```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                                                               *
;*      SUPERMON 816 MACHINE LANGUAGE MONITOR FOR THE W65C816S MICROPROCESSOR     *
;* ----------------------------------------------------------------------------- *
;*      Copyright ©1991-2014 by BCS Technology Limited.  All rights reserved.    *
;*                                                                               *
;* Permission is hereby granted to use, copy, modify and distribute this software, *
;* provided this copyright notice remains in the source code and  proper  attribu- *
;* tion is given.  Redistribution, regardless of form, must be at no charge to the *
;* end  user.  This code or any part thereof, including any derivation, MAY NOT be *
;* incorporated into any package intended for sale,  unless written permission has *
;* been given by the copyright holder.                                           *
;*                                                                               *
;* THERE IS NO WARRANTY OF ANY KIND WITH THIS SOFTWARE.  The user assumes all risk *
;* in connection with the incorporation of this software into any system.        *
;* ----------------------------------------------------------------------------- *
;* Supermon 816 is a salute to Jim Butterfield, who passed away on June 29, 2007.  *
;*                                                                               *
;* Jim, who was the unofficial  spokesman for  Commodore  International during the *
;* heyday of the company's 8 bit supremacy, scratch-developed the Supermon machine *
;* language monitor for the PET & CBM computers.   When the best-selling Commodore *
;* 64 was introduced, Jim adapted his software to the new machine & gave the adap- *
;* tation the name Supermon 64.   Commodore  subsequently  integrated a customized *
;* version of Supermon 64 into the C-128 to act as the resident M/L monitor.      *
;*                                                                               *
;* Although Supermon 816 is not an adaptation of Supermon 64,  it was  decided  to *
;* keep the Supermon name alive, since Supermon 816's general operation & user in- *
;* terface is similar to that of Supermon 64.   Supermon 816 is 100 percent native *
;* mode 65C816 code & was developed from a blank canvas.                         *
;* ----------------------------------------------------------------------------- *
;* Supermon 816 is a full featured monitor and supports the following operations:  *
;*                                                                               *
;*     A - Assemble code                                                         *
;*     C - Compare memory regions                                                *
;*     D - Disassemble code                                                      *
;*     F - Fill memory region (cannot span banks)                                *
;*     G - Execute code (stops at BRK)                                           *
;*     H - Search (hunt) memory region                                           *
;*     J - Execute code as a subroutine (stops at BRK or RTS)                    *
;*     M - Dump & display memory range                                           *
;*     R - Dump & display 65C816 registers                                       *
;*     T - Copy (transfer) memory region                                         *
;*     X - Exit Supermon 816 & return to operating environment                   *
;*     > - Modify up to 32 bytes of memory                                       *
;*     ; - Modify 65C816 registers                                               *
;*                                                                               *
;* Supermon 816 accepts binary (%), octal (%), decimal (+) and hexadecimal ($) as  *
;* input for numeric parameters.  Additionally, the H and > operations accept an   *
;* ASCII string in place of numeric values by preceding the string with ', e.g.:   *
;*                                                                               *
```

```
;*     h 042000 042FFF 'BCS Technology Limited                                  *
;*                                                                              *
;* If no radix symbol is entered hex is assumed.                                *
;*                                                                              *
;* Numeric conversion is also available.  For example, typing:                  *
;*                                                                              *
;*     +1234567 <CR>                                                            *
;*                                                                              *
;* will display:                                                                *
;*                                                                              *
;*         $12D687                                                              *
;*         +1234567                                                             *
;*         %04553207                                                            *
;*         %10010110101101000111                                                *
;*                                                                              *
;* In the above example, <CR> means the console keyboard's return or enter key. *
;*                                                                              *
;* All numeric values are internally processed as 32 bit unsigned integers.  Addr-*
;* esses may be entered as 8, 16 or 24 bit values.  During instruction assembly,  *
;* immediate mode operands may be forced to 16 bits by preceding the operand with *
;* an exclamation point if the instruction can accept a 16 bit operand, e.g.:   *
;*                                                                              *
;*     a 1f2000 lda !#4                                                         *
;*                                                                              *
;* The above will assemble as:                                                  *
;*                                                                              *
;*     A 1F2000  A9 04 00     LDA #$0004                                        *
;*                                                                              *
;* Entering:                                                                    *
;*                                                                              *
;*     a 1f2000 ldx !#+157                                                      *
;*                                                                              *
;* will assemble as:                                                            *
;*                                                                              *
;*     A 1F2000  A2 9D 00     LDX #$009D                                        *
;*                                                                              *
;* Absent the ! in the operand field, the above would have been assembled as:   *
;*                                                                              *
;*     A 1F2000  A2 9D        LDX #$9D                                          *
;*                                                                              *
;* If an immediate mode operand is greater than $FF assembly of a 16 bit operand *
;* is implied.                                                                  *
;* ---------------------------------------------------------------------------- *
;* A Note on the PEA & PEI Instructions                                         *
;* ------------------------------------                                         *
;*                                                                              *
;* The Eyes and Lichty programming manual uses the following syntax for the PEA *
;* and PEI instructions:                                                        *
;*                                                                              *
;*     PEA <operand>                                                            *
```

```
;*     PEI (<operand>)                                                          *
;*                                                                             *
;* The WDC data sheet that was published at the time of the 65C816's release in    *
;* 1984 does not indicate a recommended or preferred syntax for any of the above   *
;* instructions.  PEA pushes its operand to the stack and hence operates like any  *
;* other immediate mode instruction, in that the operand is the data (however, PEA *
;* doesn't affect the status register).  Similarly, PEI pushes the 16 bit value    *
;* stored at <operand> and <operand>+1, and hence operates like any other direct   *
;* (zero) page instruction, again without affecting the status register.           *
;*                                                                             *
;* BCS Technology Limited is of the opinion that the developer of the ORCA/M as-   *
;* sembler, which is the assembler referred to in the Eyes and Lichty manual, mis- *
;* understood how PEA and PEI behave during runtime, and hence chose an incorrect  *
;* syntax for these two instructions.  This error was subsequently carried forward *
;* by Eyes and Lichty.                                                         *
;*                                                                             *
;* Supermon 816's assembler uses the following syntax for PEA and PEI:             *
;*                                                                             *
;*     PEA #<operand>                                                          *
;*     PEI <operand>                                                           *
;*                                                                             *
;* The operand for PEA is treated as a 16 bit value, even if entered as an 8 bit   *
;* value.  The operand for PEI must be 8 bits.                                 *
;*                                                                             *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;       * * * * * * * * * * * *
;       * VERSION INFORMATION *
;       * * * * * * * * * * * *
;
        .MACRO  SOFTVERS        ;software version - change with each revision...
