```
;*******************************************************************************
;*******************************************************************************
;
; The almost completely commented Vic 20 ROM disassembly. V1.01 Lee Davison 2005-2012.

; This is a bit correct assembly listing for the Vic 20 BASIC and kernal ROMs as one 16K
; ROM. You should be able to assemble the Vic ROMs from this with most 6502 assemblers,
; as no macros or 'special' features were used. This has been tested using Michal
; Kowalski's 6502 Simulator assemble function. See http://exifpro.com/utils.html for
; this program.

; Many references were used to complete this disassembly including, but not limited to,
; "Mapping the Vic 20", "Mapping the C64", "Vic 20 Programmers reference", "Vic 20 user
; guide", "The complete Commodore inner space anthology", "VIC Revealed" and various
; text files, pictures and other documents.


;*******************************************************************************
;*******************************************************************************
;
; first a whole load of equates

; These locations contain the JMP instruction target address of the USR command. They
; are initialised so that if you try to execute a USR call without changing them you
; wil receive an ILLEGAL QUANTITY error message.

LAB_00  = $00                   ; USR() JMP instruction
LAB_01  = $01                   ; USR() vector low byte
LAB_02  = $02                   ; USR() vector high byte

; This vector points to the address of the BASIC routine which converts a floating point
; number to an integer, however BASIC does not use this vector. It may be of assistance
; to the programmer who wishes to use data that is stored in floating point format. The
; parameter passed by the USR command is available only in that format for example.

LAB_03  = $03                   ; float to fixed vector low byte
LAB_04  = $04                   ; float to fixed vector high byte

; This vector points to the address of the BASIC routine which converts an integer to a
; floating point number, however BASIC does not use this vector. It may be used by the
; programmer who needs to make such a conversion for a machine language program that
; interacts with BASIC.  To return an integer value with the USR command for example.

LAB_05  = $05                   ; fixed to float vector low byte
LAB_06  = $06                   ; fixed to float vector high byte

; The cursor column position prior to the TAB or SPC is moved here from $D3, and is used
; to calculate where the cursor ends up after one of these functions is invoked.

; Note that the value contained here shows the position of the cursor on a logical line.
; Since one logical line can be up to four physical lines long, the value stored here
; can range from 0 to 87.

LAB_07  = $07                   ; search character
LAB_08  = $08                   ; scan quotes flag
LAB_09  = $09                   ; TAB column save

; The routine that converts the text in the input buffer into lines of executable program
; tokes, and the routines that link these program lines together, use this location as an
; index into the input buffer area. After the job of converting text to tokens is done,
```

```
; the value in this location is equal to the length of the tokenized line.

; The routines which build an array or locate an element in an array use this location to
; calculate the number of DIMensions called for and the amount of storage required for a
; newly created array, or the number of subscripts when referencing an array element.

LAB_0A  = $0A                   ; load/verify flag, 0 = load, 1 = verify
LAB_0B  = $0B                   ; temporary byte, line crunch/array access/logic operators

; This is used as a flag by the routines that build an array or reference an existing
; array. It is used to determine whether a variable is in an array, whether the array
; has already been DIMensioned, and whether a new array should assume the default size.

LAB_0C  = $0C                   ; DIM flag

; This flag is used to indicate whether data being operated upon is string or numeric. A
; value of $FF in this location indicates string data while a $00 indicates numeric data.

LAB_0D  = $0D                   ; data type flag, $FF = string, $00 = numeric

; If the above flag indicates numeric then a $80 in this location identifies the number
; as an integer, and a $00 indicates a floating point number.

LAB_0E  = $0E                   ; data type flag, $80 = integer, $00 = floating point

; The garbage collection routine uses this location as a flag to indicate that garbage
; collection has already been tried before adding a new string. If there is still not
; enough memory, an OUT OF MEMORY error message will result.

; LIST uses this byte as a flag to let it know when it has come to a character string in
; quotes. It will then print the string,rather than search it for BASIC keyword tokens.

; This location is also used during the process of converting a line of text in the BASIC
; input buffer into a linked program line of BASIC keyword tokens to flag a DATA line is
; being processed.

LAB_0F  = $0F                   ; garbage collected/open quote/DATA flag

; If an opening parenthesis is found, this flag is set to indicate that the variable in
; question is either an array variable or a user-defined function.

LAB_10  = $10                   ; subscript/FNx flag

; This location is used to determine whether the sign of the value returned by the
; functions SIN, COS, ATN or TAN is positive or negative.

; Also the comparison routines use this location to indicate the outcome of the compare.
; For A <=> B the value here will be $01 if A > B, $02 if A = B, and $04 if A < B. If
; more than one comparison operator was used to compare the two variables then the value
; here will be a combination of the above values.

LAB_11  = $11                   ; input mode flag, $00 = INPUT, $40 = GET, $98 = READ
LAB_12  = $12                   ; ATN sign/comparison evaluation flag

; When the default input or output device is used the value here will be a zero, and the
; format of prompting and output will be the standard screen output format. The location
; $B8 is used to decide what device actually to put input from or output to.

; The print CR/LF code at LAB_CAD7 suggests that b7 of this byte is an AutoLF flag bit
; but if it is used as such it would break lots of other parts of the code
```

```
LAB_13  = $13                   ; current I/O channel

; Used whenever a 16 bit integer is used e.g. the target line number for GOTO, LIST, ON,
; and GOSUB also the number of a BASIC line that is to be added or replaced. additionally
; PEEK, POKE, WAIT, and SYS use this location as a pointer to the address which is the
; subject of the command.

LAB_14  = $14                   ; temporary integer low byte
LAB_15  = $15                   ; temporary integer high byte

; This location points to the next available slot in the temporary string descriptor
; stack located at $19-$21.

LAB_16  = $16                   ; descriptor stack pointer, next free

; This contains information about temporary strings which hve not yet been assigned to
; a string variable.

LAB_17  = $17                   ; current descriptor stack item pointer low byte
LAB_18  = $18                   ; current descriptor stack item pointer high byte
LAB_19  = $19                   ; to $21, descriptor stack

; These locations are used by BASIC multiplication and division routines. They are also
; used by the routines which compute the size of the area required to store an array
; which is being created.

LAB_22  = $22                   ; misc temp byte
LAB_23  = $23                   ; misc temp byte
LAB_24  = $24                   ; misc temp byte
LAB_25  = $25                   ; misc temp byte

LAB_26  = $26                   ; temp mantissa 1
LAB_27  = $27                   ; temp mantissa 2
LAB_28  = $28                   ; temp mantissa 3
LAB_29  = $29                   ; temp mantissa 4

; Two byte pointer to where the BASIC program text is stored.

LAB_2B  = $2B                   ; start of memory low byte
LAB_2C  = $2C                   ; start of memory high byte

; Two byte pointer to the start of the BASIC variable storage area.

LAB_2D  = $2D                   ; start of variables low byte
LAB_2E  = $2E                   ; start of variables high byte

; Two byte pointer to the start of the BASIC array storage area.

LAB_2F  = $2F                   ; end of variables low byte
LAB_30  = $30                   ; end of variables high byte

; Two byte pointer to end of the start of free RAM.

LAB_31  = $31                   ; end of arrays low byte
LAB_32  = $32                   ; end of arrays high byte

; Two byte pointer to the bottom of the string text storage area.

LAB_33  = $33                   ; bottom of string space low byte
LAB_34  = $34                   ; bottom of string space high byte
```

```
; Used as a temporary pointer to the most current string added by the routines which
; build strings or move them in memory.

LAB_35  = $35                   ; string utility ptr low byte
LAB_36  = $36                   ; string utility ptr high byte

; Two byte pointer to the highest address used by BASIC +1.

LAB_37  = $37                   ; end of memory low byte
LAB_38  = $38                   ; end of memory high byte

; These locations contain the line number of the BASIC statement which is currently being
; executed. A value of $FF in location $3A means that BASIC is in immediate mode.

LAB_39  = $39                   ; current line number low byte
LAB_3A  = $3A                   ; current line number high byte

; When program execution ends or stops the last line number executed is stored here.

LAB_3B  = $3B                   ; break line number low byte
LAB_3C  = $3C                   ; break line number high byte

; These locations contain the address of the start of the text of the BASIC statement
; that is being executed.  The value of the pointer to the address of the BASIC text
; character currently being scanned is stored here each time a new BASIC statement begins
; execution.

LAB_3D  = $3D                   ; continue pointer low byte
LAB_3E  = $3E                   ; continue pointer high byte

; These locations hold the line number of the current DATA statement being READ. If an
; error concerning the DATA occurs this number will be moved to $39/$3A so that the error
; message will show the line that contains the DATA statement rather than in the line that
; contains the READ statement.

LAB_3F  = $3F                   ; current DATA line number low byte
LAB_40  = $40                   ; current DATA line number high byte

; These locations point to the address where the next DATA will be READ from. RESTORE
; sets this pointer back to the address indicated by the start of BASIC pointer.

LAB_41  = $41                   ; DATA pointer low byte
LAB_42  = $42                   ; DATA pointer high byte

; READ, INPUT and GET all use this as a pointer to the address of the source of incoming
; data, such as DATA statements, or the text input buffer.

LAB_43  = $43                   ; READ pointer low byte
LAB_44  = $44                   ; READ pointer high byte

LAB_45  = $45                   ; current variable name first byte
LAB_46  = $46                   ; current variable name second byte

; These locations point to the value of the current BASIC variable Specifically they
; point to the byte just after the two-character variable name.

LAB_47  = $47                   ; current variable address low byte
LAB_48  = $48                   ; current variable address high byte

; The address of the BASIC variable which is the subject of a FOR/NEXT loop is first
; stored here before being pushed onto the stack.
```

```
LAB_49  = $49                   ; FOR/NEXT variable pointer low byte
LAB_4A  = $4A                   ; FOR/NEXT variable pointer high byte

; The expression evaluation routine creates this to let it know whether the current
; comparison operation is a < $01, = $02 or > $04 comparison or combination.

LAB_4B  = $4B                   ; BASIC execute pointer temporary low byte/precedence flag
LAB_4C  = $4C                   ; BASIC execute pointer temporary high byte
LAB_4D  = $4D                   ; comparrison evaluation flag

; These locations are used as a pointer to the function that is created during function
; definition . During function execution it points to where the evaluation results should
; be saved.

LAB_4E  = $4E                   ; FAC temp store/function/variable/garbage pointer low byte
LAB_4F  = $4F                   ; FAC temp store/function/variable/garbage pointer high byte

; Temporary Pointer to the current string descriptor.

LAB_50  = $50                   ; FAC temp store/descriptor pointer low byte
LAB_51  = $51                   ; FAC temp store/descriptor pointer high byte

LAB_53  = $53                   ; garbage collection step size

; The first byte is the 6502 JMP instruction $4C, followed by the address of the required
; function taken from the table at $C052.

LAB_54  = $54                   ; JMP opcode for functions
LAB_55  = $55                   ; functions jump vector low byte
LAB_56  = $56                   ; functions jump vector high byte

LAB_57  = $57                   ; FAC temp store
LAB_58  = $58                   ; FAC temp store
LAB_59  = $59                   ; FAC temp store
LAB_5A  = $5A                   ; FAC temp store
LAB_5B  = $5B                   ; block end high byte
LAB_5C  = $5C                   ; FAC temp store
LAB_5D  = $5D                   ; FAC temp store
LAB_5E  = $5E                   ; FAC temp store
LAB_5F  = $5F                   ; FAC temp store
LAB_60  = $60                   ; block start high byte
LAB_61  = $61                   ; FAC1 exponent
LAB_62  = $62                   ; FAC1 mantissa 1
LAB_63  = $63                   ; FAC1 mantissa 2
LAB_64  = $64                   ; FAC1 mantissa 3
LAB_65  = $65                   ; FAC1 mantissa 4
LAB_66  = $66                   ; FAC1 sign
LAB_67  = $67                   ; constant count/-ve flag
LAB_68  = $68                   ; FAC1 overflow
LAB_69  = $69                   ; FAC2 exponent
LAB_6A  = $6A                   ; FAC2 mantissa 1
LAB_6B  = $6B                   ; FAC2 mantissa 2
LAB_6C  = $6C                   ; FAC2 mantissa 3
LAB_6D  = $6D                   ; FAC2 mantissa 4
LAB_6E  = $6E                   ; FAC2 sign
LAB_6F  = $6F                   ; FAC sign comparrison
LAB_70  = $70                   ; FAC1 rounding
LAB_71  = $71                   ; temp BASIC execute/array pointer low byte/index
LAB_72  = $72                   ; temp BASIC execute/array pointer high byte
```

```
LAB_0073        = $73                   ; increment and scan memory, BASIC byte get
LAB_0079        = $79                   ; scan memory, BASIC byte get
LAB_7A  = $7A                   ; BASIC execute pointer low byte
LAB_7B  = $7B                   ; BASIC execute pointer high byte
LAB_80  = $80                   ; numeric test entry

LAB_008B        = $8B                   ; RND() seed, five bytes

LAB_90  = $90                   ; serial status byte
                                        ;       function
                                        ; bit   casette         serial bus
                                        ; ---   --------                ----------
                                        ;  7    end of tape             device not present
                                        ;  6    end of file             EOI
                                        ;  5    checksum error
                                        ;  4    read error
                                        ;  3    long block
                                        ;  2    short block
                                        ;  1                            time out read
                                        ;  0                            time out write
LAB_91  = $91                   ; keyboard row, bx = 0 = key down
                                        ; bit   key
                                        ; ---   ------
                                        ;  7    [DOWN]
                                        ;  6    /
                                        ;  5    ,
                                        ;  4    N
                                        ;  3    V
                                        ;  2    X
                                        ;  1    [L SHIFT]
                                        ;  0    [STOP]
LAB_92  = $92                   ; timing constant for tape read
LAB_93  = $93                   ; load/verify flag, load = $00, verify = $01
LAB_94  = $94                   ; serial output: deferred character flag
                                        ; $00 = no character waiting, $xx = character waiting
LAB_95  = $95                   ; serial output: deferred character
                                        ; $FF = no character waiting, $xx = waiting character
LAB_96  = $96                   ; cassette block synchronization number
LAB_97  = $97                   ; register save

; The number of currently open I/O files is stored here. The maximum number that can be
; open at one time is ten. The number stored here is used as the index to the end of the
; tables that hold the file numbers, device numbers, and secondary addresses.

LAB_98  = $98                   ; open file count

; The default value of this location is 0.

LAB_99  = $99                   ; input device number

; The default value of this location is 3.

LAB_9A  = $9A                   ; output device number
                                        ;
                                        ; number        device
                                        ; ------        ------
                                        ;  0            keyboard
                                        ;  1            cassette
                                        ;  2            RS-232C
                                        ;  3            screen
                                        ;  4-31 serial bus
```

```
LAB_9B  = $9B                   ; tape character parity
LAB_9C  = $9C                   ; byte received flag
LAB_9D  = $9D                   ; message mode flag,
                                        ; $C0 = both control and kernal messages,
                                        ; $80 = control messages only,
                                        ; $40 = kernal messages only,
                                        ; $00 = neither control or kernal messages
LAB_9E  = $9E                   ; tape Pass 1 error log/character buffer
LAB_9F  = $9F                   ; tape Pass 2 error log corrected

; These three locations form a counter which is updated 60 times a second, and serves as
; a software clock which counts the number of jiffies that have elapsed since the computer
; was turned on. After 24 hours and one jiffy these locations are set back to $000000.

LAB_A0  = $A0                   ; jiffy clock high byte
LAB_A1  = $A1                   ; jiffy clock mid byte
LAB_A2  = $A2                   ; jiffy clock low byte

LAB_A3  = $A3                   ; EOI flag byte/tape bit count

; b0 of this location reflects the current phase of the tape output cycle.

LAB_A4  = $A4                   ; tape bit cycle phase
LAB_A5  = $A5                   ; cassette synchronization byte count/serial bus bit count
LAB_A6  = $A6                   ; tape buffer index
LAB_A7  = $A7                   ; receiver input bit temp storage
LAB_A8  = $A8                   ; receiver bit count in
LAB_A9  = $A9                   ; receiver start bit check flag, $90 = no start bit
                                        ; received, $00 = start bit received
LAB_AA  = $AA                   ; receiver byte buffer/assembly location
LAB_AB  = $AB                   ; receiver parity bit storage
LAB_AC  = $AC                   ; tape buffer start pointer low byte
                                        ; scroll screen ?? byte
LAB_AD  = $AD                   ; tape buffer start pointer high byte
                                        ; scroll screen ?? byte
LAB_AE  = $AE                   ; tape buffer end pointer low byte
                                        ; scroll screen ?? byte
LAB_AF  = $AF                   ; tape buffer end pointer high byte
                                        ; scroll screen ?? byte
LAB_B0  = $B0                   ; tape timing constant min byte
LAB_B1  = $B1                   ; tape timing constant max byte

; Thess two locations point to the address of the cassette buffer. This pointer must
; be greater than or equal to $0200 or an ILLEGAL DEVICE NUMBER error will be sent
; when tape I/O is tried. This pointer must also be less that $8000 or the routine
; will terminate early.

LAB_B2  = $B2                   ; tape buffer start pointer low byte
LAB_B3  = $B3                   ; tape buffer start pointer high byte

; RS232 routines use this to count the number of bits transmitted and for parity and
; stop bit manipulation. Tape load routines use this location to flag when they are
; ready to receive data bytes.

LAB_B4  = $B4                   ; transmitter bit count out

; This location is used by the RS232 routines to hold the next bit to be sent and by the
; tape routines to indicate what part of a block the read routine is currently reading.

LAB_B5  = $B5                   ; transmitter next bit to be sent
```

```
; RS232 routines use this area to disassemble each byte to be sent from the transmission
; buffer pointed to by $F9.

LAB_B6  = $B6                   ; transmitter byte buffer/disassembly location

; Disk filenames may be up to 16 characters in length while tape filenames be up to 187
; characters in length.

; If a tape name is longer than 16 characters the excess will be truncated by the
; SEARCHING and FOUND messages, but will still be present on the tape.

; A disk file is always referred to by a name. This location will always be greater than
; zero if the current file is a disk file.

; An RS232 OPEN command may specify a filename of up to four characters. These characters
; are copied to locations $293 to $296 and determine baud rate, word length, and parity,
; or they would do if the feature was fully implemented.

LAB_B7  = $B7                   ; file name length

LAB_B8  = $B8                   ; logical file
LAB_B9  = $B9                   ; secondary address
LAB_BA  = $BA                   ; current device number
                                        ; number        device
                                        ; ------        ------
                                        ;  0            keyboard
                                        ;  1            cassette
                                        ;  2            RS-232C
                                        ;  3            screen
                                        ;  4-31 serial bus
LAB_BB  = $BB                   ; file name pointer low byte
LAB_BC  = $BC                   ; file name pointer high byte
LAB_BD  = $BD                   ; tape write byte/RS232 parity byte

; Used by the tape routines to count the number of copies of a data block remaining to
; be read or written.

LAB_BE  = $BE                   ; tape copies count
LAB_BF  = $BF                   ; parity count ??
LAB_C0  = $C0                   ; tape motor interlock
LAB_C1  = $C1                   ; I/O start addresses low byte
LAB_C2  = $C2                   ; I/O start addresses high byte
LAB_C3  = $C3                   ; kernal setup pointer low byte
LAB_C4  = $C4                   ; kernal setup pointer high byte
LAB_C5  = $C5                   ; current key pressed
                                        ;
                                        ;  # key         # key   # key   # key
                                        ; -- ---        -- ---  -- ---  -- ---
                                        ; 00 1  10 none 20 [SPACE]      30 Q
                                        ; 01 3  11 A            21 Z            31 E
                                        ; 02 5  12 D            22 C            32 T
                                        ; 03 7  13 G            23 B            33 U
                                        ; 04 9  14 J            24 M            34 O
                                        ; 05 +  15 L            25 .            35 @
                                        ; 06 [UKP]      16 ;            26 none 36 ^
                                        ; 07 [DEL]      17 [CSR R]      27 [F1] 37 [F5]
                                        ; 08 [<-]       18 [STOP]       28 none 38 2
                                        ; 09 W  19 none 29 S            39 4
                                        ; 0A R  1A X            2A F            3A 6
                                        ; 0B Y  1B V            2B H            3B 8
```

```
                                        ; 0C I  1C N            2C K            3C 0
                                        ; 0D P  1D ,            2D :            3D -
                                        ; 0E *  1E /            2E =            3E [HOME]
                                        ; 0F [RET]      1F [CSR D]      2F [F3] 3F [F7]

LAB_C6  = $C6                   ; keyboard buffer length/index

; When the [CTRL][RVS-ON] characters are printed this flag is set to $12, and the print
; routines will add $80 to the screen code of each character which is printed, so that
; the caracter will appear on the screen with its colours reversed.

; Note that the contents of this location are cleared not only upon entry of a
; [CTRL][RVS-OFF] character but also at every carriage return.

LAB_C7  = $C7                   ; reverse flag $12 = reverse, $00 = normal

; This pointer indicates the column number of the last nonblank character on the logical
; line that is to be input. Since a logical line can be up to 88 characters long this
; number can range from 0-87.

LAB_C8  = $C8                   ; input [EOL] pointer

; These locations keep track of the logical line that the cursor is on and its column
; position on that logical line.

; Each logical line may contain up to four 22 column physical lines. So there may be as
; many as 23 logical lines, or as few as 6 at any one time. Therefore, the logical line
; number might be anywhere from 1-23. Depending on the length of the logical line, the
; cursor column may be from 1-22, 1-44, 1-66 or 1-88.

; For a more on logical lines, see the description of the screen line link table, $D9.

LAB_C9  = $C9                   ; input cursor row
LAB_CA  = $CA                   ; input cursor column

; The keyscan interrupt routine uses this location to indicate which key is currently
; being pressed. The value here is then used as an index into the appropriate keyboard
; table to determine which character to print when a key is struck.

; The correspondence between the key pressed and the number stored here is as follows:

; $00   1               $10     not used        $20     [SPACE] $30     Q               $40
[NO KEY]
; $01   3               $11     A               $21     Z               $31     E
$xx     invalid
; $02   5               $12     D               $22     C               $32     T
; $03   7               $13     G               $23     B               $33     U
; $04   9               $14     J               $24     M               $34     O
; $05   +               $15     L               $25     .               $35     @
; $06   [POUND] $16     ;               $26     not used        $36     [U ARROW]
; $07   [DEL]           $17     [RIGHT] $27     [F1]            $37     [F5]
; $08   [L ARROW]       $18     [STOP]  $28     not used        $38     2
; $09   W               $19     not used        $29     S               $39     4
; $0A   R               $1A     X               $2A     F               $3A     6
; $0B   Y               $1B     V               $2B     H               $3B     8
; $0C   I               $1C     N               $2C     K               $3C     0
; $0D   P               $1D     ,               $2D     :               $3D     -
; $0E   *               $1E     /               $2E     =               $3E     [HOME]
; $0F   [RETURN]        $1F     [DOWN]  $2F     [F3]            $3F     [F7]

LAB_CB  = $CB                   ; which key
```

```
; When this flag is set to a nonzero value, it indicates to the routine that normally
; flashes the cursor not to do so. The cursor blink is turned off when there are
; characters in the keyboard buffer, or when the program is running.

LAB_CC  = $CC                   ; cursor enable, $00 = flash cursor

; The routine that blinks the cursor uses this location to tell when it's time for a
; blink. The number 20 is put here and decremented every jiffy until it reaches zero.
; Then the cursor state is changed, the number 20 is put back here, and the cycle starts
; all over again.

LAB_CD  = $CD                   ; cursor timing countdown

; The cursor is formed by printing the inverse of the character that occupies the cursor
; position. If that characters is the letter A, for example, the flashing cursor merely
; alternates between printing an A and a reverse-A. This location keeps track of the
; normal screen code of the character that is located at the cursor position, so that it
; may be restored when the cursor moves on.

LAB_CE  = $CE                   ; character under cursor

; This location keeps track of whether, during the current cursor blink, the character
; under the cursor was reversed, or was restored to normal. This location will contain
; $00 if the character is reversed, and $01 if the character is not reversed.

LAB_CF  = $CF                   ; cursor blink phase

LAB_D0  = $D0                   ; input from keyboard or screen, $xx = input is available
                                        ; from the screen, $00 = input should be obtained from
the
                                        ; keyboard

; These locations point to the address in screen RAM of the first column of the logical
; line upon which the cursor is currently positioned.

LAB_D1  = $D1                   ; current screen line pointer low byte
LAB_D2  = $D2                   ; current screen line pointer high byte

; This holds the cursor column position within the logical line pointed to by LAB_D1.
; Since a logical line can comprise up to four physical lines, this value may be from
; $00 to $57.

LAB_D3  = $D3                   ; cursor column

; A nonzero value in this location indicates that the editor is in quote mode. Quote
; mode is toggled every time that you type in a quotation mark on a given line, the
; first quote mark turns it on, the second turns it off, the third turns it on, etc.

; If the editor is in this mode when a cursor control character or other nonprinting
; character is entered, a printed equivalent will appear on the screen instead of the
; cursor movement or other control operation taking place. Instead, that action is
; deferred until the string is sent to the string by a PRINT statement, at which time
; the cursor movement or other control operation will take place.

; The exception to this rule is the DELETE key, which will function normally within
; quote mode. The only way to print a character which is equivalent to the DELETE key
; is by entering insert mode. Quote mode may be exited by printing a closing quote or
; by hitting the RETURN or SHIFT-RETURN keys.

LAB_D4  = $D4                   ; cursor quote flag
```

```
; The line editor uses this location when the end of a line has been reached to determine
; whether another physical line can be added to the current logical line or if a new
; logical line must be started.

LAB_D5  = $D5                   ; current screen line length

; This location contains the current physical screen line position of the cursor, 0 to 22.

LAB_D6  = $D6                   ; cursor row

; The ASCII value of the last character printed to the screen is held here temporarily.

LAB_D7  = $D7                   ; checksum byte/temporary last character

; When the INST key is pressed, the screen editor shifts the line to the right, allocates
; another physical line to the logical line if necessary (and possible), updates the
; screen line length in $D5, and adjusts the screen line link table at $D9. This location
; is used to keep track of the number of spaces that has been opened up in this way.

; Until the spaces that have been opened up are filled, the editor acts as if in quote
; mode. See location $D4, the quote mode flag. This means that cursor control characters
; that are normally nonprinting will leave a printed equivalent on the screen when
; entered, instead of having their normal effect on cursor movement, etc. The only
; difference between insert and quote mode is that the DELETE key will leave a printed
; equivalent in insert mode, while the INSERT key will insert spaces as normal.

LAB_D8  = $D8                   ; insert count

; This table contains 23 entries, one for each row of the screen display. Each entry has
; two functions. Bits 0-3 indicate on which of the four pages of screen memory the first
; byte of memory for that row is located. This is used in calculating the pointer to the
; starting address of a screen line at $D1.
;
; The high byte is calculated by adding the value of the starting page of screen memory
; held in $288 to the displacement page held here.
;
; The other function of this table is to establish the makeup of logical lines on the
; screen. While each screen line is only 22 characters long, BASIC allows the entry of
; program lines that contain up to 88 characters. Therefore, some method must be used
; to determine which physical lines are linked into a longer logical line, so that this
; longer logical line may be edited as a unit.
;
; The high bit of each byte here is used as a flag by the screen editor. That bit is set
; when a line is the first or only physical line in a logical line. The high bit is reset
; to 0 only when a line is an extension to this logical line.

LAB_D9  = $D9                   ; to LAB_D9 + $18 inclusive, screen line link table

; This pointer is synchronized with the pointer to the address of the first byte of
; screen RAM for the current line kept in location $D1. It holds the address of the
; first byte of colour RAM for the corresponding screen line.

LAB_F2  = $F2                   ; screen row marker
LAB_F3  = $F3                   ; colour RAM pointer low byte
LAB_F4  = $F4                   ; colour RAM pointer high byte

; This pointer points to the address of the keyboard matrix lookup table currently being
; used. Although there are only 64 keys on the keyboard matrix, each key can be used to
; print up to four different characters, depending on whether it is struck by itself or
; in combination with the SHIFT, CTRL, or C= keys.
```

```
; These tables hold the ASCII value of each of the 64 keys for one of these possible
; combinations of keypresses. When it comes time to print the character, the table that
; is used determines which character is printed.

; The addresses of the tables are:

;       LAB_EC5E                        ; unshifted
;       LAB_EC9F                        ; shifted
;       LAB_ECE0                        ; commodore
;       LAB_EDA3                        ; control

LAB_F5  = $F5                   ; keyboard pointer low byte
LAB_F6  = $F6                   ; keyboard pointer high byte

; When device the RS232 channel is opened two buffers of 256 bytes each are created at
; the top of memory. These locations point to the address of the one which is used to
; store characters as they are received.

LAB_F7  = $F7                   ; RS232 Rx pointer low byte
LAB_F8  = $F8                   ; RS232 Rx pointer high byte

; These locations point to the address of the 256 byte output buffer that is used for
; transmitting data to RS232 devices.

LAB_F9  = $F9                   ; RS232 Tx pointer low byte
LAB_FA  = $FA                   ; RS232 Tx pointer high byte

LAB_00FF        = $FF                   ; FAC1 to string output base

LAB_0100        = $0100         ; bottom of the stack page

LAB_01FC        = $01FC         ; chain link pointer high byte
LAB_01FD        = $01FD         ; chain link pointer low byte

LAB_01FE        = $01FE         ; line number low byte before crunched line
LAB_01FF        = $01FF         ; line number high byte before crunched line

LAB_0200        = $0200         ; input buffer. for some routines the byte before the input
                                        ; buffer needs to be set to a specific value for the
routine
                                        ; to work correctly
LAB_0201        = $0201         ; input buffer + 1

LAB_0259        = $0259         ; .. to LAB_0262 logical file table
LAB_0263        = $0263         ; .. to LAB_026C device number table
LAB_026D        = $026D         ; .. to LAB_0276 secondary address table
LAB_0277        = $0277         ; .. to LAB_0280 keyboard buffer
LAB_0281        = $0281         ; OS start of memory low byte
LAB_0282        = $0282         ; OS start of memory high byte
LAB_0283        = $0283         ; OS top of memory low byte
LAB_0284        = $0284         ; OS top of memory high byte
LAB_0285        = $0285         ; serial bus timeout flag
LAB_0286        = $0286         ; current colour code
LAB_0287        = $0287         ; colour under cursor
LAB_0288        = $0288         ; screen memory page
LAB_0289        = $0289         ; maximum keyboard buffer size
LAB_028A        = $028A         ; key repeat. $80 = repeat all, $40 = repeat none,
                                        ; $00 = repeat cursor movement keys, insert/delete
                                        ; key and the space bar
LAB_028B        = $028B         ; repeat speed counter
```

```
LAB_028C        = $028C         ; repeat delay counter

; This flag signals which of the SHIFT, CTRL, or C= keys are currently being pressed.

; A value of $01 signifies that one of the SHIFT keys is being pressed, a $02 shows that
; the C= key is down, and $04 means that the CTRL key is being pressed. If more than one
; key is held down, these values will be added e.g $03 indicates that SHIFT and C= are
; both held down.

; Pressing the SHIFT and C= keys at the same time will toggle the character set that is
; presently being used between the uppercase/graphics set, and the lowercase/uppercase
; set.

; While this changes the appearance of all of the characters on the screen at once it
; has nothing whatever to do with the keyboard shift tables and should not be confused
; with the printing of SHIFTed characters, which affects only one character at a time.

LAB_028D        = $028D         ; keyboard shift/control flag
                                        ; bit   key(s) 1 = down
                                        ; ---   ---------------
                                        ; 7-3   unused
                                        ;  2    CTRL
                                        ;  1    C=
                                        ;  0    SHIFT

; This location, in combination with the one above, is used to debounce the special
; SHIFT keys. This will keep the SHIFT/C= combination from changing character sets
; back and forth during a single pressing of both keys.

LAB_028E        = $028E         ; SHIFT/CTRL/C= keypress last pattern

; This location points to the address of the Operating System routine which actually
; determines which keyboard matrix lookup table will be used.

; The routine looks at the value of the SHIFT flag at $28D, and based on what value
; it finds there, stores the address of the correct table to use at location $F5.

LAB_028F        = $028F         ; keyboard decode logic pointer low byte
LAB_0290        = $0290         ; keyboard decode logic pointer high byte

; This flag is used to enable or disable the feature which lets you switch between the
; uppercase/graphics and upper/lowercase character sets by pressing the SHIFT and
; Commodore logo keys simultaneously.

LAB_0291        = $0291         ; shift mode switch, $00 = enabled, $80 = locked

; This location is used to determine whether moving the cursor past the ??xx  column of
; a logical line will cause another physical line to be added to the logical line.

; A value of 0 enables the screen to scroll the following lines down in order to add
; that line; any nonzero value will disable the scroll.

; This flag is set to disable the scroll temporarily when there are characters waiting
; in the keyboard buffer, these may include cursor movement characters that would
; eliminate the need for a scroll.

LAB_0292        = $0292         ; screen scrolling flag, $00 = enabled

LAB_0293        = $0293         ; pseudo 6551 control register. the first character of
                                        ; the OPEN RS232 filename will be stored here
                                        ; bit   function
```

```
                                        ; ---   --------
                                        ;  7    2 stop bits/1 stop bit
                                        ; 65    word length
                                        ; ---   -----------
                                        ; 00    8 bits
                                        ; 01    7 bits
                                        ; 10    6 bits
                                        ; 11    5 bits
                                        ;  4    unused
                                        ; 3210  baud rate
                                        ; ----  ---------
                                        ; 0000  user rate *
                                        ; 0001     50
                                        ; 0010     75
                                        ; 0011    110
                                        ; 0100    134.5
                                        ; 0101    150
                                        ; 0110    300
                                        ; 0111    600
                                        ; 1000   1200
                                        ; 1001   1800
                                        ; 1010   2400
                                        ; 1011   3600
                                        ; 1100   4800 *
                                        ; 1101   7200 *
                                        ; 1110   9600 *
                                        ; 1111  19200 * * = not implemented
LAB_0294        = $0294         ; pseudo 6551 command register. the second character of
                                        ; the OPEN RS232 filename will be stored here
                                        ; bit   function
                                        ; ---   --------
                                        ; 765   parity
                                        ; ---   ------
                                        ; xx0   disabled
                                        ; 001   odd
                                        ; 011   even
                                        ; 101   mark
                                        ; 111   space
                                        ;  4    duplex half/full
                                        ;  3    unused
                                        ;  2    unused
                                        ;  1    unused
                                        ;  0    handshake - X line/3 line
;LAB_0295       = $0295         ; Nonstandard Bit Timing low byte. the third character
                                        ; of the OPEN RS232 filename will be stored here
;LAB_0296       = $0296         ; Nonstandard Bit Timing high byte. the fourth character
                                        ; of the OPEN RS232 filename will be stored here
LAB_0297        = $0297         ; RS-232 status register
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    break
                                        ;  6    no DSR detected
                                        ;  5    unused
                                        ;  4    no CTS detected
                                        ;  3    unused
                                        ;  2    Rx buffer overrun
                                        ;  1    framing error
                                        ;  0    parity error
LAB_0298        = $0298         ; number of bits to be sent/received
LAB_0299        = $0299         ; time of one bit cell low byte
LAB_029A        = $029A         ; time of one bit cell high byte
```

```
LAB_029B        = $029B         ; index to Rx buffer end
LAB_029C        = $029C         ; index to Rx buffer start
LAB_029D        = $029D         ; index to Tx buffer start
LAB_029E        = $029E         ; index to Tx buffer end
LAB_029F        = $029F         ; saved IRQ low byte
LAB_02A0        = $02A0         ; saved IRQ high byte

; $02A1 to $02FF - unused

LAB_0300        = $0300         ; vector to the print BASIC error message routine
LAB_0302        = $0302         ; Vector to the main BASIC program Loop
LAB_0304        = $0304         ; Vector to the the ASCII text to keywords routine
LAB_0306        = $0306         ; Vector to the list BASIC program as ASCII routine
LAB_0308        = $0308         ; Vector to the execute next BASIC command routine
LAB_030A        = $030A         ; Vector to the get value from BASIC line routine

; Before every SYS command each of the registers is loaded with the value found in the
; corresponding storage address. Upon returning to BASIC with an RTS instruction, the new
; value of each register is stored in the appropriate storage address.

; This feature allows you to place the necessary values into the registers from BASIC
; before you SYS to a Kernal or BASIC ML routine. It also enables you to examine the
; resulting effect of the routine on the registers, and to preserve the condition of the
; registers on exit for subsequent SYS calls.

LAB_030C        = $030C         ; A for SYS command
LAB_030D        = $030D         ; X for SYS command
LAB_030E        = $030E         ; Y for SYS command
LAB_030F        = $030F         ; P for SYS command

LAB_0314        = $0314         ; IRQ vector low byte
LAB_0315        = $0315         ; IRQ vector high byte
LAB_0316        = $0316         ; BRK vector
LAB_0318        = $0318         ; NMI vector

LAB_031A        = $031A         ; kernal vector - open a logical file
LAB_031C        = $031C         ; kernal vector - close a specified logical file
LAB_031E        = $031E         ; kernal vector - open channel for input
LAB_0320        = $0320         ; kernal vector - open channel for output
LAB_0322        = $0322         ; kernal vector - close input and output channels
LAB_0324        = $0324         ; kernal vector - input character from channel
LAB_0326        = $0326         ; kernal vector - output character to channel
LAB_0328        = $0328         ; kernal vector - scan stop key
LAB_032A        = $032A         ; kernal vector - get character from keyboard queue
LAB_032C        = $032C         ; kernal vector - close all channels and files

LAB_0330        = $0330         ; kernal vector - load
LAB_0332        = $0332         ; kernal vector - save

LAB_033C        = $033C         ; to $03FB - cassette buffer


;******************************************************************************;
;******************************************************************************;
;
; hardware equates

LAB_9000        = $9000         ; Vic chip base address
LAB_9002        = $9002         ; video address and colums
                                        ; bit   function
                                        ; ---   --------
```

```
                                        ;  7    video address va9
                                        ; 6-0   number of columns
LAB_9005        = $9005         ; video memory addresses ($1E00)
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    must be 1
                                        ; 6-4   video memory address va12-va10
                                        ; 3-0   character memory start address

                                        ; 0000 ROM      $8000   set 1
                                        ; 0001  "       $8400
                                        ; 0010  "       $8800 set 2
                                        ; 0011  "       $8C00
                                        ; 1100 RAM      $1000
                                        ; 1101  "       $1400
                                        ; 1110  "       $1800
                                        ; 1111  "       $1C00

LAB_9110        = $9110         ; VIA 1 DRB
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    DSR in
                                        ;  6    CTS in
                                        ;  5
                                        ;  4    DCD in
                                        ;  3    RI in
                                        ;  2    DTR out
                                        ;  1    RTS out
                                        ;  0    data in
LAB_9111        = $9111         ; VIA 1 DRA
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    serial ATN out
                                        ;  6    cassette switch
                                        ;  5    light pen
                                        ;  4    joy 2
                                        ;  3    joy 1
                                        ;  2    joy 0
                                        ;  1    serial DATA in
                                        ;  0    serial CLK in
LAB_9112        = $9112         ; VIA 1 DDRB
LAB_9113        = $9113         ; VIA 1 DDRA
LAB_9114        = $9114         ; VIA 1 T1C_l
LAB_9115        = $9115         ; VIA 1 T1C_h
LAB_9118        = $9118         ; VIA 1 T2C_l
LAB_9119        = $9119         ; VIA 1 T2C_h
LAB_911B        = $911B         ; VIA 1 ACR
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    T1 PB7 enabled/disabled
                                        ;  6    T1 free run/one shot
                                        ;  5    T2 clock PB6/□2
                                        ; 432   function
                                        ; ---   --------
                                        ; 000   shift register disabled
                                        ; 001   shift in, rate controlled by T2
                                        ; 010   shift in, rate controlled by □2
                                        ; 011   shift in, rate controlled by externak clock
                                        ; 100   shift out, rate controlled by T2, free run
mode
                                        ; 101   shift out, rate controlled by T2
```

```
                                        ; 110   shift out, rate controlled by ø2
                                        ; 111   shift out, rate controlled by externak clock
                                        ;  1    PB latch enabled/disabled
                                        ;  0    PA latch enabled/disabled
LAB_911C        = $911C         ; VIA 1 PCR
                                        ; bit   function
                                        ; ---   --------
                                        ; 765   CB2 control
                                        ; ---   -----------
                                        ; 000   Interrupt Input Mode
                                        ; 001   Independent Interrupt Input Mode
                                        ; 010   Input Mode
                                        ; 011   Independent Input Mode
                                        ; 100   Handshake Output Mode
                                        ; 101   Pulse Output Mode
                                        ; 110   Manual Output Mode, CB2 low
                                        ; 111   Manual Output Mode, CB2 high
                                        ;  4    CB1 edge positive/negative
                                        ; 321   CA2 control
                                        ; ---   -----------
                                        ; 000   Interrupt Input Mode
                                        ; 001   Independent Interrupt Input Mode
                                        ; 010   Input Mode
                                        ; 011   Independent Input Mode
                                        ; 100   Handshake Output Mode
                                        ; 101   Pulse Output Mode
                                        ; 110   Manual Output Mode, CA2 low
                                        ; 111   Manual Output Mode, CA2 high
                                        ;  0    CA1 edge positive/negative

; the status bit is a not normal flag. it goes high if both an interrupt flag in the IFR
; and the corresponding enable bit in the IER are set. it can be cleared only by clearing
; all the active flags in the IFR or disabling all active interrupts in the IER

LAB_911D        = $911D         ; VIA 1 IFR
                                        ; bit   function                cleared by
                                        ; ---   --------                ----------
                                        ;  7    interrupt status        clearing all enabled
interrupts
                                        ;  6    T1 interrupt    read T1C_l, write T1C_h
                                        ;  5    T2 interrupt    read T2C_l, write T2C_h
                                        ;  4    CB1 transition  read or write port B
                                        ;  3    CB2 transition  read or write port B
                                        ;  2    8 shifts done   read or write the shift
register
                                        ;  1    CA1 transition  read or write port A
                                        ;  0    CA2 transition  read or write port A

; If enable/disable bit is a zero during a write to this register, each 1 in bits 0-6
; clears the corresponding bit in the IER. if this bit is a one during a write to this
; register, each 1 in bits 0-6 will set the corresponding IER bit

LAB_911E        = $911E         ; VIA 1 IER
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    enable/disable
                                        ;  6    T1 interrupt
                                        ;  5    T2 interrupt
                                        ;  4    CB1 transition
                                        ;  3    CB2 transition
                                        ;  2    8 shifts done
```

```
                                        ;  1    CA1 transition
                                        ;  0    CA2 transition

LAB_911F        = $911F         ; VIA 1 DRA, no handshake
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    ATN out
                                        ;  6    cassette switch
                                        ;  5    joystick fire, light pen
                                        ;  4    joystick left
                                        ;  3    joystick down
                                        ;  2    joystick up
                                        ;  1    serial dat in
                                        ;  0    serial clk in

LAB_9120        = $9120         ; VIA 2 DRB, keyboard column drive
LAB_9121        = $9121         ; VIA 2 DRA, keyboard row port
                                        ; Vic 20 keyboard matrix layout
                                        ;       c7      c6      c5      c4      c3      c2
c1      c0
                                        ;   +-----------------------------------------------
                                        ; r7|   [F7]    [F5]    [F3]    [F1]    [DN]    [RGT]
[RET]   [DEL]
                                        ; r6|   [Home][UP]  =   [RSH]   /       ;       *
                                        ; r5|   -       @       :       .       ,       L
P       +
                                        ; r4|   0       O       K       M       N       J
I       9
                                        ; r3|   8       U       H       B       V       G
Y       7
                                        ; r2|   6       T       F       C       X       D
R       5
                                        ; r1|   4       E       S       Z       [LSH]   A
W       3
                                        ; r0|   2       Q       [CBM]   [SP]    [RUN]   [CTL]
[LFT]   1

LAB_9122        = $9122         ; VIA 2 DDRB
LAB_9123        = $9123         ; VIA 2 DDRA
LAB_9124        = $9124         ; VIA 2 T1C_l
LAB_9125        = $9125         ; VIA 2 T1C_h
LAB_9128        = $9128         ; VIA 2 T2C_l
LAB_9129        = $9129         ; VIA 2 T2C_h
LAB_912B        = $912B         ; VIA 2 ACR
LAB_912C        = $912C         ; VIA 2 PCR

; the status bit is a not normal flag. it goes high if both an interrupt flag in the IFR
; and the corresponding enable bit in the IER are set. it can be cleared only by clearing
; all the active flags in the IFR or disabling all active interrupts in the IER

LAB_912D        = $912D         ; VIA 1 IFR
                                        ; bit   function                cleared by
                                        ; ---   --------                ----------
                                        ;  7    interrupt status        clearing all enabled
interrupts
                                        ;  6    T1 interrupt    read T1C_l, write T1C_h
                                        ;  5    T2 interrupt    read T2C_l, write T2C_h
                                        ;  4    CB1 transition  read or write port B
                                        ;  3    CB2 transition  read or write port B
                                        ;  2    8 shifts done   read or write the shift
```

```
register
                                        ;  1    CA1 transition  read or write port A
                                        ;  0    CA2 transition  read or write port A

; If enable/disable bit is a zero during a write to this register, each 1 in bits 0-6
; clears the corresponding bit in the IER. if this bit is a one during a write to this
; register, each 1 in bits 0-6 will set the corresponding IER bit

LAB_912E        = $912E         ; VIA 1 IER
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    enable/disable
                                        ;  6    T1 interrupt
                                        ;  5    T2 interrupt
                                        ;  4    CB1 transition
                                        ;  3    CB2 transition
                                        ;  2    8 shifts done
                                        ;  1    CA1 transition
                                        ;  0    CA2 transition

LAB_912F        = $912F         ; VIA 2 DRA, keyboard row, no handshake

LAB_A000        = $A000         ; autostart ROM initial entry vector
LAB_A002        = $A002         ; autostart ROM break entry
LAB_A004        = $A004         ; autostart ROM identifier string start


;**************************************************************************;
;
; BASIC keyword token values. tokens not used in the source are included for
; completeness but commented out

; command tokens

;TK_END = $80                   ; END token
TK_FOR  = $81                   ; FOR token
;TK_NEXT        = $82                   ; NEXT token
TK_DATA = $83                   ; DATA token
;TK_INFL        = $84                   ; INPUT# token
;TK_INPUT       = $85                   ; INPUT token
;TK_DIM = $86                   ; DIM token
;TK_READ        = $87                   ; READ token

;TK_LET = $88                   ; LET token
TK_GOTO = $89                   ; GOTO token
;TK_RUN = $8A                   ; RUN token
;TK_IF  = $8B                   ; IF token
;TK_RESTORE     = $8C                   ; RESTORE token
TK_GOSUB        = $8D                   ; GOSUB token
;TK_RETURN      = $8E                   ; RETURN token
TK_REM  = $8F                   ; REM token

;TK_STOP        = $90                   ; STOP token
;TK_ON  = $91                   ; ON token
;TK_WAIT        = $92                   ; WAIT token
;TK_LOAD        = $93                   ; LOAD token
;TK_SAVE        = $94                   ; SAVE token
;TK_VERIFY      = $95                   ; VERIFY token
;TK_DEF = $96                   ; DEF token
;TK_POKE        = $97                   ; POKE token
```

```
;TK_PRINFL      = $98                   ; PRINT# token
TK_PRINT        = $99                   ; PRINT token
;TK_CONT        = $9A                   ; CONT token
;TK_LIST        = $9B                   ; LIST token
;TK_CLR = $9C                   ; CLR token
;TK_CMD = $9D                   ; CMD token
;TK_SYS = $9E                   ; SYS token
;TK_OPEN        = $9F                   ; OPEN token

;TK_CLOSE       = $A0                   ; CLOSE token
;TK_GET = $A1                   ; GET token
;TK_NEW = $A2                   ; NEW token

; secondary keyword tokens

TK_TAB  = $A3                   ; TAB( token
TK_TO           = $A4                   ; TO token
TK_FN           = $A5                   ; FN token
TK_SPC  = $A6                   ; SPC( token
TK_THEN = $A7                   ; THEN token

TK_NOT  = $A8                   ; NOT token
TK_STEP = $A9                   ; STEP token

; operator tokens

TK_PLUS = $AA                   ; + token
TK_MINUS        = $AB                   ; - token
;TK_MUL = $AC                   ; * token
;TK_DIV = $AD                   ; / token
;TK_POWER       = $AE                   ; ^ token
;TK_AND = $AF                   ; AND token

;TK_OR  = $B0                   ; OR token
TK_GT           = $B1                   ; > token
TK_EQUAL        = $B2                   ; = token
;TK_LT  = $B3                   ; < token

; function tokens

TK_SGN  = $B4                   ; SGN token
;TK_INT = $B5                   ; INT token
;TK_ABS = $B6                   ; ABS token
;TK_USR = $B7                   ; USR token

;TK_FRE = $B8                   ; FRE token
;TK_POS = $B9                   ; POS token
;TK_SQR = $BA                   ; SQR token
;TK_RND = $BB                   ; RND token
;TK_LOG = $BC                   ; LOG token
;TK_EXP = $BD                   ; EXP token
;TK_COS = $BE                   ; COS token
;TK_SIN = $BF                   ; SIN token

;TK_TAN = $C0                   ; TAN token
;TK_ATN = $C1                   ; ATN token
;TK_PEEK        = $C2                   ; PEEK token
;TK_LEN = $C3                   ; LEN token
;TK_STRS        = $C4                   ; STR$ token
;TK_VAL = $C5                   ; VAL token
;TK_ASC = $C6                   ; ASC token
```

```
;TK_CHRS        = $C7                   ; CHR$ token

;TK_LEFTS       = $C8                   ; LEFT$ token
;TK_RIGHTS      = $C9                   ; RIGHT$ token
;TK_MIDS        = $CA                   ; MID$ token
TK_GO           = $CB                   ; GO token

TK_PI           = $FF                   ; PI token


;**************************************************************************************;
;**************************************************************************************;
;
; ROM start

        .ORG    $C000

LAB_C000
        .word   LAB_E378                ; BASIC cold start entry point

LAB_C002
        .word   LAB_E467                ; BASIC warm start entry point

;LAB_C004
        .byte   "CBMBASIC"              ; ROM name, unreferenced


;**************************************************************************************;
;
; action addresses for primary commands. these are called by pushing the address
; onto the stack and doing an RTS so the actual address -1 needs to be pushed

LAB_C00C
        .word   LAB_C831-1              ; perform END
        .word   LAB_C742-1              ; perform FOR
        .word   LAB_CD1E-1              ; perform NEXT
        .word   LAB_C8F8-1              ; perform DATA
        .word   LAB_CBA5-1              ; perform INPUT#
        .word   LAB_CBBF-1              ; perform INPUT
        .word   LAB_D081-1              ; perform DIM
        .word   LAB_CC06-1              ; perform READ

        .word   LAB_C9A5-1              ; perform LET
        .word   LAB_C8A0-1              ; perform GOTO
        .word   LAB_C871-1              ; perform RUN
        .word   LAB_C928-1              ; perform IF
        .word   LAB_C81D-1              ; perform RESTORE
        .word   LAB_C883-1              ; perform GOSUB
        .word   LAB_C8D2-1              ; perform RETURN
        .word   LAB_C93B-1              ; perform REM

        .word   LAB_C82F-1              ; perform STOP
        .word   LAB_C94B-1              ; perform ON
        .word   LAB_D82D-1              ; perform WAIT
        .word   LAB_E165-1              ; perform LOAD
        .word   LAB_E153-1              ; perform SAVE
        .word   LAB_E162-1              ; perform VERIFY
        .word   LAB_D3B3-1              ; perform DEF
        .word   LAB_D824-1              ; perform POKE

        .word   LAB_CA80-1              ; perform PRINT#
```

```
        .word   LAB_CAA0-1              ; perform PRINT
        .word   LAB_C857-1              ; perform CONT
        .word   LAB_C69C-1              ; perform LIST
        .word   LAB_C65E-1              ; perform CLR
        .word   LAB_CA86-1              ; perform CMD
        .word   LAB_E127-1              ; perform SYS
        .word   LAB_E1BB-1              ; perform OPEN

        .word   LAB_E1C4-1              ; perform CLOSE
        .word   LAB_CB7B-1              ; perform GET
        .word   LAB_C642-1              ; perform NEW


;******************************************************************************;
;
; action addresses for functions

LAB_C052
        .word   LAB_DC39                ; perform SGN()
        .word   LAB_DCCC                ; perform INT()
        .word   LAB_DC58                ; perform ABS()
        .word   LAB_00          ; perform USR()

        .word   LAB_D37D                ; perform FRE()
        .word   LAB_D39E                ; perform POS()
        .word   LAB_DF71                ; perform SQR()
        .word   LAB_E094                ; perform RND()
        .word   LAB_D9EA                ; perform LOG()
        .word   LAB_DFED                ; perform EXP()
        .word   LAB_E261                ; perform COS()
        .word   LAB_E268                ; perform SIN()

        .word   LAB_E2B1                ; perform TAN()
        .word   LAB_E30B                ; perform ATN()
        .word   LAB_D80D                ; perform PEEK()
        .word   LAB_D77C                ; perform LEN()
        .word   LAB_D465                ; perform STR$()
        .word   LAB_D7AD                ; perform VAL()
        .word   LAB_D78B                ; perform ASC()
        .word   LAB_D6EC                ; perform CHR$()

        .word   LAB_D700                ; perform LEFT$()
        .word   LAB_D72C                ; perform RIGHT$()
        .word   LAB_D737                ; perform MID$()


;******************************************************************************;
;
; precedence byte and action addresses for operators. like the primarry commands these
; are called by pushing the address onto the stack and doing an RTS, so again the actual
; address -1 needs to be pushed

LAB_C080
        .byte   $79
        .word   LAB_D86A-1              ; +
        .byte   $79
        .word   LAB_D853-1              ; -
        .byte   $7B
        .word   LAB_DA2B-1              ; *
        .byte   $7B
        .word   LAB_DB12-1              ; /
```

```
        .byte   $7F
        .word   LAB_DF7B-1              ; ^
        .byte   $50
        .word   LAB_CFE9-1              ; AND
        .byte   $46
        .word   LAB_CFE6-1              ; OR
        .byte   $7D
        .word   LAB_DFB4-1              ; >
        .byte   $5A
        .word   LAB_CED4-1              ; =
LAB_C09B
        .byte   $64
        .word   LAB_D016-1              ; <


;******************************************************************************;
;
; BASIC keywords. each word has b7 set in it's last character as an end marker,
; even the one character keywords such as "<" or "="

; first are the primary command keywords, only these can start a statement

LAB_C09E
        .byte   "EN",'D'+$80    ; END
        .byte   "FO",'R'+$80    ; FOR
        .byte   "NEX",'T'+$80   ; NEXT
        .byte   "DAT",'A'+$80   ; DATA
        .byte   "INPUT",'#'+$80 ; INPUT#
        .byte   "INPU",'T'+$80  ; INPUT
        .byte   "DI",'M'+$80    ; DIM
        .byte   "REA",'D'+$80   ; READ

        .byte   "LE",'T'+$80    ; LET
        .byte   "GOT",'O'+$80   ; GOTO
        .byte   "RU",'N'+$80    ; RUN
        .byte   "I",'F'+$80             ; IF
        .byte   "RESTOR",'E'+$80        ; RESTORE
        .byte   "GOSU",'B'+$80  ; GOSUB
        .byte   "RETUR",'N'+$80 ; RETURN
        .byte   "RE",'M'+$80    ; REM

        .byte   "STO",'P'+$80   ; STOP
        .byte   "O",'N'+$80             ; ON
        .byte   "WAI",'T'+$80   ; WAIT
        .byte   "LOA",'D'+$80   ; LOAD
        .byte   "SAV",'E'+$80   ; SAVE
        .byte   "VERIF",'Y'+$80 ; VERIFY
        .byte   "DE",'F'+$80    ; DEF
        .byte   "POK",'E'+$80   ; POKE

        .byte   "PRINT",'#'+$80 ; PRINT#
        .byte   "PRIN",'T'+$80  ; PRINT
        .byte   "CON",'T'+$80   ; CONT
        .byte   "LIS",'T'+$80   ; LIST
        .byte   "CL",'R'+$80    ; CLR
        .byte   "CM",'D'+$80    ; CMD
        .byte   "SY",'S'+$80    ; SYS
        .byte   "OPE",'N'+$80   ; OPEN

        .byte   "CLOS",'E'+$80  ; CLOSE
        .byte   "GE",'T'+$80    ; GET
```

```
        .byte   "NE",'W'+$80    ; NEW

; next are the secondary command keywords, these can not start a statement

        .byte   "TAB",'('+$80   ; TAB(
        .byte   "T",'O'+$80             ; TO
        .byte   "F",'N'+$80             ; FN
        .byte   "SPC",'('+$80   ; SPC(
        .byte   "THE",'N'+$80   ; THEN

        .byte   "NO",'T'+$80    ; NOT
        .byte   "STE",'P'+$80   ; STEP

; the operators

        .byte   '+'+$80         ; +
        .byte   '-'+$80         ; -
        .byte   '*'+$80         ; *
        .byte   '/'+$80         ; /
        .byte   '^'+$80         ; ^
        .byte   "AN",'D'+$80    ; AND

        .byte   "O",'R'+$80             ; OR
        .byte   '>'+$80         ; >
        .byte   '='+$80         ; =
        .byte   '<'+$80         ; <

; the functions

        .byte   "SG",'N'+$80    ; SGN
        .byte   "IN",'T'+$80    ; INT
        .byte   "AB",'S'+$80    ; ABS
        .byte   "US",'R'+$80    ; USR

        .byte   "FR",'E'+$80    ; FRE
        .byte   "PO",'S'+$80    ; POS
        .byte   "SQ",'R'+$80    ; SQR
        .byte   "RN",'D'+$80    ; RND
        .byte   "LO",'G'+$80    ; LOG
        .byte   "EX",'P'+$80    ; EXP
        .byte   "CO",'S'+$80    ; COS
        .byte   "SI",'N'+$80    ; SIN

        .byte   "TA",'N'+$80    ; TAN
        .byte   "AT",'N'+$80    ; ATN
        .byte   "PEE",'K'+$80   ; PEEK
        .byte   "LE",'N'+$80    ; LEN
        .byte   "STR",'$'+$80   ; STR$
        .byte   "VA",'L'+$80    ; VAL
        .byte   "AS",'C'+$80    ; ASC
        .byte   "CHR",'$'+$80   ; CHR$

        .byte   "LEFT",'$'+$80  ; LEFT$
        .byte   "RIGHT",'$'+$80 ; RIGHT$
        .byte   "MID",'$'+$80   ; MID$

; lastly is GO, this is an add on so that GO TO, as well as GOTO, will work

        .byte   "G",'O'+$80             ; GO

        .byte   $00                     ; end marker
```

```
;**************************************************************************************;
;
; error messages

LAB_C19E
        .byte   "TOO MANY FILE",'S'+$80
LAB_C1AC
        .byte   "FILE OPE",'N'+$80
LAB_C1B5
        .byte   "FILE NOT OPE",'N'+$80
LAB_C1C2
        .byte   "FILE NOT FOUN",'D'+$80
LAB_C1D0
        .byte   "DEVICE NOT PRESEN",'T'+$80
LAB_C1E2
        .byte   "NOT INPUT FIL",'E'+$80
LAB_C1F0
        .byte   "NOT OUTPUT FIL",'E'+$80
LAB_C1FF
        .byte   "MISSING FILE NAM",'E'+$80
LAB_C210
        .byte   "ILLEGAL DEVICE NUMBE",'R'+$80
LAB_C225
        .byte   "NEXT WITHOUT FO",'R'+$80
LAB_C235
        .byte   "SYNTA",'X'+$80
LAB_C23B
        .byte   "RETURN WITHOUT GOSU",'B'+$80
LAB_C24F
        .byte   "OUT OF DAT",'A'+$80
LAB_C25A
        .byte   "ILLEGAL QUANTIT",'Y'+$80
LAB_C26A
        .byte   "OVERFLO",'W'+$80
LAB_C272
        .byte   "OUT OF MEMOR",'Y'+$80
LAB_C27F
        .byte   "UNDEF'D STATEMEN",'T'+$80
LAB_C290
        .byte   "BAD SUBSCRIP",'T'+$80
LAB_C29D
        .byte   "REDIM'D ARRA",'Y'+$80
LAB_C2AA
        .byte   "DIVISION BY ZER",'O'+$80
LAB_C2BA
        .byte   "ILLEGAL DIREC",'T'+$80
LAB_C2C8
        .byte   "TYPE MISMATC",'H'+$80
LAB_C2D5
        .byte   "STRING TOO LON",'G'+$80
LAB_C2E4
        .byte   "FILE DAT",'A'+$80
LAB_C2ED
        .byte   "FORMULA TOO COMPLE",'X'+$80
LAB_C300
        .byte   "CAN'T CONTINU",'E'+$80
LAB_C30E
        .byte   "UNDEF'D FUNCTIO",'N'+$80
LAB_C31E
```

```
        .byte   "VERIF",'Y'+$80
LAB_C324
        .byte   "LOA",'D'+$80

; error message pointer table

LAB_C328
        .word   LAB_C19E                ; $01   TOO MANY FILES
        .word   LAB_C1AC                ; $02   FILE OPEN
        .word   LAB_C1B5                ; $03   FILE NOT OPEN
        .word   LAB_C1C2                ; $04   FILE NOT FOUND
        .word   LAB_C1D0                ; $05   DEVICE NOT PRESENT
        .word   LAB_C1E2                ; $06   NOT INPUT FILE
        .word   LAB_C1F0                ; $07   NOT OUTPUT FILE
        .word   LAB_C1FF                ; $08   MISSING FILE NAME
        .word   LAB_C210                ; $09   ILLEGAL DEVICE NUMBER
        .word   LAB_C225                ; $0A   NEXT WITHOUT FOR
        .word   LAB_C235                ; $0B   SYNTAX
        .word   LAB_C23B                ; $0C   RETURN WITHOUT GOSUB
        .word   LAB_C24F                ; $0D   OUT OF DATA
        .word   LAB_C25A                ; $0E   ILLEGAL QUANTITY
        .word   LAB_C26A                ; $0F   OVERFLOW
        .word   LAB_C272                ; $10   OUT OF MEMORY
        .word   LAB_C27F                ; $11   UNDEF'D STATEMENT
        .word   LAB_C290                ; $12   BAD SUBSCRIPT
        .word   LAB_C29D                ; $13   REDIM'D ARRAY
        .word   LAB_C2AA                ; $14   DIVISION BY ZERO
        .word   LAB_C2BA                ; $15   ILLEGAL DIRECT
        .word   LAB_C2C8                ; $16   TYPE MISMATCH
        .word   LAB_C2D5                ; $17   STRING TOO LONG
        .word   LAB_C2E4                ; $18   FILE DATA
        .word   LAB_C2ED                ; $19   FORMULA TOO COMPLEX
        .word   LAB_C300                ; $1A   CAN'T CONTINUE
        .word   LAB_C30E                ; $1B   UNDEF'D FUNCTION
        .word   LAB_C31E                ; $1C   VERIFY
        .word   LAB_C324                ; $1D   LOAD
        .word   LAB_C383                ; $1E   BREAK


;******************************************************************************;
;
; BASIC messages

LAB_C364
        .byte   $0D,"OK",$0D,$00
LAB_C369
        .byte   $0D," ERROR",$00
LAB_C371
        .byte   " IN ",$00
LAB_C376
        .byte   $0D,$0A,"READY.",$0D,$0A,$00
LAB_C381
        .byte   $0D,$0A
LAB_C383
        .byte   "BREAK",$00


;******************************************************************************;
;
; spare byte, not referenced
```

```
;LAB_C389
        .byte   $A0


;******************************************************************************;
;
; search the stack for FOR or GOSUB activity
; return Zb=1 if FOR variable found

LAB_C38A
        TSX                             ; copy stack pointer
        INX                             ; +1 pass return address
        INX                             ; +2 pass return address
        INX                             ; +3 pass calling routine return address
        INX                             ; +4 pass calling routine return address
LAB_C38F
        LDA     LAB_0100+1,X    ; get token byte from stack
        CMP     #TK_FOR         ; is it FOR token
        BNE     LAB_C3B7                ; exit if not FOR token

                                        ; was FOR token
        LDA     LAB_4A          ; get FOR/NEXT variable pointer high byte
        BNE     LAB_C3A4                ; branch if not null

        LDA     LAB_0100+2,X    ; get FOR variable pointer low byte
        STA     LAB_49          ; save FOR/NEXT variable pointer low byte
        LDA     LAB_0100+3,X    ; get FOR variable pointer high byte
        STA     LAB_4A          ; save FOR/NEXT variable pointer high byte
LAB_C3A4
        CMP     LAB_0100+3,X    ; compare variable pointer with stacked variable pointer
                                        ; high byte
        BNE     LAB_C3B0                ; branch if no match

        LDA     LAB_49          ; get FOR/NEXT variable pointer low byte
        CMP     LAB_0100+2,X    ; compare variable pointer with stacked variable pointer
                                        ; low byte
        BEQ     LAB_C3B7                ; exit if match found

LAB_C3B0
        TXA                             ; copy index
        CLC                             ; clear carry for add
        ADC     #$12                    ; add FOR stack use size
        TAX                             ; copy back to index
        BNE     LAB_C38F                ; loop if not at start of stack

LAB_C3B7
        RTS


;******************************************************************************;
;
; open up space in memory, set end of arrays

LAB_C3B8
        JSR     LAB_C408                ; check available memory, do out of memory error if no
room
        STA     LAB_31          ; set end of arrays low byte
        STY     LAB_32          ; set end of arrays high byte

; open up space in memory, don't set array end
```

```
LAB_C3BF
        SEC                             ; set carry for subtract
        LDA     LAB_5A          ; get block end low byte
        SBC     LAB_5F          ; subtract block start low byte
        STA     LAB_22          ; save MOD(block length/$100) byte
        TAY                             ; copy MOD(block length/$100) byte to Y
        LDA     LAB_5B          ; get block end high byte
        SBC     LAB_60          ; subtract block start high byte
        TAX                             ; copy block length high byte to X
        INX                             ; +1 to allow for count=0 exit
        TYA                             ; copy block length low byte to A
        BEQ     LAB_C3F3                ; branch if length low byte=0

                                        ; block is (X-1)*256+Y bytes, do the Y bytes first
        LDA     LAB_5A          ; get block end low byte
        SEC                             ; set carry for subtract
        SBC     LAB_22          ; subtract MOD(block length/$100) byte
        STA     LAB_5A          ; save corrected old block end low byte
        BCS     LAB_C3DC                ; if no underflow skip the high byte decrement

        DEC     LAB_5B          ; else decrement block end high byte
        SEC                             ; set carry for subtract
LAB_C3DC
        LDA     LAB_58          ; get destination end low byte
        SBC     LAB_22          ; subtract MOD(block length/$100) byte
        STA     LAB_58          ; save modified new block end low byte
        BCS     LAB_C3EC                ; if no underflow skip the high byte decrement

        DEC     LAB_59          ; else decrement block end high byte
        BCC     LAB_C3EC                ; branch always

LAB_C3E8
        LDA     (LAB_5A),Y              ; get byte from source
        STA     (LAB_58),Y              ; copy byte to destination
LAB_C3EC
        DEY                             ; decrement index
        BNE     LAB_C3E8                ; loop until Y=0

                                        ; now do Y=0 indexed byte
        LDA     (LAB_5A),Y              ; get byte from source
        STA     (LAB_58),Y              ; save byte to destination
LAB_C3F3
        DEC     LAB_5B          ; decrement source pointer high byte
        DEC     LAB_59          ; decrement destination pointer high byte
        DEX                             ; decrement block count
        BNE     LAB_C3EC                ; loop until count = $0

        RTS


;********************************************************************************;
;
; check there is room on the stack for A bytes
; if the stack is too deep do an out of memory error

LAB_C3FB
        ASL                             ; *2
        ADC     #$3E                    ; need at least $3E bytes free
        BCS     LAB_C435                ; if overflow go do out of memory error then warm
start
```

```
        STA     LAB_22          ; save result in temp byte
        TSX                             ; copy stack
        CPX     LAB_22          ; compare new limit with stack
        BCC     LAB_C435                ; if stack < limit do out of memory error then warm
start

        RTS


;**************************************************************************;
;
; check available memory, do out of memory error if no room

LAB_C408
        CPY     LAB_34          ; compare with bottom of string space high byte
        BCC     LAB_C434                ; if less then exit (is ok)

        BNE     LAB_C412                ; skip next test if greater (tested <)

                                        ; high byte was =, now do low byte
        CMP     LAB_33          ; compare with bottom of string space low byte
        BCC     LAB_C434                ; if less then exit (is ok)

                                        ; address is > string storage ptr (oops!)
LAB_C412
        PHA                             ; push address low byte
        LDX     #$09                    ; set index to save LAB_57 to LAB_60 inclusive
        TYA                             ; copy address high byte (to push on stack)

                                        ; save misc numeric work area
LAB_C416
        PHA                             ; push byte
        LDA     LAB_57,X                ; get byte from LAB_57 to LAB_60
        DEX                             ; decrement index
        BPL     LAB_C416                ; loop until all done

        JSR     LAB_D526                ; do garbage collection routine

                                        ; restore misc numeric work area
        LDX     #$F7                    ; set index to restore bytes
LAB_C421
        PLA                             ; pop byte
        STA     LAB_60+1,X              ; save byte to LAB_57 to LAB_60
        INX                             ; increment index
        BMI     LAB_C421                ; loop while -ve

        PLA                             ; pop address high byte
        TAY                             ; copy back to Y
        PLA                             ; pop address low byte
        CPY     LAB_34          ; compare with bottom of string space high byte
        BCC     LAB_C434                ; if less then exit (is ok)

        BNE     LAB_C435                ; if greater do out of memory error then warm start

                                        ; high byte was =, now do low byte
        CMP     LAB_33          ; compare with bottom of string space low byte
        BCS     LAB_C435                ; if >= do out of memory error then warm start

                                        ; ok exit, carry clear
LAB_C434
        RTS
```

```
;**************************************************************************;
;
; do out of memory error then warm start

LAB_C435
        LDX     #$10                    ; error code $10, out of memory error

; do error #X then warm start

LAB_C437
        JMP     (LAB_0300)              ; do error message

; do error #X then warm start, the error message vector is initialised to point here

LAB_C43A
        TXA                             ; copy error number
        ASL                             ; *2
        TAX                             ; copy to index
        LDA     LAB_C328-2,X    ; get error message pointer low byte
        STA     LAB_22          ; save it
        LDA     LAB_C328-1,X    ; get error message pointer high byte
        STA     LAB_23          ; save it
        JSR     LAB_FFCC                ; close input and output channels
        LDA     #$00                    ; clear A
        STA     LAB_13          ; clear current I/O channel, flag default
        JSR     LAB_CAD7                ; print CR/LF
        JSR     LAB_CB45                ; print "?"
