FPA is zero then exit.
DD80    18          CLC
DD81    65 D0       ADC $D0         Add other exponent byte.
DD83    90 04       BCC $DD89       Handle under/overflow.
DD85    30 1D       BMI $DDA4
DD87    18          CLC
DD88    2C 10 14    BIT $1410
DD8B    69 80       ADC #$80        Set proper offset 80 format
DD8D    85 D0       STA $D0         for exponent.
DD8F    D0 03       BNE $DD94       If exponent is zero then clear
DD91    4C B6 DB    JMP $DBB6       Founding byte and exit.

DD94    A5 DE       LDA $DE         Set sign to sign difference
DD96    85 D5       STA $D5         byte.
DD98    60          RTS

DD99    A5 D5       LDA $D5
DD9B    49 FF       EOR #$FF        If non zero then give error.
DD9D    30 05       BMI $DDA4
DD9F    68          PLA             Pull return address off stack.
DDA0    68          PLA
DDA1    4C B2 DB    JMP $DBB2       Set main FPA to 0 and exit.

DDA4    4C 39 DC    JMP $DC39       Print "OVERFLOW ERROR".

DDA7    20 E5 DE    JSR $DEE5       MULTIPLY MAIN FPA BY 10.
DDAA    AA          TAX             Copy main FPA into work FPA.
DDAB    F0 10       BEQ $DDBD       Number is zero.
DDAD    18          CLC             Multiply by 4 by adding 2 to
DDAE    69 02       ADC #$02        the exponent.
DDB0    B0 F2       BCS $DDA4       Error if too large.
DDB2    A2 00       LDX #$00
DDB4    86 DE       STX $DE
DDB6    20 32 DB    JSR $DB32       Add in original number.
DDB9    E6 D0       INC $D0         Double result.
DDBB    F0 E7       BEQ $DDA4       Error if exponent too big.
DDBD    60          RTS

DD6E    84 20 00 00 00              Floating point number - 10.

DDC3    20 E5 DE    JSR $DEE5       DIVIDE MAIN FPA BY 10.
```

```
DDC6    A9 BE       LDA #$BE        Copy main FPA to work FPA.
DDC8    A0 DD       LDY #$DD        Point to the number 10.
DDCA    A2 00       LDX #$00
DDCC    86 DE       STX $DE         Clear sign difference byte.
DDCE    20 7B DE    JSR $DE7B       Unpack number pointed to.
DDD1    4C E7 DD    JMP $DDE7       Divide numbers.

DDD4    20 AF DC    JSR $DCAF       LOG Find LN of number.
DDD7    20 E5 DE    JSR $DEE5       Copy main FPA into work FPA.
DDDA    A9 77       LDA #$77
DDDC    A0 DC       LDY #$DC        Point to conversion factor.
DDDE    20 7B DE    JSR $DE7B       Unpack number pointed to.
DDE1    4C E7 DD    JMP $DDE7       Divide to get correct result.

DDE4    20 51 DD    JSR $DD51       Unpack work FPA from memory.
DDE7    F0 76       BEQ $DE5F       / OPERATOR.
DDE9    20 F4 DE    JSR $DEF4       Round main FPA.
DDEC    A9 00       LDA #$00        Negate exponent of main FPA.
DDEE    38          SEC
DDEF    E5 D0       SBC $D0
DDF1    85 D0       STA $D0
DDF3    20 7C DD    JSR $DD7C       Check and set up exponents.
DDF6    E6 D0       INC $D0         Adjust exponent.
DDF8    F0 AA       BEQ $DDA4       Error if too big.
DDFA    A2 FC       LDX #$FC        Loop count.
DDFC    A9 01       LDA #$01        Terminator bit.
DDFE    A4 D9       LDY $D9         Compare main mantissa with
DE00    C4 D1       CPY $D1         that of work FPA.
DE02    D0 10       BNE $DE14
DE04    A4 DA       LDY $DA
DE06    C4 D2       CPY $D2
DE08    D0 0A       BNE $DE14
DE0A    A4 DB       LDY $DB
DE0C    C4 D3       CPY $D3
DE0E    D0 04       BNE $DE14
DE10    A4 DC       LDY $DC
DE12    C4 D4       CPY $D4
DE14    08          PHP             Save flags.
DE15    2A          ROL A           If not finished then don't
DE16    90 09       BCC $DE21       save it.
DE18    E8          INX             Save in workspace.
DE19    95 98       STA $98,X
DE1B    F0 32       BEQ $DE4F       Set terminator bit on last loop.
DE1D    10 34       BPL $DE53       Finished.
DE1F    A9 01       LDA #$01        Set terminator bit.
DE21    28          PLP             Restore flags.
DE22    B0 0E       BCS $DE32       Subtract if work > main FPA.
DE24    06 DC       ASL $DC         Shift up mantissa of work FPA.
DE26    26 DB       ROL $DB
DE28    26 DA       ROL $DA
DE2A    26 D9       ROL $D9         Do subtraction if bit has
DE2C    B0 E6       BCS $DE14       shifted out.
DE2E    30 CE       BMI $DDFE       Do compare if top bit set.
DE30    10 E2       BPL $DE14       Jump back.
DE32    A8          TAY             Save A.
DE33    A5 DC       LDA $DC         Subtract main FPA from work
DE35    E5 D4       SBC $D4         FPA.
DE37    85 DC       STA $DC
DE39    A5 DB       LDA $DB
DE3B    E5 D3       SBC $D3
DE3D    85 DB       STA $DB
DE3F    A5 DA       LDA $DA
DE41    E5 D2       SBC $D2
DE43    85 DA       STA $DA
```

```
DE45   A5 D9       LDA $D9
DE47   E5 D1       SBC $D1
DE49   85 D9       STA $D9
DE4B   98          TYA             Restore A.
DE4C   4C 24 DE    JMP $DE24       Jump back into loop.
DE4F   A9 40       LDA #$40        Set terminator bit for final
DE51   D0 CE       BNE $DE21       loop and jump back into loop.
DE53   0A          ASL A           Get bits into top 2 bits of
DE54   0A          ASL A           rounding byte.
DE55   0A          ASL A
DE56   0A          ASL A
DE57   0A          ASL A
DE58   0A          ASL A
DE59   85 DF       STA $DF         Store rounding byte.
DE5B   28          PLP             Remove flag from stack and
DE5C   4C 64 DE    JMP $DE64       transfer result to main FPA.

DE5F   A2 85       LDX #$85        Print "DIVISION BY ZERO
DE61   4C 7E C4    JMP $C47E       ERROR".

DE64   A5 95       LDA $95         Transfer work area into main
DE66   85 D1       STA $D1         FPA.
DE68   A5 96       LDA $96
DE6A   85 D2       STA $D2
DE6C   A5 97       LDA $97
DE6E   85 D3       STA $D3
DE70   A5 98       LDA $98
DE72   85 D4       STA $D4
DE74   4C 92 DB    JMP $DB92       Normalise and exit.

DE77   A9 7C       LDA #$7C        PI Point A, Y
DE79   A0 DC       LDY #$DC        PI.
DE7B   85 91       STA $91         UNPACK FP NUMBER POINTED TO BY
DE7D   84 92       STY $92         A,Y.
DE7F   A0 04       LDY #$04
DE81   B1 91       LDA ($91),Y     Copy the bytes down from
DE83   85 D4       STA $D4         memory starting with the LSB
DE85   88          DEY             and finishing with the MSB.
DE86   B1 91       LDA ($91),Y
DE88   85 D3       STA $D3
DE8A   88          DEY
DE8B   B1 91       LDA ($91),Y
DE8D   85 D2       STA $D2
DE8F   88          DEY
DE90   B1 91       LDA ($91),Y     Set sign byte.
DE92   85 D5       STA $D5
DE94   09 80       ORA #$80
DE96   85 D1       STA $D1         Save MSB.
DE98   88          DEY
DE99   B1 91       LDA ($91),Y     Get exponent.
DE9B   85 D0       STA $D0
DE9D   84 DF       STY $DF         Zero the rounding byte.
DE9F   60          RTS

DEA0   A2 CB       LDX #$CB        Set X to save main FPA in temp
DEA2   2C A2 C6    BIT $C6A2       locations. BIT hides LDX #$C6.
DEA5   A0 00       LDY #$00        Set Y (MSB) and branch to main
DEA7   F0 04       BEQ $DEAD       main part of routine.
DEA9   A6 B8       LDX $B8         Pack FPA into memory pointed
DEAB   A4 B9       LDY $B9         to by X,Y.
DEAD   20 F4 DE    JSR $DEF4       Round off main FPA.
DEB0   86 91       STX $91         Save pointer.
DEB2   84 92       STY $92
DEB4   A0 04       LDY #$04
```

```
DEB6    A5 D4       LDA $D4
DEB8    91 91       STA ($91),Y     Store LSB of mantissa in
DEBA    88          DEY             memory.
DEBB    A5 D3       LDA $D3
DEBD    91 91       STA ($91),Y     Store next LS6.
DEBF    88          DEY
DEC0    A5 D2       LDA $D2
DEC2    91 91       STA ($91),Y     Store next LSB.
DEC4    88          DEY
DEC5    A5 D5       LDA $D5
DEC7    09 7F       ORA #$7F
DEC9    25 D1       AND $D1
DECB    91 91       STA ($91),Y     Save MSB with sign packed in.
DECD    88          DEY
DECE    A5 D0       LDA $D0
DED0    91 91       STA ($91),Y     Save exponent.
DED2    84 DF       STY $DF
DED4    60          RTS

DED5    A5 DD       LDA $DD         Copy work FPA into main FPA.
DED7    85 D5       STA $D5         Transfer sign byte.
DED9    A2 05       LDX #$05
DEDB    B5 D7       LDA $D7,X       Transfer number.
DEDD    95 CF       STA $CF,X
DEDF    CA          DEX
DEE0    D0 F9       BNE $DEDB
DEE2    86 DF       STX $DF         Set Founding byte.
DEE4    60          RTS

DEE5    20 F4 DE    JSR $DEF4       Copy main FPA into work FPA.
DEE8    A2 06       LDX #$06
DEEA    B5 CF       LDA $CF,X       Transfer bytes including the
DEEC    95 D7       STA $D7,X       sign byte.
DEEE    CA          DEX
DEEF    D0 F9       BNE $DEEA
DEF1    86 DF       STX $DF         Set rounding byte.
DEF3    60          RTS

DEF4    A5 D0       LDA $D0         ROUND OFF MAIN FPA.
DEF6    F0 FB       BEQ $DEF3       Exit if it is zero.
DEF8    06 DF       ASL $DF         Exit if rounding byte is less
DEFA    90 F7       BCC $DEF3       than half.
DEFC    20 2A DC    JSR $DC2A       Increment mantissa.
DEFF    D0 F2       BNE $DEF3       Exit if okay otherwise jump to
DF01    4C F3 DB    JMP $DBF3       adjust exponent and mantissa.

DF04    20 A9 D2    JSR $D2A9       Convert main FPA to integer.
DF07    46 D4       LSR $D4         Go to TRUE/FALSE according to
DF09    B0 04       BCS $DF0F       lowest bit.

DF0B    A9 00       LDA #$00        FALSE
DF0D    F0 15       BEQ $DF24       Set main FPA to 0.

DF0F    A9 FF       LDA #$FF        TRUE
DF11    30 11       BMI $DF24       Set main FPA to -1

DF13    A5 D0       LDA $D0         GET SIGN OF MAIN FPA.
DF15    F0 09       BEQ $DF20
DF17    A5 D5       LDA $D5         A=0 if FPA is 0.
DF19    2A          ROL A           A=1 if FPA is +ve.
DF1A    A9 FF       LDA #$FF        A=#FF if FPA is -ve.
DF1C    B0 02       BCS $DF20
DF1E    A9 01       LDA #$01
DF20    60          RTS
```

```
DF21    20 13 DF    JSR $DF13       SGN Get sign into A.
DF24    85 D1       STA $D1         Set main FPA to signed single
DF26    A9 00       LDA #$00        byte in A.
DF28    85 D2       STA $D2
DF2A    A2 88       LDX #$88
DF2C    A5 D1       LDA $D1
DF2E    49 FF       EOR #$FF
DF30    2A          ROL A
DF31    A9 00       LDA #$00        Clear low 2 bytes of mantissa
DF33    85 D4       STA $D4
DF35    85 D3       STA $D3
DF37    86 D0       STX $D0         Set exponent to X.
DF39    85 DF       STA $DF         Clear sign and rounding bytes.
DF3B    85 D5       STA $D5
DF3D    4C 8D DB    JMP $DB8D       Normalise FPA and exit.

DF40    85 D1       STA $D1         Convert 2 byte integer in A
DF42    84 D2       STY $D2         (MSB) and Y (LSB) into
DF44    A2 90       LDX #$90        floating point number.
DF46    38          SEC             Jump to clear and normalise
DF47    B0 E8       BCS $DF31       rest of main FPA.
DF49    46 D5       LSR $D5         ABS Clear sign bit of main
DF4B    60          RTS             FPA and exit.

DF4C    85 93       STA $93         COMPARE MAIN FPA WITH NUMBER
DF4E    84 94       STY $94         IN MEMORY.
DF50    A0 00       LDY #$00        Set pointer.
DF52    B1 93       LDA ($93),Y     Get exponent.
DF54    C8          INY
DF55    AA          TAX
DF56    F0 BB       BEQ $DF13       If zero just test FPA.
DF58    B1 93       LDA ($93),Y
DF5A    45 D5       EOR $D5         If signs are different then
DF5C    30 B9       BMI $DF17       just test FPA.
DF5E    E4 D0       CPX $D0         If exponents not different
DF60    D0 21       BNE $DF83       then adjust for signs & exit.
DF62    B1 93       LDA ($93),Y
DF64    09 80       ORA #$80
DF66    C5 D1       CMP $D1         Test MSB of mantissa.
DF68    D0 19       BNE $DF83
DF6A    C8          INY             If MSB are equal then test the
DF6B    B1 93       LDA ($93),Y     next MSBs and so on until the
DF6D    C5 D2       CMP $D2         LSBs are reached.
DF6F    D0 12       BNE $DF83
DF71    C8          INY
DF72    B1 93       LDA ($93),Y
DF74    C5 D3       CMP $D3
DF76    D0 0B       BNE $DF83
DF78    C8          INY
DF79    A9 7F       LDA #$7F        Test LSBs allowing for
DF7B    C5 DF       CMP $DF         rounding.
DF7D    B1 93       LDA ($93),Y
DF7F    E5 D4       SBC $D4
DF81    F0 28       BEQ $DFAB
DF83    A5 D5       LDA $D5         Get sign byte, inverting if
DF85    90 02       BCC $DF89       FPA < memory.
DF87    49 FF       EOR #$FF
DF89    4C 19 DF    JMP $DF19       Set A accordingly.

DF8C    A5 D0       LDA $D0         CONVERT MAIN FPA TO INTEGER.
DF8E    F0 4A       BEQ $DFDA       Number is zero.
DF90    38          SEC             Calculate number of shifts to
DF91    E9 A0       SBC #$A0        the left to do.
```

```
DF93   24 D5       BIT $D5         Test sign of mantissa.
DF95   10 09       BPL $DFA0       Sign is positive.
DF97   AA          TAX
DF98   A9 FF       LDA #$FF
DF9A   85 D7       STA $D7         Invert sign extend byte and
DF9C   20 08 DC    JSR $DC08       content of mantissa.
DF9F   8A          TXA
DFA0   A2 D0       LDX #$D0
DFA2   C9 F9       CMP #$F9        If more than 7 bits of shift
DFA4   10 06       BPL $DFAC       are needed then call routine
DFA6   20 54 DC    JSR $DC54       to shift it.
DFA9   84 D7       STY $D7         Clear sign extend byte.
DFAB   60          RTS             Exit.

DFAC   A8          TAY
DFAD   A5 D5       LDA $D5         Shift mantissa as required.
DFAF   29 80       AND #$80
DFB1   46 D1       LSR $D1
DFB3   05 D1       ORA $D1
DFB5   85 D1       STA $D1
DFB7   20 6B DC    JSR $DC6B       Perform shift.
DFBA   84 D7       STY $D7         Clear sign extend byte.
DFBC   60          RTS

DFBD   A5 D0       LDA $D0         INT
DFBF   C9 A0       CMP #$A0        If number is over 2A32 then it
DFC1   B0 20       BCS $DFE3       is integer already.
DFC3   20 8C DF    JSR $DF8C       Convert to integer.
DFC6   84 DF       STY $DF         Zero sign and rounding bytes.
DFC8   A5 D5       LDA $D5
DFCA   84 D5       STY $D5
DFCC   49 80       EOR #$80
DFCE   2A          ROL A           Set carry if positive.
DFCF   A9 A0       LDA #$A0        Set exponent to 32.
DFD1   85 D0       STA $D0
DFD3   A5 D4       LDA $D4
DFD5   85 24       STA $24
DFD7   4C 8D DB    JMP $DB8D       Normalise and exit.
DFDA   85 D1       STA $D1         Zero the mantissa of main FPA.
DFDC   85 D2       STA $D2
DFDE   85 D3       STA $D3
DFE0   85 D4       STA $D4
DFE2   A8          TAY
DFE3   60          RTS             Exit.

DFE4   4C 81 E9    JMP $E981       Get hex number.

DFE7   A0 00       LDY #$00        GET NUMBER.
DFE9   A2 0A       LDX #$0A        Clear section of memory from
DFEB   94 CC       STY $CC,X       $CC to $D6 inclusive.
DFED   CA          DEX
DFEE   10 FB       BPL $DFEB
DFF0   90 13       BCC $E005       If digit, skip special tests.
DFF2   C9 23       CMP #$23
DFF4   F0 EE       BEQ $DFE4       "#" found, number is in hex.
DFF6   C9 2D       CMP #$2D
DFF8   D0 04       BNE $DFFE       No "-" sign before number.
DFFA   86 D6       STX $D6         Set sign to #FF If -ve.
DFFC   F0 04       BEQ $E002
DFFE   C9 2B       CMP #$2B
E000   D0 05       BNE $E007       No "+" sign before number.
E002   20 E2 00    JSR $00E2       Get next character.
E005   90 5B       BCC $E062       If digit, then add it in.
E007   C9 2E       CMP #$2E
```

```
E009    F0 2E       BEQ $E039       Character is ".".
E00B    C9 45       CMP #$45
E00D    D0 30       BNE $E03F       No "E" for exponent.
E00F    20 E2 00    JSR $00E2       Get next character.
E012    90 17       BCC $E02B       Character is a digit.
E014    C9 CD       CMP #$CD
E016    F0 0E       BEQ $E026       Character is a "-" token.
E018    C9 2D       CMP #$2D
E01A    F0 0A       BEQ $E026       Character is "-".
E01C    C9 CC       CMP #$CC
E01E    F0 08       BEQ $E028       Character is a "+" token.
E020    C9 2B       CMP #$2B
E022    F0 04       BEQ $E028       Character is "+".
E024    D0 07       BNE $E02D
E026    66 CF       ROR $CF         Set negative exponent.
E028    20 E2 00    JSR $00E2       Get next char.
E02B    90 5C       BCC $E089       If digit, then add it in.
E02D    24 CF       BIT $CF
E02F    10 0E       BPL $E03F       Exponent is positive.
E031    A9 00       LDA #$00
E033    38          SEC             Negate exponent.
E034    E5 CD       SBC $CD
E036    4C 41 E0    JMP $E041       Finish off.
E039    66 CE       ROR $CE         Set decimal point flag.
E03B    24 CE       BIT $CE         Loop around again if D.P. not
E03D    50 C3       BVC $E002       already set.
E03F    A5 CD       LDA $CD
E041    38          SEC             Decrement exponent & subtract
E042    E5 CC       SBC $CC         number of digits after D.P.
E044    85 CD       STA $CD         Save total exponent.
E046    F0 12       BEQ $E05A       Exponent is positive.
E048    10 09       BPL $E053
E04A    20 C3 DD    JSR $DDC3       Divide main FPA by 10, total
E04D    E6 CD       INC $CD         exponent negative.
E04F    D0 F9       BNE $E04A
E051    F0 07       BEQ $E05A
E053    20 A7 DD    JSR $DDA7       Multiply main FPA by 10, total
E056    C6 CD       DEC $CD         exponent positive.
E058    D0 F9       BNE $E053
E05A    A5 D6       LDA $D6         Negate if necessary and exit.
E05C    30 01       BMI $E05F
E05E    60          RTS

E05F    4C 71 E2    JMP $E271       Negate main FPA.

E062    48          PHA             Save digit.
E063    24 CE       BIT $CE         If char after D.P. then
E065    10 02       BPL $E069       increment decimal positions
E067    E6 CC       INC $CC         counter.
E069    20 A7 DD    JSR $DDA7       Multiply main FPA by 10.
E06C    68          PLA             Restore digit and reduce to
E06D    38          SEC             decimal digit.
E06E    E9 30       SBC #$30
E070    20 76 E0    JSR $E076       Add digit into FPA.
E073    4C 02 E0    JMP $E002       Jump back for more.

E076    48          PHA             ADD BYTE IN A TO MAIN FPA.
E077    20 E5 DE    JSR $DEE5       Save byte and copy main FPA
E07A    68          PLA             into work FPA. Restore byte.
E07B    20 24 DF    JSR $DF24       Set main FPA to signed byte
E07E    A5 DD       LDA $DD         in A.
E080    45 D5       EOR $D5
E082    85 DE       STA $DE         Set sign difference flag.
E084    A6 D0       LDX $D0
```

```
E086   4C 25 DB    JMP $DB25       Add the 2 FPAs and exit.

E089   A5 CD       LDA $CD         Deal with digit after E.
E08B   C9 0A       CMP #$0A        Test if exponent is < 10.
E08D   90 09       BCC $E098       Add in second digit.
E08F   A9 64       LDA #$64        Set underflow if negative
E091   24 CF       BIT $CF         exponent by using E-100.
E093   30 11       BMI $E0A6
E095   4C 39 DC    JMP $DC39       Print "OVERFLOW ERROR".

E098   0A          ASL A           Multiply exponent by 10.
E099   0A          ASL A
E09A   18          CLC
E09B   65 CD       ADC $CD
E09D   0A          ASL A
E09E   18          CLC
E09F   A0 00       LDY #$00
E0A1   71 E9       ADC ($E9),Y     Add next digit to exponent.
E0A3   38          SEC
E0A4   E9 30       SBC #$30        Reduce to decimal range.
E0A6   85 CD       STA $CD
E0A8   4C 28 E0    JMP $E028       Go round for next digit.

E0AB   9B 3E BC 1F FD              1E8         List of floating point
E0B0   9E 6E 6B 27 FD              9.99999E8   numbers for converting a
E0B5   9E 6E 6B 28 00              1E9         number to string.