        .BYTE   "1"             ;major
        .BYTE   "."
        .BYTE   "0"             ;minor
        .BYTE   "."
        .BYTE   "2"             ;revision
        .ENDMACRO
;
;REVISION TABLE
;
;Ver  Rev Date    Description
;-------------------------------------------------------------------------------
;1.0  2013/11/01  A) Original derived from the POC V1.1 single-board computer
;                    firmware.
;     2013/11/04  A) Fixed a problem where the B-accumulator wasn't always being
;                    be copied to shadow storage after return from execution of
;                    a J command.
;     2017/10/07  A) Converted to use WDC's 65816 assembler (D.WERNER)
;                 B) Adapt for the RBC 65c816 SBC
;                 C) Disable X command
```

```
;-------------------------------------------------------------------------------
;
;
;               COMMENT ABBREVIATIONS
;       ---------------------------------------------------
;         BCD   binary-coded decimal
;          DP   direct page or page zero
;         EOF   end-of-field
;         EOI   end-of-input
;         LSB   least significant byte/bit
;         LSD   least significant digit
;         LSN   least significant nybble
;         LSW   least significant word
;         MPU   microprocessor
;         MSB   most significant byte/bit
;         MSD   most significant digit
;         MSN   most significant nybble
;         MSW   most-significant word
;         RAM   random access memory
;          WS   whitespace, i.e., blanks & horizontal tabs
;       ---------------------------------------------------
;       A word is defined as 16 bits.
;
;          MPU REGISTER SYMBOLS
;       --------------------------
;          .A   accumulator LSB
;          .B   accumulator MSB
;          .C   16 bit accumulator
;          .X   X-index
;          .Y   Y-index
;          DB   data bank
;          DP   direct page
;          PB   program bank
;          PC   program counter
;          SP   stack pointer
;          SR   MPU status
;       ---------------------------
;
;         MPU STATUS REGISTER SYMBOLS
;       -------------------------------
;           C   carry
;           D   decimal mode
;           I   maskable interrupts
;           m   accumulator/memory size
;           N   result negative
;           V   sign overflow
;           x   index registers size
;           Z   result zero
;       -------------------------------
;
```

```
;================================================================================
;
;SYSTEM INTERFACE DEFINITIONS
;
;       ---------------------------------------------------------------
;       This section defines the interface between Supermon 816 & the host
;       system.   Change these definitions to suit your system, but do not
;       change any label names.  All definitions must have valid values in
;       order to assemble Supermon 816.
;       ---------------------------------------------------------------
;
;       ------------------------------------------------------
;.org  $008000              ;assembly address...
;
;       Set _ORIGIN_ to Supermon 816's desired assembly address.
;       ------------------------------------------------------
;
;       ------------------------------------------------------------------------
;vecexit  = $002000              ;exit to environment address...
;
;       Set VECEXIT to where Supermon 816 should go when it exits.  Supermon 816
;       will do a JML (long jump) to this address, which means VECEXIT must be a
;       24 bit address.
;       ------------------------------------------------------------------------
;
;       ------------------------------------------------------------------------
;
;getcha                 ;get keystroke from console...
;
;       GETCHA refers to an operating system API call that returns a keystroke
;       in the 8 bit accumulator.  Supermon 816  assumes that GETCHA is a non-
;       blocking subroutine & returns with carry clear to indicate that a key-
;       stroke is in .A, or with carry set to indicate that no keystroke was
;       available.  GETCHA will be called with a JSR instruction.
;
;       Supermon 816 expects .X & .Y to be preserved upon return from GETCHA.
;       You may have to modify Supermon 816 at all calls to GETCHA if your "get
;       keystroke" routine works differently than described.
;       ------------------------------------------------------------------------
getcha          = $FF74
CURSOR          = nothere
UNCURSOR        = nothere
;-------------------------------------------------------------------------
;putcha   print character on console...
;
;       PUTCHA refers to an operating system API call that prints a character to
;       the console screen.  The character to be printed will be in .A, which
;       will be set to 8-bit width.  Supermon 816 assumes that PUTCHA will block
;       until the character can be processed.  PUTCHA will be called with a JSR
;       instructions.
```

```
;
;       Supermon 816 expects .X & .Y to be preserved upon return from PUTCHA.
;       You may have to modify Supermon 816 at all calls to PUTCHA if your "put
;       character" routine works differently than described.
;
putcha          = $FF71
;
;       ---------------------------------------------------------------------
;
;       ---------------------------------------------------------------------
vecbrki         = $0302         ;BRK handler indirect vector...
;
;       Supermon 816 will modify this vector so that execution of a BRK instruc-
;       tion is intercepted & the registers  are  captured.   Your BRK front end
;       should jump through this vector after pushing the registers as follows:
;
;                phb                    ;save DB
;                phd                    ;save DP
;                rep #%00110000         ;16 bit registers
;                pha
;                phx
;                phy
;                jmp (vecbrki)          ;indirect vector
;
;       When a G or J command is issued, the above sequence will be reversed be-
;       fore a jump is made to the code to be executed.  Upon exit from Supermon
;       816, the original address at VECBRKI will be restored.
;
;       If your BRK front end doesn't conform to the above you will have to mod-
;       ify Supermon 816 to accommodate the differences.  The most likely needed
;       changes will be in the order in which registers are pushed to the stack.
;       ---------------------------------------------------------------------
;
;       ---------------------------------------------------------------------
hwstack         = $7FFF         ;top of hardware stack...
;
;       Supermon 816 initializes the stack pointer to this address when the cold
;       start at MONCOLD is called to enter the monitor.  The stack pointer will
;       be undisturbed when entry into Supermon 816 is through JMONBRK (see jump
;       table definitions).
;       ---------------------------------------------------------------------
;
;       ---------------------------------------------------------------------
zeropage        = $10           ;Supermon 816's direct page...
;
;       Supermon 816 uses direct page starting at this address.  Be sure that no
;       conflict occurs with other software.
;       ---------------------------------------------------------------------
;
;       ---------------------------------------------------------------------
```

```
stopkey         = $03           ;display abort key...
;
;       Supermon 816 will poll for a "stop key" during display operations, such
;       as code disassembly & memory dumps, so as to abort further processing &
;       return to the command prompt.  STOPKEY must be defined with the ASCII
;       value that the "stop key" will emit when typed.  The polling is via a
;       call to GETCHA (described above).  The default STOPKEY definition of $03
;       is for ASCII <ETX> or [Ctrl-C].