        LDY     #$00                    ; clear index
LAB_C456
        LDA     (LAB_22),Y              ; get byte from message
        PHA                             ; save status
        AND     #$7F                    ; mask 0xxx xxxx, clear b7
        JSR     LAB_CB47                ; output character
        INY                             ; increment index
        PLA                             ; restore status
        BPL     LAB_C456                ; loop if character was not end marker

        JSR     LAB_C67A                ; flush BASIC stack and clear continue pointer
        LDA     #<LAB_C369              ; set " ERROR" pointer low byte
        LDY     #>LAB_C369              ; set " ERROR" pointer high byte


;**************************************************************************;
;
; print string and do warm start, break entry

LAB_C469
        JSR     LAB_CB1E                ; print null terminated string
        LDY     LAB_3A          ; get current line number high byte
        INY                             ; increment it
        BEQ     LAB_C474                ; branch if was in immediate mode

        JSR     LAB_DDC2                ; do " IN " line number message


;**************************************************************************;
;
; do warm start
```

```
LAB_C474
        LDA     #<LAB_C376              ; set "READY." pointer low byte
        LDY     #>LAB_C376              ; set "READY." pointer high byte
        JSR     LAB_CB1E                ; print null terminated string
        LDA     #$80                    ; set for control messages only
        JSR     LAB_FF90                ; control kernal messages
LAB_C480
        JMP     (LAB_0302)              ; do BASIC warm start


;******************************************************************************;
;
; BASIC warm start, the warm start vector is initialised to point here

LAB_C483
        JSR     LAB_C560                ; call for BASIC input
        STX     LAB_7A          ; save BASIC execute pointer low byte
        STY     LAB_7B          ; save BASIC execute pointer high byte
        JSR     LAB_0073                ; increment and scan memory
        TAX                             ; copy byte to set flags
        BEQ     LAB_C480                ; loop if no input

; got to interpret input line now ....

        LDX     #$FF                    ; current line high byte to -1, indicates immediate
mode
        STX     LAB_3A          ; set current line number high byte
        BCC     LAB_C49C                ; if numeric character go handle new BASIC line

                                        ; no line number .. immediate mode
        JSR     LAB_C579                ; crunch keywords into BASIC tokens
        JMP     LAB_C7E1                ; go scan and interpret code

; handle new BASIC line

LAB_C49C
        JSR     LAB_C96B                ; get fixed-point number into temporary integer
        JSR     LAB_C579                ; crunch keywords into BASIC tokens
        STY     LAB_0B          ; save index pointer to end of crunched line
        JSR     LAB_C613                ; search BASIC for temporary integer line number
        BCC     LAB_C4ED                ; if not found skip the line delete

                                        ; line # already exists so delete it
        LDY     #$01                    ; set index to next line pointer high byte
        LDA     (LAB_5F),Y              ; get next line pointer high byte
        STA     LAB_23          ; save it
        LDA     LAB_2D          ; get start of variables low byte
        STA     LAB_22          ; save it
        LDA     LAB_60          ; get found line pointer high byte
        STA     LAB_25          ; save it
        LDA     LAB_5F          ; get found line pointer low byte
        DEY                             ; decrement index
        SBC     (LAB_5F),Y              ; subtract next line pointer low byte
        CLC                             ; clear carry for add
        ADC     LAB_2D          ; add start of variables low byte
        STA     LAB_2D          ; set start of variables low byte
        STA     LAB_24          ; save destination pointer low byte
        LDA     LAB_2E          ; get start of variables high byte
        ADC     #$FF                    ; -1 + carry
        STA     LAB_2E          ; set start of variables high byte
        SBC     LAB_60          ; subtract found line pointer high byte
```

```
        TAX                             ; copy to block count
        SEC                             ; set carry for subtract
        LDA     LAB_5F          ; get found line pointer low byte
        SBC     LAB_2D          ; subtract start of variables low byte
        TAY                             ; copy to bytes in first block count
        BCS     LAB_C4D7                ; if no underflow skip the high byte decrement

        INX                             ; increment block count, correct for = 0 loop exit
        DEC     LAB_25          ; decrement destination high byte
LAB_C4D7
        CLC                             ; clear carry for add
        ADC     LAB_22          ; add source pointer low byte
        BCC     LAB_C4DF                ; if no underflow skip the high byte decrement

        DEC     LAB_23          ; else decrement source pointer high byte
        CLC                             ; clear carry

                                        ; close up memory to delete old line
LAB_C4DF
        LDA     (LAB_22),Y              ; get byte from source
        STA     (LAB_24),Y              ; copy to destination
        INY                             ; increment index
        BNE     LAB_C4DF                ; while <> 0 do this block

        INC     LAB_23          ; increment source pointer high byte
        INC     LAB_25          ; increment destination pointer high byte
        DEX                             ; decrement block count
        BNE     LAB_C4DF                ; loop until all done

                                        ; got new line in buffer and no existing same #
LAB_C4ED
        JSR     LAB_C659                ; reset execution to start, clear variables, flush
stack
                                        ; and return
        JSR     LAB_C533                ; rebuild BASIC line chaining
        LDA     LAB_0200                ; get first byte from buffer
        BEQ     LAB_C480                ; if no line go do BASIC warm start

                                        ; else insert line into memory
        CLC                             ; clear carry for add
        LDA     LAB_2D          ; get start of variables low byte
        STA     LAB_5A          ; save as source end pointer low byte
        ADC     LAB_0B          ; add index pointer to end of crunched line
        STA     LAB_58          ; save as destination end pointer low byte
        LDY     LAB_2E          ; get start of variables high byte
        STY     LAB_5B          ; save as source end pointer high byte
        BCC     LAB_C508                ; if no carry skip the high byte increment

        INY                             ; else increment the high byte
LAB_C508
        STY     LAB_59          ; save as destination end pointer high byte
        JSR     LAB_C3B8                ; open up space in memory

; most of what remains to do is copy the crunched line into the space opened up in memory,
; however, before the crunched line comes the next line pointer and the line number. the
; line number is retrieved from the temporary integer and stored in memory, this
; overwrites the bottom two bytes on the stack. next the line is copied and the next line
; pointer is filled with whatever was in two bytes above the line number in the stack.
; this is ok because the line pointer gets fixed in the line chain re-build.

        LDA     LAB_14          ; get line number low byte
```

```
        LDY     LAB_15          ; get line number high byte
        STA     LAB_01FE                ; save line number low byte before crunched line
        STY     LAB_01FF                ; save line number high byte before crunched line
        LDA     LAB_31          ; get end of arrays low byte
        LDY     LAB_32          ; get end of arrays high byte
        STA     LAB_2D          ; set start of variables low byte
        STY     LAB_2E          ; set start of variables high byte
        LDY     LAB_0B          ; get index to end of crunched line
        DEY                             ; -1
LAB_C522
        LDA     LAB_01FC,Y              ; get byte from crunched line
        STA     (LAB_5F),Y              ; save byte to memory
        DEY                             ; decrement index
        BPL     LAB_C522                ; loop while more to do

; reset execution, clear variables, flush stack, rebuild BASIC chain and do warm start

LAB_C52A
        JSR     LAB_C659                ; reset execution to start, clear variables and flush
stack
        JSR     LAB_C533                ; rebuild BASIC line chaining
        JMP     LAB_C480                ; go do BASIC warm start


;**************************************************************************;
;
; rebuild BASIC line chaining

LAB_C533
        LDA     LAB_2B          ; get start of memory low byte
        LDY     LAB_2C          ; get start of memory high byte
        STA     LAB_22          ; set line start pointer low byte
        STY     LAB_23          ; set line start pointer high byte
        CLC                             ; clear carry for add
LAB_C53C
        LDY     #$01                    ; set index to pointer to next line high byte
        LDA     (LAB_22),Y              ; get pointer to next line high byte
        BEQ     LAB_C55F                ; exit if null, [EOT]

        LDY     #$04                    ; point to first code byte of line
                                        ; there is always 1 byte + [EOL] as null entries are
deleted
LAB_C544
        INY                             ; next code byte
        LDA     (LAB_22),Y              ; get byte
        BNE     LAB_C544                ; loop if not [EOL]

        INY                             ; point to byte past [EOL], start of next line
        TYA                             ; copy it
        ADC     LAB_22          ; add line start pointer low byte
        TAX                             ; copy to X
        LDY     #$00                    ; clear index, point to this line's next line pointer
        STA     (LAB_22),Y              ; set next line pointer low byte
        LDA     LAB_23          ; get line start pointer high byte
        ADC     #$00                    ; add any overflow
        INY                             ; increment index to high byte
        STA     (LAB_22),Y              ; set next line pointer high byte
        STX     LAB_22          ; set line start pointer low byte
        STA     LAB_23          ; set line start pointer high byte
        BCC     LAB_C53C                ; go do next line, branch always
```

```
LAB_C55F
        RTS


;******************************************************************************;
;
; call for BASIC input

LAB_C560
        LDX     #$00                    ; set channel $00, keyboard
LAB_C562
        JSR     LAB_E10F                ; input character from channel with error check
        CMP     #$0D                    ; compare with [CR]
        BEQ     LAB_C576                ; if [CR] set XY to LAB_200 - 1, print [CR] and exit

                                        ; character was not [CR]
        STA     LAB_0200,X              ; save character to buffer
        INX                             ; increment buffer index
        CPX     #$59                    ; compare with max+1
        BCC     LAB_C562                ; branch if < max+1

        LDX     #$17                    ; error $17, string too long error
        JMP     LAB_C437                ; do error #X then warm start

LAB_C576
        JMP     LAB_CACA                ; set XY to LAB_200 - 1 and print [CR]


;******************************************************************************;
;
; crunch BASIC tokens vector

LAB_C579
        JMP     (LAB_0304)              ; do crunch BASIC tokens


;******************************************************************************;
;
; crunch BASIC tokens, the crunch BASIC tokens vector is initialised to point here

LAB_C57C
        LDX     LAB_7A          ; get BASIC execute pointer low byte
        LDY     #$04                    ; set save index
        STY     LAB_0F          ; clear open quote/DATA flag
LAB_C582
        LDA     LAB_0200,X              ; get a byte from the input buffer
        BPL     LAB_C58E                ; if b7 clear go do crunching

        CMP     #TK_PI          ; compare with the token for PI, this toke is input
                                        ; directly from the keyboard as the PI character.
        BEQ     LAB_C5C9                ; if PI save byte then continue crunching

                                        ; this is the bit of code that stops you being able to
enter
                                        ; some keywords as just single shifted characters. If
this
                                        ; dropped through you would be able to enter GOTO as
just
                                        ; [SHIFT]G

        INX                             ; increment read index
```

```
        BNE     LAB_C582                ; loop if more to do, branch always

LAB_C58E
        CMP     #' '                    ; compare with [SPACE]
        BEQ     LAB_C5C9                ; if [SPACE] save byte then continue crunching

        STA     LAB_08          ; save buffer byte as search character
        CMP     #$22                    ; compare with quote character
        BEQ     LAB_C5EE                ; if quote go copy quoted string

        BIT     LAB_0F          ; get open quote/DATA token flag
        BVS     LAB_C5C9                ; branch if b6 of Oquote set, was DATA
                                        ; go save byte then continue crunching

        CMP     #'?'                    ; compare with "?" character
        BNE     LAB_C5A4                ; if not "?" continue crunching

        LDA     #TK_PRINT               ; else set the token for PRINT
        BNE     LAB_C5C9                ; go save byte then continue crunching ,branch always

LAB_C5A4
        CMP     #'0'                    ; compare with "0"
        BCC     LAB_C5AC                ; if < "0" continue crunching

        CMP     #'<'                    ; compare with "<"
        BCC     LAB_C5C9                ; if <, 0123456789:; go save byte then continue
crunching

                                        ; gets here with next character not numeric, ";" or
":"
LAB_C5AC
        STY     LAB_71          ; copy save index
        LDY     #$00                    ; clear table pointer
        STY     LAB_0B          ; clear word index
        DEY                             ; adjust for pre increment loop
        STX     LAB_7A          ; save BASIC execute pointer low byte, buffer index
        DEX                             ; adjust for pre increment loop
LAB_C5B6
        INY                             ; next table byte
        INX                             ; next buffer byte
LAB_C5B8
        LDA     LAB_0200,X              ; get byte from input buffer
        SEC                             ; set carry for subtract
        SBC     LAB_C09E,Y              ; subtract table byte
        BEQ     LAB_C5B6                ; go compare next if match

        CMP     #$80                    ; was it end marker match ?
        BNE     LAB_C5F5                ; if not go try the next keyword

                                        ; actually this works even if the input buffer byte is
the
                                        ; end marker, i.e. a shifted character. As you can't
enter
                                        ; any keywords as a single shifted character, see
above,
                                        ; you can enter keywords in shorthand by shifting any
                                        ; character after the first. so RETURN can be entered
as
                                        ; R[SHIFT]E, RE[SHIFT]T, RET[SHIFT]U or RETU[SHIFT]R.
                                        ; RETUR[SHIFT]N however will not work because the
[SHIFT]N
```

```
                                        ; will match the RETURN end marker so the routine will
try
                                        ; to match the next character.

                                        ; else found keyword
        ORA     LAB_0B          ; OR with word index, +$80 in A makes token
LAB_C5C7
        LDY     LAB_71          ; restore save index

; save byte then continue crunching

LAB_C5C9
        INX                             ; increment buffer read index
        INY                             ; increment save index
        STA     LAB_0200-5,Y    ; save byte to output
        LDA     LAB_0200-5,Y    ; get byte from output, set flags
        BEQ     LAB_C609                ; branch if was null [EOL]

                                        ; A holds the token here
        SEC                             ; set carry for subtract
        SBC     #':'                    ; subtract ":"
        BEQ     LAB_C5DC                ; branch if it was (is now $00)

                                        ; A now holds token-':'
        CMP     #TK_DATA-':'    ; compare with the token for DATA-':'
        BNE     LAB_C5DE                ; if not DATA go try REM

                                        ; token was : or DATA
LAB_C5DC
        STA     LAB_0F          ; save token-':'
LAB_C5DE
        SEC                             ; set carry for subtract
        SBC     #TK_REM-':'             ; subtract the token for REM-':'
        BNE     LAB_C582                ; if wasn't REM go crunch next bit of line

        STA     LAB_08          ; else was REM so set search for [EOL]

                                        ; loop for "..." etc.
LAB_C5E5
        LDA     LAB_0200,X              ; get byte from input buffer
        BEQ     LAB_C5C9                ; if null [EOL] save byte then continue crunching

        CMP     LAB_08          ; compare with stored character
        BEQ     LAB_C5C9                ; if match save byte then continue crunching

LAB_C5EE
        INY                             ; increment save index
        STA     LAB_0200-5,Y    ; save byte to output
        INX                             ; increment buffer index
        BNE     LAB_C5E5                ; loop while <> 0, should never reach 0

                                        ; not found keyword this go
LAB_C5F5
        LDX     LAB_7A          ; restore BASIC execute pointer low byte
        INC     LAB_0B          ; increment word index (next word)

                                        ; now find end of this word in the table
LAB_C5F9
        INY                             ; increment table index
        LDA     LAB_C09E-1,Y    ; get table byte
        BPL     LAB_C5F9                ; loop if not end of word yet
```

```
        LDA     LAB_C09E,Y              ; get byte from keyword table
        BNE     LAB_C5B8                ; go test next word if not zero byte, end of table

                                        ; reached end of table with no match
        LDA     LAB_0200,X              ; restore byte from input buffer
        BPL     LAB_C5C7                ; branch always, all unmatched bytes in the buffer are
                                        ; $00 to $7F, go save byte in output and continue
crunching

                                        ; reached [EOL]
LAB_C609
        STA     LAB_0200-3,Y    ; save [EOL]
        DEC     LAB_7B          ; decrement BASIC execute pointer high byte
        LDA     #$FF                    ; point to start of buffer-1
        STA     LAB_7A          ; set BASIC execute pointer low byte
        RTS


;******************************************************************************;
;
; search BASIC for temporary integer line number

LAB_C613
        LDA     LAB_2B          ; get start of memory low byte
        LDX     LAB_2C          ; get start of memory high byte

; search Basic for temp integer line number from AX
; returns carry set if found

LAB_C617
        LDY     #$01                    ; set index to next line pointer high byte
        STA     LAB_5F          ; save low byte as current
        STX     LAB_60          ; save high byte as current
        LDA     (LAB_5F),Y              ; get next line pointer high byte from address
        BEQ     LAB_C640                ; pointer was zero so done, exit

        INY                             ; increment index ...
        INY                             ; ... to line # high byte
        LDA     LAB_15          ; get temporary integer high byte
        CMP     (LAB_5F),Y              ; compare with line # high byte
        BCC     LAB_C641                ; exit if temp < this line, target line passed

        BEQ     LAB_C62E                ; go check low byte if =

        DEY                             ; else decrement index
        BNE     LAB_C637                ; branch always

LAB_C62E
        LDA     LAB_14          ; get temporary integer low byte
        DEY                             ; decrement index to line # low byte
        CMP     (LAB_5F),Y              ; compare with line # low byte
        BCC     LAB_C641                ; exit if temp < this line, target line passed

        BEQ     LAB_C641                ; exit if temp = (found line#)

                                        ; not quite there yet
LAB_C637
        DEY                             ; decrement index to next line pointer high byte
        LDA     (LAB_5F),Y              ; get next line pointer high byte
        TAX                             ; copy to X
```

```
        DEY                             ; decrement index to next line pointer low byte
        LDA     (LAB_5F),Y              ; get next line pointer low byte
        BCS     LAB_C617                ; go search for line # in temporary integer
                                        ; from AX, carry always set

LAB_C640
        CLC                             ; clear found flag
LAB_C641
        RTS


;******************************************************************************;
;
; perform NEW

LAB_C642
        BNE     LAB_C641                ; exit if following byte to allow syntax error

LAB_C644
        LDA     #$00                    ; clear A
        TAY                             ; clear index
        STA     (LAB_2B),Y              ; clear pointer to next line low byte
        INY                             ; increment index
        STA     (LAB_2B),Y              ; clear pointer to next line high byte, erase program

        LDA     LAB_2B          ; get start of memory low byte
        CLC                             ; clear carry for add
        ADC     #$02                    ; add null program length
        STA     LAB_2D          ; set start of variables low byte
        LDA     LAB_2C          ; get start of memory high byte
        ADC     #$00                    ; add carry
        STA     LAB_2E          ; set start of variables high byte

; reset execute pointer and do CLR

LAB_C659
        JSR     LAB_C68E                ; set BASIC execute pointer to start of memory - 1
        LDA     #$00                    ; set Zb for CLR entry


;******************************************************************************;
;
; perform CLR

LAB_C65E
        BNE     LAB_C68D                ; exit if following byte to allow syntax error

LAB_C660
        JSR     LAB_FFE7                ; close all channels and files
LAB_C663
        LDA     LAB_37          ; get end of memory low byte
        LDY     LAB_38          ; get end of memory high byte
        STA     LAB_33          ; set bottom of string space low byte, clear strings
        STY     LAB_34          ; set bottom of string space high byte
        LDA     LAB_2D          ; get start of variables low byte
        LDY     LAB_2E          ; get start of variables high byte
        STA     LAB_2F          ; set end of variables low byte, clear variables
        STY     LAB_30          ; set end of variables high byte
        STA     LAB_31          ; set end of arrays low byte, clear arrays
        STY     LAB_32          ; set end of arrays high byte
```

```
;**************************************************************************;
;
; do RESTORE and clear the stack

LAB_C677
        JSR     LAB_C81D                ; perform RESTORE

; flush BASIC stack and clear the continue pointer

LAB_C67A
        LDX     #LAB_19         ; get descriptor stack start
        STX     LAB_16          ; set descriptor stack pointer
        PLA                             ; pull return address low byte
        TAY                             ; copy it
        PLA                             ; pull return address high byte
        LDX     #$FA                    ; set cleared stack pointer
        TXS                             ; set stack
        PHA                             ; push return address high byte
        TYA                             ; restore return address low byte
        PHA                             ; push return address low byte
        LDA     #$00                    ; clear A
        STA     LAB_3E          ; clear continue pointer high byte
        STA     LAB_10          ; clear subscript/FNX flag
LAB_C68D
        RTS


;**************************************************************************;
;
; set BASIC execute pointer to start of memory - 1

LAB_C68E
        CLC                             ; clear carry for add
        LDA     LAB_2B          ; get start of memory low byte
        ADC     #$FF                    ; add -1 low byte
        STA     LAB_7A          ; set BASIC execute pointer low byte
        LDA     LAB_2C          ; get start of memory high byte
        ADC     #$FF                    ; add -1 high byte
        STA     LAB_7B          ; save BASIC execute pointer high byte
        RTS


;**************************************************************************;
;
; perform LIST

LAB_C69C
        BCC     LAB_C6A4                ; branch if next character not token (LIST n...)

        BEQ     LAB_C6A4                ; branch if next character [NULL] (LIST)

        CMP     #TK_MINUS               ; compare with token for -
        BNE     LAB_C68D                ; exit if not - (LIST -m)

                                        ; LIST [[n][-m]]
                                        ; this bit sets the n , if present, as the start and
end
LAB_C6A4
        JSR     LAB_C96B                ; get fixed-point number into temporary integer
        JSR     LAB_C613                ; search BASIC for temporary integer line number
```

```
        JSR     LAB_0079                ; scan memory
        BEQ     LAB_C6BB                ; branch if no more chrs

                                        ; this bit checks the - is present
        CMP     #TK_MINUS               ; compare with "-"
        BNE     LAB_C641                ; return if not "-" (will be SN error)

                                        ; LIST [n]-m
                                        ; the - was there so set m as the end value
        JSR     LAB_0073                ; increment and scan memory
        JSR     LAB_C96B                ; get fixed-point number into temporary integer
        BNE     LAB_C641                ; exit if not ok

LAB_C6BB
        PLA                             ; dump return address low byte, exit via warm start
        PLA                             ; dump return address high byte
        LDA     LAB_14          ; get temporary integer low byte
        ORA     LAB_15          ; OR temporary integer high byte
        BNE     LAB_C6C9                ; branch if start set

        LDA     #$FF                    ; set for -1
        STA     LAB_14          ; set temporary integer low byte
        STA     LAB_15          ; set temporary integer high byte
LAB_C6C9
        LDY     #$01                    ; set index for line
        STY     LAB_0F          ; clear open quote flag
        LDA     (LAB_5F),Y              ; get next line pointer high byte
        BEQ     LAB_C714                ; if null all done so exit

        JSR     LAB_C82C                ; do CRTL-C check vector
        JSR     LAB_CAD7                ; print CR/LF
        INY                             ; increment index for line
        LDA     (LAB_5F),Y              ; get line number low byte
        TAX                             ; copy to X
        INY                             ; increment index
        LDA     (LAB_5F),Y              ; get line number high byte
        CMP     LAB_15          ; compare with temporary integer high byte
        BNE     LAB_C6E6                ; branch if no high byte match

        CPX     LAB_14          ; compare with temporary integer low byte
        BEQ     LAB_C6E8                ; branch if = last line to do, < will pass next branch

LAB_C6E6                                ; else ...
        BCS     LAB_C714                ; if greater all done so exit

LAB_C6E8
        STY     LAB_49          ; save index for line
        JSR     LAB_DDCD                ; print XA as unsigned integer
        LDA     #' '                    ; space is the next character
LAB_C6EF
        LDY     LAB_49          ; get index for line
        AND     #$7F                    ; mask top out bit of character
LAB_C6F3
        JSR     LAB_CB47                ; go print the character
        CMP     #$22                    ; was it " character
        BNE     LAB_C700                ; if not skip the quote handle

                                        ; we are either entering or leaving a pair of quotes
        LDA     LAB_0F          ; get open quote flag
        EOR     #$FF                    ; toggle it
        STA     LAB_0F          ; save it back
```

```
LAB_C700
        INY                             ; increment index
        BEQ     LAB_C714                ; line too long so just bail out and do a warm start

        LDA     (LAB_5F),Y              ; get next byte
        BNE     LAB_C717                ; if not [EOL] (go print character)

                                        ; was [EOL]
        TAY                             ; else clear index
        LDA     (LAB_5F),Y              ; get next line pointer low byte
        TAX                             ; copy to X
        INY                             ; increment index
        LDA     (LAB_5F),Y              ; get next line pointer high byte
        STX     LAB_5F          ; set pointer to line low byte
        STA     LAB_60          ; set pointer to line high byte
        BNE     LAB_C6C9                ; go do next line if not [EOT]
                                        ; else ...
LAB_C714
        JMP     LAB_C474                ; do warm start


;******************************************************************************;
;
LAB_C717
        JMP     (LAB_0306)              ; do uncrunch BASIC tokens


;******************************************************************************;
;
; uncrunch BASIC tokens, the uncrunch BASIC tokens vector is initialised to point here

LAB_C71A
        BPL     LAB_C6F3                ; just go print it if not token byte
                                        ; else was token byte so uncrunch it

        CMP     #TK_PI          ; compare with the token for PI. in this case the token
                                        ; is the same as the PI character so it just needs
printing
        BEQ     LAB_C6F3                ; just print it if so

        BIT     LAB_0F          ; test the open quote flag
        BMI     LAB_C6F3                ; just go print character if open quote set

        SEC                             ; else set carry for subtract
        SBC     #$7F                    ; reduce token range to 1 to whatever
        TAX                             ; copy token # to X
        STY     LAB_49          ; save index for line
        LDY     #$FF                    ; start from -1, adjust for pre increment
LAB_C72C
        DEX                             ; decrement token #
        BEQ     LAB_C737                ; if now found go do printing

LAB_C72F
        INY                             ; else increment index
        LDA     LAB_C09E,Y              ; get byte from keyword table
        BPL     LAB_C72F                ; loop until keyword end marker

        BMI     LAB_C72C                ; go test if this is required keyword, branch always

                                        ; found keyword, it's the next one
LAB_C737
```

```
        INY                             ; increment keyword table index
        LDA     LAB_C09E,Y              ; get byte from table
        BMI     LAB_C6EF                ; go restore index, mask byte and print if
                                        ; byte was end marker

        JSR     LAB_CB47                ; else go print the character
        BNE     LAB_C737                ; go get next character, branch always


;**************************************************************************************;
;
; perform FOR

LAB_C742
        LDA     #$80                    ; set FNX
        STA     LAB_10          ; set subscript/FNX flag
        JSR     LAB_C9A5                ; perform LET
        JSR     LAB_C38A                ; search the stack for FOR or GOSUB activity
        BNE     LAB_C753                ; branch if FOR, this variable, not found

                                        ; FOR, this variable, was found so first we dump the
old one
        TXA                             ; copy index
        ADC     #$0F                    ; add FOR structure size-2
        TAX                             ; copy to index
        TXS                             ; set stack (dump FOR structure (-2 bytes))
LAB_C753
        PLA                             ; pull return address
        PLA                             ; pull return address
        LDA     #$09                    ; we need 18d bytes !
        JSR     LAB_C3FB                ; check room on stack for 2*A bytes
        JSR     LAB_C906                ; scan for next BASIC statement ([:] or [EOL])
        CLC                             ; clear carry for add
        TYA                             ; copy index to A
        ADC     LAB_7A          ; add BASIC execute pointer low byte
        PHA                             ; push onto stack
        LDA     LAB_7B          ; get BASIC execute pointer high byte
        ADC     #$00                    ; add carry
        PHA                             ; push onto stack
        LDA     LAB_3A          ; get current line number high byte
        PHA                             ; push onto stack
        LDA     LAB_39          ; get current line number low byte
        PHA                             ; push onto stack
        LDA     #TK_TO          ; set "TO" token
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        JSR     LAB_CD8D                ; check if source is numeric, else do type mismatch
        JSR     LAB_CD8A                ; evaluate expression and check is numeric, else do
                                        ; type mismatch
        LDA     LAB_66          ; get FAC1 sign (b7)
        ORA     #$7F                    ; set all non sign bits
        AND     LAB_62          ; and FAC1 mantissa 1
        STA     LAB_62          ; save FAC1 mantissa 1
        LDA     #<LAB_C78B              ; set return address low byte
        LDY     #>LAB_C78B              ; set return address high byte
        STA     LAB_22          ; save return address low byte
        STY     LAB_23          ; save return address high byte
        JMP     LAB_CE43                ; round FAC1 and put on stack, returns to next
instruction

LAB_C78B
```

```
        LDA     #<LAB_D9BC              ; set 1 pointer low address, default step size
        LDY     #>LAB_D9BC              ; set 1 pointer high address
        JSR     LAB_DBA2                ; unpack memory (AY) into FAC1
        JSR     LAB_0079                ; scan memory
        CMP     #TK_STEP                ; compare with STEP token
        BNE     LAB_C79F                ; branch if not "STEP"

                                        ; was step so ....
        JSR     LAB_0073                ; increment and scan memory
        JSR     LAB_CD8A                ; evaluate expression and check is numeric, else do
                                        ; type mismatch
LAB_C79F
        JSR     LAB_DC2B                ; get FAC1 sign, return A = $FF -ve, A = $01 +ve
        JSR     LAB_CE38                ; push sign, round FAC1 and put on stack
        LDA     LAB_4A          ; get FOR/NEXT variable pointer high byte
        PHA                             ; push on stack
        LDA     LAB_49          ; get FOR/NEXT variable pointer low byte
        PHA                             ; push on stack
        LDA     #TK_FOR         ; get FOR token
        PHA                             ; push on stack


;******************************************************************************;
;
; interpreter inner loop

LAB_C7AE
        JSR     LAB_C82C                ; do CRTL-C check vector
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        LDY     LAB_7B          ; get BASIC execute pointer high byte
        CPY     #$02                    ; compare with $02xx
        NOP                             ; unused byte
##
        BEQ     LAB_C7BE                ; if immediate mode skip the continue pointer save

        STA     LAB_3D          ; save the continue pointer low byte
        STY     LAB_3E          ; save the continue pointer high byte
LAB_C7BE
        LDY     #$00                    ; clear the index
        LDA     (LAB_7A),Y              ; get BASIC byte
        BNE     LAB_C807                ; if not [EOL] go test for ":"

        LDY     #$02                    ; else set the index
        LDA     (LAB_7A),Y              ; get next line pointer high byte
        CLC                             ; clear carry for no "BREAK" message
        BNE     LAB_C7CE                ; branch if not end of program

        JMP     LAB_C84B                ; else go to immediate mode,was immediate or [EOT]
marker

LAB_C7CE
        INY                             ; increment index
        LDA     (LAB_7A),Y              ; get line number low byte
        STA     LAB_39          ; save current line number low byte
        INY                             ; increment index
        LDA     (LAB_7A),Y              ; get line # high byte
        STA     LAB_3A          ; save current line number high byte
        TYA                             ; A now = 4
        ADC     LAB_7A          ; add BASIC execute pointer low byte, now points to code
        STA     LAB_7A          ; save BASIC execute pointer low byte
        BCC     LAB_C7E1                ; if no overflow skip the high byte increment
```

```
        INC     LAB_7B          ; else increment BASIC execute pointer high byte
LAB_C7E1
        JMP     (LAB_0308)              ; do start new BASIC code


;******************************************************************************;
;
; start new BASIC code, the start new BASIC code vector is initialised to point here

LAB_C7E4
        JSR     LAB_0073                ; increment and scan memory
        JSR     LAB_C7ED                ; go interpret BASIC code from BASIC execute pointer
        JMP     LAB_C7AE                ; loop


;******************************************************************************;
;
; go interpret BASIC code from BASIC execute pointer

LAB_C7ED
        BEQ     LAB_C82B                ; if the first byte is null just exit

LAB_C7EF
        SBC     #$80                    ; normalise the token
        BCC     LAB_C804                ; if wasn't token go do LET

        CMP     #TK_TAB-$80             ; compare with token for TAB(-$80
        BCS     LAB_C80E                ; branch if >= TAB(

        ASL                             ; *2 bytes per vector
        TAY                             ; copy to index
        LDA     LAB_C00C+1,Y    ; get vector high byte
        PHA                             ; push on stack
        LDA     LAB_C00C,Y              ; get vector low byte
        PHA                             ; push on stack
        JMP     LAB_0073                ; increment and scan memory and return. the return in
                                        ; this case calls the command code, the return from
                                        ; that will eventually return to the interpreter inner
                                        ; loop above

LAB_C804
        JMP     LAB_C9A5                ; perform LET

                                        ; was not [EOL]
LAB_C807
        CMP     #':'                    ; comapre with ":"
        BEQ     LAB_C7E1                ; if ":" go execute new code

                                        ; else ...
LAB_C80B
        JMP     LAB_CF08                ; do syntax error then warm start

                                        ; token was >= TAB(
LAB_C80E
        CMP     #TK_GO-$80              ; compare with token for GO
        BNE     LAB_C80B                ; if not "GO" do syntax error then warm start

                                        ; else was "GO"
        JSR     LAB_0073                ; increment and scan memory
        LDA     #TK_TO          ; set "TO" token
```

```
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        JMP     LAB_C8A0                ; perform GOTO


;******************************************************************************;
;
; perform RESTORE

LAB_C81D
        SEC                             ; set carry for subtract
        LDA     LAB_2B          ; get start of memory low byte
        SBC     #$01                    ; -1
        LDY     LAB_2C          ; get start of memory high byte
        BCS     LAB_C827                ; if no rollunder skip the high byte decrement

        DEY                             ; else decrement high byte
LAB_C827
        STA     LAB_41          ; set DATA pointer low byte
        STY     LAB_42          ; set DATA pointer high byte
LAB_C82B
        RTS


;******************************************************************************;
;
; do CRTL-C check vector

LAB_C82C
        JSR     LAB_FFE1                ; scan stop key


;******************************************************************************;
;
; perform STOP

LAB_C82F
        BCS     LAB_C832                ; if carry set do BREAK instead of just END


;******************************************************************************;
;
; perform END

LAB_C831
        CLC                             ; clear carry
LAB_C832
        BNE     LAB_C870                ; return if wasn't CTRL-C

        LDA     LAB_7A          ; get BASIC execute pointer low byte
        LDY     LAB_7B          ; get BASIC execute pointer high byte
        LDX     LAB_3A          ; get current line number high byte
        INX                             ; increment it
        BEQ     LAB_C849                ; branch if was immediate mode

        STA     LAB_3D          ; save continue pointer low byte
        STY     LAB_3E          ; save continue pointer high byte
        LDA     LAB_39          ; get current line number low byte
        LDY     LAB_3A          ; get current line number high byte
        STA     LAB_3B          ; save break line number low byte
        STY     LAB_3C          ; save break line number high byte
```

```
LAB_C849
        PLA                             ; dump return address low byte
        PLA                             ; dump return address high byte
LAB_C84B
        LDA     #<LAB_C381              ; set [CR][LF]"BREAK" pointer low byte
        LDY     #>LAB_C381              ; set [CR][LF]"BREAK" pointer high byte
        BCC     LAB_C854                ; branch if was program end

        JMP     LAB_C469                ; print string and do warm start

LAB_C854
        JMP     LAB_C474                ; do warm start


;******************************************************************************;
;
; perform CONT

LAB_C857
        BNE     LAB_C870                ; exit if following byte to allow syntax error

        LDX     #$1A                    ; error code $1A, can't continue error
        LDY     LAB_3E          ; get continue pointer high byte
        BNE     LAB_C862                ; go do continue if we can

        JMP     LAB_C437                ; else do error #X then warm start

                                        ; we can continue so ...
LAB_C862
        LDA     LAB_3D          ; get continue pointer low byte
        STA     LAB_7A          ; save BASIC execute pointer low byte
        STY     LAB_7B          ; save BASIC execute pointer high byte
        LDA     LAB_3B          ; get break line low byte
        LDY     LAB_3C          ; get break line high byte
        STA     LAB_39          ; set current line number low byte
        STY     LAB_3A          ; set current line number high byte
LAB_C870
        RTS


;******************************************************************************;
;
; perform RUN

LAB_C871
        PHP                             ; save status
        LDA     #$00                    ; no control or kernal messages
        JSR     LAB_FF90                ; control kernal messages
        PLP                             ; restore status
        BNE     LAB_C87D                ; branch if RUN n

        JMP     LAB_C659                ; reset execution to start, clear variables, flush
stack
                                        ; and return
LAB_C87D
        JSR     LAB_C660                ; go do "CLEAR"
        JMP     LAB_C897                ; get n and do GOTO n


;******************************************************************************;
;
```

```
; perform GOSUB

LAB_C883
        LDA     #$03                    ; need 6 bytes for GOSUB
        JSR     LAB_C3FB                ; check room on stack for 2*A bytes
        LDA     LAB_7B          ; get BASIC execute pointer high byte
        PHA                             ; save it
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        PHA                             ; save it
        LDA     LAB_3A          ; get current line number high byte
        PHA                             ; save it
        LDA     LAB_39          ; get current line number low byte
        PHA                             ; save it
        LDA     #TK_GOSUB               ; token for GOSUB
        PHA                             ; save it
LAB_C897
        JSR     LAB_0079                ; scan memory
        JSR     LAB_C8A0                ; perform GOTO
        JMP     LAB_C7AE                ; go do interpreter inner loop


;******************************************************************************;
;
; perform GOTO

LAB_C8A0
        JSR     LAB_C96B                ; get fixed-point number into temporary integer
        JSR     LAB_C909                ; scan for next BASIC line
        SEC                             ; set carry for subtract
        LDA     LAB_39          ; get current line number low byte
        SBC     LAB_14          ; subtract temporary integer low byte
        LDA     LAB_3A          ; get current line number high byte
        SBC     LAB_15          ; subtract temporary integer high byte
        BCS     LAB_C8BC                ; if current line number >= temporary integer, go
search
                                        ; from the start of memory

        TYA                             ; else copy line index to A
        SEC                             ; set carry (+1)
        ADC     LAB_7A          ; add BASIC execute pointer low byte
        LDX     LAB_7B          ; get BASIC execute pointer high byte
        BCC     LAB_C8C0                ; if no overflow skip the high byte increment

        INX                             ; increment high byte
        BCS     LAB_C8C0                ; go find the line, branch always


;******************************************************************************;
;
; search for line number in temporary integer from start of memory pointer

LAB_C8BC
        LDA     LAB_2B          ; get start of memory low byte
        LDX     LAB_2C          ; get start of memory high byte

; search for line # in temporary integer from (AX)

LAB_C8C0
        JSR     LAB_C617                ; search Basic for temp integer line number from AX
        BCC     LAB_C8E3                ; if carry clear go do unsdefined statement error
```

```
                                        ; carry all ready set for subtract
        LDA     LAB_5F          ; get pointer low byte
        SBC     #$01                    ; -1
        STA     LAB_7A          ; save BASIC execute pointer low byte
        LDA     LAB_60          ; get pointer high byte
        SBC     #$00                    ; subtract carry
        STA     LAB_7B          ; save BASIC execute pointer high byte
LAB_C8D1
        RTS


;******************************************************************************;
;
; perform RETURN

LAB_C8D2
        BNE     LAB_C8D1                ; exit if following token to allow syntax error

        LDA     #$FF                    ; set byte so no match possible
        STA     LAB_4A          ; save FOR/NEXT variable pointer high byte
        JSR     LAB_C38A                ; search the stack for FOR or GOSUB activity,
                                        ; get token off stack
        TXS                             ; correct the stack
        CMP     #TK_GOSUB               ; compare with GOSUB token
        BEQ     LAB_C8EB                ; if matching GOSUB go continue RETURN

        LDX     #$0C                    ; else error code $04, return without gosub error
        .byte   $2C                     ; makes next line BIT LAB_11A2
LAB_C8E3
        LDX     #$11                    ; error code $11, undefined statement error
        JMP     LAB_C437                ; do error #X then warm start

LAB_C8E8
        JMP     LAB_CF08                ; do syntax error then warm start

                                        ; was matching GOSUB token
LAB_C8EB
        PLA                             ; dump token byte
        PLA                             ; pull return line low byte
        STA     LAB_39          ; save current line number low byte
        PLA                             ; pull return line high byte
        STA     LAB_3A          ; save current line number high byte
        PLA                             ; pull return address low byte
        STA     LAB_7A          ; save BASIC execute pointer low byte
        PLA                             ; pull return address high byte
        STA     LAB_7B          ; save BASIC execute pointer high byte


;******************************************************************************;
;
; perform DATA

LAB_C8F8
        JSR     LAB_C906                ; scan for next BASIC statement ([:] or [EOL])

; add Y to the BASIC execute pointer

LAB_C8FB
        TYA                             ; copy index to A
        CLC                             ; clear carry for add
        ADC     LAB_7A          ; add BASIC execute pointer low byte
```

```
        STA     LAB_7A          ; save BASIC execute pointer low byte
        BCC     LAB_C905                ; skip increment if no carry

        INC     LAB_7B          ; else increment BASIC execute pointer high byte
LAB_C905
        RTS


;******************************************************************************;
;
; scan for next BASIC statement ([:] or [EOL])
; returns Y as index to [:] or [EOL]

LAB_C906
        LDX     #':'                    ; set look for character = ":"
        .byte   $2C                     ; makes next line BIT LAB_00A2

; scan for next BASIC line
; returns Y as index to [EOL]

LAB_C909
        LDX     #$00                    ; set alternate search character = [EOL]
        STX     LAB_07          ; store alternate search character
        LDY     #$00                    ; set search character = [EOL]
        STY     LAB_08          ; save the search character
LAB_C911
        LDA     LAB_08          ; get search character
        LDX     LAB_07          ; get alternate search character
        STA     LAB_07          ; make search character = alternate search character
        STX     LAB_08          ; make alternate search character = search character
LAB_C919
        LDA     (LAB_7A),Y              ; get BASIC byte
        BEQ     LAB_C905                ; exit if null [EOL]

        CMP     LAB_08          ; compare with search character
        BEQ     LAB_C905                ; exit if found

        INY                             ; else increment index
        CMP     #$22                    ; compare current character with open quote
        BNE     LAB_C919                ; if found go swap search character for alternate
search
                                        ; character

        BEQ     LAB_C911                ; loop for next character, branch always


;******************************************************************************;
;
; perform IF

LAB_C928
        JSR     LAB_CD9E                ; evaluate expression
        JSR     LAB_0079                ; scan memory
        CMP     #TK_GOTO                ; compare with "GOTO" token
        BEQ     LAB_C937                ; if it was  the token for GOTO go do IF ... GOTO

                                        ; wasn't IF ... GOTO so must be IF ... THEN
        LDA     #TK_THEN                ; $A7 = "THEN" token
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
LAB_C937
```

```
        LDA     LAB_61          ; get FAC1 exponent
        BNE     LAB_C940                ; if result was non zero continue execution
                                        ; else REM the rest of the line


;******************************************************************************;
;
; perform REM

LAB_C93B
        JSR     LAB_C909                ; scan for next BASIC line
        BEQ     LAB_C8FB                ; add Y to the BASIC execute pointer and return,
branch
                                        ; always


;******************************************************************************;
;
; IF continued .. result was non zero so do rest of line

LAB_C940
        JSR     LAB_0079                ; scan memory
        BCS     LAB_C948                ; if not numeric character, is variable or keyword

        JMP     LAB_C8A0                ; else perform GOTO n

                                        ; is variable or keyword
LAB_C948
        JMP     LAB_C7ED                ; interpret BASIC code from BASIC execute pointer


;******************************************************************************;
;
; perform ON

LAB_C94B
        JSR     LAB_D79E                ; get byte parameter
        PHA                             ; push next character
        CMP     #TK_GOSUB               ; compare with GOSUB token
        BEQ     LAB_C957                ; if GOSUB go see if it should be executed

LAB_C953
        CMP     #TK_GOTO                ; compare with GOTO token
        BNE     LAB_C8E8                ; if not GOTO do syntax error then warm start

; next character was GOTO or GOSUB, see if it should be executed

LAB_C957
        DEC     LAB_65          ; decrement the byte value
        BNE     LAB_C95F                ; if not zero go see if another line number exists

        PLA                             ; pull keyword token
        JMP     LAB_C7EF                ; go execute it

LAB_C95F
        JSR     LAB_0073                ; increment and scan memory
        JSR     LAB_C96B                ; get fixed-point number into temporary integer
                                        ; skip this n
        CMP     #','                    ; compare next character with ","
        BEQ     LAB_C957                ; loop if ","
```

```
        PLA                             ; else pull keyword token, ran out of options
LAB_C96A
        RTS


;******************************************************************************;
;
; get fixed-point number into temporary integer

LAB_C96B
        LDX     #$00                    ; clear X
        STX     LAB_14          ; clear temporary integer low byte
        STX     LAB_15          ; clear temporary integer high byte
LAB_C971
        BCS     LAB_C96A                ; return if carry set, end of scan, character was not
0-9

        SBC     #$2F                    ; subtract $30, $2F+carry, from byte
        STA     LAB_07          ; store #
        LDA     LAB_15          ; get temporary integer high byte
        STA     LAB_22          ; save it for now
        CMP     #$19                    ; compare with $19
        BCS     LAB_C953                ; branch if >= this makes the maximum line number
63999
                                        ; because the next bit does $1900 * $0A = $FA00 =
64000
                                        ; decimal. the branch target is really the SYNTAX
error
                                        ; at LAB_C8E8 but that is too far so an intermediate
                                        ; compare and branch to that location is used. the
problem
                                        ; with this is that line number that gives a partial
result
                                        ; from $8900 to $89FF, 35072x to 35327x, will pass the
new
                                        ; target compare and will try to execute the remainder
of
                                        ; the ON n GOTO/GOSUB. a solution to this is to copy
the
                                        ; byte in A before the branch to X and then branch to
                                        ; LAB_C955 skipping the second compare

        LDA     LAB_14          ; get temporary integer low byte
        ASL                             ; *2 low byte
        ROL     LAB_22          ; *2 high byte
        ASL                             ; *2 low byte
        ROL     LAB_22          ; *2 high byte (*4)
        ADC     LAB_14          ; + low byte (*5)
        STA     LAB_14          ; save it
        LDA     LAB_22          ; get high byte temp
        ADC     LAB_15          ; + high byte (*5)
        STA     LAB_15          ; save it
        ASL     LAB_14          ; *2 low byte (*10d)
        ROL     LAB_15          ; *2 high byte (*10d)
        LDA     LAB_14          ; get low byte
        ADC     LAB_07          ; add #
        STA     LAB_14          ; save low byte
        BCC     LAB_C99F                ; if no overflow skip high byte increment

        INC     LAB_15          ; else increment high byte
LAB_C99F
```

```
        JSR     LAB_0073                ; increment and scan memory
        JMP     LAB_C971                ; loop for next character


;******************************************************************************;
;
; perform LET

LAB_C9A5
        JSR     LAB_D08B                ; get variable address
        STA     LAB_49          ; save variable address low byte
        STY     LAB_4A          ; save variable address high byte
        LDA     #TK_EQUAL               ; $B2 is "=" token
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        LDA     LAB_0E          ; get data type flag, $80 = integer, $00 = float
        PHA                             ; push data type flag
        LDA     LAB_0D          ; get data type flag, $FF = string, $00 = numeric
        PHA                             ; push data type flag
        JSR     LAB_CD9E                ; evaluate expression
        PLA                             ; pop data type flag
        ROL                             ; string bit into carry
        JSR     LAB_CD90                ; do type match check
        BNE     LAB_C9D9                ; if string go assign a string value

        PLA                             ; pop integer/float data type flag

; assign value to numeric variable

LAB_C9C2
        BPL     LAB_C9D6                ; if float go assign a floating value

                                        ; expression is numeric integer
        JSR     LAB_DC1B                ; round FAC1
        JSR     LAB_D1BF                ; evaluate integer expression, no sign check
        LDY     #$00                    ; clear index
        LDA     LAB_64          ; get FAC1 mantissa 3
        STA     (LAB_49),Y              ; save as integer variable low byte
        INY                             ; increment index
        LDA     LAB_65          ; get FAC1 mantissa 4
        STA     (LAB_49),Y              ; save as integer variable high byte
        RTS

LAB_C9D6
        JMP     LAB_DBD0                ; pack FAC1 into variable pointer and return

; assign value to string variable

LAB_C9D9
        PLA                             ; dump integer/float data type flag
LAB_C9DA
        LDY     LAB_4A          ; get variable pointer high byte
        CPY     #>LAB_DF13              ; was it TI$ pointer
        BNE     LAB_CA2C                ; branch if not

                                        ; else it's TI$ = <expr$>
        JSR     LAB_D6A6                ; pop string off descriptor stack, or from top of
string
                                        ; space returns with A = length, X = pointer low byte,
                                        ; Y = pointer high byte
        CMP     #$06                    ; compare length with 6
```

```
        BNE     LAB_CA24                ; if length not 6 do illegal quantity error then warm
start

        LDY     #$00                    ; clear index
        STY     LAB_61          ; clear FAC1 exponent
        STY     LAB_66          ; clear FAC1 sign (b7)
LAB_C9ED
        STY     LAB_71          ; save index
        JSR     LAB_CA1D                ; check and evaluate numeric digit
        JSR     LAB_DAE2                ; multiply FAC1 by 10
        INC     LAB_71          ; increment index
        LDY     LAB_71          ; restore index
        JSR     LAB_CA1D                ; check and evaluate numeric digit
        JSR     LAB_DC0C                ; round and copy FAC1 to FAC2
        TAX                             ; copy FAC1 exponent
        BEQ     LAB_CA07                ; branch if FAC1 zero

        INX                             ; increment index, * 2
        TXA                             ; copy back to A
        JSR     LAB_DAED                ; FAC1 = (FAC1 + (FAC2 * 2)) * 2 = FAC1 * 6
LAB_CA07
        LDY     LAB_71          ; get index
        INY                             ; increment index
        CPY     #$06                    ; compare index with 6
        BNE     LAB_C9ED                ; loop if not 6

        JSR     LAB_DAE2                ; multiply FAC1 by 10
        JSR     LAB_DC9B                ; convert FAC1 floating to fixed
        LDX     LAB_64          ; get FAC1 mantissa 3
        LDY     LAB_63          ; get FAC1 mantissa 2
        LDA     LAB_65          ; get FAC1 mantissa 4
        JMP     LAB_FFDB                ; set real time clock and return

; check and evaluate numeric digit

LAB_CA1D
        LDA     (LAB_22),Y              ; get byte from string
        JSR     LAB_80          ; clear Cb if numeric. this call should be to LAB_84
                                        ; as the code from LAB_80 first comapres the byte with
                                        ; [SPACE] and does a BASIC increment and get if it is
        BCC     LAB_CA27                ; branch if numeric

LAB_CA24
        JMP     LAB_D248                ; do illegal quantity error then warm start

LAB_CA27
        SBC     #$2F                    ; subtract $2F + carry to convert ASCII to binary
        JMP     LAB_DD7E                ; evaluate new ASCII digit and return

; assign value to string variable, but not TI$

LAB_CA2C
        LDY     #$02                    ; index to string pointer high byte
        LDA     (LAB_64),Y              ; get string pointer high byte
        CMP     LAB_34          ; compare with bottom of string space high byte
        BCC     LAB_CA4B                ; branch if string pointer high byte is less than
bottom
                                        ; of string space high byte

        BNE     LAB_CA3D                ; branch if string pointer high byte is greater than
                                        ; bottom of string space high byte
```

```
                                        ; else high bytes were equal
        DEY                             ; decrement index to string pointer low byte
        LDA     (LAB_64),Y              ; get string pointer low byte
        CMP     LAB_33          ; compare with bottom of string space low byte
        BCC     LAB_CA4B                ; branch if string pointer low byte is less than
bottom
                                        ; of string space low byte

LAB_CA3D
        LDY     LAB_65          ; get descriptor pointer high byte
        CPY     LAB_2E          ; compare with start of variables high byte
        BCC     LAB_CA4B                ; branch if less, is on string stack

        BNE     LAB_CA52                ; if greater make space and copy string

                                        ; else high bytes were equal
        LDA     LAB_64          ; get descriptor pointer low byte
        CMP     LAB_2D          ; compare with start of variables low byte
        BCS     LAB_CA52                ; if greater or equal make space and copy string

LAB_CA4B
        LDA     LAB_64          ; get descriptor pointer low byte
        LDY     LAB_65          ; get descriptor pointer high byte
        JMP     LAB_CA68                ; go copy descriptor to variable

LAB_CA52
        LDY     #$00                    ; clear index
        LDA     (LAB_64),Y              ; get string length
        JSR     LAB_D475                ; copy descriptor pointer and make string space A
bytes long
        LDA     LAB_50          ; copy old descriptor pointer low byte
        LDY     LAB_51          ; copy old descriptor pointer high byte
        STA     LAB_6F          ; save old descriptor pointer low byte
        STY     LAB_70          ; save old descriptor pointer high byte
        JSR     LAB_D67A                ; copy string from descriptor to utility pointer
        LDA     #<LAB_61                ; get descriptor pointer low byte
        LDY     #>LAB_61                ; get descriptor pointer high byte
LAB_CA68
        STA     LAB_50          ; save descriptor pointer low byte
        STY     LAB_51          ; save descriptor pointer high byte
        JSR     LAB_D6DB                ; clean descriptor stack, YA = pointer
        LDY     #$00                    ; clear index
        LDA     (LAB_50),Y              ; get string length from new descriptor
        STA     (LAB_49),Y              ; copy string length to variable
        INY                             ; increment index
        LDA     (LAB_50),Y              ; get string pointer low byte from new descriptor
        STA     (LAB_49),Y              ; copy string pointer low byte to variable
        INY                             ; increment index
        LDA     (LAB_50),Y              ; get string pointer high byte from new descriptor
        STA     (LAB_49),Y              ; copy string pointer high byte to variable
        RTS


;******************************************************************************;
;
; perform PRINT#

LAB_CA80
        JSR     LAB_CA86                ; perform CMD
        JMP     LAB_CBB5                ; close input and output channels and return
```

```
;**************************************************************************;
;
; perform CMD

LAB_CA86
        JSR     LAB_D79E                ; get byte parameter
        BEQ     LAB_CA90                ; branch if following byte is ":" or [EOT]

        LDA     #','                    ; set ","
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
LAB_CA90
        PHP                             ; save status
        STX     LAB_13          ; set current I/O channel
        JSR     LAB_E115                ; open channel for output with error check
        PLP                             ; restore status
        JMP     LAB_CAA0                ; perform PRINT


;**************************************************************************;
;
; print string, scan memory and continue PRINT

LAB_CA9A
        JSR     LAB_CB21                ; print string from utility pointer

; scan memory and continue PRINT

LAB_CA9D
        JSR     LAB_0079                ; scan memory


;**************************************************************************;
;
; perform PRINT

LAB_CAA0
        BEQ     LAB_CAD7                ; if nothing following just print CR/LF

LAB_CAA2
        BEQ     LAB_CAE7                ; if nothing following exit, end of PRINT branch

        CMP     #TK_TAB         ; compare with token for TAB(
        BEQ     LAB_CAF8                ; if TAB( go handle it

        CMP     #TK_SPC         ; compare with token for SPC(
        CLC                             ; flag SPC(
        BEQ     LAB_CAF8                ; if SPC( go handle it

        CMP     #','                    ; compare with ","
        BEQ     LAB_CAE8                ; if "," go skip to the next TAB position

        CMP     #$3B                    ; compare with ";"
        BEQ     LAB_CB13                ; if ";" go continue the print loop

        JSR     LAB_CD9E                ; evaluate expression
        BIT     LAB_0D          ; test data type flag, $FF = string, $00 = numeric
        BMI     LAB_CA9A                ; if string go print string, scan memory and continue
PRINT
```

```
        JSR     LAB_DDDD                ; convert FAC1 to ASCII string result in (AY)
        JSR     LAB_D487                ; print " terminated string to utility pointer
        JSR     LAB_CB21                ; print string from utility pointer
        JSR     LAB_CB3B                ; print [SPACE] or [CURSOR RIGHT]
        BNE     LAB_CA9D                ; go scan memory and continue PRINT, branch always


;*******************************************************************************;
;
; set XY to LAB_0200 - 1 and print [CR]

LAB_CACA
        LDA     #$00                    ; clear A
        STA     LAB_0200,X              ; clear first byte of input buffer
        LDX     #<LAB_01FF              ; LAB_0200 - 1 low byte
        LDY     #>LAB_01FF              ; LAB_0200 - 1 high byte
        LDA     LAB_13          ; get current I/O channel
        BNE     LAB_CAE7                ; exit if not default channel


;*******************************************************************************;
;
; print CR/LF

LAB_CAD7
        LDA     #$0D                    ; set [CR]
        JSR     LAB_CB47                ; print the character
        BIT     LAB_13          ; test current I/O channel
        BPL     LAB_CAE5                ; if the AutoLF bit is not set skip the LF

; it would seem from other parts of the code that using b7 as an AutoLF flag bit is
; no longer supported and setting this bit would break things in a lot of places

        LDA     #$0A                    ; set [LF]
        JSR     LAB_CB47                ; print the character


;*******************************************************************************;
;
; toggle A

LAB_CAE5
        EOR     #$FF                    ; invert A
LAB_CAE7
        RTS


;*******************************************************************************;
;
; continuing PRINT, the character was ","

LAB_CAE8
        SEC                             ; set Cb for read cursor position
        JSR     LAB_FFF0                ; read/set X,Y cursor position
        TYA                             ; copy cursor Y
        SEC                             ; set carry for subtract
LAB_CAEE
        SBC     #$0B                    ; subtract one TAB length
        BCS     LAB_CAEE                ; loop if result was +ve
```

```
        EOR     #$FF                    ; complement it
        ADC     #$01                    ; +1, twos complement
        BNE     LAB_CB0E                ; print A spaces, branch always, result is never $00


;******************************************************************************;
;
; handle TAB( or SPC(

LAB_CAF8
        PHP                             ; save TAB( or SPC( status
        SEC                             ; set Cb for read cursor position
        JSR     LAB_FFF0                ; read/set X,Y cursor position
        STY     LAB_09          ; save current cursor position
        JSR     LAB_D79B                ; scan and get byte parameter
        CMP     #$29                    ; compare with ")"
        BNE     LAB_CB5F                ; if not ")" do syntax error

        PLP                             ; restore TAB( or SPC( status
        BCC     LAB_CB0F                ; branch if was SPC(

                                        ; else was TAB(
        TXA                             ; copy TAB() byte to A
        SBC     LAB_09          ; subtract current cursor position
        BCC     LAB_CB13                ; go loop for next if already past requited position

LAB_CB0E
        TAX                             ; copy [SPACE] count to X
LAB_CB0F
        INX                             ; increment count
LAB_CB10
        DEX                             ; decrement count
        BNE     LAB_CB19                ; branch if count was not zero

                                        ; was ";" or [SPACES] printed
LAB_CB13
        JSR     LAB_0073                ; increment and scan memory
        JMP     LAB_CAA2                ; continue print loop

LAB_CB19
        JSR     LAB_CB3B                ; print [SPACE] or [CURSOR RIGHT]
        BNE     LAB_CB10                ; loop, branch always


;******************************************************************************;
;
; print null terminated string

LAB_CB1E
        JSR     LAB_D487                ; print " terminated string to utility pointer

; print string from utility pointer

LAB_CB21
        JSR     LAB_D6A6                ; pop string off descriptor stack, or from top of
string
                                        ; space returns with A = length, X = pointer low byte,
                                        ; Y = pointer high byte
        TAX                             ; copy length
        LDY     #$00                    ; clear index
        INX                             ; increment length, for pre decrement loop
```

```
LAB_CB28
        DEX                             ; decrement length
        BEQ     LAB_CAE7                ; exit if done

        LDA     (LAB_22),Y              ; get byte from string
        JSR     LAB_CB47                ; print the character
        INY                             ; increment index
        CMP     #$0D                    ; compare byte with [CR]
        BNE     LAB_CB28                ; loop if not [CR]

        JSR     LAB_CAE5                ; toggle A, EOR #$FF. what is the point of this ??
        JMP     LAB_CB28                ; loop


;*******************************************************************************;
;
; print [SPACE] or [CURSOR RIGHT]

LAB_CB3B
        LDA     LAB_13          ; get current I/O channel
        BEQ     LAB_CB42                ; if default channel go output [CURSOR RIGHT]

        LDA     #' '                    ; else output [SPACE]
        .byte   $2C                     ; makes next line BIT LAB_1DA9
LAB_CB42
        LDA     #$1D                    ; set [CURSOR RIGHT]
        .byte   $2C                     ; makes next line BIT LAB_3FA9


;*******************************************************************************;
;
; print "?"

LAB_CB45
        LDA     #'?'                    ; set "?"


;*******************************************************************************;
;
; print a character

LAB_CB47
        JSR     LAB_E109                ; output character to channel with error check
        AND     #$FF                    ; set the flags on A
        RTS


;*******************************************************************************;
;
; bad input routine

LAB_CB4D
        LDA     LAB_11          ; get INPUT mode flag, $00 = INPUT, $40 = GET, $98 = READ
        BEQ     LAB_CB62                ; branch if INPUT

        BMI     LAB_CB57                ; branch if READ

                                        ; else was GET
        LDY     #$FF                    ; set current line high byte to -1, indicate immediate
mode
        BNE     LAB_CB5B                ; branch always
```

```
LAB_CB57
        LDA     LAB_3F          ; get current DATA line number low byte
        LDY     LAB_40          ; get current DATA line number high byte
LAB_CB5B
        STA     LAB_39          ; set current line number low byte
        STY     LAB_3A          ; set current line number high byte
LAB_CB5F
        JMP     LAB_CF08                ; do syntax error then warm start

                                        ; was INPUT
LAB_CB62
        LDA     LAB_13          ; get current I/O channel
        BEQ     LAB_CB6B                ; if default channel go do "?REDO FROM START" message

        LDX     #$18                    ; else error $18, file data error
        JMP     LAB_C437                ; do error #X then warm start

LAB_CB6B
        LDA     #<LAB_CD0C              ; set "?REDO FROM START" pointer low byte
        LDY     #>LAB_CD0C              ; set "?REDO FROM START" pointer high byte
        JSR     LAB_CB1E                ; print null terminated string
        LDA     LAB_3D          ; get continue pointer low byte
        LDY     LAB_3E          ; get continue pointer high byte
        STA     LAB_7A          ; save BASIC execute pointer low byte
        STY     LAB_7B          ; save BASIC execute pointer high byte
        RTS


;**************************************************************************;
;
; perform GET

LAB_CB7B
        JSR     LAB_D3A6                ; check not Direct, back here if ok
        CMP     #'#'                    ; compare with "#"
        BNE     LAB_CB92                ; branch if not GET#

        JSR     LAB_0073                ; increment and scan memory
        JSR     LAB_D79E                ; get byte parameter
        LDA     #','                    ; set ","
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        STX     LAB_13          ; set current I/O channel
        JSR     LAB_E11B                ; open channel for input with error check
LAB_CB92
        LDX     #<LAB_0201              ; set LAB_0200+1 pointer low byte
        LDY     #>LAB_0201              ; set LAB_0200+1 pointer high byte
        LDA     #$00                    ; clear A
        STA     LAB_0200+1              ; ensure null terminator
        LDA     #$40                    ; input mode = GET
        JSR     LAB_CC0F                ; perform GET part of READ
        LDX     LAB_13          ; get current I/O channel
        BNE     LAB_CBB7                ; if not default channel go do channel close and
return

        RTS


;**************************************************************************;
;
```

```
; perform INPUT#

LAB_CBA5
        JSR     LAB_D79E                ; get byte parameter
        LDA     #','                    ; set ","
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        STX     LAB_13          ; set current I/O channel
        JSR     LAB_E11B                ; open channel for input with error check
        JSR     LAB_CBCE                ; perform INPUT with no prompt string

; close input and output channels

LAB_CBB5
        LDA     LAB_13          ; get current I/O channel
LAB_CBB7
        JSR     LAB_FFCC                ; close input and output channels
        LDX     #$00                    ; clear X
        STX     LAB_13          ; clear current I/O channel, flag default
        RTS


;******************************************************************************;
;
; perform INPUT

LAB_CBBF
        CMP     #$22                    ; compare next byte with open quote
        BNE     LAB_CBCE                ; if no prompt string just do INPUT

        JSR     LAB_CEBD                ; print "..." string
        LDA     #$3B                    ; load A with ";"
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        JSR     LAB_CB21                ; print string from utility pointer

                                        ; done with prompt, now get data
LAB_CBCE
        JSR     LAB_D3A6                ; check not Direct, back here if ok
        LDA     #','                    ; set ","
        STA     LAB_0200-1              ; save to start of buffer - 1
LAB_CBD6
        JSR     LAB_CBF9                ; print "? " and get BASIC input
        LDA     LAB_13          ; get current I/O channel
        BEQ     LAB_CBEA                ; branch if default I/O channel

        JSR     LAB_FFB7                ; read I/O status word
        AND     #$02                    ; mask no DSR/timeout
        BEQ     LAB_CBEA                ; branch if not error

        JSR     LAB_CBB5                ; close input and output channels
        JMP     LAB_C8F8                ; perform DATA

LAB_CBEA
        LDA     LAB_0200                ; get first byte in input buffer
        BNE     LAB_CC0D                ; branch if not null

                                        ; else ..
        LDA     LAB_13          ; get current I/O channel
        BNE     LAB_CBD6                ; if not default channel go get BASIC input
```

```
        JSR     LAB_C906                ; scan for next BASIC statement ([:] or [EOL])
        JMP     LAB_C8FB                ; add Y to the BASIC execute pointer and return


;******************************************************************************;
;
; print "? " and get BASIC input

LAB_CBF9
        LDA     LAB_13          ; get current I/O channel
        BNE     LAB_CC03                ; skip "?" prompt if not default channel

        JSR     LAB_CB45                ; print "?"
        JSR     LAB_CB3B                ; print [SPACE] or [CURSOR RIGHT]
LAB_CC03
        JMP     LAB_C560                ; call for BASIC input and return


;******************************************************************************;
;
; perform READ

LAB_CC06
        LDX     LAB_41          ; get DATA pointer low byte
        LDY     LAB_42          ; get DATA pointer high byte
        LDA     #$98                    ; set input mode = READ
        .byte   $2C                     ; makes next line BIT LAB_00A9
LAB_CC0D
        LDA     #$00                    ; set input mode = INPUT


;******************************************************************************;
;
; perform GET

LAB_CC0F
        STA     LAB_11          ; set input mode flag, $00 = INPUT, $40 = GET, $98 = READ
        STX     LAB_43          ; save READ pointer low byte
        STY     LAB_44          ; save READ pointer high byte

                                        ; READ, GET or INPUT next variable from list
LAB_CC15
        JSR     LAB_D08B                ; get variable address
        STA     LAB_49          ; save address low byte
        STY     LAB_4A          ; save address high byte
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        LDY     LAB_7B          ; get BASIC execute pointer high byte
        STA     LAB_4B          ; save BASIC execute pointer low byte
        STY     LAB_4C          ; save BASIC execute pointer high byte
        LDX     LAB_43          ; get READ pointer low byte
        LDY     LAB_44          ; get READ pointer high byte
        STX     LAB_7A          ; save as BASIC execute pointer low byte
        STY     LAB_7B          ; save as BASIC execute pointer high byte
        JSR     LAB_0079                ; scan memory
        BNE     LAB_CC51                ; branch if not null

                                        ; pointer was to null entry
        BIT     LAB_11          ; test input mode flag, $00 = INPUT, $40 = GET, $98 = READ
        BVC     LAB_CC41                ; branch if not GET

                                        ; else was GET
```

```
        JSR     LAB_E121                ; get character from input device with error check
        STA     LAB_0200                ; save to buffer
        LDX     #<LAB_01FF              ; set LAB_0200-1 pointer low byte
        LDY     #>LAB_01FF              ; set LAB_0200-1 pointer high byte
        BNE     LAB_CC4D                ; go interpret single character

LAB_CC41
        BMI     LAB_CCB8                ; if READ go get some DATA

; else it was INPUT

        LDA     LAB_13          ; get current I/O channel
        BNE     LAB_CC4A                ; skip "?" prompt if not default channel

        JSR     LAB_CB45                ; print "?"
LAB_CC4A
        JSR     LAB_CBF9                ; print "? " and get BASIC input
LAB_CC4D
        STX     LAB_7A          ; save BASIC execute pointer low byte
        STY     LAB_7B          ; save BASIC execute pointer high byte
LAB_CC51
        JSR     LAB_0073                ; increment and scan memory, execute pointer now
points to
                                ; start of next data or null terminator
        BIT     LAB_0D          ; test data type flag, $FF = string, $00 = numeric
        BPL     LAB_CC89                ; branch if numeric

                                ; type is string
        BIT     LAB_11          ; test INPUT mode flag, $00 = INPUT, $40 = GET, $98 = READ
        BVC     LAB_CC65                ; branch if not GET

                                ; else do string GET
        INX                             ; clear X ??
        STX     LAB_7A          ; save BASIC execute pointer low byte
        LDA     #$00                    ; clear A
        STA     LAB_07          ; clear search character
        BEQ     LAB_CC71                ; branch always

                                ; is string INPUT or string READ
LAB_CC65
        STA     LAB_07          ; save search character
        CMP     #$22                    ; compare with "
        BEQ     LAB_CC72                ; if quote only search for "..." string

                                ; else the string is not in quotes so ":", "," or $00
are
                                ; the termination characters
        LDA     #':'                    ; set ":"
        STA     LAB_07          ; set search character
        LDA     #','                    ; set ","
LAB_CC71
        CLC                             ; clear carry for add
LAB_CC72
        STA     LAB_08          ; set scan quotes flag
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        LDY     LAB_7B          ; get BASIC execute pointer high byte
        ADC     #$00                    ; add to pointer low byte. this add increments the
pointer
                                ; if the mode is INPUT or READ and the data is a "..."