E0BA   A9 AD       LDA #$AD        Print "IN" <line number>.
E0BC   A0 C3       LDY #$C3

E0BE   20 D2 E0    JSR $E0D2       Print "IN".
E0C1   A5 A9       LDA $A9         Get number into A,X.
E0C3   A6 A8       LDX $A8

E0C5   85 D1       STA $D1         PRINT INTEGER IN A,X.
E0C7   86 D2       STX $D2         Save integer in mantissa of
E0C9   A2 90       LDX #$90        main FPA. Set exponent to 16.
E0CB   38          SEC             Set sign to positive.
E0CC   20 31 DF    JSR $DF31       Normalise main FPA.
E0CF   20 D5 E0    JSR $E0D5       Convert number to a string.
E0D2   4C B0 CC    JMP $CCB0       Print out number.
E0D5   A0 01       LDY #$01        CONVERT NUMBER TO STRING.
E0D7   A9 20       LDA #$20
E0D9   24 D5       BIT $D5         Use space if positive or "-"
E0DB   10 02       BPL $E0DF       if negative.
E0DD   A9 2D       LDA #$2D
E0DF   99 FF 00    STA $00FF,Y     Write char to string.
E0E2   85 D5       STA $D5         Number now positive.
E0E4   84 E0       STY $E0         Save pointer.
E0E6   C8          INY
E0E7   A9 30       LDA #$30        Set A to "0".
E0E9   A6 D0       LDX $D0
E0EB   D0 03       BNE $E0F0       If number is zero then set
E0ED   4C F8 E1    JMP $E1F8       the string to "0" and exit.
E0F0   A9 00       LDA #$00
E0F2   E0 80       CPX #$80
E0F4   F0 02       BEQ $E0F8       Exponent is zero.
E0F6   B0 09       BCS $E101       Exponent is positive.
E0F8   A9 B5       LDA #$B5
E0FA   A0 E0       LDY #$E0
E0FC   20 ED DC    JSR $DCED       Multiply main FPA by 1E9.
E0FF   A9 F7       LDA #$F7
E101   85 CC       STA $CC         Set initial E value to -9.
E103   A9 B0       LDA #$B0
```

```
E105   A0 E0       LDY #$E0        Compare main FPA with
E107   20 4C DF    JSR $DF4C       9.9999E8.
E10A   F0 1E       BEQ $E12A       Convert if equal.
E10C   10 12       BPL $E120       Main FPA is greater.
E10E   A9 AB       LDA #$AB
E110   A0 E0       LDY #$E0
E112   20 4C DF    JSR $DF4C       Compare main FPA with 1E8.
E115   F0 02       BEQ $E119       Number in main FPA is 1E8.
E117   10 0E       BPL $E127       Number in main FPA > 1E8.
E119   20 A7 DD    JSR $DDA7       Multiply main FPA by 10.
E11C   C6 CC       DEC $CC         Adjust exponent.
E11E   D0 EE       BNE $E10E       Go round again.
E120   20 C3 DD    JSR $DDC3       Divide main FPA by 10.
E123   E6 CC       INC $CC         Adjust exponent.
E125   D0 DC       BNE $E103       Go round again.
E127   20 04 DB    JSR $DB04       Add 0.5 to round off.
E12A   20 8C DF    JSR $DF8C       Convert main FPA to integer.
E12D   A2 01       LDX #$01        Set X to 1 place before D.P.
E12F   A5 CC       LDA $CC
E131   18          CLC
E132   69 0A       ADC #$0A        Add 10 to exponent.
E134   30 09       BMI $E13F       If -ve then use E form.
E136   C9 0B       CMP #$0B        If too big then use E form.
E138   B0 06       BCS $E140
E13A   69 FF       ADC #$FF        Set digit before ".".
E13C   AA          TAX
E13D   A9 02       LDA #$02        Force E value.
E13F   38          SEC
E140   E9 02       SBC #$02        Set proper exponent.
E142   85 CD       STA $CD
E144   86 CC       STX $CC
E146   8A          TXA
E147   F0 02       BEQ $E14B       Skip leading zeroes before
E149   10 13       BPL $E15E       decimal point.
E14B   A4 E0       LDY $E0         Get pointer to $ construction
E14D   A9 2E       LDA #$2E        area.
E14F   C8          INY
E150   99 FF 00    STA $00FF,Y     Write a decimal point.
E153   8A          TXA
E154   F0 06       BEQ $E15C       If not ".OXXX" then skip "0".
E156   A9 30       LDA #$30        Write a "0" to string.
E158   C8          INY
E159   99 FF 00    STA $00FF,Y
E15C   84 E0       STY $E0         Save pointer.
E15E   A0 00       LDY #$00
E160   A2 80       LDX #$80        Initialise decimal exponent.
E162   A5 D4       LDA $D4
E164   18          CLC
E165   79 0D E2    ADC $E20D,Y     Add or subtract the divisor
E168   85 D4       STA $D4         (depending on number added)
E16A   A5 D3       LDA $D3         from the number in main FPA.
E16C   79 0C E2    ADC $E20C,Y
E16F   85 D3       STA $D3
E171   A5 D2       LDA $D2
E173   79 0B E2    ADC $E20B,Y
E176   85 D2       STA $D2
E178   A5 D1       LDA $D1
E17A   79 0A E2    ADC $E20A,Y
E17D   85 D1       STA $D1         Adjust exponent.
E17F   E8          INX
E180   B0 04       BCS $E186
E182   10 DE       BPL $E162       If no overflow then loop.
E184   30 02       BMI $E188
E186   30 DA       BMI $E162
```

```
E188   8A          TXA             Number of shifts.
E189   90 04       BCC $E18F
E18B   49 FF       EOR #$FF        If adding then subtract from
E18D   69 0A       ADC #$0A        10 to get digit.
E18F   69 2F       ADC #$2F        Convert to Ascii.
E191   C8          INY             Update division pointer.
E192   C8          INY
E193   C8          INY
E194   C8          INY
E195   84 B6       STY $B6         Save it.
E197   A4 E0       LDY $E0         Get digit pointer.
E199   C8          INY             Advance it.
E19A   AA          TAX             Save add/subtraction direction
E19B   29 7F       AND #$7F
E19D   99 FF 00    STA $00FF,Y     Write digit to string.
E1A0   C6 CC       DEC $CC         Decrement digits before ".".
E1A2   D0 06       BNE $E1AA
E1A4   A9 2E       LDA #$2E        Write "." to string if
E1A6   C8          INY             necessary.
E1A7   99 FF 00    STA $00FF,Y
E1AA   84 E0       STY $E0         Save digit pointer.
E1AC   A4 B6       LDY $B6         Get division pointer.
E1AE   8A          TXA
E1AF   49 FF       EOR #$FF        Swap add/subtract flag.
E1B1   29 80       AND #$80
E1B3   AA          TAX
E1B4   C0 24       CPY #$24
E1B6   D0 AA       BNE $E162       Loop if not finished.
E1B8   A4 E0       LDY $E0         Get digit pointer.
E1BA   B9 FF 00    LDA $00FF,Y
E1BD   88          DEY
E1BE   C9 30       CMP #$30
E1C0   F0 F8       BEQ $E1BA       Strip trailing zeroes.
E1C2   C9 2E       CMP #$2E
E1C4   F0 01       BEQ $E1C7       Strip off "." if on end.
E1C6   C8          INY
E1C7   A9 2B       LDA #$2B        "+".
E1C9   A6 CD       LDX $CD
E1CB   F0 2E       BEQ $E1FB       If no E required then exit.
E1CD   10 08       BPL $E1D7       If positive E, skip negation.
E1CF   A9 00       LDA #$00        Negate decimal exponent and
E1D1   38          SEC             use "-".
E1D2   E5 CD       SBC $CD
E1D4   AA          TAX
E1D5   A9 2D       LDA #$2D        "-".
E1D7   99 01 01    STA $0101,Y     Put character in string.
E1DA   A9 45       LDA #$45
E1DC   99 00 01    STA $0100,Y     Put "E" in string.
E1DF   8A          TXA             Get decimal exponent.
E1E0   A2 2F       LDX #$2F        Initialise Ascii char.
E1E2   38          SEC
E1E3   E8          INX
E1E4   E9 0A       SBC #$0A        Subtract 10 to divide A by 10.
E1E6   B0 FB       BCS $E1E3       Result will be in X.
E1E8   69 3A       ADC #$3A
E1EA   99 03 01    STA $0103,Y     Least significant decimal
E1ED   8A          TXA             digit.
E1EE   99 02 01    STA $0102,Y     Write digit.
E1F1   A9 00       LDA #$00
E1F3   99 04 01    STA $0104,Y
E1F6   F0 08       BEQ $E200
E1F8   99 FF 00    STA $00FF,Y
E1FB   A9 00       LDA #$00        Terminate string with a null
E1FD   99 00 01    STA $0100,Y     and exit.
```

```
E200   A9 00       LDA #$00
E202   A0 01       LDY #$01
E204   60          RTS

E205   80 00 00 00 00            Floating point 0.5.
E20A   FA 0A 1F 00 -1E8           4 byte integers for use with
E20E   00 98 96 80 +1E7           the routine that converts
E212   FF F0 BD C0 -1E6           numbers to strings. Decimal
E216   00 01 86 A0 +1E5           values are also listed in
E21A   FF FF D8 F0 -1E4           exponent form.
E21E   00 00 03 E8 +1E3
E222   FF FF FF 9C -1E2
E226   00 00 00 0A +1E1
E22A   FF FF FF FF -1E0

E22E   20 E5 DE    JSR $DEE5      SQR Copy main to work FPA.
E231   A9 05       LDA #$05       Point to the number 0.5.
E233   A0 E2       LDY #$E2
E235   20 7B DE    JSR $DE7B      Unpack it into main FPA.
E238   F0 70       BEQ $E2AA      A OPERATOR. If zero give 1
E23A   A5 D8       LDA $D8        as result.
E23C   D0 03       BNE $E241
E23E   4C B4 DB    JMP $DBB4      If work FPA is 0 so is result.
E241   A2 BD       LDX #$BD
E243   A0 00       LDY #$00       Pack main FPA to $BD-$C1.
E245   20 AD DE    JSR $DEAD
E248   A5 DD       LDA $DD
E24A   10 0F       BPL $E25B      Branch if work FPA is +ve.
E24C   20 BD DF    JSR $DFBD      Get INT of main FPA.
E24F   A9 BD       LDA #$BD
E251   A0 00       LDY #$00       Compare new main FPA against
E253   20 4C DF    JSR $DF4C      copy of old main FPA.
E256   D0 03       BNE $E25B      If no fractional part use +ve
E258   98          TYA            result.
E259   A4 24       LDY $24        Get sign of main FPA and copy
E25B   20 D7 DE    JSR $DED7      work FPA into main FPA.
E25E   98          TYA
E25F   48          PHA
E260   20 AF DC    JSR $DCAF      Get LN of main FPA.
E263   A9 BD       LDA #$BD
E265   A0 00       LDY #$00
E267   20 ED DC    JSR $DCED      Multiply by number at $BD-$C1.
E26A   20 AA E2    JSR $E2AA      Get EXP of main FPA.
E26D   68          PLA            Get sign flag.
E26E   4A          LSR A
E26F   90 0A       BCC $E27B      Exit if positive.
E271   A5 D0       LDA $D0        UNARY "-" OPERATOR.
E273   F0 06       BEQ $E27B      Exit if zero.
E275   A5 D5       LDA $D5        Swap sign.
E277   49 FF       EOR #$FF
E279   85 D5       STA $D5
E27B   60          RTS            Exit.

E27C   81 38 AA 3B 29            Data for EXP routine.
E281   07

E282   71 34 58 3E 56
E287   74 16 7E B3 1B
E28C   77 2F EE E3 85
E291   7A 1D 84 1C 2A
E296   7C 63 59 58 0A
E29B   7E 75 FD E7 C6
E2A0   80 31 72 18 10
E2A5   81 00 00 00 00
```

```
E2AA   A9 7C       LDA #$7C        EXP
E2AC   A0 E2       LDY #$E2        Unpack number into work FPA
E2AE   20 ED DC    JSR $DCED       from $E27C.
E2B1   A5 DF       LDA $DF         Increment rounding byte and
E2B3   69 50       ADC #$50        mantissa if need be.
E2B5   90 03       BCC $E2BA
E2B7   20 FC DE    JSR $DEFC
E2BA   85 C5       STA $C5         Save copy of rounding byte.
E2BC   20 E8 DE    JSR $DEE8       Copy main FPA to work FPA.
E2BF   A5 D0       LDA $D0
E2C1   C9 88       CMP #$88
E2C3   90 03       BCC $E2C8       Exponent less than 8.
E2C5   20 99 DD    JSR $DD99       Check size.
E2C8   20 BD DF    JSR $DFBD       Find integer of number.
E2CB   A5 24       LDA $24
E2CD   18          CLC
E2CE   69 81       ADC #$81
E2D0   F0 F3       BEQ $E2C5
E2D2   38          SEC
E2D3   E9 01       SBC #$01
E2D5   48          PHA             Save exponent.
E2D6   A2 05       LDX #$05
E2D8   B5 D8       LDA $D8,X
E2DA   B4 D0       LDY $D0,X       Swap the EPAs.
E2DC   95 D0       STA $D0,X
E2DE   94 D8       STY $D8,X
E2E0   CA          DEX
E2E1   10 F5       BPL $E2D8
E2E3   A5 C5       LDA $C5
E2E5   85 DF       STA $DF         Restore rounding byte.
E2E7   20 0E DB    JSR $DB0E       Perform subtraction.
E2EA   20 71 E2    JSR $E271       Negate the result.
E2ED   A9 81       LDA #$81
E2EF   A0 E2       LDY #$E2        Set pointer to series data.
E2F1   20 13 E3    JSR $E313       Evaluate series.
E2F4   A9 00       LDA #$00        Clear sign difference byte.
E2F6   85 DE       STA $DE
E2F8   68          PLA             Pull exponent.
E2F9   20 7E DD    JSR $DD7E       Check size and set up exponent
E2FC   60          RTS             and exit.

E2FD   85 E0       STA $E0         Set pointer.
E2FF   84 E1       STY $E1
E301   20 A3 DE    JSR $DEA3       Store main FPA at $C6-$CA.
E304   A9 C6       LDA #$C6        Unpack number from memory
E306   20 ED DC    JSR $DCED       and multiply.
E309   20 17 E3    JSR $E317       Evaluate series.
E30C   A9 C6       LDA #$C6
E30E   A0 00       LDY #$00
E310   4C ED DC    JMP $DCED       Unpack main FPA from $C6-$CA.

E313   85 E0       STA $E0         SERIES EVALUATION.
E315   84 E1       STY $E1         Set pointer to data.
E317   20 A0 DE    JSR $DEA0       Store main FPA at $CB-$CF.
E31A   B1 E0       LDA ($E0),Y     Load and set up loop counter.
E31C   85 D6       STA $D6
E31E   A4 E0       LDY $E0         Increment $E0,$E1 and leave a
E320   C8          INY             copy of result in A,Y.
E321   98          TYA
E322   D0 02       BNE $E326
E324   E6 E1       INC $E1
E326   85 E0       STA $E0
E328   A4 E1       LDY $E1
```

```
E32A   20 ED DC   JSR $DCED       Unpack work FPA from memory
E32D   A5 E0      LDA $E0         and multiply.
E32F   A4 E1      LDY $E1
E331   18         CLC             Add 5 to pointer at $E0,#E1
E332   69 05      ADC #$05        so that it points to next
E334   90 01      BCC $E337       piece of data. Leave copy of
E336   C8         INY             result in A,Y.
E337   85 E0      STA $E0
E339   84 E1      STY $E1
E33B   20 22 DB   JSR $DB22       Unpack work FPA & add to main
E33E   A9 CB      LDA #$CB        FPA.
E340   A0 00      LDY #$00        Set A,Y to point to copy of
E342   C6 D6      DEC $D6         wok FPA.
E344   D0 E4      BNE $E32A       Repeat until loop has counted
E346   60         RTS             out and then exit.

E347   98 35 44 7A               Data for RND command.
E34B   68 28 B1 46

E34F   20 13 DF   JSR $DF13       RND Get sign of main FPA.
E352   AA         TAX             Save it in X.
E353   30 18      BMI $E36D       Main FPA is negative.
E355   A9 FA      LDA #$FA
E357   A0 00      LDY #$00
E359   20 7B DE   JSR $DE7B       Unpack number at $FA.
E35C   8A         TXA
E35D   F0 E7      BEQ $E346       Number is zero.
E35F   A9 47      LDA #$47
E361   A0 E3      LDY #$E3        Unpack work FPA from $E347
E363   20 ED DC   JSR $DCED       and multiply with main FPA.
E366   A9 4B      LDA #$4B
E368   A0 E3      LDY #$E3        Unpack work FPA from $E34B
E36A   20 22 DB   JSR $DB22       and add to main FPA.
E36D   A6 D4      LDX $D4
E36F   A5 D1      LDA $D1         Swap MSB and LSB of main FPA.
E371   85 D4      STA $D4
E373   86 D1      STX $D1
E375   A9 00      LDA #$00
E377   85 D5      STA $D5         Clear sign.
E379   A5 D0      LDA $D0         Transfer exponent to rounding
E37B   85 DF      STA $DF         byte.
E37D   A9 80      LDA #$80        Set exponent to zero.
E37F   85 D0      STA $D0
E381   20 92 DB   JSR $DB92       Normalise main FPA.
E384   A2 FA      LDX #$FA
E386   A0 00      LDY #$00        Pack main FPA into memory at
E388   4C AD DE   JMP $DEAD       $FA.

E38B   A9 07      LDA #$07        COS
E38D   A0 E4      LDY #$E4        Unpack work FPA from $E407 and
E38F   20 22 DB   JSR $DB22       add to main FPA.

E392   20 E5 DE   JSR $DEE5       SIN Copy main to work FPA.
E395   A9 0C      LDA #$0C
E397   A0 E4      LDY #$E4
E399   A6 DD      LDX $DD         Unpack main FPA from $EC04 and
E39B   20 CC DD   JSR $DDCC       divide by work FPA.
E39E   20 E5 DE   JSR $DEE5       Copy main FPA to work FPA.
E3A1   20 BD DF   JSR $DFBD       Get INTeger value.
E3A4   A9 00      LDA #$00        Clear sign difference byte.
E3A6   85 DE      STA $DE
E3A8   20 0E DB   JSR $DB0E       Subtract FPAs.
E3AB   A9 11      LDA #$11
E3AD   A0 E4      LDY #$E4        Unpack work FPA from $E411
```

```
E3AF   20 0B DB   JSR $DB0B       and subtact from main FPA.
E3B2   A5 D5      LDA $D5         Save sign of mantissa
E3B4   48         PHA
E3B5   10 0D      BPL $E3C4       Sign of mantissa is positive.
E3B7   20 04 DB   JSR $DB04       Add 0.5 to result.
E3BA   A5 D5      LDA $D5
E3BC   30 09      BMI $E3C7       Result negative.
E3BE   A5 2D      LDA $2D         Invert temporary operator
E3C0   49 FF      EOR #$FF        store.
E3C2   85 2D      STA $2D
E3C4   20 71 E2   JSR $E271       Negate number.
E3C7   A9 11      LDA #$11
E3C9   A0 E4      LDY #$E4        Unpack work FPA from $£411 and
E3CB   20 22 DB   JSR $DB22       add to main FPA.
E3CE   68         PLA
E3CF   10 03      BPL $E3D4       Sign is positive.
E3D1   20 71 E2   JSR $E271       Negate number.
E3D4   A9 16      LDA #$16
E3D6   A0 E4      LDY #$E4        Set pointers.
E3D8   4C FD E2   JMP $E2FD       Jump to series evaluation.
E3DB   20 A3 DE   JSR $DEA3       TAN Store main FPA at $C6.
E3DE   A9 00      LDA #$00
E3E0   85 2D      STA $2D
E3E2   20 92 E3   JSR $E392       Find SIN of number.
E3E5   A2 BD      LDX #$BD
E3E7   A0 00      LDY #$00
E3E9   20 88 E3   JSR $E388       Save result in memory.
E3EC   A9 C6      LDA #$C6
E3EE   A0 00      LDY #$00        Unpack original number from
E3F0   20 7B DE   JSR $DE7B       $C6.
E3F3   A9 00      LDA #$00        Clear main FPA sign byte.
E3F5   85 D5      STA $D5
E3F7   A5 2D      LDA $2D
E3F9   20 03 E4   JSR $E403       Execute latter half of SIN
E3FC   A9 BD      LDA #$BD        routine - to get cosine.
E3FE   A0 00      LDY #$00        Unpack work FPA from $BD and
E400   4C E4 DD   JMP $DDE4       divide to get final result.
E403   48         PHA
E404   4C C4 E3   JMP $E3C4

E407   81 49 0F DA A2            Data for the trigonometric
E40C   83 49 0F DA A2            functions.
E411   7F 00 00 00 00
E416   05

E417   84 E6 1A 2D 1B
E41C   86 28 07 FB F8
E421   87 99 68 89 01
E426   87 23 35 DF E1
E42B   86 A5 5D E7 28
E430   83 49 0F DA A2
E435   A1 54 46 8F 13
E43A   8F 52 43 89 CD

E43F   A5 D5      LDA $D5         ATN
E441   48         PHA             Save sign byte of main FPA.
E442   10 03      BPL $E447       Sign is positive.
E444   20 71 E2   JSR $E271       Negate number.
E447   A5 D0      LDA $D0         Save exponent of main FPA.
E449   48         PHA
E44A   C9 81      CMP #$81
E44C   90 07      BCC $E455       Exponent is less than 1.
E44E   A9 81      LDA #$81        Unpack work FPA from $DC81 and
E450   A0 DC      LDY #$DC        divide into main FPA.
```

```
E452   20 E4 DD    JSR $DDE4
E455   A9 6F       LDA #$6F        Evaluate series using data
E457   A0 E4       LDY #$E4        from table at $E46F.
E459   20 FD E2    JSR $E2FD
E45C   68          PLA
E45D   C9 81       CMP #$81
E45F   90 07       BCC $E468       Exponent is less than 1.
E461   A9 07       LDA #$07
E463   A0 E4       LDY #$E4        Unpack work FPA from $E407 and
E465   20 0B DB    JSR $DB0B       subtract from main FPA.
E468   68          PLA
E469   10 03       BPL $E46E       Branch if positive.
E46B   4C 71 E2    JMP $E271       Negate number.
E46E   60          RTS             Exit.

E46F   0B 76 B3 83 BD              Data for ATN.
E474   D3 79 1E F4 A6
E479   F5 7B 83 FC B0
E47E   10

E47F   7C 0C 1F 67 CA
E484   7C DE 53 CB C1
E489   7D 14 64 70 4C
E48E   7D B7 EA 51 7A
E493   7D 63 30 88 7E
E498   7E 92 44 99 3A
E49D   7E 4C CC 91 C7
E4A2   7F AA AA AA 13
E4A7   81 00 00 00 00

E4AC   20 35 E7    JSR $E735       Get in sync with tape.
E4AF   20 C9 E6    JSR $E6C9       Read byte from tape.
E4B2   C9 24       CMP #$24
E4B4   D0 F9       BNE $E4AF       Get bytes until "$" is read.
E4B6   8E B1 02    STX $02B1
E4B9   A2 09       LDX #$09
E4BB   20 C9 E6    JSR $E6C9       Read byte.
E4BE   9D A7 02    STA $02A7,X     Save in header block.
E4C1   CA          DEX
E4C2   D0 F7       BNE $E4BB
E4C4   20 C9 E6    JSR $E6C9       Get byte.
E4C7   F0 0A       BEQ $E4D3       End of file name.
E4C9   E0 10       CPX #$10        Continue for up to 16 bytes.
E4CB   B0 F7       BCS $E4C4
E4CD   9D 93 02    STA $0293,X     Save chars of file name.
E4D0   E8          INX
E4D1   D0 F1       BNE $E4C4
E4D3   9D 93 02    STA $0293,X     Store end of file indicator.
E4D6   20 94 E5    JSR $E594       Print "Found" <filename>.
E4D9   20 90 E7    JSR $E790       Compare names of files.
E4DC   8A          TXA
E4DD   D0 CD       BNE $E4AC       Correct filename is not found.
E4DF   60          RTS             Exit.

E4E0   AD A9 02    LDA $02A9       LOAD/VERIFY DATA
E4E3   AC AA 02    LDY $02AA
E4E6   85 33       STA $33         Transfer pointer.
E4E8   84 34       STY $34
E4EA   A0 00       LDY #$00
E4EC   20 C9 E6    JSR $E6C9       Read byte from tape.
E4EF   AE 5B 02    LDX $025B
E4F2   D0 05       BNE $E4F9       VERIFY the data.
E4F4   91 33       STA ($33),Y     Store in memory.
E4F6   4C 05 E5    JMP $E505       Jump to increment pointers.
```

```
E4F9    D1 33       CMP ($33),Y     Compare data to verify it.
E4FB    F0 08       BEQ $E505       Data match made.
E4FD    EE 5C 02    INC $025C       Increment error counter.
E500    D0 03       BNE $E505
E502    EE 5D 02    INC $025D
E505    20 6C E5    JSR $E56C       Increment counters.
E508    90 E2       BCC $E4EC
E50A    60          RTS

E50B    10 07 53 65 61 72 63 68       Search
E513    69 6E 67 20 2E 2E 00 10     ing ..
E51B    07 4C 6F 61 64 69 6E 67      Loading
E523    20 2E 2E 00 0A 0D 45 72      ..   Er
E52B    72 6F 72 73 20 66 6F 75     rors Fou
E533    6E 64 0D 0A 00 10 07 46     nd     F
E53B    6F 75 6E 64 20 2E 2E 00     ound ..
E543    10 07 56 65 72 69 66 79       Verify
E54B    69 6E 67 20 2E 2E 00 20     ing ..
E553    56 65 72 69 66 79 20 65     Verify E
E55B    72 72 6F 72 73 20 64 65     rrors de
E563    74 65 63 74 65 64 0D 0A     tected
E56B    00

E56C    A5 33       LDA $33         Increment counter for loading
E56E    CD AB 02    CMP $02AB       or verifying data from tape.
E571    A5 34       LDA $34
E573    ED AC 02    SBC $02AC
E576    E6 33       INC $33
E578    D0 02       BNE $E57C       Compare pointer with final
E57A    E6 34       INC $34         pointer. C=1 if end reached.
E57C    60          RTS

E57D    A9 0B       LDA #$0B        Print "Searching ..".
E57F    A0 E5       LDY #$E5
E581    20 EA E5    JSR $E5EA
E584    60          RTS

E585    A9 45       LDA #$45        Print "Saving ".
E587    A0 E6       LDY #$E6
E589    20 EA E5    JSR $E5EA
E58C    A9 7F       LDA #$7F        Print <filename>.
E58E    A0 02       LDY #$02
E590    20 B6 E5    JSR $E5B6
E593    60          RTS

E594    A9 38       LDA #$38        Print "Found " <filename>.
E596    A0 E5       LDY #$E5
E598    4C AB E5    JMP $E5AB
E59B    AD 5B 02    LDA $025B
E59E    D0 07       BNE $E5A7       VERIFYing data.
E5A0    A9 1A       LDA #$1A
E5A2    A0 E5       LDY #$E5
E5A4    4C AB E5    JMP $E5AB       Print "Loading .. ".
E5A7    A9 43       LDA #$43
E5A9    A0 E5       LDY #$E5
E5AB    20 EA E5    JSR $E5EA       Print "Verifying .. ".
E5AE    A9 93       LDA #$93
E5B0    A0 02       LDY #$02
E5B2    20 B6 E5    JSR $E5B6       Print <filename>.
E5B5    60          RTS

E5B6    20 65 F8    JSR $F865       Print message to screen.
E5B9    E8          INX
E5BA    A0 00       LDY #$00        Set end of mesage indicator.
```

```
E5BC   8C 5F 02    STY $025F
E5BF   AD AE 02    LDA $02AE       Using the table below, load up
E5C2   F0 13       BEQ $E5D7       the character appropiate to
E5C4   C8          INY             type of file being used. Then
E5C5   2C AE 02    BIT $02AE       print it to screen after the
E5C8   30 0D       BMI $E5D7       other message.
E5CA   C8          INY
E5CB   2C AF 02    BIT $02AF
E5CE   30 07       BMI $E5D7
E5D0   C8          INY
E5D1   2C B0 02    BIT $02B0
E5D4   30 01       BMI $E5D7
E5D6   C8          INY
E5D7   B9 E5 E5    LDA $E5E5,Y
E5DA   8D 5E 02    STA $025E
E5DD   A9 5E       LDA #$5E
E5DF   A0 02       LDY #$02
E5E1   20 65 F8    JSR $F865       Print chars at $025E.
E5E4   60          RTS

E5E5   42 43 53 49 52              B C S I R

E5EA   20 F5 E5    JSR $E5F5       Clear status line of screen
E5ED   A2 00       LDX #$00        and then print message to
E5EF   20 65 F8    JSR $F865       screen.
E5F2   E8          INX
E5F3   E8          INX
E5F4   60          RTS

E5F5   48          PHA             Clear cassette status
E5F6   AD 1F 02    LDA $021F       message.
E5F9   D0 0A       BNE $E605       In hires mode.
E5FB   A2 22       LDX #$22
E5FD   A9 10       LDA #$10
E5FF   9D 80 BB    STA $BB80,X     Write black paper to each
E602   CA          DEX             column of status line being
E603   10 FA       BPL $E5FF       cleared.
E605   68          PLA
E606   60          RTS