;       ---------------------------------------------------------------------
;
ibuffer         = $000200       ;input buffer &...
auxbuf          = ibuffer+s_ibuf+s_byte;auxiliary buffer...
;
;       ---------------------------------------------------------------------
;       Supermon 816 will use the above definitions for input buffers.  These
;       buffers may be located anywhere in RAM that is convenient.  The buffers
;       are stateless, which means that unless Supermon 816 has control of your
;       system, they may be overwritten without consequence.
;       ---------------------------------------------------------------------
;
;=============================================================================
;
;W65C816S INSTRUCTION SYNTHESIS MACROS -- !!!!! DO NOT EDIT !!!!!
;


        .MACRO  _ASM24_ _ad
        .BYTE   <_ad,>_ad,_ad>>16
        .ENDMACRO

;brl      .macro _ad
;_ba      =*+3
;         .BYTE $82
;         .WORD _ad-_ba
;         .ENDMACRO
;
;jml      .macro _ad
;         .BYTE $5c
;         _asm24_ _ad
;         .ENDMACRO
;
;mvn      .macro _s,_d
;         .BYTE $54,_d,_s
;         .ENDMACRO
;
;mvp      .macro _s,_d
;         .BYTE $44,_d,_s
;         .ENDMACRO
;
;pea      .macro _op
```

```
;         .BYTE $f4
;         .WORD _op
;         .ENDMACRO
;
;phb      .macro
;         .BYTE $8b
;         .ENDMACRO
;
;phk      .macro
;         .BYTE $4b
;         .ENDMACRO
;
;plb      .macro
;         .BYTE $ab
;         .ENDMACRO
;
;rep      .macro _op
;         .BYTE $c2,_op
;         .ENDMACRO
;
;sep      .macro _op
;         .BYTE $e2,_op
;         .ENDMACRO
;
;tcd      .macro
;         .BYTE $5b
;         .ENDMACRO
;
;tcs      .macro
;         .BYTE $1b
;         .ENDMACRO
;
;tdc      .macro
;         .BYTE $7b
;         .ENDMACRO
;
;tsc      .macro
;         .BYTE $3b
;         .ENDMACRO
;
;txy      .macro
;         .BYTE $9b
;         .ENDMACRO
;
;tyx      .macro
;         .BYTE $bb
;         .ENDMACRO
;
;wai      .macro
;         .BYTE $cb
```

```
;         .ENDMACRO
;
;xba      .macro
;         .BYTE $eb
;         .ENDMACRO
;
        .MACRO  ADCW _OP
        ADC     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  ANDW _OP
        AND     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  BITW _OP
        BIT     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  CMPW _OP
        CMP     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  CPXW        _OP
        CPX     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  CPYW        _OP
        CPY     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  EORW        _OP
        EOR     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  LDAW        _OP
        LDA     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  LDXW        _OP
        LDX     #<_OP
        .BYTE   >_OP
        .ENDMACRO
```

```
;
        .MACRO  LDYW        _OP
        LDY     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  ORAW        _OP
        ORA     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  SBCW        _OP
        SBC     #<_OP
        .BYTE   >_OP
        .ENDMACRO
;
        .MACRO  LDALX _AD
        .BYTE   $BF
        _ASM24_ _AD
        .ENDMACRO
;
        .MACRO  ADCIL        _AD
        .BYTE   $67,_AD
        .ENDMACRO
;
        .MACRO  ADCILY        _AD
        .BYTE   $77,_AD
        .ENDMACRO
;
        .MACRO  ANDIL        _AD
        .BYTE   $27,_AD
        .ENDMACRO
;
        .MACRO  ANDILY        _AD
        .BYTE   $37,_AD
        .ENDMACRO
;
        .MACRO  CMPIL        _AD
        .BYTE   $C7,_AD
        .ENDMACRO
;
        .MACRO  CMPILY        _AD
        .BYTE   $D7,_AD
        .ENDMACRO
;
        .MACRO  EORIL        _AD
        .BYTE   $47,_AD
        .ENDMACRO
;
        .MACRO  EORILY        _AD
```

```
        .BYTE   $57,_AD
        .ENDMACRO
;
        .MACRO  LDAIL        _AD
        .BYTE   $A7,_AD
        .ENDMACRO
;
        .MACRO  LDAILY        _AD
        .BYTE   $B7,_AD
        .ENDMACRO
;
        .MACRO  ORAIL        _AD
        .BYTE   $07,_AD
        .ENDMACRO
;
        .MACRO  ORAILY        _AD
        .BYTE   $17,_AD
        .ENDMACRO
;
        .MACRO  SBCIL        _AD
        .BYTE   $E7,_AD
        .ENDMACRO
;
        .MACRO  SBCILY        _AD
        .BYTE   $F7,_AD
        .ENDMACRO
;
        .MACRO  STAIL        _AD
        .BYTE   $87,_AD
        .ENDMACRO
;
        .MACRO  STAILY        _AD
        .BYTE   $97,_AD
        .ENDMACRO
;
        .MACRO  ADCS        _OF
        .BYTE   $63,_OF
        .ENDMACRO
;
        .MACRO  ADCSI        _OF
        .BYTE   $73,_OF
        .ENDMACRO
;
        .MACRO  ANDS        _OF
        .BYTE   $23,_OF
        .ENDMACRO
;
        .MACRO  ANDSI        _OF
        .BYTE   $33,_OF
        .ENDMACRO
```

```
;
        .MACRO  CMPS        _OF
        .BYTE   $C3,_OF
        .ENDMACRO
;
        .MACRO  CMPSI        _OF
        .BYTE   $D3,_OF
        .ENDMACRO
;
        .MACRO  EORS        _OF
        .BYTE   $43,_OF
        .ENDMACRO
;
        .MACRO  EORSI        _OF
        .BYTE   $53,_OF
        .ENDMACRO
;
        .MACRO  LDAS        _OF
        .BYTE   $A3,_OF
        .ENDMACRO
;
        .MACRO  LDASI        _OF
        .BYTE   $B3,_OF
        .ENDMACRO
;
        .MACRO  ORAS        _OF
        .BYTE   $03,_OF
        .ENDMACRO
;
        .MACRO  ORASI        _OF
        .BYTE   $13,_OF
        .ENDMACRO
;
        .MACRO  SBCS        _OF
        .BYTE   $E3,_OF
        .ENDMACRO
;
        .MACRO  SBCSI        _OF
        .BYTE   $F3,_OF
        .ENDMACRO
;
        .MACRO  STAS        _OF
        .BYTE   $83,_OF
        .ENDMACRO
;
        .MACRO  STASI        _OF
        .BYTE   $93,_OF
        .ENDMACRO
;
        .MACRO  SLONGA
```

```
        .BYTE   $C2,$20
        .ENDMACRO
;
        .MACRO  LONGR
        .BYTE   $C2,$30
        .ENDMACRO
;
        .MACRO  LONGX
        .BYTE   $C2,$10
        .ENDMACRO
;
        .MACRO  SHORTA
        .BYTE   $E2,$20
        .ENDMACRO
;
        .MACRO  SHORTI
        .BYTE   $E2,$10
        .ENDMACRO
;
        .MACRO  SHORTR
        .BYTE   $E2,$30
        .ENDMACRO
;
        .MACRO  SHORTX
        .BYTE   $E2,$10
        .ENDMACRO
;
;===============================================================================
;
;CONSOLE DISPLAY CONTROL MACROS
;
;       ------------------------------------------------------------------
;       The following macros execute terminal  control procedures  that  perform
;       such tasks as clearing the screen,  switching  between  normal & reverse
;       video, etc.  These macros are for WYSE 60 & compatible displays, such as
;       the WYSE 150, WYSE 160, WYSE 325 & WYSE GPT.   Only the functions needed
;       by Supermon 816 are included.