                                ; string
        BCC     LAB_CC7D                ; if no rollover skip the high byte increment
```

```
        INY                             ; else increment pointer high byte
LAB_CC7D
        JSR     LAB_D48D                ; print string to utility pointer
        JSR     LAB_D7E2                ; restore BASIC execute pointer from temp
        JSR     LAB_C9DA                ; perform string LET
        JMP     LAB_CC91                ; continue processing command

                                        ; GET, INPUT or READ is numeric
LAB_CC89
        JSR     LAB_DCF3                ; get FAC1 from string
        LDA     LAB_0E          ; get data type flag, $80 = integer, $00 = float
        JSR     LAB_C9C2                ; assign value to numeric variable
LAB_CC91
        JSR     LAB_0079                ; scan memory
        BEQ     LAB_CC9D                ; if ":" or [EOL] go handle the string end

        CMP     #','                    ; comparte with ","
        BEQ     LAB_CC9D                ; if "," go handle the string end

        JMP     LAB_CB4D                ; else go do bad input routine

                                        ; string terminated with ":", "," or $00
LAB_CC9D
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        LDY     LAB_7B          ; get BASIC execute pointer high byte
        STA     LAB_43          ; save READ pointer low byte
        STY     LAB_44          ; save READ pointer high byte
        LDA     LAB_4B          ; get saved BASIC execute pointer low byte
        LDY     LAB_4C          ; get saved BASIC execute pointer high byte
        STA     LAB_7A          ; restore BASIC execute pointer low byte
        STY     LAB_7B          ; restore BASIC execute pointer high byte
        JSR     LAB_0079                ; scan memory
        BEQ     LAB_CCDF                ; branch if ":" or [EOL]

        JSR     LAB_CEFD                ; scan for ",", else do syntax error then warm start
        JMP     LAB_CC15                ; go READ or INPUT next variable from list

                                        ; was READ
LAB_CCB8
        JSR     LAB_C906                ; scan for next BASIC statement ([:] or [EOL])
        INY                             ; increment index to next byte
        TAX                             ; copy byte to X
        BNE     LAB_CCD1                ; if ":" go look for the next DATA

        LDX     #$0D                    ; else set error $0D, out of data error
        INY                             ; increment index to next line pointer high byte
        LDA     (LAB_7A),Y              ; get next line pointer high byte
        BEQ     LAB_CD32                ; if program end go do error, eventually does error X

        INY                             ; increment index
        LDA     (LAB_7A),Y              ; get next line # low byte
        STA     LAB_3F          ; save current DATA line low byte
        INY                             ; increment index
        LDA     (LAB_7A),Y              ; get next line # high byte
        INY                             ; increment index
        STA     LAB_40          ; save current DATA line high byte
LAB_CCD1
        JSR     LAB_C8FB                ; add Y to the BASIC execute pointer
        JSR     LAB_0079                ; scan memory
        TAX                             ; copy byte
```

```
        CPX     #TK_DATA                ; compare with token for DATA
        BNE     LAB_CCB8                ; loop if not DATA

        JMP     LAB_CC51                ; continue evaluating READ

LAB_CCDF
        LDA     LAB_43          ; get READ pointer low byte
        LDY     LAB_44          ; get READ pointer high byte
        LDX     LAB_11          ; get INPUT mode flag, $00 = INPUT, $40 = GET, $98 = READ
        BPL     LAB_CCEA                ; if INPUT or GET go exit or ignore extra input

        JMP     LAB_C827                ; else set data pointer and exit

LAB_CCEA
        LDY     #$00                    ; clear index
        LDA     (LAB_43),Y              ; get READ byte
        BEQ     LAB_CCFB                ; exit if [EOL]

        LDA     LAB_13          ; get current I/O channel
        BNE     LAB_CCFB                ; exit if not default channel

        LDA     #<LAB_CCFC              ; set "?EXTRA IGNORED" pointer low byte
        LDY     #>LAB_CCFC              ; set "?EXTRA IGNORED" pointer high byte
        JMP     LAB_CB1E                ; print null terminated string

LAB_CCFB
        RTS


;**************************************************************************************;
;
; input error messages

LAB_CCFC
        .byte   "?EXTRA IGNORED",$0D,$00

LAB_CD0C
        .byte   "?REDO FROM START",$0D,$00


;**************************************************************************************;
;
; perform NEXT

LAB_CD1E
        BNE     LAB_CD24                ; if NEXT variable go find the variable

        LDY     #$00                    ; else clear Y
        BEQ     LAB_CD27                ; use any variable, branch always

; NEXT variable

LAB_CD24
        JSR     LAB_D08B                ; get variable address
LAB_CD27
        STA     LAB_49          ; save FOR/NEXT variable pointer low byte
        STY     LAB_4A          ; save FOR/NEXT variable pointer high byte
                                ; (high byte cleared if no variable defined)
        JSR     LAB_C38A                ; search the stack for FOR or GOSUB activity
        BEQ     LAB_CD35                ; if FOR found continue
```

```
        LDX     #$0A                    ; else set error $0A, next without for error
LAB_CD32
        JMP     LAB_C437                ; do error #X then warm start

; found this FOR variable

LAB_CD35
        TXS                             ; update stack pointer
        TXA                             ; copy stack pointer
        CLC                             ; clear carry for add
        ADC     #$04                    ; point to STEP value
        PHA                             ; save it
        ADC     #$06                    ; point to TO value
        STA     LAB_24          ; save pointer to TO variable for compare
        PLA                             ; restore pointer to STEP value
        LDY     #$01                    ; point to stack page
        JSR     LAB_DBA2                ; unpack memory (AY) into FAC1
        TSX                             ; get stack pointer back
        LDA     LAB_0100+9,X    ; get step sign
        STA     LAB_66          ; save FAC1 sign (b7)
        LDA     LAB_49          ; get FOR/NEXT variable pointer low byte
        LDY     LAB_4A          ; get FOR/NEXT variable pointer high byte
        JSR     LAB_D867                ; add FOR variable to FAC1
        JSR     LAB_DBD0                ; pack FAC1 into FOR variable
        LDY     #$01                    ; point to stack page
        JSR     LAB_DC5D                ; compare FAC1 with TO value
        TSX                             ; get stack pointer back
        SEC                             ; set carry for subtract
        SBC     LAB_0100+9,X    ; subtract step sign
        BEQ     LAB_CD78                ; if = loop complete, go unstack the FOR

                                        ; loop back and do it all again
        LDA     LAB_0100+$0F,X  ; get FOR line low byte
        STA     LAB_39          ; save current line number low byte
        LDA     LAB_0100+$10,X  ; get FOR line high byte
        STA     LAB_3A          ; save current line number high byte
        LDA     LAB_0100+$12,X  ; get BASIC execute pointer low byte
        STA     LAB_7A          ; save BASIC execute pointer low byte
        LDA     LAB_0100+$11,X  ; get BASIC execute pointer high byte
        STA     LAB_7B          ; save BASIC execute pointer high byte
LAB_CD75
        JMP     LAB_C7AE                ; go do interpreter inner loop

; NEXT loop comlete

LAB_CD78
        TXA                             ; stack copy to A
        ADC     #$11                    ; add $12, $11 + carry, to dump FOR structure
        TAX                             ; copy back to index
        TXS                             ; copy to stack pointer
        JSR     LAB_0079                ; scan memory
        CMP     #','                    ; compare with ","
        BNE     LAB_CD75                ; if not "," go do interpreter inner loop

                                        ; was "," so another NEXT variable to do
        JSR     LAB_0073                ; increment and scan memory
        JSR     LAB_CD24                ; do NEXT variable


;**************************************************************************;
;
```

```
; evaluate expression and check type mismatch

LAB_CD8A
        JSR     LAB_CD9E                ; evaluate expression

; check if source and destination are numeric

LAB_CD8D
        CLC
        .byte   $24                     ; makes next line BIT LAB_38

; check if source and destination are string

LAB_CD8F
        SEC                             ; destination is string

; type match check, set C for string, clear C for numeric

LAB_CD90
        BIT     LAB_0D          ; test data type flag, $FF = string, $00 = numeric
        BMI     LAB_CD97                ; if string go check string is required

; type found is numeric, check required

        BCS     LAB_CD99                ; if string is required go do type missmatch error
LAB_CD96
        RTS

; type found is string, check required

LAB_CD97
        BCS     LAB_CD96                ; exit if string is required

; do type missmatch error

LAB_CD99
        LDX     #$16                    ; error code $16, type missmatch error
        JMP     LAB_C437                ; do error #X then warm start


;******************************************************************************;
;
; evaluate expression

LAB_CD9E
        LDX     LAB_7A          ; get BASIC execute pointer low byte
        BNE     LAB_CDA4                ; skip next if not zero

        DEC     LAB_7B          ; else decrement BASIC execute pointer high byte
LAB_CDA4
        DEC     LAB_7A          ; decrement BASIC execute pointer low byte
        LDX     #$00                    ; set null precedence, flag done
        .byte   $24                     ; makes next line BIT LAB_48
LAB_CDA9
        PHA                             ; push compare evaluation byte if branch to here
        TXA                             ; copy precedence byte
        PHA                             ; push precedence byte
        LDA     #$01                    ; 2 bytes
        JSR     LAB_C3FB                ; check room on stack for A*2 bytes
        JSR     LAB_CE83                ; get value from line
        LDA     #$00                    ; clear A
```

```
        STA     LAB_4D          ; clear comparrison evaluation flag
LAB_CDB8
        JSR     LAB_0079                ; scan memory
LAB_CDBB
        SEC                             ; set carry for subtract
        SBC     #TK_GT          ; subtract token for ">"
        BCC     LAB_CDD7                ; if < ">" skip comparrison test check

        CMP     #$03                    ; compare with ">" to +3
        BCS     LAB_CDD7                ; if >= 3 skip comparrison test check

                                        ; was token for ">" "=" or "<"
        CMP     #$01                    ; compare with token for =
        ROL                             ; *2, b0 = carry (=1 if token was = or <)
        EOR     #$01                    ; toggle b0
        EOR     LAB_4D          ; EOR with comparrison evaluation flag
        CMP     LAB_4D          ; compare with comparrison evaluation flag
        BCC     LAB_CE30                ; if < saved flag do syntax error then warm start

        STA     LAB_4D          ; save new comparrison evaluation flag
        JSR     LAB_0073                ; increment and scan memory
        JMP     LAB_CDBB                ; go do next character

LAB_CDD7
        LDX     LAB_4D          ; get comparrison evaluation flag
        BNE     LAB_CE07                ; if compare function flagged go evaluate right hand
side

        BCS     LAB_CE58                ; go do functions

                                        ; else was < TK_GT so is operator or lower
        ADC     #$07                    ; add # of operators (+, -, *, /, ^, AND or OR)
        BCC     LAB_CE58                ; if < + operator go do the function

                                        ; carry was set so token was +, -, *, /, ^, AND or OR
        ADC     LAB_0D          ; add data type flag, $FF = string, $00 = numeric
        BNE     LAB_CDE8                ; if not string or not + token skip concatenate

                                        ; will only be $00 if type is string and token was +
        JMP     LAB_D63D                ; add strings, string 1 is in the descriptor, string 2
                                        ; is in line, and return

LAB_CDE8
        ADC     #$FF                    ; -1 (corrects for carry add)
        STA     LAB_22          ; save it
        ASL                             ; *2
        ADC     LAB_22          ; *3
        TAY                             ; copy to index
LAB_CDF0
        PLA                             ; pull previous precedence
        CMP     LAB_C080,Y              ; compare with precedence byte
        BCS     LAB_CE5D                ; if A >= go do the function

        JSR     LAB_CD8D                ; check if source is numeric, else do type mismatch
LAB_CDF9
        PHA                             ; save precedence
LAB_CDFA
        JSR     LAB_CE20                ; get vector, execute function then continue
evaluation
        PLA                             ; restore precedence
        LDY     LAB_4B          ; get precedence stacked flag
```

```
        BPL     LAB_CE19                ; if stacked values go check the precedence

        TAX                             ; copy precedence, set flags
        BEQ     LAB_CE5B                ; exit if done

        BNE     LAB_CE66                ; else pop FAC2 and return, branch always

LAB_CE07
        LSR     LAB_0D          ; clear data type flag, $FF = string, $00 = numeric
        TXA                             ; copy compare function flag
        ROL                             ; <<1, shift data type flag into b0, 1 = string, 0 =
num
        LDX     LAB_7A          ; get BASIC execute pointer low byte
        BNE     LAB_CE11                ; if no underflow skip the high byte decrement

        DEC     LAB_7B          ; else decrement BASIC execute pointer high byte
LAB_CE11
        DEC     LAB_7A          ; decrement BASIC execute pointer low byte
        LDY     #LAB_C09B-LAB_C080
                                        ; set offset to = operator precedence entry
        STA     LAB_4D          ; save new comparrison evaluation flag
        BNE     LAB_CDF0                ; branch always

LAB_CE19
        CMP     LAB_C080,Y              ; compare with stacked function precedence
        BCS     LAB_CE66                ; if A >=, pop FAC2 and return

        BCC     LAB_CDF9                ; else go stack this one and continue, branch always


;*******************************************************************************;
;
; get vector, execute function then continue evaluation

LAB_CE20
        LDA     LAB_C080+2,Y    ; get function vector high byte
        PHA                             ; onto stack
        LDA     LAB_C080+1,Y    ; get function vector low byte
        PHA                             ; onto stack
                                        ; now push sign, round FAC1 and put on stack
        JSR     LAB_CE33                ; function will return here, then the next RTS will
call
                                        ; the function
        LDA     LAB_4D          ; get comparrison evaluation flag
        JMP     LAB_CDA9                ; continue evaluating expression

LAB_CE30
        JMP     LAB_CF08                ; do syntax error then warm start

LAB_CE33
        LDA     LAB_66          ; get FAC1 sign (b7)
        LDX     LAB_C080,Y              ; get precedence byte


;*******************************************************************************;
;
; push sign, round FAC1 and put on stack

LAB_CE38
        TAY                             ; copy sign
        PLA                             ; get return address low byte
```

```
        STA     LAB_22          ; save it
        INC     LAB_22          ; increment it as return-1 is pushed
                                ; note, no check is made on the high byte so if the
calling
                                ; routine ever assembles to a page edge then this all
goes
                                ; horribly wrong!
        PLA                     ; get return address high byte
        STA     LAB_23          ; save it
        TYA                     ; restore sign
        PHA                     ; push sign


;******************************************************************************;
;
; round FAC1 and put on stack

LAB_CE43
        JSR     LAB_DC1B                ; round FAC1
        LDA     LAB_65          ; get FAC1 mantissa 4
        PHA                     ; save it
        LDA     LAB_64          ; get FAC1 mantissa 3
        PHA                     ; save it
        LDA     LAB_63          ; get FAC1 mantissa 2
        PHA                     ; save it
        LDA     LAB_62          ; get FAC1 mantissa 1
        PHA                     ; save it
        LDA     LAB_61          ; get FAC1 exponent
        PHA                     ; save it
        JMP     (LAB_22)                ; return, sort of


;******************************************************************************;
;
; do functions

LAB_CE58
        LDY     #$FF                    ; flag function
        PLA                     ; pull precedence byte
LAB_CE5B
        BEQ     LAB_CE80                ; exit if done

LAB_CE5D
        CMP     #$64                    ; compare previous precedence with $64
        BEQ     LAB_CE64                ; if was $64 (< function) skip the type check

        JSR     LAB_CD8D                ; check if source is numeric, else do type mismatch
LAB_CE64
        STY     LAB_4B          ; save precedence stacked flag

                                ; pop FAC2 and return
LAB_CE66
        PLA                     ; pop byte
        LSR                     ; shift out comparison evaluation lowest bit
        STA     LAB_12          ; save the comparison evaluation flag
        PLA                     ; pop exponent
        STA     LAB_69          ; save FAC2 exponent
        PLA                     ; pop mantissa 1
        STA     LAB_6A          ; save FAC2 mantissa 1
        PLA                     ; pop mantissa 2
        STA     LAB_6B          ; save FAC2 mantissa 2
```

```
        PLA                             ; pop mantissa 3
        STA     LAB_6C          ; save FAC2 mantissa 3
        PLA                             ; pop mantissa 4
        STA     LAB_6D          ; save FAC2 mantissa 4
        PLA                             ; pop sign
        STA     LAB_6E          ; save FAC2 sign (b7)
        EOR     LAB_66          ; EOR FAC1 sign (b7)
        STA     LAB_6F          ; save sign compare (FAC1 EOR FAC2)
LAB_CE80
        LDA     LAB_61          ; get FAC1 exponent
        RTS


;**************************************************************************;
;
; get value from line

LAB_CE83
        JMP     (LAB_030A)              ; get arithmetic element


;**************************************************************************;
;
; get arithmetic element, the get arithmetic element vector is initialised to point here

LAB_CE86
        LDA     #$00                    ; clear byte
        STA     LAB_0D          ; clear data type flag, $FF = string, $00 = numeric
LAB_CE8A
        JSR     LAB_0073                ; increment and scan memory
        BCS     LAB_CE92                ; if not numeric character continue

; else numeric string found (e.g. 123)

LAB_CE8F
        JMP     LAB_DCF3                ; get FAC1 from string and return

; get value from line .. continued, wasn't a number so ...

LAB_CE92
        JSR     LAB_D113                ; check byte, return Cb = 0 if<"A" or >"Z"
        BCC     LAB_CE9A                ; if not variable name continue

        JMP     LAB_CF28                ; variable name set-up and return

; get value from line .. continued, wasn't a variable name so ...

LAB_CE9A
        CMP     #TK_PI          ; compare with token for PI
        BNE     LAB_CEAD                ; if not PI continue

; else return PI in FAC1

        LDA     #<LAB_CEA8              ; get PI pointer low byte
        LDY     #>LAB_CEA8              ; get PI pointer high byte
        JSR     LAB_DBA2                ; unpack memory (AY) into FAC1
        JMP     LAB_0073                ; increment and scan memory and return


;**************************************************************************;
;
```

```
; PI as floating number

LAB_CEA8
        .byte   $82,$49,$0F,$DA,$A1
                                        ; 3.141592653


;******************************************************************************;
;
; get value from line .. continued, wasn't PI so ...

LAB_CEAD
        CMP     #'.'                    ; compare with "."
        BEQ     LAB_CE8F                ; if so get FAC1 from string and return, e.g. was .123

                                        ; wasn't .123 so ...
        CMP     #TK_MINUS               ; compare with token for -
        BEQ     LAB_CF0D                ; if - token, do set-up for functions

                                        ; wasn't -123 so ...
        CMP     #TK_PLUS                ; compare with token for +
        BEQ     LAB_CE8A                ; if + token ignore the leading +, +1 = 1

                                        ; it wasn't any sort of number so ...
        CMP     #$22                    ; compare with "
        BNE     LAB_CECC                ; if not open quote continue

                                        ; was open quote so get the enclosed string

; print "..." string to string utility area

LAB_CEBD
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        LDY     LAB_7B          ; get BASIC execute pointer high byte
        ADC     #$00                    ; add carry to low byte
        BCC     LAB_CEC6                ; branch if no overflow

        INY                             ; increment high byte
LAB_CEC6
        JSR     LAB_D487                ; print " terminated string to utility pointer
        JMP     LAB_D7E2                ; restore BASIC execute pointer from temp and return

; get value from line .. continued, wasn't a string so ...

LAB_CECC
        CMP     #TK_NOT         ; compare with token for NOT
        BNE     LAB_CEE3                ; if not token for NOT continue

; was NOT token

        LDY     #$18                    ; offset to NOT function
        BNE     LAB_CF0F                ; do set-up for function then execute, branch always

; do = compare

LAB_CED4
        JSR     LAB_D1BF                ; evaluate integer expression, no sign check
        LDA     LAB_65          ; get FAC1 mantissa 4
        EOR     #$FF                    ; invert it
        TAY                             ; copy it
        LDA     LAB_64          ; get FAC1 mantissa 3
```

```
        EOR     #$FF                    ; invert it
        JMP     LAB_D391                ; convert fixed integer AY to float FAC1 and return

; get value from line .. continued, wasn't NOT so ...

LAB_CEE3
        CMP     #TK_FN          ; compare with token for FN
        BNE     LAB_CEEA                ; if not token for FN continue

        JMP     LAB_D3F4                ; else go evaluate FNx

; get value from line .. continued, wasn't FN so ...

LAB_CEEA
        CMP     #TK_SGN         ; compare with token for SGN
        BCC     LAB_CEF1                ; if less than SGN token go evaluate expression in ()

                                        ; else was a function token
        JMP     LAB_CFA7                ; go set up function references, branch always


;******************************************************************************;
;
; get value from line .. continued
; if here it can only be something in brackets so ....

; evaluate expression within parentheses

LAB_CEF1
        JSR     LAB_CEFA                ; scan for "(", else do syntax error then warm start
        JSR     LAB_CD9E                ; evaluate expression


;******************************************************************************;
;
; all the 'scan for' routines return the character after the sought character

; scan for ")", else do syntax error then warm start

LAB_CEF7
        LDA     #$29                    ; load A with ")"
        .byte   $2C                     ; makes next line BIT LAB_28A9

; scan for "(", else do syntax error then warm start

LAB_CEFA
        LDA     #$28                    ; load A with "("
        .byte   $2C                     ; makes next line BIT LAB_2CA9

; scan for ",", else do syntax error then warm start

LAB_CEFD
        LDA     #','                    ; load A with ","

; scan for CHR$(A), else do syntax error then warm start

LAB_CEFF
        LDY     #$00                    ; clear index
        CMP     (LAB_7A),Y              ; compare with BASIC byte
        BNE     LAB_CF08                ; if not expected byte do syntax error then warm start
```

```
        JMP     LAB_0073                ; else increment and scan memory and return


;******************************************************************************;
;
; syntax error then warm start

LAB_CF08
        LDX     #$0B                    ; error code $0B, syntax error
        JMP     LAB_C437                ; do error #X then warm start

LAB_CF0D
        LDY     #$15                    ; set offset from base to > operator
LAB_CF0F
        PLA                             ; dump return address low byte
        PLA                             ; dump return address high byte
        JMP     LAB_CDFA                ; execute function then continue evaluation


;******************************************************************************;
;
; check address range, return Cb = 1 if address in BASIC ROM

LAB_CF14
        SEC                             ; set carry for subtract
        LDA     LAB_64          ; get variable address low byte
        SBC     #$00                    ; subtract $C000 low byte
        LDA     LAB_65          ; get variable address high byte
        SBC     #$C0                    ; subtract $C000 high byte
        BCC     LAB_CF27                ; exit if address < $C000

        LDA     #<LAB_E387              ; get end of BASIC marker low byte
        SBC     LAB_64          ; subtract variable address low byte
        LDA     #>LAB_E387              ; get end of BASIC marker high byte
        SBC     LAB_65          ; subtract variable address high byte
LAB_CF27
        RTS


;******************************************************************************;
;
; variable name set-up

LAB_CF28
        JSR     LAB_D08B                ; get variable address
        STA     LAB_64          ; save variable pointer low byte
        STY     LAB_65          ; save variable pointer high byte
        LDX     LAB_45          ; get current variable name first character
        LDY     LAB_46          ; get current variable name second character
        LDA     LAB_0D          ; get data type flag, $FF = string, $00 = numeric
        BEQ     LAB_CF5D                ; if numeric go handle a numeric variable

; variable is string

        LDA     #$00                    ; else clear A
        STA     LAB_70          ; clear FAC1 rounding byte
        JSR     LAB_CF14                ; check address range
        BCC     LAB_CF5C                ; exit if not in BASIC ROM

        CPX     #'T'                    ; compare variable name first character with "T"
        BNE     LAB_CF5C                ; exit if not "T"
```

```
        CPY     #'I'+$80                ; compare variable name second character with "I$"
        BNE     LAB_CF5C                ; exit if not "I$"

                                        ; variable name was "TI$"
        JSR     LAB_CF84                ; read real time clock into FAC1 mantissa, 0HML
        STY     LAB_5E          ; clear exponent count adjust
        DEY                             ; Y = $FF
        STY     LAB_71          ; set output string index, -1 to allow for pre increment
        LDY     #$06                    ; HH:MM:SS is six digits
        STY     LAB_5D          ; set number of characters before the decimal point
        LDY     #LAB_DF3A-LAB_DF16
                                        ; index to jiffy conversion table
        JSR     LAB_DE68                ; convert jiffy count to string
        JMP     LAB_D46F                ; exit via STR$() code tail

LAB_CF5C
        RTS

; variable name set-up, variable is numeric

LAB_CF5D
        BIT     LAB_0E          ; test data type flag, $80 = integer, $00 = float
        BPL     LAB_CF6E                ; if float go handle float

; else handle integer variable

        LDY     #$00                    ; clear index
        LDA     (LAB_64),Y              ; get integer variable low byte
        TAX                             ; copy to X
        INY                             ; increment index
        LDA     (LAB_64),Y              ; get integer variable high byte
        TAY                             ; copy to Y
        TXA                             ; copy loa byte to A
        JMP     LAB_D391                ; convert fixed integer AY to float FAC1 and return

; variable name set-up, variable is float

LAB_CF6E
        JSR     LAB_CF14                ; check address range
        BCC     LAB_CFA0                ; if not in BASIC ROM get pointer and unpack into FAC1

        CPX     #'T'                    ; compare variable name first character with "T"
        BNE     LAB_CF92                ; if not "T" skip Tx variables

        CPY     #'I'                    ; compare variable name second character with "I"
        BNE     LAB_CFA0                ; if not "I" go do plain float

                                        ; variable name was "TI"
        JSR     LAB_CF84                ; read real time clock into FAC1 mantissa, 0HML
        TYA                             ; clear A
        LDX     #$A0                    ; set exponent to 32 bit value
        JMP     LAB_DC4F                ; set exponent = X and normalise FAC1


;******************************************************************************;
;
; read real time clock into FAC1 mantissa, 0HML

LAB_CF84
        JSR     LAB_FFDE                ; read real time clock
```

```
        STX     LAB_64          ; save jiffy clock mid byte as  FAC1 mantissa 3
        STY     LAB_63          ; save jiffy clock high byte as  FAC1 mantissa 2
        STA     LAB_65          ; save jiffy clock low byte as  FAC1 mantissa 4
        LDY     #$00                    ; clear Y
        STY     LAB_62          ; clear FAC1 mantissa 1
        RTS


;******************************************************************************;
;
; variable name set-up, variable is float and not "Tx"

LAB_CF92
        CPX     #'S'                    ; compare variable name first character with "S"
        BNE     LAB_CFA0                ; if not "S" go do normal floating variable

        CPY     #'T'                    ; compare variable name second character with "
        BNE     LAB_CFA0                ; if not "T" go do normal floating variable

                                        ; variable name was "ST"
        JSR     LAB_FFB7                ; read I/O status word
        JMP     LAB_DC3C                ; save A as integer byte and return

; variable is plain float

LAB_CFA0
        LDA     LAB_64          ; get variable pointer low byte
        LDY     LAB_65          ; get variable pointer high byte
        JMP     LAB_DBA2                ; unpack memory (AY) into FAC1


;******************************************************************************;
;
; get value from line continued
; only functions left so ..

; set up function references

LAB_CFA7
        ASL                             ; *2 (2 bytes per function address)
        PHA                             ; save function offset
        TAX                             ; copy function offset
        JSR     LAB_0073                ; increment and scan memory
        CPX     #$8F                    ; compare function offset to CHR$ token offset+1
        BCC     LAB_CFD1                ; if < LEFT$ (can not be =) go do function setup

; get value from line .. continued
; was LEFT$, RIGHT$ or MID$ so..

        JSR     LAB_CEFA                ; scan for "(", else do syntax error then warm start
        JSR     LAB_CD9E                ; evaluate, should be string, expression
        JSR     LAB_CEFD                ; scan for ",", else do syntax error then warm start
        JSR     LAB_CD8F                ; check if source is string, else do type mismatch
        PLA                             ; restore function offset
        TAX                             ; copy it
        LDA     LAB_65          ; get descriptor pointer high byte
        PHA                             ; push string pointer high byte
        LDA     LAB_64          ; get descriptor pointer low byte
        PHA                             ; push string pointer low byte
        TXA                             ; restore function offset
        PHA                             ; save function offset
```

```
        JSR     LAB_D79E                ; get byte parameter
        PLA                             ; restore function offset
        TAY                             ; copy function offset
        TXA                             ; copy byte parameter to A
        PHA                             ; push byte parameter
        JMP     LAB_CFD6                ; go call function

; get value from line .. continued
; was SGN() to CHR$() so..

LAB_CFD1
        JSR     LAB_CEF1                ; evaluate expression within parentheses
        PLA                             ; restore function offset
        TAY                             ; copy to index
LAB_CFD6
        LDA     LAB_C052-$68,Y  ; get function jump vector low byte
        STA     LAB_55          ; save functions jump vector low byte
        LDA     LAB_C052-$67,Y  ; get function jump vector high byte
        STA     LAB_56          ; save functions jump vector high byte
        JSR     LAB_54          ; do function call
        JMP     LAB_CD8D                ; check if source is numeric and RTS, else do type
mismatch
                                        ; string functions avoid this by dumping the return
address


;******************************************************************************;
;
; perform OR
; this works because NOT(NOT(x) AND NOT(y)) = x OR y

LAB_CFE6
        LDY     #$FF                    ; set Y for OR
        .byte   $2C                     ; makes next line BIT LAB_00A0


;******************************************************************************;
;
; perform AND

LAB_CFE9
        LDY     #$00                    ; clear Y for AND
        STY     LAB_0B          ; set AND/OR invert value
        JSR     LAB_D1BF                ; evaluate integer expression, no sign check
        LDA     LAB_64          ; get FAC1 mantissa 3
        EOR     LAB_0B          ; EOR low byte
        STA     LAB_07          ; save it
        LDA     LAB_65          ; get FAC1 mantissa 4
        EOR     LAB_0B          ; EOR high byte
        STA     LAB_08          ; save it
        JSR     LAB_DBFC                ; copy FAC2 to FAC1, get 2nd value in expression
        JSR     LAB_D1BF                ; evaluate integer expression, no sign check
        LDA     LAB_65          ; get FAC1 mantissa 4
        EOR     LAB_0B          ; EOR high byte
        AND     LAB_08          ; AND with expression 1 high byte
        EOR     LAB_0B          ; EOR result high byte
        TAY                             ; save in Y
        LDA     LAB_64          ; get FAC1 mantissa 3
        EOR     LAB_0B          ; EOR low byte
        AND     LAB_07          ; AND with expression 1 low byte
        EOR     LAB_0B          ; EOR result low byte
```

```
        JMP     LAB_D391                ; convert fixed integer AY to float FAC1 and return


;******************************************************************************;
;
; perform comparisons

; do < compare

LAB_D016
        JSR     LAB_CD90                ; type match check, set C for string
        BCS     LAB_D02E                ; if string go do string compare

                                        ; do numeric < compare
        LDA     LAB_6E          ; get FAC2 sign (b7)
        ORA     #$7F                    ; set all non sign bits
        AND     LAB_6A          ; and FAC2 mantissa 1 (AND in sign bit)
        STA     LAB_6A          ; save FAC2 mantissa 1
        LDA     #<LAB_69                ; set pointer low byte to FAC2
        LDY     #>LAB_69                ; set pointer high byte to FAC2
        JSR     LAB_DC5B                ; compare FAC1 with (AY)
        TAX                             ; copy the result
        JMP     LAB_D061                ; go evaluate result

; do string < compare

LAB_D02E
        LDA     #$00                    ; clear byte
        STA     LAB_0D          ; clear data type flag, $FF = string, $00 = numeric
        DEC     LAB_4D          ; clear < bit in comparrison evaluation flag
        JSR     LAB_D6A6                ; pop string off descriptor stack, or from top of
string
                                        ; space returns with A = length, X = pointer low byte,
                                        ; Y = pointer high byte
        STA     LAB_61          ; save length
        STX     LAB_62          ; save string pointer low byte
        STY     LAB_63          ; save string pointer high byte
        LDA     LAB_6C          ; get descriptor pointer low byte
        LDY     LAB_6D          ; get descriptor pointer high byte
        JSR     LAB_D6AA                ; pop (YA) descriptor off stack or from top of string
space
                                        ; returns with A = length, X = pointer low byte,
                                        ; Y = pointer high byte
        STX     LAB_6C          ; save string pointer low byte
        STY     LAB_6D          ; save string pointer high byte
        TAX                             ; copy length
        SEC                             ; set carry for subtract
        SBC     LAB_61          ; subtract string 1 length
        BEQ     LAB_D056                ; if str 1 length = string 2 length go compare the
strings

        LDA     #$01                    ; set str 1 length > string 2 length
        BCC     LAB_D056                ; if so return + 1 if otherwise equal

        LDX     LAB_61          ; get string 1 length
        LDA     #$FF                    ; set str 1 length < string 2 length
LAB_D056
        STA     LAB_66          ; save length compare
        LDY     #$FF                    ; set index
        INX                             ; adjust for loop
LAB_D05B
```

```
        INY                             ; increment index
        DEX                             ; decrement count
        BNE     LAB_D066                ; if still bytes to do go compare them

        LDX     LAB_66          ; get length compare back
LAB_D061
        BMI     LAB_D072                ; branch if str 1 < str 2

        CLC                             ; flag str 1 <= str 2
        BCC     LAB_D072                ; go evaluate result, branch always

LAB_D066
        LDA     (LAB_6C),Y              ; get string 2 byte
        CMP     (LAB_62),Y              ; compare with string 1 byte
        BEQ     LAB_D05B                ; loop if bytes =

        LDX     #$FF                    ; set str 1 < string 2
        BCS     LAB_D072                ; branch if so

        LDX     #$01                    ; set str 1 > string 2
LAB_D072
        INX                             ; x = 0, 1 or 2
        TXA                             ; copy to A
        ROL                             ; * 2 (1, 2 or 4)
        AND     LAB_12          ; AND with the comparison evaluation flag
        BEQ     LAB_D07B                ; branch if 0 (compare is false)

        LDA     #$FF                    ; else set result true
LAB_D07B
        JMP     LAB_DC3C                ; save A as integer byte and return

LAB_D07E
        JSR     LAB_CEFD                ; scan for ",", else do syntax error then warm start


;******************************************************************************;
;
; perform DIM

LAB_D081
        TAX                             ; copy "DIM" flag to X
        JSR     LAB_D090                ; search for variable
        JSR     LAB_0079                ; scan memory
        BNE     LAB_D07E                ; scan for "," and loop if not null

        RTS


;******************************************************************************;
;
; search for variable

LAB_D08B
        LDX     #$00                    ; set DIM flag = $00
        JSR     LAB_0079                ; scan memory, 1st character
LAB_D090
        STX     LAB_0C          ; save DIM flag
LAB_D092
        STA     LAB_45          ; save 1st character
        JSR     LAB_0079                ; scan memory
        JSR     LAB_D113                ; check byte, return Cb = 0 if<"A" or >"Z"
```

```
        BCS     LAB_D09F                ; if ok continue

LAB_D09C
        JMP     LAB_CF08                ; else syntax error then warm start

; was variable name so ...

LAB_D09F
        LDX     #$00                    ; clear 2nd character temp
        STX     LAB_0D          ; clear data type flag, $FF = string, $00 = numeric
        STX     LAB_0E          ; clear data type flag, $80 = integer, $00 = float
        JSR     LAB_0073                ; increment and scan memory, 2nd character
        BCC     LAB_D0AF                ; if character = "0"-"9" (ok) go save 2nd character

                                        ; 2nd character wasn't "0" to "9" so ...
        JSR     LAB_D113                ; check byte, return Cb = 0 if<"A" or >"Z"
        BCC     LAB_D0BA                ; if <"A" or >"Z" go check if string

LAB_D0AF
        TAX                             ; copy 2nd character

                                        ; ignore further (valid) characters in the variable
name
LAB_D0B0
        JSR     LAB_0073                ; increment and scan memory, 3rd character
        BCC     LAB_D0B0                ; loop if character = "0"-"9" (ignore)

        JSR     LAB_D113                ; check byte, return Cb = 0 if<"A" or >"Z"
        BCS     LAB_D0B0                ; loop if character = "A"-"Z" (ignore)

                                        ; check if string variable
LAB_D0BA
        CMP     #'$'                    ; compare with "$"
        BNE     LAB_D0C4                ; if not string go check integer

                                        ; type is string
        LDA     #$FF                    ; set data type = string
        STA     LAB_0D          ; set data type flag, $FF = string, $00 = numeric
        BNE     LAB_D0D4                ; branch always

LAB_D0C4
        CMP     #$25                    ; compare with "%"
        BNE     LAB_D0DB                ; if not integer go check for an array

        LDA     LAB_10          ; get subscript/FNX flag
        BNE     LAB_D09C                ; if ?? do syntax error then warm start

        LDA     #$80                    ; set integer type
        STA     LAB_0E          ; set data type = integer
        ORA     LAB_45          ; OR current variable name first byte
        STA     LAB_45          ; save current variable name first byte
LAB_D0D4
        TXA                             ; get 2nd character back
        ORA     #$80                    ; set top bit, indicate string or integer variable
        TAX                             ; copy back to 2nd character temp
        JSR     LAB_0073                ; increment and scan memory
LAB_D0DB
        STX     LAB_46          ; save 2nd character
        SEC                             ; set carry for subtract
        ORA     LAB_10          ; or with subscript/FNX flag - or FN name
        SBC     #$28                    ; subtract "("
```

```
        BNE     LAB_D0E7                ; if not "(" go find a plain numeric variable

        JMP     LAB_D1D1                ; else go find, or make, array

; either find or create variable

                                        ; variable name wasn't xx(.... so look for plain
variable
LAB_D0E7
        LDY     #$00                    ; clear A
        STY     LAB_10          ; clear subscript/FNX flag
        LDA     LAB_2D          ; get start of variables low byte
        LDX     LAB_2E          ; get start of variables high byte
LAB_D0EF
        STX     LAB_60          ; save search address high byte
LAB_D0F1
        STA     LAB_5F          ; save search address low byte
        CPX     LAB_30          ; compare with end of variables high byte
        BNE     LAB_D0FB                ; skip next compare if <>

                                        ; high addresses were = so compare low addresses
        CMP     LAB_2F          ; compare low address with end of variables low byte
        BEQ     LAB_D11D                ; if not found go make new variable

LAB_D0FB
        LDA     LAB_45          ; get 1st character of variable to find
        CMP     (LAB_5F),Y              ; compare with variable name 1st character
        BNE     LAB_D109                ; if no match go try the next variable

                                        ; 1st characters match so compare 2nd character
        LDA     LAB_46          ; get 2nd character of variable to find
        INY                             ; index to point to variable name 2nd character
        CMP     (LAB_5F),Y              ; compare with variable name 2nd character
        BEQ     LAB_D185                ; if match go return the variable

        DEY                             ; else decrement index (now = $00)
LAB_D109
        CLC                             ; clear carry for add
        LDA     LAB_5F          ; get search address low byte
        ADC     #$07                    ; +7, offset to next variable name
        BCC     LAB_D0F1                ; loop if no overflow to high byte

        INX                             ; else increment high byte
        BNE     LAB_D0EF                ; loop always, RAM doesn't extend to $FFFF


;******************************************************************************;
;
; check byte, return Cb = 0 if<"A" or >"Z"

LAB_D113
        CMP     #$41                    ; compare with "A"
        BCC     LAB_D11C                ; exit if less

                                        ; carry is set
        SBC     #$5B                    ; subtract "Z"+1
        SEC                             ; set carry
        SBC     #$A5                    ; subtract $A5 (restore byte)
                                        ; carry clear if byte > $5A
LAB_D11C
        RTS
```

```
;******************************************************************************;
;
                                        ; reached end of variable memory without match
                                        ; ... so create new variable
LAB_D11D
        PLA                             ; pop return address low byte
        PHA                             ; push return address low byte
        CMP     #$2A                    ; compare with expected calling routine return low
byte
        BNE     LAB_D128                ; if not get variable go create new variable

; this will only drop through if the call was from LAB_xxxx and is only called
; from there if it is searching for a variable from the right hand side of a LET a=b
; statement, it prevents the creation of variables not assigned a value.

; value returned by this is either numeric zero, exponent byte is $00, or null string,
; descriptor length byte is $00. in fact a pointer to any $00 byte would have done.

                                        ; else return dummy null value
LAB_D123
        LDA     #<LAB_DF13              ; set result pointer low byte
        LDY     #>LAB_DF13              ; set result pointer high byte
        RTS

                                        ; create new numeric variable
LAB_D128
        LDA     LAB_45          ; get variable name first character
        LDY     LAB_46          ; get variable name second character
        CMP     #'T'                    ; compare first character with "T"
        BNE     LAB_D13B                ; if not "T" continue

        CPY     #'I'+$80                ; compare second character with "I$"
        BEQ     LAB_D123                ; if "I$" return null value

        CPY     #'I'                    ; compare second character with "I"
        BNE     LAB_D13B                ; if not "I" continue

                                        ; if name is "TI" do syntax error
LAB_D138
        JMP     LAB_CF08                ; do syntax error then warm start

LAB_D13B
        CMP     #'S'                    ; compare first character with "S"
        BNE     LAB_D143                ; if not "S" continue

        CPY     #'T'                    ; compare second character with "T"
        BEQ     LAB_D138                ; if name is "ST" do syntax error

LAB_D143
        LDA     LAB_2F          ; get end of variables low byte
        LDY     LAB_30          ; get end of variables high byte
        STA     LAB_5F          ; save old block start low byte
        STY     LAB_60          ; save old block start high byte
        LDA     LAB_31          ; get end of arrays low byte
        LDY     LAB_32          ; get end of arrays high byte
        STA     LAB_5A          ; save old block end low byte
        STY     LAB_5B          ; save old block end high byte
        CLC                             ; clear carry for add
        ADC     #$07                    ; +7, space for one variable
```

```
        BCC     LAB_D159                ; if no overflow skip the high byte increment

        INY                             ; else increment high byte
LAB_D159
        STA     LAB_58          ; set new block end low byte
        STY     LAB_59          ; set new block end high byte
        JSR     LAB_C3B8                ; open up space in memory
        LDA     LAB_58          ; get new start low byte
        LDY     LAB_59          ; get new start high byte (-$100)
        INY                             ; correct high byte
        STA     LAB_2F          ; set end of variables low byte
        STY     LAB_30          ; set end of variables high byte
        LDY     #$00                    ; clear index
        LDA     LAB_45          ; get variable name 1st character
        STA     (LAB_5F),Y              ; save variable name 1st character
        INY                             ; increment index
        LDA     LAB_46          ; get variable name 2nd character
        STA     (LAB_5F),Y              ; save variable name 2nd character
        LDA     #$00                    ; clear A
        INY                             ; increment index
        STA     (LAB_5F),Y              ; initialise variable byte
        INY                             ; increment index
        STA     (LAB_5F),Y              ; initialise variable byte
        INY                             ; increment index
        STA     (LAB_5F),Y              ; initialise variable byte
        INY                             ; increment index
        STA     (LAB_5F),Y              ; initialise variable byte
        INY                             ; increment index
        STA     (LAB_5F),Y              ; initialise variable byte

                                        ; found a match for variable
LAB_D185
        LDA     LAB_5F          ; get variable address low byte
        CLC                             ; clear carry for add
        ADC     #$02                    ; +2, offset past variable name bytes
        LDY     LAB_60          ; get variable address high byte
        BCC     LAB_D18F                ; if no overflow skip the high byte increment

        INY                             ; else increment high byte
LAB_D18F
        STA     LAB_47          ; save current variable pointer low byte
        STY     LAB_48          ; save current variable pointer high byte
        RTS


;******************************************************************************;
;
; set-up array pointer to first element in array

LAB_D194
        LDA     LAB_0B          ; get # of dimensions (1, 2 or 3)
        ASL                             ; *2 (also clears the carry !)
        ADC     #$05                    ; +5 (result is 7, 9 or 11 here)
        ADC     LAB_5F          ; add array start pointer low byte
        LDY     LAB_60          ; get array pointer high byte
        BCC     LAB_D1A0                ; if no overflow skip the high byte increment

        INY                             ; else increment high byte
LAB_D1A0
        STA     LAB_58          ; save array data pointer low byte
        STY     LAB_59          ; save array data pointer high byte
```

```
        RTS


;******************************************************************************;
;
; -32768 as floating value

LAB_D1A5
        .byte   $90,$80,$00,$00,$00     ; -32768


;******************************************************************************;
;
; convert float to fixed

LAB_D1AA
        JSR     LAB_D1BF                ; evaluate integer expression, no sign check
        LDA     LAB_64          ; get result low byte
        LDY     LAB_65          ; get result high byte
        RTS


;******************************************************************************;
;
; evaluate integer expression

LAB_D1B2
        JSR     LAB_0073                ; increment and scan memory
        JSR     LAB_CD9E                ; evaluate expression

; evaluate integer expression, sign check

LAB_D1B8
        JSR     LAB_CD8D                ; check if source is numeric, else do type mismatch
        LDA     LAB_66          ; get FAC1 sign (b7)
        BMI     LAB_D1CC                ; do illegal quantity error if -ve

; evaluate integer expression, no sign check

LAB_D1BF
        LDA     LAB_61          ; get FAC1 exponent
        CMP     #$90                    ; compare with exponent = 2^16 (n>2^15)
        BCC     LAB_D1CE                ; if n<2^16 go convert FAC1 floating to fixed and
return

        LDA     #<LAB_D1A5              ; set pointer low byte to -32768
        LDY     #>LAB_D1A5              ; set pointer high byte to -32768
        JSR     LAB_DC5B                ; compare FAC1 with (AY)
LAB_D1CC
        BNE     LAB_D248                ; if <> do illegal quantity error then warm start

LAB_D1CE
        JMP     LAB_DC9B                ; convert FAC1 floating to fixed and return


;******************************************************************************;
;
; an array is stored as follows
;
; array name                    ; two bytes with the following patterns for different types
;                                       ; 1st char      2nd char
```

```
;                                       ;   b7    b7            type
element size
;                                       ; --------      --------        -----
------------
;                                       ;   0             0             floating point   5
;                                       ;   0             1             string           3
;                                       ;   1             1             integer          2
; offset to next array          ; word
; dimension count                       ; byte
; 1st dimension size            ; word, this is the number of elements including 0
; 2nd dimension size            ; word, only here if the array has a second dimension
; 2nd dimension size            ; word, only here if the array has a third dimension
;                                       ; note: the dimension size word is in high byte low
byte
;                                       ; format, not like most 6502 words
; then for each element the required number of bytes given as the element size above

; find or make array

LAB_D1D1
        LDA     LAB_0C          ; get DIM flag
        ORA     LAB_0E          ; OR with data type flag
        PHA                             ; push it
        LDA     LAB_0D          ; get data type flag, $FF = string, $00 = numeric
        PHA                             ; push it
        LDY     #$00                    ; clear dimensions count

; now get the array dimension(s) and stack it (them) before the data type and DIM flag

LAB_D1DB
        TYA                             ; copy dimensions count
        PHA                             ; save it
        LDA     LAB_46          ; get array name 2nd byte
        PHA                             ; save it
        LDA     LAB_45          ; get array name 1st byte
        PHA                             ; save it
        JSR     LAB_D1B2                ; evaluate integer expression
        PLA                             ; pull array name 1st byte
        STA     LAB_45          ; restore array name 1st byte
        PLA                             ; pull array name 2nd byte
        STA     LAB_46          ; restore array name 2nd byte
        PLA                             ; pull dimensions count
        TAY                             ; restore it
        TSX                             ; copy stack pointer
        LDA     LAB_0100+2,X    ; get DIM flag
        PHA                             ; push it
        LDA     LAB_0100+1,X    ; get data type flag
        PHA                             ; push it
        LDA     LAB_64          ; get this dimension size high byte
        STA     LAB_0100+2,X    ; stack before flag bytes
        LDA     LAB_65          ; get this dimension size low byte
        STA     LAB_0100+1,X    ; stack before flag bytes
        INY                             ; increment dimensions count
        JSR     LAB_0079                ; scan memory
        CMP     #','                    ; compare with ","
        BEQ     LAB_D1DB                ; if found go do next dimension

        STY     LAB_0B          ; store dimensions count
        JSR     LAB_CEF7                ; scan for ")", else do syntax error then warm start
        PLA                             ; pull data type flag
        STA     LAB_0D          ; restore data type flag, $FF = string, $00 = numeric
```

```
        PLA                             ; pull data type flag
        STA     LAB_0E          ; restore data type flag, $80 = integer, $00 = float
        AND     #$7F                    ; mask dim flag
        STA     LAB_0C          ; restore DIM flag
        LDX     LAB_2F          ; set end of variables low byte
                                        ; (array memory start low byte)
        LDA     LAB_30          ; set end of variables high byte
                                        ; (array memory start high byte)

; now check to see if we are at the end of array memory, we would be if there were
; no arrays.

LAB_D21C
        STX     LAB_5F          ; save as array start pointer low byte
        STA     LAB_60          ; save as array start pointer high byte
        CMP     LAB_32          ; compare with end of arrays high byte
        BNE     LAB_D228                ; if not reached array memory end continue searching

        CPX     LAB_31          ; else compare with end of arrays low byte
        BEQ     LAB_D261                ; go build array if not found

                                        ; search for array
LAB_D228
        LDY     #$00                    ; clear index
        LDA     (LAB_5F),Y              ; get array name first byte
        INY                             ; increment index to second name byte
        CMP     LAB_45          ; compare with this array name first byte
        BNE     LAB_D237                ; if no match go try the next array

        LDA     LAB_46          ; else get this array name second byte
        CMP     (LAB_5F),Y              ; compare with array name second byte
        BEQ     LAB_D24D                ; array found so branch

                                        ; no match
LAB_D237
        INY                             ; increment index
        LDA     (LAB_5F),Y              ; get array size low byte
        CLC                             ; clear carry for add
        ADC     LAB_5F          ; add array start pointer low byte
        TAX                             ; copy low byte to X
        INY                             ; increment index
        LDA     (LAB_5F),Y              ; get array size high byte
        ADC     LAB_60          ; add array memory pointer high byte
        BCC     LAB_D21C                ; if no overflow go check next array

; do bad subscript error

LAB_D245
        LDX     #$12                    ; error $12, bad subscript error
        .byte   $2C                     ; makes next line BIT LAB_0EA2


;******************************************************************************;
;
; do illegal quantity error

LAB_D248
        LDX     #$0E                    ; error $0E, illegal quantity error
LAB_D24A
        JMP     LAB_C437                ; do error #X then warm start
```

```
;******************************************************************************;
;
; array found

LAB_D24D
        LDX     #$13                    ; set error $13, double dimension error
        LDA     LAB_0C          ; get DIM flag
        BNE     LAB_D24A                ; if we are trying to dimension it do error #X then
warm
                                        ; start

; found the array and we're not dimensioning it so we must find an element in it

        JSR     LAB_D194                ; set-up array pointer to first element in array
        LDA     LAB_0B          ; get dimensions count
        LDY     #$04                    ; set index to array's # of dimensions
        CMP     (LAB_5F),Y              ; compare with no of dimensions
        BNE     LAB_D245                ; if wrong do bad subscript error

        JMP     LAB_D2EA                ; found array so go get element

                                        ; array not found, so build it
LAB_D261
        JSR     LAB_D194                ; set-up array pointer to first element in array
        JSR     LAB_C408                ; check available memory, do out of memory error if no
room
        LDY     #$00                    ; clear Y
        STY     LAB_72          ; clear array data size high byte
        LDX     #$05                    ; set default element size
        LDA     LAB_45          ; get variable name 1st byte
        STA     (LAB_5F),Y              ; save array name 1st byte
        BPL     LAB_D274                ; branch if not string or floating point array

        DEX                             ; decrement element size, $04
LAB_D274
        INY                             ; increment index
        LDA     LAB_46          ; get variable name 2nd byte
        STA     (LAB_5F),Y              ; save array name 2nd byte
        BPL     LAB_D27D                ; branch if not integer or string

        DEX                             ; decrement element size, $03
        DEX                             ; decrement element size, $02
LAB_D27D
        STX     LAB_71          ; save element size
        LDA     LAB_0B          ; get dimensions count
        INY                             ; increment index ..
        INY                             ; .. to array  ..
        INY                             ; .. dimension count
        STA     (LAB_5F),Y              ; save array dimension count
LAB_D286
        LDX     #$0B                    ; set default dimension size low byte
        LDA     #$00                    ; set default dimension size high byte
        BIT     LAB_0C          ; test DIM flag
        BVC     LAB_D296                ; if default to be used don't pull a dimension

        PLA                             ; pull dimension size low byte
        CLC                             ; clear carry for add
        ADC     #$01                    ; add 1, allow for zeroeth element
        TAX                             ; copy low byte to X
        PLA                             ; pull dimension size high byte
```

```
        ADC     #$00                    ; add carry to high byte
LAB_D296
        INY                             ; incement index to dimension size high byte
        STA     (LAB_5F),Y              ; save dimension size high byte
        INY                             ; incement index to dimension size low byte
        TXA                             ; copy dimension size low byte
        STA     (LAB_5F),Y              ; save dimension size low byte
        JSR     LAB_D34C                ; compute array size
        STX     LAB_71          ; save result low byte
        STA     LAB_72          ; save result high byte
        LDY     LAB_22          ; restore index
        DEC     LAB_0B          ; decrement dimensions count
        BNE     LAB_D286                ; loop if not all done

        ADC     LAB_59          ; add array data pointer high byte
        BCS     LAB_D30B                ; if overflow do out of memory error then warm start

        STA     LAB_59          ; save array data pointer high byte
        TAY                             ; copy array data pointer high byte
        TXA                             ; copy array size low byte
        ADC     LAB_58          ; add array data pointer low byte
        BCC     LAB_D2B9                ; if no rollover skip the high byte increment

        INY                             ; else increment next array pointer high byte
        BEQ     LAB_D30B                ; if rolled over do out of memory error then warm
start

LAB_D2B9
        JSR     LAB_C408                ; check available memory, do out of memory error if no
room
        STA     LAB_31          ; set end of arrays low byte
        STY     LAB_32          ; set end of arrays high byte

; now the aray is created we need to zero all the elements in it

        LDA     #$00                    ; clear A for array clear
        INC     LAB_72          ; increment array size high byte, now block count
        LDY     LAB_71          ; get array size low byte, now index to block
        BEQ     LAB_D2CD                ; if $00 go do the high byte decrement
LAB_D2C8
        DEY                             ; decrement index, do 0 to n-1
        STA     (LAB_58),Y              ; clear array element byte
        BNE     LAB_D2C8                ; loop until this block done

LAB_D2CD
        DEC     LAB_59          ; decrement array pointer high byte
        DEC     LAB_72          ; decrement block count high byte
        BNE     LAB_D2C8                ; loop until all blocks done

        INC     LAB_59          ; correct for last loop
        SEC                             ; set carry for subtract
        LDA     LAB_31          ; get end of arrays low byte
        SBC     LAB_5F          ; subtract array start low byte
        LDY     #$02                    ; index to array size low byte
        STA     (LAB_5F),Y              ; save array size low byte
        LDA     LAB_32          ; get end of arrays high byte
        INY                             ; index to array size high byte
        SBC     LAB_60          ; subtract array start high byte
        STA     (LAB_5F),Y              ; save array size high byte
        LDA     LAB_0C          ; get default DIM flag
        BNE     LAB_D34B                ; exit if this was a DIM command
```

```
                                        ; else, find element
        INY                             ; set index to # of dimensions, the dimension indeces
                                        ; are on the stack and will be removed as the position
                                        ; of the array element is calculated

LAB_D2EA
        LDA     (LAB_5F),Y              ; get array's dimension count
        STA     LAB_0B          ; save it
        LDA     #$00                    ; clear byte
        STA     LAB_71          ; clear array data pointer low byte
LAB_D2F2
        STA     LAB_72          ; save array data pointer high byte
        INY                             ; increment index, point to array bound high byte
        PLA                             ; pull array index low byte
        TAX                             ; copy to X
        STA     LAB_64          ; save index low byte to FAC1 mantissa 3
        PLA                             ; pull array index high byte
        STA     LAB_65          ; save index high byte to FAC1 mantissa 4
        CMP     (LAB_5F),Y              ; compare with array bound high byte
        BCC     LAB_D30E                ; if within bounds continue

        BNE     LAB_D308                ; if outside bounds do bad subscript error

                                        ; else high byte was = so test low bytes
        INY                             ; index to array bound low byte
        TXA                             ; get array index low byte
        CMP     (LAB_5F),Y              ; compare with array bound low byte
        BCC     LAB_D30F                ; if within bounds continue

LAB_D308
        JMP     LAB_D245                ; do bad subscript error

LAB_D30B
        JMP     LAB_C435                ; do out of memory error then warm start

LAB_D30E
        INY                             ; index to array bound low byte
LAB_D30F
        LDA     LAB_72          ; get array data pointer high byte
        ORA     LAB_71          ; OR with array data pointer low byte
        CLC                             ; clear carry for either add, carry always clear here
??
        BEQ     LAB_D320                ; if array data pointer = null skip the multiply

        JSR     LAB_D34C                ; compute array size
        TXA                             ; get result low byte
        ADC     LAB_64          ; add index low byte from FAC1 mantissa 3
        TAX                             ; save result low byte
        TYA                             ; get result high byte
        LDY     LAB_22          ; restore index
LAB_D320
        ADC     LAB_65          ; add index high byte from FAC1 mantissa 4
        STX     LAB_71          ; save array data pointer low byte
        DEC     LAB_0B          ; decrement dimensions count
        BNE     LAB_D2F2                ; loop if dimensions still to do

        STA     LAB_72          ; save array data pointer high byte
        LDX     #$05                    ; set default element size
        LDA     LAB_45          ; get variable name 1st byte
        BPL     LAB_D331                ; branch if not string or floating point array
```

```
        DEX                             ; decrement element size, $04
LAB_D331
        LDA     LAB_46          ; get variable name 2nd byte
        BPL     LAB_D337                ; branch if not integer or string

        DEX                             ; decrement element size, $03
        DEX                             ; decrement element size, $02
LAB_D337
        STX     LAB_28          ; save dimension size low byte
        LDA     #$00                    ; clear dimension size high byte
        JSR     LAB_D355                ; compute array size
        TXA                             ; copy array size low byte
        ADC     LAB_58          ; add array data start pointer low byte
        STA     LAB_47          ; save as current variable pointer low byte
        TYA                             ; copy array size high byte
        ADC     LAB_59          ; add array data start pointer high byte
        STA     LAB_48          ; save as current variable pointer high byte
        TAY                             ; copy high byte to Y
        LDA     LAB_47          ; get current variable pointer low byte
                                        ; pointer to element is now in AY
LAB_D34B
        RTS


;******************************************************************************;
;
; compute array size, result in XY

LAB_D34C
        STY     LAB_22          ; save index
        LDA     (LAB_5F),Y              ; get dimension size low byte
        STA     LAB_28          ; save dimension size low byte
        DEY                             ; decrement index
        LDA     (LAB_5F),Y              ; get dimension size high byte
LAB_D355
        STA     LAB_29          ; save dimension size high byte
        LDA     #$10                    ; count = $10 (16 bit multiply)
        STA     LAB_5D          ; save bit count
        LDX     #$00                    ; clear result low byte
        LDY     #$00                    ; clear result high byte
LAB_D35F
        TXA                             ; get result low byte
        ASL                             ; *2
        TAX                             ; save result low byte
        TYA                             ; get result high byte
        ROL                             ; *2
        TAY                             ; save result high byte
        BCS     LAB_D30B                ; if overflow go do "Out of memory" error

        ASL     LAB_71          ; shift element size low byte
        ROL     LAB_72          ; shift element size high byte
        BCC     LAB_D378                ; skip add if no carry

        CLC                             ; else clear carry for add
        TXA                             ; get result low byte
        ADC     LAB_28          ; add dimension size low byte
        TAX                             ; save result low byte
        TYA                             ; get result high byte
        ADC     LAB_29          ; add dimension size high byte
        TAY                             ; save result high byte
```

```
        BCS     LAB_D30B                ; if overflow go do "Out of memory" error

LAB_D378
        DEC     LAB_5D          ; decrement bit count
        BNE     LAB_D35F                ; loop until all done

        RTS


;*******************************************************************************;
;
; perform FRE()

LAB_D37D
        LDA     LAB_0D          ; get data type flag, $FF = string, $00 = numeric
        BEQ     LAB_D384                ; if numeric don't pop the string

        JSR     LAB_D6A6                ; pop string off descriptor stack, or from top of
string
                                        ; space returns with A = length, X=$71=pointer low
byte,
                                        ; Y=$72=pointer high byte

                                        ; FRE(n) was numeric so do this
LAB_D384
        JSR     LAB_D526                ; go do garbage collection
        SEC                             ; set carry for subtract
        LDA     LAB_33          ; get bottom of string space low byte
        SBC     LAB_31          ; subtract end of arrays low byte
        TAY                             ; copy result to Y
        LDA     LAB_34          ; get bottom of string space high byte
        SBC     LAB_32          ; subtract end of arrays high byte


;*******************************************************************************;
;
; convert fixed integer AY to float FAC1

LAB_D391
        LDX     #$00                    ; set type = numeric
        STX     LAB_0D          ; clear data type flag, $FF = string, $00 = numeric
        STA     LAB_62          ; save FAC1 mantissa 1
        STY     LAB_63          ; save FAC1 mantissa 2
        LDX     #$90                    ; set exponent=2^16 (integer)
        JMP     LAB_DC44                ; set exp = X, clear FAC1 3 and 4, normalise and
return


;*******************************************************************************;
;
; perform POS()

LAB_D39E
        SEC                             ; set Cb for read cursor position
        JSR     LAB_FFF0                ; read/set X,Y cursor position
LAB_D3A2
        LDA     #$00                    ; clear high byte
        BEQ     LAB_D391                ; convert fixed integer AY to float FAC1, branch
always
```

```
;**************************************************************************;
;
; check not Direct, used by DEF and INPUT

LAB_D3A6
        LDX     LAB_3A          ; get current line number high byte
        INX                             ; increment it
        BNE     LAB_D34B                ; return if not direct mode

                                        ; else do illegal direct error
        LDX     #$15                    ; error $15, illegal direct error
        .byte   $2C                     ; makes next line BIT LAB_1BA2
LAB_D3AE
        LDX     #$1B                    ; error $1B, undefined function error
        JMP     LAB_C437                ; do error #X then warm start


;**************************************************************************;
;
; perform DEF

LAB_D3B3
        JSR     LAB_D3E1                ; check FNx syntax
        JSR     LAB_D3A6                ; check not direct, back here if ok
        JSR     LAB_CEFA                ; scan for "(", else do syntax error then warm start
        LDA     #$80                    ; set flag for FNx
        STA     LAB_10          ; save subscript/FNx flag
        JSR     LAB_D08B                ; get variable address
        JSR     LAB_CD8D                ; check if source is numeric, else do type mismatch
        JSR     LAB_CEF7                ; scan for ")", else do syntax error then warm start
        LDA     #TK_EQUAL               ; get = token
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        PHA                             ; push next character
        LDA     LAB_48          ; get current variable pointer high byte
        PHA                             ; push it
        LDA     LAB_47          ; get current variable pointer low byte
        PHA                             ; push it
        LDA     LAB_7B          ; get BASIC execute pointer high byte
        PHA                             ; push it
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        PHA                             ; push it
        JSR     LAB_C8F8                ; perform DATA
        JMP     LAB_D44F                ; put execute pointer and variable pointer into
function
                                        ; and return


;**************************************************************************;
;
; check FNx syntax

LAB_D3E1
        LDA     #TK_FN          ; set FN token
        JSR     LAB_CEFF                ; scan for CHR$(A), else do syntax error then warm
start
        ORA     #$80                    ; set FN flag bit
        STA     LAB_10          ; save FN name
        JSR     LAB_D092                ; search for FN variable
        STA     LAB_4E          ; save function pointer low byte
        STY     LAB_4F          ; save function pointer high byte
```

```
        JMP     LAB_CD8D                ; check if source is numeric and return, else do type
                                        ; mismatch


;******************************************************************************;
;
; Evaluate FNx

LAB_D3F4
        JSR     LAB_D3E1                ; check FNx syntax
        LDA     LAB_4F          ; get function pointer high byte
        PHA                             ; push it
        LDA     LAB_4E          ; get function pointer low byte
        PHA                             ; push it
        JSR     LAB_CEF1                ; evaluate expression within parentheses
        JSR     LAB_CD8D                ; check if source is numeric, else do type mismatch
        PLA                             ; pop function pointer low byte
        STA     LAB_4E          ; restore it
        PLA                             ; pop function pointer high byte
        STA     LAB_4F          ; restore it
        LDY     #$02                    ; index to variable pointer high byte
        LDA     (LAB_4E),Y              ; get variable address low byte
        STA     LAB_47          ; save current variable pointer low byte
        TAX                             ; copy address low byte
        INY                             ; index to variable address high byte
        LDA     (LAB_4E),Y              ; get variable pointer high byte
        BEQ     LAB_D3AE                ; if high byte zero go do undefined function error

        STA     LAB_48          ; save current variable pointer high byte
        INY                             ; index to mantissa 3

                                        ; now stack the function variable value before use
LAB_D418
        LDA     (LAB_47),Y              ; get byte from variable
        PHA                             ; stack it
        DEY                             ; decrement index
        BPL     LAB_D418                ; loop until variable stacked

        LDY     LAB_48          ; get current variable pointer high byte
        JSR     LAB_DBD4                ; pack FAC1 into (XY)
        LDA     LAB_7B          ; get BASIC execute pointer high byte
        PHA                             ; push it
        LDA     LAB_7A          ; get BASIC execute pointer low byte
        PHA                             ; push it
        LDA     (LAB_4E),Y              ; get function execute pointer low byte
        STA     LAB_7A          ; save BASIC execute pointer low byte
        INY                             ; index to high byte
        LDA     (LAB_4E),Y              ; get function execute pointer high byte
        STA     LAB_7B          ; save BASIC execute pointer high byte
        LDA     LAB_48          ; get current variable pointer high byte
        PHA                             ; push it
        LDA     LAB_47          ; get current variable pointer low byte
        PHA                             ; push it
        JSR     LAB_CD8A                ; evaluate expression and check is numeric, else do
                                        ; type mismatch
        PLA                             ; pull variable address low byte
        STA     LAB_4E          ; save variable address low byte
        PLA                             ; pull variable address high byte
        STA     LAB_4F          ; save variable address high byte
        JSR     LAB_0079                ; scan memory
        BEQ     LAB_D449                ; if null (should be [EOL] marker) continue
```

```
        JMP     LAB_CF08                ; else syntax error then warm start

; restore BASIC execute pointer and function variable from stack

LAB_D449
        PLA                             ; pull BASIC execute pointer low byte
        STA     LAB_7A          ; save BASIC execute pointer low byte
        PLA                             ; pull BASIC execute pointer high byte
        STA     LAB_7B          ; save BASIC execute pointer high byte

; put execute pointer and variable pointer into function

LAB_D44F
        LDY     #$00                    ; clear index
        PLA                             ; pull BASIC execute pointer low byte
        STA     (LAB_4E),Y              ; save to function
        PLA                             ; pull BASIC execute pointer high byte
        INY                             ; increment index
        STA     (LAB_4E),Y              ; save to function
        PLA                             ; pull current variable address low byte
        INY                             ; increment index
        STA     (LAB_4E),Y              ; save to function
        PLA                             ; pull current variable address high byte
        INY                             ; increment index
        STA     (LAB_4E),Y              ; save to function
        PLA                             ; pull ??
        INY                             ; increment index
        STA     (LAB_4E),Y              ; save to function
        RTS


;******************************************************************************;
;
; perform STR$()

LAB_D465
        JSR     LAB_CD8D                ; check if source is numeric, else do type mismatch
        LDY     #$00                    ; set string index
        JSR     LAB_DDDF                ; convert FAC1 to string
        PLA                             ; dump return address (skip type check)
        PLA                             ; dump return address (skip type check)
LAB_D46F
        LDA     #<LAB_00FF              ; set result string low pointer
        LDY     #>LAB_00FF              ; set result string high pointer
        BEQ     LAB_D487                ; print null terminated string to utility pointer


;******************************************************************************;
;
; do string vector
; copy descriptor pointer and make string space A bytes long

LAB_D475
        LDX     LAB_64          ; get descriptor pointer low byte
        LDY     LAB_65          ; get descriptor pointer high byte
        STX     LAB_50          ; save descriptor pointer low byte
        STY     LAB_51          ; save descriptor pointer high byte


;******************************************************************************;
```

```
;
; make string space A bytes long

LAB_D47D
        JSR     LAB_D4F4                ; make space in string memory for string A long
        STX     LAB_62          ; save string pointer low byte
        STY     LAB_63          ; save string pointer high byte
        STA     LAB_61          ; save length
        RTS


;******************************************************************************;
;
; scan, set up string
; print " terminated string to utility pointer

LAB_D487
        LDX     #$22                    ; set terminator to "
        STX     LAB_07          ; set search character, terminator 1
        STX     LAB_08          ; set terminator 2

; print search or alternate terminated string to utility pointer
; source is AY

LAB_D48D
        STA     LAB_6F          ; store string start low byte
        STY     LAB_70          ; store string start high byte
        STA     LAB_62          ; save string pointer low byte
        STY     LAB_63          ; save string pointer high byte
        LDY     #$FF                    ; set length to -1
LAB_D497
        INY                             ; increment length
        LDA     (LAB_6F),Y              ; get byte from string
        BEQ     LAB_D4A8                ; exit loop if null byte [EOS]

        CMP     LAB_07          ; compare with search character, terminator 1
        BEQ     LAB_D4A4                ; branch if terminator

        CMP     LAB_08          ; compare with terminator 2
        BNE     LAB_D497                ; loop if not terminator 2

LAB_D4A4
        CMP     #$22                    ; compare with "
        BEQ     LAB_D4A9                ; branch if " (carry set if = !)

LAB_D4A8
        CLC                             ; clear carry for add (only if [EOL] terminated
string)
LAB_D4A9
        STY     LAB_61          ; save length in FAC1 exponent
        TYA                             ; copy length to A
        ADC     LAB_6F          ; add string start low byte
        STA     LAB_71          ; save string end low byte
        LDX     LAB_70          ; get string start high byte
        BCC     LAB_D4B5                ; if no low byte overflow skip the high byte increment

        INX                             ; else increment high byte
LAB_D4B5
        STX     LAB_72          ; save string end high byte
        LDA     LAB_70          ; get string start high byte
        BEQ     LAB_D4BF                ; branch if in utility area
```

```
        CMP     #$02                    ; compare with input buffer memory high byte
        BNE     LAB_D4CA                ; branch if not in input buffer memory

                                        ; string in input buffer or utility area, move to
string
                                        ; memory
LAB_D4BF
        TYA                             ; copy length to A
        JSR     LAB_D475                ; copy descriptor pointer and make string space A
bytes long
        LDX     LAB_6F          ; get string start low byte
        LDY     LAB_70          ; get string start high byte
        JSR     LAB_D688                ; store string A bytes long from XY to utility pointer

; check for space on descriptor stack then ...
; put string address and length on descriptor stack and update stack pointers

LAB_D4CA
        LDX     LAB_16          ; get descriptor stack pointer
        CPX     #$22                    ; compare with max+1
        BNE     LAB_D4D5                ; branch if space on string stack

                                        ; else do string too complex error
        LDX     #$19                    ; error $19, string too complex error
LAB_D4D2
        JMP     LAB_C437                ; do error #X then warm start

; put string address and length on descriptor stack and update stack pointers

LAB_D4D5
        LDA     LAB_61          ; get string length
        STA     LAB_00,X                ; put on string stack
        LDA     LAB_62          ; get string pointer low byte
        STA     LAB_01,X                ; put on string stack
        LDA     LAB_63          ; get string pointer high byte
        STA     LAB_02,X                ; put on string stack
        LDY     #$00                    ; clear Y
        STX     LAB_64          ; save string descriptor pointer low byte
        STY     LAB_65          ; save string descriptor pointer high byte, always $00
        STY     LAB_70          ; clear FAC1 rounding byte
        DEY                             ; Y = $FF
        STY     LAB_0D          ; save data type flag, $FF = string
        STX     LAB_17          ; save current descriptor stack item pointer low byte
        INX                             ; update stack pointer
        INX                             ; update stack pointer
        INX                             ; update stack pointer
        STX     LAB_16          ; set new descriptor stack pointer
        RTS

; make space in string memory for string A long
; return X = pointer low byte, Y = pointer high byte

LAB_D4F4
        LSR     LAB_0F          ; clear garbage collected flag (b7)

                                        ; make space for string A long
LAB_D4F6
        PHA                             ; save string length
        EOR     #$FF                    ; complement it
        SEC                             ; set carry for subtract, two's complement add
```

```
        ADC     LAB_33          ; add bottom of string space low byte, subtract length
        LDY     LAB_34          ; get bottom of string space high byte
        BCS     LAB_D501                ; skip decrement if no underflow

        DEY                             ; decrement bottom of string space high byte
LAB_D501
        CPY     LAB_32          ; compare with end of arrays high byte
        BCC     LAB_D516                ; do out of memory error if less

        BNE     LAB_D50B                ; if not = skip next test

        CMP     LAB_31          ; compare with end of arrays low byte
        BCC     LAB_D516                ; do out of memory error if less

LAB_D50B
        STA     LAB_33          ; save bottom of string space low byte
        STY     LAB_34          ; save bottom of string space high byte
        STA     LAB_35          ; save string utility ptr low byte
        STY     LAB_36          ; save string utility ptr high byte
        TAX                             ; copy low byte to X
        PLA                             ; get string length back
        RTS

LAB_D516
        LDX     #$10                    ; error code $10, out of memory error
        LDA     LAB_0F          ; get garbage collected flag
        BMI     LAB_D4D2                ; if set then do error code X

        JSR     LAB_D526                ; else go do garbage collection
        LDA     #$80                    ; flag for garbage collected
        STA     LAB_0F          ; set garbage collected flag
        PLA                             ; pull length
        BNE     LAB_D4F6                ; go try again (loop always, length should never be =
$00)


;**************************************************************************;
;
; garbage collection routine

LAB_D526
        LDX     LAB_37          ; get end of memory low byte
        LDA     LAB_38          ; get end of memory high byte

; re-run routine from last ending

LAB_D52A
        STX     LAB_33          ; set bottom of string space low byte
        STA     LAB_34          ; set bottom of string space high byte
        LDY     #$00                    ; clear index
        STY     LAB_4F          ; clear working pointer high byte
        STY     LAB_4E          ; clear working pointer low byte
        LDA     LAB_31          ; get end of arrays low byte
        LDX     LAB_32          ; get end of arrays high byte
        STA     LAB_5F          ; save as highest uncollected string pointer low byte
        STX     LAB_60          ; save as highest uncollected string pointer high byte
        LDA     #LAB_19         ; set descriptor stack pointer
        LDX     #$00                    ; clear X
        STA     LAB_22          ; save descriptor stack pointer low byte
        STX     LAB_23          ; save descriptor stack pointer high byte ($00)
LAB D544
```

```
        CMP     LAB_16          ; compare with descriptor stack pointer
        BEQ     LAB_D54D                ; branch if =

        JSR     LAB_D5C7                ; check string salvageability
        BEQ     LAB_D544                ; loop always

                                        ; done stacked strings, now do string variables
LAB_D54D
        LDA     #$07                    ; set step size = $07, collecting variables
        STA     LAB_53          ; save garbage collection step size
        LDA     LAB_2D          ; get start of variables low byte
        LDX     LAB_2E          ; get start of variables high byte
        STA     LAB_22          ; save as pointer low byte
        STX     LAB_23          ; save as pointer high byte
LAB_D559
        CPX     LAB_30          ; compare end of variables high byte,
                                        ; start of arrays high byte
        BNE     LAB_D561                ; branch if no high byte match

        CMP     LAB_2F          ; else compare end of variables low byte,
                                        ; start of arrays low byte
        BEQ     LAB_D566                ; branch if = variable memory end

LAB_D561
        JSR     LAB_D5BD                ; check variable salvageability
        BEQ     LAB_D559                ; loop always

                                        ; done string variables, now do string arrays
LAB_D566
        STA     LAB_58          ; save start of arrays low byte as working pointer
        STX     LAB_59          ; save start of arrays high byte as working pointer
        LDA     #$03                    ; set step size, collecting descriptors
        STA     LAB_53          ; save step size
LAB_D56E
        LDA     LAB_58          ; get pointer low byte
        LDX     LAB_59          ; get pointer high byte
LAB_D572
        CPX     LAB_32          ; compare with end of arrays high byte
        BNE     LAB_D57D                ; branch if not at end

        CMP     LAB_31          ; else compare with end of arrays low byte
        BNE     LAB_D57D                ; branch if not at end

        JMP     LAB_D606                ; collect string, tidy up and exit if at end ??