E607   20 5A E7    JSR $E75A       OUTPUT FILE HEADER.
E60A   A9 24       LDA #$24        Output tape leader and then
E60C   20 5E E6    JSR $E65E       a $ character.
E60F   A2 09       LDX #$09
E611   BD A7 02    LDA $02A7,X     Output header information.
E614   20 5E E6    JSR $E65E
E617   CA          DEX
E618   D0 F7       BNE $E611
E61A   BD 7F 02    LDA $027F,X     Output filename with a null
E61D   F0 06       BEQ $E625       afer it.
E61F   20 5E E6    JSR $E65E
E622   E8          INX
E623   D0 F5       BNE $E61A
E625   20 5E E6    JSR $E65E
E628   A2 00       LDX #$00
E62A   CA          DEX             Wait about 1.3mS.
E62B   D0 FD       BNE $E62A
E62D   60          RTS             Exit

E62E   AD A9 02    LDA $02A9
E631   AC AA 02    LDY $02AA       Transfer start of DATA.
E634   85 33       STA $33
E636   84 34       STY $34
E638   A0 00       LDY #$00
```

```
E63A   B1 33       LDA ($33),Y     Load next byte.
E63C   20 5E E6    JSR $E65E       Output next byte.
E63F   20 6C E5    JSR $E56C       Increment pointers.
E642   90 F6       BCC $E63A       More to do.
E644   60          RTS

E645   10 07 53 61 76 69           Data for "Saving"
E64B   6E 67 20 2E 2E 00

E651   AD B1 02    LDA $02B1       Print out string after " if
E654   F0 07       BEQ $E65D       there was an error in format.
E656   A9 27       LDA #$27
E658   A0 E5       LDY #$E5
E65A   20 B0 CC    JSR $CCB0
E65D   60          RTS

E65E   85 2F       STA $2F         OUTPUT BYTE TO CASSETTE.
E660   8A          TXA             $2F holds byte going out.
E661   48          PHA
E662   98          TYA
E663   48          PHA
E664   20 C0 E6    JSR $E6C0       Wait until timer 1 has counted
E667   18          CLC             out.
E668   A0 09       LDY #$09
E66A   A9 00       LDA #$00
E66C   F0 06       BEQ $E674
E66E   46 2F       LSR $2F
E670   08          PHP             Shift out the byte to be sent
E671   69 00       ADC #$00        a bit at a time until whole
E673   28          PLP             byte is done.
E674   20 8B E6    JSR $E68B       Output bit.
E677   88          DEY
E678   D0 F4       BNE $E66E
E67A   49 01       EOR #$01
E67C   4A          LSR A
E67D   A0 04       LDY #$04        Output 4 extra bits of zero
E67F   20 8B E6    JSR $E68B       at end of each byte.
E682   38          SEC
E683   88          DEY
E684   D0 F9       BNE $E67F
E686   68          PLA
E687   A8          TAY
E688   68          PLA
E689   AA          TAX
E68A   60          RTS
E68B   48          PHA             Output bit to tape.
E68C   08          PHP
E68D   AD 4D 02    LDA $024D
E690   D0 0A       BNE $E69C       Slow tape speed.
E692   38          SEC
E693   20 B2 E6    JSR $E6B2       Set timer 1 and wait until
E696   28          PLP             timeout twice so whole cycle
E697   20 B2 E6    JSR $E6B2       is output on cassette line -
E69A   68          PLA             PB7.
E69B   60          RTS

E69C   20 B2 E6    JSR $E6B2       Slow tape speed - wait an
E69F   A2 0F       LDX #$0F        extra 7 times as long for
E6A1   28          PLP             cycle.
E6A2   B0 02       BCS $E6A6
E6A4   A2 07       LDX #$07
E6A6   20 AB E6    JSR $E6AB
E6A9   68          PLA
E6AA   60          RTS
```

```
E6AB   20 C0 E6   JSR $E6C0       Wait until timer 1 has counted
E6AE   CA         DEX             out X times over.
E6AF   D0 FA      BNE $E6AB
E6B1   60         RTS

E6B2   A9 D0      LDA #$D0        Set timer 1 and wait for a
E6B4   A2 00      LDX #$00        time out. No interrupt is
E6B6   B0 02      BCS $E6BA       generated, the interrupt flag
E6B8   0A         ASL A           register is polled until time
E6B9   E8         INX             out.
E6BA   8D 06 03   STA $0306
E6BD   8E 07 03   STX $0307
E6C0   AD 04 03   LDA $0304
E6C3   2C 0D 03   BIT $030D
E6C6   50 FB      BVC $E6C3
E6C8   60         RTS

E6C9   98         TYA             READ BYTE FROM TAPE.
E6CA   48         PHA             The byte is generated by
E6CB   8A         TXA             shifting a series of bits into
E6CC   48         PHA             $2F. This routine does a
E6CD   20 1C E7   JSR $E71C       series of timings using timer
E6D0   20 1C E7   JSR $E71C       2 to get each bit of data. 8
E6D3   B0 FB      BCS $E6D0       bits are then compiled into
E6D5   20 FF E6   JSR $E6FF       the next byte.
E6D8   B0 16      BCS $E6F0
E6DA   A9 00      LDA #$00
E6DC   A0 08      LDY #$08
E6DE   20 FC E6   JSR $E6FC
E6E1   08         PHP
E6E2   66 2F      ROR $2F
E6E4   28         PLP
E6E5   69 00      ADC #$00
E6E7   88         DEY
E6E8   D0 F4      BNE $E6DE
E6EA   20 FC E6   JSR $E6FC
E6ED   E9 00      SBC #$00
E6EF   4A         LSR A
E6F0   90 03      BCC $E6F5
E6F2   2E B1 02   ROL $02B1
E6F5   68         PLA
E6F6   AA         TAX
E6F7   68         PLA
E6F8   A8         TAY
E6F9   A5 2F      LDA $2F
E6FB   60         RTS

E6FC   20 1C E7   JSR $E71C       Depending whether the
E6FF   48         PHA             cassette load is slow or fast
E700   AD 4D 02   LDA $024D       , this routine waits for a
E703   F0 15      BEQ $E71A       series of active pulses from
E705   20 1C E7   JSR $E71C       the cassette input.
E708   A2 02      LDX #$02
E70A   90 02      BCC $E70E
E70C   A2 06      LDX #$06
E70E   A9 00      LDA #$00
E710   20 1C E7   JSR $E71C
E713   69 00      ADC #$00
E715   CA         DEX
E716   D0 F8      BNE $E710
E718   C9 04      CMP #$04
E71A   68         PLA
E71B   60         RTS
```

```
E71C   48          PHA             Cassette input timing.
E71D   AD 00 03    LDA $0300       This routine waits for an
E720   AD 0D 03    LDA $030D       active transition of the
E723   29 10       AND #$10        cassette input line (CB1 of
E725   F0 F9       BEQ $E720       6522). The time taken to
E727   AD 09 03    LDA $0309       receive it is measured using
E72A   48          PHA             timer 2 of 6522.
E72B   A9 FF       LDA #$FF
E72D   8D 09 03    STA $0309
E730   68          PLA
E731   C9 FE       CMP #$FE
E733   68          PLA
E734   60          RTS

E735   20 FC E6    JSR $E6FC       GET IN SYNC WITH CASSETTE DATA
E738   66 2F       ROR $2F         Get bits in until byte holds
E73A   A9 16       LDA #$16        #16 - the value of the bytes
E73C   C5 2F       CMP $2F         sent out as tape leader.
E73E   D0 F5       BNE $E735
E740   AD 4D 02    LDA $024D
E743   F0 08       BEQ $E74D       Fast load (2400 baud).
E745   20 1C E7    JSR $E71C
E748   20 1C E7    JSR $E71C
E74B   B0 FB       BCS $E748
E74D   A2 03       LDX #$03        Read 3 successive bytes of #16
E74F   20 C9 E6    JSR $E6C9       from cassette. If any byte is
E752   C9 16       CMP #$16        not #16 then start again.
E754   D0 DF       BNE $E735
E756   CA          DEX
E757   D0 F6       BNE $E74F
E759   60          RTS

E75A   A2 02       LDX #$02        OUTPUT TAPE LEADER.
E75C   A0 03       LDY #$03
E75E   A9 16       LDA #$16        Use X and Y to count out 259
E760   20 5E E6    JSR $E65E       bytes of #16 that are sent out
E763   88          DEY             as tape leader.
E764   D0 F8       BNE $E75E
E766   CA          DEX
E767   D0 F5       BNE $E75E
E769   60          RTS

E76A   20 1A EE    JSR $EE1A       SET 6522 FOR CASSETTE SYSTEM.
E76D   A0 06       LDY #$06        Disable timer 1 interrupts and
E76F   78          SEI             then load up the 6522's
E770   BE 82 E7    LDX $E782,Y     registers with data in the
E773   B9 89 E7    LDA $E789,Y     table below.
E776   9D 00 03    STA $0300,X
E779   88          DEY
E77A   10 F4       BPL $E770
E77C   A9 40       LDA #$40
E77E   8D 00 03    STA $0300
E781   60          RTS

E782   05 04 0B 02 0C 08 0E       List of registers and data for
E789   00 D0 C0 FF 10 F4 7F       the routine above.

E790   A0 00       LDY #$00        Routine to compare the names
E792   A2 00       LDX #$00        of the file wanted and that
E794   AD 7F 02    LDA $027F       whose header has just been
E797   F0 15       BEQ $E7AE       loaded.
E799   B9 7F 02    LDA $027F,Y
E79C   D9 93 02    CMP $0293,Y
```

```
E79F   F0 01      BEQ $E7A2
E7A1   E8         INX
E7A2   99 93 02   STA $0293,Y
E7A5   C8         INY
E7A6   C0 11      CPY #$11
E7A8   B0 04      BCS $E7AE
E7AA   48         PHA
E7AB   68         PLA
E7AC   D0 EB      BNE $E799
E7AE   60         RTS

E7AF   4C 70 D0   JMP $D070       Print "SYNTAX ERROR".

E7B2   A9 00      LDA #$00        CHECK CSAVE/CLOAD PARAMETERS
E7B4   8D 4D 02   STA $024D       Default Speed.
E7B7   8D AD 02   STA $02AD       Reset AUTO flag.
E7BA   8D AE 02   STA $02AE       Reset file type to Basic.
E7BD   8D 5B 02   STA $025B       Clear VERIFY flag.
E7C0   8D 5A 02   STA $025A       Clear JOIN fig.
E7C3   8D 5C 02   STA $025C       Clear error counter LSB.
E7C6   8D 5D 02   STA $025D       Clear error counter MSB.
E7C9   8D B1 02   STA $02B1       Clear error in file format.
E7CC   20 17 CF   JSR $CF17       Evaluate expression.
E7CF   24 28      BIT $28
E7D1   10 DC      BPL $E7AF       Error if not string type.
E7D3   20 D0 D7   JSR $D7D0       Set up string in main FPA.
E7D6   AA         TAX
E7D7   A0 00      LDY #$00
E7D9   E8         INX
E7DA   CA         DEX
E7DB   F0 0A      BEQ $E7E7
E7DD   B1 91      LDA ($91),Y     Transfer name of file to be
E7DF   99 7F 02   STA $027F,Y     loaded, saved or verified.
E7E2   C8         INY
E7E3   C0 10      CPY #$10
E7E5   D0 F3      BNE $E7DA
E7E7   A9 00      LDA #$00
E7E9   99 7F 02   STA $027F,Y     End filename with a null.
E7EC   20 E8 00   JSR $00E8       Clear spaces in text.
E7EF   F0 61      BEQ $E852       End of statement.
E7F1   C9 2C      CMP #$2C        Error if next character is not
E7F3   D0 BA      BNE $E7AF       a comma.
E7F5   20 E2 00   JSR $00E2       Clear spaces.
E7F8   F0 58      BEQ $E852       End of statement.
E7FA   C9 2C      CMP #$2C        Get next character if comma
E7FC   F0 F7      BEQ $E7F5       found.
E7FE   C9 C7      CMP #$C7
E800   D0 05      BNE $E807       'AUTO' token not found.
E802   8D AD 02   STA $02AD       Set AUTO indicator.
E805   B0 EE      BCS $E7F5
E807   C9 53      CMP #$53
E809   D0 05      BNE $E810       No 'S' for slow tape speed.
E80B   8D 4D 02   STA $024D       Set slow tape speed.
E80E   B0 E5      BCS $E7F5
E810   C9 56      CMP #$56
E812   D0 05      BNE $E819       No 'V' for file verify.
E814   8D 5B 02   STA $025B       Set verify flag.
E817   B0 DC      BCS $E7F5
E819   C9 4A      CMP #$4A
E81B   D0 05      BNE $E822       No 'J' for JOINing files.
E81D   8D 5A 02   STA $025A       Set JOIN flag.
E820   B0 D3      BCS $E7F5
E822   C9 41      CMP #$41
E824   F0 04      BEQ $E82A       'A' found - machine code prog.
```

```
E826   C9 45       CMP #$45        No 'E' to indicate end of
E828   D0 47       BNE $E871       machine code program
E82A   85 0E       STA $0E         Save A/E - start/end indic'r.
E82C   20 E2 00    JSR $00E2       Clear space.
E82F   A2 80       LDX #$80        Inhibit AUTO loading of
E831   8E AE 02    STX $02AE       machine code programs.
E834   20 53 E8    JSR $E853       Get numeric integer.
E837   A5 33       LDA $33
E839   A4 34       LDY $34         Transfer integer to pointers
E83B   A6 0E       LDX $0E         in page 2 depending whether it
E83D   E0 41       CPX #$41        it is the start or end address
E83F   D0 08       BNE $E849       of the machine code routine or
E841   8D A9 02    STA $02A9       block of data.
E844   8C AA 02    STY $02AA
E847   B0 A3       BCS $E7EC
E849   8D AB 02    STA $02AB
E84C   8C AC 02    STY $02AC
E84F   4C EC E7    JMP $E7EC       Jump back for more input.
E852   60          RTS             Exit.

E853   20 03 CF    JSR $CF03       Get numeric expression and
E856   20 22 D9    JSR $D922       convert it into integer at $33
E859   18          CLC             and $34.
E85A   60          RTS

E85B   08          PHP             CLOAD
E85C   20 B2 E7    JSR $E7B2       Set up variables.
E85F   AD AD 02    LDA $02AD
E862   0D AE 02    ORA $02AE       Error if trying the AUTO load
E865   D0 0A       BNE $E871       of a non Basic program.
E867   AD 5A 02    LDA $025A       Give error also if JOIN and
E86A   F0 08       BEQ $E874       VERIFY both set.
E86C   AD 5B 02    LDA $025B
E86F   F0 03       BEQ $E874
E871   4C 70 D0    JMP $D070       Print "SYNTAX ERROR".
E874   20 6A E7    JSR $E76A       Set 6522.
E877   20 7D E5    JSR $E57D       Print "SEARCHING".
E87A   20 AC E4    JSR $E4AC       Read file header.
E87D   2C AE 02    BIT $02AE
E880   70 F8       BVS $E87A
E882   AD 5A 02    LDA $025A
E885   F0 2C       BEQ $E8B3
E887   AD AE 02    LDA $02AE
E88A   D0 EE       BNE $E87A
E88C   A5 9C       LDA $9C
E88E   A4 9D       LDY $9D
E890   38          SEC
E891   E9 02       SBC #$02
E893   B0 01       BCS $E896
E895   88          DEY
E896   8D A9 02    STA $02A9
E899   8C AA 02    STY $02AA
E89C   38          SEC             Set pointers to the amount of
E89D   E5 9A       SBC $9A         data and where it is to be
E89F   AA          TAX             loaded.
E8A0   98          TYA
E8A1   E5 9B       SBC $9B
E8A3   A8          TAY
E8A4   18          CLC
E8A5   8A          TXA
E8A6   6D AB 02    ADC $02AB
E8A9   8D AB 02    STA $02AB
E8AC   98          TYA
E8AD   6D AC 02    ADC $02AC
```

```
E8B0   8D AC 02   STA $02AC
E8B3   20 9B E5   JSR $E59B       "Loading/Verifying" filename.
E8B6   20 E0 E4   JSR $E4E0       Load/verify data from tape.
E8B9   20 3D E9   JSR $E93D       Reset cassette status.
E8BC   28         PLP
E8BD   AD 5B 02   LDA $025B
E8C0   F0 11      BEQ $E8D3       Not VERIFYing data.
E8C2   AE 5C 02   LDX $025C       Print number of verify errors.
E8C5   AD 5D 02   LDA $025D
E8C8   20 C5 E0   JSR $E0C5
E8CB   A9 52      LDA #$52
E8CD   A0 E5      LDY #$E5        Print "Verify errors
E8CF   20 B0 CC   JSR $CCB0       detected".
E8D2   60         RTS

E8D3   20 51 E6   JSR $E651       Print filename if there is a
E8D6   AD AE 02   LDA $02AE       format error.
E8D9   F0 0E      BEQ $E8E9
E8DB   AD AD 02   LDA $02AD       Jump to start of machine code
E8DE   F0 08      BEQ $E8E8       program if correct file type
E8E0   AD B1 02   LDA $02B1       and there are no loading errors.
E8E3   EA         NOP
E8E4   EA         NOP
E8E5   6C A9 02   JMP ($02A9)
E8E8   60         RTS

E8E9   AE AB 02   LDX $02AB       Transfer end of Basic to zero
E8EC   AD AC 02   LDA $02AC       page pointer.
E8EF   86 9C      STX $9C
E8F1   85 9D      STA $9D
E8F3   20 5F C5   JSR $C55F       Set up line link pointers.
E8F6   AD AD 02   LDA $02AD
E8F9   F0 08      BEQ $E903       Not AUTO run.
E8FB   AD B1 02   LDA $02B1
E8FE   EA         NOP
E8FF   EA         NOP
E900   4C 08 C7   JMP $C708       Jump to CLEAR & run program.

E903   20 08 C7   JSR $C708       CLEAR.
E906   4C A8 C4   JMP $C4A8       Restart Basic.

E909   A5 9A      LDA $9A         CSAVE
E90B   A4 9B      LDY $9B         Transfer Start Basic pointer -
E90D   8D A9 02   STA $02A9       start of data.
E910   8C AA 02   STY $02AA
E913   A5 9C      LDA $9C         Transfer End Basic pointer -
E915   A4 9D      LDY $9D         end of data.
E917   8D AB 02   STA $02AB
E91A   8C AC 02   STY $02AC
E91D   08         PHP
E91E   20 B2 E7   JSR $E7B2       Process rest of statement.
E921   AD 5A 02   LDA $025A       Give error if trying to JOIN
E924   0D 5B 02   ORA $025B       and VERIFY program together.
E927   F0 03      BEQ $E92C
E929   4C 70 D0   JMP $D070       Print "SYNTAX ERROR".

E92C   20 6A E7   JSR $E76A       Set 6522 for cassette routine.
E92F   20 85 E5   JSR $E585       Print "Saving"
E932   20 07 E6   JSR $E607       Output file header.
E935   20 2E E6   JSR $E62E       Output data to cassette.
E938   20 3D E9   JSR $E93D       Reset cassette status.
E93B   28         PLP
E93C   60         RTS
```

```
E93D   20 F5 E5   JSR $E5F5       Reset cassette status by
E940   20 AA F9   JSR $F9AA       clearing status line, reset
E943   4C E0 ED   JMP $EDE0       6522 and the 16 bit counters.

E946   20 53 E8   JSR $E853       CALL Evaluate numeric
E949   6C 33 00   JMP ($0033)     integer and jump through it.

E94C   A2 00      LDX #$00        Get hex number into A and Y.
E94E   86 0C      STX $0C         Set initial value to 0.
E950   86 0D      STX $0D
E952   F0 13      BEQ $E967
E954   A2 03      LDX #$03        Set bit counter.
E956   0A         ASL A           Shift digit into top nibble.
E957   0A         ASL A
E958   0A         ASL A
E959   0A         ASL A
E95A   0A         ASL A
E95B   26 0C      ROL $0C         Shift bit into number.
E95D   26 0D      ROL $0D
E95F   90 03      BCC $E964
E961   4C 39 DC   JMP $DC39       "OVERFLOW ERROR".
E964   CA         DEX             If another bit then do
E965   10 F3      BPL $E95A       another shift.
E967   20 E2 00   JSR $00E2       Get next char.
E96A   C9 80      CMP #$80
E96C   B0 0E      BCS $E97C       Exit if token.
E96E   09 80      ORA #$80
E970   49 B0      EOR #$B0        Reduce to digit range.
E972   C9 0A      CMP #$0A
E974   90 DE      BCC $E954       If 0-9 then put in hex digit.
E976   69 88      ADC #$88
E978   C9 FA      CMP #$FA        If A-F then put in hex digit.
E97A   B0 D8      BCS $E954
E97C   A5 0D      LDA $0D         Exit with number in A (MSB)
E97E   A4 0C      LDY $0C         and Y (LSB).
E980   60         RTS

E981   20 4C E9   JSR $E94C       Get hex number and put it into
E984   4C 40 DF   JMP $DF40       main Floating Point Acc.

E987   08         PHP             STORE
E988   20 57 EA   JSR $EA57       Process rest of statement.
E98B   A9 40      LDA #$40        Set type of data.
E98D   8D AE 02   STA $02AE
E990   A5 28      LDA $28         Set type of Array.
E992   8D AF 02   STA $02AF
E995   A5 29      LDA $29
E997   8D B0 02   STA $02B0
E99A   20 85 E5   JSR $E585       Print "Saving".
E99D   20 07 E6   JSR $E607       Output file header.
E9A0   20 9E EA   JSR $EA9E
E9A3   20 2E E6   JSR $E62E       Transfer data.
E9A6   24 28      BIT $28
E9A8   10 22      BPL $E9CC       Not string type.
E9AA   A0 00      LDY #$00
E9AC   B1 0C      LDA ($0C),Y
E9AE   F0 17      BEQ $E9C7
E9B0   AA         TAX
E9B1   A0 02      LDY #$02
E9B3   B1 0C      LDA ($0C),Y
E9B5   99 D0 00   STA $00D0,Y
E9B8   88         DEY
E9B9   D0 F8      BNE $E9B3
E9BB   E8         INX
```

```
E9BC   CA          DEX
E9BD   F0 08       BEQ $E9C7
E9BF   B1 D1       LDA ($D1),Y     Output next string from array
E9C1   20 5E E6    JSR $E65E       to cassette.
E9C4   C8          INY
E9C5   D0 F5       BNE $E9BC
E9C7   20 42 EA    JSR $EA42       Advance pointer to next string
E9CA   90 DE       BCC $E9AA       pointer. Branch if more.
E9CC   20 3D E9    JSR $E93D       Reset cassette status.
E9CF   28          PLP
E9D0   60          RTS

E9D1   20 50 D6    JSR $D650       RECALL Attempt Garbage
E9D4   08          PHP             Collection.
E9D5   20 57 EA    JSR $EA57       Get parameters & test syntax.
E9D8   20 7D E5    JSR $E57D       Print "Searching".
E9DB   20 AC E4    JSR $E4AC       Get in sync with tape.
E9DE   2C AE 02    BIT $02AE
E9E1   50 F8       BVC $E9DB
E9E3   AD AF 02    LDA $02AF
E9E6   45 28       EOR $28         String array flags do not
E9E8   D0 F1       BNE $E9DB       match.
E9EA   AD B0 02    LDA $02B0
E9ED   45 29       EOR $29         Integer array flags do not
E9EF   D0 EA       BNE $E9DB       match.
E9F1   20 9B E5    JSR $E59B       "Loading/Verifying" filename.
E9F4   A0 02       LDY #$02
E9F6   B1 CE       LDA ($CE),Y
E9F8   CD A9 02    CMP $02A9
E9FB   C8          INY
E9FC   B1 CE       LDA ($CE),Y     Test if there is enough space
E9FE   ED AA 02    SBC $02AA       to load in array.
EA01   B0 06       BCS $EA09
EA03   20 3D E9    JSR $E93D       Reset cassette status.
EA06   4C 7C C4    JMP $C47C       Print "OUT 0F MEMORY ERROR".
EA09   20 9E EA    JSR $EA9E
EA0C   20 E0 E4    JSR $E4E0       Load/Verify the data.
EA0F   24 28       BIT $28
EA11   10 27       BPL $EA3A
EA13   A0 00       LDY #$00
EA15   B1 0C       LDA ($0C),Y
EA17   F0 1C       BEQ $EA35
EA19   20 AB D5    JSR $D5AB       Get space for string.
EA1C   A0 00       LDY #$00
EA1E   AA          TAX
EA1F   E8          INX
EA20   CA          DEX
EA21   F0 08       BEQ $EA2B
EA23   20 C9 E6    JSR $E6C9       Read and save next string from
EA26   91 D1       STA ($D1),Y     cassette and put it into the
EA28   C8          INY             array.
EA29   D0 F5       BNE $EA20
EA2B   A0 02       LDY #$02
EA2D   B9 D0 00    LDA $00D0,Y
EA30   91 0C       STA ($0C),Y
EA32   88          DEY
EA33   D0 F8       BNE $EA2D
EA35   20 42 EA    JSR $EA42       Advance pointer in array to
EA38   90 D9       BCC $EA13       next string pointer.
EA3A   20 3D E9    JSR $E93D       Reset 6522.
EA3D   20 51 E6    JSR $E651       Print out string name if there
EA40   28          PLP             is a format error.
EA41   60          RTS
```

```
EA42   18          CLC             Advance pointer at $0C/$0D to
EA43   A9 03       LDA #$03        point to next string pointer
EA45   65 0C       ADC $0C         in array and test if all
EA47   85 0C       STA $0C         loaded yet.
EA49   90 02       BCC $EA4D
EA4B   E6 0D       INC $0D
EA4D   A8          TAY
EA4E   A5 0D       LDA $0D
EA50   CC AB 02    CPY $02AB
EA53   ED AC 02    SBC $02AC
EA56   60          RTS             C=0 if more strings to load.