;
;       If your console is not WYSE 60 compatible, you will need to  edit  these
;       macros as required to control your particular console or terminal.  Note
;       that in some cases one macro may call another.  Exercise caution in your
;       edits to avoid introducing display bugs.
;
;       If your console display cannot execute one of these procedures,  such as
;       'CL' (clear to end of line), you will have to develop an alternative.
;       ------------------------------------------------------------------
;
;
;
;       cursor control...
```

```
;
        .MACRO  CRR             ;carriage return
        .BYTE   a_cr
        .ENDMACRO
;

        .MACRO  LF              ;carriage return/line feed
        CRR
        .BYTE   a_lf
        .ENDMACRO
;
;       miscellaneous control...
;
        .MACRO  RB              ;ring "bell"
        .BYTE   a_bel
        .ENDMACRO
;
;
;===============================================================================
;
;ASCII CONTROL DEFINITIONS (menmonic order)
;
a_bel           = $07           ;<BEL> alert/ring bell
a_bs            = $08           ;<BS>  backspace
a_cr            = $0d           ;<CR>  carriage return
a_del           = $7f           ;<DEL> delete
a_esc           = $1b           ;<ESC> escape
a_ht            = $09           ;<HT>  horizontal tabulation
a_lf            = $0a           ;<LF>  linefeed
;
;
;       miscellaneous (description order)...
;
a_blank         = ' '           ;blank (whitespace)
a_asclch        = 'z'           ;end of lowercase ASCII
a_lctouc        = $5f           ;LC to UC conversion mask
a_asclcl        = 'a'           ;start of lowercase ASCII
;
;===============================================================================
;
;GLOBAL ATOMIC CONSTANTS
;
;
;       data type sizes...
;
s_byte          = 1             ;byte
s_word          = 2             ;word (16 bits)
s_xword         = 3             ;extended word (24 bits)
s_dword         = 4             ;double word (32 bits)
s_rampag        = $0100         ;65xx RAM page
```

```
;
;
;       data type sizes in bits...
;
s_bibyte        = 8             ;byte
s_bnybbl        = 4             ;nybble
;
;
;       miscellaneous...
;
bitabs          = $2c           ;absolute BIT opcode
bitzp           = $24           ;zero page BIT opcode
;
;================================================================================
;
;W65C816S NATIVE MODE STATUS REGISTER DEFINITIONS
;
s_mpudbx        = s_byte        ;data bank size
s_mpudpx        = s_word        ;direct page size
s_mpupbx        = s_byte        ;program bank size
s_mpupcx        = s_word        ;program counter size
s_mpuspx        = s_word        ;stack pointer size
s_mpusrx        = s_byte        ;status size
;
;
;       status register flags...
;
sr_car          = %00000001     ;C
sr_zer          = sr_car<<1     ;Z
sr_irq          = sr_zer<<1     ;I
sr_bdm          = sr_irq<<1     ;D
sr_ixw          = sr_bdm<<1     ;x
sr_amw          = sr_ixw<<1     ;m
sr_ovl          = sr_amw<<1     ;V
sr_neg          = sr_ovl<<1     ;N
;
;       NVmxDIZC
;       xxxxxxxx
;       ||||||||
;       |||||||+---> 1 = carry set/generated
;       ||||||+----> 1 = result = zero
;       |||||+-----> 1 = IRQs ignored
;       ||||+------> 0 = binary arithmetic mode
;       ||||         1 = decimal arithmetic mode
;       |||+-------> 0 = 16 bit index
;       |||          1 = 8 bit index
;       ||+--------> 0 = 16 bit .A & memory
;       ||           1 = 8 bit .A & memory
;       |+---------> 1 = sign overflow
;       +----------> 1 = result = negative
```

```
;
;================================================================================
;
;"SIZE-OF" CONSTANTS
;
s_addr          = s_xword       ;24 bit address
s_auxbuf        = 32            ;auxiliary buffer
s_ibuf          = 69            ;input buffer
s_mnemon        = 3             ;MPU ASCII mnemonic
s_mnepck        = 2             ;MPU encoded mnemonic
s_mvinst        = 3             ;MVN/MVP instruction
s_opcode        = s_byte        ;MPU opcode
s_oper          = s_xword       ;operand
s_pfac          = s_dword       ;primary math accumulator
s_sfac          = s_dword+s_word;secondary math accumulators
;
;================================================================================
;
;"NUMBER-OF" CONSTANTS
;
n_dbytes        = 21            ;default disassembly bytes
n_dump          = 8             ;bytes per memory dump line
n_mbytes        = s_rampag-1    ;default memory dump bytes
n_hccols        = 10            ;compare/hunt display columns
n_opcols        = 3*s_oper      ;disassembly operand columns
n_opslsr        = 4             ;LSRs to extract instruction size
n_shfenc        = 5             ;shifts to encode/decode mnemonic
;
;================================================================================
;
;NUMERIC CONVERSION CONSTANTS
;
a_hexdec        = 'A'-'9'-2     ;hex to decimal difference
c_bin           = '%'           ;binary prefix
c_dec           = '+'           ;decimal prefix
c_hex           = '$'           ;hexadecimal prefix
c_oct           = '@'           ;octal prefix
k_hex           = 'f'           ;hex ASCII conversion
m_bits          = s_pfac*s_bibyte;operand bit size
m_cbits         = s_sfac*s_bibyte;workspace bit size
bcdumask        = %00001111     ;isolate BCD units mask
btoamask        = %00110000     ;binary to ASCII mask
;
;================================================================================
;
;ASSEMBLER/DISASSEMBLER CONSTANTS
;
a_mnecvt        = '?'           ;encoded mnemonic conversion base
aimmaska        = %00011111     ;.A immediate opcode test #1
aimmaskb        = %00001001     ;.A immediate opcode test #2
```

```
asmprfx         = 'A'           ;assemble code prefix
ascprmct        = 9             ;assembler prompt "size-of"
disprfx         = '.'           ;disassemble code prefix
flimmask        = %11000000     ;force long immediate flag
opc_cpxi        = $e0           ;CPX # opcode
opc_cpyi        = $c0           ;CPY # opcode
opc_ldxi        = $a2           ;LDX # opcode
opc_ldyi        = $a0           ;LDY # opcode
opc_mvn         = $54           ;MVN opcode
opc_mvp         = $44           ;MVP opcode
opc_rep         = $c2           ;REP opcode
opc_sep         = $e2           ;SEP opcode
pfmxmask        = sr_amw|sr_ixw ;MPU m & x flag bits mask
;
;
;       assembler prompt buffer offsets...