LAB_D57D
        STA     LAB_22          ; save pointer low byte
        STX     LAB_23          ; save pointer high byte
        LDY     #$00                    ; set index
        LDA     (LAB_22),Y              ; get array name first byte
        TAX                             ; copy it
        INY                             ; increment index
        LDA     (LAB_22),Y              ; get array name second byte
        PHP                             ; push the flags
        INY                             ; increment index
        LDA     (LAB_22),Y              ; get array size low byte
        ADC     LAB_58          ; add start of this array low byte
        STA     LAB_58          ; save start of next array low byte
        INY                             ; increment index
        LDA     (LAB_22),Y              ; get array size high byte
        ADC     LAB_59          ; add start of this array high byte
```

```
        STA     LAB_59          ; save start of next array high byte
        PLP                             ; restore the flags
        BPL     LAB_D56E                ; skip if not string array

; was possibly string array so ...

        TXA                             ; get name first byte back
        BMI     LAB_D56E                ; skip if not string array

        INY                             ; increment index
        LDA     (LAB_22),Y              ; get # of dimensions
        LDY     #$00                    ; clear index
        ASL                             ; *2
        ADC     #$05                    ; +5 (array header size)
        ADC     LAB_22          ; add pointer low byte
        STA     LAB_22          ; save pointer low byte
        BCC     LAB_D5AE                ; if no rollover skip the high byte increment

        INC     LAB_23          ; else increment pointer hgih byte
LAB_D5AE
        LDX     LAB_23          ; get pointer high byte
LAB_D5B0
        CPX     LAB_59          ; compare pointer high byte with end of this array high byte
        BNE     LAB_D5B8                ; branch if not there yet

        CMP     LAB_58          ; compare pointer low byte with end of this array low byte
        BEQ     LAB_D572                ; if at end of this array go check next array

LAB_D5B8
        JSR     LAB_D5C7                ; check string salvageability
        BEQ     LAB_D5B0                ; loop

; check variable salvageability

LAB_D5BD
        LDA     (LAB_22),Y              ; get variable name first byte
        BMI     LAB_D5F6                ; add step and exit if not string

        INY                             ; increment index
        LDA     (LAB_22),Y              ; get variable name second byte
        BPL     LAB_D5F6                ; add step and exit if not string

        INY                             ; increment index

; check string salvageability

LAB_D5C7
        LDA     (LAB_22),Y              ; get string length
        BEQ     LAB_D5F6                ; add step and exit if null string

        INY                             ; increment index
        LDA     (LAB_22),Y              ; get string pointer low byte
        TAX                             ; copy to X
        INY                             ; increment index
        LDA     (LAB_22),Y              ; get string pointer high byte
        CMP     LAB_34          ; compare string pointer high byte with bottom of string
                                        ; space high byte
        BCC     LAB_D5DC                ; if bottom of string space greater go test against
highest
                                        ; uncollected string
```

```
        BNE     LAB_D5F6                ; if bottom of string space less string has been
collected
                                        ; so go update pointers, step to next and return

                                        ; high bytes were equal so test low bytes
        CPX     LAB_33          ; compare string pointer low byte with bottom of string
                                        ; space low byte
        BCS     LAB_D5F6                ; if bottom of string space less string has been
collected
                                        ; so go update pointers, step to next and return

                                        ; else test string against highest uncollected string
so far
LAB_D5DC
        CMP     LAB_60          ; compare string pointer high byte with highest uncollected
                                        ; string high byte
        BCC     LAB_D5F6                ; if highest uncollected string is greater then go
update
                                        ; pointers, step to next and return

        BNE     LAB_D5E6                ; if highest uncollected string is less then go set
this
                                        ; string as highest uncollected so far

                                        ; high bytes were equal so test low bytes
        CPX     LAB_5F          ; compare string pointer low byte with highest uncollected
                                        ; string low byte
        BCC     LAB_D5F6                ; if highest uncollected string is greater then go
update
                                        ; pointers, step to next and return

                                        ; else set current string as highest uncollected
string
LAB_D5E6
        STX     LAB_5F          ; save string pointer low byte as highest uncollected string
                                        ; low byte
        STA     LAB_60          ; save string pointer high byte as highest uncollected
                                        ; string high byte
        LDA     LAB_22          ; get descriptor pointer low byte
        LDX     LAB_23          ; get descriptor pointer high byte
        STA     LAB_4E          ; save working pointer high byte
        STX     LAB_4F          ; save working pointer low byte
        LDA     LAB_53          ; get step size
        STA     LAB_55          ; copy step size
LAB_D5F6
        LDA     LAB_53          ; get step size
        CLC                             ; clear carry for add
        ADC     LAB_22          ; add pointer low byte
        STA     LAB_22          ; save pointer low byte
        BCC     LAB_D601                ; if no rollover skip the high byte increment

        INC     LAB_23          ; else increment pointer high byte
LAB_D601
        LDX     LAB_23          ; get pointer high byte
        LDY     #$00                    ; flag not moved
        RTS


;******************************************************************************;
;
; collect string
```

```
LAB_D606
        LDA     LAB_4F          ; get working pointer low byte
        ORA     LAB_4E          ; OR working pointer high byte
        BEQ     LAB_D601                ; exit if nothing to collect

        LDA     LAB_55          ; get copied step size
        AND     #$04                    ; mask step size, $04 for variables, $00 for array or
stack
        LSR                             ; >> 1
        TAY                             ; copy to index
        STA     LAB_55          ; save offset to descriptor start
        LDA     (LAB_4E),Y              ; get string length low byte
        ADC     LAB_5F          ; add string start low byte
        STA     LAB_5A          ; set block end low byte
        LDA     LAB_60          ; get string start high byte
        ADC     #$00                    ; add carry
        STA     LAB_5B          ; set block end high byte
        LDA     LAB_33          ; get bottom of string space low byte
        LDX     LAB_34          ; get bottom of string space high byte
        STA     LAB_58          ; save destination end low byte
        STX     LAB_59          ; save destination end high byte
        JSR     LAB_C3BF                ; open up space in memory, don't set array end. this
                                        ; copies the string from where it is to the end of the
                                        ; uncollected string memory
        LDY     LAB_55          ; restore offset to descriptor start
        INY                             ; increment index to string pointer low byte
        LDA     LAB_58          ; get new string pointer low byte
        STA     (LAB_4E),Y              ; save new string pointer low byte
        TAX                             ; copy string pointer low byte
        INC     LAB_59          ; increment new string pointer high byte
        LDA     LAB_59          ; get new string pointer high byte
        INY                             ; increment index to string pointer high byte
        STA     (LAB_4E),Y              ; save new string pointer high byte
        JMP     LAB_D52A                ; re-run routine from last ending, XA holds new bottom
                                        ; of string memory pointer


;******************************************************************************;
;
; concatenate
; add strings, the first string is in the descriptor, the second string is in line

LAB_D63D
        LDA     LAB_65          ; get descriptor pointer high byte
        PHA                             ; put on stack
        LDA     LAB_64          ; get descriptor pointer low byte
        PHA                             ; put on stack
        JSR     LAB_CE83                ; get value from line
        JSR     LAB_CD8F                ; check if source is string, else do type mismatch
        PLA                             ; get descriptor pointer low byte back
        STA     LAB_6F          ; set pointer low byte
        PLA                             ; get descriptor pointer high byte back
        STA     LAB_70          ; set pointer high byte
        LDY     #$00                    ; clear index
        LDA     (LAB_6F),Y              ; get length of first string from descriptor
        CLC                             ; clear carry for add
        ADC     (LAB_64),Y              ; add length of second string
        BCC     LAB_D65D                ; if no overflow continue

        LDX     #$17                    ; else error $17, string too long error
```

```
        JMP     LAB_C437                ; do error #X then warm start

LAB_D65D
        JSR     LAB_D475                ; copy descriptor pointer and make string space A
bytes long
        JSR     LAB_D67A                ; copy string from descriptor to utility pointer
        LDA     LAB_50          ; get descriptor pointer low byte
        LDY     LAB_51          ; get descriptor pointer high byte
        JSR     LAB_D6AA                ; pop (YA) descriptor off stack or from top of string
space
                                        ; returns with A = length, X = pointer low byte,
                                        ; Y = pointer high byte
        JSR     LAB_D68C                ; store string from pointer to utility pointer
        LDA     LAB_6F          ; get descriptor pointer low byte
        LDY     LAB_70          ; get descriptor pointer high byte
        JSR     LAB_D6AA                ; pop (YA) descriptor off stack or from top of string
space
                                        ; returns with A = length, X = pointer low byte,
                                        ; Y = pointer high byte
        JSR     LAB_D4CA                ; check space on descriptor stack then put string
address
                                        ; and length on descriptor stack and update stack
pointers
        JMP     LAB_CDB8                ; continue evaluation


;**************************************************************************;
;
; copy string from descriptor to utility pointer

LAB_D67A
        LDY     #$00                    ; clear index
        LDA     (LAB_6F),Y              ; get string length
        PHA                             ; save it
        INY                             ; increment index
        LDA     (LAB_6F),Y              ; get string pointer low byte
        TAX                             ; copy to X
        INY                             ; increment index
        LDA     (LAB_6F),Y              ; get string pointer high byte
        TAY                             ; copy to Y
        PLA                             ; get length back
LAB_D688
        STX     LAB_22          ; save string pointer low byte
        STY     LAB_23          ; save string pointer high byte


;**************************************************************************;
;
; store string from pointer to utility pointer

LAB_D68C
        TAY                             ; copy length as index
        BEQ     LAB_D699                ; branch if null string

        PHA                             ; save length
LAB_D690
        DEY                             ; decrement length/index
        LDA     (LAB_22),Y              ; get byte from string
        STA     (LAB_35),Y              ; save byte to destination
        TYA                             ; copy length/index
        BNE     LAB_D690                ; loop if not all done yet
```

```
        PLA                             ; restore length
LAB_D699
        CLC                             ; clear carry for add
        ADC     LAB_35          ; add string utility ptr low byte
        STA     LAB_35          ; save string utility ptr low byte
        BCC     LAB_D6A2                ; if no rollover skip the high byte increment

        INC     LAB_36          ; increment string utility ptr high byte
LAB_D6A2
        RTS


;******************************************************************************;
;
; evaluate string

LAB_D6A3
        JSR     LAB_CD8F                ; check if source is string, else do type mismatch

; pop string off descriptor stack, or from top of string space
; returns with A = length, X = pointer low byte, Y = pointer high byte

LAB_D6A6
        LDA     LAB_64          ; get descriptor pointer low byte
        LDY     LAB_65          ; get descriptor pointer high byte

; pop (YA) descriptor off stack or from top of string space
; returns with A = length, X = pointer low byte, Y = pointer high byte

LAB_D6AA
        STA     LAB_22          ; save string pointer low byte
        STY     LAB_23          ; save string pointer high byte
        JSR     LAB_D6DB                ; clean descriptor stack, YA = pointer
        PHP                             ; save status flags
        LDY     #$00                    ; clear index
        LDA     (LAB_22),Y              ; get length from string descriptor
        PHA                             ; put on stack
        INY                             ; increment index
        LDA     (LAB_22),Y              ; get string pointer low byte from descriptor
        TAX                             ; copy to X
        INY                             ; increment index
        LDA     (LAB_22),Y              ; get string pointer high byte from descriptor
        TAY                             ; copy to Y
        PLA                             ; get string length back
        PLP                             ; restore status
        BNE     LAB_D6D6                ; branch if pointer <> last_sl,last_sh

        CPY     LAB_34          ; compare with bottom of string space high byte
        BNE     LAB_D6D6                ; branch if <>

        CPX     LAB_33          ; else compare with bottom of string space low byte
        BNE     LAB_D6D6                ; branch if <>

        PHA                             ; save string length
        CLC                             ; clear carry for add
        ADC     LAB_33          ; add bottom of string space low byte
        STA     LAB_33          ; set bottom of string space low byte
        BCC     LAB_D6D5                ; skip increment if no overflow

        INC     LAB 34          ; increment bottom of string space high byte
```

```
LAB_D6D5
        PLA                             ; restore string length
LAB_D6D6
        STX     LAB_22          ; save string pointer low byte
        STY     LAB_23          ; save string pointer high byte
        RTS


;******************************************************************************;
;
; clean descriptor stack, YA = pointer
; checks if AY is on the descriptor stack, if so does a stack discard

LAB_D6DB
        CPY     LAB_18          ; compare high byte with current descriptor stack item
                                ; pointer high byte
        BNE     LAB_D6EB                ; exit if <>

        CMP     LAB_17          ; compare low byte with current descriptor stack item
                                ; pointer low byte
        BNE     LAB_D6EB                ; exit if <>

        STA     LAB_16          ; set descriptor stack pointer
        SBC     #$03                    ; update last string pointer low byte
        STA     LAB_17          ; save current descriptor stack item pointer low byte
        LDY     #$00                    ; clear high byte
LAB_D6EB
        RTS


;******************************************************************************;
;
; perform CHR$()

LAB_D6EC
        JSR     LAB_D7A1                ; evaluate byte expression, result in X
        TXA                             ; copy to A
        PHA                             ; save character
        LDA     #$01                    ; string is single byte
        JSR     LAB_D47D                ; make string space A bytes long
        PLA                             ; get character back
        LDY     #$00                    ; clear index
        STA     (LAB_62),Y              ; save byte in string - byte IS string!
        PLA                             ; dump return address (skip type check)
        PLA                             ; dump return address (skip type check)
        JMP     LAB_D4CA                ; check space on descriptor stack then put string
address
                                        ; and length on descriptor stack and update stack
pointers


;******************************************************************************;
;
; perform LEFT$()

LAB_D700
        JSR     LAB_D761                ; pull string data and byte parameter from stack
                                        ; return pointer in descriptor, byte in A (and X), Y=0
        CMP     (LAB_50),Y              ; compare byte parameter with string length
        TYA                             ; clear A
LAB D706
```

```
        BCC     LAB_D70C                ; branch if string length > byte parameter

        LDA     (LAB_50),Y              ; else make parameter = length
        TAX                             ; copy to byte parameter copy
        TYA                             ; clear string start offset
LAB_D70C
        PHA                             ; save string start offset
LAB_D70D
        TXA                             ; copy byte parameter (or string length if <)
LAB_D70E
        PHA                             ; save string length
        JSR     LAB_D47D                ; make string space A bytes long
        LDA     LAB_50          ; get descriptor pointer low byte
        LDY     LAB_51          ; get descriptor pointer high byte
        JSR     LAB_D6AA                ; pop (YA) descriptor off stack or from top of string
space
                                        ; returns with A = length, X = pointer low byte,
                                        ; Y = pointer high byte
        PLA                             ; get string length back
        TAY                             ; copy length to Y
        PLA                             ; get string start offset back
        CLC                             ; clear carry for add
        ADC     LAB_22          ; add start offset to string start pointer low byte
        STA     LAB_22          ; save string start pointer low byte
        BCC     LAB_D725                ; if no overflow skip the high byte increment

        INC     LAB_23          ; else increment string start pointer high byte
LAB_D725
        TYA                             ; copy length to A
        JSR     LAB_D68C                ; store string from pointer to utility pointer
        JMP     LAB_D4CA                ; check space on descriptor stack then put string
address
                                        ; and length on descriptor stack and update stack
pointers


;******************************************************************************;
;
; perform RIGHT$()

LAB_D72C
        JSR     LAB_D761                ; pull string data and byte parameter from stack
                                        ; return pointer in descriptor, byte in A (and X), Y=0
        CLC                             ; clear carry for add-1
        SBC     (LAB_50),Y              ; subtract string length
        EOR     #$FF                    ; invert it (A=LEN(expression$)-1)
        JMP     LAB_D706                ; go do rest of LEFT$()


;******************************************************************************;
;
; perform MID$()

LAB_D737
        LDA     #$FF                    ; set default length = 255
        STA     LAB_65          ; save default length
        JSR     LAB_0079                ; scan memory
        CMP     #$29                    ; compare with ")"
        BEQ     LAB_D748                ; branch if = ")" (skip second byte get)

        JSR     LAB CEFD                ; scan for ",", else do syntax error then warm start
```

```
        JSR     LAB_D79E                ; get byte parameter
LAB_D748
        JSR     LAB_D761                ; pull string data and byte parameter from stack
                                        ; return pointer in descriptor, byte in A (and X), Y=0
        BEQ     LAB_D798                ; if null do illegal quantity error then warm start

        DEX                             ; decrement start index
        TXA                             ; copy to A
        PHA                             ; save string start offset
        CLC                             ; clear carry for sub-1
        LDX     #$00                    ; clear output string length
        SBC     (LAB_50),Y              ; subtract string length
        BCS     LAB_D70D                ; if start>string length go do null string

        EOR     #$FF                    ; complement -length
        CMP     LAB_65          ; compare byte parameter
        BCC     LAB_D70E                ; if length>remaining string go do RIGHT$

        LDA     LAB_65          ; get length byte
        BCS     LAB_D70E                ; go do string copy, branch always


;******************************************************************************;
;
; pull string data and byte parameter from stack
; return pointer in descriptor, byte in A (and X), Y=0

LAB_D761
        JSR     LAB_CEF7                ; scan for ")", else do syntax error then warm start
        PLA                             ; pull return address low byte
        TAY                             ; save return address low byte
        PLA                             ; pull return address high byte
        STA     LAB_55          ; save return address high byte
        PLA                             ; dump call to function vector low byte
        PLA                             ; dump call to function vector high byte
        PLA                             ; pull byte parameter
        TAX                             ; copy byte parameter to X
        PLA                             ; pull string pointer low byte
        STA     LAB_50          ; save it
        PLA                             ; pull string pointer high byte
        STA     LAB_51          ; save it
        LDA     LAB_55          ; get return address high byte
        PHA                             ; back on stack
        TYA                             ; get return address low byte
        PHA                             ; back on stack
        LDY     #$00                    ; clear index
        TXA                             ; copy byte parameter
        RTS


;******************************************************************************;
;
; perform LEN()

LAB_D77C
        JSR     LAB_D782                ; evaluate string, get length in A (and Y)
        JMP     LAB_D3A2                ; convert Y to byte in FAC1 and return


;******************************************************************************;
;
```

```
; evaluate string, get length in Y

LAB_D782
        JSR     LAB_D6A3                ; evaluate string
        LDX     #$00                    ; set data type = numeric
        STX     LAB_0D          ; clear data type flag, $FF = string, $00 = numeric
        TAY                             ; copy length to Y
        RTS


;******************************************************************************;
;
; perform ASC()

LAB_D78B
        JSR     LAB_D782                ; evaluate string, get length in A (and Y)
        BEQ     LAB_D798                ; if null do illegal quantity error then warm start

        LDY     #$00                    ; set index to first character
        LDA     (LAB_22),Y              ; get byte
        TAY                             ; copy to Y
        JMP     LAB_D3A2                ; convert Y to byte in FAC1 and return


;******************************************************************************;
;
; do illegal quantity error then warm start

LAB_D798
        JMP     LAB_D248                ; do illegal quantity error then warm start


;******************************************************************************;
;
; scan and get byte parameter

LAB_D79B
        JSR     LAB_0073                ; increment and scan memory

; get byte parameter

LAB_D79E
        JSR     LAB_CD8A                ; evaluate expression and check is numeric, else do
                                        ; type mismatch

; evaluate byte expression, result in X

LAB_D7A1
        JSR     LAB_D1B8                ; evaluate integer expression, sign check

        LDX     LAB_64          ; get FAC1 mantissa 3
        BNE     LAB_D798                ; if not null do illegal quantity error then warm
start

        LDX     LAB_65          ; get FAC1 mantissa 4
        JMP     LAB_0079                ; scan memory and return


;******************************************************************************;
;
; perform VAL()
```

```
LAB_D7AD
        JSR     LAB_D782                ; evaluate string, get length in A (and Y)
        BNE     LAB_D7B5                ; if not a null string go evaluate it

                                        ; string was null so set result = $00
        JMP     LAB_D8F7                ; clear FAC1 exponent and sign and return

LAB_D7B5
        LDX     LAB_7A          ; get BASIC execute pointer low byte
        LDY     LAB_7B          ; get BASIC execute pointer high byte
        STX     LAB_71          ; save BASIC execute pointer low byte
        STY     LAB_72          ; save BASIC execute pointer high byte
        LDX     LAB_22          ; get string pointer low byte
        STX     LAB_7A          ; save BASIC execute pointer low byte
        CLC                             ; clear carry for add
        ADC     LAB_22          ; add string length
        STA     LAB_24          ; save string end low byte
        LDX     LAB_23          ; get string pointer high byte
        STX     LAB_7B          ; save BASIC execute pointer high byte
        BCC     LAB_D7CD                ; if no rollover skip the high byte increment

        INX                             ; increment string end high byte
LAB_D7CD
        STX     LAB_25          ; save string end high byte
        LDY     #$00                    ; set index to $00
        LDA     (LAB_24),Y              ; get string end byte
        PHA                             ; push it
        TYA                             ; clear A
        STA     (LAB_24),Y              ; terminate string with $00
        JSR     LAB_0079                ; scan memory
        JSR     LAB_DCF3                ; get FAC1 from string
        PLA                             ; restore string end byte
        LDY     #$00                    ; clear index
        STA     (LAB_24),Y              ; put string end byte back

; restore BASIC execute pointer from temp

LAB_D7E2
        LDX     LAB_71          ; get BASIC execute pointer low byte back
        LDY     LAB_72          ; get BASIC execute pointer high byte back
        STX     LAB_7A          ; save BASIC execute pointer low byte
        STY     LAB_7B          ; save BASIC execute pointer high byte
        RTS


;******************************************************************************;
;
; get parameters for POKE/WAIT

LAB_D7EB
        JSR     LAB_CD8A                ; evaluate expression and check is numeric, else do
                                        ; type mismatch
        JSR     LAB_D7F7                ; convert FAC_1 to integer in temporary integer
LAB_D7F1
        JSR     LAB_CEFD                ; scan for ",", else do syntax error then warm start
        JMP     LAB_D79E                ; get byte parameter and return


;******************************************************************************;
;
```

```
; convert FAC_1 to integer in temporary integer

LAB_D7F7
        LDA     LAB_66          ; get FAC1 sign
        BMI     LAB_D798                ; if -ve do illegal quantity error then warm start

        LDA     LAB_61          ; get FAC1 exponent
        CMP     #$91                    ; compare with exponent = 2^16
        BCS     LAB_D798                ; if >= do illegal quantity error then warm start

        JSR     LAB_DC9B                ; convert FAC1 floating to fixed
        LDA     LAB_64          ; get FAC1 mantissa 3
        LDY     LAB_65          ; get FAC1 mantissa 4
        STY     LAB_14          ; save temporary integer low byte
        STA     LAB_15          ; save temporary integer high byte
        RTS


;******************************************************************************;
;
; perform PEEK()

LAB_D80D
        LDA     LAB_15          ; get line number high byte
        PHA                             ; save line number high byte
        LDA     LAB_14          ; get line number low byte
        PHA                             ; save line number low byte
        JSR     LAB_D7F7                ; convert FAC_1 to integer in temporary integer
        LDY     #$00                    ; clear index
        LDA     (LAB_14),Y              ; read byte
        TAY                             ; copy byte to A
        PLA                             ; pull byte
        STA     LAB_14          ; restore line number low byte
        PLA                             ; pull byte
        STA     LAB_15          ; restore line number high byte
        JMP     LAB_D3A2                ; convert Y to byte in FAC_1 and return


;******************************************************************************;
;
; perform POKE

LAB_D824
        JSR     LAB_D7EB                ; get parameters for POKE/WAIT
        TXA                             ; copy byte to A
        LDY     #$00                    ; clear index
        STA     (LAB_14),Y              ; write byte
        RTS


;******************************************************************************;
;
; perform WAIT

LAB_D82D
        JSR     LAB_D7EB                ; get parameters for POKE/WAIT
        STX     LAB_49          ; save byte
        LDX     #$00                    ; clear mask
        JSR     LAB_0079                ; scan memory
        BEQ     LAB_D83C                ; skip if no third argument
```

```
        JSR     LAB_D7F1                ; scan for "," and get byte, else syntax error then
                                        ; warm start
LAB_D83C
        STX     LAB_4A          ; save EOR argument
        LDY     #$00                    ; clear index
LAB_D840
        LDA     (LAB_14),Y              ; get byte via temporary integer        (address)
        EOR     LAB_4A          ; EOR with second argument              (mask)
        AND     LAB_49          ; AND with first argument               (byte)
        BEQ     LAB_D840                ; loop if result is zero

LAB_D848
        RTS


;******************************************************************************;
;
; add 0.5 to FAC1 (round FAC1)

LAB_D849
        LDA     #<LAB_DF11              ; set 0.5 pointer low byte
        LDY     #>LAB_DF11              ; set 0.5 pointer high byte
        JMP     LAB_D867                ; add (AY) to FAC1


;******************************************************************************;
;
; perform subtraction, FAC1 from (AY)

LAB_D850
        JSR     LAB_DA8C                ; unpack memory (AY) into FAC2

; perform subtraction, FAC1 from FAC2

LAB_D853
        LDA     LAB_66          ; get FAC1 sign (b7)
        EOR     #$FF                    ; complement it
        STA     LAB_66          ; save FAC1 sign (b7)
        EOR     LAB_6E          ; EOR with FAC2 sign (b7)
        STA     LAB_6F          ; save sign compare (FAC1 EOR FAC2)
        LDA     LAB_61          ; get FAC1 exponent
        JMP     LAB_D86A                ; add FAC2 to FAC1 and return

LAB_D862
        JSR     LAB_D999                ; shift FACX A times right (>8 shifts)
        BCC     LAB_D8A3                ; go subtract the mantissas, branch always


;******************************************************************************;
;
; add (AY) to FAC1

LAB_D867
        JSR     LAB_DA8C                ; unpack memory (AY) into FAC2

; add FAC2 to FAC1

LAB_D86A
        BNE     LAB_D86F                ; if FAC1 is not zero go do the add

        JMP     LAB DBFC                ; FAC1 was zero so copy FAC2 to FAC1 and return
```

```
                                        ; FAC1 is non zero
LAB_D86F
        LDX     LAB_70          ; get FAC1 rounding byte
        STX     LAB_56          ; save as FAC2 rounding byte
        LDX     #LAB_69         ; set index to FAC2 exponent address
        LDA     LAB_69          ; get FAC2 exponent
LAB_D877
        TAY                             ; copy exponent
        BEQ     LAB_D848                ; exit if zero

        SEC                             ; set carry for subtract
        SBC     LAB_61          ; subtract FAC1 exponent
        BEQ     LAB_D8A3                ; if equal go add mantissas

        BCC     LAB_D893                ; if FAC2 < FAC1 then go shift FAC2 right

                                        ; else FAC2 > FAC1
        STY     LAB_61          ; save FAC1 exponent
        LDY     LAB_6E          ; get FAC2 sign (b7)
        STY     LAB_66          ; save FAC1 sign (b7)
        EOR     #$FF                    ; complement A
        ADC     #$00                    ; +1, twos complement, carry is set
        LDY     #$00                    ; clear Y
        STY     LAB_56          ; clear FAC2 rounding byte
        LDX     #LAB_61         ; set index to FAC1 exponent address
        BNE     LAB_D897                ; branch always

                                        ; FAC2 < FAC1
LAB_D893
        LDY     #$00                    ; clear Y
        STY     LAB_70          ; clear FAC1 rounding byte
LAB_D897
        CMP     #$F9                    ; compare exponent diff with $F9
        BMI     LAB_D862                ; branch if range $79-$F8

        TAY                             ; copy exponent difference to Y
        LDA     LAB_70          ; get FAC1 rounding byte
        LSR     LAB_01,X                ; shift FAC? mantissa 1
        JSR     LAB_D9B0                ; shift FACX Y times right

                                        ; exponents are equal now do mantissa subtract
LAB_D8A3
        BIT     LAB_6F          ; test sign compare (FAC1 EOR FAC2)
        BPL     LAB_D8FE                ; if = add FAC2 mantissa to FAC1 mantissa and return

        LDY     #LAB_61         ; set index to FAC1 exponent address
        CPX     #LAB_69         ; compare X to FAC2 exponent address
        BEQ     LAB_D8AF                ; branch if =

        LDY     #LAB_69         ; else set index to FAC2 exponent address

                                        ; subtract smaller from bigger (take sign of bigger)
LAB_D8AF
        SEC                             ; set carry for subtract
        EOR     #$FF                    ; ones complement A
        ADC     LAB_56          ; add FAC2 rounding byte
        STA     LAB_70          ; save FAC1 rounding byte
        LDA     LAB_04,Y                ; get FACY mantissa 4
        SBC     LAB_04,X                ; subtract FACX mantissa 4
        STA     LAB 65          ; save FAC1 mantissa 4
```

```
        LDA     LAB_03,Y                ; get FACY mantissa 3
        SBC     LAB_03,X                ; subtract FACX mantissa 3
        STA     LAB_64          ; save FAC1 mantissa 3
        LDA     LAB_02,Y                ; get FACY mantissa 2
        SBC     LAB_02,X                ; subtract FACX mantissa 2
        STA     LAB_63          ; save FAC1 mantissa 2
        LDA     LAB_01,Y                ; get FACY mantissa 1
        SBC     LAB_01,X                ; subtract FACX mantissa 1
        STA     LAB_62          ; save FAC1 mantissa 1


;*******************************************************************************;
;
; do ABS and normalise FAC1

LAB_D8D2
        BCS     LAB_D8D7                ; branch if number is +ve

        JSR     LAB_D947                ; negate FAC1

; normalise FAC1

LAB_D8D7
        LDY     #$00                    ; clear Y
        TYA                             ; clear A
        CLC                             ; clear carry for add
LAB_D8DB
        LDX     LAB_62          ; get FAC1 mantissa 1
        BNE     LAB_D929                ; if not zero normalise FAC1

        LDX     LAB_63          ; get FAC1 mantissa 2
        STX     LAB_62          ; save FAC1 mantissa 1
        LDX     LAB_64          ; get FAC1 mantissa 3
        STX     LAB_63          ; save FAC1 mantissa 2
        LDX     LAB_65          ; get FAC1 mantissa 4
        STX     LAB_64          ; save FAC1 mantissa 3
        LDX     LAB_70          ; get FAC1 rounding byte
        STX     LAB_65          ; save FAC1 mantissa 4
        STY     LAB_70          ; clear FAC1 rounding byte
        ADC     #$08                    ; add x to exponent offset
        CMP     #$20                    ; compare with $20, max offset, all bits would be = 0
        BNE     LAB_D8DB                ; loop if not max


;*******************************************************************************;
;
; clear FAC1 exponent and sign

LAB_D8F7
        LDA     #$00                    ; clear A
LAB_D8F9
        STA     LAB_61          ; set FAC1 exponent

; save FAC1 sign

LAB_D8FB
        STA     LAB_66          ; save FAC1 sign (b7)
        RTS


;*******************************************************************************;
```

```
;
; add FAC2 mantissa to FAC1 mantissa

LAB_D8FE
        ADC     LAB_56          ; add FAC2 rounding byte
        STA     LAB_70          ; save FAC1 rounding byte
        LDA     LAB_65          ; get FAC1 mantissa 4
        ADC     LAB_6D          ; add FAC2 mantissa 4
        STA     LAB_65          ; save FAC1 mantissa 4
        LDA     LAB_64          ; get FAC1 mantissa 3
        ADC     LAB_6C          ; add FAC2 mantissa 3
        STA     LAB_64          ; save FAC1 mantissa 3
        LDA     LAB_63          ; get FAC1 mantissa 2
        ADC     LAB_6B          ; add FAC2 mantissa 2
        STA     LAB_63          ; save FAC1 mantissa 2
        LDA     LAB_62          ; get FAC1 mantissa 1
        ADC     LAB_6A          ; add FAC2 mantissa 1
        STA     LAB_62          ; save FAC1 mantissa 1
        JMP     LAB_D936                ; test and normalise FAC1 for C=0/1

LAB_D91D
        ADC     #$01                    ; add 1 to exponent offset
        ASL     LAB_70          ; shift FAC1 rounding byte
        ROL     LAB_65          ; shift FAC1 mantissa 4
        ROL     LAB_64          ; shift FAC1 mantissa 3
        ROL     LAB_63          ; shift FAC1 mantissa 2
        ROL     LAB_62          ; shift FAC1 mantissa 1


;******************************************************************************;
;
; normalise FAC1

LAB_D929
        BPL     LAB_D91D                ; loop if not normalised

        SEC                             ; set carry for subtract
        SBC     LAB_61          ; subtract FAC1 exponent
        BCS     LAB_D8F7                ; branch if underflow (set result = $0)

        EOR     #$FF                    ; complement exponent
        ADC     #$01                    ; +1 (twos complement)
        STA     LAB_61          ; save FAC1 exponent

; test and normalise FAC1 for C=0/1

LAB_D936
        BCC     LAB_D946                ; exit if no overflow

; normalise FAC1 for C=1

LAB_D938
        INC     LAB_61          ; increment FAC1 exponent
        BEQ     LAB_D97E                ; if zero do overflow error then warm start

        ROR     LAB_62          ; shift FAC1 mantissa 1
        ROR     LAB_63          ; shift FAC1 mantissa 2
        ROR     LAB_64          ; shift FAC1 mantissa 3
        ROR     LAB_65          ; shift FAC1 mantissa 4
        ROR     LAB_70          ; shift FAC1 rounding byte
LAB D946
```

```
        RTS


;*******************************************************************************;
;
; negate FAC1

LAB_D947
        LDA     LAB_66          ; get FAC1 sign (b7)
        EOR     #$FF                    ; complement it
        STA     LAB_66          ; save FAC1 sign (b7)

; twos complement FAC1 mantissa

LAB_D94D
        LDA     LAB_62          ; get FAC1 mantissa 1
        EOR     #$FF                    ; complement it
        STA     LAB_62          ; save FAC1 mantissa 1
        LDA     LAB_63          ; get FAC1 mantissa 2
        EOR     #$FF                    ; complement it
        STA     LAB_63          ; save FAC1 mantissa 2
        LDA     LAB_64          ; get FAC1 mantissa 3
        EOR     #$FF                    ; complement it
        STA     LAB_64          ; save FAC1 mantissa 3
        LDA     LAB_65          ; get FAC1 mantissa 4
        EOR     #$FF                    ; complement it
        STA     LAB_65          ; save FAC1 mantissa 4
        LDA     LAB_70          ; get FAC1 rounding byte
        EOR     #$FF                    ; complement it
        STA     LAB_70          ; save FAC1 rounding byte
        INC     LAB_70          ; increment FAC1 rounding byte
        BNE     LAB_D97D                ; exit if no overflow

; increment FAC1 mantissa

LAB_D96F
        INC     LAB_65          ; increment FAC1 mantissa 4
        BNE     LAB_D97D                ; finished if no rollover

        INC     LAB_64          ; increment FAC1 mantissa 3
        BNE     LAB_D97D                ; finished if no rollover

        INC     LAB_63          ; increment FAC1 mantissa 2
        BNE     LAB_D97D                ; finished if no rollover

        INC     LAB_62          ; increment FAC1 mantissa 1
LAB_D97D
        RTS


;*******************************************************************************;
;
; do overflow error then warm start

LAB_D97E
        LDX     #$0F                    ; error $0F, overflow error
        JMP     LAB_C437                ; do error #X then warm start


;*******************************************************************************;
:
```

```
; shift FCAtemp << A+8 times

LAB_D983
        LDX     #$25                    ; set offset to FACtemp
LAB_D985
        LDY     LAB_04,X                ; get FACX mantissa 4
        STY     LAB_70          ; save as FAC1 rounding byte
        LDY     LAB_03,X                ; get FACX mantissa 3
        STY     LAB_04,X                ; save FACX mantissa 4
        LDY     LAB_02,X                ; get FACX mantissa 2
        STY     LAB_03,X                ; save FACX mantissa 3
        LDY     LAB_01,X                ; get FACX mantissa 1
        STY     LAB_02,X                ; save FACX mantissa 2
        LDY     LAB_68          ; get FAC1 overflow byte
        STY     LAB_01,X                ; save FACX mantissa 1

; shift FACX -A times right (> 8 shifts)

LAB_D999
        ADC     #$08                    ; add 8 to shift count
        BMI     LAB_D985                ; go do 8 shift if still -ve

        BEQ     LAB_D985                ; go do 8 shift if zero

        SBC     #$08                    ; else subtract 8 again
        TAY                             ; save count to Y
        LDA     LAB_70          ; get FAC1 rounding byte
        BCS     LAB_D9BA                ;.

LAB_D9A6
        ASL     LAB_01,X                ; shift FACX mantissa 1
        BCC     LAB_D9AC                ; branch if +ve

        INC     LAB_01,X                ; this sets b7 eventually
LAB_D9AC
        ROR     LAB_01,X                ; shift FACX mantissa 1 (correct for ASL)
        ROR     LAB_01,X                ; shift FACX mantissa 1 (put carry in b7)

; shift FACX Y times right

LAB_D9B0
        ROR     LAB_02,X                ; shift FACX mantissa 2
        ROR     LAB_03,X                ; shift FACX mantissa 3
        ROR     LAB_04,X                ; shift FACX mantissa 4
        ROR                             ; shift FACX rounding byte
        INY                             ; increment exponent diff
        BNE     LAB_D9A6                ; branch if range adjust not complete

LAB_D9BA
        CLC                             ; just clear it
        RTS


;*******************************************************************************;
;
; constants and series for LOG(n)

LAB_D9BC
        .byte   $81,$00,$00,$00,$00     ; 1

LAB D9C1
```

```
        .byte   $03                             ; series counter
        .byte   $7F,$5E,$56,$CB,$79
        .byte   $80,$13,$9B,$0B,$64
        .byte   $80,$76,$38,$93,$16
        .byte   $82,$38,$AA,$3B,$20

LAB_D9D6
        .byte   $80,$35,$04,$F3,$34     ; 0.70711       1/root 2
LAB_D9DB
        .byte   $81,$35,$04,$F3,$34     ; 1.41421       root 2
LAB_D9E0
        .byte   $80,$80,$00,$00,$00     ; -0.5  1/2
LAB_D9E5
        .byte   $80,$31,$72,$17,$F8     ; 0.69315       LOG(2)


;******************************************************************************;
;
; perform LOG()

LAB_D9EA
        JSR     LAB_DC2B                ; test sign and zero
        BEQ     LAB_D9F1                ; if zero do illegal quantity error then warm start

        BPL     LAB_D9F4                ; skip error if +ve

LAB_D9F1
        JMP     LAB_D248                ; do illegal quantity error then warm start

LAB_D9F4
        LDA     LAB_61          ; get FAC1 exponent
        SBC     #$7F                    ; normalise it
        PHA                             ; save it
        LDA     #$80                    ; set exponent to zero
        STA     LAB_61          ; save FAC1 exponent
        LDA     #<LAB_D9D6              ; pointer to 1/root 2 low byte
        LDY     #>LAB_D9D6              ; pointer to 1/root 2 high byte
        JSR     LAB_D867                ; add (AY) to FAC1 (1/root2)
        LDA     #<LAB_D9DB              ; pointer to root 2 low byte
        LDY     #>LAB_D9DB              ; pointer to root 2 high byte
        JSR     LAB_DB0F                ; convert AY and do (AY)/FAC1 (root2/(x+(1/root2)))
        LDA     #<LAB_D9BC              ; pointer to 1 low byte
        LDY     #>LAB_D9BC              ; pointer to 1 high byte
        JSR     LAB_D850                ; subtract FAC1 ((root2/(x+(1/root2)))-1) from (AY)
        LDA     #<LAB_D9C1              ; pointer to series for LOG(n) low byte
        LDY     #>LAB_D9C1              ; pointer to series for LOG(n) high byte
        JSR     LAB_E040                ; ^2 then series evaluation
        LDA     #<LAB_D9E0              ; pointer to -0.5 low byte
        LDY     #>LAB_D9E0              ; pointer to -0.5 high byte
        JSR     LAB_D867                ; add (AY) to FAC1
        PLA                             ; restore FAC1 exponent
        JSR     LAB_DD7E                ; evaluate new ASCII digit
        LDA     #<LAB_D9E5              ; pointer to LOG(2) low byte
        LDY     #>LAB_D9E5              ; pointer to LOG(2) high byte

; do convert AY, FCA1*(AY)

LAB_DA28
        JSR     LAB_DA8C                ; unpack memory (AY) into FAC2
LAB_DA2B
        BNE     LAB_DA30                ; multiply FAC1 by FAC2 ??
```

```
        JMP     LAB_DA8B                ; exit if zero

LAB_DA30
        JSR     LAB_DAB7                ; test and adjust accumulators
        LDA     #$00                    ; clear A
        STA     LAB_26          ; clear temp mantissa 1
        STA     LAB_27          ; clear temp mantissa 2
        STA     LAB_28          ; clear temp mantissa 3
        STA     LAB_29          ; clear temp mantissa 4
        LDA     LAB_70          ; get FAC1 rounding byte
        JSR     LAB_DA59                ; go do shift/add FAC2
        LDA     LAB_65          ; get FAC1 mantissa 4
        JSR     LAB_DA59                ; go do shift/add FAC2
        LDA     LAB_64          ; get FAC1 mantissa 3
        JSR     LAB_DA59                ; go do shift/add FAC2
        LDA     LAB_63          ; get FAC1 mantissa 2
        JSR     LAB_DA59                ; go do shift/add FAC2
        LDA     LAB_62          ; get FAC1 mantissa 1
        JSR     LAB_DA5E                ; go do shift/add FAC2
        JMP     LAB_DB8F                ; copy temp to FAC1, normalise and return

LAB_DA59
        BNE     LAB_DA5E                ; branch if byte <> zero

        JMP     LAB_D983                ; shift FCAtemp << A+8 times

                                        ; else do shift and add
LAB_DA5E
        LSR                             ; shift byte
        ORA     #$80                    ; set top bit (mark for 8 times)
LAB_DA61
        TAY                             ; copy result
        BCC     LAB_DA7D                ; skip next if bit was zero

        CLC                             ; clear carry for add
        LDA     LAB_29          ; get temp mantissa 4
        ADC     LAB_6D          ; add FAC2 mantissa 4
        STA     LAB_29          ; save temp mantissa 4
        LDA     LAB_28          ; get temp mantissa 3
        ADC     LAB_6C          ; add FAC2 mantissa 3
        STA     LAB_28          ; save temp mantissa 3
        LDA     LAB_27          ; get temp mantissa 2
        ADC     LAB_6B          ; add FAC2 mantissa 2
        STA     LAB_27          ; save temp mantissa 2
        LDA     LAB_26          ; get temp mantissa 1
        ADC     LAB_6A          ; add FAC2 mantissa 1
        STA     LAB_26          ; save temp mantissa 1
LAB_DA7D
        ROR     LAB_26          ; shift temp mantissa 1
        ROR     LAB_27          ; shift temp mantissa 2
        ROR     LAB_28          ; shift temp mantissa 3
        ROR     LAB_29          ; shift temp mantissa 4
        ROR     LAB_70          ; shift temp rounding byte
        TYA                             ; get byte back
        LSR                             ; shift byte
        BNE     LAB_DA61                ; loop if all bits not done

LAB_DA8B
        RTS
```

```
;******************************************************************************;
;
; unpack memory (AY) into FAC2

LAB_DA8C
        STA     LAB_22          ; save pointer low byte
        STY     LAB_23          ; save pointer high byte
        LDY     #$04                    ; 5 bytes to get (0-4)
        LDA     (LAB_22),Y              ; get mantissa 4
        STA     LAB_6D          ; save FAC2 mantissa 4
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get mantissa 3
        STA     LAB_6C          ; save FAC2 mantissa 3
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get mantissa 2
        STA     LAB_6B          ; save FAC2 mantissa 2
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get mantissa 1 + sign
        STA     LAB_6E          ; save FAC2 sign (b7)
        EOR     LAB_66          ; EOR with FAC1 sign (b7)
        STA     LAB_6F          ; save sign compare (FAC1 EOR FAC2)
        LDA     LAB_6E          ; recover FAC2 sign (b7)
        ORA     #$80                    ; set 1xxx xxx (set normal bit)
        STA     LAB_6A          ; save FAC2 mantissa 1
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get exponent byte
        STA     LAB_69          ; save FAC2 exponent
        LDA     LAB_61          ; get FAC1 exponent
        RTS


;******************************************************************************;
;
; test and adjust accumulators

LAB_DAB7
        LDA     LAB_69          ; get FAC2 exponent

LAB_DAB9
        BEQ     LAB_DADA                ; branch if FAC2 = $00 (handle underflow)

        CLC                             ; clear carry for add
        ADC     LAB_61          ; add FAC1 exponent
        BCC     LAB_DAC4                ; branch if sum of exponents < $0100

        BMI     LAB_DADF                ; do overflow error

        CLC                             ; clear carry for the add
        .byte   $2C                     ; makes next line BIT LAB_1410
LAB_DAC4
        BPL     LAB_DADA                ; if +ve go handle underflow

        ADC     #$80                    ; adjust exponent
        STA     LAB_61          ; save FAC1 exponent
        BNE     LAB_DACF                ; branch if not zero

        JMP     LAB_D8FB                ; save FAC1 sign and return


LAB DACF
```

```
        LDA     LAB_6F          ; get sign compare (FAC1 EOR FAC2)
        STA     LAB_66          ; save FAC1 sign (b7)
        RTS


;******************************************************************************;
;
; handle overflow and underflow

LAB_DAD4
        LDA     LAB_66          ; get FAC1 sign (b7)
        EOR     #$FF                    ; complement it
        BMI     LAB_DADF                ; do overflow error

                                        ; handle underflow
LAB_DADA
        PLA                             ; pop return address low byte
        PLA                             ; pop return address high byte
        JMP     LAB_D8F7                ; clear FAC1 exponent and sign and return

LAB_DADF
        JMP     LAB_D97E                ; do overflow error then warm start


;******************************************************************************;
;
; multiply FAC1 by 10

LAB_DAE2
        JSR     LAB_DC0C                ; round and copy FAC1 to FAC2
        TAX                             ; copy exponent (set the flags)
        BEQ     LAB_DAF8                ; exit if zero

        CLC                             ; clear carry for add
        ADC     #$02                    ; add two to exponent (*4)
        BCS     LAB_DADF                ; do overflow error if > $FF

; FAC1 = (FAC1 + FAC2) * 2

LAB_DAED
        LDX     #$00                    ; clear byte
        STX     LAB_6F          ; clear sign compare (FAC1 EOR FAC2)
        JSR     LAB_D877                ; add FAC2 to FAC1 (*5)
        INC     LAB_61          ; increment FAC1 exponent (*10)
        BEQ     LAB_DADF                ; if exponent now zero go do overflow error

LAB_DAF8
        RTS


;******************************************************************************;
;
; 10 as a floating value

LAB_DAF9
        .byte   $84,$20,$00,$00,$00     ; 10


;******************************************************************************;
;
; divide FAC1 by 10
```

```
LAB_DAFE
        JSR     LAB_DC0C                ; round and copy FAC1 to FAC2
        LDA     #<LAB_DAF9              ; set 10 pointer low byte
        LDY     #>LAB_DAF9              ; set 10 pointer high byte
        LDX     #$00                    ; clear sign

; divide by (AY) (X=sign)

LAB_DB07
        STX     LAB_6F          ; save sign compare (FAC1 EOR FAC2)
        JSR     LAB_DBA2                ; unpack memory (AY) into FAC1
        JMP     LAB_DB12                ; do FAC2/FAC1

                                        ; Perform divide-by

; convert AY and do (AY)/FAC1

LAB_DB0F
        JSR     LAB_DA8C                ; unpack memory (AY) into FAC2
LAB_DB12
        BEQ     LAB_DB8A                ; if zero go do /0 error

        JSR     LAB_DC1B                ; round FAC1
        LDA     #$00                    ; clear A
        SEC                             ; set carry for subtract
        SBC     LAB_61          ; subtract FAC1 exponent (2s complement)
        STA     LAB_61          ; save FAC1 exponent
        JSR     LAB_DAB7                ; test and adjust accumulators
        INC     LAB_61          ; increment FAC1 exponent
        BEQ     LAB_DADF                ; if zero do overflow error

        LDX     #$FC                    ; set index to FAC temp
        LDA     #$01                    ;.set byte
LAB_DB29
        LDY     LAB_6A          ; get FAC2 mantissa 1
        CPY     LAB_62          ; compare FAC1 mantissa 1
        BNE     LAB_DB3F                ; if <> go use the result

        LDY     LAB_6B          ; get FAC2 mantissa 2
        CPY     LAB_63          ; compare FAC1 mantissa 2
        BNE     LAB_DB3F                ; if <> go use the result

        LDY     LAB_6C          ; get FAC2 mantissa 3
        CPY     LAB_64          ; compare FAC1 mantissa 3
        BNE     LAB_DB3F                ; if <> go use the result

        LDY     LAB_6D          ; get FAC2 mantissa 4
        CPY     LAB_65          ; compare FAC1 mantissa 4
LAB_DB3F
        PHP                             ; save the FAC2-FAC1 compare status
        ROL                             ;.shift byte
        BCC     LAB_DB4C                ; skip next if no carry

        INX                             ; increment index to FAC temp
        STA     LAB_29,X                ;.
        BEQ     LAB_DB7A                ;.

        BPL     LAB_DB7E                ;.

        LDA     #$01                    :.
```

```
LAB_DB4C
        PLP                             ; restore FAC2-FAC1 compare status
        BCS     LAB_DB5D                ; if FAC2 >= FAC1 then do subtract

                                        ; FAC2 = FAC2*2
LAB_DB4F
        ASL     LAB_6D          ; shift FAC2 mantissa 4
        ROL     LAB_6C          ; shift FAC2 mantissa 3
        ROL     LAB_6B          ; shift FAC2 mantissa 2
        ROL     LAB_6A          ; shift FAC2 mantissa 1
        BCS     LAB_DB3F                ; loop with no compare

        BMI     LAB_DB29                ; loop with compare

        BPL     LAB_DB3F                ; loop always with no compare

LAB_DB5D
        TAY                             ; save FAC2-FAC1 compare status
        LDA     LAB_6D          ; get FAC2 mantissa 4
        SBC     LAB_65          ; subtract FAC1 mantissa 4
        STA     LAB_6D          ; save FAC2 mantissa 4
        LDA     LAB_6C          ; get FAC2 mantissa 3
        SBC     LAB_64          ; subtract FAC1 mantissa 3
        STA     LAB_6C          ; save FAC2 mantissa 3
        LDA     LAB_6B          ; get FAC2 mantissa 2
        SBC     LAB_63          ; subtract FAC1 mantissa 2
        STA     LAB_6B          ; save FAC2 mantissa 2
        LDA     LAB_6A          ; get FAC2 mantissa 1
        SBC     LAB_62          ; subtract FAC1 mantissa 1
        STA     LAB_6A          ; save FAC2 mantissa 1
        TYA                             ; restore FAC2-FAC1 compare status
        JMP     LAB_DB4F                ; go shift FAC2

LAB_DB7A
        LDA     #$40                    ;.
        BNE     LAB_DB4C                ; branch always

; do A<<6, save as FAC1 rounding byte, normalise and return

LAB_DB7E
        ASL                             ;
        ASL                             ;
        ASL                             ;
        ASL                             ;
        ASL                             ;
        ASL                             ;
        STA     LAB_70          ; save FAC1 rounding byte
        PLP                             ; dump FAC2-FAC1 compare status
        JMP     LAB_DB8F                ; copy temp to FAC1, normalise and return

; do "Divide by zero" error

LAB_DB8A
        LDX     #$14                    ; error $14, divide by zero error
        JMP     LAB_C437                ; do error #X then warm start

LAB_DB8F
        LDA     LAB_26          ; get temp mantissa 1
        STA     LAB_62          ; save FAC1 mantissa 1
        LDA     LAB_27          ; get temp mantissa 2
        STA     LAB 63          : save FAC1 mantissa 2
```

```
        LDA     LAB_28          ; get temp mantissa 3
        STA     LAB_64          ; save FAC1 mantissa 3
        LDA     LAB_29          ; get temp mantissa 4
        STA     LAB_65          ; save FAC1 mantissa 4
        JMP     LAB_D8D7                ; normalise FAC1 and return


;******************************************************************************;
;
; unpack memory (AY) into FAC1

LAB_DBA2
        STA     LAB_22          ; save pointer low byte
        STY     LAB_23          ; save pointer high byte
        LDY     #$04                    ; 5 bytes to do
        LDA     (LAB_22),Y              ; get fifth byte
        STA     LAB_65          ; save FAC1 mantissa 4
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get fourth byte
        STA     LAB_64          ; save FAC1 mantissa 3
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get third byte
        STA     LAB_63          ; save FAC1 mantissa 2
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get second byte
        STA     LAB_66          ; save FAC1 sign (b7)
        ORA     #$80                    ; set 1xxx xxxx (add normal bit)
        STA     LAB_62          ; save FAC1 mantissa 1
        DEY                             ; decrement index
        LDA     (LAB_22),Y              ; get first byte (exponent)
        STA     LAB_61          ; save FAC1 exponent
        STY     LAB_70          ; clear FAC1 rounding byte
        RTS


;******************************************************************************;
;
; pack FAC1 into LAB_5C

LAB_DBC7
        LDX     #<LAB_5C                ; set pointer low byte
        .byte   $2C                     ; makes next line BIT LAB_57A2

; pack FAC1 into LAB_57

LAB_DBCA
        LDX     #<LAB_57                ; set pointer low byte
        LDY     #>LAB_57                ; set pointer high byte
        BEQ     LAB_DBD4                ; pack FAC1 into (XY) and return, branch always

; pack FAC1 into variable pointer

LAB_DBD0
        LDX     LAB_49          ; get destination pointer low byte
        LDY     LAB_4A          ; get destination pointer high byte

; pack FAC1 into (XY)

LAB_DBD4
        JSR     LAB_DC1B                ; round FAC1
        STX     LAB_22          ; save pointer low byte
```

```
        STY     LAB_23          ; save pointer high byte
        LDY     #$04                    ; set index
        LDA     LAB_65          ; get FAC1 mantissa 4
        STA     (LAB_22),Y              ; store in destination
        DEY                             ; decrement index
        LDA     LAB_64          ; get FAC1 mantissa 3
        STA     (LAB_22),Y              ; store in destination
        DEY                             ; decrement index
        LDA     LAB_63          ; get FAC1 mantissa 2
        STA     (LAB_22),Y              ; store in destination
        DEY                             ; decrement index
        LDA     LAB_66          ; get FAC1 sign (b7)
        ORA     #$7F                    ; set bits x111 1111
        AND     LAB_62          ; AND in FAC1 mantissa 1
        STA     (LAB_22),Y              ; store in destination
        DEY                             ; decrement index
        LDA     LAB_61          ; get FAC1 exponent
        STA     (LAB_22),Y              ; store in destination
        STY     LAB_70          ; clear FAC1 rounding byte
        RTS


;******************************************************************************;
;
; copy FAC2 to FAC1

LAB_DBFC
        LDA     LAB_6E          ; get FAC2 sign (b7)

; save FAC1 sign and copy ABS(FAC2) to FAC1

LAB_DBFE
        STA     LAB_66          ; save FAC1 sign (b7)
        LDX     #$05                    ; 5 bytes to copy
LAB_DC02
        LDA     LAB_68,X                ; get byte from FAC2,X
        STA     LAB_60,X                ; save byte at FAC1,X
        DEX                             ; decrement count
        BNE     LAB_DC02                ; loop if not all done

        STX     LAB_70          ; clear FAC1 rounding byte
        RTS


;******************************************************************************;
;
; round and copy FAC1 to FAC2

LAB_DC0C
        JSR     LAB_DC1B                ; round FAC1

; copy FAC1 to FAC2

LAB_DC0F
        LDX     #$06                    ; 6 bytes to copy
LAB_DC11
        LDA     LAB_60,X                ; get byte from FAC1,X
        STA     LAB_68,X                ; save byte at FAC2,X
        DEX                             ; decrement count
        BNE     LAB_DC11                ; loop if not all done
```

```
        STX     LAB_70          ; clear FAC1 rounding byte
LAB_DC1A
        RTS


;******************************************************************************;
;
; round FAC1

LAB_DC1B
        LDA     LAB_61          ; get FAC1 exponent
        BEQ     LAB_DC1A                ; exit if zero

        ASL     LAB_70          ; shift FAC1 rounding byte
        BCC     LAB_DC1A                ; exit if no overflow

; round FAC1 (no check)

LAB_DC23
        JSR     LAB_D96F                ; increment FAC1 mantissa
        BNE     LAB_DC1A                ; branch if no overflow

        JMP     LAB_D938                ; nornalise FAC1 for C=1 and return

; get FAC1 sign
; return A = $FF, Cb = 1/-ve A = $01, Cb = 0/+ve, A = $00, Cb = ?/0

LAB_DC2B
        LDA     LAB_61          ; get FAC1 exponent
        BEQ     LAB_DC38                ; exit if zero (allready correct SGN(0)=0)

; return A = $FF, Cb = 1/-ve A = $01, Cb = 0/+ve
; no = 0 check

LAB_DC2F
        LDA     LAB_66          ; else get FAC1 sign (b7)

; return A = $FF, Cb = 1/-ve A = $01, Cb = 0/+ve
; no = 0 check, sign in A

LAB_DC31
        ROL                             ; move sign bit to carry
        LDA     #$FF                    ; set byte for -ve result
        BCS     LAB_DC38                ; return if sign was set (-ve)

        LDA     #$01                    ; else set byte for +ve result
LAB_DC38
        RTS


;******************************************************************************;
;
; perform SGN()

LAB_DC39
        JSR     LAB_DC2B                ; get FAC1 sign, return A = $FF -ve, A = $01 +ve

; save A as integer byte

LAB_DC3C
        STA     LAB 62          : save FAC1 mantissa 1
```

```
        LDA     #$00                    ; clear A
        STA     LAB_63          ; clear FAC1 mantissa 2
        LDX     #$88                    ; set exponent

; set exponent = X, clear FAC1 3 and 4 and normalise

LAB_DC44
        LDA     LAB_62          ; get FAC1 mantissa 1
        EOR     #$FF                    ; complement it
        ROL                             ; sign bit into carry

; set exponent = X, clear mantissa 4 and 3 and normalise FAC1

LAB_DC49
        LDA     #$00                    ; clear A
        STA     LAB_65          ; clear FAC1 mantissa 4
        STA     LAB_64          ; clear FAC1 mantissa 3

; set exponent = X and normalise FAC1

LAB_DC4F
        STX     LAB_61          ; set FAC1 exponent
        STA     LAB_70          ; clear FAC1 rounding byte
        STA     LAB_66          ; clear FAC1 sign (b7)
        JMP     LAB_D8D2                ; do ABS and normalise FAC1

; perform ABS()

LAB_DC58
        LSR     LAB_66          ; clear FAC1 sign, put zero in b7
        RTS


;*******************************************************************************;
;
; compare FAC1 with (AY)
; returns A=$00 if FAC1 = (AY)
; returns A=$01 if FAC1 > (AY)
; returns A=$FF if FAC1 < (AY)

LAB_DC5B
        STA     LAB_24          ; save pointer low byte
LAB_DC5D
        STY     LAB_25          ; save pointer high byte
        LDY     #$00                    ; clear index
        LDA     (LAB_24),Y              ; get exponent
        INY                             ; increment index
        TAX                             ; copy (AY) exponent to X
        BEQ     LAB_DC2B                ; branch if (AY) exponent=0 and get FAC1 sign
                                        ; A = $FF, Cb = 1/-ve A = $01, Cb = 0/+ve

        LDA     (LAB_24),Y              ; get (AY) mantissa 1, with sign
        EOR     LAB_66          ; EOR FAC1 sign (b7)
        BMI     LAB_DC2F                ; if signs <> do return A = $FF, Cb = 1/-ve
                                        ; A = $01, Cb = 0/+ve and return

        CPX     LAB_61          ; compare (AY) exponent with FAC1 exponent
        BNE     LAB_DC92                ; branch if different

        LDA     (LAB_24),Y              ; get (AY) mantissa 1, with sign
        ORA     #$80                    ; normalise top bit
```

```
        CMP     LAB_62          ; compare with FAC1 mantissa 1
        BNE     LAB_DC92                ; branch if different

        INY                             ; increment index
        LDA     (LAB_24),Y              ; get mantissa 2
        CMP     LAB_63          ; compare with FAC1 mantissa 2
        BNE     LAB_DC92                ; branch if different

        INY                             ; increment index
        LDA     (LAB_24),Y              ; get mantissa 3
        CMP     LAB_64          ; compare with FAC1 mantissa 3
        BNE     LAB_DC92                ; branch if different

        INY                             ; increment index
        LDA     #$7F                    ; set for 1/2 value rounding byte
        CMP     LAB_70          ; compare with FAC1 rounding byte (set carry)
        LDA     (LAB_24),Y              ; get mantissa 4
        SBC     LAB_65          ; subtract FAC1 mantissa 4
        BEQ     LAB_DCBA                ; exit if mantissa 4 equal

; gets here if number <> FAC1

LAB_DC92
        LDA     LAB_66          ; get FAC1 sign (b7)
        BCC     LAB_DC98                ; branch if FAC1 > (AY)

        EOR     #$FF                    ; else toggle FAC1 sign
LAB_DC98
        JMP     LAB_DC31                ; return A = $FF, Cb = 1/-ve A = $01, Cb = 0/+ve


;************************************************************************************;
;
; convert FAC1 floating to fixed

LAB_DC9B
        LDA     LAB_61          ; get FAC1 exponent
        BEQ     LAB_DCE9                ; if zero go clear FAC1 and return

        SEC                             ; set carry for subtract
        SBC     #$A0                    ; subtract maximum integer range exponent
        BIT     LAB_66          ; test FAC1 sign (b7)
        BPL     LAB_DCAF                ; branch if FAC1 +ve

                                        ; FAC1 was -ve
        TAX                             ; copy subtracted exponent
        LDA     #$FF                    ; overflow for -ve number
        STA     LAB_68          ; set FAC1 overflow byte
        JSR     LAB_D94D                ; twos complement FAC1 mantissa
        TXA                             ; restore subtracted exponent
LAB_DCAF
        LDX     #$61                    ; set index to FAC1
        CMP     #$F9                    ; compare exponent result
        BPL     LAB_DCBB                ; if < 8 shifts shift FAC1 A times right and return

        JSR     LAB_D999                ; shift FAC1 A times right (> 8 shifts)
        STY     LAB_68          ; clear FAC1 overflow byte
LAB_DCBA
        RTS
```

```
;**************************************************************************;
;
; shift FAC1 A times right

LAB_DCBB
        TAY                             ; copy shift count
        LDA     LAB_66          ; get FAC1 sign (b7)
        AND     #$80                    ; mask sign bit only (x000 0000)
        LSR     LAB_62          ; shift FAC1 mantissa 1
        ORA     LAB_62          ; OR sign in b7 FAC1 mantissa 1
        STA     LAB_62          ; save FAC1 mantissa 1
        JSR     LAB_D9B0                ; shift FAC1 Y times right
        STY     LAB_68          ; clear FAC1 overflow byte
        RTS


;**************************************************************************;
;
; perform INT()

LAB_DCCC
        LDA     LAB_61          ; get FAC1 exponent
        CMP     #$A0                    ; compare with max int
        BCS     LAB_DCF2                ; exit if >= (allready int, too big for fractional
part!)

        JSR     LAB_DC9B                ; convert FAC1 floating to fixed
        STY     LAB_70          ; save FAC1 rounding byte
        LDA     LAB_66          ; get FAC1 sign (b7)
        STY     LAB_66          ; save FAC1 sign (b7)
        EOR     #$80                    ; toggle FAC1 sign
        ROL                             ; shift into carry
        LDA     #$A0                    ; set new exponent
        STA     LAB_61          ; save FAC1 exponent
        LDA     LAB_65          ; get FAC1 mantissa 4
        STA     LAB_07          ; save FAC1 mantissa 4 for power function
        JMP     LAB_D8D2                ; do ABS and normalise FAC1


;**************************************************************************;
;
; clear FAC1 and return

LAB_DCE9
        STA     LAB_62          ; clear FAC1 mantissa 1
        STA     LAB_63          ; clear FAC1 mantissa 2
        STA     LAB_64          ; clear FAC1 mantissa 3
        STA     LAB_65          ; clear FAC1 mantissa 4
        TAY                             ; clear Y
LAB_DCF2
        RTS


;**************************************************************************;
;
; get FAC1 from string

LAB_DCF3
        LDY     #$00                    ; clear Y
        LDX     #$0A                    ; set index
LAB DCF7
```

```
        STY     LAB_5D,X                ; clear byte
        DEX                             ; decrement index
        BPL     LAB_DCF7                ; loop until numexp to negnum (and FAC1) = $00

        BCC     LAB_DD0D                ; branch if first character is numeric

        CMP     #'-'                    ; else compare with "-"
        BNE     LAB_DD06                ; branch if not "-"

        STX     LAB_67          ; set flag for -ve n (negnum = $FF)
        BEQ     LAB_DD0A                ; branch always

LAB_DD06
        CMP     #'+'                    ; else compare with "+"
        BNE     LAB_DD0F                ; branch if not "+"

LAB_DD0A
        JSR     LAB_0073                ; increment and scan memory
LAB_DD0D
        BCC     LAB_DD6A                ; branch if numeric character

LAB_DD0F
        CMP     #'.'                    ; else compare with "."
        BEQ     LAB_DD41                ; branch if "."

        CMP     #'E'                    ; else compare with "E"
        BNE     LAB_DD47                ; branch if not "E"

                                        ; was "E" so evaluate exponential part
        JSR     LAB_0073                ; increment and scan memory
        BCC     LAB_DD33                ; branch if numeric character

        CMP     #TK_MINUS               ; else compare with token for -
        BEQ     LAB_DD2E                ; branch if token for -

        CMP     #'-'                    ; else compare with "-"
        BEQ     LAB_DD2E                ; branch if "-"

        CMP     #TK_PLUS                ; else compare with token for +
        BEQ     LAB_DD30                ; branch if token for +

        CMP     #'+'                    ; else compare with "+"
        BEQ     LAB_DD30                ; branch if "+"

        BNE     LAB_DD35                ; branch always

LAB_DD2E
        ROR     LAB_60          ; set exponent -ve flag (C, which=1, into b7)
LAB_DD30
        JSR     LAB_0073                ; increment and scan memory
LAB_DD33
        BCC     LAB_DD91                ; branch if numeric character

LAB_DD35
        BIT     LAB_60          ; test exponent -ve flag
        BPL     LAB_DD47                ; if +ve go evaluate exponent

                                        ; else do exponent = -exponent
        LDA     #$00                    ; clear result
        SEC                             ; set carry for subtract
        SBC     LAB_5E          ; subtract exponent byte
```

```
        JMP     LAB_DD49                ; go evaluate exponent

LAB_DD41
        ROR     LAB_5F          ; set decimal point flag
        BIT     LAB_5F          ; test decimal point flag
        BVC     LAB_DD0A                ; branch if only one decimal point so far

                                        ; evaluate exponent
LAB_DD47
        LDA     LAB_5E          ; get exponent count byte
LAB_DD49
        SEC                             ; set carry for subtract
        SBC     LAB_5D          ; subtract numerator exponent
        STA     LAB_5E          ; save exponent count byte
        BEQ     LAB_DD62                ; branch if no adjustment

        BPL     LAB_DD5B                ; else if +ve go do FAC1*10^expcnt

                                        ; else go do FAC1/10^(0-expcnt)
LAB_DD52
        JSR     LAB_DAFE                ; divide FAC1 by 10
        INC     LAB_5E          ; increment exponent count byte
        BNE     LAB_DD52                ; loop until all done

        BEQ     LAB_DD62                ; branch always

LAB_DD5B
        JSR     LAB_DAE2                ; multiply FAC1 by 10
        DEC     LAB_5E          ; decrement exponent count byte
        BNE     LAB_DD5B                ; loop until all done

LAB_DD62
        LDA     LAB_67          ; get -ve flag
        BMI     LAB_DD67                ; if -ve do - FAC1 and return

        RTS

; do - FAC1 and return

LAB_DD67
        JMP     LAB_DFB4                ; do - FAC1

; do unsigned FAC1*10+number

LAB_DD6A
        PHA                             ; save character
        BIT     LAB_5F          ; test decimal point flag
        BPL     LAB_DD71                ; skip exponent increment if not set

        INC     LAB_5D          ; else increment number exponent
LAB_DD71
        JSR     LAB_DAE2                ; multiply FAC1 by 10
        PLA                             ; restore character
        SEC                             ; set carry for subtract
        SBC     #'0'                    ; convert to binary
        JSR     LAB_DD7E                ; evaluate new ASCII digit
        JMP     LAB_DD0A                ; go do next character

; evaluate new ASCII digit
; multiply FAC1 by 10 then (ABS) add in new digit
```

```
LAB_DD7E
        PHA                             ; save digit
        JSR     LAB_DC0C                ; round and copy FAC1 to FAC2
        PLA                             ; restore digit
        JSR     LAB_DC3C                ; save A as integer byte
        LDA     LAB_6E          ; get FAC2 sign (b7)
        EOR     LAB_66          ; toggle with FAC1 sign (b7)
        STA     LAB_6F          ; save sign compare (FAC1 EOR FAC2)
        LDX     LAB_61          ; get FAC1 exponent
        JMP     LAB_D86A                ; add FAC2 to FAC1 and return

; evaluate next character of exponential part of number

LAB_DD91
        LDA     LAB_5E          ; get exponent count byte
        CMP     #$0A                    ; compare with 10 decimal
        BCC     LAB_DDA0                ; branch if less

        LDA     #$64                    ; make all -ve exponents = -100 decimal (causes
underflow)
        BIT     LAB_60          ; test exponent -ve flag
        BMI     LAB_DDAE                ; branch if -ve

        JMP     LAB_D97E                ; else do overflow error then warm start

LAB_DDA0
        ASL                             ; *2
        ASL                             ; *4
        CLC                             ; clear carry for add
        ADC     LAB_5E          ; *5
        ASL                             ; *10
        CLC                             ; clear carry for add
        LDY     #$00                    ; set index
        ADC     (LAB_7A),Y              ; add character (will be $30 too much!)