EA57   A9 40       LDA #$40        Set STORE/RECALL flag.
EA59   85 2B       STA $2B
EA5B   20 88 D1    JSR $D188       Get variable from text.
EA5E   A9 00       LDA #$00        Clear STORE/RECALL flag.
EA60   85 2B       STA $2B
EA62   A0 03       LDY #$03        Load $02A9/$02AA with start of
EA64   B1 CE       LDA ($CE),Y     data.
EA66   8D AA 02    STA $02AA
EA69   88          DEY
EA6A   B1 CE       LDA ($CE),Y
EA6C   8D A9 02    STA $02A9
EA6F   D0 03       BNE $EA74
EA71   CE AA 02    DEC $02AA       Decrement address.
EA74   CE A9 02    DEC $02A9
EA77   20 65 D0    JSR $D065       Test comma.
EA7A   A5 29       LDA $29         Save Variable type bytes.
EA7C   48          PHA
EA7D   A5 28       LDA $28
EA7F   48          PHA
EA80   20 B2 E7    JSR $E7B2       Process syntax of rest of
EA83   68          PLA             command.
EA84   85 28       STA $28
EA86   68          PLA             Restore variable type bytes.
EA87   85 29       STA $29
EA89   AD 5B 02    LDA $025B       Ensure that incorrect
EA8C   0D AD 02    ORA $02AD       combinations of join, verify,
EA8F   0D AE 02    ORA $02AE       AUTO are not allowed - can
EA92   0D 5A 02    ORA $025A       have default values.
EA95   F0 03       BEQ $EA9A
EA97   4C 70 D0    JMP $D070       Print "SYNTAX ERROR".
EA9A   20 6A E7    JSR $E76A       Set 6522 for cassette system.
EA9D   60          RTS

EA9E   18          CLC
EA9F   A5 CE       LDA $CE
EAA1   6D A9 02    ADC $02A9
EAA4   8D AB 02    STA $02AB
EAA7   A5 CF       LDA $CF
EAA9   6D AA 02    ADC $02AA
EAAC   8D AC 02    STA $02AC
EAAF   A0 04       LDY #$04
EAB1   B1 CE       LDA ($CE),Y
EAB3   20 88 D2    JSR $D288
EAB6   8D A9 02    STA $02A9
EAB9   8C AA 02    STY $02AA
EABC   85 0C       STA $0C
EABE   84 0D       STY $0D
EAC0   60          RTS

EAC1   3F FB 17 FC CF FB C7 F0     This table holds the start
EAC9   FC F0 0F F1 7E F3 1C F1     addresses less 1 for the
EAD1   67 F2 2C F1 03 F2 0F F2     sound and hires commands. They
```

```
EAD9    03 04 04 03 03 03 02 01     are in order of token value.
EAE1    03 03 01 01 00 00 00 00     The second part holds data
EAE9    01 01 00 00 00 00 00        associated with each routine.

EAF0    AD C0 02    LDA $02C0       Entry point for the hires
EAF3    29 01       AND #$01        screen commands. Test that one
EAF5    D0 05       BNE $EAFC       is in hires first- else print
EAF7    A2 A3       LDX #$A3        DISP TYPE MISMATCH.
EAF9    4C 7E C4    JMP $C47E
EAFC    C0 4E       CPY #$4E        Entry point for sound commands
EAFE    B0 03       BCS $EB03       No need to test if in hires.
EB00    4C 70 D0    JMP $D070       Test that tokens are in
EB03    C0 66       CPY #$66        correct range - else error.
EB05    B0 F9       BCS $EB00
EB07    98          TYA             Use the token value to look
EB08    38          SEC             up the start address of the
EB09    E9 4E       SBC #$4E        appropriate routine. The
EB0B    A8          TAY             address -1 is used as it is
EB0C    B9 C2 EA    LDA $EAC2,Y     saved on the stack and an
EB0F    48          PHA             RTS is done which increments
EB10    B9 C1 EA    LDA $EAC1,Y     the address pulled of stack.
EB13    48          PHA
EB14    98          TYA             Halve the value in Y and use
EB15    4A          LSR A           it to load the number of
EB16    A8          TAY             parameters for each command
EB17    B9 D9 EA    LDA $EAD9,Y     and whether the hires cursor
EB1A    48          PHA             is to be moved relative to its
EB1B    B9 E5 EA    LDA $EAE5,Y     current position or not
EB1E    8D C3 02    STA $02C3       respectively.
EB21    A9 00       LDA #$00
EB23    8D F0 02    STA $02F0
EB26    20 03 CF    JSR $CF03       Evaluate next argument.
EB29    AD C3 02    LDA $02C3
EB2C    D0 06       BNE $EB34
EB2E    20 22 D9    JSR $D922       Convert Floating point acc'r
EB31    4C 3B EB    JMP $EB3B       to integer.
EB34    A5 D0       LDA $D0
EB36    C9 90       CMP #$90
EB38    20 2A D9    JSR $D92A
EB3B    AC F0 02    LDY $02F0       Place the next argument into
EB3E    A5 33       LDA $33         its correct place in the
EB40    99 E1 02    STA $02E1,Y     parameter block starting at
EB43    A5 34       LDA $34         #02E1.
EB45    99 E2 02    STA $02E2,Y
EB48    C8          INY             #2F0 now points just beyond
EB49    C8          INY             the last parameter placed in
EB4A    8C F0 02    STY $02F0       block at #2E1.
EB4D    68          PLA
EB4E    A8          TAY             Decrement the counter of the
EB4F    88          DEY             number of parameters to be
EB50    F0 08       BEQ $EB5A       evaluated. Continue evaluation
EB52    98          TYA             until the appropriate number
EB53    48          PHA             is done.
EB54    20 65 D0    JSR $D065       Search for comma, return only
EB57    4C 26 EB    JMP $EB26       if found. Continue arg. eval.
EB5A    A9 00       LDA #$00        Initialise the error status.
EB5C    8D E0 02    STA $02E0
EB5F    68          PLA             This section inserts on to the
EB60    AA          TAX             stack an address such that
EB61    68          PLA             when the appropriate sound /
EB62    A8          TAY             graphics command is finished,
EB63    A9 EB       LDA #$EB        the next RTS instruction will
EB65    48          PHA             take the program to a routine
EB66    A9 6D       LDA #$6D        that checks the error status
```

```
EB68   48          PHA             of #02E0.
EB69   98          TYA
EB6A   48          PHA
EB6B   8A          TXA
EB6C   48          PHA             The RTS is used as a means
EB6D   60          RTS             of doing an indirect jump.

EB6E   A9 01       LDA #$01        If contents of #02E0 is not
EB70   2C E0 02    BIT $02E0       zero then print ILLEGAL
EB73   F0 F8       BEQ $EB6D       QUANTITY ERROR.
EB75   4C 36 D3    JMP $D336
EB78   AD DF 02    LDA $02DF       This routine checks whether a
EB7B   10 0B       BPL $EB88       new key is ready to be
EB7D   08          PHP             processed. If there is, the
EB7E   29 7F       AND #$7F        Ascii char for it is loaded
EB80   48          PHA             into A and #02DF cleared.
EB81   A9 00       LDA #$00
EB83   8D DF 02    STA $02DF       If N=0 at exit then no new
EB86   68          PLA             key has been received.
EB87   28          PLP
EB88   60          RTS

EB89   C4 9D       CPY $9D         Test if new Himem is not
EB8B   B0 02       BCS $EB8F       below end of Basic pointer.
EB8D   38          SEC             C=1 if Himem is too low.
EB8E   60          RTS

EB8F   D0 06       BNE $EB97       Test if low half of new
EB91   C5 9C       CMP $9C         Himem is below end Basic.
EB93   90 F9       BCC $EB8E       The branches are in error,
EB95   F0 F7       BEQ $EB8E       they should go to #EB8D !
EB97   20 B5 EB    JSR $EBB5       Test if Hires screen could be
EB9A   90 F2       BCC $EB8E       affected and branch if not.
EB9C   AA          TAX
EB9D   AD C0 02    LDA $02C0
EBA0   29 02       AND #$02        An error will be given if
EBA2   08          PHP             one tries to put Himem beyond
EBA3   8A          TXA             character sets when in hires
EBA4   28          PLP             mode.
EBA5   D0 E6       BNE $EB8D
EBA7   98          TYA             This section tests whether
EBA8   48          PHA             the character sets of the text
EBA9   38          SEC             mode would be lower than the
EBAA   E9 1C       SBC #$1C        new Himem. The appropriate
EBAC   A8          TAY             value of the C flag is left in
EBAD   8A          TXA             the status register at the end
EBAE   20 B5 EB    JSR $EBB5       of the routine.
EBB1   68          PLA
EBB2   A8          TAY
EBB3   8A          TXA
EBB4   60          RTS

EBB5   CC C2 02    CPY $02C2       This routine tests whether the
EBB8   90 02       BCC $EBBC       address in A (low) and Y
EBBA   F0 01       BEQ $EBBD       (high) is greater than that of
EBBC   60          RTS             the character sets in hires
EBBD   CD C1 02    CMP $02C1       mode. C is set if A,Y are
EBC0   60          RTS             greater.

EBC1   AC C2 02    LDY $02C2       This routine loads A (low)
EBC4   AD C1 02    LDA $02C1       and Y (high) with the address
EBC7   D0 01       BNE $EBCA       of the start of the character
EBC9   88          DEY             sets in the hires mode and
EBCA   38          SEC             then subtracts 1 from that
```

```
EBCB   E9 01       SBC #$01        address.
EBCD   60          RTS

EBCE   20 03 CF    JSR $CF03       HIMEM
EBD1   20 22 D9    JSR $D922       Evaluate argument and convert
EBD4   A5 33       LDA $33         it to a 2 byte integer.
EBD6   A4 34       LDY $34
EBD8   20 89 EB    JSR $EB89       Test and branch if sufficient
EBDB   90 03       BCC $EBE0       memory to allow new Himem.
EBDD   4C 7C C4    JMP $C47C       Print "OUT 0F MEMORY ERROR".
EBE0   85 A6       STA $A6         Update current himem pointer.
EBE2   84 A7       STY $A7
EBE4   4C 0F C7    JMP $C70F       Clear up pointers and finish.
EBE7   AD 60 02    LDA $0260       GRAB
EBEA   D0 F1       BNE $EBDD
EBEC   AD C0 02    LDA $02C0       Load Screen status. Give error
EBEF   48          PHA             if already in hires mode.
EBF0   29 01       AND #$01
EBF2   F0 05       BEQ $EBF9
EBF4   A2 A3       LDX #$A3        Print "DISP TYPE MISMATCH
EBF6   4C 7E C4    JMP $C47E       ERROR".
EBF9   68          PLA
EBFA   29 FD       AND #$FD        Set screen to GRAB status.
EBFC   8D C0 02    STA $02C0
EBFF   20 C1 EB    JSR $EBC1       Load A and Y with the address
EC02   48          PHA             before the start of the hires
EC03   98          TYA             character set
EC04   18          CLC
EC05   69 1C       ADC #$1C
EC07   A8          TAY
EC08   68          PLA
EC09   4C E0 EB    JMP $EBE0

EC0C   20 C1 EB    JSR $EBC1       RELEASE
EC0F   20 89 EB    JSR $EB89       Load address of byte below
EC12   B0 C9       BCS $EBDD       start of hires char set and
EC14   48          PHA             test that it is not below end
EC15   AD C0 02    LDA $02C0       of Basic. Set screen status
EC18   09 02       ORA #$02        to allow hires mode. Finally
EC1A   8D C0 02    STA $02C0       write the new value of himem.
EC1D   68          PLA
EC1E   4C E0 EB    JMP $EBE0

EC21   AD C0 02    LDA $02C0       TEXT
EC24   A8          TAY
EC25   29 01       AND #$01
EC27   F0 09       BEQ $EC32       Already in text mode.
EC29   98          TYA
EC2A   29 FE       AND #$FE        Set screen status to text.
EC2C   8D C0 02    STA $02C0
EC2F   20 67 F9    JSR $F967       Set screen to text.
EC32   60          RTS

EC33   AD C0 02    LDA $02C0       HIRES
EC36   48          PHA
EC37   29 02       AND #$02        Error if hires mode cannot be
EC39   F0 B9       BEQ $EBF4       entered.
EC3B   68          PLA             Set status to indicate hires
EC3C   09 01       ORA #$01        mode.
EC3E   8D C0 02    STA $02C0
EC41   20 20 F9    JSR $F920       Set screen to hires mode.
EC44   60          RTS

EC45   20 62 D0    JSR $D062       POINT
```

```
EC48   20 17 CF   JSR $CF17       Check '(' is present; if so
EC4B   A5 34      LDA $34         then evaluate the X parameter.
EC4D   48         PHA             Save contents of #33 and #34
EC4E   A5 33      LDA $33         on the stack.
EC50   48         PHA
EC51   20 22 D9   JSR $D922       Convert X parameter to integer
EC54   A5 33      LDA $33         and transfer result to page 2.
EC56   8D E1 02   STA $02E1
EC59   A5 34      LDA $34
EC5B   8D E2 02   STA $02E2
EC5E   68         PLA             Restore values in #33 and #34.
EC5F   85 33      STA $33
EC61   68         PLA
EC62   85 34      STA $34
EC64   20 65 D0   JSR $D065       Search for ','
EC67   20 17 CF   JSR $CF17       Evaluate Y parameter.
EC6A   A5 34      LDA $34         Save contents of #33 and #34
EC6C   48         PHA             on the stack. Must be done
EC6D   A5 33      LDA $33         twice in case of error in
EC6F   48         PHA             searching for ',' etc..
EC70   20 22 D9   JSR $D922       Convert Y parameter to integer
EC73   A5 34      LDA $34         and transfer result to page 2.
EC75   8D E4 02   STA $02E4
EC78   A5 33      LDA $33
EC7A   8D E3 02   STA $02E3
EC7D   68         PLA             Restore contents of #33 and
EC7E   85 33      STA $33         #34 to original state.
EC80   68         PLA
EC81   85 34      STA $34
EC83   20 C8 F1   JSR $F1C8       Test pixel in question.
EC86   AC E1 02   LDY $02E1
EC89   AD E0 02   LDA $02E0
EC8C   29 01      AND #$01
EC8E   D0 09      BNE $EC99       Error found.
EC90   AD E2 02   LDA $02E2
EC93   20 99 D4   JSR $D499       Put signed integer in FPA.
EC96   4C 5F D0   JMP $D05F       Jump to test for '('.
EC99   4C C2 D8   JMP $D8C2       "ILLEGAL QUANTITY ERROR".

EC9C   E6 E9      INC $E9         This is data for the routine
EC9E   D0 02      BNE $ECA2       which gets copied into page
ECA0   E6 EA      INC $EA         zero of memory at $E2. It
ECA2   AD 60 EA   LDA $EA60       holds the current program
ECA5   C9 20      CMP #$20        position and is used to step
ECA7   F0 F3      BEQ $EC9C       through the spaces in a
ECA9   20 B9 EC   JSR $ECB9       program until a non space char
ECAC   60         RTS             is found.

ECAD   2C 60 EA   BIT $EA60
ECB0   2C 60 EA   BIT $EA60
ECB3   60         RTS

ECB4   80 4F C7 52 58             Initial random number.

ECB9   C9 C8      CMP #$C8        Routine to test for statement
ECBB   F0 0E      BEQ $ECCB       delimiter or a number.
ECBD   C9 27      CMP #$27
ECBF   F0 0A      BEQ $ECCB
ECC1   C9 3A      CMP #$3A
ECC3   B0 06      BCS $ECCB
ECC5   38         SEC
ECC6   E9 30      SBC #$30
ECC8   38         SEC             Z is set if colon or null
ECC9   E9 D0      SBC #$D0        found, C is cleared if digit
```

```
ECCB   60          RTS             between 0-9 found.

ECCC   D8          CLD             START OF BASIC
ECCD   A2 FF       LDX #$FF
ECCF   86 A9       STX $A9         Set immediate mode.
ECD1   9A          TXS             Set stack pointer.
ECD2   A9 CC       LDA #$CC        Set up address of start of
ECD4   A0 EC       LDY #$EC        Basic as a jump at #1A.
ECD6   85 1B       STA $1B
ECD8   84 1C       STY $1C
ECDA   A9 4C       LDA #$4C
ECDC   85 1A       STA $1A
ECDE   85 C3       STA $C3         Set up jump opcodes for USR, &
ECE0   85 21       STA $21         and numeric function executer.
ECE2   8D FB 02    STA $02FB
ECE5   A9 36       LDA #$36
ECE7   A0 D3       LDY #$D3
ECE9   85 22       STA $22         Set up default USR address -
ECEB   84 23       STY $23         to give ILLEGAL QUANTITY
ECED   8D FC 02    STA $02FC       ERROR. Do same for & command
ECF0   8C FD 02    STY $02FD       and ! command.
ECF3   8D F5 02    STA $02F5
ECF6   8C F6 02    STY $02F6
ECF9   A2 1C       LDX #$1C        Copy the self-modifying-code
ECFB   BD 9B EC    LDA $EC9B,X     routine into zero page. It is
ECFE   95 E1       STA $E1,X       used to step through the
ED00   CA          DEX             commands being executed (in
ED01   D0 F8       BNE $ECFB       program or immediate mode).
ED03   A9 03       LDA #$03
ED05   85 C2       STA $C2         This section sets up a series
ED07   8A          TXA             of variables.
ED08   85 D7       STA $D7         Clear sign extend byte.
ED0A   85 87       STA $87         Clear top active string ptr.
ED0C   85 2F       STA $2F         Clear next byte to tape.
ED0E   48          PHA
ED0F   85 2E       STA $2E         Clear CTRL 0 flag.
ED11   8D F2 02    STA $02F2       Clear EDIT flag.
ED14   A2 88       LDX #$88        Set string block pointer.
ED16   86 85       STX $85
ED18   A8          TAY
ED19   A9 02       LDA #$02        Set screen to text.
ED1B   8D C0 02    STA $02C0
ED1E   A9 28       LDA #$28        Set up line width on screen.
ED20   8D 57 02    STA $0257
ED23   A9 50       LDA #$50        Set up line width on printer.
ED25   8D 56 02    STA $0256
ED28   A9 00       LDA #$00        Set up TAB positon of cursor.
ED2A   85 30       STA $30         Set Basic's cursor column.
ED2C   8D 58 02    STA $0258       Clear printer cursor position.
ED2F   8D 59 02    STA $0259       Clear screen cursor position.
ED32   20 3E C8    JSR $C83E       Printer off & set variables.
ED35   20 CE CC    JSR $CCCE       CLS command.
ED38   A9 96       LDA #$96        Load start address of initial
ED3A   A0 ED       LDY #$ED        message printed on screen.
ED3C   20 B0 CC    JSR $CCB0       Print message "ORIC EXT..."
ED3F   20 F0 CB    JSR $CBF0
ED42   A2 00       LDX #$00        Set up Start Basic pointer to
ED44   A0 05       LDY #$05        #0500.
ED46   86 9A       STX $9A
ED48   84 9B       STY $9B
ED4A   A0 00       LDY #$00
ED4C   98          TYA
ED4D   91 9A       STA ($9A),Y     Zero the first byte in Basic
ED4F   E6 9A       INC $9A         and increment Start Basic
```

```
ED51    D0 02       BNE $ED55       pointer by 1.
ED53    E6 9B       INC $9B
ED55    20 F0 C6    JSR $C6F0       Set up other Basic Pointers.
ED58    A5 9A       LDA $9A
ED5A    A4 9B       LDY $9B         Test if Begin Basic is beyond
ED5C    20 44 C4    JSR $C444       last string allocated.
ED5F    20 F0 CB    JSR $CBF0
ED62    A5 A6       LDA $A6
ED64    38          SEC             Calculate amount of free
ED65    E5 9A       SBC $9A         memory between Start Basic
ED67    AA          TAX             and Himem. Then print it on
ED68    A5 A7       LDA $A7         the screen in decimal.
ED6A    E5 9B       SBC $9B
ED6C    20 C5 E0    JSR $E0C5
ED6F    A9 88       LDA #$88        Load address of message "BYTES
ED71    A0 ED       LDY #$ED        FREE"
ED73    20 B0 CC    JSR $CCB0       Print above message.
ED76    A9 B0       LDA #$B0
ED78    A0 CC       LDY #$CC        Alter jump location at #1A to
ED7A    85 1B       STA $1B         be able to print "Ready"
ED7C    84 1C       STY $1C         messages.
ED7E    A9 10       LDA #$10
ED80    8D F8 02    STA $02F8
ED83    4C A8 C4    JMP $C4A8       Goto main part of Basic.

ED86    00 00 20 42 59 54 45 53     .. BYTES
ED8E    20 46 52 45 45 0A 0D 00      FREE...
ED96    4F 52 49 43 20 45 58 54     ORIC EXT
ED9E    45 4E 44 45 44 20 42 41     ENDED BA
EDA6    53 49 43 20 56 31 2E 31     SIC V1.1
EDAE    0D 0A 60 20 31 39 38 33     ..© 1983
EDB6    20 54 41 4E 47 45 52 49      TANGERI
EDBE    4E 45 0D 0A 00 00           NE

EDC4    A2 00       LDX #$00        This routine transfers a block
EDC6    A0 00       LDY #$00        of data using #0C as the
EDC8    C4 10       CPY $10         source pointer and #0E as the
EDCA    D0 04       BNE $EDD0       destination pointer. The
EDCC    E4 11       CPX $11         length of data to be moved is
EDCE    F0 0F       BEQ $EDDF       held in locations #10/#11.
EDD0    B1 0C       LDA ($0C),Y
EDD2    91 0E       STA ($0E),Y
EDD4    C8          INY
EDD5    D0 F1       BNE $EDC8
EDD7    E6 0D       INC $0D
EDD9    E6 0F       INC $0F
EDDB    E8          INX
EDDC    4C C8 ED    JMP $EDC8
EDDF    60          RTS

EDE0    48          PHA             This routine sets the three 16
EDE1    20 8C EE    JSR $EE8C       bit counters (#272/3, #274/5 &
EDE4    A9 00       LDA #$00        #276/7) after setting them to
EDE6    A2 00       LDX #$00        zero.
EDE8    A0 03       LDY #$03        #272/3 is set to 3 and is used
EDEA    20 AB EE    JSR $EEAB       as a counter for keyboard
EDED    A9 01       LDA #$01        scanning. #274/5 is set to 25
EDEF    A0 19       LDY #$19        and is used as a counter for
EDF1    20 AB EE    JSR $EEAB       toggling the cursor. #276/7 is
EDF4    A9 00       LDA #$00        not set here but is used in
EDF6    8D 71 02    STA $0271       the WAIT command.
EDF9    AD 0B 03    LDA $030B
EDFC    29 7F       AND #$7F        This section sets up the 6522
EDFE    09 40       ORA #$40        to generate interrupts from
```

```
EE00   8D 0B 03   STA $030B       timer 1 every 10mS (in its
EE03   A9 C0      LDA #$C0        free running mode).
EE05   8D 0E 03   STA $030E
EE08   A9 10      LDA #$10
EE0A   8D 06 03   STA $0306
EE0D   8D 04 03   STA $0304
EE10   A9 27      LDA #$27
EE12   8D 07 03   STA $0307
EE15   8D 05 03   STA $0305
EE18   68         PLA
EE19   60         RTS

EE1A   48         PHA             Disable timer 1 interrupts
EE1B   A9 40      LDA #$40        from the 6522. This routine
EE1D   8D 0E 03   STA $030E       is used by the cassette
EE20   68         PLA             commands.
EE21   60         RTS

EE22   48         PHA             IRQ Handler.
EE23   AD 0D 03   LDA $030D       Test that timer 1 has timed
EE26   29 40      AND #$40        out; if so then go to service
EE28   F0 06      BEQ $EE30       subroutine. The interrupt
EE2A   8D 0D 03   STA $030D       routine is terminated by
EE2D   20 34 EE   JSR $EE34       jumping to the RTI instruction
EE30   68         PLA             at #24A.
EE31   4C 4A 02   JMP $024A
EE34   48         PHA
EE35   8A         TXA
EE36   48         PHA
EE37   98         TYA
EE38   48         PHA
EE39   A0 00      LDY #$00        This section decrements each
EE3B   B9 72 02   LDA $0272,Y     of the three 16 bit counters
EE3E   38         SEC             in page 2 by 1.
EE3F   E9 01      SBC #$01
EE41   99 72 02   STA $0272,Y
EE44   C8         INY
EE45   B9 72 02   LDA $0272,Y
EE48   E9 00      SBC #$00
EE4A   99 72 02   STA $0272,Y
EE4D   C8         INY
EE4E   C0 06      CPY #$06
EE50   D0 E9      BNE $EE3B
EE52   A9 00      LDA #$00        Load X (high) and Y (low) with
EE54   20 9D EE   JSR $EE9D       content of first counter. If
EE57   C0 00      CPY #$00        has reached zero then reload
EE59   D0 10      BNE $EE6B       it with the value of 3.
EE5B   A2 00      LDX #$00
EE5D   A0 03      LDY #$03
EE5F   20 AB EE   JSR $EEAB       After each countdown to zero
EE62   20 95 F4   JSR $F495       strobe the keyboard; the
EE65   8A         TXA             result will be in X and bit 7
EE66   10 03      BPL $EE6B       set if a valid key.
EE68   8E DF 02   STX $02DF       Save the new key.
EE6B   A9 01      LDA #$01        Load X and Y with content of
EE6D   20 9D EE   JSR $EE9D       the second 16 bit counter. If
EE70   C0 00      CPY #$00        it has reached zero then
EE72   D0 12      BNE $EE86       reload it with the value of
EE74   A2 00      LDX #$00        25. When zero, toggle the
EE76   A0 19      LDY #$19        cursor flag in #271.
EE78   20 AB EE   JSR $EEAB       Then place a copy of cursor
EE7B   AD 71 02   LDA $0271       on screen if it is enabled.
EE7E   49 01      EOR #$01
EE80   8D 71 02   STA $0271
```

```
EE83   20 01 F8   JSR $F801
EE86   68         PLA
EE87   A8         TAY
EE88   68         PLA
EE89   AA         TAX
EE8A   68         PLA
EE8B   60         RTS

EE8C   48         PHA            This routine sets to zero
EE8D   98         TYA            the three 16 bit counters
EE8E   48         PHA            at #272/3, #274/5 and #276/7.
EE8F   A0 05      LDY #$05
EE91   A9 00      LDA #$00
EE93   99 72 02   STA $0272,Y
EE96   88         DEY
EE97   10 FA      BPL $EE93
EE99   68         PLA
EE9A   A8         TAY
EE9B   68         PLA
EE9C   60         RTS