;
apadrbkh        = s_word        ;instruction address bank MSN
apadrbkl        = apadrbkh+s_byte;instruction address bank LSN
apadrmbh        = apadrbkl+s_byte;instruction address MSB MSN
apadrmbl        = apadrmbh+s_byte;instruction address MSB LSN
apadrlbh        = apadrmbl+s_byte;instruction address LSB MSN
apadrlbl        = apadrlbh+s_byte;instruction address LSB LSN
;
;
;       addressing mode preamble symbols...
;
amp_flim        = '!'           ;force long immediate
amp_imm         = '#'           ;immediate
amp_ind         = '('           ;indirect
amp_indl        = '['           ;indirect long
;
;
;       addressing mode symbolic translation indices...
;
am_nam          = %0000         ;no symbol
am_imm          = %0001         ;#
am_adrx         = %0010         ;<addr>,X
am_adry         = %0011         ;<addr>,Y
am_ind          = %0100         ;(<addr>)
am_indl         = %0101         ;[<dp>]
am_indly        = %0110         ;[<dp>],Y
am_indx         = %0111         ;(<addr>,X)
am_indy         = %1000         ;(<dp>),Y
am_stk          = %1001         ;<offset>,S
am_stky         = %1010         ;(<offset>,S),Y
am_move         = %1011         ;<sbnk>,<dbnk>
;
;
;       operand size translation indices...
```

```
;
ops0            = %0000<<4      ;no operand
ops1            = %0001<<4      ;8 bit operand
ops2            = %0010<<4      ;16 bit operand
ops3            = %0011<<4      ;24 bit operand
bop1            = %0101<<4      ;8 bit relative branch
bop2            = %0110<<4      ;16 bit relative branch
vops            = %1001<<4      ;8 or 16 bit operand
;
;
;       operand size & addressing mode extraction masks...
;
amodmask        = %00001111     ;addressing mode index
opsmask         = %00110000     ;operand size
vopsmask        = %11000000     ;BOPx & VOPS flag bits
;
;
;       instruction mnemonic encoding...
;
mne_adc         = $2144         ;ADC
mne_and         = $2bc4         ;AND
mne_asl         = $6d04         ;ASL
mne_bcc         = $2106         ;BCC
mne_bcs         = $a106         ;BCS
mne_beq         = $9186         ;BEQ
mne_bit         = $aa86         ;BIT
mne_bmi         = $5386         ;BMI
mne_bne         = $33c6         ;BNE
mne_bpl         = $6c46         ;BPL
mne_bra         = $14c6         ;BRA
mne_brk         = $64c6         ;BRK
mne_brl         = $6cc6         ;BRL
mne_bvc         = $25c6         ;BVC
mne_bvs         = $a5c6         ;BVS
mne_clc         = $2348         ;CLC
mne_cld         = $2b48         ;CLD
mne_cli         = $5348         ;CLI
mne_clv         = $bb48         ;CLV
mne_cmp         = $8b88         ;CMP
mne_cop         = $8c08         ;COP
mne_cpx         = $cc48         ;CPX
mne_cpy         = $d448         ;CPY
mne_dec         = $218a         ;DEC
mne_dex         = $c98a         ;DEX
mne_dey         = $d18a         ;DEY
mne_eor         = $9c0c         ;EOR
mne_inc         = $23d4         ;INC
mne_inx         = $cbd4         ;INX
mne_iny         = $d3d4         ;INY
mne_jml         = $6b96         ;JML
```

```
mne_jmp         = $8b96         ;JMP
mne_jsl         = $6d16         ;JSL
mne_jsr         = $9d16         ;JSR
mne_lda         = $115a         ;LDA
mne_ldx         = $c95a         ;LDX
mne_ldy         = $d15a         ;LDY
mne_lsr         = $9d1a         ;LSR
mne_mvn         = $7ddc         ;MVN
mne_mvp         = $8ddc         ;MVP
mne_nop         = $8c1e         ;NOP
mne_ora         = $14e0         ;ORA
mne_pea         = $11a2         ;PEA
mne_pei         = $51a2         ;PEI
mne_per         = $99a2         ;PER
mne_pha         = $1262         ;PHA
mne_phb         = $1a62         ;PHB
mne_phd         = $2a62         ;PHD
mne_phk         = $6262         ;PHK
mne_php         = $8a62         ;PHP
mne_phx         = $ca62         ;PHX
mne_phy         = $d262         ;PHY
mne_pla         = $1362         ;PLA
mne_plb         = $1b62         ;PLB
mne_pld         = $2b62         ;PLD
mne_plp         = $8b62         ;PLP
mne_plx         = $cb62         ;PLX
mne_ply         = $d362         ;PLY
mne_rep         = $89a6         ;REP
mne_rol         = $6c26         ;ROL
mne_ror         = $9c26         ;ROR
mne_rti         = $5566         ;RTI
mne_rtl         = $6d66         ;RTL
mne_rts         = $a566         ;RTS
mne_sbc         = $20e8         ;SBC
mne_sec         = $21a8         ;SEC
mne_sed         = $29a8         ;SED
mne_sei         = $51a8         ;SEI
mne_sep         = $89a8         ;SEP
mne_sta         = $1568         ;STA
mne_stp         = $8d68         ;STP
mne_stx         = $cd68         ;STX
mne_sty         = $d568         ;STY
mne_stz         = $dd68         ;STZ
mne_tax         = $c8aa         ;TAX
mne_tay         = $d0aa         ;TAY
mne_tcd         = $292a         ;TCD
mne_tcs         = $a12a         ;TCS
mne_tdc         = $216a         ;TDC
mne_trb         = $1cea         ;TRB
mne_tsb         = $1d2a         ;TSB
```

```
mne_tsc         = $252a         ;TSC
mne_tsx         = $cd2a         ;TSX
mne_txa         = $166a         ;TXA
mne_txs         = $a66a         ;TXS
mne_txy         = $d66a         ;TXY
mne_tya         = $16aa         ;TYA
mne_tyx         = $ceaa         ;TYX
mne_wai         = $50b0         ;WAI
mne_wdm         = $7170         ;WDM
mne_xba         = $10f2         ;XBA
mne_xce         = $3132         ;XCE
;
;
;       encoded instruction mnemonic indices...