        SEC                             ; set carry for subtract
        SBC     #'0'                    ; convert character to binary
LAB_DDAE
        STA     LAB_5E          ; save exponent count byte
        JMP     LAB_DD30                ; go get next character


;******************************************************************************;
;
LAB_DDB3
        .byte   $9B,$3E,$BC,$1F,$FD
                                        ; 99999999.90625, maximum value with at least one
decimal
LAB_DDB8
        .byte   $9E,$6E,$6B,$27,$FD
                                        ; 999999999.25, maximum value before scientific
notation
LAB_DDBD
        .byte   $9E,$6E,$6B,$28,$00
                                        ; 1000000000


;******************************************************************************;
;
; do " IN " line number message

LAB_DDC2
```

```
        LDA     #<LAB_C371              ; set " IN " pointer low byte
        LDY     #>LAB_C371              ; set " IN " pointer high byte
        JSR     LAB_DDDA                ; print null terminated string
        LDA     LAB_3A          ; get the current line number high byte
        LDX     LAB_39          ; get the current line number low byte


;**************************************************************************;
;
; print XA as unsigned integer

LAB_DDCD
        STA     LAB_62          ; save high byte as FAC1 mantissa1
        STX     LAB_63          ; save low byte as FAC1 mantissa2
        LDX     #$90                    ; set exponent to 16d bits
        SEC                             ; set integer is +ve flag
        JSR     LAB_DC49                ; set exponent = X, clear mantissa 4 and 3 and
normalise
                                        ; FAC1
        JSR     LAB_DDDF                ; convert FAC1 to string
LAB_DDDA
        JMP     LAB_CB1E                ; print null terminated string


;**************************************************************************;
;
; convert FAC1 to ASCII string result in (AY)

LAB_DDDD
        LDY     #$01                    ; set index = 1
LAB_DDDF
        LDA     #' '                    ; character = " " (assume +ve)
        BIT     LAB_66          ; test FAC1 sign (b7)
        BPL     LAB_DDE7                ; if +ve skip the - sign set

        LDA     #'-'                    ; else character = "-"
LAB_DDE7
        STA     LAB_00FF,Y              ; save leading character (" " or "-")
        STA     LAB_66          ; save FAC1 sign (b7)
        STY     LAB_71          ; save the index
        INY                             ; increment index
        LDA     #'0'                    ; set character = "0"
        LDX     LAB_61          ; get FAC1 exponent
        BNE     LAB_DDF8                ; if FAC1<>0 go convert it

                                        ; exponent was $00 so FAC1 is 0
        JMP     LAB_DF04                ; save last character, [EOT] and exit

; FAC1 is some non zero value

LAB_DDF8
        LDA     #$00                    ; clear (number exponent count)
        CPX     #$80                    ; compare FAC1 exponent with $80 (<1.00000)
        BEQ     LAB_DE00                ; branch if 0.5 <= FAC1 < 1.0

        BCS     LAB_DE09                ; branch if FAC1=>1

LAB_DE00
        LDA     #<LAB_DDBD              ; set 1000000000 pointer low byte
        LDY     #>LAB_DDBD              ; set 1000000000 pointer high byte
        JSR     LAB DA28                ; do convert AY, FCA1*(AY)
```

```
        LDA     #$F7                    ; set number exponent count
LAB_DE09
        STA     LAB_5D          ; save number exponent count
LAB_DE0B
        LDA     #<LAB_DDB8              ; set 999999999.25 pointer low byte (max before sci
note)
        LDY     #>LAB_DDB8              ; set 999999999.25 pointer high byte
        JSR     LAB_DC5B                ; compare FAC1 with (AY)
        BEQ     LAB_DE32                ; exit if FAC1 = (AY)

        BPL     LAB_DE28                ; go do /10 if FAC1 > (AY)

                                        ; FAC1 < (AY)
LAB_DE16
        LDA     #<LAB_DDB3              ; set 99999999.90625 pointer low byte
        LDY     #>LAB_DDB3              ; set 99999999.90625 pointer high byte
        JSR     LAB_DC5B                ; compare FAC1 with (AY)
        BEQ     LAB_DE21                ; branch if FAC1 = (AY) (allow decimal places)

        BPL     LAB_DE2F                ; branch if FAC1 > (AY) (no decimal places)

                                        ; FAC1 <= (AY)
LAB_DE21
        JSR     LAB_DAE2                ; multiply FAC1 by 10
        DEC     LAB_5D          ; decrement number exponent count
        BNE     LAB_DE16                ; go test again, branch always

LAB_DE28
        JSR     LAB_DAFE                ; divide FAC1 by 10
        INC     LAB_5D          ; increment number exponent count
        BNE     LAB_DE0B                ; go test again, branch always

; now we have just the digits to do

LAB_DE2F
        JSR     LAB_D849                ; add 0.5 to FAC1 (round FAC1)
LAB_DE32
        JSR     LAB_DC9B                ; convert FAC1 floating to fixed
        LDX     #$01                    ; set default digits before dp = 1
        LDA     LAB_5D          ; get number exponent count
        CLC                             ; clear carry for add
        ADC     #$0A                    ; up to 9 digits before point
        BMI     LAB_DE47                ; if -ve then 1 digit before dp

        CMP     #$0B                    ; A>=$0B if n>=1E9
        BCS     LAB_DE48                ; branch if >= $0B

                                        ; carry is clear
        ADC     #$FF                    ; take 1 from digit count
        TAX                             ; copy to X
        LDA     #$02                    ; set the exponent adjust
LAB_DE47
        SEC                             ; set carry for subtract
LAB_DE48
        SBC     #$02                    ; -2
        STA     LAB_5E          ; save the exponent adjust
        STX     LAB_5D          ; save digits before dp count
        TXA                             ; copy digits before dp count to A
        BEQ     LAB_DE53                ; if no digits before the dp go do the "."

        BPL     LAB_DE66                ; if there are digits before the dp go do them
```

```
LAB_DE53
        LDY     LAB_71          ; get the output string index
        LDA     #'.'                    ; character "."
        INY                             ; increment the index
        STA     LAB_0100-1,Y    ; save the "." to the output string
        TXA                             ; copy digits before dp count to A
        BEQ     LAB_DE64                ; if no digits before the dp skip the "0"

        LDA     #'0'                    ; character "0"
        INY                             ; increment index
        STA     LAB_0100-1,Y    ; save the "0" to the output string
LAB_DE64
        STY     LAB_71          ; save the output string index
LAB_DE66
        LDY     #$00                    ; clear the powers of 10 index (point to -100,000,000)
LAB_DE68
        LDX     #$80                    ; clear the digit, set the test sense
LAB_DE6A
        LDA     LAB_65          ; get FAC1 mantissa 4
        CLC                             ; clear carry for add
        ADC     LAB_DF16+3,Y    ; add byte 4, least significant
        STA     LAB_65          ; save FAC1 mantissa4
        LDA     LAB_64          ; get FAC1 mantissa 3
        ADC     LAB_DF16+2,Y    ; add byte 3
        STA     LAB_64          ; save FAC1 mantissa3
        LDA     LAB_63          ; get FAC1 mantissa 2
        ADC     LAB_DF16+1,Y    ; add byte 2
        STA     LAB_63          ; save FAC1 mantissa2
        LDA     LAB_62          ; get FAC1 mantissa 1
        ADC     LAB_DF16+0,Y    ; add byte 1, most significant
        STA     LAB_62          ; save FAC1 mantissa1
        INX                             ; increment the digit, set the sign on the test sense
bit
        BCS     LAB_DE8E                ; if the carry is set go test if the result was
positive

                                        ; else the result needs to be negative
        BPL     LAB_DE6A                ; not -ve so try again

        BMI     LAB_DE90                ; else done so return the digit

LAB_DE8E
        BMI     LAB_DE6A                ; not +ve so try again

; else done so return the digit

LAB_DE90
        TXA                             ; copy the digit
        BCC     LAB_DE97                ; if Cb=0 just use it

        EOR     #$FF                    ; else make the 2's complement ..
        ADC     #$0A                    ; .. and subtract it from 10
LAB_DE97
        ADC     #'0'-1          ; add "0"-1 to result
        INY                             ; increment ..
        INY                             ; .. index to..
        INY                             ; .. next less ..
        INY                             ; .. power of ten
        STY     LAB_47          ; save the powers of ten table index
        LDY     LAB_71          ; get output string index
```

```
        INY                             ; increment output string index
        TAX                             ; copy character to X
        AND     #$7F                    ; mask out top bit
        STA     LAB_0100-1,Y    ; save to output string
        DEC     LAB_5D          ; decrement # of characters before the dp
        BNE     LAB_DEB2                ; if still characters to do skip the decimal point

                                        ; else output the point
        LDA     #'.'                    ; character "."
        INY                             ; increment output string index
        STA     LAB_0100-1,Y    ; save to output string
LAB_DEB2
        STY     LAB_71          ; save the output string index
        LDY     LAB_47          ; get the powers of ten table index
        TXA                             ; get the character back
        EOR     #$FF                    ; toggle the test sense bit
        AND     #$80                    ; clear the digit
        TAX                             ; copy it to the new digit
        CPY     #LAB_DF3A-LAB_DF16
                                        ; compare the table index with the max for decimal
numbers
        BEQ     LAB_DEC4                ; if at the max exit the digit loop

        CPY     #LAB_DF52-LAB_DF16
                                        ; compare the table index with the max for time
        BNE     LAB_DE6A                ; loop if not at the max

; now remove trailing zeroes

LAB_DEC4
        LDY     LAB_71          ; restore the output string index
LAB_DEC6
        LDA     LAB_0100-1,Y    ; get character from output string
        DEY                             ; decrement output string index
        CMP     #'0'                    ; compare with "0"
        BEQ     LAB_DEC6                ; loop until non "0" character found

        CMP     #'.'                    ; compare with "."
        BEQ     LAB_DED3                ; branch if was dp

                                        ; restore last character
        INY                             ; increment output string index
LAB_DED3
        LDA     #'+'                    ; character "+"
        LDX     LAB_5E          ; get exponent count
        BEQ     LAB_DF07                ; if zero go set null terminator and exit

                                        ; exponent isn't zero so write exponent
        BPL     LAB_DEE3                ; branch if exponent count +ve

        LDA     #$00                    ; clear A
        SEC                             ; set carry for subtract
        SBC     LAB_5E          ; subtract exponent count adjust (convert -ve to +ve)
        TAX                             ; copy exponent count to X
        LDA     #'-'                    ; character "-"
LAB_DEE3
        STA     LAB_0100+1,Y    ; save to output string
        LDA     #'E'                    ; character "E"
        STA     LAB_0100,Y              ; save exponent sign to output string
        TXA                             ; get exponent count back
        LDX     #$2F                    : one less than "0" character
```

```
        SEC                             ; set carry for subtract
LAB_DEEF
        INX                             ; increment 10's character
        SBC     #$0A                    ; subtract 10 from exponent count
        BCS     LAB_DEEF                ; loop while still >= 0

        ADC     #':'                    ; add character ":" ($30+$0A, result is 10 less that
value)
        STA     LAB_0100+3,Y    ; save to output string
        TXA                             ; copy 10's character
        STA     LAB_0100+2,Y    ; save to output string
        LDA     #$00                    ; set null terminator
        STA     LAB_0100+4,Y    ; save to output string
        BEQ     LAB_DF0C                ; go set string pointer (AY) and exit, branch always

                                        ; save last character, [EOT] and exit
LAB_DF04
        STA     LAB_0100-1,Y    ; save last character to output string

                                        ; set null terminator and exit
LAB_DF07
        LDA     #$00                    ; set null terminator
        STA     LAB_0100,Y              ; save after last character

                                        ; set string pointer (AY) and exit
LAB_DF0C
        LDA     #<LAB_0100              ; set result string pointer low byte
        LDY     #>LAB_0100              ; set result string pointer high byte
        RTS


;******************************************************************************;
;

LAB_DF11
        .byte   $80,$00         ; 0.5, first two bytes
LAB_DF13
        .byte   $00,$00,$00             ; null return for undefined variables

; decimal conversion tabls

LAB_DF16
        .byte   $FA,$0A,$1F,$00 ; -100000000
        .byte   $00,$98,$96,$80 ;  +10000000
        .byte   $FF,$F0,$BD,$C0 ;   -1000000
        .byte   $00,$01,$86,$A0 ;    +100000
        .byte   $FF,$FF,$D8,$F0 ;     -10000
        .byte   $00,$00,$03,$E8 ;      +1000
        .byte   $FF,$FF,$FF,$9C ;       -100
        .byte   $00,$00,$00,$0A ;        +10
        .byte   $FF,$FF,$FF,$FF ;         -1

; jiffy count conversion table

LAB_DF3A
        .byte   $FF,$DF,$0A,$80 ; -2160000      10s hours
        .byte   $00,$03,$4B,$C0 ;  +216000          hours
        .byte   $FF,$FF,$73,$60 ;   -36000      10s mins
        .byte   $00,$00,$0E,$10 ;    +3600          mins
        .byte   $FF,$FF,$FD,$A8 ;     -600      10s secs
        .byte   $00,$00,$00,$3C ;      +60          secs
```

```
LAB_DF52


;******************************************************************************;
;
; spare bytes, not referenced

        .byte   $BF,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA
        .byte   $AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA,$AA


;******************************************************************************;
;
; perform SQR()

LAB_DF71
        JSR     LAB_DC0C                ; round and copy FAC1 to FAC2
        LDA     #<LAB_DF11              ; set 0.5 pointer low address
        LDY     #>LAB_DF11              ; set 0.5 pointer high address
        JSR     LAB_DBA2                ; unpack memory (AY) into FAC1


;******************************************************************************;
;
; perform power function

LAB_DF7B
        BEQ     LAB_DFED                ; perform EXP()

        LDA     LAB_69          ; get FAC2 exponent
        BNE     LAB_DF84                ; branch if FAC2<>0

        JMP     LAB_D8F9                ; clear FAC1 exponent and sign and return

LAB_DF84
        LDX     #<LAB_4E                ; set destination pointer low byte
        LDY     #>LAB_4E                ; set destination pointer high byte
        JSR     LAB_DBD4                ; pack FAC1 into (XY)
        LDA     LAB_6E          ; get FAC2 sign (b7)
        BPL     LAB_DF9E                ; branch if FAC2>0

                                        ; else FAC2 is -ve and can only be raised to an
                                        ; integer power which gives an x + j0 result
        JSR     LAB_DCCC                ; perform INT()
        LDA     #<LAB_4E                ; set source pointer low byte
        LDY     #>LAB_4E                ; set source pointer high byte
        JSR     LAB_DC5B                ; compare FAC1 with (AY)
        BNE     LAB_DF9E                ; branch if FAC1 <> (AY) to allow Function Call error
                                        ; this will leave FAC1 -ve and cause a Function Call
                                        ; error when LOG() is called

        TYA                             ; clear sign b7
        LDY     LAB_07          ; get FAC1 mantissa 4 from INT() function as sign in
                                        ; Y for possible later negation, b0 only needed
LAB_DF9E
        JSR     LAB_DBFE                ; save FAC1 sign and copy ABS(FAC2) to FAC1
        TYA                             ; copy sign back ..
        PHA                             ; .. and save it
        JSR     LAB_D9EA                ; perform LOG()
        LDA     #<LAB_4E                ; set pointer low byte
        LDY     #>LAB_4E                ; set pointer high byte
```

```
        JSR     LAB_DA28                ; do convert AY, FCA1*(AY)
        JSR     LAB_DFED                ; perform EXP()
        PLA                             ; pull sign from stack
        LSR                             ; b0 is to be tested
        BCC     LAB_DFBE                ; if no bit then exit

; do - FAC1

LAB_DFB4
        LDA     LAB_61          ; get FAC1 exponent
        BEQ     LAB_DFBE                ; exit if FAC1_e = $00

        LDA     LAB_66          ; get FAC1 sign (b7)
        EOR     #$FF                    ; complement it
        STA     LAB_66          ; save FAC1 sign (b7)
LAB_DFBE
        RTS


;******************************************************************************;
;
; exp(n) constant and series

LAB_DFBF
        .byte   $81,$38,$AA,$3B,$29     ; 1.443

LAB_DFC4
        .byte   $07                             ; series count
        .byte   $71,$34,$58,$3E,$56     ; 2.14987637E-5
        .byte   $74,$16,$7E,$B3,$1B     ; 1.43523140E-4
        .byte   $77,$2F,$EE,$E3,$85     ; 1.34226348E-3
        .byte   $7A,$1D,$84,$1C,$2A     ; 9.61401701E-3
        .byte   $7C,$63,$59,$58,$0A     ; 5.55051269E-2
        .byte   $7E,$75,$FD,$E7,$C6     ; 2.40226385E-1
        .byte   $80,$31,$72,$18,$10     ; 6.93147186E-1
        .byte   $81,$00,$00,$00,$00     ; 1.00000000


;******************************************************************************;
;
; perform EXP()

LAB_DFED
        LDA     #<LAB_DFBF              ; set 1.443 pointer low byte
        LDY     #>LAB_DFBF              ; set 1.443 pointer high byte
        JSR     LAB_DA28                ; do convert AY, FCA1*(AY)
        LDA     LAB_70          ; get FAC1 rounding byte
        ADC     #$50                    ; +$50/$100
        BCC     LAB_DFFD                ; skip rounding if no carry

        JSR     LAB_DC23                ; round FAC1 (no check)
LAB_DFFD
        STA     LAB_56          ; save FAC2 rounding byte
        JSR     LAB_DC0F                ; copy FAC1 to FAC2
        LDA     LAB_61          ; get FAC1 exponent
        CMP     #$88                    ; compare with EXP limit (256d)
        BCC     LAB_E00B                ; branch if less

LAB_E008
        JSR     LAB_DAD4                ; handle overflow and underflow
LAB_E00B
```

```
        JSR     LAB_DCCC                ; perform INT()
        LDA     LAB_07          ; get mantissa 4 from INT()
        CLC                             ; clear carry for add
        ADC     #$81                    ; normalise +1
        BEQ     LAB_E008                ; if $00 result has overflowed so go handle it

        SEC                             ; set carry for subtract
        SBC     #$01                    ; exponent now correct
        PHA                             ; save FAC2 exponent
                                        ; swap FAC1 and FAC2
        LDX     #$05                    ; 4 bytes to do
LAB_E01B
        LDA     LAB_69,X                ; get FAC2,X
        LDY     LAB_61,X                ; get FAC1,X
        STA     LAB_61,X                ; save FAC1,X
        STY     LAB_69,X                ; save FAC2,X
        DEX                             ; decrement count/index
        BPL     LAB_E01B                ; loop if not all done

        LDA     LAB_56          ; get FAC2 rounding byte
        STA     LAB_70          ; save as FAC1 rounding byte
        JSR     LAB_D853                ; perform subtraction, FAC2 from FAC1
        JSR     LAB_DFB4                ; do - FAC1
        LDA     #<LAB_DFC4              ; set counter pointer low byte
        LDY     #>LAB_DFC4              ; set counter pointer high byte
        JSR     LAB_E056                ; go do series evaluation
        LDA     #$00                    ; clear A
        STA     LAB_6F          ; clear sign compare (FAC1 EOR FAC2)
        PLA                             ; pull the saved FAC2 exponent
        JSR     LAB_DAB9                ; test and adjust accumulators
        RTS


;*******************************************************************************;
;
; ^2 then series evaluation

LAB_E040
        STA     LAB_71          ; save count pointer low byte
        STY     LAB_72          ; save count pointer high byte
        JSR     LAB_DBCA                ; pack FAC1 into LAB_57
        LDA     #<LAB_57                ; set pointer low byte (Y already $00)
        JSR     LAB_DA28                ; do convert AY, FCA1*(AY)
        JSR     LAB_E05A                ; go do series evaluation
        LDA     #<LAB_57                ; pointer to original # low byte
        LDY     #>LAB_57                ; pointer to original # high byte
        JMP     LAB_DA28                ; do convert AY, FCA1*(AY)


;*******************************************************************************;
;
; do series evaluation

LAB_E056
        STA     LAB_71          ; save count pointer low byte
        STY     LAB_72          ; save count pointer high byte

; do series evaluation

LAB_E05A
        JSR     LAB_DBC7                ; pack FAC1 into LAB_5C
```

```
        LDA     (LAB_71),Y              ; get constants count
        STA     LAB_67          ; save constants count
        LDY     LAB_71          ; get count pointer low byte
        INY                             ; increment it (now constants pointer)
        TYA                             ; copy it
        BNE     LAB_E069                ; skip next if no overflow

        INC     LAB_72          ; else increment high byte
LAB_E069
        STA     LAB_71          ; save low byte
        LDY     LAB_72          ; get high byte
LAB_E06D
        JSR     LAB_DA28                ; do convert AY, FCA1*(AY)
        LDA     LAB_71          ; get constants pointer low byte
        LDY     LAB_72          ; get constants pointer high byte
        CLC                             ; clear carry for add
        ADC     #$05                    ; +5 to low pointer (5 bytes per constant)
        BCC     LAB_E07A                ; skip next if no overflow

        INY                             ; increment high byte
LAB_E07A
        STA     LAB_71          ; save pointer low byte
        STY     LAB_72          ; save pointer high byte
        JSR     LAB_D867                ; add (AY) to FAC1
        LDA     #<LAB_5C                ; set pointer low byte to partial
        LDY     #>LAB_5C                ; set pointer high byte to partial
        DEC     LAB_67          ; decrement constants count
        BNE     LAB_E06D                ; loop until all done

        RTS


;**************************************************************************;
;
; RND values

LAB_E08A
        .byte   $98,$35,$44,$7A,$00
                                        ; 11879546                      multiplier
LAB_E08F
        .byte   $68,$28,$B1,$46,$00
                                        ; 3.927677739E-8                offset


;**************************************************************************;
;
; perform RND()

LAB_E094
        JSR     LAB_DC2B                ; get FAC1 sign
                                        ; return A = $FF -ve, A = $01 +ve
        BMI     LAB_E0D0                ; if n<0 copy byte swapped FAC1 into RND() seed

        BNE     LAB_E0BB                ; if n>0 get next number in RND() sequence

                                        ; else n=0 so get the RND() number from VIA 1 timers
        JSR     LAB_FFF3                ; return base address of I/O devices
        STX     LAB_22          ; save pointer low byte
        STY     LAB_23          ; save pointer high byte
        LDY     #$04                    ; set index to T1 low byte
        LDA     (LAB_22),Y              ; get T1 low byte
```

```
        STA     LAB_62          ; save FAC1 mantissa 1
        INY                     ; increment index
        LDA     (LAB_22),Y      ; get T1 high byte
        STA     LAB_64          ; save FAC1 mantissa 3
        LDY     #$08            ; set index to T2 low byte
        LDA     (LAB_22),Y      ; get T2 low byte
        STA     LAB_63          ; save FAC1 mantissa 2
        INY                     ; increment index
        LDA     (LAB_22),Y      ; get T2 high byte
        STA     LAB_65          ; save FAC1 mantissa 4
        JMP     LAB_E0E0        ; set exponent and exit

LAB_E0BB
        LDA     #<LAB_008B      ; set seed pointer low address
        LDY     #>LAB_008B      ; set seed pointer high address
        JSR     LAB_DBA2        ; unpack memory (AY) into FAC1
        LDA     #<LAB_E08A      ; set 11879546 pointer low byte
        LDY     #>LAB_E08A      ; set 11879546 pointer high byte
        JSR     LAB_DA28        ; do convert AY, FCA1*(AY)
        LDA     #<LAB_E08F      ; set 3.927677739E-8 pointer low byte
        LDY     #>LAB_E08F      ; set 3.927677739E-8 pointer high byte
        JSR     LAB_D867        ; add (AY) to FAC1
LAB_E0D0
        LDX     LAB_65          ; get FAC1 mantissa 4
        LDA     LAB_62          ; get FAC1 mantissa 1
        STA     LAB_65          ; save FAC1 mantissa 4
        STX     LAB_62          ; save FAC1 mantissa 1
        LDX     LAB_63          ; get FAC1 mantissa 2
        LDA     LAB_64          ; get FAC1 mantissa 3
        STA     LAB_63          ; save FAC1 mantissa 2
        STX     LAB_64          ; save FAC1 mantissa 3
LAB_E0E0
        LDA     #$00            ; clear byte
        STA     LAB_66          ; clear FAC1 sign (always +ve)
        LDA     LAB_61          ; get FAC1 exponent
        STA     LAB_70          ; save FAC1 rounding byte
        LDA     #$80            ; set exponent = $80
        STA     LAB_61          ; save FAC1 exponent
        JSR     LAB_D8D7        ; normalise FAC1
        LDX     #<LAB_008B      ; set seed pointer low address
        LDY     #>LAB_008B      ; set seed pointer high address


;*******************************************************************************;
;
; pack FAC1 into (XY)

LAB_E0F3
        JMP     LAB_DBD4        ; pack FAC1 into (XY)


;*******************************************************************************;
;
; handle BASIC I/O error

LAB_E0F6
        CMP     #$F0            ; compare error with $F0
        BNE     LAB_E101        ; branch if not $F0

        STY     LAB_38          ; set end of memory high byte
        STX     LAB_37          ; set end of memory low byte
```

```
        JMP     LAB_C663                ; clear from start to end and return

                                        ; error was not $F0
LAB_E101
        TAX                             ; copy error #
        BNE     LAB_E106                ; branch if not $00

        LDX     #$1E                    ; else error $1E, break error
LAB_E106
        JMP     LAB_C437                ; do error #X then warm start


;******************************************************************************;
;
; output character to channel with error check

LAB_E109
        JSR     LAB_FFD2                ; output character to channel
        BCS     LAB_E0F6                ; if error go handle BASIC I/O error

        RTS


;******************************************************************************;
;
; input character from channel with error check

LAB_E10F
        JSR     LAB_FFCF                ; input character from channel
        BCS     LAB_E0F6                ; if error go handle BASIC I/O error

        RTS


;******************************************************************************;
;
; open channel for output with error check

LAB_E115
        JSR     LAB_FFC9                ; open channel for output
        BCS     LAB_E0F6                ; if error go handle BASIC I/O error

        RTS


;******************************************************************************;
;
; open channel for input with error check

LAB_E11B
        JSR     LAB_FFC6                ; open channel for input
        BCS     LAB_E0F6                ; if error go handle BASIC I/O error

        RTS


;******************************************************************************;
;
; get character from input device with error check

LAB E121
```

```
        JSR     LAB_FFE4                ; get character from input device
        BCS     LAB_E0F6                ; if error go handle BASIC I/O error

        RTS


;******************************************************************************;
;
; perform SYS

LAB_E127
        JSR     LAB_CD8A                ; evaluate expression and check is numeric, else do
                                        ; type mismatch
        JSR     LAB_D7F7                ; convert FAC_1 to integer in temporary integer
        LDA     #>LAB_E143              ; get return address high byte
        PHA                             ; push as return address
        LDA     #<LAB_E143              ; get return address low byte
        PHA                             ; push as return address
        LDA     LAB_030F                ; get saved status register
        PHA                             ; put on stack
        LDA     LAB_030C                ; get saved A
        LDX     LAB_030D                ; get saved X
        LDY     LAB_030E                ; get saved Y
        PLP                             ; pull processor status
        JMP     (LAB_14)                ; call SYS address

; tail end of the SYS code
;
; the LAB_E143 is needed because the following code is to be executed once the user code
; returns. this is done by pushing the target return address - 1 onto the stack

LAB_E143        = *-1
;LAB_E144
        PHP                             ; save status
        STA     LAB_030C                ; save returned A
        STX     LAB_030D                ; save returned X
        STY     LAB_030E                ; save returned Y
        PLA                             ; restore saved status
        STA     LAB_030F                ; save status
        RTS


;******************************************************************************;
;
; perform SAVE

LAB_E153
        JSR     LAB_E1D1                ; get parameters for LOAD/SAVE
        LDX     LAB_2D          ; get start of variables low byte
        LDY     LAB_2E          ; get start of variables high byte
        LDA     #LAB_2B         ; index to start of program memory
        JSR     LAB_FFD8                ; save RAM to device, A = index to start address, XY =
end
                                        ; address low/high
        BCS     LAB_E0F6                ; if error go handle BASIC I/O error

        RTS


;******************************************************************************;
:
```

```
; perform VERIFY

LAB_E162
        LDA     #$01                    ; flag verify
        .byte   $2C                     ; makes next line BIT LAB_00A9


;******************************************************************************;
;
; perform LOAD

LAB_E165
        LDA     #$00                    ; flag load
        STA     LAB_0A          ; set load/verify flag
        JSR     LAB_E1D1                ; get parameters for LOAD/SAVE
        LDA     LAB_0A          ; get load/verify flag
        LDX     LAB_2B          ; get start of memory low byte
        LDY     LAB_2C          ; get start of memory high byte
        JSR     LAB_FFD5                ; load RAM from a device
        BCS     LAB_E1CE                ; if error go handle BASIC I/O error

        LDA     LAB_0A          ; get load/verify flag
        BEQ     LAB_E195                ; branch if load

        LDX     #$1C                    ; error $1C, verify error
        JSR     LAB_FFB7                ; read I/O status word
        AND     #$10                    ; mask for tape read error
        BEQ     LAB_E187                ; branch if no read error

        JMP     LAB_C437                ; do error #X then warm start

LAB_E187
        LDA     LAB_7A          ; get BASIC execute pointer low byte
                                        ; is this correct ?? won't this mean the "OK" prompt
                                        ; when doing a load from within a program ?
        CMP     #$02                    ;.
        BEQ     LAB_E194                ; if ?? skip "OK" prompt

        LDA     #<LAB_C364              ; set "OK" pointer low byte
        LDY     #>LAB_C364              ; set "OK" pointer high byte
        JMP     LAB_CB1E                ; print null terminated string

LAB_E194
        RTS


;******************************************************************************;
;
; do READY return to BASIC ??

LAB_E195
        JSR     LAB_FFB7                ; read I/O status word
        AND     #$BF                    ; mask x0xx xxxx, clear read error
        BEQ     LAB_E1A1                ; branch if no errors

        LDX     #$1D                    ; error $1D, load error
        JMP     LAB_C437                ; do error #X then warm start

LAB_E1A1
        LDA     LAB_7B          ; get BASIC execute pointer high byte
        CMP     #$02                    : compare with $02xx
```

```
        BNE     LAB_E1B5                ; branch if not immediate mode

        STX     LAB_2D          ; set start of variables low byte
        STY     LAB_2E          ; set start of variables high byte
        LDA     #<LAB_C376              ; set "READY." pointer low byte
        LDY     #>LAB_C376              ; set "READY." pointer high byte
        JSR     LAB_CB1E                ; print null terminated string
        JMP     LAB_C52A                ; reset execution, clear variables, flush stack,
                                        ; rebuild BASIC chain and do warm start

LAB_E1B5
        JSR     LAB_C68E                ; set BASIC execute pointer to start of memory - 1
        JMP     LAB_E476                ; rebuild BASIC line chaining, do RESTORE and return


;******************************************************************************;
;
; perform OPEN

LAB_E1BB
        JSR     LAB_E216                ; get parameters for OPEN/CLOSE
        JSR     LAB_FFC0                ; open a logical file
        BCS     LAB_E1CE                ; branch if error

        RTS


;******************************************************************************;
;
; perform CLOSE

LAB_E1C4
        JSR     LAB_E216                ; get parameters for OPEN/CLOSE
        LDA     LAB_49          ; get logical file number
        JSR     LAB_FFC3                ; close a specified logical file
        BCC     LAB_E194                ; exit if no error

LAB_E1CE
        JMP     LAB_E0F6                ; go handle BASIC I/O error


;******************************************************************************;
;
; get parameters for LOAD/SAVE

LAB_E1D1
        LDA     #$00                    ; clear file name length
        JSR     LAB_FFBD                ; clear filename
        LDX     #$01                    ; set default device number, cassette
        LDY     #$00                    ; set default command
        JSR     LAB_FFBA                ; set logical, first and second addresses
        JSR     LAB_E203                ; exit function if [EOT] or ":"
        JSR     LAB_E254                ; set filename
        JSR     LAB_E203                ; exit function if [EOT] or ":"
        JSR     LAB_E1FD                ; scan and get byte, else do syntax error then warm
start
        LDY     #$00                    ; clear command
        STX     LAB_49          ; save device number
        JSR     LAB_FFBA                ; set logical, first and second addresses
        JSR     LAB_E203                ; exit function if [EOT] or ":"
        JSR     LAB_E1FD                ; scan and get byte, else do syntax error then warm
```

```
start
        TXA                             ; copy command to A
        TAY                             ; copy command to Y
        LDX     LAB_49          ; get device number back
        JMP     LAB_FFBA                ; set logical, first and second addresses and return


;******************************************************************************;
;
; scan and get byte, else do syntax error then warm start

LAB_E1FD
        JSR     LAB_E20B                ; scan for ",byte", else do syntax error then warm
start
        JMP     LAB_D79E                ; get byte parameter and return


;******************************************************************************;
;
; exit function if [EOT] or ":"

LAB_E203
        JSR     LAB_0079                ; scan memory
        BNE     LAB_E20A                ; branch if not [EOL] or ":"

        PLA                             ; dump return address low byte
        PLA                             ; dump return address high byte
LAB_E20A
        RTS


;******************************************************************************;
;
; scan for ",valid byte", else do syntax error then warm start

LAB_E20B
        JSR     LAB_CEFD                ; scan for ",", else do syntax error then warm start

; scan for valid byte, not [EOL] or ":", else do syntax error then warm start

LAB_E20E
        JSR     LAB_0079                ; scan memory
        BNE     LAB_E20A                ; exit if following byte

        JMP     LAB_CF08                ; else do syntax error then warm start


;******************************************************************************;
;
; get parameters for OPEN/CLOSE

LAB_E216
        LDA     #$00                    ; clear file name length
        JSR     LAB_FFBD                ; clear filename
        JSR     LAB_E20E                ; scan for valid byte, else do syntax error then warm
start
        JSR     LAB_D79E                ; get byte parameter, logical file number
        STX     LAB_49          ; save logical file number
        TXA                             ; copy logical file number to A
        LDX     #$01                    ; set default device number, cassette
        LDY     #$00                    ; set default command
```

```
        JSR     LAB_FFBA                ; set logical, first and second addresses
        JSR     LAB_E203                ; exit function if [EOT] or ":"
        JSR     LAB_E1FD                ; scan and get byte, else do syntax error then warm
start
        STX     LAB_4A          ; save device number
        LDY     #$00                    ; clear command
        LDA     LAB_49          ; get logical file number
        CPX     #$03                    ; compare device number with screen
        BCC     LAB_E23C                ; branch if less than screen

        DEY                             ; else decrement command
LAB_E23C
        JSR     LAB_FFBA                ; set logical, first and second addresses
        JSR     LAB_E203                ; exit function if [EOT] or ":"
        JSR     LAB_E1FD                ; scan and get byte, else do syntax error then warm
start
        TXA                             ; copy command to A
        TAY                             ; copy command to Y
        LDX     LAB_4A          ; get device number
        LDA     LAB_49          ; get logical file number
        JSR     LAB_FFBA                ; set logical, first and second addresses
        JSR     LAB_E203                ; exit function if [EOT] or ":"
        JSR     LAB_E20B                ; scan for ",byte", else do syntax error then warm
start


;******************************************************************************;
;
; set filename

LAB_E254
        JSR     LAB_CD9E                ; evaluate expression
        JSR     LAB_D6A3                ; evaluate string
        LDX     LAB_22          ; get string pointer low byte
        LDY     LAB_23          ; get string pointer high byte
        JMP     LAB_FFBD                ; set filename and return


;******************************************************************************;
;
; perform COS()

LAB_E261
        LDA     #<LAB_E2DD              ; set pi/2 pointer low byte
        LDY     #>LAB_E2DD              ; set pi/2 pointer high byte
        JSR     LAB_D867                ; add (AY) to FAC1


;******************************************************************************;
;
; perform SIN()

LAB_E268
        JSR     LAB_DC0C                ; round and copy FAC1 to FAC2
        LDA     #<LAB_E2E2              ; set 2*pi pointer low byte
        LDY     #>LAB_E2E2              ; set 2*pi pointer high byte
        LDX     LAB_6E          ; get FAC2 sign (b7)
        JSR     LAB_DB07                ; divide by (AY) (X=sign)
        JSR     LAB_DC0C                ; round and copy FAC1 to FAC2
        JSR     LAB_DCCC                ; perform INT()
        LDA     #$00                    : clear byte
```

```
        STA     LAB_6F          ; clear sign compare (FAC1 EOR FAC2)
        JSR     LAB_D853                ; perform subtraction, FAC2 from FAC1
        LDA     #<LAB_E2E7              ; set 0.25 pointer low byte
        LDY     #>LAB_E2E7              ; set 0.25 pointer high byte
        JSR     LAB_D850                ; perform subtraction, FAC1 from (AY)
        LDA     LAB_66          ; get FAC1 sign (b7)
        PHA                             ; save FAC1 sign
        BPL     LAB_E29A                ; branch if +ve

                                        ; FAC1 sign was -ve
        JSR     LAB_D849                ; add 0.5 to FAC1 (round FAC1)
        LDA     LAB_66          ; get FAC1 sign (b7)
        BMI     LAB_E29D                ; branch if -ve

        LDA     LAB_12          ; get the comparison evaluation flag
        EOR     #$FF                    ; toggle flag
        STA     LAB_12          ; save the comparison evaluation flag
LAB_E29A
        JSR     LAB_DFB4                ; do - FAC1
LAB_E29D
        LDA     #<LAB_E2E7              ; set 0.25 pointer low byte
        LDY     #>LAB_E2E7              ; set 0.25 pointer high byte
        JSR     LAB_D867                ; add (AY) to FAC1
        PLA                             ; restore FAC1 sign
        BPL     LAB_E2AA                ; branch if was +ve

                                        ; else correct FAC1
        JSR     LAB_DFB4                ; do - FAC1
LAB_E2AA
        LDA     #<LAB_E2EC              ; set pointer low byte to counter
        LDY     #>LAB_E2EC              ; set pointer high byte to counter
        JMP     LAB_E040                ; ^2 then series evaluation and return


;**************************************************************************;
;
; perform TAN()

LAB_E2B1
        JSR     LAB_DBCA                ; pack FAC1 into LAB_57
        LDA     #$00                    ; clear A
        STA     LAB_12          ; clear the comparison evaluation flag
        JSR     LAB_E268                ; perform SIN()
        LDX     #<LAB_4E                ; set sin(n) pointer low byte
        LDY     #>LAB_4E                ; set sin(n) pointer high byte
        JSR     LAB_E0F3                ; pack FAC1 into (XY)
        LDA     #<LAB_57                ; set n pointer low byte
        LDY     #>LAB_57                ; set n pointer high byte
        JSR     LAB_DBA2                ; unpack memory (AY) into FAC1
        LDA     #$00                    ; clear byte
        STA     LAB_66          ; clear FAC1 sign (b7)
        LDA     LAB_12          ; get the comparison evaluation flag
        JSR     LAB_E2D9                ; save flag and go do series evaluation
        LDA     #<LAB_4E                ; set sin(n) pointer low byte
        LDY     #>LAB_4E                ; set sin(n) pointer high byte
        JMP     LAB_DB0F                ; convert AY and do (AY)/FAC1


;**************************************************************************;
;
; save comparison flag and do series evaluation
```

```
LAB_E2D9
        PHA                             ; save comparison flag
        JMP     LAB_E29A                ; add 0.25, ^2 then series evaluation


;******************************************************************************;
;
; constants and series for SIN/COS(n)

LAB_E2DD
        .byte   $81,$49,$0F,$DA,$A2     ; 1.570796371, pi/2, as floating number
LAB_E2E2
        .byte   $83,$49,$0F,$DA,$A2     ; 6.28319, 2*pi, as floating number
LAB_E2E7
        .byte   $7F,$00,$00,$00,$00     ; 0.25

LAB_E2EC
        .byte   $05                             ; series counter
        .byte   $84,$E6,$1A,$2D,$1B     ; -14.3813907
        .byte   $86,$28,$07,$FB,$F8     ;  42.0077971
        .byte   $87,$99,$68,$89,$01     ; -76.7041703
        .byte   $87,$23,$35,$DF,$E1     ;  81.6052237
        .byte   $86,$A5,$5D,$E7,$28     ; -41.3417021
        .byte   $83,$49,$0F,$DA,$A2     ;  6.28318531


;******************************************************************************;
;
; perform ATN()

LAB_E30B
        LDA     LAB_66          ; get FAC1 sign (b7)
        PHA                             ; save sign
        BPL     LAB_E313                ; branch if +ve

        JSR     LAB_DFB4                ; else do - FAC1
LAB_E313
        LDA     LAB_61          ; get FAC1 exponent
        PHA                             ; push exponent
        CMP     #$81                    ; compare with 1
        BCC     LAB_E321                ; branch if FAC1 < 1

        LDA     #<LAB_D9BC              ; pointer to 1 low byte
        LDY     #>LAB_D9BC              ; pointer to 1 high byte
        JSR     LAB_DB0F                ; convert AY and do (AY)/FAC1
LAB_E321
        LDA     #<LAB_E33B              ; pointer to series low byte
        LDY     #>LAB_E33B              ; pointer to series high byte
        JSR     LAB_E040                ; ^2 then series evaluation
        PLA                             ; restore old FAC1 exponent
        CMP     #$81                    ; compare with 1
        BCC     LAB_E334                ; branch if FAC1 < 1

        LDA     #<LAB_E2DD              ; pointer to (pi/2) low byte
        LDY     #>LAB_E2DD              ; pointer to (pi/2) low byte
        JSR     LAB_D850                ; perform subtraction, FAC1 from (AY)
LAB_E334
        PLA                             ; restore FAC1 sign
        BPL     LAB_E33A                ; exit if was +ve
```

```
        JMP     LAB_DFB4                ; else do - FAC1 and return

LAB_E33A
        RTS


;*******************************************************************************;
;
; series for ATN(n)

LAB_E33B
        .byte   $0B                             ; series counter
        .byte   $76,$B3,$83,$BD,$D3     ;-6.84793912e-04
        .byte   $79,$1E,$F4,$A6,$F5     ; 4.85094216e-03
        .byte   $7B,$83,$FC,$B0,$10     ;-0.0161117015
        .byte   $7C,$0C,$1F,$67,$CA     ; 0.034209638
        .byte   $7C,$DE,$53,$CB,$C1     ;-0.054279133
        .byte   $7D,$14,$64,$70,$4C     ; 0.0724571965
        .byte   $7D,$B7,$EA,$51,$7A     ;-0.0898019185
        .byte   $7D,$63,$30,$88,$7E     ; 0.110932413
        .byte   $7E,$92,$44,$99,$3A     ;-0.142839808
        .byte   $7E,$4C,$CC,$91,$C7     ; 0.19999912
        .byte   $7F,$AA,$AA,$AA,$13     ;-0.333333316
        .byte   $81,$00,$00,$00,$00     ; 1.000000000


;*******************************************************************************;
;
; BASIC cold start entry point

LAB_E378
        JSR     LAB_E45B                ; initialise BASIC vector table
        JSR     LAB_E3A4                ; initialise BASIC RAM locations
        JSR     LAB_E404                ; print start up message and initialise memory
pointers
        LDX     #$FB                    ; value for start stack
        TXS                             ; set stack pointer
        JMP     LAB_C474                ; do "READY." warm start


;*******************************************************************************;
;
; character get subroutine for zero page

; the target address for the LDA LAB_EA60 becomes the BASIC execute pointer once the
; block is copied to it's destination, any non zero page address will do at assembly
; time, to assemble a three byte instruction.

; page 0 initialisation table from LAB_0073
; increment and scan memory

LAB_E387
        INC     LAB_7A          ; increment BASIC execute pointer low byte
        BNE     LAB_E38D                ; branch if no carry
                                        ; else
        INC     LAB_7B          ; increment BASIC execute pointer high byte

; page 0 initialisation table from LAB_0079
; scan memory

LAB_E38D
```

```
        LDA     LAB_EA60                ; get byte to scan, address set by call routine
        CMP     #':'                    ; compare with ":"
        BCS     LAB_E39E                ; exit if>=

; page 0 initialisation table from LAB_0080
; clear Cb if numeric

        CMP     #' '                    ; compare with " "
        BEQ     LAB_E387                ; if " " go do next

        SEC                             ; set carry for SBC
        SBC     #'0'                    ; subtract "0"
        SEC                             ; set carry for SBC
        SBC     #$D0                    ; subtract -"0"
                                        ; clear carry if byte = "0"-"9"
LAB_E39E
        RTS


;*******************************************************************************;
;
; spare bytes, not referenced

;LAB_E39F
        .byte   $80,$4F,$C7,$52,$58
                                        ; 0.811635157


;*******************************************************************************;
;
; initialise BASIC RAM locations

LAB_E3A4
        LDA     #$4C                    ; opcode for JMP
        STA     LAB_54          ; save for functions vector jump
        STA     LAB_00          ; save for USR() vector jump
                                        ; set USR() vector to illegal quantity error
        LDA     #<LAB_D248              ; set USR() vector low byte
        LDY     #>LAB_D248              ; set USR() vector high byte
        STA     LAB_01          ; save USR() vector low byte
        STY     LAB_02          ; save USR() vector high byte
        LDA     #<LAB_D391              ; set fixed to float vector low byte
        LDY     #>LAB_D391              ; set fixed to float vector high byte
        STA     LAB_05          ; save fixed to float vector low byte
        STY     LAB_06          ; save fixed to float vector high byte
        LDA     #<LAB_D1AA              ; set float to fixed vector low byte
        LDY     #>LAB_D1AA              ; set float to fixed vector high byte
        STA     LAB_03          ; save float to fixed vector low byte
        STY     LAB_04          ; save float to fixed vector high byte

; copy block from LAB_E387 to LAB_0074

        LDX     #$1C                    ; set byte count
LAB_E3C4
        LDA     LAB_E387,X              ; get byte from table
        STA     LAB_0073,X              ; save byte in page zero
        DEX                             ; decrement count
        BPL     LAB_E3C4                ; loop if not all done

        LDA     #$03                    ; set step size, collecting descriptors
        STA     LAB_53          ; save garbage collection step size
```

```
        LDA     #$00                    ; clear A
        STA     LAB_68          ; clear FAC1 overflow byte
        STA     LAB_13          ; clear current I/O channel, flag default
        STA     LAB_18          ; clear current descriptor stack item pointer high byte
        LDX     #$01                    ; set X
        STX     LAB_01FD                ; set chain link pointer low byte
        STX     LAB_01FC                ; set chain link pointer high byte
        LDX     #LAB_19         ; initial value for descriptor stack
        STX     LAB_16          ; set descriptor stack pointer
        SEC                             ; set Cb = 1 to read the bottom of memory
        JSR     LAB_FF9C                ; read/set the bottom of memory
        STX     LAB_2B          ; save start of memory low byte
        STY     LAB_2C          ; save start of memory high byte
        SEC                             ; set Cb = 1 to read the top of memory
        JSR     LAB_FF99                ; read/set the top of memory
        STX     LAB_37          ; save end of memory low byte
        STY     LAB_38          ; save end of memory high byte
        STX     LAB_33          ; set bottom of string space low byte
        STY     LAB_34          ; set bottom of string space high byte
        LDY     #$00                    ; clear index
        TYA                             ; clear A
        STA     (LAB_2B),Y              ; clear first byte of memory
        INC     LAB_2B          ; increment start of memory low byte
        BNE     LAB_E403                ; branch if no rollover

        INC     LAB_2C          ; increment start of memory high byte
LAB_E403
        RTS


;*******************************************************************************;
;
; print start up message and initialise memory pointers

LAB_E404
        LDA     LAB_2B          ; get start of memory low byte
        LDY     LAB_2C          ; get start of memory high byte
        JSR     LAB_C408                ; check available memory, do out of memory error if no
room
        LDA     #<LAB_E436              ; set "**** CBM BASIC V2 ****" pointer low byte
        LDY     #>LAB_E436              ; set "**** CBM BASIC V2 ****" pointer high byte
        JSR     LAB_CB1E                ; print null terminated string
        LDA     LAB_37          ; get end of memory low byte
        SEC                             ; set carry for subtract
        SBC     LAB_2B          ; subtract start of memory low byte
        TAX                             ; copy result to X
        LDA     LAB_38          ; get end of memory high byte
        SBC     LAB_2C          ; subtract start of memory high byte
        JSR     LAB_DDCD                ; print XA as unsigned integer
        LDA     #<LAB_E429              ; set " BYTES FREE" pointer low byte
        LDY     #>LAB_E429              ; set " BYTES FREE" pointer high byte
        JSR     LAB_CB1E                ; print null terminated string
        JMP     LAB_C644                ; do NEW, CLEAR, RESTORE and return


;*******************************************************************************;
;
LAB_E429
        .byte   " BYTES FREE",$0D,$00
LAB_E436
        .byte   $93,"**** CBM BASIC V2 ****",$0D,$00
```

```
;**************************************************************************;
;
; BASIC vectors, these are copied to RAM from LAB_0300 onwards

LAB_E44F
        .word   LAB_C43A                ; error message                         LAB_0300
        .word   LAB_C483                ; BASIC warm start                      LAB_0302
        .word   LAB_C57C                ; crunch BASIC tokens                   LAB_0304
        .word   LAB_C71A                ; uncrunch BASIC tokens                 LAB_0306
        .word   LAB_C7E4                ; start new BASIC code                  LAB_0308
        .word   LAB_CE86                ; get arithmetic element                LAB_030A


;**************************************************************************;
;
; initialise BASIC vectors

LAB_E45B
        LDX     #$0B                    ; set byte count
LAB_E45D
        LDA     LAB_E44F,X              ; get byte from table
        STA     LAB_0300,X              ; save byte to RAM
        DEX                             ; decrement index
        BPL     LAB_E45D                ; loop if more to do

        RTS


;**************************************************************************;
;
; BASIC warm start entry point

LAB_E467
        JSR     LAB_FFCC                ; close input and output channels
        LDA     #$00                    ; clear A
        STA     LAB_13          ; set current I/O channel, flag default
        JSR     LAB_C67A                ; flush BASIC stack and clear continue pointer
        CLI                             ; enable interrupts
        JMP     LAB_C474                ; do warm start


;**************************************************************************;
;
; checksum byte, not referenced

;LAB_E475
        .byte   $E8                     ; [PAL]
;       .byte   $41                     ; [NTSC]


;**************************************************************************;
;
; rebuild BASIC line chaining and do RESTORE

LAB_E476
        JSR     LAB_C533                ; rebuild BASIC line chaining
        JMP     LAB_C677                ; do RESTORE, clear stack and return
```

```
;*******************************************************************************
;
; spare bytes, not referenced

;LAB_E47C
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF


;*******************************************************************************
;
; set serial data out high

LAB_E4A0
        LDA     LAB_912C                ; get VIA 2 PCR
        AND     #$DF                    ; set CB2 low, serial data out high
        STA     LAB_912C                ; set VIA 2 PCR
        RTS


;*******************************************************************************
;
; set serial data out low

LAB_E4A9
        LDA     LAB_912C                ; get VIA 2 PCR
        ORA     #$20                    ; set CB2 high, serial data out low
        STA     LAB_912C                ; set VIA 2 PCR
        RTS


;*******************************************************************************
;
; get serial clock status

LAB_E4B2
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        CMP     LAB_911F                ; compare with self
        BNE     LAB_E4B2                ; loop if changing

        LSR                             ; shift serial clock to Cb
        RTS


;*******************************************************************************
;
; get seconday address and print "Searching..."

LAB_E4BC
        LDX     LAB_B9          ; get secondary address
        JMP     LAB_F647                ; print "Searching..." and return


;*******************************************************************************
;
; set LOAD address if secondary address = 0

LAB_E4C1
        TXA                             ; copy secondary address
        BNE     LAB_E4CC                ; load location not set in LOAD call, so
```

```
                                        ; continue with load
        LDA     LAB_C3          ; get load address low byte
        STA     LAB_AE          ; save program start address low byte
        LDA     LAB_C4          ; get load address high byte
        STA     LAB_AF          ; save program start address high byte
LAB_E4CC
        JMP     LAB_F66A                ; display "LOADING" or "VERIFYING" and return


;******************************************************************************;
;
; patch for CLOSE

LAB_E4CF
        JSR     LAB_F8E3                ; initiate tape write
        BCC     LAB_E4D7                ; branch if no error

        PLA                             ; else dump stacked exit code
        LDA     #$00                    ; clear exit code
LAB_E4D7
        JMP     LAB_F39E                ; go do I/O close


;******************************************************************************;
;
; spare bytes, not referenced

;LAB_E4DA
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF


;******************************************************************************;
;
; return base address of I/O devices

; this routine will set XY to the address of the memory section where the memory
; mapped I/O devices are located. This address can then be used with an offset to
; access the memory mapped I/O devices in the computer.

LAB_E500
        LDX     #<LAB_9110              ; get I/O base address low byte
        LDY     #>LAB_9110              ; get I/O base address high byte
        RTS


;******************************************************************************;
;
; return X,Y organization of screen

; this routine returns the x,y organisation of the screen in X,Y

LAB_E505
        LDX     #$16                    ; get screen X, 22 columns
        LDY     #$17                    ; get screen Y, 23 rows
        RTS


;******************************************************************************;
;
```

```
; read/set X,Y cursor position, Cb = 1 to read, Cb = 0 to set

; this routine, when called with the carry flag set, loads the current position of
; the cursor on the screen into the X and Y registers. X is the column number of
; the cursor location and Y is the row number of the cursor. A call with the carry
; bit clear moves the cursor to the position determined by the X and Y registers.

LAB_E50A
        BCS     LAB_E513                ; if read cursor skip the set cursor

        STX     LAB_D6          ; save cursor row
        STY     LAB_D3          ; save cursor column
        JSR     LAB_E587                ; set screen pointers for cursor row, column
LAB_E513
        LDX     LAB_D6          ; get cursor row
        LDY     LAB_D3          ; get cursor column
        RTS


;*******************************************************************************;
;
; initialise hardware

LAB_E518
        JSR     LAB_E5BB                ; set default devices and initialise Vic chip
        LDA     LAB_0288                ; get screen memory page
        AND     #$FD                    ; mask xxxx xx0x, all but va9
        ASL                             ; << 1 xxxx x0x0
        ASL                             ; << 2 xxxx 0x00
        ORA     #$80                    ; set  1xxx 0x00
        STA     LAB_9005                ; set screen and character memory location
        LDA     LAB_0288                ; get screen memory page
        AND     #$02                    ; mask bit 9
        BEQ     LAB_E536                ; if zero just go normalise screen

                                        ; else set va9 in vic chip
        LDA     #$80                    ; set b7
        ORA     LAB_9002                ; OR in as video address 9
        STA     LAB_9002                ; save new va9

                                        ; now normalise screen
LAB_E536
        LDA     #$00                    ; clear A
        STA     LAB_0291                ; clear shift mode switch
        STA     LAB_CF          ; clear cursor blink phase
        LDA     #<LAB_EBDC              ; get keyboard decode logic pointer low byte
        STA     LAB_028F                ; set keyboard decode logic pointer low byte
        LDA     #>LAB_EBDC              ; get keyboard decode logic pointer high byte
        STA     LAB_0290                ; set keyboard decode logic pointer high byte
        LDA     #$0A                    ; 10d
        STA     LAB_0289                ; set maximum size of keyboard buffer
        STA     LAB_028C                ; set repeat delay counter
        LDA     #$06                    ; colour blue
        STA     LAB_0286                ; set current colour code
        LDA     #$04                    ; speed 4
        STA     LAB_028B                ; set repeat speed counter
        LDA     #$0C                    ; cursor flash timing
        STA     LAB_CD          ; set cursor timing countdown
        STA     LAB_CC          ; set cursor enable, $00 = flash cursor

; clear screen
```

```
LAB_E55F
        LDA     LAB_0288                ; get screen memory page
        ORA     #$80                    ; set high bit, flag every line is logical line start
        TAY                             ; copy to Y
        LDA     #$00                    ; clear line start low byte
        TAX                             ; clear index
LAB_E568
        STY     LAB_D9,X                ; save start of line X pointer high byte
        CLC                             ; clear carry for add
        ADC     #$16                    ; add line length to low byte
        BCC     LAB_E570                ; if no rollover skip the high byte increment

        INY                             ; else increment high byte
LAB_E570
        INX                             ; increment line index
        CPX     #$18                    ; compare with number of lines + 1
        BNE     LAB_E568                ; loop if not all done

        LDA     #$FF                    ; end of table marker ??
        STA     LAB_D9,X                ; mark end of table
        LDX     #$16                    ; set line count, 23 lines to do, 0 to 22
LAB_E57B
        JSR     LAB_EA8D                ; clear screen line X
        DEX                             ; decrement count
        BPL     LAB_E57B                ; loop if more to do

; home cursor

LAB_E581
        LDY     #$00                    ; clear Y
        STY     LAB_D3          ; clear cursor column
        STY     LAB_D6          ; clear cursor row

; set screen pointers for cursor row, column

LAB_E587
        LDX     LAB_D6          ; get cursor row
        LDA     LAB_D3          ; get cursor column
LAB_E58B
        LDY     LAB_D9,X                ; get start of line X pointer high byte
        BMI     LAB_E597                ; continue if logical line start

        CLC                             ; else clear carry for add
        ADC     #$16                    ; add one line length
        STA     LAB_D3          ; save cursor column
        DEX                             ; decrement cursor row
        BPL     LAB_E58B                ; loop, branch always

LAB_E597
        LDA     LAB_D9,X                ; get start of line X pointer high byte
        AND     #$03                    ; mask 0000 00xx, line memory page
        ORA     LAB_0288                ; OR with screen memory page
        STA     LAB_D2          ; set current screen line pointer high byte
        LDA     LAB_EDFD,X              ; get start of line low byte from ROM table
        STA     LAB_D1          ; set current screen line pointer low byte
        LDA     #$15                    ; set line length
        INX                             ; increment cursor row
LAB_E5A8
        LDY     LAB_D9,X                ; get start of line X pointer high byte
        BMI     LAB_E5B2                ; exit if logical line start
```

```
        CLC                             ; else clear carry for add
        ADC     #$16                    ; add one line length to current line length
        INX                             ; increment cursor row
        BPL     LAB_E5A8                ; loop, branch always

LAB_E5B2
        STA     LAB_D5          ; save current screen line length
        RTS


;*******************************************************************************;
;
; set default devices, initialise Vic chip and home cursor
;
; unreferenced code

;LAB_E5B5
        JSR     LAB_E5BB                ; set default devices and initialise Vic chip
        JMP     LAB_E581                ; home cursor and return


;*******************************************************************************;
;
; set default devices and initialise Vic chip

LAB_E5BB
        LDA     #$03                    ; set screen
        STA     LAB_9A          ; set output device number
        LDA     #$00                    ; set keyboard
        STA     LAB_99          ; set input device number

; initialise Vic chip

LAB_E5C3
        LDX     #$10                    ; set byte count
LAB_E5C5
        LDA     LAB_EDE4-1,X    ; get byte from setup table
        STA     LAB_9000-1,X    ; save byte to Vic chip
        DEX                             ; decrement count/index
        BNE     LAB_E5C5                ; loop if more to do

        RTS


;*******************************************************************************;
;
; input from keyboard buffer

LAB_E5CF
        LDY     LAB_0277                ; get current character from buffer
        LDX     #$00                    ; clear index
LAB_E5D4
        LDA     LAB_0277+1,X    ; get next character,X from buffer
        STA     LAB_0277,X              ; save as current character,X in buffer
        INX                             ; increment index
        CPX     LAB_C6          ; compare with keyboard buffer index
        BNE     LAB_E5D4                ; loop if more to do

        DEC     LAB_C6          ; decrement keyboard buffer index
        TYA                             ; copy key to A
```

```
        CLI                             ; enable interrupts
        CLC                             ; flag got byte
        RTS


;******************************************************************************;
;
; write character and wait for key

LAB_E5E5
        JSR     LAB_E742                ; output character

; wait for key from keyboard

LAB_E5E8
        LDA     LAB_C6          ; get keyboard buffer index
        STA     LAB_CC          ; cursor enable, $00 = flash cursor, $xx = no flash
        STA     LAB_0292                ; screen scrolling flag, $00 = scroll, $xx = no scroll
                                        ; this disables both the cursor flash and the screen
scroll
                                        ; while there are characters in the keyboard buffer
        BEQ     LAB_E5E8                ; loop if buffer empty

        SEI                             ; disable interrupts
        LDA     LAB_CF          ; get cursor blink phase
        BEQ     LAB_E602                ; branch if cursor phase

                                        ; else character phase
        LDA     LAB_CE          ; get character under cursor
        LDX     LAB_0287                ; get colour under cursor
        LDY     #$00                    ; clear Y
        STY     LAB_CF          ; clear cursor blink phase
        JSR     LAB_EAA1                ; print character A and colour X
LAB_E602
        JSR     LAB_E5CF                ; input from keyboard buffer
        CMP     #$83                    ; compare with [SHIFT][RUN]
        BNE     LAB_E619                ; branch if not [SHIFT][RUN]

                                        ; keys are [SHIFT][RUN] so put "LOAD",$0D,"RUN",$0D
into
                                        ; the buffer
        LDX     #$09                    ; set byte count
        SEI                             ; disable interrupts
        STX     LAB_C6          ; set keyboard buffer index
LAB_E60E
        LDA     LAB_EDF4-1,X    ; get byte from auto load/run table
        STA     LAB_0277-1,X    ; save to keyboard buffer
        DEX                             ; decrement count/index
        BNE     LAB_E60E                ; loop while more to do

        BEQ     LAB_E5E8                ; loop for next key, branch always

                                        ; was not [SHIFT][RUN]
LAB_E619
        CMP     #$0D                    ; compare with [CR]
        BNE     LAB_E5E5                ; if not [CR] print character and get next key

                                        ; was [CR]
        LDY     LAB_D5          ; get current screen line length
        STY     LAB_D0          ; input from keyboard or screen, $xx = screen,
                                        ; $00 = keyboard
```

```
LAB_E621
        LDA     (LAB_D1),Y              ; get character from current screen line
        CMP     #' '                    ; compare with [SPACE]
        BNE     LAB_E62A                ; branch if not [SPACE]

        DEY                             ; else eliminate the space, decrement end of input
line
        BNE     LAB_E621                ; loop, branch always

LAB_E62A
        INY                             ; increment past last non space character on line
        STY     LAB_C8          ; save input [EOL] pointer
        LDY     #$00                    ; clear A
        STY     LAB_0292                ; clear screen scrolling flag, $00 = scroll, $xx = no
scroll
        STY     LAB_D3          ; clear cursor column
        STY     LAB_D4          ; clear cursor quote flag, $xx = quote, $00 = no quote
        LDA     LAB_C9          ; get input cursor row
        BMI     LAB_E657                ;.

        LDX     LAB_D6          ; get cursor row
        JSR     LAB_E719                ; find and set pointers for start of logical line
        CPX     LAB_C9          ; compare with input cursor row
        BNE     LAB_E657                ;.

        BNE     LAB_E657                ;.?? what's this? just to make sure or something

        LDA     LAB_CA          ; get input cursor column
        STA     LAB_D3          ; save cursor column
        CMP     LAB_C8          ; compare with input [EOL] pointer
        BCC     LAB_E657                ; branch if less, cursor is in line

        BCS     LAB_E691                ; else cursor is beyond the line end, branch always


;**************************************************************************;
;
; input from screen or keyboard

LAB_E64F
        TYA                             ; copy Y
        PHA                             ; save Y
        TXA                             ; copy X
        PHA                             ; save X
        LDA     LAB_D0          ; input from keyboard or screen, $xx = screen,
                                        ; $00 = keyboard
        BEQ     LAB_E5E8                ; if keyboard go wait for key

LAB_E657
        LDY     LAB_D3          ; get cursor column
        LDA     (LAB_D1),Y              ; get character from the current screen line
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ; just a few wasted cycles.
        NOP                             ;
```

```
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        STA     LAB_D7          ; save temporary last character
        AND     #$3F                    ; mask key bits
        ASL     LAB_D7          ; << temporary last character
        BIT     LAB_D7          ; test it
        BPL     LAB_E67E                ; branch if not [NO KEY]

        ORA     #$80                    ;.
LAB_E67E
        BCC     LAB_E684                ;.

        LDX     LAB_D4          ; get cursor quote flag, $xx = quote, $00 = no quote
        BNE     LAB_E688                ; branch if in quote mode

LAB_E684
        BVS     LAB_E688                ;.

        ORA     #$40                    ;.
LAB_E688
        INC     LAB_D3          ; increment cursor column
        JSR     LAB_E6B8                ; if open quote toggle cursor quote flag
        CPY     LAB_C8          ; compare with input [EOL] pointer
        BNE     LAB_E6A8                ; branch if not at line end

LAB_E691
        LDA     #$00                    ;.
        STA     LAB_D0          ; clear input from keyboard or screen, $xx = screen,
                                        ; $00 = keyboard
        LDA     #$0D                    ; set character [CR]
        LDX     LAB_99          ; get input device number
        CPX     #$03                    ; compare with screen
        BEQ     LAB_E6A3                ; branch if screen

        LDX     LAB_9A          ; get output device number
        CPX     #$03                    ; compare with screen
        BEQ     LAB_E6A6                ; branch if screen

LAB_E6A3
        JSR     LAB_E742                ; output character
LAB_E6A6
        LDA     #$0D                    ; set character [CR]
LAB_E6A8
        STA     LAB_D7          ; save character
        PLA                             ; pull X
        TAX                             ; restore X
        PLA                             ; pull Y
        TAY                             ; restore Y
        LDA     LAB_D7          ; restore character
        CMP     #$DE                    ;.
```

```
        BNE     LAB_E6B6                ;.

        LDA     #$FF                    ;.
LAB_E6B6
        CLC                             ; flag ok
        RTS


;******************************************************************************;
;
; if open quote toggle cursor quote flag

LAB_E6B8
        CMP     #$22                    ; comapre byte with "
        BNE     LAB_E6C4                ; exit if not "

        LDA     LAB_D4          ; get cursor quote flag, $xx = quote, $00 = no quote
        EOR     #$01                    ; toggle it
        STA     LAB_D4          ; save cursor quote flag
        LDA     #$22                    ; restore the "
LAB_E6C4
        RTS


;******************************************************************************;
;
; insert uppercase/graphic character

LAB_E6C5
        ORA     #$40                    ; change to uppercase/graphic
LAB_E6C7
        LDX     LAB_C7          ; get reverse flag
        BEQ     LAB_E6CD                ; branch if not reverse

                                        ; else ..
; insert reversed character

LAB_E6CB
        ORA     #$80                    ; reverse character
LAB_E6CD
        LDX     LAB_D8          ; get insert count
        BEQ     LAB_E6D3                ; branch if none

        DEC     LAB_D8          ; else decrement insert count
LAB_E6D3
        LDX     LAB_0286                ; get current colour code
        JSR     LAB_EAA1                ; print character A and colour X
        JSR     LAB_E6EA                ; advance cursor

; restore registers, set quote flag and exit

LAB_E6DC
        PLA                             ; pull Y
        TAY                             ; restore Y
        LDA     LAB_D8          ; get insert count
        BEQ     LAB_E6E4                ; skip quote flag clear if inserts to do

        LSR     LAB_D4          ; clear cursor quote flag, $xx = quote, $00 = no quote
LAB_E6E4
        PLA                             ; pull X
        TAX                             : restore X
```

```
        PLA                             ; restore A
        CLC                             ;.
        CLI                             ; enable interrupts
        RTS


;******************************************************************************;
;
; advance cursor

LAB_E6EA
        JSR     LAB_E8FA                ; test for line increment
        INC     LAB_D3          ; increment cursor column
        LDA     LAB_D5          ; get current screen line length
        CMP     LAB_D3          ; compare with cursor column
        BCS     LAB_E72C                ; exit if line length >= cursor column

        CMP     #$57                    ; compare with max length
        BEQ     LAB_E723                ; if at max clear column, back cursor up and do
newline

        LDA     LAB_0292                ; get autoscroll flag
        BEQ     LAB_E701                ; branch if autoscroll on

        JMP     LAB_E9F0                ;.else open space on screen

LAB_E701
        LDX     LAB_D6          ; get cursor row
        CPX     #$17                    ; compare with max + 1
        BCC     LAB_E70E                ; if less than max + 1 go add this row to the current
                                        ; logical line

        JSR     LAB_E975                ; else scroll screen
        DEC     LAB_D6          ; decrement cursor row
        LDX     LAB_D6          ; get cursor row

; add this row to the current logical line

LAB_E70E
        ASL     LAB_D9,X                ; shift start of line X pointer high byte
        LSR     LAB_D9,X                ; shift start of line X pointer high byte back,
                                        ; clear b7, start of logical line
        JMP     LAB_ED5B                ; make next screen line start of logical line,
increment
                                        ; line length and set pointers

; add one line length and set pointers for start of line

LAB_E715
        ADC     #$16                    ; add one line length
        STA     LAB_D5          ; save current screen line length

; find and set pointers for start of logical line

LAB_E719
        LDA     LAB_D9,X                ; get start of line X pointer high byte
        BMI     LAB_E720                ; exit loop if start of logical line

        DEX                             ; else back up one line
        BNE     LAB_E719                ; loop if not on first line
```

```
LAB_E720
        JMP     LAB_EA7E                ; set start of line X and return

; clear cursor column, back cursor up one line and do newline

LAB_E723
        DEC     LAB_D6          ; decrement cursor row. if the cursor was incremented past
                                ; the last line then this decrement and the scroll
will
                                ; leave the cursor one line above the botom of the
screen
        JSR     LAB_E8C3                ; do newline
        LDA     #$00                    ; clear A
        STA     LAB_D3          ; clear cursor column
LAB_E72C
        RTS

; back onto previous line if possible

LAB_E72D
        LDX     LAB_D6          ; get cursor row
        BNE     LAB_E737                ; branch if not top row

        STX     LAB_D3          ; clear cursor column
        PLA                             ; dump return address low byte
        PLA                             ; dump return address high byte
        BNE     LAB_E6DC                ; restore registers, set quote flag and exit, branch
always

LAB_E737
        DEX                             ; decrement cursor row
        STX     LAB_D6          ; save cursor row
        JSR     LAB_E587                ; set screen pointers for cursor row, column
        LDY     LAB_D5          ; get current screen line length
        STY     LAB_D3          ; save as cursor column
        RTS


;******************************************************************************;
;
;## output character to screen

LAB_E742
        PHA                             ; save character
        STA     LAB_D7          ; save temporary last character
        TXA                             ; copy X
        PHA                             ; save X
        TYA                             ; copy Y
        PHA                             ; save Y
        LDA     #$00                    ; clear A
        STA     LAB_D0          ; clear input from keyboard or screen, $xx = screen,
                                ; $00 = keyboard
        LDY     LAB_D3          ; get cursor column
        LDA     LAB_D7          ; restore last character
        BPL     LAB_E756                ; branch if unshifted

        JMP     LAB_E800                ; do shifted characters and return

LAB_E756
        CMP     #$0D                    ; compare with [CR]
        BNE     LAB_E75D                ; branch if not [CR]
```

```
        JMP     LAB_E8D8                ; else output [CR] and return

LAB_E75D
        CMP     #' '                    ; compare with [SPACE]
        BCC     LAB_E771                ; branch if < [SPACE]

        CMP     #$60                    ;.