EE9D   48         PHA            This routine loads X (high)
EE9E   0A         ASL A          and Y (low) with the content
EE9F   A8         TAY            of the 16 bit counter
EEA0   78         SEI            specified by the content of A.
EEA1   B9 72 02   LDA $0272,Y    The valid values of A are 0, 1
EEA4   BE 73 02   LDX $0273,Y    and 2 which load the 1st, 2nd
EEA7   58         CLI            and 3rd counters respectively.
EEA8   A8         TAY
EEA9   68         PLA
EEAA   60         RTS

EEAB   48         PHA            This routine loads the 16 bit
EEAC   8A         TXA            counter specified by A with
EEAD   48         PHA            the contents of X (high) and
EEAE   98         TYA            Y (low).
EEAF   48         PHA            Values of 0, 1 and 2 in A
EEB0   BA         TSX            access the 1st, 2nd and 3rd
EEB1   BD 03 01   LDA $0103,X    counters respectively.
EEB4   0A         ASL A
EEB5   A8         TAY
EEB6   68         PLA
EEB7   48         PHA
EEB8   78         SEI
EEB9   99 72 02   STA $0272,Y
EEBC   BD 02 01   LDA $0102,X
EEBF   99 73 02   STA $0273,Y
EEC2   58         CLI
EEC3   68         PLA
EEC4   A8         TAY
EEC5   68         PLA
EEC6   AA         TAX
EEC7   68         PLA
EEC8   60         RTS

EEC9   20 AB EE   JSR $EEAB      Load the 16 bit counter
EECC   20 9D EE   JSR $EE9D      specified by A with the
EECF   C0 00      CPY #$00       contents of X and Y and then
EED1   D0 F9      BNE $EECC      wait until that counter has
EED3   E0 00      CPX #$00       decremented to zero.
EED5   D0 F5      BNE $EECC
EED7   60         RTS

EED8   AD 13 02   LDA $0213      Transfer the FB code from bits
```

```
EEDB   8D 14 02   STA $0214       0 and 1 to bits 6 and 7 of
EEDE   4E 12 02   LSR $0212       $0212. The pattern code is
EEE1   6E 12 02   ROR $0212       transferred to a works
EEE4   6E 12 02   ROR $0212       register at $214.
EEE7   60         RTS

EEE8   48         PHA             Write a pixel to the hires
EEE9   98         TYA             screen.
EEEA   48         PHA             Calculate the address of the
EEEB   20 DE EE   JSR $EEDE       byte to write to the screen,
EEEE   20 49 F0   JSR $F049       the position of the pixel in
EEF1   20 24 F0   JSR $F024       that byte and the FB code.
EEF4   68         PLA
EEF5   A8         TAY
EEF6   68         PLA
EEF7   60         RTS

EEF8   D8         CLD             This routine puts lines on the
EEF9   20 D8 EE   JSR $EED8       screen for the DRAW command.
EEFC   2C E2 02   BIT $02E2       Test and branch if X arg't is
EEFF   10 0A      BPL $EF0B       a positive number.
EF01   A9 FF      LDA #$FF        Gets 2's complement of the low
EF03   4D E1 02   EOR $02E1       byte of X argument.
EF06   AA         TAX
EF07   E8         INX
EF08   8E E1 02   STX $02E1
EF0B   2C E4 02   BIT $02E4
EF0E   10 0A      BPL $EF1A       Y argument is positive.
EF10   A9 FF      LDA #$FF        Get 2's complement of the low
EF12   4D E3 02   EOR $02E3       byte of the Y argument.
EF15   AA         TAX
EF16   E8         INX
EF17   8E E3 02   STX $02E3
EF1A   AD E1 02   LDA $02E1
EF1D   CD E3 02   CMP $02E3       X argument is smaller than
EF20   90 0F      BCC $EF31       that of Y.
EF22   AE E1 02   LDX $02E1
EF25   F0 09      BEQ $EF30       Both X and Y args are zero.
EF27   AD E3 02   LDA $02E3       Calculate the slope Y/X of the
EF2A   20 40 EF   JSR $EF40       line.
EF2D   20 84 EF   JSR $EF84       Draw the line.
EF30   60         RTS

EF31   AE E3 02   LDX $02E3
EF34   F0 09      BEQ $EF3F       Both X and Y args are zero.
EF36   AD E1 02   LDA $02E1       Calculate the slope X/Y of the
EF39   20 40 EF   JSR $EF40       line.
EF3C   20 5C EF   JSR $EF5C       Draw the line.
EF3F   60         RTS

EF40   85 0D      STA $0D         Set up the variables for the
EF42   8E 00 02   STX $0200       division routine to find the
EF45   A9 00      LDA #$00        slope of the line.
EF47   85 0C      STA $0C
EF49   8D 01 02   STA $0201
EF4C   20 C8 EF   JSR $EFC8       Calculate slope.
EF4F   20 FA EF   JSR $EFFA       Round up answer.
EF52   A9 00      LDA #$00        Clear remainder and divisor.
EF54   85 0E      STA $0E
EF56   85 0F      STA $0F
EF58   8D 00 02   STA $0200
EF5B   60         RTS

EF5C   2C E4 02   BIT $02E4       Draw line for the case Y > X.
```

```
EF5F   10 06      BPL $EF67       Y is positive.
EF61   20 95 F0   JSR $F095       Move cursor up a line.
EF64   4C 6A EF   JMP $EF6A
EF67   20 89 F0   JSR $F089       Move cursor down a line.
EF6A   20 AC EF   JSR $EFAC
EF6D   F0 0E      BEQ $EF7D       Line is off target.
EF6F   2C E2 02   BIT $02E2
EF72   10 06      BPL $EF7A       X argument is positive.
EF74   20 B2 F0   JSR $F0B2       Move cursor left a pixel.
EF77   4C 7D EF   JMP $EF7D
EF7A   20 A1 F0   JSR $F0A1       Move cursor right a pixel.
EF7D   20 16 F0   JSR $F016       Send pixel to screen.
EF80   CA         DEX             Continue until correct number
EF81   D0 D9      BNE $EF5C       of rows are done.
EF83   60         RTS

EF84   2C E2 02   BIT $02E2       Draw line for the case X > Y.
EF87   10 06      BPL $EF8F       X argument is positive.
EF89   20 B2 F0   JSR $F0B2       Move cursor left a pixel.
EF8C   4C 92 EF   JMP $EF92
EF8F   20 A1 F0   JSR $F0A1       Move cursor right a pixel.
EF92   20 AC EF   JSR $EFAC
EF95   F0 0E      BEQ $EFA5       Line is off target.
EF97   2C E4 02   BIT $02E4
EF9A   10 06      BPL $EFA2       Y argument is positive.
EF9C   20 95 F0   JSR $F095       Move cursor up a line.
EF9F   4C A5 EF   JMP $EFA5
EFA2   20 89 F0   JSR $F089       Move cursor down a line.
EFA5   20 16 F0   JSR $F016       Send pixel to screen.
EFA8   CA         DEX             Continue until correct number
EFA9   D0 D9      BNE $EF84       of columns are done.
EFAB   60         RTS

EFAC   D8         CLD             This routine adds the slope of
EFAD   18         CLC             the line being drawn to #E/F.
EFAE   A5 0E      LDA $0E         This is done so that the
EFB0   65 0C      ADC $0C         drawing routines can keep the
EFB2   85 0E      STA $0E         slope of the line on target.
EFB4   A5 0F      LDA $0F         This is indicated by Z at the
EFB6   65 0D      ADC $0D         end of the routine; if set
EFB8   85 0F      STA $0F         the calling routine misses
EFBA   24 0E      BIT $0E         out drawing a pixel in one
EFBC   10 03      BPL $EFC1       direction. On shallow or
EFBE   18         CLC             steep lines this will give the
EFBF   69 01      ADC #$01        line a step like appearance.
EFC1   CD 00 02   CMP $0200
EFC4   8D 00 02   STA $0200
EFC7   60         RTS

EFC8   48         PHA             This is a division routine
EFC9   8A         TXA             that is used to calculate the
EFCA   48         PHA             slope of a line being drawn.
EFCB   98         TYA
EFCC   48         PHA             The routine acts on 16 bit
EFCD   A9 00      LDA #$00        numbers.
EFCF   85 0E      STA $0E
EFD1   85 0F      STA $0F         Divisor is in #0200/1 and
EFD3   A2 10      LDX #$10        dividend is in #0C/0D. Must be
EFD5   06 0C      ASL $0C         set before routine is called.
EFD7   26 0D      ROL $0D         The quotient ends up in #0C/0D
EFD9   26 0E      ROL $0E         and the remainder in #0E/0F.
EFDB   26 0F      ROL $0F
EFDD   A5 0E      LDA $0E         A, X and Y are unaffected by
EFDF   38         SEC             this routine.
```

```
EFE0    ED 00 02    SBC $0200
EFE3    A8          TAY
EFE4    A5 0F       LDA $0F
EFE6    ED 01 02    SBC $0201
EFE9    90 06       BCC $EFF1
EFEB    E6 0C       INC $0C
EFED    84 0E       STY $0E
EFEF    85 0F       STA $0F
EFF1    CA          DEX
EFF2    D0 E1       BNE $EFD5
EFF4    68          PLA
EFF5    A8          TAY
EFF6    68          PLA
EFF7    AA          TAX
EFF8    68          PLA
EFF9    60          RTS

EFFA    48          PHA             This routine rounds up the
EFFB    0E 00 02    ASL $0200       quotient of the above routine
EFFE    2E 01 02    ROL $0201       if twice the divisor is less
F001    AD 00 02    LDA $0200       than the remainder.
F004    38          SEC
F005    E5 0E       SBC $0E
F007    AD 01 02    LDA $0201
F00A    E5 0F       SBC $0F
F00C    B0 06       BCS $F014
F00E    E6 0C       INC $0C
F010    D0 02       BNE $F014
F012    E6 0D       INC $0D
F014    68          PLA
F015    60          RTS

F016    2C 14 02    BIT $0214       This routine places a pixel
F019    18          CLC             on the screen at the current
F01A    10 04       BPL $F020       cursor position subject to
F01C    20 24 F0    JSR $F024       the data in the PATTERN
F01F    38          SEC             register (in #213).
F020    2E 14 02    ROL $0214
F023    60          RTS

F024    A0 00       LDY #$00        Write a pixel to current
F026    B1 10       LDA ($10),Y     cursor position unless cursor
F028    29 40       AND #$40        is over a location holding a
F02A    F0 1C       BEQ $F048       colour attribute.
F02C    AD 15 02    LDA $0215       Load bit pat'rn for that byte.
F02F    2C 12 02    BIT $0212       Test and branch if the FB code
F032    30 0E       BMI $F042       is 2 or 3.
F034    70 07       BVS $F03D       FB code is 1.
F036    49 FF       EOR #$FF        FB code is 0 therefore set
F038    31 10       AND ($10),Y     pixel to background.
F03A    91 10       STA ($10),Y
F03C    60          RTS

F03D    11 10       ORA ($10),Y     FB code is 1 therefore set
F03F    91 10       STA ($10),Y     pixel to foreground.
F041    60          RTS

F042    70 04       BVS $F048       Exit if FB code is 3.
F044    51 10       EOR ($10),Y     FB code is 2 therefore invert
F046    91 10       STA ($10),Y     the current pixel.
F048    60          RTS

F049    D8          CLD             This routine is entered with
F04A    48          PHA             X and Y holding the horiz'l
```

```
F04B   98          TYA             and vertical cursor positions
F04C   48          PHA             on the hires screen
F04D   20 31 F7    JSR $F731       respectively. This routine
F050   18          CLC             calculates the corresponding
F051   69 00       ADC #$00        address of the byte on the
F053   85 10       STA $10         screen and the position of the
F055   98          TYA             cursor in that byte. The
F056   69 A0       ADC #$A0        latter is held in #215.
F058   85 11       STA $11
F05A   A9 00       LDA #$00        The address of the cursor byte
F05C   85 0D       STA $0D         ends up in #10 and #11.
F05E   8D 01 02    STA $0201
F061   86 0C       STX $0C
F063   A9 06       LDA #$06
F065   8D 00 02    STA $0200
F068   20 C8 EF    JSR $EFC8
F06B   18          CLC
F06C   A5 0C       LDA $0C
F06E   65 10       ADC $10
F070   85 10       STA $10
F072   A9 00       LDA #$00
F074   65 11       ADC $11
F076   85 11       STA $11
F078   A9 20       LDA #$20
F07A   A4 0E       LDY $0E
F07C   F0 04       BEQ $F082
F07E   4A          LSR A
F07F   88          DEY
F080   90 FA       BCC $F07C
F082   8D 15 02    STA $0215
F085   68          PLA
F086   A8          TAY
F087   68          PLA
F088   60          RTS

F089   18          CLC             This routine moves the address
F08A   A5 10       LDA $10         of the cursor to the
F08C   69 28       ADC #$28        corresponding position on the
F08E   85 10       STA $10         line below.
F090   90 02       BCC $F094
F092   E6 11       INC $11
F094   60          RTS

F095   38          SEC             This routine moves the address
F096   A5 10       LDA $10         of the cursor to the
F098   E9 28       SBC #$28        corresponding position on the
F09A   85 10       STA $10         line above.
F09C   B0 02       BCS $F0A0
F09E   C6 11       DEC $11
F0A0   60          RTS

F0A1   4E 15 02    LSR $0215       Move the pixel position within
F0A4   90 0B       BCC $F0B1       the byte on the hires screen
F0A6   A9 20       LDA #$20        one place to the right. Wrap-
F0A8   8D 15 02    STA $0215       around will occur.
F0AB   E6 10       INC $10
F0AD   D0 02       BNE $F0B1
F0AF   E6 11       INC $11
F0B1   60          RTS

F0B2   0E 15 02    ASL $0215       Move the pixel positon within
F0B5   2C 15 02    BIT $0215       the byte on the hires screen
F0B8   50 0D       BVC $F0C7       one place to the left. Wrap-
F0BA   A9 01       LDA #$01        around will occur.
```

```
F0BC   8D 15 02    STA $0215
F0BF   A5 10       LDA $10
F0C1   D0 02       BNE $F0C5
F0C3   C6 11       DEC $11
F0C5   C6 10       DEC $10
F0C7   60          RTS

F0C8   A9 04       LDA #$04        CURSET
F0CA   A2 E5       LDX #$E5
F0CC   20 F8 F2    JSR $F2F8       Test FB code range.
F0CF   B0 28       BCS $F0F9       FB code is out of range.
F0D1   AD E5 02    LDA $02E5       Transfer FB code to a work's
F0D4   8D 12 02    STA $0212       byte.
F0D7   A9 F0       LDA #$F0
F0D9   A2 E1       LDX #$E1
F0DB   20 F8 F2    JSR $F2F8       Test and branch if X coord'te
F0DE   B0 19       BCS $F0F9       is out of range.
F0E0   A9 C8       LDA #$C8
F0E2   A2 E3       LDX #$E3
F0E4   20 F8 F2    JSR $F2F8       Test Y co-ordinate range.
F0E7   B0 10       BCS $F0F9       Y co-ordinate out of range.
F0E9   AE E1 02    LDX $02E1       Update value of hires cursor
F0EC   8E 19 02    STX $0219       and call routine that
F0EF   AC E3 02    LDY $02E3       calculates its new address and
F0F2   8C 1A 02    STY $021A       writes it on screen.
F0F5   20 E8 EE    JSR $EEE8
F0F8   60          RTS

F0F9   EE E0 02    INC $02E0       Indicate error.
F0FC   60          RTS

F0FD   20 0A F3    JSR $F30A       CURMOV
F100   B0 0A       BCS $F10C       Calculate destination cursor
F102   AE 19 02    LDX $0219       position; if C is 1 then it is
F105   AC 1A 02    LDY $021A       out of range. Update cursor
F108   20 E8 EE    JSR $EEE8       position and write cursor to
F10B   60          RTS             screen.

F10C   EE E0 02    INC $02E0       Indicates error.
F10F   60          RTS

F110   20 0A F3    JSR $F30A       DRAW
F113   B0 04       BCS $F119       Calculate cursor destination
F115   20 F8 EE    JSR $EEF8       and call a routine to draw
F118   60          RTS             line on screen.

F119   EE E0 02    INC $02E0       Indicates error.
F11C   60          RTS

F11D   AE E2 02    LDX $02E2       PATTERN
F120   D0 07       BNE $F129       Branch if pattern argument is
F122   AE E1 02    LDX $02E1       over 255 otherwise update
F125   8E 13 02    STX $0213       pattern register.
F128   60          RTS

F129   EE E0 02    INC $02E0       Indicates error.
F12C   60          RTS

F12D   AE E2 02    LDX $02E2       CHAR
F130   D0 3B       BNE $F16D       Error if character is out of
F132   AE E1 02    LDX $02E1       range or is a control char.
F135   E0 20       CPX #$20
F137   90 34       BCC $F16D
F139   E0 80       CPX #$80
```

```
F13B   B0 30       BCS $F16D
F13D   A9 02       LDA #$02        Set parameter limit to 2.
F13F   A2 E3       LDX #$E3        Error if character set
F141   20 F8 F2    JSR $F2F8       indicator is out of range.
F144   B0 27       BCS $F16D
F146   A9 04       LDA #$04
F148   A2 E5       LDX #$E5
F14A   20 F8 F2    JSR $F2F8       Check FB code range.
F14D   B0 1E       BCS $F16D       FB code is out of range.
F14F   AD 19 02    LDA $0219       Check that the character will
F152   C9 EB       CMP #$EB        fit on the screen, i.e. it is
F154   B0 17       BCS $F16D       not too close to the edge.
F156   AD 1A 02    LDA $021A
F159   C9 C1       CMP #$C1
F15B   B0 10       BCS $F16D
F15D   20 71 F1    JSR $F171       Get start address of char code
F160   20 9B F1    JSR $F19B       and write it to the screen.
F163   AE 19 02    LDX $0219       Write new cursor position to
F166   AC 1A 02    LDY $021A       the screen.
F169   20 49 F0    JSR $F049
F16C   60          RTS

F16D   EE E0 02    INC $02E0       Indicates error.
F170   60          RTS

F171   D8          CLD             Calculate start address of
F172   AD E5 02    LDA $02E5       character's bit pattern.
F175   8D 12 02    STA $0212       Transfer FB code to bits 6 and
F178   20 DE EE    JSR $EEDE       7 of #212.
F17B   AD E1 02    LDA $02E1       Transfer character code to #0C
F17E   85 0C       STA $0C         and multiply it by 8 since
F180   A9 00       LDA #$00        each character has 8 bytes of
F182   85 0D       STA $0D         pattern information.
F184   A2 03       LDX #$03
F186   06 0C       ASL $0C
F188   26 0D       ROL $0D
F18A   CA          DEX
F18B   D0 F9       BNE $F186
F18D   AD E3 02    LDA $02E3       If alternate char set is used
F190   0A          ASL A           then add length of standard
F191   0A          ASL A           character set.
F192   18          CLC
F193   69 98       ADC #$98        Add start address of standard
F195   18          CLC             char set; $0C/D now holds
F196   65 0D       ADC $0D         start address of bit pattern
F198   85 0D       STA $0D         for that character.
F19A   60          RTS

F19B   D8          CLD             This routine writes to the
F19C   A0 00       LDY #$00        hires screen the char whose
F19E   84 0F       STY $0F         bit pattern start address is
F1A0   B1 0C       LDA ($0C),Y     in #0C/D.
F1A2   85 0E       STA $0E         Store bit pattern for row.
F1A4   20 5D F3    JSR $F35D       Save current hires cursor.
F1A7   26 0E       ROL $0E         Shift bit pattern in to top
F1A9   26 0E       ROL $0E         6 bits of #0E.
F1AB   A2 06       LDX #$06        Used as a pixel counter.
F1AD   26 0E       ROL $0E         Branch if next pixel is off,
F1AF   90 03       BCC $F1B4       no need to print it to screen.
F1B1   20 24 F0    JSR $F024       Print pixel to screen.
F1B4   20 A1 F0    JSR $F0A1       Shift right pixel pos'n.
F1B7   CA          DEX
F1B8   D0 F3       BNE $F1AD       Continue until end of row.
F1BA   20 6E F3    JSR $F36E       Restore original hires cursor.
```

```
F1BD   20 89 F0    JSR $F089       Move cursor down a line.
F1C0   A4 0F       LDY $0F
F1C2   C8          INY
F1C3   C0 08       CPY #$08        Repeat for 8 rows.
F1C5   D0 D7       BNE $F19E
F1C7   60          RTS

F1C8   A9 F0       LDA #$F0        Main routine for 'POINT'.
F1CA   A2 E1       LDX #$E1
F1CC   20 F8 F2    JSR $F2F8       Test X parameter.
F1CF   B0 2F       BCS $F200       Out of range - over 239.
F1D1   A9 C8       LDA #$C8
F1D3   A2 E3       LDX #$E3
F1D5   20 F8 F2    JSR $F2F8       Test Y parameter.
F1D8   B0 26       BCS $F200       Out of range - over 199.
F1DA   AE E1 02    LDX $02E1       Set hires cursor location to
F1DD   8E 19 02    STX $0219       position given.
F1E0   AC E3 02    LDY $02E3
F1E3   8C 1A 02    STY $021A
F1E6   20 49 F0    JSR $F049       Calculate address of cursor.
F1E9   A0 00       LDY #$00
F1EB   B1 10       LDA ($10),Y     Load pixel byte.
F1ED   2D 15 02    AND $0215       Isolate pixel.
F1F0   F0 05       BEQ $F1F7       Pixel is background.
F1F2   A9 FF       LDA #$FF        Load A for result of -1.
F1F4   4C F9 F1    JMP $F1F9
F1F7   A9 00       LDA #$00
F1F9   8D E1 02    STA $02E1       Result is 0 if background and
F1FC   8D E2 02    STA $02E2       -1 if foreground.
F1FF   60          RTS

F200   EE E0 02    INC $02E0       Indicate error.
F203   60          RTS

F204   A9 10       LDA #$10        PAPER
F206   85 0C       STA $0C         Content of #0C is added to the
F208   A9 00       LDA #$00        paper colour to give attribute
F20A   85 0D       STA $0D         code. #D=0 indicates paper.
F20C   20 1C F2    JSR $F21C       Process argument and write new
F20F   60          RTS             paper colour to screen.

F210   A9 00       LDA #$00        INK
F212   85 0C       STA $0C         Content of #0C is added to the
F214   A9 01       LDA #$01        ink colour to give attribute
F216   85 0D       STA $0D         code. #D=1 indicates ink.
F218   20 1C F2    JSR $F21C       Process argument and write new
F21B   60          RTS             ink colour to screen.

F21C   A9 08       LDA #$08        Set parameter limit to 8.
F21E   A2 E1       LDX #$E1
F220   20 F8 F2    JSR $F2F8       Test paper/ink value given.
F223   B0 3F       BCS $F264       Out of range.
F225   20 5D F3    JSR $F35D       Save hires cursor location.
F228   AD E1 02    LDA $02E1
F22B   05 0C       ORA $0C         Produce and save the correct
F22D   8D 02 02    STA $0202       paper/ink attribute code.
F230   AE 1F 02    LDX $021F
F233   D0 12       BNE $F247       In Hires mode.
F235   A6 0D       LDX $0D         Save the paper/ink colour in
F237   9D 6B 02    STA $026B,X     appropriate location.
F23A   A9 A8       LDA #$A8        Set X to low half of address
F23C   18          CLC             of first row on text screen to
F23D   65 0D       ADC $0D         have its paper/ink changed.
F23F   AA          TAX             Load A with number of rows to
```

```
F240   A0 BB       LDY #$BB        be changed and Y with high
F242   A9 1B       LDA #$1B        byte of start address.
F244   4C 51 F2    JMP $F251
F247   A9 00       LDA #$00        Set X and Y to the low and
F249   18          CLC             high halves of the first row
F24A   65 0D       ADC $0D         to have ink/paper changed.
F24C   AA          TAX             Load A with number of rows to
F24D   A0 A0       LDY #$A0        be done for hires screen.
F24F   A9 C8       LDA #$C8
F251   8D 00 02    STA $0200
F254   86 10       STX $10
F256   84 11       STY $11
F258   A9 01       LDA #$01
F25A   8D 01 02    STA $0201       Fill the appropriate number
F25D   20 CD F2    JSR $F2CD       of rows with new ink/paper.
F260   20 6E F3    JSR $F36E       Restore old hires cursor
F263   60          RTS             address.