;
mne_adcx        = 16            ;ADC
mne_andx        = 29            ;AND
mne_aslx        = 44            ;ASL
mne_bccx        = 15            ;BCC
mne_bcsx        = 65            ;BCS
mne_beqx        = 59            ;BEQ
mne_bitx        = 70            ;BIT
mne_bmix        = 36            ;BMI
mne_bnex        = 31            ;BNE
mne_bplx        = 42            ;BPL
mne_brax        = 5             ;BRA
mne_brkx        = 39            ;BRK
mne_brlx        = 43            ;BRL
mne_bvcx        = 23            ;BVC
mne_bvsx        = 68            ;BVS
mne_clcx        = 20            ;CLC
mne_cldx        = 27            ;CLD
mne_clix        = 35            ;CLI
mne_clvx        = 71            ;CLV
mne_cmpx        = 53            ;CMP
mne_copx        = 55            ;COP
mne_cpxx        = 78            ;CPX
mne_cpyx        = 88            ;CPY
mne_decx        = 18            ;DEC
mne_dexx        = 74            ;DEX
mne_deyx        = 84            ;DEY
mne_eorx        = 61            ;EOR
mne_incx        = 21            ;INC
mne_inxx        = 77            ;INX
mne_inyx        = 87            ;INY
mne_jmlx        = 40            ;JML
mne_jmpx        = 54            ;JMP
mne_jslx        = 45            ;JSL
mne_jsrx        = 63            ;JSR
mne_ldax        = 1             ;LDA
```

```
mne_ldxx        = 73            ;LDX
mne_ldyx        = 83            ;LDY
mne_lsrx        = 64            ;LSR
mne_mvnx        = 48            ;MVN
mne_mvpx        = 58            ;MVP
mne_nopx        = 56            ;NOP
mne_orax        = 6             ;ORA
mne_peax        = 2             ;PEA
mne_peix        = 33            ;PEI
mne_perx        = 60            ;PER
mne_phax        = 3             ;PHA
mne_phbx        = 10            ;PHB
mne_phdx        = 26            ;PHD
mne_phkx        = 38            ;PHK
mne_phpx        = 51            ;PHP
mne_phxx        = 75            ;PHX
mne_phyx        = 85            ;PHY
mne_plax        = 4             ;PLA
mne_plbx        = 11            ;PLB
mne_pldx        = 28            ;PLD
mne_plpx        = 52            ;PLP
mne_plxx        = 76            ;PLX
mne_plyx        = 86            ;PLY
mne_repx        = 49            ;REP
mne_rolx        = 41            ;ROL
mne_rorx        = 62            ;ROR
mne_rtix        = 37            ;RTI
mne_rtlx        = 46            ;RTL
mne_rtsx        = 67            ;RTS
mne_sbcx        = 14            ;SBC
mne_secx        = 19            ;SEC
mne_sedx        = 25            ;SED
mne_seix        = 34            ;SEI
mne_sepx        = 50            ;SEP
mne_stax        = 7             ;STA
mne_stpx        = 57            ;STP
mne_stxx        = 80            ;STX
mne_styx        = 89            ;STY
mne_stzx        = 91            ;STZ
mne_taxx        = 72            ;TAX
mne_tayx        = 82            ;TAY
mne_tcdx        = 24            ;TCD
mne_tcsx        = 66            ;TCS
mne_tdcx        = 17            ;TDC
mne_trbx        = 12            ;TRB
mne_tsbx        = 13            ;TSB
mne_tscx        = 22            ;TSC
mne_tsxx        = 79            ;TSX
mne_txax        = 8             ;TXA
mne_txsx        = 69            ;TXS
```

```
mne_txyx        = 90            ;TXY
mne_tyax        = 9             ;TYA
mne_tyxx        = 81            ;TYX
mne_waix        = 32            ;WAI
mne_wdmx        = 47            ;WDM
mne_xbax        = 0             ;XBA
mne_xcex        = 30            ;XCE
;
;================================================================================
;
;MISCELLANEOUS CONSTANTS
;
halftab         = 4             ;1/2 tabulation spacing
memprfx         = '>'           ;memory dump prefix
memsepch        = ':'           ;memory dump separator
memsubch        = '.'           ;memory dump non-print char
srinit          = %00110000     ;SR initialization value
;
;================================================================================
;
;DIRECT PAGE STORAGE
;
reg_pbx         = zeropage      ;PB
reg_pcx         = reg_pbx+s_mpupbx;PC
reg_srx         = reg_pcx+s_mpupcx;SR
reg_ax          = reg_srx+s_mpusrx;.C
reg_xx          = reg_ax+s_word ;.X
reg_yx          = reg_xx+s_word ;.Y
reg_spx         = reg_yx+s_word ;SP
reg_dpx         = reg_spx+s_mpuspx;DP
reg_dbx         = reg_dpx+s_mpudpx;DB
;
;
;       general workspace...