        BCC     LAB_E769                ; branch if $20 to $5F

                                        ; character is $60 or greater
        AND     #$DF                    ;.
        BNE     LAB_E76B                ;.

LAB_E769
        AND     #$3F                    ;.
LAB_E76B
        JSR     LAB_E6B8                ; if open quote toggle cursor direct/programmed flag
        JMP     LAB_E6C7                ;.

                                        ; character was < [SPACE] so is a control character
                                        ; of some sort
LAB_E771
        LDX     LAB_D8          ; get insert count
        BEQ     LAB_E778                ; branch if no characters to insert

        JMP     LAB_E6CB                ; insert reversed character

LAB_E778
        CMP     #$14                    ; compare with [INSERT]/[DELETE]
        BNE     LAB_E7AA                ; branch if not [INSERT]/[DELETE]

        TYA                             ;.
        BNE     LAB_E785                ;.

        JSR     LAB_E72D                ; back onto previous line if possible
        JMP     LAB_E79F                ;.

LAB_E785
        JSR     LAB_E8E8                ; test for line decrement

                                        ; now close up the line
        DEY                             ; decrement index to previous character
        STY     LAB_D3          ; save cursor column
        JSR     LAB_EAB2                ; calculate pointer to colour RAM
LAB_E78E
        INY                             ; increment index to next character
        LDA     (LAB_D1),Y              ; get character from current screen line
        DEY                             ; decrement index to previous character
        STA     (LAB_D1),Y              ; save character to current screen line
        INY                             ; increment index to next character
        LDA     (LAB_F3),Y              ; get colour RAM byte
        DEY                             ; decrement index to previous character
        STA     (LAB_F3),Y              ; save colour RAM byte
        INY                             ; increment index to next character
        CPY     LAB_D5          ; compare with current screen line length
        BNE     LAB_E78E                ; loop if not there yet

LAB_E79F
        LDA     #' '                    ; set [SPACE]
        STA     (LAB_D1),Y              ; clear last character on current screen line
```

```
        LDA     LAB_0286                ; get current colour code
        STA     (LAB_F3),Y              ; save to colour RAM
        BPL     LAB_E7F7                ; branch always

LAB_E7AA
        LDX     LAB_D4          ; get cursor quote flag, $xx = quote, $00 = no quote
        BEQ     LAB_E7B1                ; branch if not quote mode

        JMP     LAB_E6CB                ; insert reversed character

LAB_E7B1
        CMP     #$12                    ; compare with [RVS ON]
        BNE     LAB_E7B7                ; branch if not [RVS ON]

        STA     LAB_C7          ; set reverse flag
LAB_E7B7
        CMP     #$13                    ; compare with [CLR HOME]
        BNE     LAB_E7BE                ; branch if not [CLR HOME]

        JSR     LAB_E581                ; home cursor
LAB_E7BE
        CMP     #$1D                    ; compare with [CURSOR RIGHT]
        BNE     LAB_E7D9                ; branch if not [CURSOR RIGHT]

        INY                             ; increment cursor column
        JSR     LAB_E8FA                ; test for line increment
        STY     LAB_D3          ; save cursor column
        DEY                             ; decrement cursor column
        CPY     LAB_D5          ; compare cursor column with current screen line length
        BCC     LAB_E7D6                ; exit if less

                                        ; else the cursor column is >= the current screen line
                                        ; length so back onto the current line and do a
newline
        DEC     LAB_D6          ; decrement cursor row
        JSR     LAB_E8C3                ; do newline
        LDY     #$00                    ; clear cursor column
LAB_E7D4
        STY     LAB_D3          ; save cursor column
LAB_E7D6
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_E7D9
        CMP     #$11                    ; compare with [CURSOR DOWN]
        BNE     LAB_E7FA                ; branch if not [CURSOR DOWN]

        CLC                             ; clear carry for add
        TYA                             ; copy cursor column
        ADC     #$16                    ; add one line
        TAY                             ; copy back to A
        INC     LAB_D6          ; increment cursor row
        CMP     LAB_D5          ; compare cursor column with current screen line length
        BCC     LAB_E7D4                ; save cursor column and exit if less

        BEQ     LAB_E7D4                ; save cursor column and exit if equal

                                        ; else the cursor has moved beyond the end of this
line
                                        ; so back it up until it's on the start of the logical
line
        DEC     LAB_D6          ; decrement cursor row
```

```
LAB_E7EC
        SBC     #$16                    ; subtract one line
        BCC     LAB_E7F4                ; exit loop if on previous line

        STA     LAB_D3          ; else save cursor column
        BNE     LAB_E7EC                ; loop if not at start of line

LAB_E7F4
        JSR     LAB_E8C3                ; do newline
LAB_E7F7
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_E7FA
        JSR     LAB_E912                ; set the colour from the character in A
        JMP     LAB_ED21                ;.

LAB_E800
        NOP                             ; just a few wasted cycles
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        AND     #$7F                    ; mask 0xxx xxxx, clear b7
        CMP     #$7F                    ; was it $FF before the mask
        BNE     LAB_E81D                ; branch if not

        LDA     #$5E                    ; else make it $5E
LAB_E81D
        NOP                             ; just a few wasted cycles
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        CMP     #' '                    ; compare with [SPACE]
        BCC     LAB_E82A                ; branch if < [SPACE]

        JMP     LAB_E6C5                ; insert uppercase/graphic character and return

                                        ; character was $80 to $9F and is now $00 to $1F
LAB_E82A
        CMP     #$0D                    ; compare with [CR]
        BNE     LAB_E831                ; branch if not [CR]

        JMP     LAB_E8D8                ; else output [CR] and return
```

```
                                        ; was not [CR]
LAB_E831
        LDX     LAB_D4          ; get cursor quote flag, $xx = quote, $00 = no quote
        BNE     LAB_E874                ; branch if quote mode

        CMP     #$14                    ; compare with [INSERT DELETE]
        BNE     LAB_E870                ; branch if not [INSERT DELETE]

        LDY     LAB_D5          ; get current screen line length
        LDA     (LAB_D1),Y              ; get character from current screen line
        CMP     #' '                    ; compare with [SPACE]
        BNE     LAB_E845                ; branch if not [SPACE]

        CPY     LAB_D3          ; compare current column with cursor column
        BNE     LAB_E84C                ; if not cursor column go open up space on line

LAB_E845
        CPY     #$57                    ; compare current column with max line length
        BEQ     LAB_E86D                ; exit if at line end

        JSR     LAB_E9EE                ; else open space on screen
                                        ; now open up space on the line to insert a character
LAB_E84C
        LDY     LAB_D5          ; get current screen line length
        JSR     LAB_EAB2                ; calculate pointer to colour RAM
LAB_E851
        DEY                             ; decrement index to previous character
        LDA     (LAB_D1),Y              ; get character from current screen line
        INY                             ; increment index to next character
        STA     (LAB_D1),Y              ; save character to current screen line
        DEY                             ; decrement index to previous character
        LDA     (LAB_F3),Y              ; get current screen line colour RAM byte
        INY                             ; increment index to next character
        STA     (LAB_F3),Y              ; save current screen line colour RAM byte
        DEY                             ; decrement index to previous character
        CPY     LAB_D3          ; compare with cursor column
        BNE     LAB_E851                ; loop if not there yet

        LDA     #' '                    ; set [SPACE]
        STA     (LAB_D1),Y              ; clear character at cursor position on current screen
line
        LDA     LAB_0286                ; get current colour code
        STA     (LAB_F3),Y              ; save to cursor position on current screen line
colour RAM
        INC     LAB_D8          ; increment insert count
LAB_E86D
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_E870
        LDX     LAB_D8          ; get insert count
        BEQ     LAB_E879                ; branch if no insert space

LAB_E874
        ORA     #$40                    ; change to uppercase/graphic
        JMP     LAB_E6CB                ; insert reversed character

LAB_E879
        CMP     #$11                    ; compare with [CURSOR UP]
        BNE     LAB_E893                ; branch if not [CURSOR UP]
```

```
        LDX     LAB_D6          ; get cursor row
        BEQ     LAB_E8B8                ; branch if on top line

        DEC     LAB_D6          ; decrement cursor row
        LDA     LAB_D3          ; get cursor column
        SEC                             ; set carry for subtract
        SBC     #$16                    ; subtract one line length
        BCC     LAB_E88E                ; branch if stepped back to previous line

        STA     LAB_D3          ; else save cursor column ..
        BPL     LAB_E8B8                ; .. and exit, branch always

LAB_E88E
        JSR     LAB_E587                ; set screen pointers for cursor row, column ..
        BNE     LAB_E8B8                ; .. and exit, branch always

LAB_E893
        CMP     #$12                    ; compare with [RVS OFF]
        BNE     LAB_E89B                ; branch if not [RVS OFF]

        LDA     #$00                    ; clear A
        STA     LAB_C7          ; clear reverse flag
LAB_E89B
        CMP     #$1D                    ; compare with [CURSOR LEFT]
        BNE     LAB_E8B1                ; branch if not [CURSOR LEFT]

        TYA                             ; copy cursor column
        BEQ     LAB_E8AB                ; branch if at start of line

        JSR     LAB_E8E8                ; test for line decrement
        DEY                             ; decrement cursor column
        STY     LAB_D3          ; save cursor column
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_E8AB
        JSR     LAB_E72D                ; back onto previous line if possible
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_E8B1
        CMP     #$13                    ; compare with [CLR]
        BNE     LAB_E8BB                ; branch if not [CLR]

        JSR     LAB_E55F                ; clear screen
LAB_E8B8
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_E8BB
        ORA     #$80                    ; restore b7, colour can only be black, cyan, magenta
                                        ; or yellow
        JSR     LAB_E912                ; set the colour from the character in A
        JMP     LAB_ED30                ;.


;******************************************************************************;
;
; do newline

LAB_E8C3
        LSR     LAB_C9          ; shift >> input cursor row
        LDX     LAB_D6          ; get cursor row
LAB_E8C7
```

```
        INX                             ; increment row
        CPX     #$17                    ; compare with last row + 1
        BNE     LAB_E8CF                ; branch if not last row + 1

        JSR     LAB_E975                ; else scroll screen
LAB_E8CF
        LDA     LAB_D9,X                ; get start of line X pointer high byte
        BPL     LAB_E8C7                ; loop if not start of logical line

        STX     LAB_D6          ; else save cursor row
        JMP     LAB_E587                ; set screen pointers for cursor row, column and
return


;**************************************************************************;
;
; output [CR]

LAB_E8D8
        LDX     #$00                    ; clear X
        STX     LAB_D8          ; clear insert count
        STX     LAB_C7          ; clear reverse flag
        STX     LAB_D4          ; clear cursor quote flag, $xx = quote, $00 = no quote
        STX     LAB_D3          ; clear cursor column
        JSR     LAB_E8C3                ; do newline
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit


;**************************************************************************;
;
; test for line decrement

LAB_E8E8
        LDX     #$04                    ; set count
        LDA     #$00                    ; set column
LAB_E8EC
        CMP     LAB_D3          ; compare with cursor column
        BEQ     LAB_E8F7                ; branch if at start of line

        CLC                             ; else clear carry for add
        ADC     #$16                    ; increment to next line
        DEX                             ; decrement loop count
        BNE     LAB_E8EC                ; loop if more to test

        RTS

LAB_E8F7
        DEC     LAB_D6          ; else decrement cursor row
        RTS


;**************************************************************************;
;
; test for line increment. if at end of line, but not at end of last line, increment the
; cursor row

LAB_E8FA
        LDX     #$04                    ; set count
        LDA     #$15                    ; set column
LAB_E8FE
        CMP     LAB_D3          ; compare with cursor column
```

```
        BEQ     LAB_E909                ; if at end of line test and possibly increment cursor
row

        CLC                             ; else clear carry for add
        ADC     #$16                    ; increment to next line
        DEX                             ; decrement loop count
        BNE     LAB_E8FE                ; loop if more to test

        RTS

                                        ; cursor is at end of line
LAB_E909
        LDX     LAB_D6          ; get cursor row
        CPX     #$17                    ; compare with end of screen
        BEQ     LAB_E911                ; exit if end of screen

        INC     LAB_D6          ; else increment cursor row
LAB_E911
        RTS


;******************************************************************************;
;
; set colour code. enter with the colour character in A. if A does not contain a
; colour character this routine exits without changing the colour

LAB_E912
        LDX     #LAB_E928-LAB_E921
                                        ; set colour code count
LAB_E914
        CMP     LAB_E921,X              ; compare the character with the table code
        BEQ     LAB_E91D                ; if a match go save the colour and exit

        DEX                             ; else decrement the index
        BPL     LAB_E914                ; loop if more to do

        RTS

LAB_E91D
        STX     LAB_0286                ; set current colour code
        RTS


;******************************************************************************;
;
; ASCII colour code table
                                        ; CHR$()        colour
LAB_E921                                ; ------        ------
        .byte   $90                     ;  144  black
        .byte   $05                     ;    5  white
        .byte   $1C                     ;   28  red
        .byte   $9F                     ;  159  cyan
        .byte   $9C                     ;  156  magenta
        .byte   $1E                     ;   30  green
        .byte   $1F                     ;   31  blue
LAB_E928
        .byte   $9E                     ;  158  yellow


;******************************************************************************;
;
```

```
; code conversion, these don't seem to be used anywhere

;LAB_E929
        .byte   $EF,$A1,$DF,$A6,$E1,$B1,$E2,$B2,$E3,$B3,$E4,$B4,$E5,$B5,$E6,$B6
        .byte   $E7,$B7,$E8,$B8,$E9,$B9,$FA,$BA,$FB,$BB,$FC,$BC,$EC,$BD,$FE,$BE
        .byte   $84,$BF,$F7,$C0,$F8,$DB,$F9,$DD,$EA,$DE,$5E,$E0,$5B,$E1,$5D,$E2
        .byte   $40,$B0,$61,$B1,$78,$DB,$79,$DD,$66,$B6,$77,$C0,$70,$F0,$71,$F1
        .byte   $72,$F2,$73,$F3,$74,$F4,$75,$F5,$76,$F6,$7D,$FD


;******************************************************************************;
;
; scroll screen

LAB_E975
        LDA     LAB_AC          ; copy tape buffer start pointer
        PHA                             ; save it
        LDA     LAB_AD          ; copy tape buffer start pointer
        PHA                             ; save it
        LDA     LAB_AE          ; copy tape buffer end pointer
        PHA                             ; save it
        LDA     LAB_AF          ; copy tape buffer end pointer
        PHA                             ; save it
LAB_E981
        LDX     #$FF                    ; set to -1 for pre increment loop
        DEC     LAB_D6          ; decrement cursor row
        DEC     LAB_C9          ; decrement input cursor row
        DEC     LAB_F2          ; decrement screen row marker
LAB_E989
        INX                             ; increment line number
        JSR     LAB_EA7E                ; set start of line X
        CPX     #$16                    ; compare with last line
        BCS     LAB_E99D                ; branch if >= $16

        LDA     LAB_EDFD+1,X    ; get start of next line pointer low byte
        STA     LAB_AC          ; save next line pointer low byte
        LDA     LAB_D9+1,X              ; get start of next line pointer high byte
        JSR     LAB_EA56                ; shift screen line up
        BMI     LAB_E989                ; loop, branch always

LAB_E99D
        JSR     LAB_EA8D                ; clear screen line X

                                        ; now shift up the start of logical line bits
        LDX     #$00                    ; clear index
LAB_E9A2
        LDA     LAB_D9,X                ; get start of line X pointer high byte
        AND     #$7F                    ; clear line X start of logical line bit
        LDY     LAB_D9+1,X              ; get start of next line pointer high byte
        BPL     LAB_E9AC                ; branch if next line not start of line

        ORA     #$80                    ; set line X start of logical line bit
LAB_E9AC
        STA     LAB_D9,X                ; set start of line X pointer high byte
        INX                             ; increment line number
        CPX     #$16                    ; compare with last line
        BNE     LAB_E9A2                ; loop if not last line

        LDA     LAB_D9+$16              ; get start of last line pointer high byte
        ORA     #$80                    ; mark as start of logical line
        STA     LAB_D9+$16              ; set start of last line pointer high byte
```

```
        LDA     LAB_D9          ; get start of first line pointer high byte
        BPL     LAB_E981                ; if not start of logical line loop back and
                                        ; scroll the screen up another line

        INC     LAB_D6          ; increment cursor row
        INC     LAB_F2          ; increment screen row marker
        LDA     #$FB                    ; set keyboard column c2
        STA     LAB_9120                ; set VIA 2 DRB, keyboard column
        LDA     LAB_9121                ; get VIA 2 DRA, keyboard row
        CMP     #$FE                    ; compare with row r0 active, [CTL]
        PHP                             ; save status
        LDA     #$F7                    ; set keyboard column c3
        STA     LAB_9120                ; set VIA 2 DRB, keyboard column
        PLP                             ; restore status
        BNE     LAB_E9DF                ; skip delay if ??

                                        ; first time round the inner loop X will be $16
        LDY     #$00                    ; clear delay outer loop count, do this 256 times
LAB_E9D6
        NOP                             ; waste cycles
        DEX                             ; decrement inner loop count
        BNE     LAB_E9D6                ; loop if not all done

        DEY                             ; decrement outer loop count
        BNE     LAB_E9D6                ; loop if not all done

        STY     LAB_C6          ; clear keyboard buffer index
LAB_E9DF
        LDX     LAB_D6          ; get cursor row
        PLA                             ; pull tape buffer end pointer
        STA     LAB_AF          ; restore it
        PLA                             ; pull tape buffer end pointer
        STA     LAB_AE          ; restore it
        PLA                             ; pull tape buffer pointer
        STA     LAB_AD          ; restore it
        PLA                             ; pull tape buffer pointer
        STA     LAB_AC          ; restore it
        RTS


;******************************************************************************;
;
; open space on screen

LAB_E9EE
        LDX     LAB_D6          ; get cursor row
LAB_E9F0
        INX                             ; increment row
        LDA     LAB_D9,X                ; get start of line X pointer high byte
        BPL     LAB_E9F0                ; branch if not start of logical line

        STX     LAB_F2          ; set screen row marker
        CPX     #$16                    ; compare with last line
        BEQ     LAB_EA08                ; branch if = last line

        BCC     LAB_EA08                ; branch if < last line

                                        ; else was > last line
        JSR     LAB_E975                ; else scroll screen
        LDX     LAB_F2          ; get screen row marker
        DEX                             ; decrement screen row marker
```

```
        DEC     LAB_D6          ; decrement cursor row
        JMP     LAB_E70E                ; add this row to the current logical line and return

LAB_EA08
        LDA     LAB_AC          ; copy tape buffer pointer
        PHA                             ; save it
        LDA     LAB_AD          ; copy tape buffer pointer
        PHA                             ; save it
        LDA     LAB_AE          ; copy tape buffer end pointer
        PHA                             ; save it
        LDA     LAB_AF          ; copy tape buffer end pointer
        PHA                             ; save it
        LDX     #$17                    ; set to end line + 1 for predecrement loop
LAB_EA16
        DEX                             ; decrement line number
        JSR     LAB_EA7E                ; set start of line X
        CPX     LAB_F2          ; compare with screen row marker
        BCC     LAB_EA2C                ; branch if < screen row marker

        BEQ     LAB_EA2C                ; branch if = screen row marker

        LDA     LAB_EDFD-1,X    ; else get start of previous line low byte from ROM table
        STA     LAB_AC          ; save previous line pointer low byte
        LDA     LAB_D9-1,X              ; get start of previous line pointer high byte
        JSR     LAB_EA56                ; shift screen line down
        BMI     LAB_EA16                ; loop, branch always

LAB_EA2C
        JSR     LAB_EA8D                ; clear screen line X
        LDX     #$15                    ;.
LAB_EA31
        CPX     LAB_F2          ;.compare with screen row marker
        BCC     LAB_EA44                ;.

        LDA     LAB_D9+1,X              ;.
        AND     #$7F                    ;.
        LDY     LAB_D9,X                ; get start of line X pointer high byte
        BPL     LAB_EA3F                ;.

        ORA     #$80                    ;.
LAB_EA3F
        STA     LAB_D9+1,X              ;.
        DEX                             ;.
        BNE     LAB_EA31                ;.

LAB_EA44
        LDX     LAB_F2          ;.get screen row marker
        JSR     LAB_E70E                ; add this row to the current logical line
        PLA                             ; pull tape buffer end pointer
        STA     LAB_AF          ; restore it
        PLA                             ; pull tape buffer end pointer
        STA     LAB_AE          ; restore it
        PLA                             ; pull tape buffer pointer
        STA     LAB_AD          ; restore it
        PLA                             ; pull tape buffer pointer
        STA     LAB_AC          ; restore it
        RTS


;*******************************************************************************;
.
```

```
; shift screen line up/down

LAB_EA56
        AND     #$03                    ; mask 0000 00xx, line memory page
        ORA     LAB_0288                ; OR with screen memory page
        STA     LAB_AD          ; save next/previous line pointer high byte
        JSR     LAB_EA6E                ; calculate pointers to screen lines colour RAM
LAB_EA60
        LDY     #$15                    ; set column count
LAB_EA62
        LDA     (LAB_AC),Y              ; get character from next/previous screen line
        STA     (LAB_D1),Y              ; save character to current screen line
        LDA     (LAB_AE),Y              ; get colour from next/previous screen line colour RAM
        STA     (LAB_F3),Y              ; save colour to current screen line colour RAM
        DEY                             ; decrement column index/count
        BPL     LAB_EA62                ; loop if more to do

        RTS


;******************************************************************************;
;
; calculate pointers to screen lines colour RAM

LAB_EA6E
        JSR     LAB_EAB2                ; calculate pointer to current screen line colour RAM
        LDA     LAB_AC          ; get next screen line pointer low byte
        STA     LAB_AE          ; save next screen line colour RAM pointer low byte
        LDA     LAB_AD          ; get next screen line pointer high byte
        AND     #$03                    ; mask 0000 00xx, line memory page
        ORA     #$94                    ; set  1001 01xx, colour memory page
        STA     LAB_AF          ; save next screen line colour RAM pointer high byte
        RTS


;******************************************************************************;
;
; set start of line X

LAB_EA7E
        LDA     LAB_EDFD,X              ; get start of line low byte from ROM table
        STA     LAB_D1          ; set current screen line pointer low byte
        LDA     LAB_D9,X                ; get start of line high byte from RAM table
        AND     #$03                    ; mask 0000 00xx, line memory page
        ORA     LAB_0288                ; OR with screen memory page
        STA     LAB_D2          ; set current screen line pointer high byte
        RTS


;******************************************************************************;
;
; clear screen line X

LAB_EA8D
        LDY     #$15                    ; set number of columns to clear
        JSR     LAB_EA7E                ; set start of line X
        JSR     LAB_EAB2                ; calculate pointer to colour RAM
LAB_EA95
        LDA     #' '                    ; set [SPACE]
        STA     (LAB_D1),Y              ; clear character in current screen line
        LDA     #$01                    ; set colour, blue on white
```

```
        STA     (LAB_F3),Y              ; set colour RAM in current screen line
        DEY                             ; decrement index
        BPL     LAB_EA95                ; loop if more to do

        RTS


;******************************************************************************;
;
; print character A and colour X to screen

LAB_EAA1
        TAY                             ; copy character
        LDA     #$02                    ; set count to $02, usually $14 ??
        STA     LAB_CD          ; set cursor countdown
        JSR     LAB_EAB2                ; calculate pointer to colour RAM
        TYA                             ; get character back

; save character and colour to screen @ cursor

LAB_EAAA
        LDY     LAB_D3          ; get cursor column
        STA     (LAB_D1),Y              ; save character from current screen line
        TXA                             ; copy colour to A
        STA     (LAB_F3),Y              ; save to colour RAM
        RTS


;******************************************************************************;
;
; calculate pointer to colour RAM

LAB_EAB2
        LDA     LAB_D1          ; get current screen line pointer low byte
        STA     LAB_F3          ; save pointer to colour RAM low byte
        LDA     LAB_D2          ; get current screen line pointer high byte
        AND     #$03                    ; mask 0000 00xx, line memory page
        ORA     #$94                    ; set  1001 01xx, colour memory page
        STA     LAB_F4          ; save pointer to colour RAM high byte
        RTS


;******************************************************************************;
;
; update the clock, flash the cursor, control the cassette and scan the keyboard

; IRQ handler

LAB_EABF
        JSR     LAB_FFEA                ; increment real time clock
        LDA     LAB_CC          ; get cursor enable
        BNE     LAB_EAEF                ; branch if not flash cursor

        DEC     LAB_CD          ; else decrement cursor timing countdown
        BNE     LAB_EAEF                ; branch if not done

        LDA     #$14                    ; set count
        STA     LAB_CD          ; save cursor timing countdown
        LDY     LAB_D3          ; get cursor column
        LSR     LAB_CF          ; shift b0 cursor blink phase into carry
        LDX     LAB_0287                ; get colour under cursor
```

```
        LDA     (LAB_D1),Y              ; get character from current screen line
        BCS     LAB_EAEA                ; branch if cursor phase b0 was 1

        INC     LAB_CF          ; set cursor blink phase to 1
        STA     LAB_CE          ; save character under cursor
        JSR     LAB_EAB2                ; calculate pointer to colour RAM
        LDA     (LAB_F3),Y              ; get colour RAM byte
        STA     LAB_0287                ; save colour under cursor
        LDX     LAB_0286                ; get current colour code
        LDA     LAB_CE          ; get character under cursor
LAB_EAEA
        EOR     #$80                    ; toggle b7 of character under cursor
        JSR     LAB_EAAA                ; save character and colour to screen @ cursor
LAB_EAEF
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        AND     #$40                    ; mask cassette switch sense
        BEQ     LAB_EB01                ; branch if cassette sense low

                                        ; cassette sense was high so turn off motor and clear
                                        ; the interlock
        LDY     #$00                    ; clear Y
        STY     LAB_C0          ; clear the tape motor interlock
        LDA     LAB_911C                ; get VIA 1 PCR
        ORA     #$02                    ; set CA2 high, turn off motor
        BNE     LAB_EB0A                ; branch always

                                        ; cassette sense was low so turn on motor, perhaps
LAB_EB01
        LDA     LAB_C0          ; get tape motor interlock
        BNE     LAB_EB12                ; if cassette interlock <> 0 don't turn on motor

        LDA     LAB_911C                ; get VIA 1 PCR
        AND     #$FD                    ; set CA2 low, turn on motor
LAB_EB0A
        BIT     LAB_911E                ; test VIA 1 IER
        BVS     LAB_EB12                ; if T1 interrupt enabled don't change motor state

        STA     LAB_911C                ; set VIA 1 PCR, set CA2 high/low
LAB_EB12
        JSR     LAB_EB1E                ; scan keyboard
        BIT     LAB_9124                ; test VIA 2 T1C_l, clear the timer interrupt flag
        PLA                             ; pull Y
        TAY                             ; restore Y
        PLA                             ; pull X
        TAX                             ; restore X
        PLA                             ; restore A
        RTI


;******************************************************************************;
;
; scan keyboard performs the following ..
;
; 1)    check if key pressed, if not then exit the routine
;
; 2)    init I/O ports of VIA 2 for keyboard scan and set pointers to decode table 1.
;       clear the character counter
;
; 3)    set one line of port B low and test for a closed key on port A by shifting the
;       byte read from the port. if the carry is clear then a key is closed so save the
;       count which is incremented on each shift. check for shift/stop/cbm keys and
```

```
;       flag if closed
;
; 4)    repeat step 3 for the whole matrix
;
; 5)    evaluate the SHIFT/CTRL/C= keys, this may change the decode table selected
;
; 6)    use the key count saved in step 3 as an index into the table selected in step 5
;
; 7)    check for key repeat operation
;
; 8)    save the decoded key to the buffer if first press or repeat

; scan keyboard

; this routine will scan the keyboard and check for pressed keys. It is the same
; routine called by the interrupt handler. If a key is down, its ASCII value is
; placed in the keyboard queue.

LAB_EB1E
        LDA     #$00                    ; clear A
        STA     LAB_028D                ; clear keyboard shift/control/c= flag
        LDY     #$40                    ; set no key
        STY     LAB_CB          ; save which key
        STA     LAB_9120                ; clear VIA 2 DRB, keyboard column
        LDX     LAB_9121                ; get VIA 2 DRA, keyboard row
        CPX     #$FF                    ; compare with all bits set
        BEQ     LAB_EB8F                ; if no key pressed clear current key and exit (does
                                        ; further BEQ to LAB_EBBA)

        LDA     #$FE                    ; set column 0 low
        STA     LAB_9120                ; set VIA 2 DRB, keyboard column
        LDY     #$00                    ; clear key count
        LDA     #<LAB_EC5E              ; get decode table low byte
        STA     LAB_F5          ; set keyboard pointer low byte
        LDA     #>LAB_EC5E              ; get decode table high byte
        STA     LAB_F6          ; set keyboard pointer high byte
LAB_EB40
        LDX     #$08                    ; set row count
        LDA     LAB_9121                ; get VIA 2 DRA, keyboard row
        CMP     LAB_9121                ; compare with itself
        BNE     LAB_EB40                ; loop if changing

LAB_EB4A
        LSR                             ; shift row to Cb
        BCS     LAB_EB63                ; if no key closed on this row go do next row

        PHA                             ; save row
        LDA     (LAB_F5),Y              ; get character from decode table
        CMP     #$05                    ; compare with $05, there is no $05 key but the
control
                                        ; keys are all less than $05
        BCS     LAB_EB60                ; if not shift/control/c=/stop go save key count

                                        ; else was shift/control/c=/stop key
        CMP     #$03                    ; compare with $03, stop
        BEQ     LAB_EB60                ; if stop go save key count and continue

                                        ; character is $01 - shift, $02 - c= or $04 - control
        ORA     LAB_028D                ; OR keyboard shift/control/c= flag
        STA     LAB_028D                ; save keyboard shift/control/c= flag
        BPL     LAB_EB62                ; skip save key, branch always
```

```
LAB_EB60
        STY     LAB_CB          ; save key count
LAB_EB62
        PLA                             ; restore row
LAB_EB63
        INY                             ; increment key count
        CPY     #$41                    ; compare with max+1
        BCS     LAB_EB71                ; exit loop if >= max+1

                                        ; else still in matrix
        DEX                             ; decrement row count
        BNE     LAB_EB4A                ; loop if more rows to do

        SEC                             ; set carry for keyboard column shift
        ROL     LAB_9120                ; shift VIA 2 DRB, keyboard column
        BNE     LAB_EB40                ; loop for next column, branch always

LAB_EB71
        JMP     (LAB_028F)              ; evaluate the SHIFT/CTRL/C= keys, LAB_EBDC

; key decoding continues here after the SHIFT/CTRL/C= keys are evaluated

LAB_EB74
        LDY     LAB_CB          ; get saved key count
        LDA     (LAB_F5),Y              ; get character from decode table
        TAX                             ; copy character to X
        CPY     LAB_C5          ; compare key count with last key count
        BEQ     LAB_EB84                ; if this key = current key, key held, go test repeat

        LDY     #$10                    ; set repeat delay count
        STY     LAB_028C                ; save repeat delay count
        BNE     LAB_EBBA                ; go save key to buffer and exit, branch always

LAB_EB84
        AND     #$7F                    ; clear b7
        BIT     LAB_028A                ; test key repeat
        BMI     LAB_EBA1                ; branch if repeat all

        BVS     LAB_EBD6                ; branch if repeat none

        CMP     #$7F                    ; compare with end marker
LAB_EB8F
        BEQ     LAB_EBBA                ; if $00/end marker go save key to buffer and exit

        CMP     #$14                    ; compare with [INSERT]/[DELETE]
        BEQ     LAB_EBA1                ; if [INSERT]/[DELETE] go test for repeat

        CMP     #' '                    ; compare with [SPACE]
        BEQ     LAB_EBA1                ; if [SPACE] go test for repeat

        CMP     #$1D                    ; compare with [CURSOR RIGHT]
        BEQ     LAB_EBA1                ; if [CURSOR RIGHT] go test for repeat

        CMP     #$11                    ; compare with [CURSOR DOWN]
        BNE     LAB_EBD6                ; if not [CURSOR DOWN] just exit

                                        ; was one of the cursor movement keys, insert/delete
                                        ; key or the space bar so always do repeat tests
LAB_EBA1
        LDY     LAB_028C                ; get repeat delay counter
```

```
        BEQ     LAB_EBAB                ; branch if delay expired

        DEC     LAB_028C                ; else decrement repeat delay counter
        BNE     LAB_EBD6                ; branch if delay not expired

                                        ; repeat delay counter has expired
LAB_EBAB
        DEC     LAB_028B                ; decrement repeat speed counter
        BNE     LAB_EBD6                ; branch if repeat speed count not expired

        LDY     #$04                    ; set for 4/60ths of a second
        STY     LAB_028B                ; set repeat speed counter
        LDY     LAB_C6          ; get keyboard buffer index
        DEY                             ; decrement it
        BPL     LAB_EBD6                ; if the buffer isn't empty just exit

                                        ; else repeat the key immediately

; possibly save the key to the keyboard buffer. if there was no key pressed or the key
; was not found during the scan (possibly due to key bounce) then X will be $FF here

LAB_EBBA
        LDY     LAB_CB          ; get the key count
        STY     LAB_C5          ; save as the current key count
        LDY     LAB_028D                ; get keyboard shift/control/c= flag
        STY     LAB_028E                ; save as last keyboard shift pattern
        CPX     #$FF                    ; compare character with table end marker or no key
        BEQ     LAB_EBD6                ; if table end marker or no key just exit

        TXA                             ; copy character to A
        LDX     LAB_C6          ; get keyboard buffer index
        CPX     LAB_0289                ; compare with keyboard buffer size
        BCS     LAB_EBD6                ; if buffer full just exit

        STA     LAB_0277,X              ; save character to keyboard buffer
        INX                             ; increment index
        STX     LAB_C6          ; save keyboard buffer index
LAB_EBD6
        LDA     #$F7                    ; enable column 3 for stop key
        STA     LAB_9120                ; set VIA 2 DRB, keyboard column
        RTS

; evaluate SHIFT/CTRL/C= keys
;
; 0     $00     EC5E
; 1     $02     EC9F
; 2     $04     ECE0
; 3     ..      ....
; 4     $06     EDA3
; 5     $06     EDA3
; 6     $06     EDA3
; 7     $06     EDA3

LAB_EBDC
        LDA     LAB_028D                ; get keyboard shift/control/c= flag
        CMP     #$03                    ; compare with [SHIFT][C=]
        BNE     LAB_EC0F                ; branch if not [SHIFT][C=]

        CMP     LAB_028E                ; compare with last
        BEQ     LAB_EBD6                ; exit if still the same
```

```
        LDA     LAB_0291                ; get shift mode switch $00 = enabled, $80 = locked
        BMI     LAB_EC43                ; if locked continue keyboard decode

        NOP                             ; just a few wasted cycles
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;

                                        ; toggle text mode
        LDA     LAB_9005                ; get start of character memory, ROM
        EOR     #$02                    ; toggle $8000,$8800
        STA     LAB_9005                ; set start of character memory, ROM
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        JMP     LAB_EC43                ; continue keyboard decode

                                        ; was not [SHIFT][C=] but could be any other
combination
LAB_EC0F
        ASL                             ; << 1
        CMP     #$08                    ; compare with [CTRL]
        BCC     LAB_EC18                ; branch if not [CTRL] pressed

        LDA     #$06                    ; else [CTRL] was pressed so make index = $06
        NOP                             ;
        NOP                             ;
LAB_EC18
        NOP                             ; just a few wasted cycles
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
```

```
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
        TAX                             ; copy index to X
        LDA     LAB_EC46,X              ; get decode table pointer low byte
        STA     LAB_F5          ; save decode table pointer low byte
        LDA     LAB_EC46+1,X    ; get decode table pointer high byte
        STA     LAB_F6          ; save decode table pointer high byte
LAB_EC43
        JMP     LAB_EB74                ; continue keyboard decode


;******************************************************************************;
;
; keyboard decode table pointers

LAB_EC46
        .word   LAB_EC5E                ; unshifted
        .word   LAB_EC9F                ; shifted
        .word   LAB_ECE0                ; commodore
        .word   LAB_EDA3                ; control
        .word   LAB_EC5E                ; unshifted
        .word   LAB_EC9F                ; shifted
        .word   LAB_ED69                ; shfited
        .word   LAB_EDA3                ; control
        .word   LAB_ED21                ; graphics/text control
        .word   LAB_ED69                ; shifted
        .word   LAB_ED69                ; shifted
        .word   LAB_EDA3                ; control

; keyboard decode table - unshifted

LAB_EC5E
        .byte   $31,$33,$35,$37,$39,$2B,$5C,$14
        .byte   $5F,$57,$52,$59,$49,$50,$2A,$0D
        .byte   $04,$41,$44,$47,$4A,$4C,$3B,$1D
        .byte   $03,$01,$58,$56,$4E,$2C,$2F,$11
        .byte   $20,$5A,$43,$42,$4D,$2E,$01,$85
        .byte   $02,$53,$46,$48,$4B,$3A,$3D,$86
        .byte   $51,$45,$54,$55,$4F,$40,$5E,$87
        .byte   $32,$34,$36,$38,$30,$2D,$13,$88
        .byte   $FF

; keyboard decode table - shifted

LAB_EC9F
        .byte   $21,$23,$25,$27,$29,$DB,$A9,$94
        .byte   $5F,$D7,$D2,$D9,$C9,$D0,$C0,$8D
        .byte   $04,$C1,$C4,$C7,$CA,$CC,$5D,$9D
```

```
        .byte   $83,$01,$D8,$D6,$CE,$3C,$3F,$91
        .byte   $A0,$DA,$C3,$C2,$CD,$3E,$01,$89
        .byte   $02,$D3,$C6,$C8,$CB,$5B,$3D,$8A
        .byte   $D1,$C5,$D4,$D5,$CF,$BA,$DE,$8B
        .byte   $22,$24,$26,$28,$30,$DD,$93,$8C
        .byte   $FF

; keyboard decode table - commodore

LAB_ECE0
        .byte   $21,$23,$25,$27,$29,$A6,$A8,$94
        .byte   $5F,$B3,$B2,$B7,$A2,$AF,$DF,$8D
        .byte   $04,$B0,$AC,$A5,$B5,$B6,$5D,$9D
        .byte   $83,$01,$BD,$BE,$AA,$3C,$3F,$91
        .byte   $A0,$AD,$BC,$BF,$A7,$3E,$01,$89
        .byte   $02,$AE,$BB,$B4,$A1,$5B,$3D,$8A
        .byte   $AB,$B1,$A3,$B8,$B9,$A4,$DE,$8B
        .byte   $22,$24,$26,$28,$30,$DC,$93,$8C
        .byte   $FF


;******************************************************************************;
;
;## graphics/text control

LAB_ED21
        CMP     #$0E                    ; compare with [SWITCH TO LOWER CASE]
        BNE     LAB_ED30                ; branch if not [SWITCH TO LOWER CASE]

        LDA     #$02                    ; set for $8800, lower case characters
        ORA     LAB_9005                ; OR with start of character memory, ROM
        STA     LAB_9005                ; save start of character memory, ROM
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_ED30
        CMP     #$8E                    ; compare with [SWITCH TO UPPER CASE]
        BNE     LAB_ED3F                ; branch if not [SWITCH TO UPPER CASE]

        LDA     #$FD                    ; set for $8000, upper case characters
        AND     LAB_9005                ; AND with start of character memory, ROM
        STA     LAB_9005                ; save start of character memory, ROM

LAB_ED3C
        JMP     LAB_E6DC                ; restore registers, set quote flag and exit

LAB_ED3F
        CMP     #$08                    ; compare with disable [SHIFT][C=]
        BNE     LAB_ED4D                ; branch if not disable [SHIFT][C=]

        LDA     #$80                    ; set to lock shift mode switch
        ORA     LAB_0291                ; OR with shift mode switch, $00 = enabled, $80 =
locked
        STA     LAB_0291                ; save shift mode switch
        BMI     LAB_ED3C                ; branch always

LAB_ED4D
        CMP     #$09                    ; compare with enable [SHIFT][C=]
        BNE     LAB_ED3C                ; exit if not enable [SHIFT][C=]

        LDA     #$7F                    ; set to unlock shift mode switch
        AND     LAB_0291                ; AND with shift mode switch, $00 = enabled, $80 =
```

```
locked
        STA     LAB_0291                ; save shift mode switch
        BPL     LAB_ED3C                ; branch always

; make next screen line start of logical line, increment line length and set pointers

LAB_ED5B
        INX                             ; increment screen row
        LDA     LAB_D9,X                ; get start of line X pointer high byte
        ORA     #$80                    ; mark as start of logical line
        STA     LAB_D9,X                ; set start of line X pointer high byte
        DEX                             ; restore screen row
        LDA     LAB_D5          ; get current screen line length
        CLC                             ; clear carry for add
        JMP     LAB_E715                ; add one line length, set pointers for start of line
and
                                        ; return


;******************************************************************************;
;
; keyboard decode table - shifted

LAB_ED69
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$04,$FF,$FF,$FF,$FF,$FF,$E2
        .byte   $9D,$83,$01,$FF,$FF,$FF,$FF,$FF
        .byte   $91,$A0,$FF,$FF,$FF,$FF,$EE,$01
        .byte   $89,$02,$FF,$FF,$FF,$FF,$E1,$FD
        .byte   $8A,$FF,$FF,$FF,$FF,$FF,$B0,$E0
        .byte   $8B,$F2,$F4,$F6,$FF,$F0,$ED,$93
        .byte   $8C,$FF

; keyboard decode table - control

LAB_EDA3
        .byte   $90,$1C,$9C,$1F,$12,$FF,$FF,$FF
        .byte   $06,$FF,$12,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        .byte   $05,$9F,$1E,$9E,$92,$FF,$FF,$FF
        .byte   $FF


;******************************************************************************;
;
; initial values for VIC registers

LAB_EDE4
        .byte   $0C                     ; interlace and horizontal center [PAL]
;       .byte   $05                     ; interlace and horizontal center [NTSC]
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    interlace / non interlace
                                        ; 6-0   horizontal origin
        .byte   $26                     ; vertical origin [PAL]
;       .byte   $19                     ; vertical origin [NTSC]
        .byte   $16                     ; video address and colums, $9400 for colour RAM
```

```
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    video address va9
                                        ; 6-0   number of columns
        .byte   $2E                     ; rows and character size
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    b9 raster line
                                        ; 6-1   number of rows
                                        ;  0    8x16 / 8x8 characters
        .byte   $00                     ; raster line
        .byte   $C0                     ; video memory addresses, RAM $1000, ROM $8000
                                        ; bit   function
                                        ; ---   --------
                                        ;  7    must be 1
                                        ; 6-4   video memory address va12-va10
                                        ; 3-0   character memory start address

                                        ; 0000 ROM      $8000   set 1 - we use this
                                        ; 0001  "       $8400
                                        ; 0010  "       $8800 set 2
                                        ; 0011  "       $8C00
                                        ; 1100 RAM      $1000
                                        ; 1101  "       $1400
                                        ; 1110  "       $1800
                                        ; 1111  "       $1C00

        .byte   $00                     ; light pen horizontal position
        .byte   $00                     ; light pen vertical position

        .byte   $00                     ; paddle X
        .byte   $00                     ; paddle Y
        .byte   $00                     ; oscillator 1 frequency
        .byte   $00                     ; oscillator 2 frequency
        .byte   $00                     ; oscillator 3 frequency
        .byte   $00                     ; noise source frequency
        .byte   $00                     ; aux colour and volume
                                        ; bit   function
                                        ; ---   --------
                                        ; 7-4   auxiliary colour information
                                        ; 3-0   volume
        .byte   $1B                     ; screen and border colour
                                        ; bit   function
                                        ; ---   --------
                                        ; 7-4   background colour
                                        ;  3    inverted or normal mode
                                        ; 2-0   border colour


;******************************************************************************;
;
; keyboard buffer for auto load/run

LAB_EDF4
        .byte   "LOAD",$0D,"RUN",$0D


;******************************************************************************;
;
; low byte screen line addresses
```

```
LAB_EDFD
        .byte   $00,$16,$2C,$42
        .byte   $58,$6E,$84,$9A
        .byte   $B0,$C6,$DC,$F2
        .byte   $08,$1E,$34,$4A
        .byte   $60,$76,$8C,$A2
        .byte   $B8,$CE,$E4


;******************************************************************************;
;
; command a serial bus device to TALK

; to use this routine the accumulator must first be loaded with a device number
; between 4 and 30. When called this routine converts this device number to a talk
; address. Then this data is transmitted as a command on the Serial bus.

LAB_EE14
        ORA     #$40                    ; OR with the TALK command
        .byte   $2C                     ; makes next line BIT LAB_2009


;******************************************************************************;
;
; command devices on the serial bus to LISTEN

; this routine will command a device on the serial bus to receive data. The
; accumulator must be loaded with a device number between 4 and 31 before calling
; this routine. LISTEN convert this to a listen address then transmit this data as
; a command on the serial bus. The specified device will then go into listen mode
; and be ready to accept information.

LAB_EE17
        ORA     #$20                    ; OR with the LISTEN command
        JSR     LAB_F160                ; check RS232 bus idle


;******************************************************************************;
;
; send control character

LAB_EE1C
        PHA                             ; save device address
        BIT     LAB_94          ; test deferred character flag
        BPL     LAB_EE2B                ; branch if no defered character

        SEC                             ; flag EOI
        ROR     LAB_A3          ; rotate into EOI flag byte
        JSR     LAB_EE49                ; Tx byte on serial bus
        LSR     LAB_94          ; clear deferred character flag
        LSR     LAB_A3          ; clear EOI flag
LAB_EE2B
        PLA                             ; restore device address
        STA     LAB_95          ; save as serial defered character
        JSR     LAB_E4A0                ; set serial data out high
        CMP     #$3F                    ; compare read byte with $3F
        BNE     LAB_EE38                ; branch if not $3F, this branch will always be taken
as
                                        ; after VIA 2's PCR is read it is ANDed with $DF, so
the
                                        ; result can never be $3F
```

```
        JSR     LAB_EF84                ; set serial clock high
LAB_EE38
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        ORA     #$80                    ; set serial ATN low
        STA     LAB_911F                ; set VIA 1 DRA, no handshake


;******************************************************************************;
;
; if the code drops through to here the serial clock is low and the serial data has been
; released so the following code will have no effect apart from delaying the first byte
; by 1ms

;## set clk/data, wait and Tx byte on serial bus

LAB_EE40
        JSR     LAB_EF8D                ; set serial clock low
        JSR     LAB_E4A0                ; set serial data out high
        JSR     LAB_EF96                ; 1ms delay


;******************************************************************************;
;
; Tx byte on serial bus

LAB_EE49
        SEI                             ; disable interrupts
        JSR     LAB_E4A0                ; set serial data out high
        JSR     LAB_E4B2                ; get serial clock status
        LSR                             ; shift serial data to Cb
        BCS     LAB_EEB4                ; if data high do device not present

        JSR     LAB_EF84                ; set serial clock high
        BIT     LAB_A3          ; test EOI flag
        BPL     LAB_EE66                ; branch if not EOI

; I think this is the EOI sequence so the serial clock has been released and the serial
; data is being held low by the peripherals. first up wait for the serial data to rise

LAB_EE5A
        JSR     LAB_E4B2                ; get serial clock status
        LSR                             ; shift serial data to Cb
        BCC     LAB_EE5A                ; loop if data low

; now the data is high, EOI is signalled by waiting for at least 200us without pulling
; the serial clock line low again. the listener should respond by pulling the serial
; data line low

LAB_EE60
        JSR     LAB_E4B2                ; get serial clock status
        LSR                             ; shift serial data to Cb
        BCS     LAB_EE60                ; loop if data high

; the serial data has gone low ending the EOI sequence, now just wait for the serial
; data line to go high again or, if this isn't an EOI sequence, just wait for the serial
; data to go high the first time

LAB_EE66
        JSR     LAB_E4B2                ; get serial clock status
        LSR                             ; shift serial data to Cb
```

```
        BCC     LAB_EE66                ; loop if data low

; serial data is high now pull the clock low, preferably within 60us

        JSR     LAB_EF8D                ; set serial clock low

; now the Vic has to send the eight bits, LSB first. first it sets the serial data line
; to reflect the bit in the byte, then it sets the serial clock to high. The serial
; clock is left high for 26 cycles, 23us on a PAL Vic, before it is again pulled low
; and the serial data is allowed high again

        LDA     #$08                    ; eight bits to do
        STA     LAB_A5          ; set serial bus bit count
LAB_EE73
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        CMP     LAB_911F                ; compare with self
        BNE     LAB_EE73                ; loop if changing

        LSR                             ; serial clock to carry
        LSR                             ; serial data to carry
        BCC     LAB_EEB7                ; if data low do timeout on serial bus

        ROR     LAB_95          ; rotate transmit byte
        BCS     LAB_EE88                ; branch if bit = 1

        JSR     LAB_E4A9                ; else set serial data out low
        BNE     LAB_EE8B                ; branch always

LAB_EE88
        JSR     LAB_E4A0                ; set serial data out high
LAB_EE8B
        JSR     LAB_EF84                ; set serial clock high
        NOP                             ; waste ..
        NOP                             ; .. a ..
        NOP                             ; .. cycle ..
        NOP                             ; .. or two
        LDA     LAB_912C                ; get VIA 2 PCR
        AND     #$DF                    ; set CB2 low, serial data out high
        ORA     #$02                    ; set CA2 high, serial clock out low
        STA     LAB_912C                ; save VIA 2 PCR
        DEC     LAB_A5          ; decrement serial bus bit count
        BNE     LAB_EE73                ; loop if not all done

; now all eight bits have been sent it's up to the peripheral to signal the byte was
; received by pulling the serial data low. this should be done within one milisecond

        LDA     #$04                    ; wait for up to about 1ms
        STA     LAB_9129                ; set VIA 2 T2C_h
LAB_EEA5
        LDA     LAB_912D                ; get VIA 2 IFR
        AND     #$20                    ; mask T2 interrupt
        BNE     LAB_EEB7                ; if T2 interrupt do timeout on serial bus

        JSR     LAB_E4B2                ; get serial clock status
        LSR                             ; shift serial data to Cb
        BCS     LAB_EEA5                ; if data high go wait some more

        CLI                             ; enable interrupts
        RTS
```

```
;*******************************************************************************;
;
; device not present

LAB_EEB4
        LDA     #$80                    ; error $80, device not present
        .byte   $2C                     ; makes next line BIT LAB_03A9


;*******************************************************************************;
;
; timeout on serial bus

LAB_EEB7
        LDA     #$03                    ; error $03, write timeout
LAB_EEB9
        JSR     LAB_FE6A                ; OR into serial status byte
        CLI                             ; enable interrupts
        CLC                             ; clear for branch
        BCC     LAB_EF09                ; ATN high, delay, clock high then data high, branch
always


;*******************************************************************************;
;
; send secondary address after LISTEN

; this routine is used to send a secondary address to an I/O device after a call to
; the LISTEN routine is made and the device commanded to LISTEN. The routine cannot
; be used to send a secondary address after a call to the TALK routine.

; A secondary address is usually used to give set-up information to a device before
; I/O operations begin.

; When a secondary address is to be sent to a device on the serial bus the address
; must first be ORed with $60.

LAB_EEC0
        STA     LAB_95          ; save defered byte
        JSR     LAB_EE40                ; set clk/data, wait and Tx byte on serial bus

; set serial ATN high

LAB_EEC5
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        AND     #$7F                    ; set serial ATN high
        STA     LAB_911F                ; set VIA 1 DRA, no handshake
        RTS


;*******************************************************************************;
;
; send secondary address after TALK

; this routine transmits a secondary address on the serial bus for a TALK device.
; This routine must be called with a number between 4 and 31 in the accumulator.
; The routine will send this number as a secondary address command over the serial
; bus. This routine can only be called after a call to the TALK routine. It will
; not work after a LISTEN.

LAB_EECE
```

```
        STA     LAB_95          ; save the secondary address byte to transmit
        JSR     LAB_EE40                ; set clk/data, wait and Tx byte on serial bus


;*******************************************************************************;
;
; wait for bus end after send

LAB_EED3
        SEI                             ; disable interrupts
        JSR     LAB_E4A9                ; set serial data out low
        JSR     LAB_EEC5                ; set serial ATN high
        JSR     LAB_EF84                ; set serial clock high
LAB_EEDD
        JSR     LAB_E4B2                ; get serial clock status
        BCS     LAB_EEDD                ; branch if clock high

        CLI                             ; enable interrupts
        RTS


;*******************************************************************************;
;
; output a byte to the serial bus

; this routine is used to send information to devices on the serial bus. A call to
; this routine will put a data byte onto the serial bus using full handshaking.
; Before this routine is called the LISTEN routine, LAB_FFB1, must be used to
; command a device on the serial bus to get ready to receive data.

; the accumulator is loaded with a byte to output as data on the serial bus. A
; device must be listening or the status word will return a timeout. This routine
; always buffers one character. So when a call to the UNLISTEN routine, LAB_FFAE,
; is made to end the data transmission, the buffered character is sent with EOI
; set. Then the UNLISTEN command is sent to the device.

LAB_EEE4
        BIT     LAB_94          ; test deferred character flag
        BMI     LAB_EEED                ; branch if defered character

        SEC                             ; set carry
        ROR     LAB_94          ; shift into deferred character flag
        BNE     LAB_EEF2                ; save byte and exit, branch always

LAB_EEED
        PHA                             ; save byte
        JSR     LAB_EE49                ; Tx byte on serial bus
        PLA                             ; restore byte
LAB_EEF2
        STA     LAB_95          ; save defered byte
        CLC                             ; flag ok
        RTS


;*******************************************************************************;
;
; command the serial bus to UNTALK

; this routine will transmit an UNTALK command on the serial bus. All devices
; previously set to TALK will stop sending data when this command is received.
```

```
LAB_EEF6
        JSR     LAB_EF8D                ; set serial clock low
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        ORA     #$80                    ; set serial ATN low
        STA     LAB_911F                ; set VIA 1 DRA, no handshake

        LDA     #$5F                    ; set the UNTALK command
        .byte   $2C                     ; makes next line BIT LAB_3FA9


;******************************************************************************;
;
; command the serial bus to UNLISTEN

; this routine commands all devices on the serial bus to stop receiving data from
; the computer. Calling this routine results in an UNLISTEN command being transmitted
; on the serial bus. Only devices previously commanded to listen will be affected.

; This routine is normally used after the computer is finished sending data to
; external devices. Sending the UNLISTEN will command the listening devices to get
; off the serial bus so it can be used for other purposes.

LAB_EF04
        LDA     #$3F                    ; set the UNLISTEN command
        JSR     LAB_EE1C                ; send control character

; ATN high, delay, clock high then data high

LAB_EF09
        JSR     LAB_EEC5                ; set serial ATN high

; 1ms delay, clock high then data high

LAB_EF0C
        TXA                             ; save device number
        LDX     #$0B                    ; short delay
LAB_EF0F
        DEX                             ; decrement count
        BNE     LAB_EF0F                ; loop if not all done

        TAX                             ; restore device number
        JSR     LAB_EF84                ; set serial clock high
        JMP     LAB_E4A0                ; set serial data out high and return


;******************************************************************************;
;
; input a byte from the serial bus

; this routine reads a byte of data from the serial bus using full handshaking. the
; data is returned in the accumulator. before using this routine the TALK routine,
; LAB_FFB4, must have been called first to command the device on the serial bus to
; send data on the bus. if the input device needs a secondary command it must be sent
; by using the TKSA routine, LAB_FF96, before calling this routine.

; errors are returned in the status word which can be read by calling the READST
; routine, LAB_FFB7.

LAB_EF19
        SEI                             ; disable interrupts
        LDA     #$00                    ; clear A
```

```
        STA     LAB_A5          ; clear serial bus bit count
        JSR     LAB_EF84                ; set serial clock high
LAB_EF21
        JSR     LAB_E4B2                ; get serial clock status
        BCC     LAB_EF21                ; loop while clock low

        JSR     LAB_E4A0                ; set serial data out high
LAB_EF29
        LDA     #$01                    ; set timeout count high byte
        STA     LAB_9129                ; set VIA 2 T2C_h
LAB_EF2E
        LDA     LAB_912D                ; get VIA 2 IFR
        AND     #$20                    ; mask T2 interrupt
        BNE     LAB_EF3C                ; branch if T2 interrupt

        JSR     LAB_E4B2                ; get serial clock status
        BCS     LAB_EF2E                ; loop if clock high

        BCC     LAB_EF54                ; else go se 8 bits to do, branch always

                                        ; T2 timed out
LAB_EF3C
        LDA     LAB_A5          ; get serial bus bit count
        BEQ     LAB_EF45                ; if not already EOI then go flag EOI

        LDA     #$02                    ; error $02, read timeour
        JMP     LAB_EEB9                ; set serial status and exit

LAB_EF45
        JSR     LAB_E4A9                ; set serial data out low
        JSR     LAB_EF0C                ; 1ms delay, clock high then data high
        LDA     #$40                    ; set EOI
        JSR     LAB_FE6A                ; OR into serial status byte
        INC     LAB_A5          ; increment serial bus bit count, do error on next timeout
        BNE     LAB_EF29                ; go try again

LAB_EF54
        LDA     #$08                    ; 8 bits to do
        STA     LAB_A5          ; set serial bus bit count
LAB_EF58
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        CMP     LAB_911F                ; compare with self
        BNE     LAB_EF58                ; loop if changing

        LSR                             ; serial clock into carry
        BCC     LAB_EF58                ; loop while serial clock low

        LSR                             ; serial data into carry
        ROR     LAB_A4          ; shift data bit into receive byte
LAB_EF66
        LDA     LAB_911F                ; get VIA 1 DRA, no handshake
        CMP     LAB_911F                ; compare with self
        BNE     LAB_EF66                ; loop if changing

        LSR                             ; serial clock into carry
        BCS     LAB_EF66                ; loop while serial clock high

        DEC     LAB_A5          ; decrement serial bus bit count
        BNE     LAB_EF58                ; loop if not all done

        JSR     LAB_E4A9                ; set serial data out low
```

```
        LDA     LAB_90          ; get serial status byte
        BEQ     LAB_EF7F                ; branch if no error

        JSR     LAB_EF0C                ; 1ms delay, clock high then data high
LAB_EF7F
        LDA     LAB_A4          ; get receive byte
        CLI                             ; enable interrupts
        CLC
        RTS


;*******************************************************************************;
;
; set serial clock high

LAB_EF84
        LDA     LAB_912C                ; get VIA 2 PCR
        AND     #$FD                    ; set CA2 low, serial clock out high
        STA     LAB_912C                ; set VIA 2 PCR
        RTS


;*******************************************************************************;
;
; set serial clock low

LAB_EF8D
        LDA     LAB_912C                ; get VIA 2 PCR
        ORA     #$02                    ; set CA2 high, serial clock out low
        STA     LAB_912C                ; set VIA 2 PCR
        RTS


;*******************************************************************************;
;
; 1ms delay

LAB_EF96
        LDA     #$04                    ; set for 1024 cycles
        STA     LAB_9129                ; set VIA 2 T2C_h
LAB_EF9B
        LDA     LAB_912D                ; get VIA 2 IFR
        AND     #$20                    ; mask T2 interrupt
        BEQ     LAB_EF9B                ; loop until T2 interrupt

        RTS


;*******************************************************************************;
;
; RS232 Tx NMI routine

LAB_EFA3
        LDA     LAB_B4          ; get RS232 bit count
        BEQ     LAB_EFEE                ; if zero go setup next RS232 Tx byte and return

        BMI     LAB_EFE8                ; if -ve go do stop bit(s)

                                        ; else bit count is non zero and +ve
        LSR     LAB_B6          ; shift RS232 output byte buffer
        LDX     #$00                    ; set $00 for bit = 0
```

```
        BCC     LAB_EFB0                ; branch if bit was 0

        DEX                             ; set $FF for bit = 1
LAB_EFB0
        TXA                             ; copy bit to A
        EOR     LAB_BD          ; EOR with RS232 parity byte
        STA     LAB_BD          ; save RS232 parity byte
        DEC     LAB_B4          ; decrement RS232 bit count
        BEQ     LAB_EFBF                ; if RS232 bit count now zero go do parity bit

; save bit and exit

LAB_EFB9
        TXA                             ; copy bit to A
        AND     #$20                    ; mask for CB2 control bit
        STA     LAB_B5          ; save RS232 next bit to send
        RTS

; do RS232 parity bit, enters with RS232 bit count = 0

LAB_EFBF
        LDA     #$20                    ; mask 00x0 0000, parity enable bit
        BIT     LAB_0294                ; test pseudo 6551 command register
        BEQ     LAB_EFDA                ; branch if parity disabled

        BMI     LAB_EFE4                ; branch if fixed mark or space parity

        BVS     LAB_EFDE                ; branch if even parity

                                        ; else odd parity
        LDA     LAB_BD          ; get RS232 parity byte
        BNE     LAB_EFCF                ; if parity not zero leave parity bit = 0

LAB_EFCE
        DEX                             ; make parity bit = 1
LAB_EFCF
        DEC     LAB_B4          ; decrement RS232 bit count, 1 stop bit
        LDA     LAB_0293                ; get pseudo 6551 control register
        BPL     LAB_EFB9                ; if 1 stop bit save parity bit and exit

                                        ; else two stop bits ..
        DEC     LAB_B4          ; decrement RS232 bit count, 2 stop bits
        BNE     LAB_EFB9                ; save bit and exit, branch always

                                        ; parity is disabled so the parity bit becomes the
first,
                                        ; and possibly only, stop bit. to do this increment
the bit
                                        ; count which effectively decrements the stop bit
count.
LAB_EFDA
        INC     LAB_B4          ; increment RS232 bit count, = -1 stop bit
        BNE     LAB_EFCE                ; set stop bit = 1 and exit

                                        ; do even parity
LAB_EFDE
        LDA     LAB_BD          ; get RS232 parity byte
        BEQ     LAB_EFCF                ; if parity zero leave parity bit = 0

        BNE     LAB_EFCE                ; else make parity bit = 1, branch always
```

```
                                        ; fixed mark or space parity
LAB_EFE4
        BVS     LAB_EFCF                ; if fixed space parity leave parity bit = 0

        BVC     LAB_EFCE                ; else fixed mark parity make parity bit = 1, branch
always

; decrement stop bit count, set stop bit = 1 and exit. $FF is one stop bit, $FE is two
; stop bits

LAB_EFE8
        INC     LAB_B4          ; decrement RS232 bit count
        LDX     #$FF                    ; set stop bit = 1
        BNE     LAB_EFB9                ; save stop bit and exit, branch always

; setup next RS232 Tx byte

LAB_EFEE
        LDA     LAB_0294                ; get 6551 pseudo command register
        LSR                             ; handshake bit inot Cb
        BCC     LAB_EFFB                ; branch if 3 line interface

        BIT     LAB_9120                ; test VIA 2 DRB, this is wrong, the adress should be
                                        ; LAB_9110 which is VIA 1 which is where the DSR and
                                        ; CTS inputs really are ##

        BPL     LAB_F016                ; if DSR = 0 set DSR signal not present and exit

        BVC     LAB_F019                ; if CTS = 0 set CTS signal not present and exit

                                        ; was 3 line interface
LAB_EFFB
        LDA     #$00                    ; clear A
        STA     LAB_BD          ; clear RS232 parity byte
        STA     LAB_B5          ; clear RS232 next bit to send
        LDX     LAB_0298                ; get number of bits to be sent/received
        STX     LAB_B4          ; set RS232 bit count
        LDY     LAB_029D                ; get index to Tx buffer start
        CPY     LAB_029E                ; compare with index to Tx buffer end
        BEQ     LAB_F021                ; if all done go disable T1 interrupt and return

        LDA     (LAB_F9),Y              ; else get byte from buffer
        STA     LAB_B6          ; save to RS232 output byte buffer
        INC     LAB_029D                ; increment index to Tx buffer start
        RTS


;******************************************************************************;
;
;## exit or quit
; set DSR signal not present

LAB_F016
        LDA     #$40                    ; set DSR signal not present
        .byte   $2C                     ; makes next line BIT LAB_10A9

; set CTS signal not present

LAB_F019
        LDA     #$10                    ; set CTS signal not present
        ORA     LAB_0297                ; OR with RS232 status register
```

```
        STA     LAB_0297                ; save RS232 status register

; disable T1 interrupt

LAB_F021
        LDA     #$40                    ; disable T1 interrupt
        STA     LAB_911E                ; set VIA 1 IER
        RTS


;******************************************************************************;
;
; compute bit count

LAB_F027
        LDX     #$09                    ; set bit count to 9, 8 data + 1 stop bit
        LDA     #$20                    ; mask for 8/7 data bits
        BIT     LAB_0293                ; test pseudo 6551 control register
        BEQ     LAB_F031                ; branch if 8 bits

        DEX                             ; else decrement count for 7 data bits
LAB_F031
        BVC     LAB_F035                ; branch if 7 bits

        DEX                             ; else decrement count ..
        DEX                             ; .. for 5 data bits
LAB_F035
        RTS


;******************************************************************************;
;
; RS232 Rx NMI

LAB_F036
        LDX     LAB_A9          ; get start bit check flag
        BNE     LAB_F068                ; branch if no start bit received

        DEC     LAB_A8          ; decrement receiver bit count in
        BEQ     LAB_F06F                ;.

        BMI     LAB_F04D                ;.

        LDA     LAB_A7          ; get receiver input bit temporary storage
        EOR     LAB_AB          ;.
        STA     LAB_AB          ;.
        LSR     LAB_A7          ; shift receiver input bit temporary storage
        ROR     LAB_AA          ;.
LAB_F04A
        RTS

LAB_F04B
        DEC     LAB_A8          ; decrement receiver bit count in
LAB_F04D
        LDA     LAB_A7          ; get receiver input bit temporary storage
        BEQ     LAB_F0B3                ;.

        LDA     LAB_0293                ; get pseudo 6551 control register
        ASL                             ;.
        LDA     #$01                    ;.
        ADC     LAB_A8          ; add receiver bit count in
```

```
        BNE     LAB_F04A                ;.


;*******************************************************************************;
;
;## setup to Rx

LAB_F05B
        LDA     #$90                    ; enable CB1 interrupt
        STA     LAB_911E                ; set VIA 1 IER
        STA     LAB_A9          ; set start bit check flag, set no start bit received
        LDA     #$20                    ; disable T2 interrupt
        STA     LAB_911E                ; set VIA 1 IER
        RTS


;*******************************************************************************;
;
; no RS232 start bit received

LAB_F068
        LDA     LAB_A7          ; get receiver input bit temporary storage
        BNE     LAB_F05B                ;.

        STA     LAB_A9          ; set start bit check flag, set start bit received
        RTS


;*******************************************************************************;
;
; ??

LAB_F06F
        LDY     LAB_029B                ; get index to Rx buffer end
        INY                             ; increment index
        CPY     LAB_029C                ; compare with index to Rx buffer start
        BEQ     LAB_F0A2                ; if buffer full go do Rx overrun error

        STY     LAB_029B                ; save index to Rx buffer end
        DEY                             ; decrement index
        LDA     LAB_AA          ; get assembled byte
        LDX     LAB_0298                ; get bit count
LAB_F081
        CPX     #$09                    ; compare with byte + stop
        BEQ     LAB_F089                ; branch if all nine bits received

        LSR                             ; else shift byte
        INX                             ; increment bit count
        BNE     LAB_F081                ; loop, branch always

LAB_F089
        STA     (LAB_F7),Y              ; save received byte to Rx buffer
        LDA     #$20                    ; mask 00x0 0000, parity enable bit
        BIT     LAB_0294                ; test pseudo 6551 command register
        BEQ     LAB_F04B                ; branch if parity disabled

        BMI     LAB_F04A                ; branch if mark or space parity

        LDA     LAB_A7          ; get receiver input bit temporary storage
        EOR     LAB_AB          ;.
        BEQ     LAB_F09D                ;
```

```
        BVS     LAB_F04A                ;.

        .byte   $2C                     ; makes next line BIT LAB_AB50
LAB_F09D
        BVC     LAB_F04A                ;.

        LDA     #$01                    ; set Rx parity error
        .byte   $2C                     ; makes next line BIT LAB_04A9

LAB_F0A2
        LDA     #$04                    ; set Rx overrun error
        .byte   $2C                     ; makes next line BIT LAB_80A9

LAB_F0A5
        LDA     #$80                    ; Rx break error
        .byte   $2C                     ; makes next line BIT LAB_02A9

LAB_F0A8
        LDA     #$02                    ; Rx frame error
        ORA     LAB_0297                ; OR with RS232 status byte
        STA     LAB_0297                ; save RS232 status byte
        JMP     LAB_F05B                ;.

LAB_F0B3
        LDA     LAB_AA          ;.
        BNE     LAB_F0A8                ; if ?? do frame error

        BEQ     LAB_F0A5                ; else do break error, branch always


;******************************************************************************;
;
; do illegal device number

LAB_F0B9
        JMP     LAB_F796                ; do illegal device number and return


;******************************************************************************;
;
; open RS232 channel for output

LAB_F0BC
        STA     LAB_9A          ; save output device number
        LDA     LAB_0294                ; get pseudo 6551 command register
        LSR                             ; shift handshake bit to carry
        BCC     LAB_F0EB                ; branch if 3 line interface

        LDA     #$02                    ; mask for RTS out
        BIT     LAB_9110                ; test VIA 1 DRB
        BPL     LAB_F0E8                ; if DSR = 0 set DSR not present and exit

        BNE     LAB_F0EB                ; if RTS = 1 just exit

LAB_F0CD
        LDA     LAB_911E                ; get VIA 1 IER
        AND     #$30                    ; mask 00xx 0000, T2 and CB1 interrupts
        BNE     LAB_F0CD                ; loop while either enabled

LAB_F0D4
```

```
        BIT     LAB_9110                ; test VIA 1 DRB
        BVS     LAB_F0D4                ; loop while CTS high

        LDA     LAB_9110                ; get VIA 1 DRB
        ORA     #$02                    ; set RTS high
        STA     LAB_9110                ; save VIA 1 DRB
LAB_F0E1
        BIT     LAB_9110                ; test VIA 1 DRB
        BVS     LAB_F0EB                ; exit if CTS high

        BMI     LAB_F0E1                ; loop while DSR high

LAB_F0E8
        JSR     LAB_F016                ; set DSR signal not present
LAB_F0EB
        CLC                             ; flag ok
        RTS


;**************************************************************************************;
;
; send byte to RS232 buffer

LAB_F0ED
        LDY     LAB_029E                ; get index to Tx buffer end
        INY                             ; + 1
        CPY     LAB_029D                ; compare with index to Tx buffer start
        BEQ     LAB_F0ED                ; loop while buffer full

        STY     LAB_029E                ; set index to Tx buffer end
        DEY                             ; index to available buffer byte
        STA     (LAB_F9),Y              ; save byte to buffer
        BIT     LAB_911E                ; test VIA 1 IER
        BVC     LAB_F102                ; branch if T1 not enabled

        RTS

LAB_F102
        LDA     LAB_0299                ; get baud rate bit time low byte
        STA     LAB_9114                ; set VIA 1 T1C_l
        LDA     LAB_029A                ; get baud rate bit time high byte
        STA     LAB_9115                ; set VIA 1 T1C_h
        LDA     #$C0                    ; enable T1 interrupt
        STA     LAB_911E                ; set VIA 1 IER
        JMP     LAB_EFEE                ; setup next RS232 Tx byte and return


;**************************************************************************************;
;
; input from RS232 buffer

LAB_F116
        STA     LAB_99          ; save input device number
        LDA     LAB_0294                ; get pseudo 6551 command register
        LSR                             ;.