F264   EE E0 02    INC $02E0       Indicates error.
F267   60          RTS

F268   D8          CLD             FILL Parameters passed in
F269   AD E3 02    LDA $02E3       block at $02E1.
F26C   8D 01 02    STA $0201
F26F   F0 58       BEQ $F2C9       Error; can't fill 0 coloumns.
F271   A0 00       LDY #$00
F273   AD 19 02    LDA $0219       Horizontal cursor position.
F276   38          SEC
F277   E9 06       SBC #$06        Find byte position of cursor
F279   90 04       BCC $F27F       in row by repeated subtraction
F27B   C8          INY             of 6 until 0 is passed.
F27C   4C 76 F2    JMP $F276
F27F   98          TYA
F280   18          CLC
F281   6D E3 02    ADC $02E3       Test if the final column
F284   A8          TAY             of fill will go off screen.
F285   AD E4 02    LDA $02E4       Generate error if so.
F288   69 00       ADC #$00
F28A   D0 3D       BNE $F2C9
F28C   C0 29       CPY #$29
F28E   B0 39       BCS $F2C9       Third parameter (the byte to
F290   AD E6 02    LDA $02E6       be written to screen) must not
F293   D0 34       BNE $F2C9       be over 255.
F295   AD E1 02    LDA $02E1       Transfer number of rows to be
F298   8D 00 02    STA $0200       filled.
F29B   F0 2C       BEQ $F2C9       Error; can't fill 0 rows.
F29D   18          CLC
F29E   6D 1A 02    ADC $021A
F2A1   A8          TAY
F2A2   AD E2 02    LDA $02E2       Check that the final row will
F2A5   69 00       ADC #$00        not go off bottom of screen.
F2A7   D0 20       BNE $F2C9
F2A9   C0 C9       CPY #$C9        Test that final row is not 201
F2AB   B0 1C       BCS $F2C9       or greater.
F2AD   C0 C8       CPY #$C8
F2AF   D0 02       BNE $F2B3       Set row to 0 if it would
F2B1   A0 00       LDY #$00        otherwise end up at row 200.
F2B3   8C 1A 02    STY $021A       Save final Y cursor position.
F2B6   AD E5 02    LDA $02E5       Transfer byte to be written
F2B9   8D 02 02    STA $0202       on screen.
F2BC   20 CD F2    JSR $F2CD       Fill each row until finished.
F2BF   AC 1A 02    LDY $021A
F2C2   AE 19 02    LDX $0219       Write current hires cursor
F2C5   20 49 F0    JSR $F049       position to screen.
```

```
F2C8   60          RTS

F2C9   EE E0 02    INC $02E0       Indicates error in routine.
F2CC   60          RTS

F2CD   D8          CLD             This routine puts the bit
F2CE   AD 02 02    LDA $0202       pattern held in $202 on the
F2D1   A0 00       LDY #$00        screen. This is repeated
F2D3   91 10       STA ($10),Y     on the following row until
F2D5   C8          INY             the content of $200 is
F2D6   CC 01 02    CPY $0201       decremented to zero.
F2D9   D0 F8       BNE $F2D3
F2DB   20 89 F0    JSR $F089
F2DE   CE 00 02    DEC $0200
F2E1   D0 EB       BNE $F2CE
F2E3   60          RTS

F2E4   8D 04 02    STA $0204       This routine tests whether
F2E7   BD 01 02    LDA $0201,X     the content of $204 is
F2EA   D0 0A       BNE $F2F6       greater than the content of
F2EC   BD 00 02    LDA $0200,X     indexed location. This
F2EF   F0 05       BEQ $F2F6       routine is used in range
F2F1   CD 04 02    CMP $0204       checking of arguments. The
F2F4   90 01       BCC $F2F7       argument is given the error
F2F6   38          SEC             status (C=1) if zero.
F2F7   60          RTS

F2F8   8D 04 02    STA $0204       This routine operates in the
F2FB   BD 01 02    LDA $0201,X     same way as the one above
F2FE   D0 08       BNE $F308       but does not set the error
F300   BD 00 02    LDA $0200,X     condition if the argument is
F303   CD 04 02    CMP $0204       zero.
F306   90 01       BCC $F309
F308   38          SEC
F309   60          RTS

F30A   A9 04       LDA #$04        This routine is used by DRAW
F30C   A2 E5       LDX #$E5        and CURMOV to calculate the
F30E   20 F8 F2    JSR $F2F8       destination of the hires
F311   B0 49       BCS $F35C       cursor. Each parameter is
F313   18          CLC             checked so that the cursor
F314   AD E1 02    LDA $02E1       does not go off the screen
F317   6D 19 02    ADC $0219       and that the wrong FB code is
F31A   8D 00 02    STA $0200       used.
F31D   AD E2 02    LDA $02E2       $2E1/2 and $2E3/4 hold the
F320   69 00       ADC #$00        respective X and Y arguments.
F322   8D 01 02    STA $0201
F325   A2 00       LDX #$00        The address of the hires
F327   A9 F0       LDA #$F0        cursor is not calculated.
F329   20 F8 F2    JSR $F2F8
F32C   B0 2E       BCS $F35C
F32E   18          CLC
F32F   AD E3 02    LDA $02E3
F332   6D 1A 02    ADC $021A
F335   8D 02 02    STA $0202
F338   AD E4 02    LDA $02E4
F33B   69 00       ADC #$00
F33D   8D 03 02    STA $0203
F340   A2 02       LDX #$02
F342   A9 C8       LDA #$C8
F344   20 F8 F2    JSR $F2F8
F347   B0 13       BCS $F35C
F349   AD E5 02    LDA $02E5
F34C   8D 12 02    STA $0212
```

```
F34F   AD 00 02   LDA $0200
F352   8D 19 02   STA $0219
F355   AD 02 02   LDA $0202
F358   8D 1A 02   STA $021A
F35B   18         CLC
F35C   60         RTS

F35D   A5 10      LDA $10        This routine saves the
F35F   8D 16 02   STA $0216      address of the hires cursor
F362   A5 11      LDA $11        at locations $216/7. The
F364   8D 17 02   STA $0217      pixel byte ($215) is also
F367   AD 15 02   LDA $0215      saved at $218.
F36A   8D 18 02   STA $0218
F36D   60         RTS

F36E   AD 16 02   LDA $0216      This routine restores the
F371   85 10      STA $10        hires cursor and pixel
F373   AD 17 02   LDA $0217      position to their original
F376   85 11      STA $11        positions. Used in
F378   AD 18 02   LDA $0218      conjunction with routine
F37B   8D 15 02   STA $0215      above.
F37E   60         RTS

F37F   D8         CLD            CIRCLE
F380   AD E2 02   LDA $02E2      Check that radius is not 0 or
F383   D0 3D      BNE $F3C2      over 255.
F385   AD E1 02   LDA $02E1
F388   F0 38      BEQ $F3C2
F38A   AD 19 02   LDA $0219      Check that the circle will fit
F38D   CD E1 02   CMP $02E1      on the screen horizontally.
F390   90 30      BCC $F3C2
F392   18         CLC
F393   6D E1 02   ADC $02E1
F396   C9 F0      CMP #$F0
F398   B0 28      BCS $F3C2
F39A   AD 1A 02   LDA $021A      Check that the cursor will fit
F39D   CD E1 02   CMP $02E1      on the screen vertically.
F3A0   90 20      BCC $F3C2
F3A2   18         CLC
F3A3   6D E1 02   ADC $02E1
F3A6   C9 C8      CMP #$C8
F3A8   B0 18      BCS $F3C2
F3AA   A2 E3      LDX #$E3       Check that the FB code is not
F3AC   A9 04      LDA #$04       out of range.
F3AE   20 F8 F2   JSR $F2F8
F3B1   B0 0F      BCS $F3C2
F3B3   AD E3 02   LDA $02E3
F3B6   8D 12 02   STA $0212
F3B9   20 D8 EE   JSR $EED8      Put FB in bits 6,7 of $212.
F3BC   20 C6 F3   JSR $F3C6      Draw the circle.
F3BF   4C C5 F3   JMP $F3C5
F3C2   EE E0 02   INC $02E0
F3C5   60         RTS

F3C6   20 5D F3   JSR $F35D      Save hires cursor address.
F3C9   AD 1A 02   LDA $021A
F3CC   38         SEC            Calculate smallest Y co-ord.
F3CD   ED E1 02   SBC $02E1
F3D0   A8         TAY
F3D1   AE 19 02   LDX $0219      Load horizontal cursor.
F3D4   20 49 F0   JSR $F049      Draw cursor at top of circle.
F3D7   AD E1 02   LDA $02E1
F3DA   85 0F      STA $0F
F3DC   20 85 F4   JSR $F485
```

```
F3DF   A9 80      LDA #$80
F3E1   8D 1B 02   STA $021B
F3E4   8D 1D 02   STA $021D
F3E7   A9 00      LDA #$00
F3E9   8D 1C 02   STA $021C
F3EC   AD E1 02   LDA $02E1
F3EF   8D 1E 02   STA $021E
F3F2   A9 00      LDA #$00
F3F4   85 0F      STA $0F
F3F6   20 14 F4   JSR $F414
F3F9   20 44 F4   JSR $F444
F3FC   A5 0F      LDA $0F
F3FE   F0 03      BEQ $F403
F400   20 16 F0   JSR $F016
F403   AD 1C 02   LDA $021C
F406   D0 EA      BNE $F3F2
F408   AD 1E 02   LDA $021E
F40B   CD E1 02   CMP $02E1
F40E   D0 E2      BNE $F3F2
F410   20 6E F3   JSR $F36E
F413   60         RTS

F414   AD 1D 02   LDA $021D
F417   AE 1E 02   LDX $021E
F41A   20 74 F4   JSR $F474
F41D   A5 0C      LDA $0C
F41F   18         CLC
F420   6D 1B 02   ADC $021B
F423   8D 1B 02   STA $021B
F426   AD 1C 02   LDA $021C
F429   85 0C      STA $0C
F42B   65 0D      ADC $0D
F42D   8D 1C 02   STA $021C
F430   C5 0C      CMP $0C
F432   F0 0F      BEQ $F443
F434   B0 06      BCS $F43C
F436   20 A1 F0   JSR $F0A1
F439   4C 3F F4   JMP $F43F
F43C   20 B2 F0   JSR $F0B2
F43F   A9 01      LDA #$01
F441   85 0F      STA $0F
F443   60         RTS

F444   AD 1B 02   LDA $021B
F447   AE 1C 02   LDX $021C
F44A   20 74 F4   JSR $F474
F44D   38         SEC
F44E   AD 1D 02   LDA $021D
F451   E5 0C      SBC $0C
F453   8D 1D 02   STA $021D
F456   AD 1E 02   LDA $021E
F459   85 0C      STA $0C
F45B   E5 0D      SBC $0D
F45D   8D 1E 02   STA $021E
F460   C5 0C      CMP $0C
F462   F0 0F      BEQ $F473
F464   B0 06      BCS $F46C
F466   20 89 F0   JSR $F089
F469   4C 6F F4   JMP $F46F
F46C   20 95 F0   JSR $F095
F46F   A9 01      LDA #$01
F471   85 0F      STA $0F
F473   60         RTS
```

```
F474   85 0C       STA $0C         This routine does an
F476   86 0D       STX $0D         arithmetic shift right on the
F478   A6 0E       LDX $0E         16 bit integer in $C and $D.
F47A   A5 0D       LDA $0D         This is repeated according to
F47C   2A          ROL A           the content of $E.
F47D   66 0D       ROR $0D
F47F   66 0C       ROR $0C
F481   CA          DEX
F482   D0 F6       BNE $F47A
F484   60          RTS

F485   E6 0F       INC $0F         Set A to 2 raised to the
F487   A9 00       LDA #$00        power of the content of $0F.
F489   85 0E       STA $0E         $0E used as a counter.
F48B   A9 01       LDA #$01
F48D   0A          ASL A
F48E   E6 0E       INC $0E
F490   C5 0F       CMP $0F
F492   90 F9       BCC $F48D
F494   60          RTS

F495   48          PHA             STROBE KEYBOARD.
F496   08          PHP
F497   98          TYA
F498   48          PHA
F499   D8          CLD
F49A   AD 08 02    LDA $0208
F49D   10 1E       BPL $F4BD       No key pressed from last time.
F49F   29 87       AND #$87
F4A1   8D 10 02    STA $0210       Test if same key is still
F4A4   AE 0A 02    LDX $020A       pressed.
F4A7   20 61 F5    JSR $F561
F4AA   CD 10 02    CMP $0210
F4AD   D0 0E       BNE $F4BD       Key is no longer pressed.
F4AF   CE 0E 02    DEC $020E       Decrement repeat counter.
F4B2   D0 33       BNE $F4E7
F4B4   AD 4F 02    LDA $024F       Reload repeat counter for
F4B7   8D 0E 02    STA $020E       following repeat.
F4BA   4C C6 F4    JMP $F4C6
F4BD   AD 4E 02    LDA $024E       Reset repeat counter for
F4C0   8D 0E 02    STA $020E       first repeat.
F4C3   20 23 F5    JSR $F523       Find key.
F4C6   20 EF F4    JSR $F4EF       Convert key to Ascii code.
F4C9   AA          TAX
F4CA   10 1D       BPL $F4E9       Unrecognised key.
F4CC   48          PHA
F4CD   AD 6A 02    LDA $026A
F4D0   29 08       AND #$08
F4D2   D0 0F       BNE $F4E3       Keyclick disabled.
F4D4   68          PLA
F4D5   48          PHA
F4D6   C9 A0       CMP #$A0
F4D8   90 06       BCC $F4E0       Change keyclick if CTRL char.
F4DA   20 14 FB    JSR $FB14       High pitch keyclick.
F4DD   4C E3 F4    JMP $F4E3
F4E0   20 2A FB    JSR $FB2A       Low pitch keyclick.
F4E3   68          PLA
F4E4   4C E9 F4    JMP $F4E9
F4E7   A9 00       LDA #$00        X holds the Ascii code of the
F4E9   AA          TAX             key pressed and bit 7 will be
F4EA   68          PLA             set. If no key is pressed then
F4EB   A8          TAY             bit 7 of X will be clear.
F4EC   28          PLP
F4ED   68          PLA             A, Y and P are unaffected.
```

```
F4EE   60          RTS

F4EF   AD 09 02    LDA $0209       CONVERT KEY TO ASCII CODE.
F4F2   A8          TAY             Test if the shift keys are
F4F3   A9 00       LDA #$00        pressed. If so then add #40
F4F5   C0 A4       CPY #$A4        to the keycode.
F4F7   F0 04       BEQ $F4FD
F4F9   C0 A7       CPY #$A7
F4FB   D0 03       BNE $F500
F4FD   18          CLC
F4FE   69 40       ADC #$40
F500   18          CLC
F501   6D 08 02    ADC $0208
F504   10 1C       BPL $F522
F506   29 7F       AND #$7F        Transfer keycode to X for use
F508   AA          TAX             as an index into look up
F509   BD 78 FF    LDA $FF78,X     table.
F50C   2D 0C 02    AND $020C
F50F   10 03       BPL $F514       CAPS is off.
F511   38          SEC
F512   E9 20       SBC #$20        Alter code if CAPS is on.
F514   29 7F       AND #$7F
F516   C0 A2       CPY #$A2
F518   D0 06       BNE $F520       CONTROL key is not pressed.
F51A   C9 40       CMP #$40        Don't convert characters
F51C   30 02       BMI $F520       before @ in the Ascii set to
F51E   29 1F       AND #$1F        control characters.
F520   09 80       ORA #$80        Set bit 7 to indicate valid
F522   60          RTS             Ascii code.

F523   A9 38       LDA #$38        FIND KEY.
F525   8D 0D 02    STA $020D       Initialise counters.
F528   8D 08 02    STA $0208
F52B   8D 09 02    STA $0209
F52E   A9 7F       LDA #$7F        Set up first column.
F530   48          PHA
F531   68          PLA
F532   48          PHA
F533   AA          TAX             X holds column data.
F534   A9 07       LDA #$07
F536   20 61 F5    JSR $F561       Send X to 8912 I/O port.
F539   0D 0D 02    ORA $020D
F53C   10 12       BPL $F550
F53E   A2 00       LDX #$00
F540   A0 20       LDY #$20
F542   CC 0D 02    CPY $020D
F545   D0 01       BNE $F548       Column 4 of KB not being used.
F547   E8          INX             Save the code of the pressed
F548   9D 08 02    STA $0208,X     key, different location is
F54B   68          PLA             used if key is SHIFT/CTRL.
F54C   48          PHA
F54D   9D 0A 02    STA $020A,X
F550   38          SEC
F551   68          PLA             Shift the zero bit in A to
F552   6A          ROR A           select the next column.
F553   48          PHA
F554   38          SEC
F555   AD 0D 02    LDA $020D       Decrement key counter by 8
F558   E9 08       SBC #$08        so as to obtain start of next
F55A   8D 0D 02    STA $020D       column. Continue until all 8
F55D   10 D2       BPL $F531
F55F   68          PLA             columns have been done.
F560   60          RTS
```

```
F561   48          PHA             TEST KEYS IN COLUMN HELD IN X.
F562   A9 0E       LDA #$0E        Set A to I/O register E.
F564   20 90 F5    JSR $F590       Write X to register A.
F567   68          PLA
F568   29 07       AND #$07
F56A   AA          TAX             Send content of A to the row/
F56B   8D 11 02    STA $0211       multiplexer which is accessed
F56E   09 B8       ORA #$B8        via bits 0-2 of port B of the
F570   8D 00 03    STA $0300       6522 (at $300).
F573   A0 04       LDY #$04        Pause for a while.
F575   88          DEY
F576   D0 FD       BNE $F575
F578   AD 00 03    LDA $0300       Read in input and test if
F57B   29 08       AND #$08        that key is pressed.
F57D   D0 0D       BNE $F58C       Key is pressed.
F57F   CA          DEX
F580   8A          TXA             Continue with other rows
F581   29 07       AND #$07        until a key is found or end
F583   AA          TAX             of row is reached.
F584   CD 11 02    CMP $0211
F587   D0 E5       BNE $F56E
F589   A9 00       LDA #$00        Set bit 7 of A to 0 to
F58B   60          RTS             indicate key not found.

F58C   8A          TXA
F58D   09 80       ORA #$80        Set bit 7 of A to 1 to
F58F   60          RTS             indicate key found.

F590   08          PHP             WRITE X TO REGISTER A 0F 8912.
F591   78          SEI
F592   8D 0F 03    STA $030F       Send A to port A of 6522.
F595   A8          TAY
F596   8A          TXA
F597   C0 07       CPY #$07        If writing to register 7, set
F599   D0 02       BNE $F59D       1/0 port to output.
F59B   09 40       ORA #$40
F59D   48          PHA
F59E   AD 0C 03    LDA $030C       Set CA2 (BC1 of 8912) to 1,
F5A1   09 EE       ORA #$EE        set CB2 (BDIR of 8912) to 1.
F5A3   8D 0C 03    STA $030C       8912 latches the address.
F5A6   29 11       AND #$11        Set CA2 and CB2 to 0, BC1 and
F5A8   09 CC       ORA #$CC        BDIR in inactive state.
F5AA   8D 0C 03    STA $030C
F5AD   AA          TAX
F5AE   68          PLA
F5AF   8D 0F 03    STA $030F       Send data to 8912 register.
F5B2   8A          TXA
F5B3   09 EC       ORA #$EC        Set CA2 to 0 and CB2 to 1,
F5B5   8D 0C 03    STA $030C       8912 latches data.
F5B8   29 11       AND #$11        Set CA2 and CB2 to 0, BC1 and
F5BA   09 CC       ORA #$CC        BDIR in inactive state.
F5BC   8D 0C 03    STA $030C
F5BF   28          PLP
F5C0   60          RTS

F5C1   08          PHP             SEND CHAR TO PRINTER PORT.
F5C2   78          SEI
F5C3   8D 01 03    STA $0301       Send A to port A of 6522.
F5C6   AD 00 03    LDA $0300
F5C9   29 EF       AND #$EF        Send the strobe line low.
F5CB   8D 00 03    STA $0300
F5CE   AD 00 03    LDA $0300
F5D1   09 10       ORA #$10        Send the strobe line high.
F5D3   8D 00 03    STA $0300
```

```
F5D6    28          PLP
F5D7    AD 0D 03    LDA $030D       Wait in a loop until active
F5DA    29 02       AND #$02        transition of CA1 -
F5DC    F0 F9       BEQ $F5D7       acknowledging the byte.
F5DE    AD 0D 03    LDA $030D
F5E1    60          RTS

F5E2    CF CF CF CF A3 CF A6 CC     Offset table for each of the
F5EA    00 27 34 0F 66 99 60 CF     control character routines.
F5F2    A7 B3 CF A8 BE CF CF CF
F5FA    CF CF A5 A5 CF A4 84 CF

F602    29 1F       AND #$1F        CONTROL CHARACTER ROUTINE.
F604    AA          TAX
F605    BD E2 F5    LDA $F5E2,X     Use char code to look up
F608    18          CLC             routine offset and calculate
F609    69 2F       ADC #$2F        an indirect jump address at
F60B    8D 61 02    STA $0261       $0261.
F60E    A9 00       LDA #$00
F610    69 F6       ADC #$F6
F612    8D 62 02    STA $0262
F615    AD 6A 02    LDA $026A
F618    48          PHA
F619    29 FE       AND #$FE        Temporarily disable cursor.
F61B    8D 6A 02    STA $026A
F61E    68          PLA
F61F    29 01       AND #$01
F621    8D 51 02    STA $0251
F624    A9 00       LDA #$00        Turn the cursor off.
F626    20 01 F8    JSR $F801
F629    38          SEC
F62A    A9 00       LDA #$00
F62C    6C 61 02    JMP ($0261)     Jump to appropriate routine.
F62F    CE 69 02    DEC $0269       Cursor left one place.
F632    30 05       BMI $F639
F634    20 D7 F7    JSR $F7D7
F637    D0 40       BNE $F679       Finish.
F639    A9 27       LDA #$27
F63B    8D 69 02    STA $0269
F63E    AD 68 02    LDA $0268       Cursor up one place.
F641    C9 01       CMP #$01
F643    F0 34       BEQ $F679       Finish if on top line.
F645    CE 68 02    DEC $0268       Cursor row number.
F648    38          SEC             Adjust start of line pointer.
F649    A5 12       LDA $12
F64B    E9 28       SBC #$28
F64D    85 12       STA $12
F64F    B0 02       BCS $F653
F651    C6 13       DEC $13
F653    4C FE F6    JMP $F6FE       Finish.
F656    EE 69 02    INC $0269       Cursor right one place.
F659    A2 27       LDX #$27
F65B    EC 69 02    CPX $0269
F65E    10 19       BPL $F679
F660    20 0D F7    JSR $F70D
F663    AD 68 02    LDA $0268       Cursor down one place.
F666    CD 7E 02    CMP $027E
F669    F0 11       BEQ $F67C
F66B    EE 68 02    INC $0268       Cursor row number.
F66E    18          CLC             Adjust start of line pointer.
F66F    A5 12       LDA $12
F671    69 28       ADC #$28
F673    85 12       STA $12
F675    90 02       BCC $F679
```

```
F677   E6 13       INC $13
F679   4C FE F6    JMP $F6FE       Finish.
F67C   20 5D F3    JSR $F35D
F67F   A2 06       LDX #$06
F681   BD 77 02    LDA $0277,X
F684   95 0B       STA $0B,X
F686   CA          DEX
F687   D0 F8       BNE $F681
F689   20 C4 ED    JSR $EDC4       Block transfer.
F68C   20 6E F3    JSR $F36E
F68F   20 1A F7    JSR $F71A       CTRL N. Clear current row.
F692   4C FE F6    JMP $F6FE       Finish.
F695   AE 7E 02    LDX $027E       CTRL L. Clear screen.
F698   AD 7A 02    LDA $027A       Reset row start address to top
F69B   85 12       STA $12         line.
F69D   AD 7B 02    LDA $027B
F6A0   85 13       STA $13
F6A2   20 1A F7    JSR $F71A       Clear current line.
F6A5   18          CLC
F6A6   A5 12       LDA $12         Adjust start of row pointer.
F6A8   69 28       ADC #$28
F6AA   85 12       STA $12
F6AC   90 02       BCC $F6B0
F6AE   E6 13       INC $13
F6B0   CA          DEX             Clear lines until whole
F6B1   D0 EF       BNE $F6A2       screen is done.
F6B3   20 0D F7    JSR $F70D       Set cursor to start of line.
F6B6   A9 01       LDA #$01        Set cursor row to top line.
F6B8   8D 68 02    STA $0268
F6BB   AD 7A 02    LDA $027A       Set row start address to that
F6BE   85 12       STA $12         of top line of text.
F6C0   AD 7B 02    LDA $027B
F6C3   85 13       STA $13
F6C5   4C FE F6    JMP $F6FE       Finish.
F6C8   20 0D F7    JSR $F70D       CTRL M. Carriage return.
F6CB   8E 53 02    STX $0253
F6CE   4C FE F6    JMP $F6FE       Finish.
F6D1   2A          ROL A
F6D2   2A          ROL A           CTRL D.
F6D3   2A          ROL A           CTRL ].
F6D4   2A          ROL A           ESCAPE.
F6D5   2A          ROL A           CTRL F.
F6D6   2A          ROL A           CTRL P.
F6D7   2A          ROL A           CTRL S.
F6D8   2A          ROL A
F6D9   4D 6A 02    EOR $026A       Toggle appropriate flag in
F6DC   8D 6A 02    STA $026A       $026A.
F6DF   4C FE F6    JMP $F6FE       Finish.
F6E2   AD 51 02    LDA $0251       CTRL Q.
F6E5   49 01       EOR #$01
F6E7   8D 51 02    STA $0251
F6EA   4C FE F6    JMP $F6FE       Finish.
F6ED   AD 0C 02    LDA $020C       CTRL T.
F6F0   49 80       EOR #$80        Invert CAPS flag.
F6F2   8D 0C 02    STA $020C
F6F5   20 5A F7    JSR $F75A       Write message to status line.
F6F8   4C FE F6    JMP $F6FE       Finish.
F6FB   20 9F FA    JSR $FA9F       CTRL G. Calls PING routine.
F6FE   AD 6A 02    LDA $026A       All control char routines
F701   0D 51 02    ORA $0251       end here by restoring the
F704   8D 6A 02    STA $026A       original cursor status.
F707   A9 01       LDA #$01
F709   20 01 F8    JSR $F801
F70C   60          RTS
```

```
F70D   A2 00       LDX #$00        This routine sets the cursor
F70F   20 DE F7    JSR $F7DE       to the start of the line,
F712   D0 02       BNE $F716       taking in to account if the
F714   E8          INX             screen is protected or not.
F715   E8          INX
F716   8E 69 02    STX $0269
F719   60          RTS

F71A   A0 27       LDY #$27        CLEAR CURRENT LINE.
F71C   A9 20       LDA #$20        This routine writes space
F71E   91 12       STA ($12),Y     characters to the whole line
F720   88          DEY             and then wites the paper and
F721   10 FB       BPL $F71E       ink colours to the first two
F723   A0 00       LDY #$00        columns.
F725   AD 6B 02    LDA $026B
F728   91 12       STA ($12),Y
F72A   AD 6C 02    LDA $026C
F72D   C8          INY
F72E   91 12       STA ($12),Y
F730   60          RTS

F731   A0 00       LDY #$00        This routine multiplies the
F733   8C 63 02    STY $0263       content of the accumulator by
F736   8D 64 02    STA $0264       #28 (40). Y holds the high
F739   0A          ASL A           byte of the result. The page
F73A   2E 63 02    ROL $0263       2 locations store temporary
F73D   0A          ASL A           results.
F73E   2E 63 02    ROL $0263
F741   18          CLC
F742   6D 64 02    ADC $0264       The result is calculated by
F745   90 03       BCC $F74A       adding 4 x A to A and then
F747   EE 63 02    INC $0263       double the result.
F74A   0A          ASL A
F74B   2E 63 02    ROL $0263
F74E   0A          ASL A
F74F   2E 63 02    ROL $0263
F752   0A          ASL A
F753   2E 63 02    ROL $0263
F756   AC 63 02    LDY $0263
F759   60          RTS