;
addra           = reg_dbx+s_mpudbx;address #1
addrb           = addra+s_addr  ;address #2
faca            = addrb+s_addr  ;primary accumulator
facax           = faca+s_pfac   ;extended primary accumulator
facb            = facax+s_pfac  ;secondary accumulator
facc            = facb+s_sfac   ;tertiary accumulator
operand         = facc+s_sfac   ;instruction operand
auxbufix        = operand+s_oper;auxiliary buffer index
ibufidx         = auxbufix+s_byte;input buffer index
bitsdig         = ibufidx+s_byte;bits per numeral
numeral         = bitsdig+s_byte;numeral buffer
radix           = numeral+s_byte;radix index
admodidx        = radix+s_byte  ;addressing mode index
charcnt         = admodidx+s_byte;character counter
instsize        = charcnt+s_word;instruction size
```

```
mnepck          = instsize+s_word;encoded mnemonic
opcode          = mnepck+s_mnepck;current opcode
status          = opcode+s_byte ;I/O status flag
xrtemp          = status+s_byte ;temp .X storage
eopsize         = xrtemp+s_byte ;entered operand size
flimflag        = eopsize+s_byte;forced long immediate...
vecbrkia        = flimflag+s_byte;system indirect BRK vector
;
;       xx000000
;       ||
;       |+---------> 0: .X/.Y =  8 bits
;       |            1: .X/.Y =  18 bits
;       +----------> 0: .A =  8 bits
;                    1: .A =  16 bits
;
;       ---------------------------------------------------------------------
;       During assembly, FLIMFLAG indicates the operand size used with an immed-
;       iate mode instruction, thus causing the following disassembly to display
;       the assembled  operand size.   During disassembly,  FLIMFLAG will mirror
;       the effect of the most recent REP or SEP instruction.
;       ---------------------------------------------------------------------
;
iopsize         = flimflag+s_byte;operand size
range           = iopsize+s_byte;allowable radix range
vopsflag        = range+s_byte  ;VOPS & ROPS mode bits
;
;
;       copy/fill workspace (overlaps some of the above)...
;
mcftwork        = faca          ;start of copy/fill code
mcftopc         = mcftwork+s_byte;instruction opcode
mcftbnk         = mcftopc+s_byte;banks
;
;=============================================================================
;
;SUPERMON 816 JUMP TABLE
;
;
JMON:
        BRA     mon             ;cold start entry
JMONBRK:
        BRA     monbrk          ;software interrupt intercept
;
;
;=============================================================================
;
;mon: SUPERMON 816 COLD START
;
mon:
        INDEX16
```

```
        ACCUMULATOR8
        LDY     #$0000          ; LOAD $00 INTO Y
OUTSTRLP:
        LDA     ALIVEM,Y        ; LOAD NEXT CHAR FROM STRING INTO ACC
        CMP     #$00            ; IS NULL?
        BEQ     ENDOUTSTR       ; YES, END PRINT OUT
        JSR     putcha          ; PRINT CHAR IN ACC
        INY                     ; Y=Y+1 (BUMP INDEX)
        JMP     OUTSTRLP        ; DO NEXT CHAR
ENDOUTSTR:
        INDEX8


        SLONGA
        LDA     vecbrki         ;BRK vector
        CMPW    monbrk          ;pointing at monitor?
        BNE     moncontinue
        JMP     monreg          ;yes, ignore cold start
;
moncontinue:
        STA     vecbrkia        ;save vector for exit
        LDAW    monbrk          ;Supermon 816 intercepts...
        STA     vecbrki         ;BRK handler
        SHORTR                  ;8 bit registers
        LDX     #vopsflag-reg_pbx
;
_0000010:
        STZ     reg_pbx,x       ;clear DP storage
        DEX
        BPL     _0000010
;
;
;       initialize register shadows...
;
        LDA     #srinit
        STA     reg_srx         ;status register
        SLONGA                  ;16 bit .A
        LDAW    hwstack         ;top of hardware stack
        TCS                     ;set SP
        TDC                     ;get & save...
        STA     reg_dpx         ;DP register
        LDAW    0
        SHORTA
        PHK
        PLA                     ;capture PB &...
        STA     reg_pbx         ;set
        PHB
        PLA                     ;capture DB &...
        STA     reg_dbx         ;set
;
```

```
;
;       print startup banner...
;
        PEA     mm_entry        ;"...ready..."
        BRA     moncom
;
;================================================================================
;
;monbrk: SOFTWARE INTERRUPT INTERCEPT
;
;       --------------------------------------------------------------------
;       This is the entry point taken when a BRK instruction is executed.  It is
;       assumed that the BRK  handler has pushed the registers to the stack that
;       are not automatically pushed by the MPU in response to BRK.
;       --------------------------------------------------------------------
;
monbrk:
        CLD                     ; VERIFY DECIMAL MODE IS OFF
        CLC                     ;
        XCE                     ; SET NATIVE MODE
        PHB                     ;save DB
        PHD                     ;save DP
        SLONGA                  ;16 bit .A
        PHA
        LDAW    $0000           ;set DPR
        TCD                     ;
        PLA
        LONGR                   ;store 16 bit registers
        STA     <reg_ax         ;.A
        STX     <reg_xx         ;.X
        STY     <reg_yx         ;.Y
        PLA                     ;get DP &...
        STA     <reg_dpx        ;store
        SHORTX
        PLX                     ;get DB &...
        STX     <reg_dbx        ;store
        PLX                     ;get SR &...
        STX     <reg_srx        ;store
        PLA                     ;get PC &...
        STA     <reg_pcx        ;store
        PLX                     ;get PB &...
        STX     <reg_pbx        ;store
        SLONGA
        LDAW    hwstack         ;top of hardware stack
        TCS                     ;set SPR
;        CLI                     ;reenable IRQs
        SEI                     ;Disable Interrupts
        SHORTA
        LDA     #$00            ;set DBR
        PHA
```

```
        PLB
        PEA     mm_brk          ;"*BRK"
;
;===============================================================================
;
;moncom: COMMON ENTRY POINT
;
;       ------------------------------------
;       DO NOT directly call this entry point!
;       ------------------------------------
;
moncom:
        JSR     sprint          ;print heading
        SLONGA
        TSC                     ;get SP &...
        STA     <reg_spx        ;store
        REP     #%11111111      ;clear SR &...
        SEP     #srinit         ;set default state
        SEC                     ;see next
;
;===============================================================================
;
;monreg: DISPLAY MPU REGISTERS
;
;       ---------
;       syntax: R
;       ---------
;
monreg:
        BCS     _0010010        ;okay to proceed
;
        JMP     monerr          ;error if called with a parm
;
_0010010:
        PEA     mm_regs
        JSR     sprint          ;display heading
;
;
;       display program bank & counter...