        BCC     LAB_F146                ; branch if 3 line interface

        AND     #$08                    ; mask duplex bit, pseudo 6551 command is >> 1
        BEQ     LAB_F146                ; branch if full duplex

        LDA     #$02                    .
```

```
        BIT     LAB_9110                ; test VIA 1 DRB
        BPL     LAB_F0E8                ;.

        BEQ     LAB_F144                ;.

LAB_F12B
        BIT     LAB_911E                ; test VIA 1 IER
        BVS     LAB_F12B                ; loop while T1 interrupt enabled

        LDA     LAB_9110                ; get VIA 1 DRB
        AND     #$FD                    ; mask xxxx xx0x, clear RTS out
        STA     LAB_9110                ; save VIA 1 DRB
LAB_F138
        LDA     LAB_9110                ; get VIA 1 DRB
        AND     #$04                    ; mask xxxx x1xx, DTR
        BEQ     LAB_F138                ; loop while DTR low

LAB_F13F
        LDA     #$90                    ; enable CB1 interrupt
        STA     LAB_911E                ; set VIA 1 IER
LAB_F144
        CLC                             ;.
        RTS

LAB_F146
        LDA     LAB_911E                ; get VIA 1 IER
        AND     #$30                    ; mask 0xx0 0000, T1 and T2 interrupts
        BEQ     LAB_F13F                ; if both interrupts disabled go enable CB1
                                        ; interrupt and exit

        CLC                             ;.
        RTS


;******************************************************************************;
;
; get byte from RS232 buffer

LAB_F14F
        LDY     LAB_029C                ; get index to Rx buffer start
        CPY     LAB_029B                ; compare with index to Rx buffer end
        BEQ     LAB_F15D                ; return null if buffer empty

        LDA     (LAB_F7),Y              ; get byte from Rx buffer
        INC     LAB_029C                ; increment index to Rx buffer start
        RTS

LAB_F15D
        LDA     #$00                    ; return null
        RTS


;******************************************************************************;
;
; check RS232 bus idle

LAB_F160
        PHA                             ; save A
        LDA     LAB_911E                ; get VIA 1 IER
        BEQ     LAB_F172                ; branch if no interrupts enabled. this branch will
                                        ; never be taken as b7 of IER always reads as 1
```

```
                                        ; according to the 6522 data sheet
LAB_F166
        LDA     LAB_911E                ; get VIA 1 IER
        AND     #$60                    ; mask 0xx0 0000, T1 and T2 interrupts
        BNE     LAB_F166                ; loop if T1 or T2 active

        LDA     #$10                    ; disable CB1 interrupt
        STA     LAB_911E                ; set VIA 1 IER
LAB_F172
        PLA                             ; restore A
        RTS


;******************************************************************************;
;
; kernel I/O messages

LAB_F174
        .byte   $0D,"I/O ERROR ",'#'+$80
LAB_F180
        .byte   $0D,"SEARCHING",' '+$80
LAB_F18B
        .byte   "FOR",' '+$80
LAB_F18F
        .byte   $0D,"PRESS PLAY ON TAP",'E'+$80
LAB_F1A2
        .byte   "PRESS RECORD & PLAY ON TAP",'E'+$80
LAB_F1BD
        .byte   $0D,"LOADIN",'G'+$80
LAB_F1C5
        .byte   $0D,"SAVING",' '+$80
LAB_F1CD
        .byte   $0D,"VERIFYIN",'G'+$80
LAB_F1D7
        .byte   $0D,"FOUND",' '+$80
LAB_F1DE
        .byte   $0D,"OK",$0D+$80


;******************************************************************************;
;
; display control I/O message if in direct mode

LAB_F1E2
        BIT     LAB_9D          ; test message mode flag
        BPL     LAB_F1F3                ; exit if control messages off

; display kernel I/O message

LAB_F1E6
        LDA     LAB_F174,Y              ; get byte from message table
        PHP                             ; save status
        AND     #$7F                    ; clear b7
        JSR     LAB_FFD2                ; output character to channel
        INY                             ; increment index
        PLP                             ; restore status
        BPL     LAB_F1E6                ; loop if not end of message

LAB_F1F3
        CLC                             ;.
        RTS
```

```
;*******************************************************************************
;
; get a character from the input device

; in practice this routine operates identically to the CHRIN routine, LAB_FFCF,
; for all devices except for the keyboard. If the keyboard is the current input
; device this routine will get one character from the keyboard buffer. It depends
; on the IRQ routine to read the keyboard and put characters into the buffer.

; If the keyboard buffer is empty the value returned in the accumulator will be zero

LAB_F1F5
        LDA     LAB_99          ; get input device number
        BNE     LAB_F201                ; branch if not keyboard

                                        ; input device was keyboard
        LDA     LAB_C6          ; get keyboard buffer length
        BEQ     LAB_F26A                ; if buffer empty go flag no byte and return

        SEI                             ; disable interrupts
        JMP     LAB_E5CF                ; input from keyboard buffer and return

                                        ; input device was not keyboard
LAB_F201
        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_F21D                ; branch if not RS232 device

                                        ; input device is RS232 device
LAB_F205
        STY     LAB_97          ; save Y
        JSR     LAB_F14F                ; get byte from RS232 buffer
        LDY     LAB_97          ; restore Y
        CLC                             ; flag no error
        RTS


;*******************************************************************************
;
; input character from channel

; this routine will get a byte of data from the channel already set up as the input
; channel by the CHKIN routine, LAB_FFC6.

; If CHKIN, LAB_FFC6, has not been used to define another input channel the data is
; expected to be from the keyboard. the data byte is returned in the accumulator. the
; channel remains open after the call.

; input from the keyboard is handled in a special way. first, the cursor is turned on
; and it will blink until a carriage return is typed on the keyboard. all characters
; on the logical line, up to 88 characters, will be stored in the BASIC input buffer.
; then the characters can be returned one at a time by calling this routine once for
; each character. when the carriage return is returned the entire line has been
; processed. the next time this routine is called the whole process begins again.

LAB_F20E
        LDA     LAB_99          ; get input device number
        BNE     LAB_F21D                ; if it's not the keyboard continue

                                        ; the input device is the keyboard
```

```
        LDA     LAB_D3          ; get cursor column
        STA     LAB_CA          ; set input cursor column
        LDA     LAB_D6          ; get cursor row
        STA     LAB_C9          ; set input cursor row
        JMP     LAB_E64F                ; go get input from the keyboard

; the input device was not the keyboard

LAB_F21D
        CMP     #$03                    ; compare device number with screen
        BNE     LAB_F22A                ; if it's not the screen continue

                                        ; the input device is the screen
        STA     LAB_D0          ; input from keyboard or screen, $xx = screen,
                                        ; $00 = keyboard
        LDA     LAB_D5          ; get current screen line length
        STA     LAB_C8          ; save input [EOL] pointer
        JMP     LAB_E64F                ; go get input from the screen

; the input device was not the screen

LAB_F22A
        BCS     LAB_F264                ; if input device is the serial bus go handle it

; the input device is < the screen do must be the RS232 or tape device

        CMP     #$02                    ; compare device with RS232 device
        BEQ     LAB_F26F                ; if it's the RS232 device go handle it

; else there's only the tape device left ..

        STX     LAB_97          ; save X
        JSR     LAB_F250                ; get byte from tape
        BCS     LAB_F24D                ; exit if error

        PHA                             ; save byte
        JSR     LAB_F250                ; get next byte from tape
        BCS     LAB_F24A                ; exit if error

        BNE     LAB_F244                ; branch if end reached

        LDA     #$40                    ; set [EOF] bit
        JSR     LAB_FE6A                ; OR into serial status byte
LAB_F244
        DEC     LAB_A6          ; decrement tape buffer index
        LDX     LAB_97          ; restore X
        PLA                             ; restore saved byte
        RTS

; error exit from input character

LAB_F24A
        TAX                             ; copy error byte ??
        PLA                             ; dump saved byte
        TXA                             ; restore error byte ??
LAB_F24D
        LDX     LAB_97          ; restore X
        RTS


·*******************************************************************************·
```

```
;
; get byte from tape

LAB_F250
        JSR     LAB_F88A                ; bump tape pointer
        BNE     LAB_F260                ; if not end get next byte and exit

        JSR     LAB_F8C0                ; initiate tape read
        BCS     LAB_F26B                ; exit if error flagged

        LDA     #$00                    ; clear A
        STA     LAB_A6          ; clear tape buffer index
        BEQ     LAB_F250                ; loop, branch always

LAB_F260
        LDA     (LAB_B2),Y              ; get next byte from buffer
        CLC                             ; flag no error
        RTS


;******************************************************************************;
;
; the input device was the serial bus

LAB_F264
        LDA     LAB_90          ; get serial status byte
        BEQ     LAB_F26C                ; if no errors flagged go input byte and return

        LDA     #$0D                    ; else return [EOL]
LAB_F26A
        CLC                             ; flag no error
LAB_F26B
        RTS

LAB_F26C
        JMP     LAB_EF19                ; input a byte from the serial bus and return

                                        ; input device was RS232 device
LAB_F26F
        JSR     LAB_F205                ; get byte from RS232 device
        BCS     LAB_F279                ; branch if error, this doesn't get taken as the last
                                        ; instruction in the get byte from RS232 device
routine
                                        ; is CLC
        CMP     #$00                    ; compare with null
        BEQ     LAB_F26F                ; loop if null

        CLC                             ; flag no error
LAB_F279
        RTS


;******************************************************************************;
;
; output a character to channel

; this routine will output a character to an already opened channel. Use the OPEN
; routine, LAB_FFC0, and the CHKOUT routine, LAB_FFC9, to set up the output channel
; before calling this routine. If these calls are omitted, data will be sent to the
; default output device, device 3, the screen. The data byte to be output is loaded
; into the accumulator, and this routine is called. The data is then sent to the
```

```
; specified output device. The channel is left open after the call.

; NOTE: Care must be taken when using routine to send data to a serial device since
; data will be sent to all open output channels on the bus. Unless this is desired,
; all open output channels on the serial bus other than the actually intended
; destination channel must be closed by a call to the KERNAL close channel routine.

LAB_F27A
        PHA                             ; save the character to send
        LDA     LAB_9A          ; get output device number
        CMP     #$03                    ; compare device number with screen
        BNE     LAB_F285                ; if output device not screen continue

; the output device is the screen

        PLA                             ; restore character to send
        JMP     LAB_E742                ; output character and return

; the output device was not the screen

LAB_F285
        BCC     LAB_F28B                ; if output device < screen continue

; the output device was > screen so it is a serial bus device

        PLA                             ; restore character to send
        JMP     LAB_EEE4                ; output a byte to the serial bus and return

; the output device is < screen

LAB_F28B
        CMP     #$02                    ; compare the device with RS232 device
        BEQ     LAB_F2B9                ; if output device is RS232 device go handle it

; else the output device is the cassette

        PLA                             ; restore the character to send


;*******************************************************************************;
;
; output a character to the cassette

LAB_F290
        STA     LAB_9E          ; save character to character buffer
        PHA                             ; save A
        TXA                             ; copy X
        PHA                             ; save X
        TYA                             ; copy Y
        PHA                             ; save Y
        JSR     LAB_F88A                ; bump tape pointer
        BNE     LAB_F2AA                ; if not end save next byte and exit

        JSR     LAB_F8E3                ; initiate tape write
        BCS     LAB_F2AF                ; exit if error

        LDA     #$02                    ; set data block type ??
        LDY     #$00                    ; clear index
        STA     (LAB_B2),Y              ; save type to buffer ??
        INY                             ; increment index
        STY     LAB_A6          ; save tape buffer index
```

```
LAB_F2AA
        LDA     LAB_9E          ; restore character from character buffer
        STA     (LAB_B2),Y              ; save to buffer
        CLC                             ; flag no error
LAB_F2AF
        PLA                             ; pull Y
        TAY                             ; restore Y
        PLA                             ; pull X
        TAX                             ; restore X
        PLA                             ; restore A
        BCC     LAB_F2B8                ; exit if no error

        LDA     #$00                    ; else clear A
LAB_F2B8
        RTS


;*******************************************************************************;
;
; the output device is RS232 device

LAB_F2B9
        PLA                             ; restore character to send
        STX     LAB_97          ; save X
        STY     LAB_9E          ; save Y
        JSR     LAB_F0ED                ; send byte to RS232 buffer
        LDX     LAB_97          ; restore Y
        LDY     LAB_9E          ; restore X
        CLC                             ; flag ok
        RTS


;*******************************************************************************;
;
; open a channel for input

; any logical file that has already been opened by the OPEN routine, LAB_FFC0, can be
; defined as an input channel by this routine. the device on the channel must be an
; input device or an error will occur and the routine will abort.

; if you are getting data from anywhere other than the keyboard, this routine must be
; called before using either the CHRIN routine, LAB_FFCF, or the GETIN routine,
; LAB_FFE4. if you are getting data from the keyboard and no other input channels are
; open then the calls to this routine and to the OPEN routine, LAB_FFC0, are not needed.

; when used with a device on the serial bus this routine will automatically send the
; listen address specified by the OPEN routine, LAB_FFC0, and any secondary address.

; possible errors are:
;
;       3 : file not open
;       5 : device not present
;       6 : file is not an input file

LAB_F2C7
        JSR     LAB_F3CF                ; find file
        BEQ     LAB_F2CF                ; branch if file opened

        JMP     LAB_F784                ; do file not open error and return

LAB_F2CF
```

```
        JSR     LAB_F3DF                ; set file details from table,X
        LDA     LAB_BA          ; get device number
        BEQ     LAB_F2EC                ; if device was keyboard save device #, flag ok and
exit

        CMP     #$03                    ; compare device number with screen
        BEQ     LAB_F2EC                ; if device was screen save device #, flag ok and exit

        BCS     LAB_F2F0                ; branch if serial bus device

        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_F2E3                ; branch if not RS 232 device

        JMP     LAB_F116                ; else get input from RS232 buffer and return

LAB_F2E3
        LDX     LAB_B9          ; get secondary address
        CPX     #$60                    ;.
        BEQ     LAB_F2EC                ;.

        JMP     LAB_F78D                ; do not input file error and return

LAB_F2EC
        STA     LAB_99          ; save input device number
        CLC                             ; flag ok
        RTS

                                        ; device was serial bus device
LAB_F2F0
        TAX                             ; copy device number to X
        JSR     LAB_EE14                ; command a serial bus device to TALK
        LDA     LAB_B9          ; get secondary address
        BPL     LAB_F2FE                ;.

        JSR     LAB_EED3                ; wait for bus end after send
        JMP     LAB_F301                ;.

LAB_F2FE
        JSR     LAB_EECE                ; send secondary address after TALK
LAB_F301
        TXA                             ; copy device back to A
        BIT     LAB_90          ; test serial status byte
        BPL     LAB_F2EC                ; if device present save device number and exit

        JMP     LAB_F78A                ; do device not present error and return


;******************************************************************************;
;
; open a channel for output

; any logical file that has already been opened by the OPEN routine, LAB_FFC0, can be
; defined as an output channel by this routine the device on the channel must be an
; output device or an error will occur and the routine will abort.

; if you are sending data to anywhere other than the screen this routine must be
; called before using the CHROUT routine, LAB_FFD2. if you are sending data to the
; screen and no other output channels are open then the calls to this routine and to
; the OPEN routine, LAB_FFC0, are not needed.

; when used with a device on the serial bus this routine will automatically send the
```

```
; listen address specified by the OPEN routine, LAB_FFC0, and any secondary address.

; possible errors are:
;
;       3 : file not open
;       5 : device not present
;       7 : file is not an output file

LAB_F309
        JSR     LAB_F3CF                ; find file
        BEQ     LAB_F311                ; branch if file found

        JMP     LAB_F784                ; do file not open error and return

LAB_F311
        JSR     LAB_F3DF                ; set file details from table,X
        LDA     LAB_BA          ; get device number
        BNE     LAB_F31B                ; branch if device is not keyboard

LAB_F318
        JMP     LAB_F790                ; do not output file error and return

LAB_F31B
        CMP     #$03                    ; compare device number with screen
        BEQ     LAB_F32E                ; if screen save output device number and exit

        BCS     LAB_F332                ; branch if > screen, serial bus device

        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_F328                ; branch if not RS232 device, must be tape

        JMP     LAB_F0BC                ; open RS232 channel for output

                                        ; open tape channel for output
LAB_F328
        LDX     LAB_B9          ; get secondary address
        CPX     #$60                    ;.
        BEQ     LAB_F318                ; if ?? do not output file error and return

LAB_F32E
        STA     LAB_9A          ; save output device number
        CLC                             ; flag ok
        RTS

LAB_F332
        TAX                             ; copy device number
        JSR     LAB_EE17                ; command devices on the serial bus to LISTEN
        LDA     LAB_B9          ; get secondary address
        BPL     LAB_F33F                ; branch if address to send

        JSR     LAB_EEC5                ; else set serial ATN high
        BNE     LAB_F342                ; branch always

LAB_F33F
        JSR     LAB_EEC0                ; send secondary address after LISTEN
LAB_F342
        TXA                             ; copy device number back to A
        BIT     LAB_90          ; test serial status byte
        BPL     LAB_F32E                ; if device present save output device number and exit

        JMP     LAB_F78A                ; else do device not present error and return
```

```
;******************************************************************************;
;
; close a specified logical file

; this routine is used to close a logical file after all I/O operations have been
; completed on that file. This routine is called after the accumulator is loaded
; with the logical file number to be closed, the same number used when the file was
; opened using the OPEN routine.

LAB_F34A
        JSR     LAB_F3D4                ; find file A
        BEQ     LAB_F351                ; if the file is found go close it

        CLC                             ; else thr file was closed so just flag ok
        RTS

; found the file so close it

LAB_F351
        JSR     LAB_F3DF                ; set file details from table,X
        TXA                             ; copy file index to A
        PHA                             ; save file index
        LDA     LAB_BA          ; get device number
        BEQ     LAB_F3B1                ; if $00, keyboard, restore index and close file

        CMP     #$03                    ; compare device number with screen
        BEQ     LAB_F3B1                ; if screen restore index and close file

        BCS     LAB_F3AE                ; if > screen go do serial bus device close

        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_F38D                ; branch if not RS232 device

                                        ; else close RS232 device
        PLA                             ; restore file index
        JSR     LAB_F3B2                ; close file index X
        LDA     #$7D                    ; disable T1, T2, CB1, CB2, SR and CA2
        STA     LAB_911E                ; set VIA 1 IER
        LDA     #$06                    ; set DTR and RTS high
        STA     LAB_9110                ; set VIA 1 DRB
        LDA     #$EE                    ; CB2 high, CB1 -ve edge, CA2 high, CA1 -ve edge
        STA     LAB_911C                ; set VIA 1 PCR
        JSR     LAB_FE75                ; read the top of memory
        LDA     LAB_F8          ; get RS232 input buffer pointer high byte
        BEQ     LAB_F37F                ; branch if no RS232 input buffer

        INY                             ; else reclaim RS232 input buffer memory
LAB_F37F
        LDA     LAB_FA          ; get RS232 output buffer pointer high byte
        BEQ     LAB_F384                ; branch if no RS232 output buffer

        INY                             ; else reclaim RS232 output buffer memory
LAB_F384
        LDA     #$00                    ; clear A
        STA     LAB_F8          ; clear RS232 input buffer pointer high byte
        STA     LAB_FA          ; clear RS232 output buffer pointer high byte
        JMP     LAB_F53C                ; go set top of memory and exit

LAB_F38D
```

```
        LDA     LAB_B9          ; get secondary address
        AND     #$0F                    ;.
        BEQ     LAB_F3B1                ; if ?? restore index and close file

        JSR     LAB_F84D                ; get tape buffer start pointer in XY
        LDA     #$00                    ; character $00
        JSR     LAB_F290                ; output character to cassette
        JMP     LAB_E4CF                ; go do CLOSE tail

LAB_F39E
        BCS     LAB_F3CE                ; just exit if error

        LDA     LAB_B9          ; get secondary address
        CMP     #$62                    ;.
        BNE     LAB_F3B1                ; if not ?? restore index and close file

        LDA     #$05                    ; set logical end of the tape
        JSR     LAB_F7E7                ; write tape header
        JMP     LAB_F3B1                ; restore index and close file


;******************************************************************************;
;
; do serial bus device file close

LAB_F3AE
        JSR     LAB_F6DA                ; close serial bus device
LAB_F3B1
        PLA                             ; restore file index


;******************************************************************************;
;
; close file index X

LAB_F3B2
        TAX                             ; copy index to file to close
        DEC     LAB_98          ; decrement open file count
        CPX     LAB_98          ; compare index with open file count
        BEQ     LAB_F3CD                ; exit if equal, last entry was closing file

                                        ; else entry was not last in list so copy last table
entry
                                        ; file details over the details of the closing one
        LDY     LAB_98          ; get open file count as index
        LDA     LAB_0259,Y              ; get last+1 logical file number from logical file
table
        STA     LAB_0259,X              ; save logical file number over closed file
        LDA     LAB_0263,Y              ; get last+1 device number from device number table
        STA     LAB_0263,X              ; save device number over closed file
        LDA     LAB_026D,Y              ; get last+1 secondary address from secondary address
table
        STA     LAB_026D,X              ; save secondary address over closed file
LAB_F3CD
        CLC                             ;.
LAB_F3CE
        RTS


;******************************************************************************;
.
```

```
; find file

LAB_F3CF
        LDA     #$00                    ; clear A
        STA     LAB_90          ; clear serial status byte
        TXA                             ; copy logical file number to A

; find file A

LAB_F3D4
        LDX     LAB_98          ; get open file count
LAB_F3D6
        DEX                             ; decrememnt count to give index
        BMI     LAB_F3EE                ; exit if no files

        CMP     LAB_0259,X              ; compare logical file number with table logical file
number
        BNE     LAB_F3D6                ; loop if no match

        RTS


;*******************************************************************************;
;
; set file details from table,X

LAB_F3DF
        LDA     LAB_0259,X              ; get logical file from logical file table
        STA     LAB_B8          ; set logical file
        LDA     LAB_0263,X              ; get device number from device number table
        STA     LAB_BA          ; set device number
        LDA     LAB_026D,X              ; get secondary address from secondary address table
        STA     LAB_B9          ; set secondary address
LAB_F3EE
        RTS


;*******************************************************************************;
;
; close all channels and files

; this routine closes all open files. When this routine is called, the pointers into
; the open file table are reset, closing all files. Also the routine automatically
; resets the I/O channels.

LAB_F3EF
        LDA     #$00                    ; clear A
        STA     LAB_98          ; clear open file count


;*******************************************************************************;
;
; close input and output channels

; this routine is called to clear all open channels and restore the I/O channels to
; their original default values. It is usually called after opening other I/O
; channels and using them for input/output operations. The default input device is
; 0, the keyboard. The default output device is 3, the screen.

; If one of the channels to be closed is to the serial port, an UNTALK signal is sent
; first to clear the input channel or an UNLISTEN is sent to clear the output channel
```

```
; By not calling this routine and leaving listener(s) active on the serial bus,
; several devices can receive the same data from the VIC at the same time. One way to
; take advantage of this would be to command the printer to TALK and the disk to
; LISTEN. This would allow direct printing of a disk file.

LAB_F3F3
        LDX     #$03                    ; set X to screen
        CPX     LAB_9A          ; compare output device number with screen
        BCS     LAB_F3FC                ; branch if >= screen

                                        ; else was serial bus
        JSR     LAB_EF04                ; command the serial bus to UNLISTEN
LAB_F3FC
        CPX     LAB_99          ; compare input device number with screen
        BCS     LAB_F403                ; branch if >= screen

                                        ; else was serial bus
        JSR     LAB_EEF6                ; command the serial bus to UNTALK
LAB_F403
        STX     LAB_9A          ; set output device number to screen
        LDA     #$00                    ; set for keyboard
        STA     LAB_99          ; set input device number to keyboard
        RTS


;*******************************************************************************;
;
; open a logical file

; this routine is used to open a logical file. Once the logical file is set up it
; can be used for input/output operations. Most of the I/O KERNAL routines call on
; this routine to create the logical files to operate on. No arguments need to be
; set up to use this routine, but both the SETLFS, LAB_FFBA, and SETNAM, LAB_FFBD,
; KERNAL routines must be called before using this routine.

LAB_F40A
        LDX     LAB_B8          ; get logical file number
        BNE     LAB_F411                ; branch if there is a file

        JMP     LAB_F78D                ; else do not input file error and return

LAB_F411
        JSR     LAB_F3CF                ; find file
        BNE     LAB_F419                ; branch if file not found

        JMP     LAB_F781                ; else do file already open error and return

LAB_F419
        LDX     LAB_98          ; get open file count
        CPX     #$0A                    ; compare with max
        BCC     LAB_F422                ; branch if less

        JMP     LAB_F77E                ; else do too many files error and return

LAB_F422
        INC     LAB_98          ; increment open file count
        LDA     LAB_B8          ; get logical file number
        STA     LAB_0259,X              ; save to logical file table
        LDA     LAB_B9          ; get secondary address
        ORA     #$60                    ; OR with the OPEN CHANNEL command
        STA     LAB_B9          ; set secondary address
```

```
        STA     LAB_026D,X              ; save to secondary address table
        LDA     LAB_BA          ; get device number
        STA     LAB_0263,X              ; save to device number table
        BEQ     LAB_F493                ; do ok exit if keyboard

        CMP     #$03                    ; compare device number with screen
        BEQ     LAB_F493                ; do ok exit if screen

        BCC     LAB_F444                ; branch if < screen, tape or RS232

                                        ; else is serial bus device
        JSR     LAB_F495                ; send secondary address and filename
        BCC     LAB_F493                ; do ok exit

LAB_F444
        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_F44B                ; branch if not RS232 device, must be tape

        JMP     LAB_F4C7                ; go open RS232 device and return

LAB_F44B
        JSR     LAB_F84D                ; get tape buffer start pointer in XY
        BCS     LAB_F453                ; branch if >= $0200

        JMP     LAB_F796                ; do illegal device number and return

LAB_F453
        LDA     LAB_B9          ; get secondary address
        AND     #$0F                    ;.
        BNE     LAB_F478                ;.

        JSR     LAB_F894                ; wait for PLAY
        BCS     LAB_F494                ; exit if STOP was pressed

        JSR     LAB_F647                ; print "Searching..."
        LDA     LAB_B7          ; get file name length
        BEQ     LAB_F46F                ; if null file name just go find header

        JSR     LAB_F867                ; find specific tape header
        BCC     LAB_F482                ; branch if no error

        BEQ     LAB_F494                ; exit if ??

LAB_F46C
        JMP     LAB_F787                ; do file not found error and return

LAB_F46F
        JSR     LAB_F7AF                ; find tape header, exit with header in buffer
        BEQ     LAB_F494                ; exit if end of tape found

        BCC     LAB_F482                ;.

        BCS     LAB_F46C                ;.

LAB_F478
        JSR     LAB_F8B7                ; wait for PLAY/RECORD
        BCS     LAB_F494                ; exit if STOP was pressed

        LDA     #$04                    ; set data file header
        JSR     LAB_F7E7                ; write tape header
LAB_F482
```

```
        LDA     #$BF                    ;.
        LDY     LAB_B9          ; get secondary address
        CPY     #$60                    ;.
        BEQ     LAB_F491                ;.

        LDY     #$00                    ; clear index
        LDA     #$02                    ;.
        STA     (LAB_B2),Y              ;.save to tape buffer
        TYA                             ;.clear A
LAB_F491
        STA     LAB_A6          ;.save tape buffer index
LAB_F493
        CLC                             ; flag ok
LAB_F494
        RTS


;******************************************************************************;
;
; send secondary address and filename

LAB_F495
        LDA     LAB_B9          ; get secondary address
        BMI     LAB_F4C5                ; ok exit if -ve

        LDY     LAB_B7          ; get file name length
        BEQ     LAB_F4C5                ; ok exit if null

        LDA     LAB_BA          ; get device number
        JSR     LAB_EE17                ; command devices on the serial bus to LISTEN
        LDA     LAB_B9          ; get the secondary address
        ORA     #$F0                    ; OR with the OPEN command
        JSR     LAB_EEC0                ; send secondary address after LISTEN
        LDA     LAB_90          ; get serial status byte
        BPL     LAB_F4B2                ; branch if device present

        PLA                             ; else dump calling address low byte
        PLA                             ; dump calling address high byte
        JMP     LAB_F78A                ; do device not present error and return

LAB_F4B2
        LDA     LAB_B7          ; get file name length
        BEQ     LAB_F4C2                ; branch if null name

        LDY     #$00                    ; clear index
LAB_F4B8
        LDA     (LAB_BB),Y              ; get file name byte
        JSR     LAB_EEE4                ; output a byte to the serial bus
        INY                             ; increment index
        CPY     LAB_B7          ; compare with file name length
        BNE     LAB_F4B8                ; loop if not all done

LAB_F4C2
        JSR     LAB_EF04                ; command the serial bus to UNLISTEN
LAB_F4C5
        CLC                             ; flag ok
        RTS


;******************************************************************************;
.
```

```
; open RS232

LAB_F4C7
        LDA     #$06                    ; IIII IOOI, DTR and RTS only as outputs
        STA     LAB_9112                ; set VIA 1 DDRB
        STA     LAB_9110                ; set VIA 1 DRB, DTR and RTS high
        LDA     #$EE                    ; CB2 high, CB1 -ve edge, CA2 high, CA1 -ve edge
        STA     LAB_911C                ; set VIA 1 PCR
        LDY     #$00                    ; clear index
        STY     LAB_0297                ; clear RS232 status byte
LAB_F4D9
        CPY     LAB_B7          ; compare with file name length
        BEQ     LAB_F4E7                ; exit loop if done

        LDA     (LAB_BB),Y              ; get file name byte
        STA     LAB_0293,Y              ; copy to 6551 register set
        INY                             ; increment index
        CPY     #$04                    ; compare with $04
        BNE     LAB_F4D9                ; loop if not to 4 yet

LAB_F4E7
        JSR     LAB_F027                ; compute bit count
        STX     LAB_0298                ; save bit count
        LDA     LAB_0293                ; get pseudo 6551 control register
        AND     #$0F                    ; mask 0000 xxxx, baud rate
        BNE     LAB_F4F4                ; branch nowhere. perhaps there was going to be some
                                        ; error trapping for unimplemented baud rates but
                                        ; this was ever done
LAB_F4F4
        ASL                             ; * 2
        TAX                             ; copy to index
        LDA     LAB_FF5C-2,X    ; get timer constant low byte
        ASL                             ; * 2
        TAY                             ; copy to Y
        LDA     LAB_FF5C-1,X    ; get timer constant high byte
        ROL                             ; * 2
        PHA                             ; save it
        TYA                             ; get timer constant low byte back
        ADC     #$C8                    ; + $C8, carry cleared by previous ROL
        STA     LAB_0299                ; save bit cell time low byte
        PLA                             ; restore high  byte
        ADC     #$00                    ; add carry
        STA     LAB_029A                ; save bit cell time high byte
        LDA     LAB_0294                ; get pseudo 6551 command register
        LSR                             ; shift b0 into Cb
        BCC     LAB_F51B                ; branch if 3 line interface

        LDA     LAB_9120                ; get VIA 2 DRB, this is wrong, the adress should be
                                        ; LAB_9110 which is VIA 1 which is where the DSR input
                                        ; really is
        ASL                             ; shift DSR into Cb
        BCS     LAB_F51B                ; branch if DSR = 1

        JMP     LAB_F016                ; set DSR signal not present and return

LAB_F51B
        LDA     LAB_029B                ; get index to Rx buffer end
        STA     LAB_029C                ; set index to Rx buffer start, clear Rx buffer
        LDA     LAB_029E                ; get index to Tx buffer end
        STA     LAB_029D                ; set index to Tx buffer start, clear Tx buffer
        JSR     LAB_FF75                ; read the top of memory
```

```
        LDA     LAB_F8          ; get Rx buffer pointer high byte
        BNE     LAB_F533                ; branch if buffer already set

        DEY                             ; decrement top of memory high byte, 256 byte buffer
        STY     LAB_F8          ; set Rx buffer pointer high byte
        STX     LAB_F7          ; set Rx buffer pointer low byte
LAB_F533
        LDA     LAB_FA          ; get Tx buffer pointer high byte
        BNE     LAB_F53C                ; branch if buffer already set

        DEY                             ; decrement Rx buffer pointer high byte, 256 byte
buffer
        STY     LAB_FA          ; set Tx buffer pointer high byte
        STX     LAB_F9          ; set Tx buffer pointer low byte
LAB_F53C
        SEC                             ;.
        LDA     #$F0                    ;.
        JMP     LAB_FE7B                ; set the top of memory and return


;*******************************************************************************;
;
; load RAM from a device

; this routine will load data bytes from any input device directly into the memory
; of the computer. It can also be used for a verify operation comparing data from a
; device with the data already in memory, leaving the data stored in RAM unchanged.

; The accumulator must be set to 0 for a load operation or 1 for a verify. If the
; input device was OPENed with a secondary address of 0 the header information from
; device will be ignored. In this case XY must contain the starting address for the
; load. If the device was addressed with a secondary address of 1 or 2 the data will
; load into memory starting at the location specified by the header. This routine
; returns the address of the highest RAM location which was loaded.

; Before this routine can be called, the SETLFS, LAB_FFBA, and SETNAM, LAB_FFBD,
; routines must be called.

LAB_F542
        STX     LAB_C3          ; set kernal setup pointer low byte
        STY     LAB_C4          ; set kernal setup pointer high byte
        JMP     (LAB_0330)              ; do LOAD vector, usually points to LAB_F549


;*******************************************************************************;
;
; load

LAB_F549
        STA     LAB_93          ; save load/verify flag
        LDA     #$00                    ; clear A
        STA     LAB_90          ; clear serial status byte
        LDA     LAB_BA          ; get device number
        BNE     LAB_F556                ; branch if not keyboard

                                        ; can't load form keyboard so ..
LAB_F553
        JMP     LAB_F796                ; do illegal device number and return

LAB_F556
        CMP     #$03                    ; compare device number with screen
```

```
        BEQ     LAB_F553                ; if screen go do illegal device number and return

        BCC     LAB_F5CA                ; branch if less than screen

                                        ; else is serial bus device
        LDY     LAB_B7          ; get file name length
        BNE     LAB_F563                ; branch if not null name

        JMP     LAB_F793                ; else do missing file name error and return

LAB_F563
        JSR     LAB_E4BC                ; get seconday address and print "Searching..."
        LDA     #$60                    ;.
        STA     LAB_B9          ; save the secondary address
        JSR     LAB_F495                ; send secondary address and filename
        LDA     LAB_BA          ; get device number
        JSR     LAB_EE14                ; command a serial bus device to TALK
        LDA     LAB_B9          ; get secondary address
        JSR     LAB_EECE                ; send secondary address after TALK
        JSR     LAB_EF19                ; input a byte from the serial bus
        STA     LAB_AE          ; save program start address low byte
        LDA     LAB_90          ; get serial status byte
        LSR                             ; shift time out read ..
        LSR                             ; .. into carry bit
        BCS     LAB_F5C7                ; if timed out go do file not found error and return

        JSR     LAB_EF19                ; input a byte from the serial bus
        STA     LAB_AF          ; save program start address high byte
        JSR     LAB_E4C1                ; set LOAD address if secondary address = 0
LAB_F58A
        LDA     #$FD                    ; mask xxxx xx0x, clear time out read bit
        AND     LAB_90          ; mask serial status byte
        STA     LAB_90          ; set serial status byte
        JSR     LAB_FFE1                ; scan stop key, return Zb = 1 = [STOP]
        BNE     LAB_F598                ; branch if not [STOP]

        JMP     LAB_F6CB                ; else close the serial bus device and flag stop

LAB_F598
        JSR     LAB_EF19                ; input a byte from the serial bus
        TAX                             ; copy byte
        LDA     LAB_90          ; get serial status byte
        LSR                             ; shift time out read ..
        LSR                             ; .. into carry bit
        BCS     LAB_F58A                ; if timed out go ??

        TXA                             ; copy received byte back
        LDY     LAB_93          ; get load/verify flag
        BEQ     LAB_F5B3                ; branch if load

                                        ; else is verify
        LDY     #$00                    ; clear index
        CMP     (LAB_AE),Y              ; compare byte with previously loaded byte
        BEQ     LAB_F5B5                ; branch if match

        LDA     #$10                    ; flag read error
        JSR     LAB_FE6A                ; OR into serial status byte
        .byte   $2C                     ; makes next line BIT LAB_AE91
LAB_F5B3
        STA     (LAB_AE),Y              ; save byte to memory
LAB_F5B5
```

```
        INC     LAB_AE          ; increment save pointer low byte
        BNE     LAB_F5BB                ; if no rollover skip the high byte increment

        INC     LAB_AF          ; else increment save pointer high byte
LAB_F5BB
        BIT     LAB_90          ; test serial status byte
        BVC     LAB_F58A                ; loop if not end of file

        JSR     LAB_EEF6                ; command the serial bus to UNTALK
        JSR     LAB_F6DA                ; close serial bus device
        BCC     LAB_F641                ; if ?? go flag ok and exit

LAB_F5C7
        JMP     LAB_F787                ; do file not found error and return

LAB_F5CA
        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_F5D1                ; if not RS232 device continue

        JMP     LAB_F0B9                ; else do illegal device number and return

LAB_F5D1
        JSR     LAB_F84D                ; get tape buffer start pointer in XY
        BCS     LAB_F5D9                ; branch if >= $0200

        JMP     LAB_F796                ; do illegal device number and return

LAB_F5D9
        JSR     LAB_F894                ; wait for PLAY
        BCS     LAB_F646                ; exit if STOP was pressed

        JSR     LAB_F647                ; print "Searching..."
LAB_F5E1
        LDA     LAB_B7          ; get file name length
        BEQ     LAB_F5EE
        JSR     LAB_F867                ; find specific tape header
        BCC     LAB_F5F5                ; if no error continue

        BEQ     LAB_F646                ; exit if ??

        BCS     LAB_F5C7                ;., branch always

LAB_F5EE
        JSR     LAB_F7AF                ; find tape header, exit with header in buffer
        BEQ     LAB_F646                ; exit if ??

        BCS     LAB_F5C7                ;.

LAB_F5F5
        LDA     LAB_90          ; get serial status byte
        AND     #$10                    ; mask 000x 0000, read error
        SEC                             ; flag fail
        BNE     LAB_F646                ; if read error just exit

        CPX     #$01                    ;.
        BEQ     LAB_F611                ;.

        CPX     #$03                    ;.
        BNE     LAB_F5E1                ;.

LAB_F604
```

```
        LDY     #$01                    ;.
        LDA     (LAB_B2),Y              ;.
        STA     LAB_C3          ;.
        INY                             ;.
        LDA     (LAB_B2),Y              ;.
        STA     LAB_C4          ;.
        BCS     LAB_F615                ;.

LAB_F611
        LDA     LAB_B9          ; get secondary address
        BNE     LAB_F604                ;.

LAB_F615
        LDY     #$03                    ;.
        LDA     (LAB_B2),Y              ;.
        LDY     #$01                    ;.
        SBC     (LAB_B2),Y              ;.
        TAX                             ;.
        LDY     #$04                    ;.
        LDA     (LAB_B2),Y              ;.
        LDY     #$02                    ;.
        SBC     (LAB_B2),Y              ;.
        TAY                             ;.
        CLC                             ;.
        TXA                             ;.
        ADC     LAB_C3          ;.
        STA     LAB_AE          ;.
        TYA                             ;.
        ADC     LAB_C4          ;.
        STA     LAB_AF          ;.
        LDA     LAB_C3          ;.
        STA     LAB_C1          ; set I/O start addresses low byte
        LDA     LAB_C4          ;.
        STA     LAB_C2          ; set I/O start addresses high byte
        JSR     LAB_F66A                ; display "LOADING" or "VERIFYING"
        JSR     LAB_F8C9                ; do the tape read
        .byte   $24                     ; makes next line BIT LAB_18, keep the error flag in
Cb
LAB_F641
        CLC                             ; flag ok
        LDX     LAB_AE          ; get the LOAD end pointer low byte
        LDY     LAB_AF          ; get the LOAD end pointer high byte
LAB_F646
        RTS


;*******************************************************************************;
;
; print "searching"

LAB_F647
        LDA     LAB_9D          ; get message mode flag
        BPL     LAB_F669                ; exit if control messages off

        LDY     #LAB_F180-LAB_F174
                                        ; index to "SEARCHING "
        JSR     LAB_F1E6                ; display kernel I/O message
        LDA     LAB_B7          ; get file name length
        BEQ     LAB_F669                ; exit if null name

        LDY     #LAB_F18B-LAB_F174
```

```
                                        ; else index to "FOR "
        JSR     LAB_F1E6                ; display kernel I/O message

; print file name

LAB_F659
        LDY     LAB_B7          ; get file name length
        BEQ     LAB_F669                ; exit if null file name

        LDY     #$00                    ; clear index
LAB_F65F
        LDA     (LAB_BB),Y              ; get file name byte
        JSR     LAB_FFD2                ; output character to channel
        INY                             ; increment index
        CPY     LAB_B7          ; compare with file name length
        BNE     LAB_F65F                ; loop if more to do

LAB_F669
        RTS

; display "LOADING" or "VERIFYING"

LAB_F66A
        LDY     #LAB_F1BD-LAB_F174
                                        ; point to "LOADING"
        LDA     LAB_93          ; get load/verify flag
        BEQ     LAB_F672                ; branch if load

        LDY     #LAB_F1CD-LAB_F174
                                        ; point to "VERIFYING"
LAB_F672
        JMP     LAB_F1E2                ; display kernel I/O message if in direct mode and
return


;******************************************************************************;
;
; save RAM to device, A = index to start address, XY = end address low/high

; this routine saves a section of memory. Memory is saved from an indirect address
; on page 0 specified by A, to the address stored in XY, to a logical file. The
; SETLFS, LAB_FFBA, and SETNAM, LAB_FFBD, routines must be used before calling this
; routine. However, a file name is not required to SAVE to device 1, the cassette.
; Any attempt to save to other devices without using a file name results in an error.

; NOTE: device 0, the keyboard, and device 3, the screen, cannot be SAVEd to. If
; the attempt is made, an error will occur, and the SAVE stopped.

LAB_F675
        STX     LAB_AE          ; save end address low byte
        STY     LAB_AF          ; save end address high byte
        TAX                             ; copy index to start pointer
        LDA     LAB_00,X                ; get start address low byte
        STA     LAB_C1          ; set I/O start addresses low byte
        LDA     LAB_01,X                ; get start address high byte
        STA     LAB_C2          ; set I/O start addresses high byte
        JMP     (LAB_0332)              ; go save, usually points to LAB_F685


;******************************************************************************;
```

```
; save

LAB_F685
        LDA     LAB_BA          ; get device number
        BNE     LAB_F68C                ; branch if not keyboard

                                        ; else ..
LAB_F689
        JMP     LAB_F796                ; do illegal device number and return

LAB_F68C
        CMP     #$03                    ; compare device number with screen
        BEQ     LAB_F689                ; if screen do illegal device number and return

        BCC     LAB_F6F1                ; branch if < screen

                                        ; is greater than screen so is serial bus
        LDA     #$61                    ; set secondary address to $01
                                        ; when a secondary address is to be sent to a device
on
                                        ; the serial bus the address must first be ORed with
$60
        STA     LAB_B9          ; save secondary address
        LDY     LAB_B7          ; get file name length
        BNE     LAB_F69D                ; branch if filename not null

        JMP     LAB_F793                ; else do missing file name error and return

LAB_F69D
        JSR     LAB_F495                ; send secondary address and filename
        JSR     LAB_F728                ; print saving [file name]
        LDA     LAB_BA          ; get device number
        JSR     LAB_EE17                ; command devices on the serial bus to LISTEN
        LDA     LAB_B9          ; get secondary address
        JSR     LAB_EEC0                ; send secondary address after LISTEN
        LDY     #$00                    ; clear index
        JSR     LAB_FBD2                ; copy I/O start address to buffer address
        LDA     LAB_AC          ; get buffer address low byte
        JSR     LAB_EEE4                ; output a byte to the serial bus
        LDA     LAB_AD          ; get buffer address high byte
        JSR     LAB_EEE4                ; output a byte to the serial bus
LAB_F6BC
        JSR     LAB_FD11                ; check read/write pointer, return Cb = 1 if pointer
>= end
        BCS     LAB_F6D7                ; go do UNLISTEN if at end

        LDA     (LAB_AC),Y              ; get byte from buffer
        JSR     LAB_EEE4                ; output a byte to the serial bus
        JSR     LAB_FFE1                ; scan stop key
        BNE     LAB_F6D2                ; if stop not pressed go increment pointer and loop
for next

                                        ; else ..

; close the serial bus device and flag stop

LAB_F6CB
        JSR     LAB_F6DA                ; close serial bus device
        LDA     #$00                    ;.
        SEC                             ; flag stop
        RTS
```

```
LAB_F6D2
        JSR     LAB_FD1B                ; increment read/write pointer
        BNE     LAB_F6BC                ; loop, branch always


;******************************************************************************;
;
; ??

LAB_F6D7
        JSR     LAB_EF04                ; command the serial bus to UNLISTEN

; close the serial bus device

LAB_F6DA
        BIT     LAB_B9          ; test the secondary address
        BMI     LAB_F6EF                ; if already closed just exit

        LDA     LAB_BA          ; get the device number
        JSR     LAB_EE17                ; command devices on the serial bus to LISTEN
        LDA     LAB_B9          ; get secondary address
        AND     #$EF                    ; mask the channel number
        ORA     #$E0                    ; OR with the CLOSE command
        JSR     LAB_EEC0                ; send secondary address after LISTEN
        JSR     LAB_EF04                ; command the serial bus to UNLISTEN
LAB_F6EF
        CLC                             ; flag ok
        RTS

LAB_F6F1
        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_F6F8                ; branch if not RS232 device

        JMP     LAB_F0B9                ; else do illegal device number and return

LAB_F6F8
        JSR     LAB_F84D                ; get tape buffer start pointer in XY
        BCC     LAB_F689                ; if < $0200 do illegal device number and return

        JSR     LAB_F8B7                ; wait for PLAY/RECORD
        BCS     LAB_F727                ; exit if STOP was pressed

        JSR     LAB_F728                ; print saving [file name]
        LDX     #$03                    ; set header for a non relocatable program file
        LDA     LAB_B9          ; get secondary address
        AND     #$01                    ; mask non relocatable bit
        BNE     LAB_F70F                ; branch if non relocatable program

        LDX     #$01                    ; else set header for a relocatable program file
LAB_F70F
        TXA                             ; copy header type to A
        JSR     LAB_F7E7                ; write tape header
        BCS     LAB_F727                ; exit if error

        JSR     LAB_F8E6                ; do tape write, 20 cycle count
        BCS     LAB_F727                ; exit if error

        LDA     LAB_B9          ; get secondary address
        AND     #$02                    ; mask end of tape flag
        BEQ     LAB_F726                ; branch if not end of tape
```

```
        LDA     #$05                    ; else set logical end of the tape
        JSR     LAB_F7E7                ; write tape header
        .byte   $24                     ; makes next line BIT LAB_18 so Cb is not changed
LAB_F726
        CLC                             ; flag ok
LAB_F727
        RTS


;******************************************************************************;
;
; print saving [file name]

LAB_F728
        LDA     LAB_9D          ; get message mode flag
        BPL     LAB_F727                ; exit if control messages off

        LDY     #LAB_F1C5-LAB_F174
                                        ; index to "SAVING "
        JSR     LAB_F1E6                ; display kernel I/O message
        JMP     LAB_F659                ; print file name and return


;******************************************************************************;
;
; increment real time clock

; this routine updates the system clock. Normally this routine is called by the
; normal KERNAL interrupt routine every 1/60th of a second. If the user program
; processes its own interrupts this routine must be called to update the time. Also,
; the STOP key routine must be called if the stop key is to remain functional.

LAB_F734
        LDX     #$00                    ; clear X
        INC     LAB_A2          ; increment jiffy low byte
        BNE     LAB_F740                ; if no rollover skip the mid byte increment

        INC     LAB_A1          ; increment jiffy mid byte
        BNE     LAB_F740                ; if no rollover skip the high byte increment

        INC     LAB_A0          ; increment jiffy high byte

                                        ; now subtract a days worth of jiffies from current
count
                                        ; and remember only the Cb result
LAB_F740
        SEC                             ; set carry for subtract
        LDA     LAB_A2          ; get jiffy clock low byte
        SBC     #$01                    ; subtract $4F1A01 low byte
        LDA     LAB_A1          ; get jiffy clock mid byte
        SBC     #$1A                    ; subtract $4F1A01 mid byte
        LDA     LAB_A0          ; get jiffy clock high byte
        SBC     #$4F                    ; subtract $4F1A01 high byte
        BCC     LAB_F755                ; branch if less than $4F1A01 jiffies

                                        ; else ..
        STX     LAB_A0          ; clear jiffies high byte
        STX     LAB_A1          ; clear jiffies mid byte
        STX     LAB_A2          ; clear jiffies low byte
                                        ; this is wrong, there are $4F1A00 jiffies in a day so
```

```
                                        ; the reset to zero should occur when the value
reaches
                                        ; $4F1A00 and not $4F1A01. this would give an extra
jiffy
                                        ; every day and a possible TI value of 24:00:00
LAB_F755
        LDA     LAB_912F                ; get VIA 2 DRA, keyboard row, no handshake
        CMP     LAB_912F                ; compare with self
        BNE     LAB_F755                ; loop if changing

        STA     LAB_91          ; save VIA 2 DRA, keyboard row
        RTS


;*******************************************************************************;
;
; read the real time clock

; this routine returns the time, in jiffies, in AXY. The accumulator contains the
; most significant byte.

LAB_F760
        SEI                             ; disable interrupts
        LDA     LAB_A2          ; get jiffy clock low byte
        LDX     LAB_A1          ; get jiffy clock mid byte
        LDY     LAB_A0          ; get jiffy clock high byte


;*******************************************************************************;
;
; set the real time clock

; the system clock is maintained by an interrupt routine that updates the clock
; every 1/60th of a second. The clock is three bytes long which gives the capability
; to count from zero up to 5,184,000 jiffies - 24 hours plus one jiffy. At that point
; the clock resets to zero. Before calling this routine to set the clock the new time,
; in jiffies, should be in YXA, the accumulator containing the most significant byte.

LAB_F767
        SEI                             ; disable interrupts
        STA     LAB_A2          ; save jiffy clock low byte
        STX     LAB_A1          ; save jiffy clock mid byte
        STY     LAB_A0          ; save jiffy clock high byte
        CLI                             ; enable interrupts
        RTS


;*******************************************************************************;
;
; scan stop key, return Zb = 1 = [STOP]

; if the STOP key on the keyboard is pressed when this routine is called the Z flag
; will be set. All other flags remain unchanged. If the STOP key is not pressed then
; the accumulator will contain a byte representing the last row of the keyboard scan.

; The user can also check for certain other keys this way.

LAB_F770
        LDA     LAB_91          ; get keyboard row
        CMP     #$FE                    ; compare with r0 down
        BNE     LAB_F77D                ; branch if not just r0
```

```
        PHP                             ; save status
        JSR     LAB_FFCC                ; close input and output channels
        STA     LAB_C6          ; save keyboard buffer length
        PLP                             ; restore status
LAB_F77D
        RTS


;******************************************************************************;
;
; file error messages

LAB_F77E
        LDA     #$01                    ; too many files
        .byte   $2C                     ; makes next line BIT LAB_02A9
LAB_F781
        LDA     #$02                    ; file already open
        .byte   $2C                     ; makes next line BIT LAB_03A9
LAB_F784
        LDA     #$03                    ; file not open
        .byte   $2C                     ; makes next line BIT LAB_04A9
LAB_F787
        LDA     #$04                    ; file not found
        .byte   $2C                     ; makes next line BIT LAB_05A9
LAB_F78A
        LDA     #$05                    ; device not present
        .byte   $2C                     ; makes next line BIT LAB_06A9
LAB_F78D
        LDA     #$06                    ; not input file
        .byte   $2C                     ; makes next line BIT LAB_07A9
LAB_F790
        LDA     #$07                    ; not output file
        .byte   $2C                     ; makes next line BIT LAB_08A9
LAB_F793
        LDA     #$08                    ; missing file name
        .byte   $2C                     ; makes next line BIT LAB_09A9
LAB_F796
        LDA     #$09                    ; illegal device number

        PHA                             ; save error #
        JSR     LAB_FFCC                ; close input and output channels
        LDY     #LAB_F174-LAB_F174
                                        ; index to "I/O ERROR #"
        BIT     LAB_9D          ; test message mode flag
        BVC     LAB_F7AC                ; exit if kernal messages off

        JSR     LAB_F1E6                ; display kernel I/O message
        PLA                             ; restore error #
        PHA                             ; copy error #
        ORA     #'0'                    ; convert to ASCII
        JSR     LAB_FFD2                ; output character to channel
LAB_F7AC
        PLA                             ; pull error number
        SEC                             ; flag error
        RTS


;******************************************************************************;
;
; find tape header, exit with header in buffer
```

```
LAB_F7AF
        LDA     LAB_93          ; get load/verify flag
        PHA                             ; save load/verify flag
        JSR     LAB_F8C0                ; initiate tape read
        PLA                             ; restore load/verify flag
        STA     LAB_93          ; save load/verify flag
        BCS     LAB_F7E6                ; exit if error

        LDY     #$00                    ; clear index
        LDA     (LAB_B2),Y              ; read first byte from tape buffer
        CMP     #$05                    ; compare with logical end of the tape
        BEQ     LAB_F7E6                ; exit if end of the tape

        CMP     #$01                    ; compare with header for a relocatable program file
        BEQ     LAB_F7CE                ; branch if program file header

        CMP     #$03                    ; compare with header for a non relocatable program
file
        BEQ     LAB_F7CE                ; branch if program file header

        CMP     #$04                    ; compare with data file header
        BNE     LAB_F7AF                ; if data file loop to find tape header

                                        ; was program file header
LAB_F7CE
        TAX                             ; copy header type
        BIT     LAB_9D          ; get message mode flag
        BPL     LAB_F7E4                ; exit if control messages off

        LDY     #LAB_F1D7-LAB_F174
                                        ; index to "FOUND "
        JSR     LAB_F1E6                ; display kernel I/O message
        LDY     #$05                    ; index to tape filename
LAB_F7DA
        LDA     (LAB_B2),Y              ; get byte from tape buffer
        JSR     LAB_FFD2                ; output character to channel
        INY                             ; increment index
        CPY     #$15                    ; compare with end+1
        BNE     LAB_F7DA                ; loop if more to do

LAB_F7E4
        CLC                             ; flag no error
        DEY                             ; decrement index
LAB_F7E6
        RTS


;******************************************************************************;
;
; write tape header

LAB_F7E7
        STA     LAB_9E          ; save header type
        JSR     LAB_F84D                ; get tape buffer start pointer in XY
        BCC     LAB_F84C                ; exit if < $0200

        LDA     LAB_C2          ; get I/O start address high byte
        PHA                             ; save it
        LDA     LAB_C1          ; get I/O start address low byte
        PHA                             ; save it
```

```
        LDA     LAB_AF          ; get tape end address high byte
        PHA                             ; save it
        LDA     LAB_AE          ; get tape end address low byte
        PHA                             ; save it

        LDY     #$BF                    ; index to header end
        LDA     #' '                    ; clear byte, [SPACE]
LAB_F7FE
        STA     (LAB_B2),Y              ; clear header byte
        DEY                             ; decrement index
        BNE     LAB_F7FE                ; loop if more to do

        LDA     LAB_9E          ; get header type back
        STA     (LAB_B2),Y              ; write to header
        INY                             ; increment index
        LDA     LAB_C1          ; get I/O start address low byte
        STA     (LAB_B2),Y              ; write to header
        INY                             ; increment index
        LDA     LAB_C2          ; get I/O start address high byte
        STA     (LAB_B2),Y              ; write to header
        INY                             ; increment index
        LDA     LAB_AE          ; get tape end address low byte
        STA     (LAB_B2),Y              ; write to header
        INY                             ; increment index
        LDA     LAB_AF          ; get tape end address high byte
        STA     (LAB_B2),Y              ; write to header
        INY                             ; increment index
        STY     LAB_9F          ; save index
        LDY     #$00                    ; clear Y
        STY     LAB_9E          ; clear name index
LAB_F822
        LDY     LAB_9E          ; get name index
        CPY     LAB_B7          ; compare with file name length
        BEQ     LAB_F834                ; exit loop if all done

        LDA     (LAB_BB),Y              ; get file name byte
        LDY     LAB_9F          ; get buffer index
        STA     (LAB_B2),Y              ; save file name byte to buffer
        INC     LAB_9E          ; increment file name index
        INC     LAB_9F          ; increment tape buffer index
        BNE     LAB_F822                ; loop, branch always

LAB_F834
        JSR     LAB_F854                ; set tape buffer start and end pointers
        LDA     #$69                    ; set write lead cycle count
        STA     LAB_AB          ; save write lead cycle count
        JSR     LAB_F8EA                ; do tape write, no cycle count set
        TAY                             ;.
        PLA                             ; pull tape end address low byte
        STA     LAB_AE          ; restore it
        PLA                             ; pull tape end address high byte
        STA     LAB_AF          ; restore it
        PLA                             ; pull I/O start addresses low byte
        STA     LAB_C1          ; restore it
        PLA                             ; pull I/O start addresses high byte
        STA     LAB_C2          ; restore it
        TYA                             ;.
LAB_F84C
        RTS
```

```
;*******************************************************************************;
;
; get tape buffer start pointer

LAB_F84D
        LDX     LAB_B2          ; get tape buffer start pointer low byte
        LDY     LAB_B3          ; get tape buffer start pointer high byte
        CPY     #$02                    ; compare high byte with $02xx
        RTS


;*******************************************************************************;
;
; set tape buffer start and end pointers

LAB_F854
        JSR     LAB_F84D                ; get tape buffer start pointer in XY
        TXA                             ; copy tape buffer start pointer low byte
        STA     LAB_C1          ; save as I/O address pointer low byte
        CLC                             ; clear carry for add
        ADC     #$C0                    ; add buffer length low byte
        STA     LAB_AE          ; save tape buffer end pointer low byte
        TYA                             ; copy tape buffer start pointer high byte
        STA     LAB_C2          ; save as I/O address pointer high byte
        ADC     #$00                    ; add buffer length high byte
        STA     LAB_AF          ; save tape buffer end pointer high byte
        RTS


;*******************************************************************************;
;
; find specific tape header

LAB_F867
        JSR     LAB_F7AF                ; find tape header, exit with header in buffer
        BCS     LAB_F889                ; just exit if error

        LDY     #$05                    ; index to name
        STY     LAB_9F          ; save as tape buffer index
        LDY     #$00                    ; clear Y
        STY     LAB_9E          ; save as name buffer index
LAB_F874
        CPY     LAB_B7          ; compare with file name length
        BEQ     LAB_F888                ; ok exit if match

        LDA     (LAB_BB),Y              ; get file name byte
        LDY     LAB_9F          ; get index to tape buffer
        CMP     (LAB_B2),Y              ; compare with tape header name byte
        BNE     LAB_F867                ; if no match go get next header

        INC     LAB_9E          ; else increment name buffer index
        INC     LAB_9F          ; increment tape buffer index
        LDY     LAB_9E          ; get name buffer index
        BNE     LAB_F874                ; loop, branch always

LAB_F888
        CLC                             ; flag ok
LAB_F889
        RTS
```

```
;*******************************************************************************
;
; bump tape pointer

LAB_F88A
        JSR     LAB_F84D                ; get tape buffer start pointer in XY
        INC     LAB_A6          ; increment tape buffer index
        LDY     LAB_A6          ; get tape buffer index
        CPY     #$C0                    ; compare with buffer length
        RTS


;*******************************************************************************
;
; wait for PLAY

LAB_F894
        JSR     LAB_F8AB                ; return cassette sense in Zb
        BEQ     LAB_F8B5                ; exit if switch closed

                                        ; cassette switch was open
        LDY     #LAB_F18F-LAB_F174
                                        ; index to "PRESS PLAY ON TAPE"
LAB_F89B
        JSR     LAB_F1E6                ; display kernel I/O message
LAB_F89E
        JSR     LAB_F94B                ; scan stop key and flag abort if pressed
                                        ; note if STOP was pressed the return is to the
                                        ; routine that called this one and not here
        JSR     LAB_F8AB                ; return cassette sense in Zb
        BNE     LAB_F89E                ; loop if cassette switch open

        LDY     #LAB_F1DE-LAB_F174
                                        ; index to "OK"
        JMP     LAB_F1E6                ; display kernel I/O message and return


;*******************************************************************************
;
; return cassette sense in Zb

LAB_F8AB
        LDA     #$40                    ; mask for cassette switch
        BIT     LAB_911F                ; test VIA 1 DRA, no handshake
        BNE     LAB_F8B5                ; branch if cassette sense high

        BIT     LAB_911F                ; test VIA 1 DRA again
LAB_F8B5
        CLC
        RTS


;*******************************************************************************
;
; wait for PLAY/RECORD

LAB_F8B7
        JSR     LAB_F8AB                ; return cassette sense in Zb
        BEQ     LAB_F8B5                ; exit if switch closed

                                        ; cassette switch was open
```

```
        LDY     #LAB_F1A2-LAB_F174
                                        ; index to "PRESS RECORD & PLAY ON TAPE"
        BNE     LAB_F89B                ; display message and wait for switch, branch always


;******************************************************************************;
;
; initiate tape read

LAB_F8C0
        LDA     #$00                    ; clear A
        STA     LAB_90          ; clear serial status byte
        STA     LAB_93          ; clear the load/verify flag
        JSR     LAB_F854                ; set tape buffer start and end pointers
LAB_F8C9
        JSR     LAB_F894                ; wait for PLAY
        BCS     LAB_F8ED                ; exit if STOP was pressed, uses further BCS at target
                                        ; address to reach final target at LAB_F957

        SEI                             ; disable interrupts
        LDA     #$00                    ; clear A
        STA     LAB_AA          ;.
        STA     LAB_B4          ;.
        STA     LAB_B0          ; clear tape timing constant min byte
        STA     LAB_9E          ; clear tape pass 1 error log/char buffer
        STA     LAB_9F          ; clear tape pass 2 error log corrected
        STA     LAB_9C          ; clear byte received flag
        LDA     #$82                    ; enable CA1 interrupt
        LDX     #$0E                    ; set index for tape read vector
        BNE     LAB_F8F4                ; go do tape read/write, branch always


;******************************************************************************;
;
; initiate tape write

LAB_F8E3
        JSR     LAB_F854                ; set tape buffer start and end pointers

; do tape write, 20 cycle count

LAB_F8E6
        LDA     #$14                    ; set write lead cycle count
        STA     LAB_AB          ; save write lead cycle count

; do tape write, no cycle count set

LAB_F8EA
        JSR     LAB_F8B7                ; wait for PLAY/RECORD
LAB_F8ED
        BCS     LAB_F957                ; if STOPped clear save IRQ address and exit

        SEI                             ; disable interrupts
        LDA     #$A0                    ; enable VIA 2 T2 interrupt
        LDX     #$08                    ; set index for tape write tape leader vector


;******************************************************************************;
;
; tape read/write
```

```
LAB_F8F4
        LDY     #$7F                    ; disable all interrupts
        STY     LAB_912E                ; set VIA 2 IER, disable interrupts
        STA     LAB_912E                ; set VIA 2 IER, enable interrupts according to A
        JSR     LAB_F160                ; check RS232 bus idle
        LDA     LAB_0314                ; get IRQ vector low byte
        STA     LAB_029F                ; save IRQ vector low byte
        LDA     LAB_0315                ; get IRQ vector high byte
        STA     LAB_02A0                ; save IRQ vector high byte
        JSR     LAB_FCFB                ; set tape vector
        LDA     #$02                    ; set copies count. the first copy is the load copy,
the
                                        ; second copy is the verify copy
        STA     LAB_BE          ; save copies count
        JSR     LAB_FBDB                ; new tape byte setup
        LDA     LAB_911C                ; get VIA 1 PCR
        AND     #$FD                    ; CA2 low, turn on tape motor
        ORA     #$0C                    ; manual output mode
        STA     LAB_911C                ; set VIA 1 PCR
        STA     LAB_C0          ; set tape motor interlock

                                        ; 326656 cycle delay, allow tape motor speed to
stabilise
        LDX     #$FF                    ; outer loop count
LAB_F923
        LDY     #$FF                    ; inner loop count
LAB_F925
        DEY                             ; decrement inner loop count
        BNE     LAB_F925                ; loop if more to do

        DEX                             ; decrement outer loop count
        BNE     LAB_F923                ; loop if more to do

        STA     LAB_9129                ; set VIA 2 T2C_h
        CLI                             ; enable tape interrupts
LAB_F92F
        LDA     LAB_02A0                ; get saved IRQ high byte
        CMP     LAB_0315                ; compare with the current IRQ high byte
        CLC                             ; flag ok
        BEQ     LAB_F957                ; if tape write done go clear saved IRQ address and
exit

        JSR     LAB_F94B                ; scan stop key and flag abort if pressed
                                        ; note if STOP was pressed the return is to the
                                        ; routine that called this one and not here
        LDA     LAB_912D                ; get VIA 2 IFR
        AND     #$40                    ; mask T1 interrupt
        BEQ     LAB_F92F                ; loop if not T1 interrupt

                                        ; else increment jiffy clock
        LDA     LAB_9114                ; get VIA 1 T1C_l, clear T1 flag
        JSR     LAB_F734                ; increment the real time clock
        JMP     LAB_F92F                ; loop


;******************************************************************************;
;
; scan stop key and flag abort if pressed

LAB_F94B
        JSR     LAB_FFE1                ; scan stop key
```

```
        CLC                             ; flag no stop
        BNE     LAB_F95C                ; exit if no stop

        JSR     LAB_FCCF                ; restore everything for STOP
        SEC                             ; flag stopped
        PLA                             ; dump return address low byte
        PLA                             ; dump return address high byte


;******************************************************************************;
;
; clear saved IRQ address

LAB_F957
        LDA     #$00                    ; clear A
        STA     LAB_02A0                ; clear saved IRQ address high byte
LAB_F95C
        RTS


;******************************************************************************;
;
;## set timing

LAB_F95D
        STX     LAB_B1          ; save tape timing constant max byte
        LDA     LAB_B0          ; get tape timing constant min byte
        ASL                             ; *2
        ASL                             ; *4
        CLC                             ; clear carry for add
        ADC     LAB_B0          ; add tape timing constant min byte *5
        CLC                             ; clear carry for add
        ADC     LAB_B1          ; add tape timing constant max byte
        STA     LAB_B1          ; save tape timing constant max byte
        LDA     #$00                    ;.
        BIT     LAB_B0          ; test tape timing constant min byte
        BMI     LAB_F972                ; branch if b7 set

        ROL                             ; else shift carry into ??
LAB_F972
        ASL     LAB_B1          ; shift tape timing constant max byte
        ROL                             ;.
        ASL     LAB_B1          ; shift tape timing constant max byte
        ROL                             ;.
        TAX                             ;.
LAB_F979
        LDA     LAB_9128                ; get VIA 2 T2C_l
        CMP     #$15                    ;.compare with ??
        BCC     LAB_F979                ; loop if less

        ADC     LAB_B1          ; add tape timing constant max byte
        STA     LAB_9124                ; set VIA 2 T1C_l
        TXA                             ;.
        ADC     LAB_9129                ; add VIA 2 T2C_h
        STA     LAB_9125                ; set VIA 2 T1C_h
        CLI                             ; enable interrupts
        RTS


;******************************************************************************;
.
```

```
;;      On Commodore computers, the streams consist of four kinds of symbols
;;      that denote different kinds of low-to-high-to-low transitions on the
;;      read or write signals of the Commodore cassette interface.
;;
;;      A       A break in the communications, or a pulse with very long cycle
;;              time.
;;
;;      B       A short pulse, whose cycle time typically ranges from 296 to 424
;;              microseconds, depending on the computer model.
;;
;;      C       A medium-length pulse, whose cycle time typically ranges from
;;              440 to 576 microseconds, depending on the computer model.
;;
;;      D       A long pulse, whose cycle time typically ranges from 600 to 744
;;              microseconds, depending on the computer model.
;;
;;   The actual interpretation of the serial data takes a little more work to
;; explain. The typical ROM tape loader (and the turbo loaders) will
;; initialize a timer with a specified value and start it counting down. If
;; either the tape data changes or the timer runs out, an IRQ will occur. The
;; loader will determine which condition caused the IRQ. If the tape data
;; changed before the timer ran out, we have a short pulse, or a "0" bit. If
;; the timer ran out first, we have a long pulse, or a "1" bit. Doing this
;; continuously and we decode the entire file.

; read tape bits, IRQ routine

; read T2C which has been counting down from $FFFF. subtract this from $FFFF

LAB_F98E
        LDX     LAB_9129                ; get VIA 2 T2C_h
        LDY     #$FF                    ;.set $FF
        TYA                             ;.A = $FF
        SBC     LAB_9128                ; subtract VIA 2 T2C_l
        CPX     LAB_9129                ; compare VIA 2 T2C_h with previous
        BNE     LAB_F98E                ; loop if timer low byte rolled over

        STX     LAB_B1          ; save tape timing constant max byte
        TAX                             ;.copy $FF - T2C_l
        STY     LAB_9128                ; set VIA 2 T2C_l to $FF
        STY     LAB_9129                ; set VIA 2 T2C_h to $FF
        TYA                             ;.$FF
        SBC     LAB_B1          ; subtract tape timing constant max byte
                                        ; A = $FF - T2C_h
        STX     LAB_B1          ; save tape timing constant max byte
                                        ; LAB_B1 = $FF - T2C_l
        LSR                             ;.A = $FF - T2C_h >> 1
        ROR     LAB_B1          ; shift tape timing constant max byte
                                        ; LAB_B1 = $FF - T2C_l >> 1
        LSR                             ;.A = $FF - T2C_h >> 1
        ROR     LAB_B1          ; shift tape timing constant max byte
                                        ; LAB_B1 = $FF - T2C_l >> 1
        LDA     LAB_B0          ; get tape timing constant min byte
        CLC                             ; clear carry for add
        ADC     #$3C                    ;.