F75A   AD 0C 02    LDA $020C       This routine writes a message
F75D   10 07       BPL $F766       to the status line depending
F75F   A9 70       LDA #$70        on the state of the CAPS flag.
F761   A0 F7       LDY #$F7
F763   4C 6A F7    JMP $F76A       If CAPS is on then "CAPS" is
F766   A9 76       LDA #$76        written to screen otherwise
F768   A0 F7       LDY #$F7        cleared by writing spaces in
F76A   A2 23       LDX #$23        same place.
F76C   20 65 F8    JSR $F865
F76F   60          RTS

F770   07 43 41 50 53 00           Data for the above routine.
F776   07 20 20 20 20 00

F77C   48          PHA             PRINT CHAR TO SCREEN (in X).
F77D   08          PHP
F77E   98          TYA             Save all registers on stack.
F77F   48          PHA
F780   8A          TXA
F781   48          PHA             Leave a copy of X in A.
F782   D8          CLD
F783   E0 13       CPX #$13        Test for CTRL S, T, and F.
```

```
F785   F0 46       BEQ $F7CD       If either of them are in X
F787   E0 14       CPX #$14        then go to CTRL CHAR routine.
F789   F0 42       BEQ $F7CD
F78B   E0 06       CPX #$06
F78D   F0 3E       BEQ $F7CD
F78F   AD 6A 02    LDA $026A
F792   29 02       AND #$02
F794   F0 3A       BEQ $F7D0       Screen printing inhibited.
F796   8A          TXA
F797   C9 20       CMP #$20
F799   90 32       BCC $F7CD       Control character present.
F79B   AD 6A 02    LDA $026A       Test and branch if the
F79E   29 10       AND #$10        ESCAPE key was not the last
F7A0   F0 13       BEQ $F7B5       printed.
F7A2   8A          TXA             If character after ESCAPE is
F7A3   38          SEC             a CTRL character then print a
F7A4   E9 40       SBC #$40        space instead. Otherwise
F7A6   30 09       BMI $F7B1       convert key to attribute code
F7A8   29 1F       AND #$1F        and print to screen.
F7AA   20 E4 F7    JSR $F7E4
F7AD   A9 1B       LDA #$1B
F7AF   D0 1C       BNE $F7CD
F7B1   A9 20       LDA #$20
F7B3   10 F5       BPL $F7AA
F7B5   E0 7F       CPX #$7F        Character is DELete.
F7B7   F0 08       BEQ $F7C1
F7B9   68          PLA
F7BA   48          PHA

F7BB   20 E4 F7    JSR $F7E4       Print accumulator on screen.
F7BE   4C D0 F7    JMP $F7D0
F7C1   A9 08       LDA #$08        DEL is done by moving cursor
F7C3   20 02 F6    JSR $F602       back 1 place and printing a
F7C6   A9 20       LDA #$20        space character, and moving
F7C8   20 E4 F7    JSR $F7E4       cursor back again 1 place.
F7CB   A9 08       LDA #$08
F7CD   20 02 F6    JSR $F602
F7D0   68          PLA
F7D1   AA          TAX
F7D2   68          PLA             Registers not affected at
F7D3   A8          TAY             end of the routine.
F7D4   28          PLP
F7D5   68          PLA
F7D6   60          RTS

F7D7   AD 69 02    LDA $0269       This routine sets 1=1 if the
F7DA   29 FE       AND #$FE        cursor is on columns 1 and 2
F7DC   D0 05       BNE $F7E3       of a protected screen.
F7DE   AD 6A 02    LDA $026A
F7E1   29 20       AND #$20
F7E3   60          RTS

F7E4   48          PHA             PRINT ACCUMULATOR ON SCREEN.
F7E5   AC 69 02    LDY $0269
F7E8   91 12       STA ($12),Y
F7EA   2C 6A 02    BIT $026A
F7ED   50 0B       BVC $F7FA       Double height flag is clear.
F7EF   AD 69 02    LDA $0269
F7F2   18          CLC             In double height mode the
F7F3   69 28       ADC #$28        char is printed on the line
F7F5   A8          TAY             below in the same column.
F7F6   68          PLA
F7F7   48          PHA
F7F8   91 12       STA ($12),Y     Put A on screen.
```

```
F7FA   A9 09       LDA #$09        Move cursor forward by 1
F7FC   20 02 F6    JSR $F602       column.
F7FF   68          PLA
F800   60          RTS

F801   2D 6A 02    AND $026A       This routine turns the cursor
F804   4A          LSR A           on or off depending on value
F805   6A          ROR A           in A. 0 for off, 1 for on.
F806   8D 65 02    STA $0265       Cursor being turned on is
F809   AC 69 02    LDY $0269       subject to cursor flag being
F80C   B1 12       LDA ($12),Y     enabled.
F80E   29 7F       AND #$7F
F810   0D 65 02    ORA $0265
F813   91 12       STA ($12),Y
F815   60          RTS

F816   A9 00       LDA #$00        GENERATE ALT. CHAR SET.
F818   85 0C       STA $0C
F81A   A9 B9       LDA #$B9        The set is generated in two
F81C   85 0D       STA $0D         halves with A counting from
F81E   A9 00       LDA #$00        #0 to #1F and then #20 to #3F.
F820   20 2D F8    JSR $F82D
F823   A0 BA       LDY #$BA        The set is dumped between
F825   84 0D       STY $0D         $B900 and $BAFF.
F827   A9 20       LDA #$20
F829   20 2D F8    JSR $F82D
F82C   60          RTS

F82D   A0 00       LDY #$00        This and the following
F82F   48          PHA             routine create the alternate
F830   20 54 F8    JSR $F854       set by producing all
F833   91 0C       STA ($0C),Y     combinations of data and
F835   C8          INY             placing it in memory.
F836   68          PLA
F837   48          PHA
F838   20 52 F8    JSR $F852
F83B   68          PLA
F83C   48          PHA
F83D   20 50 F8    JSR $F850
F840   91 0C       STA ($0C),Y
F842   C8          INY
F843   C0 00       CPY #$00
F845   F0 07       BEQ $F84E
F847   68          PLA
F848   18          CLC
F849   69 01       ADC #$01
F84B   4C 2F F8    JMP $F82F
F84E   68          PLA
F84F   60          RTS

F850   4A          LSR A           Part of the routine to
F851   4A          LSR A           produce the alternate char.
F852   4A          LSR A           set. Has 2 entry points, at
F853   4A          LSR A           $F850 and $F854 which copy
F854   29 03       AND #$03        the data into two successive
F856   AA          TAX             memory locations
F857   BD 61 F8    LDA $F861,X
F85A   91 0C       STA ($0C),Y
F85C   C8          INY
F85D   91 0C       STA ($0C),Y
F85F   C8          INY
F860   60          RTS

F861   00 38 07 3F                 Data for alt. char. set.
```

```
F865   85 0C       STA $0C          PRINT TO STATUS LINE.
F867   84 0D       STY $0D          Subject to the content of
F869   AD 1F 02    LDA $021F        $021F being zero, the message
F86C   D0 0D       BNE $F87B        whose start address is held
F86E   A0 00       LDY #$00         in A (low) and Y (high) is
F870   B1 0C       LDA ($0C),Y      printed on to the status line
F872   F0 07       BEQ $F87B        of the screen starting at
F874   9D 80 BB    STA $BB80,X      $BB80. Message must terminate
F877   E8          INX              with a zero byte.
F878   C8          INY
F879   D0 F5       BNE $F870
F87B   60          RTS

F87C   4C 7C F7    JMP $F77C        This is data that is copied
F87F   4C 78 EB    JMP $EB78        in to page 2 of memory as
F882   4C C1 F5    JMP $F5C1        various jump vectors. Set up
F885   4C 65 F8    JMP $F865        by reset of machine.
F888   4C 22 EE    JMP $EE22
F88B   4C B2 F8    JMP $F8B2
F88E   40          RTI

F88F   A2 FF       LDX #$FF         RESET
F891   9A          TXS              Set stack pointer to #FF.
F892   58          CLI
F893   D8          CLD              Clear decimal mode.
F894   A2 12       LDX #$12         Copy the above jump table in
F896   BD 7C F8    LDA $F87C,X      to page 2 of memory.
F899   9D 38 02    STA $0238,X
F89C   CA          DEX
F89D   10 F7       BPL $F896
F89F   A9 20       LDA #$20         Set up initial repeat delay.
F8A1   8D 4E 02    STA $024E
F8A4   A9 04       LDA #$04         Set up successive repeat
F8A6   8D 4F 02    STA $024F        delay.
F8A9   20 14 FA    JSR $FA14        Find quantity and test RAM.
F8AC   20 B8 F8    JSR $F8B8        Set up system.

F8AF   4C CC EC    JMP $ECCC        START BASIC
F8B2   20 B8 F8    JSR $F8B8        NMI service routine.

F8B5   4C 71 C4    JMP $C471        RESTART BASIC.
F8B8   20 AA F9    JSR $F9AA        Set 6522, with no interrupts.
F8BB   A9 07       LDA #$07
F8BD   A2 40       LDX #$40         Set I/0 port on 8912 to
F8BF   20 90 F5    JSR $F590        output.
F8C2   20 E0 ED    JSR $EDE0        Set the 3 16 bit counters.
F8C5   20 0E F9    JSR $F90E        Set up paper/ink colours.
F8C8   A9 FF       LDA #$FF
F8CA   8D 0C 02    STA $020C        Set CAPS to on.
F8CD   20 C9 F9    JSR $F9C9        Set up initial text screen.
F8D0   A2 05       LDX #$05         Set up the Standard character
F8D2   20 82 F9    JSR $F982        set in memory.
F8D5   20 16 F8    JSR $F816        Generate alternate char. set.
F8D8   20 5A F7    JSR $F75A        Write CAPS status to screen.
F8DB   60          RTS

F8DC   48          PHA              Set up some page 2 variables
F8DD   8A          TXA              for HIRES.
F8DE   48          PHA
F8DF   A9 01       LDA #$01
F8E1   8D 1F 02    STA $021F        Set hires indicator.
F8E4   A9 BF       LDA #$BF
F8E6   8D 7B 02    STA $027B        Set the address of the first
```

```
F8E9    8D 79 02    STA $0279       line of text section of screen
F8EC    A9 68       LDA #$68        to $BF68 and that of the
F8EE    8D 7A 02    STA $027A       second line to $BF90.
F8F1    A9 90       LDA #$90
F8F3    8D 78 02    STA $0278
F8F6    A9 03       LDA #$03        Set the maximum number of
F8F8    8D 7E 02    STA $027E       rows of text available.
F8FB    A9 00       LDA #$00
F8FD    8D 7D 02    STA $027D       Set number of characters used
F900    A9 50       LDA #$50        in screen scrolling to 80 -
F902    8D 7C 02    STA $027C       two lines worth.
F905    A2 0C       LDX #$0C
F907    20 38 02    JSR $0238       Clear screen.
F90A    68          PLA
F90B    AA          TAX
F90C    68          PLA
F90D    60          RTS

F90E    48          PHA             Set up default state of flags
F90F    A9 03       LDA #$03        controlling screen.
F911    8D 6A 02    STA $026A
F914    A9 00       LDA #$00
F916    8D 6C 02    STA $026C       Set ink to black.
F919    A9 17       LDA #$17
F91B    8D 6B 02    STA $026B       Set paper to white.
F91E    68          PLA
F91F    60          RTS

F920    48          PHA             SET SCREEN TO HIRES.
F921    AD 1F 02    LDA $021F
F924    D0 05       BNE $F92B
F926    A2 0B       LDX #$0B
F928    20 82 F9    JSR $F982
F92B    A9 FE       LDA #$FE        Disable cursor.
F92D    2D 6A 02    AND $026A
F930    8D 6A 02    STA $026A
F933    A9 1E       LDA #$1E        Write 50Hz attribute to last
F935    8D DF BF    STA $BFDF       location on screen.
F938    A9 40       LDA #$40
F93A    8D 00 A0    STA $A000
F93D    A2 17       LDX #$17
F93F    20 82 F9    JSR $F982
F942    A9 00       LDA #$00        Set X and Y cursor coordinates
F944    8D 19 02    STA $0219       to zero.
F947    8D 1A 02    STA $021A
F94A    85 10       STA $10         Set cursor address to #A000.
F94C    A9 A0       LDA #$A0
F94E    85 11       STA $11
F950    A9 20       LDA #$20        Set cursor position within
F952    8D 15 02    STA $0215       byte on screen.
F955    A9 FF       LDA #$FF
F957    8D 13 02    STA $0213       Set pattern register.
F95A    20 DC F8    JSR $F8DC       Set up some page 2 variables.
F95D    A9 01       LDA #$01        Re-enable cursor.
F95F    0D 6A 02    ORA $026A
F962    8D 6A 02    STA $026A
F965    68          PLA
F966    60          RTS

F967    48          PHA             SET SCREEN TO TEXT.
F968    A9 FE       LDA #$FE
F96A    2D 6A 02    AND $026A       Disable cursor.
F96D    8D 6A 02    STA $026A
F970    A2 11       LDX #$11        Copy char sets into original
```

```
F972   20 82 F9    JSR $F982       position in memory.
F975   20 C9 F9    JSR $F9C9       Set pointers.
F978   A9 01       LDA #$01
F97A   0D 6A 02    ORA $026A       Re-enable cursor.
F97D   8D 6A 02    STA $026A
F980   68          PLA
F981   60          RTS

F982   A0 06       LDY #$06        This routine writes addresses
F984   BD 92 F9    LDA $F992,X     from table below to locations
F987   99 0B 00    STA $000B,Y     #0C to #11 inclusive. The
F98A   CA          DEX             value of X determines which
F98B   88          DEY             part of the table is copied.
F98C   D0 F6       BNE $F984       The data is then used in a
F98E   20 C4 ED    JSR $EDC4       block transfer routine.
F991   60          RTS

F992   78 FC 00 B5 00 03 00 B4
F99A   00 98 80 07 00 98 00 B4
F9A2   80 07 00 A0 01 A0 3F 1F

F9AA   A9 FF       LDA #$FF        RESET 6522.
F9AC   8D 03 03    STA $0303       Port A all output.
F9AF   A9 F7       LDA #$F7        Port B all output except
F9B1   8D 02 03    STA $0302       bit 4.
F9B4   A9 B7       LDA #$B7        Turn off cassette motor.
F9B6   8D 00 03    STA $0300
F9B9   A9 DD       LDA #$DD        Set CA2 and CB2 to 0 and set
F9BB   8D 0C 03    STA $030C       CA1 and CB1 active L to H.
F9BE   A9 7F       LDA #$7F
F9C0   8D 0E 03    STA $030E       Disable all interrupts.
F9C3   A9 00       LDA #$00
F9C5   8D 0B 03    STA $030B       Set the ACR.
F9C8   60          RTS

F9C9   A9 1A       LDA #$1A        Set up TEXT SCREEN.
F9CB   20 07 FA    JSR $FA07       Write 50Hz attribute to last
F9CE   A9 20       LDA #$20        screen location and clear
F9D0   A0 28       LDY #$28        line.
F9D2   99 7F BB    STA $BB7F,Y
F9D5   88          DEY
F9D6   D0 FA       BNE $F9D2
F9D8   A9 00       LDA #$00
F9DA   8D 1F 02    STA $021F       Set screen status to lores.
F9DD   A9 BB       LDA #$BB
F9DF   8D 7B 02    STA $027B       Set the address of the first
F9E2   8D 79 02    STA $0279       line of text to $BBA8 and that
F9E5   A9 A8       LDA #$A8        of the second to $BBD0.
F9E7   8D 7A 02    STA $027A
F9EA   A9 D0       LDA #$D0
F9EC   8D 78 02    STA $0278
F9EF   A9 1B       LDA #$1B        Set number of rows of text
F9F1   8D 7E 02    STA $027E       available to 27.
F9F4   A9 04       LDA #$04
F9F6   8D 7D 02    STA $027D       Set number of characters that
F9F9   A9 10       LDA #$10        are moved in screen scroll to
F9FB   8D 7C 02    STA $027C       #0410 (1040 or 26 lines full).
F9FE   A2 0C       LDX #$0C
FA00   20 38 02    JSR $0238       Clear screen.
FA03   20 5A F7    JSR $F75A       Write CAPS to screen if on.
FA06   60          RTS

FA07   8D DF BF    STA $BFDF       This routine writes A to
FA0A   A9 02       LDA #$02        very last location on screen
```

```
FA0C   A2 00       LDX #$00        and then waits for 40mS
FA0E   A0 03       LDY #$03
FA10   20 C9 EE    JSR $EEC9
FA13   60          RTS

FA14   A0 00       LDY #$00        TEST AND FIND QUANTITY 0F RAM
FA16   8C 60 02    STY $0260
FA19   8C 20 02    STY $0220
FA1C   8C 00 05    STY $0500
FA1F   84 0E       STY $0E
FA21   88          DEY
FA22   84 0C       STY $0C
FA24   8C 00 45    STY $4500
FA27   AD 00 05    LDA $0500
FA2A   D0 04       BNE $FA30       Branch if 16k computer.
FA2C   A9 C0       LDA #$C0
FA2E   D0 05       BNE $FA35
FA30   EE 20 02    INC $0220       $220=1 for 16k, 0 for 48k.
FA33   A9 40       LDA #$40        A holds high byte of possible
FA35   85 0F       STA $0F         extent of ram.
FA37   C8          INY
FA38   A9 03       LDA #$03        $C and $E are used as to test
FA3A   85 0D       STA $0D         each byte in turn. $C is used
FA3C   E6 0C       INC $0C         as current location and $E is
FA3E   D0 02       BNE $FA42       used as end of memory pointer.
FA40   E6 0D       INC $0D
FA42   A5 0C       LDA $0C
FA44   C5 0E       CMP $0E
FA46   D0 06       BNE $FA4E
FA48   A5 0D       LDA $0D
FA4A   C5 0F       CMP $0F
FA4C   F0 0F       BEQ $FA5D
FA4E   A9 AA       LDA #$AA
FA50   91 0C       STA ($0C),Y
FA52   D1 0C       CMP ($0C),Y
FA54   D0 07       BNE $FA5D
FA56   4A          LSR A
FA57   91 0C       STA ($0C),Y
FA59   D1 0C       CMP ($0C),Y
FA5B   F0 DF       BEQ $FA3C
FA5D   38          SEC
FA5E   A5 0F       LDA $0F
FA60   E9 28       SBC #$28
FA62   85 0F       STA $0F
FA64   A5 0E       LDA $0E
FA66   C5 0C       CMP $0C
FA68   A5 0F       LDA $0F
FA6A   E5 0D       SBC $0D
FA6C   90 09       BCC $FA77
FA6E   A5 0C       LDA $0C
FA70   A4 0D       LDY $0D
FA72   EE 60 02    INC $0260
FA75   D0 04       BNE $FA7B
FA77   A5 0E       LDA $0E
FA79   A4 0F       LDY $0F
FA7B   85 A6       STA $A6
FA7D   84 A7       STY $A7
FA7F   8D C1 02    STA $02C1
FA82   8C C2 02    STY $02C2
FA85   60          RTS

FA86   08          PHP             This routine takes X and Y
FA87   78          SEI             as the low and high halves of
FA88   86 14       STX $14         the start address of a table
```

```
FA8A   84 15       STY $15          to send data to the sound
FA8C   A0 00       LDY #$00         chip from.
FA8E   B1 14       LDA ($14),Y      14 bytes are sent to the 8912
FA90   AA          TAX              starting with register 0 and
FA91   98          TYA              working up in order until
FA92   48          PHA              register D. The data from
FA93   20 90 F5    JSR $F590        the table is used starting
FA96   68          PLA              from the low address.
FA97   A8          TAY
FA98   C8          INY              The 1/0 port is not written
FA99   C0 0E       CPY #$0E         to.
FA9B   D0 F1       BNE $FA8E
FA9D   28          PLP
FA9E   60          RTS

FA9F   A2 A7       LDX #$A7         PING
FAA1   A0 FA       LDY #$FA         Sets X and Y to point to the
FAA3   20 86 FA    JSR $FA86        data below to generate the
FAA6   60          RTS              sound.

FAA7   18 00 00 00 00 00 00         Data for Ping command.
FAAE   3E 10 00 00 00 0F 00

FAB5   A2 BD       LDX #$BD         SHOOT
FAB7   A0 FA       LDY #$FA         Sets X and Y to point to the
FAB9   20 86 FA    JSR $FA86        data below to generate the
FABC   60          RTS              sound.

FABD   00 00 00 00 00 00 0F         Data for Shoot command.
FAC4   07 10 10 10 00 08 00

FACB   A2 D3       LDX #$D3         EXPLODE
FACD   A0 FA       LDY #$FA         Sets X and Y to point to the
FACF   20 86 FA    JSR $FA86        data below to generate the
FAD2   60          RTS              sound.

FAD3   00 00 00 00 00 00 1F         Data for Explode command.
FADA   07 10 10 10 00 18 00

FAE1   A2 06       LDX #$06         ZAP
FAE3   A0 FB       LDY #$FB         send sound data to 8912 as
FAE5   20 86 FA    JSR $FA86        in Shoot etc.
FAE8   A9 00       LDA #$00
FAEA   AA          TAX              This section writes to the
FAEB   8A          TXA              tone channel A at regular
FAEC   48          PHA              intervals with increasing
FAED   A9 00       LDA #$00         tone periods. Thus
FAEF   20 90 F5    JSR $F590        successively lower frequncies
FAF2   A2 00       LDX #$00         are produced. The delay loop
FAF4   CA          DEX              takes about 1.25mS to
FAF5   D0 FD       BNE $FAF4        execute.

FAF7   68          PLA
FAF8   AA          TAX
FAF9   E8          INX
FAFA   E0 70       CPX #$70         The main loop is executed 112
FAFC   D0 ED       BNE $FAEB        times in total.
FAFE   A9 08       LDA #$08         Zero channel A amplitude.
FB00   A2 00       LDX #$00
FB02   20 90 F5    JSR $F590
FB05   60          RTS

FB06   00 00 00 00 00 00 00         Data for ZAP command.
FB0D   3E 0F 00 00 00 00 00
```

```
FB14   A2 1C       LDX #$1C        KEYCLICK high pitch.
FB16   A0 FB       LDY #$FB        Sets X and Y to point to the
FB18   20 86 FA    JSR $FA86       data below to generate the
FB1B   60          RTS             sound.

FB1C   1F 00 00 00 00 00 00        Data for high pitch keyclick.
FB23   3E 10 00 00 1F 00 00

FB2A   A2 32       LDX #$32        KEYCLICK low pitch.
FB2C   A0 FB       LDY #$FB        Sets X and Y to point to the
FB2E   20 86 FA    JSR $FA86       data below to generate the
FB31   60          RTS             sound.

FB32   2F 00 00 00 00 00 00        Data for low pitch keyclick.
FB39   3E 10 00 00 1F 00 00

FB40   AD E1 02    LDA $02E1       SOUND
FB43   C9 01       CMP #$01        Branch if tone channel A is
FB45   D0 22       BNE $FB69       not being used.
FB47   A9 00       LDA #$00        Write the tone period for
FB49   AE E3 02    LDX $02E3       channel A to the sound chip
FB4C   20 90 F5    JSR $F590       Write low byte of period.
FB4F   A9 01       LDA #$01
FB51   AE E4 02    LDX $02E4
FB54   20 90 F5    JSR $F590       Write high byte of period.
FB57   AD E5 02    LDA $02E5       Load amplitude and keep it in
FB5A   29 0F       AND #$0F        the range 0-15. If amplitude
FB5C   D0 04       BNE $FB62       is zero then use envelope
FB5E   A2 10       LDX #$10        control.
FB60   D0 01       BNE $FB63
FB62   AA          TAX
FB63   A9 08       LDA #$08
FB65   20 90 F5    JSR $F590
FB68   60          RTS

FB69   C9 02       CMP #$02        Branch if tone channel B is
FB6B   D0 22       BNE $FB8F       not being used.
FB6D   A9 02       LDA #$02
FB6F   AE E3 02    LDX $02E3       Write low byte of tone period
FB72   20 90 F5    JSR $F590       to the sound chip.
FB75   A9 03       LDA #$03
FB77   AE E4 02    LDX $02E4       Write high byte of tone period
FB7A   20 90 F5    JSR $F590       to the sound chip.
FB7D   AD E5 02    LDA $02E5       Load and set amplitude in
FB80   29 0F       AND #$0F        range 0-15.
FB82   D0 04       BNE $FB88
FB84   A2 10       LDX #$10        If amplitude is zero then use
FB86   D0 01       BNE $FB89       envelope control.
FB88   AA          TAX
FB89   A9 09       LDA #$09
FB8B   20 90 F5    JSR $F590
FB8E   60          RTS

FB8F   C9 03       CMP #$03        Branch if tone channel C is
FB91   D0 22       BNE $FBB5       not being used.
FB93   A9 04       LDA #$04
FB95   AE E3 02    LDX $02E3       Write low byte of tone period
FB98   20 90 F5    JSR $F590       to the sound chip.
FB9B   A9 05       LDA #$05
FB9D   AE E4 02    LDX $02E4       Write high byte of tone period
FBA0   20 90 F5    JSR $F590       to the sound chip.
FBA3   AD E5 02    LDA $02E5       Load and set the amplitude in
FBA6   29 0F       AND #$0F        the range 0-15.
```

```
FBA8   D0 04       BNE $FBAE
FBAA   A2 10       LDX #$10        If amplitude is zero then use
FBAC   D0 01       BNE $FBAF       envelope control.
FBAE   AA          TAX
FBAF   A9 0A       LDA #$0A
FBB1   20 90 F5    JSR $F590
FBB4   60          RTS

FBB5   A9 06       LDA #$06        This routine sets up the noise
FBB7   AE E3 02    LDX $02E3       period to be used.
FBBA   20 90 F5    JSR $F590       Sound channels 4, 5 & 6
FBBD   AD E1 02    LDA $02E1       produce noise on tone channels
FBC0   C9 04       CMP #$04        A, B & C respectively.
FBC2   F0 93       BEQ $FB57
FBC4   C9 05       CMP #$05
FBC6   F0 B5       BEQ $FB7D
FBC8   C9 06       CMP #$06
FBCA   F0 D7       BEQ $FBA3       An error is produced if the
FBCC   EE E0 02    INC $02E0       sound channels are not in
FBCF   60          RTS             correct range.

FBD0   AD E3 02    LDA $02E3       PLAY
FBD3   0A          ASL A           Combine the tone and sound
FBD4   0A          ASL A           channels into a single byte.
FBD5   0A          ASL A           Invert the result and send it
FBD6   0D E1 02    ORA $02E1       to the mixer register in the
FBD9   49 3F       EOR #$3F        sound chip.
FBDB   AA          TAX
FBDC   A9 07       LDA #$07
FBDE   20 90 F5    JSR $F590
FBE1   18          CLC
FBE2   AD E7 02    LDA $02E7       Double the duration given in
FBE5   0A          ASL A           the command.
FBE6   8D E7 02    STA $02E7
FBE9   AD E8 02    LDA $02E8
FBEC   2A          ROL A
FBED   8D E8 02    STA $02E8
FBF0   A9 0B       LDA #$0B
FBF2   AE E7 02    LDX $02E7       Write low byte of envelope
FBF5   20 90 F5    JSR $F590       period to 8912.
FBF8   A9 0C       LDA #$0C
FBFA   AE E8 02    LDX $02E8       Write high byte of envelope
FBFD   20 90 F5    JSR $F590       period to 8912.
FC00   AD E5 02    LDA $02E5
FC03   29 07       AND #$07
FC05   A8          TAY
FC06   B9 10 FC    LDA $FC10,Y     Look up envelope pattern
FC09   AA          TAX             using table below.
FC0A   A9 0D       LDA #$0D
FC0C   20 90 F5    JSR $F590
FC0F   60          RTS

FC10   00 00 04 08 0A 0B 0C 0D     Envelope patterns used.

FC18   A2 E1       LDX #$E1        MUSIC
FC1A   A9 04       LDA #$04        Test channel for range.
FC1C   20 E4 F2    JSR $F2E4
FC1F   B0 39       BCS $FC5A       Channel number out of range.
FC21   A2 E3       LDX #$E3
FC23   A9 08       LDA #$08        Test ocatve range.
FC25   20 F8 F2    JSR $F2F8
FC28   B0 30       BCS $FC5A       Octave number out of range.
FC2A   A2 E5       LDX #$E5
FC2C   A9 0D       LDA #$0D        Test note range.
```

```
FC2E   20 E4 F2    JSR $F2E4
FC31   B0 27       BCS $FC5A       Note number is out of range.
FC33   AC E3 02    LDY $02E3       Use the octave and note
FC36   AE E5 02    LDX $02E5       values to look up the tone
FC39   BD 5E FC    LDA $FC5E,X     periods in the table below.
FC3C   8D E4 02    STA $02E4
FC3F   BD 6B FC    LDA $FC6B,X
FC42   8D E3 02    STA $02E3
FC45   AD E7 02    LDA $02E7
FC48   8D E5 02    STA $02E5
FC4B   88          DEY
FC4C   30 09       BMI $FC57
FC4E   4E E4 02    LSR $02E4
FC51   6E E3 02    ROR $02E3
FC54   4C 4B FC    JMP $FC4B
FC57   4C 40 FB    JMP $FB40       Goto Sound command.
FC5A   EE E0 02    INC $02E0
FC5D   60          RTS

FC5E   00 07 07 06 06 05 05 05     Data for the Music command.
FC66   04 04 04 04 03 00 77 0B     Converts the notes into tone
FC6E   A6 47 EC 97 47 FB B3 70     periods.
FC76   30 F4

FC78   00 00 00 00 00 00 00 00     Space Start of standard
FC80   08 08 08 08 08 00 08 00     !     character set. Each
FC88   14 14 14 00 00 00 00 00     "     row of 8 bytes
FC90   14 14 3E 14 3E 14 14 00     #     represents the bit
FC98   08 1E 28 1C 0A 3C 08 00     $     pattern for each
FCA0   30 32 04 08 10 26 06 00     %     character. The first
FCA8   10 28 28 10 2A 24 1A 00     &     byte is the bit
FCB0   08 08 08 00 00 00 00 00     '     pattern for the top
FCB8   08 10 20 20 20 10 08 00     (     row and the last is
FCC0   08 04 02 02 02 04 08 00     )     that for the bottom
FCC8   08 2A 1C 08 1C 2A 08 00     *     row. The list works
FCD0   00 08 08 3E 08 08 00 00     +     its way up the Ascii
FCD8   00 00 00 00 00 08 08 10     ,     set from SPACE to DEL.
FCE0   00 00 00 3E 00 00 00 00     -
FCE8   00 00 00 00 00 04 00 00     .     On power up, this data
FCF0   00 02 04 08 10 20 00 00     /     is copied to below the
FCF8   1C 22 26 2A 32 22 1C 00     0     screen memory.
FD00   08 18 08 08 08 08 1C 00     1
FD08   1C 22 02 04 08 10 3E 00     2
FD10   3E 02 04 0C 02 22 1C 00     3
FD18   04 0C 14 24 3E 04 04 00     4
FD20   3E 20 3C 02 02 22 1C 00     5
FD28   0C 10 20 3C 22 22 1C 00     6
FD30   3E 02 04 08 10 10 10 00     7
FD38   1C 22 22 1C 22 22 1C 00     8
FD40   1C 22 22 1E 02 04 18 00     9
FD48   00 00 08 00 00 08 00 00     :
FD50   00 00 08 00 00 08 08 10     ;
FD58   04 08 10 20 10 08 04 00     <
FD60   00 00 3E 00 3E 00 00 00     =
FD68   10 08 04 02 04 08 10 00     >
FD70   1C 22 04 08 08 00 08 00     ?
FD78   1C 22 2A 2E 2C 20 1E 00     @
FD80   08 14 22 22 3E 22 22 00     A
FD88   3C 22 22 3C 22 22 3C 00     B
FD90   1C 22 20 20 20 22 1C 00     C
FD98   3C 22 22 22 22 22 3C 00     D
FDA0   3E 20 20 3C 20 20 3E 00     E
FDA8   3E 20 20 3C 20 20 20 00     F
FDB0   1E 20 20 20 26 22 1E 00     G
```

```
FDB8    22 22 22 3E 22 22 22 00      H
FDC0    1C 08 08 08 08 08 1C 00      I
FDC8    02 02 02 02 02 22 1C 00      J
FDD0    22 24 28 30 28 24 22 00      K
FDD8    20 20 20 20 20 20 3E 00      L
FDE0    22 36 2A 2A 22 22 22 00      M
FDE8    22 22 32 2A 26 22 22 00      N
FDF0    1C 22 22 22 22 22 1C 00      0
FDF8    3C 22 22 3C 20 20 20 00      P
FE00    1C 22 22 22 2A 24 1A 00      Q
FE08    3C 22 22 3C 28 24 22 00      R
FE10    1C 22 20 1C 02 22 1C 00      S
FE18    3E 08 08 08 08 08 08 00      T
FE20    22 22 22 22 22 22 1C 00      U
FE28    22 22 22 22 22 14 08 00      V
FE30    22 22 22 2A 2A 36 22 00      W
FE38    22 22 14 08 14 22 22 00      X
FE40    22 22 14 08 08 08 08 00      Y
FE48    3E 02 04 08 10 20 3E 00      Z
FE50    1E 10 10 10 10 10 1E 00      [
FE58    00 20 10 08 04 02 00 00      \
FE60    3C 04 04 04 04 04 3C 00      ]
FE68    08 14 2A 08 08 08 08 00      f
FE70    0E 10 10 10 3C 10 3E 00      g
FE78    0C 12 2D 29 29 2D 12 0C      ©
FE80    00 00 1C 02 1E 22 1E 00      a
FE88    20 20 3C 22 22 22 3C 00      b
FE90    00 00 1E 20 20 20 1E 00      c
FE98    02 02 1E 22 22 22 1E 00      d
FEA0    00 00 1C 22 3E 20 1E 00      e
FEA8    0C 12 10 3C 10 10 10 00      f
FEB0    00 00 1C 22 22 1E 02 1C      g
FEB8    20 20 3C 22 22 22 22 00      h
FEC0    08 00 18 08 08 08 1C 00      i
FEC8    04 00 0C 04 04 04 24 18      j
FED0    20 20 22 24 38 24 22 00      k
FED8    18 08 08 08 08 08 1C 00      1
FEE0    00 00 36 2A 2A 2A 22 00      m
FEE8    00 00 3C 22 22 22 22 00      n
FEF0    00 00 1C 22 22 22 1C 00      o
FEF8    00 00 3C 22 22 3C 20 20      p
FF00    00 00 1E 22 22 1E 02 02      q
FF08    00 00 2E 30 20 20 20 00      r
FF10    00 00 1E 20 1C 02 3C 00      s
FF18    10 10 3C 10 10 12 0C 00      t
FF20    00 00 22 22 22 26 1A 00      u
FF28    00 00 22 22 22 14 08 00      v
FF30    00 00 22 22 2A 2A 36 00      w
FF38    00 00 22 14 08 14 22 00      x
FF40    00 00 22 22 22 1E 02 1C      y
FF48    00 00 3E 04 08 10 3E 00      z
FF50    0E 18 18 30 18 18 0E 00      {
FF58    08 08 08 08 08 08 08 08      |
FF60    38 0C 0C 06 0C 0C 38 00      }
FF68    2A 15 2A 15 2A 15 2A 15      Chequered grid.
FF70    3F 3F 3F 3F 3F 3F 3F 3F      DEL

FF78    37 EA ED EB 20 F5 F9 38      Look up table for the
FF80    EE F4 36 39 2C E9 E8 EC      conversion of the key-code to
FF88    35 F2 E2 3B 2E EF E7 30      corresponding Ascii character.
FF90    F6 E6 34 2D 0B F0 E5 2F      The first half of the table
FF98    00 00 00 00 00 00 00 00      corresponds to the Ascii
FFA0    31 1B FA 00 08 7F E1 0D      values with the shift key off.
FFA8    F8 F1 32 5C 0A 5D F3 00      The second half of the table
```

```
FFB0    33 E4 E3 27 09 5B F7 3D      corresponds to the Ascii
FFB8    26 4A 4D 4B 20 55 59 2A      values with the shift key
FFC0    4E 54 5E 28 3C 49 48 4C      pressed.
FFC8    25 52 42 3A 3E 4F 47 29
FFD0    56 46 24 5F 0B 50 45 3F
FFD8    00 00 00 00 00 00 00 00
FFE0    21 1B 5A 00 08 7F 41 0D
FFE8    58 51 40 7C 0A 7D 53 00
FFF0    23 44 43 22 09 7B 57 2B
FFF8    D0 01

FFFA    47 02       N.M.I.           Vector, $0247
FFFC    8F F8       RESET            Vector, $F88F
FFFE    44 02       I.R.Q.           Vector, $0244
```

Appendix A

Token Table

Token	Keyword	Start Address V1.0	Start Address V1.1	Token	Keyword	Start Address V1.0	Start Address V1.1
#80	END	#C941	#C973	#81	EDIT	#C6A5	#C692
#82	*1	#CFE4	#E987	#83	*2	#CFE4	#E9D1
#84	TRON	#CC8C	#CD16	#85	TROFF	#CC8F	#CD19
#86	POP	#C9E0	#CA12	#87	PLOT	#D9C6	#DA51
#88	PULL	#DA16	#DAA1	#89	LORES	#D937	#D9DE
#8A	DOKE	#D8AC	#D967	#8B	REPEAT	#D9FA	#DA85
#8C	UNTIL	#DA16	#DAA1	#8D	FOR	#C841	#C855
#8E	LLIST	#C824	#C7FD	#8F	LPRINT	#C832	#C809
#90	NEXT	#CE0C	#CE98	#91	DATA	#CA0A	#CA3C
#92	INPUT	#CCC9	#CD55	#93	DIM	#D0F2	#D17E
#94	CLS	#CC0A	#CCCE	#95	READ	#CCFD	#CD89
#96	LET	#CAD2	#CB1C	#97	GOTO	#C9B3	#C9E5
#98	RUN	#C98B	#C9BD	#99	IF	#CA3E	#CA70
#9A	RESTORE	#C91F	#C952	#9B	60SUB	#C996	#C9C8
#9C	RETURN	#C9E0	#CA12	#9D	REM	#CA61	#CA99
#9E	HIMEM	#E95B	#EBCE	#9F	GRAB	#E974	#EBE7
#A0	RELEASE	#E994	#EC0C	#A1	TEXT	#E9A9	#EC21
#A2	HIRES	#E9BB	#EC33	#A3	SHOOT	#F415	#FAB5
#A4	EXPLODE	#F418	#FACB	#A5	ZAP	#F41B	#FAE1
#A6	PING	#F41B	#FA9F	#A7	SOUND	#E889	#EAFC
#A8	MUSIC	#E889	#EAFC	#A9	PLAY	#E889	#EAFC
#AA	CURSET	#E87D	#EAF0	#AB	CURMOV	#E87D	#EAF0
#AC	DRAW	#E87D	#EAF0	#AD	CIRCLE	#E87D	#EAF0
#AE	PATTERN	#E87D	#EAF0	#AF	FILL	#E87D	#EAF0
#B0	CHAR	#E87D	#EAF0	#B1	PAPER	#E889	#EAFC
#B2	INK	#E889	#EAFC	#B3	STOP	#C93F	#C971
#B4	ON	#CA78	#CAC2	#B5	WAIT	#D89D	#D958
#B6	CLOAD	#E7AA	#E85B	#B7	CSAVE	#E7DB	#E909
#B8	DEF	#D401	#D4BA	#B9	POKE	#D894	#D94F
#BA	PRINT	#CB61	#CBAB	#BB	CONT	#C96E	#C9A0
#BC	LIST	#C773	#C748	#BD	CLEAR	#C738	#C70D
#BE	GET	#CCBA	#CD46	#BF	CALL	#E80D	#E946
#C0	!	#CC89	#CD13	#C1	NEW	#C719	#C6EE
#C2	TAB(	–	–	#C3	TO	–	–
#C4	FN	–	–	#C5	SPC	–	–
#C6	@	–	–	#C7	AUTO	–	–
#C8	ELSE	–	–	#C9	THEN	–	–
#CA	NOT	–	–	#CB	STEP	–	–
#CC	+	–	–	#CD	–	–	–
#CE	*	–	–	#CF	/	–	–
#D0	^	–	–	#D1	AND	–	–
#D2	OR	–	–	#D3	>	–	–
#D4	=	–	–	#D5	<	–	–
#D6	SGN	#DF12	#DF21	#D7	INT	#DFA5	#DFBD
#D8	ABS	#DF31	#DF49	#D9	USR	#0021	#0021
#DA	FRE	#D3D6	#D47E	#DB	POS	#D3FA	#D4A6
#DC	HEX$	#D917	#D9B5	#DD	&	#02FB	#02FB
#DE	SQR	#E22A	#E22E	#DF	RND	#E34B	#E34F
#E0	LN	#DC79	#DCAF	#E1	EXP	#E2A6	#E2AA
#E2	COS	#E387	#E38B	#E3	SIN	#E38E	#E392
#E4	TAN	#E3D7	#E3DB	#E5	ATN	#E43B	#E43F
#E6	PEEK	#D87D	#D938	#E7	DEEK	#D8C8	#D983
#E8	LOG	#DDD0	#DDD4	#E9	LEN	#D7EB	#D8A6
#EA	STR$	#D4D8	#D593	#EB	VAL	#D81C	#D8D7
#EC	ASC	#D7FA	#D8B5	#ED	CHR$	#D75B	#D816
#EE	PI	#D8EE	#DE77	#EF	TRUE	#DF00	#DF0F
#F0	FALSE	#DEFC	#DF0B	#F1	KEY$	#DA4F	#DADA

#F2	SCRN	#D9B4	#DA3F	#F3	POINT	#E9CD	#EC45
#F4	LEFT$	#D76F	#D82A	#F5	RIGHT$	#D79B	#D856
#F6	MID$	#D7A6	#D861	–			

Appendix C

Page 0 Allocation
This page of memory (*between #0000 and #00FF*) is used to store most of the variables used by BASIC and a few of those used by the operating system. Many of the locations are used for more than one purpose, those without comments are not used. The use of each memory location is the same for the **Oric-1** and **Atmos** unless otherwise indicated. This is done by indicating the ROM version of the computer to which the description is applicable. The version number is written at the top of the screen on power up.

Address	Function
$00-$0B	
$0C,$0D	Indirect pointer for screen and Hex number construction area.
$0E,$0F	Indirect pointer for the screen.
$10,$11	Address of hires cursor.
$12,$13	Address of text cursor.
$14—$16	Expression workspace.
$17	Set to 1 if CTRL C pressed, otherwise 0.
$18,$19	Tokenising pointer.
$1A—$1C	Jump location to print 'Ready'.
$1D,$1E	Counter for searching through lines of program.
$1F,$20	Calculation of cursor address.
$21—$23	Jump location for USR command.
$24—$26	Expression workspace.
$27	Temporary storage — often for characters being printed.
$28	Set to #FF if dealing with strings.
$29	Bit 7 is set if using integer variable.
$2A	Garbage collection flag or flag for skipping through DATA statements.
$2B	Bit 7 when set inhibits the use of integers. Bit 6 when set indicates STORE or RECALL commands in use.
$2C	Zero if REDOing input FROM START.
$2D	Temporary storage for expression evaluator.
$2E	CTRL O flag. 0 if output to screen enabled.
$2F	Next byte to/from cassette.
$30	Cursor position for Basic printout.
$31	Screen line width.
$32	8 - multiple line width.
$33,$34	Integer values to/from main floating point accumulator.
$35-$84	Input buffer. (*79 bytes*)
$35-$48	Name of program required for CLOAD (V1.0 only).
$49-$5D	Name of program just loaded (V1.0 only).
$5F,$60	Start address of data to/from tape (V1.0 only).
$61,$62	End address of data to/from tape (V1.0 only).
$63	1 when using AUTO else 0 (V1.0 only).
$64	0 for Basic, 1 for machine code (Vl.0 only).
$67	Tape speed - 0 fast, 1 slow (Vl.0 only).
$85	String block stack pointer.
$86,$87	Address of top active string in memory.
$88-$90	Temporary string stack.
$91,$92	String address pointer.
$93,$94	General memory pointer.
$95—$99	Work area for multiply and divide routines.
$9A,$9B	Start of Basic pointer.
$9C,$9D	End of Basic pointer.
$9E,$9F	End of variables pointer.
$A0,$A1	End of Arrays pointer.
$A2,$A3	Bottom of string area pointer.
$A4,$A5	Work pointer for allocating strings.

$A6,$A7	Himem.
$A8,$A9	Current line number, top byte is #FF if in command mode.
$AA,$AB	Previous line number.
$AC,$AD	Last line start address.
$AE,$AF	Temporary copy of line number.
$B0,$B1	Data pointer.
$B2,$B3	Data pointer.
$B4,$B5	Last variable name accessed.
$B6,$B7	Address of last variable value accessed.
$B8,$B9	Destination pointer for temporary assignment of variable.
$BA	Temporary storage for expression evaluator.
$BB,$BC	
$BD—$C1	Temporary storage of floating point accumulator.
$BD,$BE	FN (function) pointer.
$BF,$C0	String pointer.
$C2	String pointer size, used in Garbage Collection.
$C3—$C5	Jump location to evaluate numeric functions. $C5 also used as a temporary store of the rounding byte for mathematical operations.
$C6—$CA	Temporary storage of floating point accumulator.
$C7,$C8	Pointer.
$C9,$CA	Pointer.
$CB—$CF	Temporary storage of floating point accumulator.
$CE,$CF	Pointer for STORE.
$D0	Exponent of main floating point accumulator.
$D1—$D4	Mantissa of main floating point accumulator.
$D5	Sign of mantissa for main FPA when unpacked.
$D6	Series evaluation counter.
$D7	Sign extend byte.
$D8	Exponent of work floating point accumulator.
$D9—$DC	Mantissa of work floating point accumulator.
$DD	Sign of mantissa for work FPA when unpacked.
$DE,$DF	String pointer.
$DE	Holds Exclusive OR of sign byte of both FPAs.
$DF	Rounding byte for calculations.
$E0,$E1	Array and string workspace.
$E2—$F2	Routine to step through program to find next non space char. *See $EC9C of disassembly*.
$E9,$EA	Position pointer in program.
$F3-$F9	
$FA—$FE	Copy of floating point number used by RND.
$FF	Used in number to string conversion.

Appendix D

Page 2 Allocation
This page of memory (*between #0200 and #02FF*) is used to store most of the variables used by the operating system and a few of those used by BASIC. Many of the locations are used for more than one purpose, those without comments are not used. The use of each memory location is the same for the **Oric-1** and **Atmos** unless otherwise indicated. This is done by indicating the ROM version of the computer to which the description is applicable. The version number is written at the top of the screen on power up.

Address	Function
$200,$201	Pointer for screen handling.
$202,$203	Pointer for screen handling.
$204-$207	Work bytes for Hires routines.
$208	Key address if pressed — #38 if no key pressed.
$209	Key status, #38 - Default, #A2 - CONTROL, #A4 - Left SHIFT, #A5 - FUNCTION (*Atmos only*), #A7 - Right SHIFT.
$20A	Saved key column for repeat.
$20B	Not used but gets written over by routine that sets $208 - $20A.
$20C	Bit 7 is set if CAPS is on otherwise clear.
$20D	
$20E	Repeat counter for keyboard.
$20F	
$210	Temporary store of row of key being tested for repeat.
$211	Temporary store of keyboard row during strobe routine.
$212	Holds FB code in hires commands.
$213	Pattern data for hires screen.
$214	Temporary copy of pattern byte for drawing lines.
$215	Holds position of pixel in byte corresponding to position of cursor on hires screen.
$216,$217	Temporary store of hires X and Y cursor positions.
$218	Temporary store for content of $215.
$219	Hires cursor — X coordinate.
$21A	Hires cursor — Y coordinate.
$21B-$21E	
$21F	0 - LORES, 1 - HIRES.
$220	0 - 48K Oric, 1 - 16K Oric.
$221-$227	
$228-$22A	Jump to IRQ routine (V1.0).
$22B-$22D	Jump to NMI routine (V1.0).
$22E,$22F	
$230	RTI instruction (V1.0).
$231-$237	
$238-$23A	Jump to PRINT CHARACTER on screen (V1.1).
$23B-$23D	Jump to GET KEY routine (V1.1).
$23E-$240	Jump to SEND BYTE TO PRINTER (V1.1).
$241-$243	Jump to PRINT TO STATUS LINE (V1.l).
$244-$246	Jump to IRQ routine (Vl.l).
$247-$249	Jump to NMI routine (Vl.l).
$24A—$24C	RTI instruction which can be intercepted by a jump (Vl.l).
$24D	Tape speed, 0 - fast, 1 - slow (Vl.l).
$24E	Keyboard initial repeat deiay (V1.l).
$24F	Keyboard successive repeat delay (V1.1).
$250	
$251	Cursor enabie in CTRL routines (Vl.l).

$252 ELSE pending flag, 1 - on, 0-off (Vl.l).
$253-$255
$256 Printer width (Vl.l).
$257 Screen width (Vl.l).
$258 Printer cursor position (V1.1).
$259 Screen cursor position (V1.1).
$25A Cassette JOIN flag, off when 0 (V1.1).
$25B Cassette VERIFY flag, off when 0 (Vl.l).
$25C,$25D Cassette verify error counter (Vl.l).
$25E
$25F,$260 Contains 1 byte messages printed to status line.
$261,$262 Indirect jump for CTRL character routine.
$263,$264 Temporary storage.
$265 Current cursor state indicator, 0 - off, 1 — on.
$266,$267
$268 Cursor row number, (status line is row 0).
$269 Cursor column position.
$26A Flag byte.

Bit	**Flag function when bit is set to 1**.
0	Cursor on.
1	Printout to screen enabled.
2	Unused.
3	Disable keyciick.
4	Previous printed character was ESC
5	Protect columns 0 and 1 of screen.
6	Double height characters.
7	Unused.

$26B Paper colour (+16).
$26C Ink colour.
$26D,$26E Start address of screen memory.
$26F Number of text lines available on screen (V1.0).
$270 Cursor on/off flag.
$271 Cursor invert flag.
$272,$273 Keyboard timer.
$274,$275 Cursor timer.
$276,$277 Spare counter — also used by WAIT (and printer in V1.0).
$278,$279 Address of second line on screen (V1.1).
$27A,$27B Address of first line on screen (V1.1).
$27C,$27D Number of characters normally used in screen scrolling, *26 x 40 - 1,040 or #410* (V1.1).
$27E Number of rows of text available (V1.1).
$27F-$28F Name of program to be loaded off cassette (V1.1).
$290-$292
$293-$2A3 Name of file just loaded off cassette (Vl.l).
$2A4-$2A8
$2A9,$2AA Start address of data for / from cassette (Vl.l).
$2AB,$2AC End address of data for/from cassette (Vl.l).
$2AD Auto indicator, 0 is off (Vl.l).
$2AE Program type.
$2AF Array type-copy of $28(Vl.l).
$2B0 Array type - copy of $29 (V1. 1).
$2B1 Bit 7 set to 1 if format error.
$2B2-$2BF
$2C0 Screen status, 0=GRAB, 2=TEXT and 3=HIRES.
$2C1,$2C2 Char set start address in HIRES mode (V1.1).
$2C3 Cursor movement in hires, 0 = Absolute, 1 =
$2C4-$2DE
$2DF Latest key from keyboard. Bit 7 set if valid.
$2E0 Non zero if error in Sound / Graphics routines.
$2E1,$2E2 First parameter location for the sound and graphics commands (*including INK and PAPER*). Lower byte of integer is at $02E1.

```
$2E3,$2E4     Second parameter location for the sound and graphics commands.
              Lower byte of integer is at $2E3.
$2E5,$2E6     Third parameter location for the sound and graphics commands.
              Lower byte of integer is at $2E5.
$2E7,$2E8     Fourth parameter location which is only used by MUSIC and PLAY.
              Lower byte of integer is at $2E7.
$2E9-$2F0
$2F1          Bit 7 set to 1 if printer enabled.
$2F2          Bit 7 is EDIT flag, set to 1 when on.
$2F3
$2F4          TRACE flag, set if bit 7 is set.
$2F5,$2F6     Indirect jump for '!' routine.
$2F7
$2F8          Temporary row indicator for PLOT.
$2F9,$2FA
$2FB-$2FD     Jump to '&' routine.
$2FE-$2FF
```