        BIT     LAB_9121                ; test VIA 2 DRA, keyboard row
        CMP     LAB_B1          ; compare with tape timing constant max byte
                                        ; compare with ($FFFF - T2C) >> 2
        BCS     LAB_FA06                ;.branch if min + $3C >= ($FFFF - T2C) >> 2

                                        ; min + $3C < ($FFFF - T2C) >> 2
```

```
        LDX     LAB_9C          ;.get byte received flag
        BEQ     LAB_F9C3                ;.branch if not byte received

        JMP     LAB_FAAD                ;.store tape character

LAB_F9C3
        LDX     LAB_A3          ;.get EOI flag byte
        BMI     LAB_F9E2                ;.

        LDX     #$00                    ;.
        ADC     #$30                    ;.
        ADC     LAB_B0          ; add tape timing constant min byte
        CMP     LAB_B1          ; compare with tape timing constant max byte
        BCS     LAB_F9ED                ;.

        INX                             ;.
        ADC     #$26                    ;.
        ADC     LAB_B0          ; add tape timing constant min byte
        CMP     LAB_B1          ; compare with tape timing constant max byte
        BCS     LAB_F9F1                ;.

        ADC     #$2C                    ;.
        ADC     LAB_B0          ; add tape timing constant min byte
        CMP     LAB_B1          ; compare with tape timing constant max byte
        BCC     LAB_F9E5                ;.

LAB_F9E2
        JMP     LAB_FA60                ;.

LAB_F9E5
        LDA     LAB_B4          ; get bit count
        BEQ     LAB_FA06                ; branch if zero

        STA     LAB_A8          ; save receiver bit count in
        BNE     LAB_FA06                ; branch always

LAB_F9ED
        INC     LAB_A9          ; increment ?? start bit check flag
        BCS     LAB_F9F3                ;.

LAB_F9F1
        DEC     LAB_A9          ; decrement ?? start bit check flag
LAB_F9F3
        SEC                             ;.
        SBC     #$13                    ;.
        SBC     LAB_B1          ; subtract tape timing constant max byte
        ADC     LAB_92          ; add timing constant for tape
        STA     LAB_92          ; save timing constant for tape
        LDA     LAB_A4          ;.get tape bit cycle phase
        EOR     #$01                    ;.
        STA     LAB_A4          ;.save tape bit cycle phase
        BEQ     LAB_FA25                ;.

        STX     LAB_D7          ;.
LAB_FA06
        LDA     LAB_B4          ; get bit count
        BEQ     LAB_FA22                ; exit if zero

        BIT     LAB_912D                ; test get 2 IFR
        BVC     LAB_FA22                ; exit if no T1 interrupt
```

```
        LDA     #$00                    ;.
        STA     LAB_A4          ; clear tape bit cycle phase
        LDA     LAB_A3          ;.get EOI flag byte
        BPL     LAB_FA47                ;.

        BMI     LAB_F9E2                ;.

LAB_FA19
        LDX     #$A6                    ; set timimg max byte
        JSR     LAB_F95D                ; set timing
        LDA     LAB_9B          ;.
        BNE     LAB_F9E5                ;.
LAB_FA22
        JMP     LAB_FF56                ; restore registers and exit interrupt

LAB_FA25
        LDA     LAB_92          ; get timing constant for tape
        BEQ     LAB_FA30                ;.

        BMI     LAB_FA2E                ;.

        DEC     LAB_B0          ; decrement tape timing constant min byte
        .byte   $2C                     ; makes next line BIT LAB_B0E6
LAB_FA2E
        INC     LAB_B0          ; increment tape timing constant min byte
LAB_FA30
        LDA     #$00                    ;.
        STA     LAB_92          ; clear timing constant for tape
        CPX     LAB_D7          ;.
        BNE     LAB_FA47                ;.

        TXA                             ;.
        BNE     LAB_F9E5                ;.

        LDA     LAB_A9          ; get start bit check flag
        BMI     LAB_FA06                ;.

        CMP     #$10                    ;.
        BCC     LAB_FA06                ;.

        STA     LAB_96          ;.save cassette block synchronization number
        BCS     LAB_FA06                ;.

LAB_FA47
        TXA
        EOR     LAB_9B          ;.
        STA     LAB_9B          ;.
        LDA     LAB_B4          ;.
        BEQ     LAB_FA22                ;.

        DEC     LAB_A3          ;.decrement EOI flag byte
        BMI     LAB_FA19                ;.

        LSR     LAB_D7          ;.
        ROR     LAB_BF          ;.parity count
        LDX     #$DA                    ; set timimg max byte
        JSR     LAB_F95D                ; set timing
        JMP     LAB_FF56                ; restore registers and exit interrupt

LAB_FA60
        LDA     LAB_96          ; get cassette block synchronization number
```

```
        BEQ     LAB_FA68                ;.

        LDA     LAB_B4          ;.
        BEQ     LAB_FA6C                ;.

LAB_FA68
        LDA     LAB_A3          ;.get EOI flag byte
        BPL     LAB_F9F1                ;.

LAB_FA6C
        LSR     LAB_B1          ; shift tape timing constant max byte
        LDA     #$93                    ;.
        SEC                             ;.
        SBC     LAB_B1          ; subtract tape timing constant max byte
        ADC     LAB_B0          ; add tape timing constant min byte
        ASL                             ;.
        TAX                             ; copy timimg high byte
        JSR     LAB_F95D                ; set timing
        INC     LAB_9C          ;.
        LDA     LAB_B4          ;.
        BNE     LAB_FA91                ;.

        LDA     LAB_96          ;.get cassette block synchronization number
        BEQ     LAB_FAAA                ;.

        STA     LAB_A8          ; save receiver bit count in
        LDA     #$00                    ;.
        STA     LAB_96          ;.clear cassette block synchronization number
        LDA     #$C0                    ; enable T1 interrupt
        STA     LAB_912E                ; set VIA 2 IER
        STA     LAB_B4          ;.
LAB_FA91
        LDA     LAB_96          ;.get cassette block synchronization number
        STA     LAB_B5          ;.
        BEQ     LAB_FAA0                ;.

        LDA     #$00                    ;.
        STA     LAB_B4          ;.
        LDA     #$40                    ; disable T1 interrupt
        STA     LAB_912E                ; set VIA 2 IER
LAB_FAA0
        LDA     LAB_BF          ;.parity count
        STA     LAB_BD          ;.save RS232 parity byte
        LDA     LAB_A8          ; get receiver bit count in
        ORA     LAB_A9          ; OR with start bit check flag
        STA     LAB_B6          ;.
LAB_FAAA
        JMP     LAB_FF56                ; restore registers and exit interrupt


;******************************************************************************;
;
;## store character

LAB_FAAD
        JSR     LAB_FBDB                ; new tape byte setup
        STA     LAB_9C          ; clear byte received flag
        LDX     #$DA                    ; set timimg max byte
        JSR     LAB_F95D                ; set timing
        LDA     LAB_BE          ;.get copies count
        BEQ     LAB_FABD                ;.
```

```
        STA     LAB_A7          ; save receiver input bit temporary storage
LAB_FABD
        LDA     #$0F                    ;.
        BIT     LAB_AA          ;.
        BPL     LAB_FADA                ;.

        LDA     LAB_B5          ;.
        BNE     LAB_FAD3                ;.

        LDX     LAB_BE          ;.get copies count
        DEX                             ;.
        BNE     LAB_FAD7                ; if ?? restore registers and exit interrupt

        LDA     #$08                    ; set short block
        JSR     LAB_FE6A                ; OR into serial status byte
        BNE     LAB_FAD7                ; restore registers and exit interrupt, branch always

LAB_FAD3
        LDA     #$00                    ;.
        STA     LAB_AA          ;.
LAB_FAD7
        JMP     LAB_FF56                ; restore registers and exit interrupt

LAB_FADA
        BVS     LAB_FB0D                ;.

        BNE     LAB_FAF6                ;.

        LDA     LAB_B5          ;.
        BNE     LAB_FAD7                ;.

        LDA     LAB_B6          ;.
        BNE     LAB_FAD7                ;.

        LDA     LAB_A7          ; get receiver input bit temporary storage
        LSR                             ;.
        LDA     LAB_BD          ;.get RS232 parity byte
        BMI     LAB_FAF0                ;.

        BCC     LAB_FB07                ;.

        CLC                             ;.
LAB_FAF0
        BCS     LAB_FB07                ;.

        AND     #$0F                    ;.
        STA     LAB_AA          ;.
LAB_FAF6
        DEC     LAB_AA          ;.
        BNE     LAB_FAD7                ;.

        LDA     #$40                    ;.
        STA     LAB_AA          ;.
        JSR     LAB_FBD2                ; copy I/O start address to buffer address
        LDA     #$00                    ;.
        STA     LAB_AB          ;.
        BEQ     LAB_FAD7                ;., branch always

.*******************************************************************************.
```

```
;
;## reset pointer

LAB_FB07
        LDA     #$80                    ;.
        STA     LAB_AA          ;.
        BNE     LAB_FAD7                ; restore registers and exit interrupt, branch always

LAB_FB0D
        LDA     LAB_B5          ;.
        BEQ     LAB_FB1B                ;.

        LDA     #$04                    ;.
        JSR     LAB_FE6A                ; OR into serial status byte
        LDA     #$00                    ;.
        JMP     LAB_FB97                ;.

LAB_FB1B
        JSR     LAB_FD11                ; check read/write pointer, return Cb = 1 if pointer
>= end
        BCC     LAB_FB23                ;.

        JMP     LAB_FB95                ;.

LAB_FB23
        LDX     LAB_A7          ; get receiver input bit temporary storage
        DEX                             ;.
        BEQ     LAB_FB55                ;.

        LDA     LAB_93          ; get load/verify flag
        BEQ     LAB_FB38                ; branch if load

        LDY     #$00                    ; clear index
        LDA     LAB_BD          ;.get RS232 parity byte
        CMP     (LAB_AC),Y              ;.
        BEQ     LAB_FB38                ;.

        LDA     #$01                    ;.
        STA     LAB_B6          ;.
LAB_FB38
        LDA     LAB_B6          ;.
        BEQ     LAB_FB87                ;.

        LDX     #$3D                    ;.
        CPX     LAB_9E          ;.
        BCC     LAB_FB80                ;.

        LDX     LAB_9E          ;.
        LDA     LAB_AD          ;.
        STA     LAB_0100+1,X    ;.
        LDA     LAB_AC          ;.
        STA     LAB_0100,X              ;.
        INX                             ;.
        INX                             ;.
        STX     LAB_9E          ;.
        JMP     LAB_FB87                ;.

LAB_FB55
        LDX     LAB_9F          ;.
        CPX     LAB_9E          ;.
        BEQ     LAB_FB90                ;.
```

```
        LDA     LAB_AC          ;.
        CMP     LAB_0100,X              ;.
        BNE     LAB_FB90                ;.

        LDA     LAB_AD          ;.
        CMP     LAB_0100+1,X    ;.
        BNE     LAB_FB90                ;.

        INC     LAB_9F          ;.
        INC     LAB_9F          ;.
        LDA     LAB_93          ; get load/verify flag
        BEQ     LAB_FB7C                ; branch if load

        LDA     LAB_BD          ;.get RS232 parity byte
        LDY     #$00                    ;.
        CMP     (LAB_AC),Y              ;.
        BEQ     LAB_FB90                ;.

        INY                             ;.
        STY     LAB_B6          ;.
LAB_FB7C
        LDA     LAB_B6          ;.
        BEQ     LAB_FB87                ;.

LAB_FB80
        LDA     #$10                    ;.
        JSR     LAB_FE6A                ; OR into serial status byte
        BNE     LAB_FB90                ;.

LAB_FB87
        LDA     LAB_93          ; get load/verify flag
        BNE     LAB_FB90                ; branch if verify

        TAY                             ;.
        LDA     LAB_BD          ;.get RS232 parity byte
        STA     (LAB_AC),Y              ;.
LAB_FB90
        JSR     LAB_FD1B                ; increment read/write pointer
        BNE     LAB_FBCF                ; restore registers and exit interrupt, branch always

LAB_FB95
        LDA     #$80                    ;.
LAB_FB97
        STA     LAB_AA          ;.
        LDX     LAB_BE          ;.get copies count
        DEX                             ;.
        BMI     LAB_FBA0                ;.

        STX     LAB_BE          ;.save copies count
LAB_FBA0
        DEC     LAB_A7          ; decrement receiver input bit temporary storage
        BEQ     LAB_FBAC                ;.

        LDA     LAB_9E          ;.
        BNE     LAB_FBCF                ; if ?? restore registers and exit interrupt

        STA     LAB_BE          ;.save copies count
        BEQ     LAB_FBCF                ; restore registers and exit interrupt, branch always

LAB_FBAC
```

```
        JSR     LAB_FCCF                ; restore everything for STOP
        JSR     LAB_FBD2                ; copy I/O start address to buffer address
        LDY     #$00                    ; clear index
        STY     LAB_AB          ; clear checksum
LAB_FBB6
        LDA     (LAB_AC),Y              ; get byte from buffer
        EOR     LAB_AB          ; XOR with checksum
        STA     LAB_AB          ; save new checksum
        JSR     LAB_FD1B                ; increment read/write pointer
        JSR     LAB_FD11                ; check read/write pointer, return Cb = 1 if pointer
>= end
        BCC     LAB_FBB6                ; loop if not at end

        LDA     LAB_AB          ; get computed checksum
        EOR     LAB_BD          ; compare with stored checksum ??
        BEQ     LAB_FBCF                ; if checksum ok restore registers and exit interrupt

        LDA     #$20                    ; else set checksum error
        JSR     LAB_FE6A                ; OR into serial status byte
LAB_FBCF
        JMP     LAB_FF56                ; restore registers and exit interrupt


;*********************************************************************************;
;
; copy I/O start address to buffer address

LAB_FBD2
        LDA     LAB_C2          ; get I/O start address high byte
        STA     LAB_AD          ; set buffer address high byte
        LDA     LAB_C1          ; get I/O start address low byte
        STA     LAB_AC          ; set buffer address low byte
        RTS


;*********************************************************************************;
;
; new tape byte setup

LAB_FBDB
        LDA     #$08                    ; eight bits to do
        STA     LAB_A3          ; set bit count
        LDA     #$00                    ; clear A
        STA     LAB_A4          ; clear tape bit cycle phase
        STA     LAB_A8          ; clear start bit first cycle done flag
        STA     LAB_9B          ; clear byte parity
        STA     LAB_A9          ; clear start bit check flag, set no start bit yet
        RTS


;*********************************************************************************;
;
; send lsb from tape write byte to tape

; this routine tests the least significant bit in the tape write byte and sets VIA 2 T2
; depending on the state of the bit. if the bit is a 1 a time of $00B0 cycles is set, if
; the bot is a 0 a time of $0060 cycles is set. note that this routine does not shift the
; bits of the tape write byte but uses a copy of that byte, the byte itself is shifted
; elsewhere

LAB_FBEA
```

```
        LDA     LAB_BD          ; get tape write byte
        LSR                             ; shift lsb into Cb
        LDA     #$60                    ; set time constant low byte for bit = 0
        BCC     LAB_FBF3                ; branch if bit was 0

; set time constant for bit = 1 and toggle tape

LAB_FBF1
        LDA     #$B0                    ; set time constant low byte for bit = 1

; write time constant and toggle tape

LAB_FBF3
        LDX     #$00                    ; set time constant high byte

; write time constant and toggle tape

LAB_FBF5
        STA     LAB_9128                ; set VIA 2 T2C_l
        STX     LAB_9129                ; set VIA 2 T2C_h
        LDA     LAB_9120                ; get VIA 2 DRB, keyboard column
        EOR     #$08                    ; toggle tape out bit
        STA     LAB_9120                ; set VIA 2 DRB
        AND     #$08                    ; mask tape out bit
        RTS


;******************************************************************************;
;
; flag block done and exit interrupt

LAB_FC06
        SEC                             ; set carry flag
        ROR     LAB_AD          ; set buffer address high byte negative, flag all sync,
                                        ; data and checksum bytes written
        BMI     LAB_FC47                ; restore registers and exit interrupt, branch always


;******************************************************************************;
;
; tape write, IRQ routine.

; this is the routine that writes the bits to the tape. it is called each time VIA 2 T2
; times out and checks if the start bit is done, if so checks if the data bits are done,
; if so it checks if the byte is done, if so it checks if the synchronisation bytes are
; done, if so it checks if the data bytes are done, if so it checks if the checksum byte
; is done, if so it checks if both the load and verify copies have been done, if so it
; stops the tape

LAB_FC0B
        LDA     LAB_A8          ; get start bit first cycle done flag
        BNE     LAB_FC21                ; if first cycle done go do rest of byte

; each byte sent starts with two half cycles of $0110 ststem clocks and the whole block
; ends with two more such half cycles

        LDA     #$10                    ; set first start cycle time constant low byte
        LDX     #$01                    ; set first start cycle time constant high byte
        JSR     LAB_FBF5                ; write time constant and toggle tape
        BNE     LAB_FC47                ; if first half cycle go restore registers and exit
                                        ; interrupt
```

```
        INC     LAB_A8          ; set start bit first start cycle done flag
        LDA     LAB_AD          ; get buffer address high byte
        BPL     LAB_FC47                ; if block not complete go restore registers and exit
                                        ; interrupt. the end of a block is indicated by the
tape
                                        ; buffer high byte b7 being set to 1

        JMP     LAB_FC95                ; else do tape routine, block complete exit

; continue tape byte write. the first start cycle, both half cycles of it, is complete
; so the routine drops straight through to here

LAB_FC21
        LDA     LAB_A9          ; get start bit check flag
        BNE     LAB_FC2E                ; if the start bit is complete go send the byte bits

; after the two half cycles of $0110 ststem clocks the start bit is completed with two
; half cycles of $00B0 system clocks. this is the same as the first part of a 1 bit

        JSR     LAB_FBF1                ; set time constant for bit = 1 and toggle tape
        BNE     LAB_FC47                ; if first half cycle go restore registers and exit
                                        ; interrupt

        INC     LAB_A9          ; set start bit check flag
        BNE     LAB_FC47                ; restore registers and exit interrupt, branch always

; continue tape byte write. the start bit, both cycles of it, is complete so the routine
; drops straight through to here. now the cycle pairs for each bit, and the parity bit,
; are sent

LAB_FC2E
        JSR     LAB_FBEA                ; send lsb from tape write byte to tape
        BNE     LAB_FC47                ; if first half cycle go restore registers and exit
                                        ; interrupt

                                        ; else two half cycles have been done
        LDA     LAB_A4          ; get tape bit cycle phase
        EOR     #$01                    ; toggle b0
        STA     LAB_A4          ; save tape bit cycle phase
        BEQ     LAB_FC4A                ; if bit cycle phase complete go setup for next bit

; each bit is written as two full cycles. a 1 is sent as a full cycle of $0160 system
; clocks then a full cycle of $00C0 system clocks. a 0 is sent as a full cycle of $00C0
; system clocks then a full cycle of $0160 system clocks. to do this each bit from the
; write byte is inverted during the second bit cycle phase. as the bit is inverted it
; is also added to the, one bit, parity count for this byte

        LDA     LAB_BD          ; get tape write byte
        EOR     #$01                    ; invert bit being sent
        STA     LAB_BD          ; save tape write byte
        AND     #$01                    ; mask b0
        EOR     LAB_9B          ; EOR with tape write byte parity bit
        STA     LAB_9B          ; save tape write byte parity bit
LAB_FC47
        JMP     LAB_FF56                ; restore registers and exit interrupt

; the bit cycle phase is complete so shift out the just written bit and test for byte
; end

LAB_FC4A
```

```
        LSR     LAB_BD          ; shift bit out of tape write byte
        DEC     LAB_A3          ; decrement tape write bit count
        LDA     LAB_A3          ; get tape write bit count
        BEQ     LAB_FC8C                ; if all the data bits have been written go setup for
                                        ; sending the parity bit next and exit the interrupt

        BPL     LAB_FC47                ; if all the data bits are not yet sent just restore
the
                                        ; registers and exit the interrupt

; do next tape byte

; the byte is complete. the start bit, data bits and parity bit have been written to
; the tape so setup for the next byte

LAB_FC54
        JSR     LAB_FBDB                ; new tape byte setup
        CLI                             ; enable interrupts
        LDA     LAB_A5          ; get cassette synchronization character count
        BEQ     LAB_FC6E                ; if synchronisation characters done go do block data

; at the start of each block sent to tape there are a number of synchronisation bytes
; that count down to the actual data. the commodore tape system saves two copies of all
; the tape data, the first is loaded and is indicated by the synchronisation bytes
; having b7 set, and the second copy is indicated by the synchronisation bytes having b7
; clear. the sequence goes $09, $08, ..... $02, $01, data bytes

        LDX     #$00                    ; clear X
        STX     LAB_D7          ; clear checksum byte
        DEC     LAB_A5          ; decrement cassette synchronization byte count
        LDX     LAB_BE          ; get cassette copies count
        CPX     #$02                    ; compare with load block indicator
        BNE     LAB_FC6A                ; branch if not the load block

        ORA     #$80                    ; this is the load block so make the synchronisation
count
                                        ; go $89, $88, ..... $82, $81
LAB_FC6A
        STA     LAB_BD          ; save the synchronisation byte as the tape write byte
        BNE     LAB_FC47                ; restore registers and exit interrupt, branch always

; the synchronization bytes have been done so now check and do the actual block data

LAB_FC6E
        JSR     LAB_FD11                ; check read/write pointer, return Cb = 1 if pointer
>= end
        BCC     LAB_FC7D                ; if not all done yet go get the byte to send

        BNE     LAB_FC06                ; if pointer > end go flag block done and exit
interrupt

                                        ; else the block is complete, it only remains to write
the
                                        ; checksum byte to the tape so setup for that
        INC     LAB_AD          ; increment buffer pointer high byte, this means the block
                                        ; done branch will always be taken next time without
having
                                        ; to worry about the low byte wrapping to zero
        LDA     LAB_D7          ; get checksum byte
        STA     LAB_BD          ; save checksum as tape write byte
        BCS     LAB_FC47                ; restore registers and exit interrupt, branch always
```

```
; the block isn't finished so get the next byte to write to tape

LAB_FC7D
        LDY     #$00                    ; clear index
        LDA     (LAB_AC),Y              ; get byte from buffer
        STA     LAB_BD          ; save as tape write byte
        EOR     LAB_D7          ; XOR with checksum byte
        STA     LAB_D7          ; save new checksum byte
        JSR     LAB_FD1B                ; increment read/write pointer
        BNE     LAB_FC47                ; restore registers and exit interrupt, branch always

; set parity as next bit and exit interrupt

LAB_FC8C
        LDA     LAB_9B          ; get parity bit
        EOR     #$01                    ; toggle it
        STA     LAB_BD          ; save as tape write byte
LAB_FC92
        JMP     LAB_FF56                ; restore registers and exit interrupt

; tape routine, block complete exit

LAB_FC95
        DEC     LAB_BE          ; decrement copies remaining to read/write
        BNE     LAB_FC9C                ; branch if more to do

        JSR     LAB_FD08                ; else stop cassette motor
LAB_FC9C
        LDA     #$50                    ; set tape write leader count
        STA     LAB_A7          ; save tape write leader count
        LDX     #$08                    ; set index for write tape leader vector
        SEI                             ; disable interrupts
        JSR     LAB_FCFB                ; set tape vector
        BNE     LAB_FC92                ; restore registers and exit interrupt, branch always


;******************************************************************************;
;
; write tape leader IRQ routine

LAB_FCA8
        LDA     #$78                    ; set time constant low byte for bit = leader
        JSR     LAB_FBF3                ; write time constant and toggle tape
        BNE     LAB_FC92                ; if tape bit high restore registers and exit
interrupt

        DEC     LAB_A7          ; decrement cycle count
        BNE     LAB_FC92                ; if not all done restore registers and exit interrupt

        JSR     LAB_FBDB                ; new tape byte setup
        DEC     LAB_AB          ; decrement cassette leader count
        BPL     LAB_FC92                ; if not all done restore registers and exit interrupt

        LDX     #$0A                    ; set index for tape write vector
        JSR     LAB_FCFB                ; set tape vector
        CLI                             ; enable interrupts
        INC     LAB_AB          ; clear cassette leader counter, was $FF
        LDA     LAB_BE          ; get cassette block count
        BEQ     LAB_FCF6                ; if all done restore everything for STOP and exit
interrupt
```

```
        JSR     LAB_FBD2                ; copy I/O start address to buffer address
        LDX     #$09                    ; set nine synchronisation bytes
        STX     LAB_A5          ; save cassette synchronization byte count
        BNE     LAB_FC54                ; go do next tape byte, branch always


;******************************************************************************;
;
; restore everything for STOP

LAB_FCCF
        PHP                             ; save status
        SEI                             ; disable interrupts
        JSR     LAB_FD08                ; stop cassette motor
        LDA     #$7F                    ; disable all interrupts
        STA     LAB_912E                ; set VIA 2 IER
        LDA     #$F7                    ; set keyboard column 3 active
        STA     LAB_9120                ; set VIA 2 DRB, keyboard column
        LDA     #$40                    ; set T1 free run, T2 clock f2,
                                        ; SR disabled, latches disabled
        STA     LAB_912B                ; set VIA 2 ACR
        JSR     LAB_FE39                ; set 60Hz and enable timer
        LDA     LAB_02A0                ; get saved IRQ vector high byte
        BEQ     LAB_FCF4                ; branch if null

        STA     LAB_0315                ; restore IRQ vector high byte
        LDA     LAB_029F                ; get saved IRQ vector low byte
        STA     LAB_0314                ; restore IRQ vector low byte
LAB_FCF4
        PLP                             ; restore status
        RTS


;******************************************************************************;
;
; reset vector

LAB_FCF6
        JSR     LAB_FCCF                ; restore everything for STOP
        BEQ     LAB_FC92                ; restore registers and exit interrupt, branch always


;******************************************************************************;
;
; set tape vector

LAB_FCFB
        LDA     LAB_FDF1-8,X    ; get tape IRQ vector low byte
        STA     LAB_0314                ; set IRQ vector low byte
        LDA     LAB_FDF1-7,X    ; get tape IRQ vector high byte
        STA     LAB_0315                ; set IRQ vector high byte
        RTS


;******************************************************************************;
;
; stop cassette motor

LAB_FD08
        LDA     LAB_911C                ; get VIA 1 PCR
```

```
        ORA     #$0E                    ; set CA2 high, cassette motor off
        STA     LAB_911C                ; set VIA 1 PCR
        RTS


;******************************************************************************;
;
; check read/write pointer
; return Cb = 1 if pointer >= end

LAB_FD11
        SEC                             ; set carry for subtract
        LDA     LAB_AC          ; get buffer address low byte
        SBC     LAB_AE          ; subtract buffer end low byte
        LDA     LAB_AD          ; get buffer address high byte
        SBC     LAB_AF          ; subtract buffer end high byte
        RTS


;******************************************************************************;
;
; increment read/write pointer

LAB_FD1B
        INC     LAB_AC          ; increment buffer address low byte
        BNE     LAB_FD21                ; if no overflow skip the high byte increment

        INC     LAB_AD          ; increment buffer address high byte
LAB_FD21
        RTS


;******************************************************************************;
;
; RESET, hardware reset starts here

LAB_FD22
        LDX     #$FF                    ; set X for stack
        SEI                             ; disable interrupts
        TXS                             ; clear stack
        CLD                             ; clear decimal mode
        JSR     LAB_FD3F                ; scan for autostart ROM at $A000
        BNE     LAB_FD2F                ; if not there continue Vic startup

        JMP     (LAB_A000)              ; call ROM start code

LAB_FD2F
        JSR     LAB_FD8D                ; initialise and test RAM
        JSR     LAB_FD52                ; restore default I/O vectors
        JSR     LAB_FDF9                ; initialize I/O registers
        JSR     LAB_E518                ; initialise hardware
        CLI                             ; enable interrupts
        JMP     (LAB_C000)              ; execute BASIC


;******************************************************************************;
;
; scan for autostart ROM at $A000, returns Zb=1 if ROM found

LAB_FD3F
        LDX     #$05                    ; five characters to test
```

```
LAB_FD41
        LDA     LAB_FD4D-1,X    ; get test character
        CMP     LAB_A004-1,X    ; compare wiith byte in ROM space
        BNE     LAB_FD4C                ; exit if no match

        DEX                             ; decrement index
        BNE     LAB_FD41                ; loop if not all done

LAB_FD4C
        RTS


;*******************************************************************************;
;
; autostart ROM signature

LAB_FD4D
        .byte   "A0",$C3,$C2,$CD        ; A0CBM


;*******************************************************************************;
;
; restore default I/O vectors

; This routine restores the default values of all system vectors used in KERNAL and
; BASIC routines and interrupts. The KERNAL VECTOR routine is used to read and alter
; individual system vectors.

LAB_FD52
        LDX     #<LAB_FD6D              ; pointer to vector table low byte
        LDY     #>LAB_FD6D              ; pointer to vector table high byte
        CLC                             ; flag set vectors


;*******************************************************************************;
;
; set/read vectored I/O from (XY), Cb = 1 to read, Cb = 0 to set

; this routine manages all system vector jump addresses stored in RAM. calling this
; routine with the accumulator carry bit set will store the current contents of the
; RAM vectors in a list pointed to by the X and Y registers.

; When this routine is called with the carry bit clear, the user list pointed to by
; the X and Y registers is transferred to the system RAM vectors.

; NOTE: This routine requires caution in its use. The best way to use it is to first
; read the entire vector contents into the user area, alter the desired vectors, and
; then copy the contents back to the system vectors.

LAB_FD57
        STX     LAB_C3          ; save pointer low byte
        STY     LAB_C4          ; save pointer high byte
        LDY     #$1F                    ; set byte count
LAB_FD5D
        LDA     LAB_0314,Y              ; read vector byte from vectors
        BCS     LAB_FD64                ; if read vectors skip the read from XY

        LDA     (LAB_C3),Y              ; read vector byte from (XY)
LAB_FD64
        STA     (LAB_C3),Y              ; save byte to (XY)
        STA     LAB_0314,Y              ; save byte to vector
```

```
        DEY                             ; decrement index
        BPL     LAB_FD5D                ; loop if more to do

        RTS

;; The above code works but it tries to write to the ROM. while this is usually harmless
;; systems that use flash ROM may suffer. Here is a version that makes the extra write
;; to RAM instead but is otherwise identical in function. ##
;
;; set/read vectored I/O from (XY), Cb = 1 to read, Cb = 0 to set
;
;LAB_FD57
;       STX     LAB_C3          ; save pointer low byte
;       STY     LAB_C4          ; save pointer high byte
;       LDY     #$1F                    ; set byte count
;LAB_FD5D
;       LDA     (LAB_C3),Y              ; read vector byte from (XY)
;       BCC     LAB_FD66                ; if set vectors skip the read from XY
;
;       LDA     LAB_0314,Y              ; else read vector byte from vectors
;       STA     (LAB_C3),Y              ; save byte to (XY)
;LAB_FD66
;       STA     LAB_0314,Y              ; save byte to vector
;       DEY                             ; decrement index
;       BPL     LAB_FD5D                ; loop if more to do
;
;       RTS


;*******************************************************************************;
;
; kernal vectors

LAB_FD6D
        .word   LAB_EABF                ; LAB_0314      IRQ vector
        .word   LAB_FED2                ; LAB_0316      BRK vector
        .word   LAB_FEAD                ; LAB_0318      NMI vector
        .word   LAB_F40A                ; LAB_031A      open a logical file
        .word   LAB_F34A                ; LAB_031C      close a specified logical file
        .word   LAB_F2C7                ; LAB_031E      open channel for input
        .word   LAB_F309                ; LAB_0320      open channel for output
        .word   LAB_F3F3                ; LAB_0322      close input and output channels
        .word   LAB_F20E                ; LAB_0324      input character from channel
        .word   LAB_F27A                ; LAB_0326      output character to channel
        .word   LAB_F770                ; LAB_0328      scan stop key
        .word   LAB_F1F5                ; LAB_032A      get character from keyboard queue
        .word   LAB_F3EF                ; LAB_032C      close all channels and files
        .word   LAB_FED2                ; LAB_032E      user function

; Vector to user defined command, currently points to BRK.

; This appears to be a holdover from PET days, when the built-in machine language monitor
; would jump through the LAB_032E vector when it encountered a command that it did not
; understand, allowing the user to add new commands to the monitor.

; Although this vector is initialized to point to the routine called by STOP/RESTORE and
; the BRK interrupt, and is updated by the kernal vector routine at $FD57, it no longer
; has any function.

        .word   LAB_F549                ; LAB_0330      load
        .word   LAB_F685                ; LAB_0332      save
```

```
;*******************************************************************************;
;
; initialise and test RAM, the RAM from $000 to $03FF is never tested and is just assumed
; to work. first a search is done from $0401 for the start of memory and this is saved, if
; this start is at or beyond $1100 then the routine dead ends. once the start of memory is
; found the routine looks for the end of memory, if this end is before $2000 the routine
; again dead ends. lastly, if the end of memory is at $2000 then the screen is set to
; $1E00, but if the memory extends to or beyond $2100 then the screen is moved to $1000

LAB_FD8D
        LDA     #$00                    ; clear A
        TAX                             ; clear index
LAB_FD90
        STA     LAB_00,X                ; clear page 0
        STA     LAB_0200,X              ; clear page 2
        STA     LAB_0300,X              ; clear page 3
        INX                             ; increment index
        BNE     LAB_FD90                ; loop if more to do

        LDX     #<LAB_033C              ; set cassette buffer pointer low byte
        LDY     #>LAB_033C              ; set cassette buffer pointer high byte
        STX     LAB_B2          ; save tape buffer start pointer low byte
        STY     LAB_B3          ; save tape buffer start pointer high byte

        STA     LAB_C1          ; clear RAM test pointer low byte
        STA     LAB_97          ; clear looking for end flag
        STA     LAB_0281                ; clear OS start of memory low byte

        TAY                             ; clear Y
        LDA     #$04                    ; set RAM test pointer high byte
        STA     LAB_C2          ; save RAM test pointer high byte
LAB_FDAF
        INC     LAB_C1          ; increment RAM test pointer low byte
        BNE     LAB_FDB5                ; if no rollover skip the high byte increment

        INC     LAB_C2          ; increment RAM test pointer high byte
LAB_FDB5
        JSR     LAB_FE91                ; test RAM byte, return Cb=0 if failed
        LDA     LAB_97          ; test looking for end flag
        BEQ     LAB_FDDE                ; branch if not looking for end

                                        ; else now looking for the end of memory
        BCS     LAB_FDAF                ; loop if byte test passed

        LDY     LAB_C2          ; get test address high byte
        LDX     LAB_C1          ; get test address low byte
        CPY     #$20                    ; compare with $2000, RAM should always end at or
after
                                        ; $2000 even with no expansion memory as the built in
RAM
                                        ; ends at $1FFF. therefore the following test should
                                        ; never branch
        BCC     LAB_FDEB                ; if end address < $2000 go do dead end loop

        CPY     #$21                    ; compare with $2100
        BCS     LAB_FDD2                ; branch if >= $2100

                                        ; else memory ended before $2100
        LDY     #$1E                    ; set screen memory page to $1E00
```

```
        STY     LAB_0288                ; save screen memory page
LAB_FDCF
        JMP     LAB_FE7B                ; set the top of memory and return

                                        ; memory ends beyond $2100
LAB_FDD2
        LDA     #$12                    ; set OS start of memory high byte
        STA     LAB_0282                ; save OS start of memory high byte
        LDA     #$10                    ; set screen memory page to $1000
        STA     LAB_0288                ; save screen memory page
        BNE     LAB_FDCF                ; set the top of memory and return, branch always

LAB_FDDE
        BCC     LAB_FDAF                ; loop if byte test failed, not found start yet

                                        ; else found start of RAM
        LDA     LAB_C2          ; get test address high byte
        STA     LAB_0282                ; save OS start of memory high byte
        STA     LAB_97          ; set looking for end flag
        CMP     #$11                    ; compare start with $1100, RAM should always start
before
                                        ; $1100 even with no expansion memory as the built in
RAM
                                        ; starts at $1000. therefore the following test should
                                        ; always branch
        BCC     LAB_FDAF                ; go find end of RAM, branch always

                                        ; if the code drops through here then the RAM has
failed
                                        ; and there is not much else to be done
LAB_FDEB
        JSR     LAB_E5C3                ; initialise Vic chip
        JMP     LAB_FDEB                ; loop forever


;*************************************************************************;
;
; tape IRQ vectors

LAB_FDF1
        .word   LAB_FCA8                ; $08   write tape leader IRQ routine
        .word   LAB_FC0B                ; $0A   tape write IRQ routine
        .word   LAB_EABF                ; $0C   normal IRQ vector
        .word   LAB_F98E                ; $0E   read tape bits IRQ routine


;*************************************************************************;
;
; initialize I/O registers

LAB_FDF9
        LDA     #$7F                    ; disable all interrupts
        STA     LAB_911E                ; on VIA 1 IER ..
        STA     LAB_912E                ; .. and VIA 2 IER

        LDA     #$40                    ; set T1 free run, T2 clock ◆2,
                                        ; SR disabled, latches disabled
        STA     LAB_912B                ; set VIA 2 ACR

        LDA     #$40                    ; set T1 free run, T2 clock ◆2,
```

```
        STA     LAB_911B                ; set VIA 1 ACR

        LDA     #$FE                    ; CB2 high, RS232 Tx
                                        ; CB1 +ve edge,
                                        ; CA2 high, tape motor off
                                        ; CA1 -ve edge
        STA     LAB_911C                ; set VIA 1 PCR

        LDA     #$DE                    ; CB2 low, serial data out high
                                        ; CB1 +ve edge,
                                        ; CA2 high, serial clock out low
                                        ; CA1 -ve edge
        STA     LAB_912C                ; set VIA 2 PCR

        LDX     #$00                    ; all inputs, RS232 interface or parallel user port
        STX     LAB_9112                ; set VIA 1 DDRB

        LDX     #$FF                    ; all outputs, keyboard column
        STX     LAB_9122                ; set VIA 2 DDRB

        LDX     #$00                    ; all inputs, keyboard row
        STX     LAB_9123                ; set VIA 2 DDRA

        LDX     #$80                    ; OIII IIII, ATN out, light pen, joystick, serial data
                                        ; in, serial clk in
        STX     LAB_9113                ; set VIA 1 DDRA

        LDX     #$00                    ; ATN out low, set ATN high
        STX     LAB_911F                ; set VIA 1 DRA, no handshake

        JSR     LAB_EF84                ; set serial clock high
        LDA     #$82                    ; enable CA1 interrupt, [RESTORE] key
        STA     LAB_911E                ; set VIA 1 IER
        JSR     LAB_EF8D                ; set serial clock low


;******************************************************************************;
;
; set 60Hz and enable timer

LAB_FE39
        LDA     #$C0                    ; enable T1 interrupt
        STA     LAB_912E                ; set VIA 2 IER
        LDA     #$26                    ; set timer constant low byte [PAL]
;       LDA     #$89                    ; set timer constant low byte [NTSC]
        STA     LAB_9124                ; set VIA 2 T1C_l
        LDA     #$48                    ; set timer constant high byte [PAL]
;       LDA     #$42                    ; set timer constant high byte [NTSC]
        STA     LAB_9125                ; set VIA 2 T1C_h
        RTS


;******************************************************************************;
;
; set filename

; this routine is used to set up the file name for the OPEN, SAVE, or LOAD routines.
; The accumulator must be loaded with the length of the file and XY with the pointer
; to file name, X being the low byte. The address can be any valid memory address in
; the system where a string of characters for the file name is stored. If no file
; name desired the accumulator must be set to 0, representing a zero file length
```

```
; in that case  XY may be set to any memory address.

LAB_FE49
        STA     LAB_B7          ; set file name length
        STX     LAB_BB          ; set file name pointer low byte
        STY     LAB_BC          ; set file name pointer high byte
        RTS


;*******************************************************************************;
;
; set logical file, first and second addresses

; this routine will set the logical file number, device address, and secondary
; address, command number, for other KERNAL routines.

; the logical file number is used by the system as a key to the file table created
; by the OPEN file routine. Device addresses can range from 0 to 30. The following
; codes are used by the computer to stand for the following CBM devices:

; ADDRESS       DEVICE
; =======       ======
;  0            Keyboard
;  1            Cassette #1
;  2            RS-232C device
;  3            CRT display
;  4            Serial bus printer
;  8            CBM Serial bus disk drive

; device numbers of four or greater automatically refer to devices on the serial
; bus.

; a command to the device is sent as a secondary address on the serial bus after
; the device number is sent during the serial attention handshaking sequence. If
; no secondary address is to be sent Y should be set to $FF.

LAB_FE50
        STA     LAB_B8          ; set logical file
        STX     LAB_BA          ; set device number
        STY     LAB_B9          ; set secondary address or command
        RTS


;*******************************************************************************;
;
; read I/O status word

; this routine returns the current status of the I/O device in the accumulator. The
; routine is usually called after new communication to an I/O device. The routine
; will give information about device status, or errors that have occurred during the
; I/O operation.

LAB_FE57
        LDA     LAB_BA          ; get device number
        CMP     #$02                    ; compare device with RS232 device
        BNE     LAB_FE68                ; branch if not RS232 device

                                        ; get RS232 device status
        LDA     LAB_0297                ; read RS232 status word

        LDA     #$00                    ; clear A
```

```
        STA     LAB_0297                ; clear RS232 status

; the above code is wrong. the RS232 status is in A but A is cleared and that is used
; to clear the RS232 status byte. so whatever the status the result is always $00 and
; the status byte is always cleared. A solution is to use X to clear the status after
; it is read instead of the above like this ..
;
;       LDX     #$00                    ; clear X
;       STX     LAB_0297                ; clear RS232 status ##
        RTS


;*******************************************************************************;
;
; control kernal messages

; this routine controls the printing of error and control messages by the KERNAL.
; Either print error messages or print control messages can be selected by setting
; the accumulator when the routine is called.

; FILE NOT FOUND is an example of an error message. PRESS PLAY ON CASSETTE is an
; example of a control message.

; bits 6 and 7 of this value determine where the message will come from. If bit 7
; is set one of the error messages from the KERNAL will be printed. If bit 6 is set
; a control message will be printed.

LAB_FE66
        STA     LAB_9D          ; set message mode flag
LAB_FE68
        LDA     LAB_90          ; read serial status byte

; OR into serial status byte

LAB_FE6A
        ORA     LAB_90          ; OR with serial status byte
        STA     LAB_90          ; save serial status byte
        RTS


;*******************************************************************************;
;
; set timeout on serial bus

; this routine sets the timeout flag for the serial bus. When the timeout flag is
; set, the computer will wait for a device on the serial port for 64 milliseconds.
; If the device does not respond to the computer's DAV signal within that time the
; computer will recognize an error condition and leave the handshake sequence. When
; this routine is called and the accumulator contains a 0 in bit 7, timeouts are
; enabled. A 1 in bit 7 will disable the timeouts.

; NOTE: The the timeout feature is used to communicate that a disk file is not found
; on an attempt to OPEN a file.

LAB_FE6F
        STA     LAB_0285                ; save serial bus timeout flag
        RTS


;*******************************************************************************;
```

```
; read/set the top of memory, Cb = 1 to read, Cb = 0 to set

; this routine is used to read and set the top of RAM. When this routine is called
; with the carry bit set the pointer to the top of RAM will be loaded into XY. When
; this routine is called with the carry bit clear XY will be saved as the top of
; memory pointer changing the top of memory.

LAB_FE73
        BCC     LAB_FE7B                ; if Cb clear go set the top of memory

; read the top of memory

LAB_FE75
        LDX     LAB_0283                ; get memory top low byte
        LDY     LAB_0284                ; get memory top high byte

; set the top of memory

LAB_FE7B
        STX     LAB_0283                ; set memory top low byte
        STY     LAB_0284                ; set memory top high byte
        RTS


;*******************************************************************************;
;
; read/set the bottom of memory, Cb = 1 to read, Cb = 0 to set

; this routine is used to read and set the bottom of RAM. When this routine is
; called with the carry bit set the pointer to the bottom of RAM will be loaded
; into XY. When this routine is called with the carry bit clear XY will be saved as
; the bottom of memory pointer changing the bottom of memory.

LAB_FE82
        BCC     LAB_FE8A                ; if Cb clear go set the bottom of memory

; read the bottom of memory

        LDX     LAB_0281                ; read OS start of memory low byte
        LDY     LAB_0282                ; read OS start of memory high byte

; set the bottom of memory

LAB_FE8A
        STX     LAB_0281                ; set OS start of memory low byte
        STY     LAB_0282                ; set OS start of memory high byte
        RTS


;*******************************************************************************;
;
; non-destructive test RAM byte, return Cb=0 if failed

LAB_FE91
        LDA     (LAB_C1),Y              ; get existing RAM byte
        TAX                             ; copy to X
        LDA     #$55                    ; set first test byte
        STA     (LAB_C1),Y              ; save to RAM
        CMP     (LAB_C1),Y              ; compare with saved
        BNE     LAB_FEA4                ; branch if fail
```

```
        ROR                             ; make byte $AA, carry is set here
        STA     (LAB_C1),Y              ; save to RAM
        CMP     (LAB_C1),Y              ; compare with saved
        BNE     LAB_FEA4                ; branch if fail
        .byte   $A9                     ; makes next line LDA #$18

LAB_FEA4
        CLC                             ; flag test failed
        TXA                             ; get original byte back
        STA     (LAB_C1),Y              ; restore original byte
        RTS


;*******************************************************************************;
;
; NMI vector

LAB_FEA9
        SEI                             ; disable interrupts
        JMP     (LAB_0318)              ; do NMI vector


;*******************************************************************************;
;
; NMI handler

LAB_FEAD
        PHA                             ; save A
        TXA                             ; copy X
        PHA                             ; save X
        TYA                             ; copy Y
        PHA                             ; save Y
        LDA     LAB_911D                ; get VIA 1 IFR
        BPL     LAB_FEFF                ; if no interrupt restore registers and exit

        AND     LAB_911E                ; AND with VIA 1 IER
        TAX                             ; copy to X
        AND     #$02                    ; mask [RESTORE] key
        BEQ     LAB_FEDE                ; branch if not [RESTORE] key

                                        ; else was [RESTORE] key ..
        JSR     LAB_FD3F                ; scan for autostart ROM at $A000
        BNE     LAB_FEC7                ; branch if no autostart ROM

        JMP     (LAB_A002)              ; else do autostart ROM break entry

LAB_FEC7
        BIT     LAB_9111                ; test VIA 1 DRA
        JSR     LAB_F734                ; increment the real time clock
        JSR     LAB_FFE1                ; scan stop key
        BNE     LAB_FEFF                ; if not [STOP] restore registers and exit interrupt


;*******************************************************************************;
;
; BRK handler

LAB_FED2
        JSR     LAB_FD52                ; restore default I/O vectors
        JSR     LAB_FDF9                ; initialize I/O registers
        JSR     LAB_FF18                ; initialise hardware
```

```
        JMP     (LAB_C002)              ; do BASIC break entry


;*******************************************************************************;
;
; RS232 NMI routine

LAB_FEDE
        LDA     LAB_911E                ; get VIA 1 IER
        ORA     #$80                    ; set enable bit, this bit should be set according to
the
                                        ; Rockwell 6522 datasheet but clear acording to the
MOS
                                        ; datasheet. best to assume it's not in the state
required
                                        ; and set it so
        PHA                             ; save to re-enable interrupts
        LDA     #$7F                    ; disable all interrupts
        STA     LAB_911E                ; set VIA 1 IER
        TXA                             ; get active interrupts back
        AND     #$40                    ; mask T1 interrupt
        BEQ     LAB_FF02                ; branch if not T1 interrupt

                                        ; was VIA 1 T1 interrupt
        LDA     #$CE                    ; CB2 low, CB1 -ve edge, CA2 high, CA1 -ve edge
        ORA     LAB_B5          ; OR RS232 next bit to send, sets CB2 high if set
        STA     LAB_911C                ; set VIA 1 PCR
        LDA     LAB_9114                ; get VIA 1 T1C_l
        PLA                             ; restore interrupt enable byte to rstore previously
                                        ; enabled interrupts
        STA     LAB_911E                ; set VIA 1 IER
        JSR     LAB_EFA3                ; RS232 Tx NMI routine
LAB_FEFF
        JMP     LAB_FF56                ; restore registers and exit interrupt

                                        ; was not VIA 1 T1 interrupt
LAB_FF02
        TXA                             ; get active interrupts back
        AND     #$20                    ; mask T2 interrupt
        BEQ     LAB_FF2C                ; branch if not T2 interrupt

                                        ; was VIA 1 T2 interrupt
        LDA     LAB_9110                ; get VIA 1 DRB
        AND     #$01                    ; mask RS232 data in
        STA     LAB_A7          ; save receiver input bit temp storage
        LDA     LAB_9118                ; get VIA 1 T2C_l
        SBC     #$16                    ;.
        ADC     LAB_0299                ; add baud rate bit time low byte
        STA     LAB_9118                ; set VIA 1 T2C_l
        LDA     LAB_9119                ; get VIA 1 T2C_h
        ADC     LAB_029A                ; add baud rate bit time high byte
        STA     LAB_9119                ; set VIA 1 T2C_h
        PLA                             ; restore interrupt enable byte to restore previously
                                        ; enabled interrupts
        STA     LAB_911E                ; set VIA 1 IER, restore interrupts
        JSR     LAB_F036                ; RS232 Rx
        JMP     LAB_FF56                ; restore registers and exit interrupt

                                        ; was not VIA 1 T2 interrupt
LAB_FF2C
        TXA                             ; get active interrupts back
```

```
        AND     #$10                    ; mask CB1 interrupt, Rx data bit transition
        BEQ     LAB_FF56                ; if no bit restore registers and exit interrupt

        LDA     LAB_0293                ; get pseudo 6551 control register
        AND     #$0F                    ; clear non baud bits
        BNE     LAB_FF38                ; short delay. was this to be a branch to code to
implement
                                        ; the user baud rate ??
LAB_FF38
        ASL                             ; *2, 2 bytes per baud count
        TAX                             ; copy to index
        LDA     LAB_FF5C-2,X    ; get baud count low byte
        STA     LAB_9118                ; set VIA 1 T2C_l
        LDA     LAB_FF5C-1,X    ; get baud count high byte
        STA     LAB_9119                ; set VIA 1 T2C_h
        LDA     LAB_9110                ; read VIA 1 DRB, clear interrupt flag
        PLA                             ; restore interrupt enable byte to rstore previously
                                        ; enabled interrupts
        ORA     #$20                    ; enable T2 interrupt
        AND     #$EF                    ; disable CB1 interrupt
        STA     LAB_911E                ; set VIA 1 IER
        LDX     LAB_0298                ; get number of bits to be sent/received
        STX     LAB_A8          ; save receiver bit count in


;*******************************************************************************;
;
; restore the registers and exit the interrupt
;
; if you write your own interrupt code you should either return from the interrupt
; using code that ends up here ot code that replicates this code.

LAB_FF56
        PLA                             ; pull Y
        TAY                             ; restore Y
        PLA                             ; pull X
        TAX                             ; restore X
        PLA                             ; restore A
        RTI


;*******************************************************************************;
;
; baud rate word is calculated from ..
;
; (system clock / baud rate) / 2 - 100
;
;               system clock
;               ------------
; PAL           1108404 Hz
; NTSC  1022727 Hz

; baud rate tables for PAL Vic 20

LAB_FF5C
        .word   $2AE6                   ;   50   baud
        .word   $1C78                   ;   75   baud
        .word   $1349                   ;  110   baud
        .word   $0FB1                   ;  134.5 baud
        .word   $0E0A                   ;  150   baud
        .word   $06D2                   ;  300   baud
```

```
        .word   $0338                   ;  600   baud
        .word   $016A                   ; 1200   baud
        .word   $00D0                   ; 1800   baud
        .word   $0083                   ; 2400   baud
        .word   $0036                   ; 3600   baud

; baud rate tables for NTSC Vic 20

;       .word   $2792                   ;   50   baud
;       .word   $1A40                   ;   75   baud
;       .word   $11C6                   ;  110   baud
;       .word   $0E74                   ;  134.5 baud
;       .word   $0CEE                   ;  150   baud
;       .word   $0645                   ;  300   baud
;       .word   $02F1                   ;  600   baud
;       .word   $0146                   ; 1200   baud
;       .word   $00B8                   ; 1800   baud
;       .word   $0071                   ; 2400   baud
;       .word   $002A                   ; 3600   baud


;******************************************************************************;
;
; IRQ vector

LAB_FF72
        PHA                             ; save A
        TXA                             ; copy X
        PHA                             ; save X
        TYA                             ; copy Y
        PHA                             ; save Y
        TSX                             ; copy stack pointer
        LDA     LAB_0100+4,X    ; get the stacked status register
        AND     #$10                    ; mask the BRK flag bit
        BEQ     LAB_FF82                ; if not BRK go do the hardware IRQ vector

        JMP     (LAB_0316)              ; else do the BRK vector (iBRK)

LAB_FF82
        JMP     (LAB_0314)              ; do IRQ vector (iIRQ)


;******************************************************************************;
;
; spare bytes, not referenced

;LAB_FF85
        .byte   $FF,$FF,$FF,$FF,$FF


;******************************************************************************;
;
; restore default I/O vectors

; This routine restores the default values of all system vectors used in KERNAL and
; BASIC routines and interrupts. The KERNAL VECTOR routine is used to read and alter
; individual system vectors.


;LAB_FF8A
        JMP     LAB_FD52                ; restore default I/O vectors
```

```
;*******************************************************************************
;
; read/set vectored I/O

; this routine manages all system vector jump addresses stored in RAM. calling this
; routine with the accumulator carry bit set will store the current contents of the
; RAM vectors in a list pointed to by the X and Y registers.

; When this routine is called with the carry bit clear, the user list pointed to by
; the X and Y registers is transferred to the system RAM vectors.

; NOTE: This routine requires caution in its use. The best way to use it is to first
; read the entire vector contents into the user area, alter the desired vectors, and
; then copy the contents back to the system vectors.

;LAB_FF8D
        JMP     LAB_FD57                ; set/read vectored I/O from (XY)


;*******************************************************************************
;
; control kernal messages

; this routine controls the printing of error and control messages by the KERNAL.
; Either print error messages or print control messages can be selected by setting
; the accumulator when the routine is called.

; FILE NOT FOUND is an example of an error message. PRESS PLAY ON CASSETTE is an
; example of a control message.

; bits 6 and 7 of this value determine where the message will come from. If bit 7
; is set one of the error messages from the KERNAL will be printed. If bit 6 is set
; a control message will be printed.

LAB_FF90
        JMP     LAB_FE66                ; control kernal messages


;*******************************************************************************
;
; send secondary address after LISTEN

; this routine is used to send a secondary address to an I/O device after a call to
; the LISTEN routine is made and the device commanded to LISTEN. The routine cannot
; be used to send a secondary address after a call to the TALK routine.

; A secondary address is usually used to give set-up information to a device before
; I/O operations begin.

; When a secondary address is to be sent to a device on the serial bus the address
; must first be ORed with $60.

;LAB_FF93
        JMP     LAB_EEC0                ; send secondary address after LISTEN


;*******************************************************************************
;
; send secondary address after TALK
```

```
; this routine transmits a secondary address on the serial bus for a TALK device.
; This routine must be called with a number between 4 and 31 in the accumulator.
; The routine will send this number as a secondary address command over the serial
; bus. This routine can only be called after a call to the TALK routine. It will
; not work after a LISTEN.

;LAB_FF96
        JMP     LAB_EECE                ; send secondary address after TALK


;*******************************************************************************;
;
; read/set the top of memory

; this routine is used to read and set the top of RAM. When this routine is called
; with the carry bit set the pointer to the top of RAM will be loaded into XY. When
; this routine is called with the carry bit clear XY will be saved as the top of
; memory pointer changing the top of memory.

LAB_FF99
        JMP     LAB_FE73                ; read/set the top of memory


;*******************************************************************************;
;
; read/set the bottom of memory

; this routine is used to read and set the bottom of RAM. When this routine is
; called with the carry bit set the pointer to the bottom of RAM will be loaded
; into XY. When this routine is called with the carry bit clear XY will be saved as
; the bottom of memory pointer changing the bottom of memory.

LAB_FF9C
        JMP     LAB_FE82                ; read/set the bottom of memory


;*******************************************************************************;
;
; scan the keyboard

; this routine will scan the keyboard and check for pressed keys. It is the same
; routine called by the interrupt handler. If a key is down, its ASCII value is
; placed in the keyboard queue.

;LAB_FF9F
        JMP     LAB_EB1E                ; scan keyboard


;*******************************************************************************;
;
; set timeout on serial bus

; this routine sets the timeout flag for the serial bus. When the timeout flag is
; set, the computer will wait for a device on the serial port for 64 milliseconds.
; If the device does not respond to the computer's DAV signal within that time the
; computer will recognize an error condition and leave the handshake sequence. When
; this routine is called and the accumulator contains a 0 in bit 7, timeouts are
; enabled. A 1 in bit 7 will disable the timeouts.

; NOTE: The the timeout feature is used to communicate that a disk file is not found
```

```
; on an attempt to OPEN a file.

;LAB_FFA2
        JMP     LAB_FE6F                ; set timeout on serial bus


;*******************************************************************************
;
; input a byte from the serial bus

; this routine reads a byte of data from the serial bus using full handshaking. the
; data is returned in the accumulator. before using this routine the TALK routine,
; LAB_FFB4, must have been called first to command the device on the serial bus to
; send data on the bus. if the input device needs a secondary command it must be sent
; by using the TKSA routine, LAB_FF96, before calling this routine.

; errors are returned in the status word which can be read by calling the READST
; routine, LAB_FFB7.

;LAB_FFA5
        JMP     LAB_EF19                ; input byte from serial bus


;*******************************************************************************
;
; output a byte to the serial bus

; this routine is used to send information to devices on the serial bus. A call to
; this routine will put a data byte onto the serial bus using full handshaking.
; Before this routine is called the LISTEN routine, LAB_FFB1, must be used to
; command a device on the serial bus to get ready to receive data.

; the accumulator is loaded with a byte to output as data on the serial bus. A
; device must be listening or the status word will return a timeout. This routine
; always buffers one character. So when a call to the UNLISTEN routine, LAB_FFAE,
; is made to end the data transmission, the buffered character is sent with EOI
; set. Then the UNLISTEN command is sent to the device.

;LAB_FFA8
        JMP     LAB_EEE4                ; output a byte to the serial bus


;*******************************************************************************;
;
; command the serial bus to UNTALK

; this routine will transmit an UNTALK command on the serial bus. All devices
; previously set to TALK will stop sending data when this command is received.

;LAB_FFAB
        JMP     LAB_EEF6                ; command the serial bus to UNTALK


;*******************************************************************************;
;
; command the serial bus to UNLISTEN

; this routine commands all devices on the serial bus to stop receiving data from
; the computer. Calling this routine results in an UNLISTEN command being transmitted
; on the serial bus. Only devices previously commanded to listen will be affected.
```

```
; This routine is normally used after the computer is finished sending data to
; external devices. Sending the UNLISTEN will command the listening devices to get
; off the serial bus so it can be used for other purposes.

;LAB_FFAE
        JMP     LAB_EF04                ; command the serial bus to UNLISTEN


;******************************************************************************;
;
; command devices on the serial bus to LISTEN

; this routine will command a device on the serial bus to receive data. The
; accumulator must be loaded with a device number between 4 and 31 before calling
; this routine. LISTEN convert this to a listen address then transmit this data as
; a command on the serial bus. The specified device will then go into listen mode
; and be ready to accept information.

;LAB_FFB1
        JMP     LAB_EE17                ; command devices on the serial bus to LISTEN


;******************************************************************************;
;
; command a serial bus device to TALK

; to use this routine the accumulator must first be loaded with a device number
; between 4 and 30. When called this routine converts this device number to a talk
; address. Then this data is transmitted as a command on the Serial bus.

;LAB_FFB4
        JMP     LAB_EE14                ; command serial bus device to TALK


;******************************************************************************;
;
; read I/O status word

; this routine returns the current status of the I/O device in the accumulator. The
; routine is usually called after new communication to an I/O device. The routine
; will give information about device status, or errors that have occurred during the
; I/O operation.

LAB_FFB7
        JMP     LAB_FE57                ; read I/O status word


;******************************************************************************;
;
; set logical, first and second addresses

; this routine will set the logical file number, device address, and secondary
; address, command number, for other KERNAL routines.

; the logical file number is used by the system as a key to the file table created
; by the OPEN file routine. Device addresses can range from 0 to 30. The following
; codes are used by the computer to stand for the following CBM devices:

; ADDRESS       DEVICE
; =======       ======
; 0             Keyboard
```

```
;  1            Cassette #1
;  2            RS-232C device
;  3            CRT display
;  4            Serial bus printer
;  8            CBM Serial bus disk drive

; device numbers of four or greater automatically refer to devices on the serial
; bus.

; a command to the device is sent as a secondary address on the serial bus after
; the device number is sent during the serial attention handshaking sequence. If
; no secondary address is to be sent Y should be set to $FF.

LAB_FFBA
        JMP     LAB_FE50                ; set logical, first and second addresses


;*******************************************************************************;
;
; set the filename

; this routine is used to set up the file name for the OPEN, SAVE, or LOAD routines.
; The accumulator must be loaded with the length of the file and XY with the pointer
; to file name, X being th low byte. The address can be any valid memory address in
; the system where a string of characters for the file name is stored. If no file
; name desired the accumulator must be set to 0, representing a zero file length,
; in that case  XY may be set to any memory address.

LAB_FFBD
        JMP     LAB_FE49                ; set filename


;*******************************************************************************;
;
; open a logical file

; this routine is used to open a logical file. Once the logical file is set up it
; can be used for input/output operations. Most of the I/O KERNAL routines call on
; this routine to create the logical files to operate on. No arguments need to be
; set up to use this routine, but both the SETLFS, LAB_FFBA, and SETNAM, LAB_FFBD,
; KERNAL routines must be called before using this routine.

LAB_FFC0
        JMP     (LAB_031A)              ; do open file vector


;*******************************************************************************;
;
; close a specified logical file

; this routine is used to close a logical file after all I/O operations have been
; completed on that file. This routine is called after the accumulator is loaded
; with the logical file number to be closed, the same number used when the file was
; opened using the OPEN routine.

LAB_FFC3
        JMP     (LAB_031C)              ; do close file vector


;*******************************************************************************
```

```
; open a channel for input

; any logical file that has already been opened by the OPEN routine, LAB_FFC0, can be
; defined as an input channel by this routine. the device on the channel must be an
; input device or an error will occur and the routine will abort.

; if you are getting data from anywhere other than the keyboard, this routine must be
; called before using either the CHRIN routine, LAB_FFCF, or the GETIN routine,
; LAB_FFE4. if you are getting data from the keyboard and no other input channels are
; open then the calls to this routine and to the OPEN routine, LAB_FFC0, are not needed.

; when used with a device on the serial bus this routine will automatically send the
; listen address specified by the OPEN routine, LAB_FFC0, and any secondary address.

; possible errors are:
;
;       3 : file not open
;       5 : device not present
;       6 : file is not an input file

LAB_FFC6
        JMP     (LAB_031E)              ; do open for input vector


;*******************************************************************************
;
; open a channel for output

; any logical file that has already been opened by the OPEN routine, LAB_FFC0, can be
; defined as an output channel by this routine the device on the channel must be an
; output device or an error will occur and the routine will abort.

; if you are sending data to anywhere other than the screen this routine must be
; called before using the CHROUT routine, LAB_FFD2. if you are sending data to the
; screen and no other output channels are open then the calls to this routine and to
; the OPEN routine, LAB_FFC0, are not needed.

; when used with a device on the serial bus this routine will automatically send the
; listen address specified by the OPEN routine, LAB_FFC0, and any secondary address.

; possible errors are:
;
;       3 : file not open
;       5 : device not present
;       7 : file is not an output file

LAB_FFC9
        JMP     (LAB_0320)              ; do open for output vector


;*******************************************************************************
;
; close input and output channels

; this routine is called to clear all open channels and restore the I/O channels to
; their original default values. It is usually called after opening other I/O
; channels and using them for input/output operations. The default input device is
; 0, the keyboard. The default output device is 3, the screen.

; If one of the channels to be closed is to the serial port, an UNTALK signal is sent
; first to clear the input channel or an UNLISTEN is sent to clear the output channel
```

```
; By not calling this routine and leaving listener(s) active on the serial bus,
; several devices can receive the same data from the VIC at the same time. One way to
; take advantage of this would be to command the printer to TALK and the disk to
; LISTEN. This would allow direct printing of a disk file.

LAB_FFCC
        JMP     (LAB_0322)              ; do close vector


;*******************************************************************************
;
; input character from channel

; this routine will get a byte of data from the channel already set up as the input
; channel by the CHKIN routine, LAB_FFC6.

; If CHKIN, LAB_FFC6, has not been used to define another input channel the data is
; expected to be from the keyboard. the data byte is returned in the accumulator. the
; channel remains open after the call.

; input from the keyboard is handled in a special way. first, the cursor is turned on
; and it will blink until a carriage return is typed on the keyboard. all characters
; on the logical line, up to 88 characters, will be stored in the BASIC input buffer.
; then the characters can be returned one at a time by calling this routine once for
; each character. when the carriage return is returned the entire line has been
; processed. the next time this routine is called the whole process begins again.

LAB_FFCF
        JMP     (LAB_0324)              ; do input vector


;*******************************************************************************
;
; output a character to channel

; this routine will output a character to an already opened channel. Use the OPEN
; routine, LAB_FFC0, and the CHKOUT routine, LAB_FFC9, to set up the output channel
; before calling this routine. If these calls are omitted, data will be sent to the
; default output device, device 3, the screen. The data byte to be output is loaded
; into the accumulator, and this routine is called. The data is then sent to the
; specified output device. The channel is left open after the call.

; NOTE: Care must be taken when using routine to send data to a serial device since
; data will be sent to all open output channels on the bus. Unless this is desired,
; all open output channels on the serial bus other than the actually intended
; destination channel must be closed by a call to the KERNAL close channel routine.

LAB_FFD2
        JMP     (LAB_0326)              ; do output vector


;*******************************************************************************;
;
; load RAM from a device

; this routine will load data bytes from any input device directly into the memory
; of the computer. It can also be used for a verify operation comparing data from a
; device with the data already in memory, leaving the data stored in RAM unchanged.

; The accumulator must be set to 0 for a load operation or 1 for a verify. If the
; input device was OPENed with a secondary address of 0 the header information from
```

```
; device will be ignored. In this case XY must contain the starting address for the
; load. If the device was addressed with a secondary address of 1 or 2 the data will
; load into memory starting at the location specified by the header. This routine
; returns the address of the highest RAM location which was loaded.

; Before this routine can be called, the SETLFS, LAB_FFBA, and SETNAM, LAB_FFBD,
; routines must be called.

LAB_FFD5
        JMP     LAB_F542                ; load RAM from a device


;*******************************************************************************;
;
; save RAM to a device

; this routine saves a section of memory. Memory is saved from an indirect address
; on page 0 specified by A, to the address stored in XY, to a logical file. The
; SETLFS, LAB_FFBA, and SETNAM, LAB_FFBD, routines must be used before calling this
; routine. However, a file name is not required to SAVE to device 1, the cassette.
; Any attempt to save to other devices without using a file name results in an error.

; NOTE: device 0, the keyboard, and device 3, the screen, cannot be SAVEd to. If
; the attempt is made, an error will occur, and the SAVE stopped.

LAB_FFD8
        JMP     LAB_F675                ; save RAM to device


;*******************************************************************************;
;
; set the real time clock

; the system clock is maintained by an interrupt routine that updates the clock
; every 1/60th of a second. The clock is three bytes long which gives the capability
; to count from zero up to 5,184,000 jiffies - 24 hours plus one jiffy. At that point
; the clock resets to zero. Before calling this routine to set the clock the new time,
; in jiffies, should be in YXA, the accumulator containing the most significant byte.

LAB_FFDB
        JMP     LAB_F767                ; set real time clock


;*******************************************************************************;
;
; read the real time clock

; this routine returns the time, in jiffies, in AXY. The accumulator contains the
; most significant byte.

LAB_FFDE
        JMP     LAB_F760                ; read real time clock


;*******************************************************************************;
;
; scan the stop key

; if the STOP key on the keyboard is pressed when this routine is called the Z flag
; will be set. All other flags remain unchanged. If the STOP key is not pressed then
; the accumulator will contain a byte representing the last row of the keyboard scan
```

```
; The user can also check for certain other keys this way.

LAB_FFE1
        JMP     (LAB_0328)              ; do stop key vector


;*******************************************************************************;
;
; get a character from an input device

; in practice this routine operates identically to the CHRIN routine, LAB_FFCF,
; for all devices except for the keyboard. If the keyboard is the current input
; device this routine will get one character from the keyboard buffer. It depends
; on the IRQ routine to read the keyboard and put characters into the buffer.

; If the keyboard buffer is empty the value returned in the accumulator will be zero

LAB_FFE4
        JMP     (LAB_032A)              ; do get vector


;*******************************************************************************;
;
; close all channels and files

; this routine closes all open files. When this routine is called, the pointers into
; the open file table are reset, closing all files. Also the routine automatically
; resets the I/O channels.

LAB_FFE7
        JMP     (LAB_032C)              ; do close all vector


;*******************************************************************************;
;
; increment the real time clock

; this routine updates the system clock. Normally this routine is called by the
; normal KERNAL interrupt routine every 1/60th of a second. If the user program
; processes its own interrupts this routine must be called to update the time. Also,
; the STOP key routine must be called if the stop key is to remain functional.

LAB_FFEA
        JMP     LAB_F734                ; increment real time clock


;*******************************************************************************;
;
; return X,Y organization of screen

; this routine returns the x,y organisation of the screen in X,Y

;LAB_FFED
        JMP     LAB_E505                ; return X,Y organization of screen


;*******************************************************************************;
;
; read/set X,Y cursor position
```

```
; this routine, when called with the carry flag set, loads the current position of
; the cursor on the screen into the X and Y registers. X is the column number of
; the cursor location and Y is the row number of the cursor. A call with the carry
; bit clear moves the cursor to the position determined by the X and Y registers.

LAB_FFF0
        JMP     LAB_E50A                ; read/set X,Y cursor position


;******************************************************************************;
;
; return the base address of the I/O devices

; this routine will set XY to the address of the memory section where the memory
; mapped I/O devices are located. This address can then be used with an offset to
; access the memory mapped I/O devices in the computer.

LAB_FFF3
        JMP     LAB_E500                ; return base address of I/O devices


;******************************************************************************;
;
; spare bytes, not referenced

        .byte   $FF,$FF,$FF,$FF


;******************************************************************************;
;
; hardware vectors

        .word   LAB_FEA9                ; NMI vector
        .word   LAB_FD22                ; RESET vector
        .word   LAB_FF72                ; IRQ vector

        .END


;******************************************************************************;
;******************************************************************************;
```