;
        SHORTA
        LDA     <reg_pbx        ;PB
        JSR     dpyhex          ;display as hex ASCII
        JSR     printspc        ;inter-field space

        SLONGA
        LDA     <reg_pcx
        SHORTA
        JSR     dpyhexw         ;display PC
        LDX     #2
```

```
        JSR     multspc         ;inter-field spacing
;
;
;       display SR in bitwise fashion...
;
        LDX     <reg_srx        ;SR
        LDY     #s_bibyte       ;bits in a byte
;
_0010020:
        TXA                     ;remaining SR bits
        ASL                     ;grab one of them
        TAX                     ;save remainder
        LDA     #'0'            ;a clear bit but...
        ADC     #0              ;adjust if set &...
        JSR     putcha          ;print
        DEY                     ;bit processed
        BNE     _0010020        ;do another
;
;
;       display .C, .X, .Y, SP & DP...
;
_0010030:
        JSR     printspc        ;spacing
        SLONGA
        LDA     reg_ax,y        ;get register value
        SHORTA
        JSR     dpyhexw         ;convert & display

;         .rept s_word
        INY
        INY
;         .endr

        CPY     #reg_dbx-reg_ax-2
        BCC     _0010030        ;next

        PEA     mm_regs1
        JSR     sprint          ;display heading
        SLONGA
        LDA     <reg_dpx        ;get register value
        SHORTA
        JSR     dpyhexw         ;convert & display
;
;
;       display DB...
;
        JSR     printspc        ;more spacing
        LDA     <reg_dbx        ;get DB &...
        JSR     dpyhex          ;display it
;
```

```
;===============================================================================
;
;monce: COMMAND EXECUTIVE
;
monce:
        SHORTA
        LDA     #0              ;default buffer index
;
moncea:
        SHORTR                  ;alternate entry point
        STA     ibufidx         ;(re)set buffer index
        PEA     mm_prmpt
        JSR     sprint          ;display input prompt
        JSR     input           ;await some input
;
_0020010:
        JSR     getcharc        ;read from buffer
        BEQ     monce           ;terminator, just loop
;
        CMP     #a_blank
        BEQ     _0020010        ;strip leading blanks
;
        LDX     #n_mpctab-1     ;number of primary commands
;
_0020020:
        CMP     mpctab,x        ;search primary command list
        BNE     _0020030
;
        TXA                     ;get index
        ASL                     ;double for offset
        TAX
        SLONGA
        LDA     mpcextab,x      ;command address -1
        PHA                     ;prime the stack
        SHORTA
        JMP     getparm         ;evaluate parm & execute command
;
_0020030:
        DEX
        BPL     _0020020        ;continue searching primary commands
;
        LDX     #n_radix-1      ;number of radices
;
_0020040:
        CMP     radxtab,x       ;search conversion command list
        BNE     _0020050
;
        JMP     monenv          ;convert & display parameter
;
_0020050:
```

```
        DEX
        BPL     _0020040
;
;===============================================================================
;
;monerr: COMMON ERROR HANDLER
;
monerr:
        SHORTR                  ;8 bit registers
;
monerraa:
        JSR     dpyerr          ;indicate an error &...
        BRA     monce           ;return to input loop

;
;===============================================================================
;
;monasc: ASSEMBLE CODE
;
;       ---------------------------------------------------------------------
;       syntax: A <addr> <mnemonic> [<argument>]
;
;       After a line of code has been successfully assembled it will be disass-
;       embled & displayed,  & the monitor will prompt with the next address to
;       which code may be assembled.
;       ---------------------------------------------------------------------
;
monasc:
        BCC     _0030020        ;assembly address entered
;
_0030010:
        JMP     monerr          ;terminate w/error
;
;
;       evaluate assembly address...
;
_0030020:
        JSR     facasize        ;check address...
        CMP     #s_dword        ;range
        BCS     _0030010        ;out of range - error
;
        JSR     facaddra        ;store assembly address
;
;
;       initialize workspace...
;
        LDX     #s_auxbuf-s_byte
;
_0030030:
        STZ     auxbuf,x        ;clear addressing mode buffer
```

```
        DEX
        BNE     _0030030
;
        LDA     #a_blank
        STA     auxbuf          ;preamble placeholder
        JSR     clroper         ;clear operand
        STZ     auxbufix        ;reset addressing mode index
        STZ     flimflag        ;clear forced long immediate
        STZ     mnepck          ;clear encoded...
        STZ     mnepck+s_byte   ;mnemonic workspace
        STZ     vopsflag        ;clear 8/16 or relative flag
;
;
;       encode mnemonic...
;
        LDY     #s_mnemon       ;expected mnemonic size
;
_0030040:
        JSR     getcharw        ;get from buffer wo/whitespace
        BNE     _0030060        ;gotten
;
        CPY     #s_mnemon       ;any input at all?
        BCC     _0030050        ;yes
;
        JMP     monce           ;no, abort further assembly
;
_0030050:
        JMP     monasc10        ;incomplete mnemonic - error
;
_0030060:
        SEC
        SBC     #a_mnecvt       ;ASCII to binary factor
        LDX     #n_shfenc       ;shifts required to encode
;
_0030070:
        LSR                     ;shift out a bit...
        ROR     mnepck+s_byte   ;into...
        ROR     mnepck          ;encoded mnemonic
        DEX
        BNE     _0030070        ;next bit
;
        DEY
        BNE     _0030040        ;get next char
;
;
;       test for copy instruction...
;       ----------------------------------------------------------------------
;       The MVN & MVP instructions accept two operands & hence have an irregular
;       syntax.  Therefore, special handling is necessary to assemble either of
;       these instructions.
```

```
;
;       The official WDC syntax has the programmer entering a pair of 24 bit ad-
;       dresses as operands, with the assembler isolating bits 16-23 to use as
;       operands.  This formality has been dispensed with in this monitor & the
;       operands are expected to be 8 bit bank values.
;       ---------------------------------------------------------------------
;
        SLONGA                  ;16 bit load
        LDA     mnepck          ;packed menmonic
        LDX     #opc_mvn        ;MVN opcode
        CMPW    mne_mvn         ;is it MVN?
        BEQ     monasc01        ;yes
;
        LDX     #opc_mvp        ;MVP opcode
        CMPW    mne_mvp         ;is it MVP?
        BNE     monasc02        ;no
;
;
;       assemble copy instruction...
;
monasc01:
        STX     opcode          ;store relevant opcode
        SHORTA
        JSR     instdata        ;get instruction data
        STX     eopsize         ;effective operand size
        INX
        STX     instsize        ;instruction size
        LDX     #s_oper-s_word  ;operand index
        STX     xrtemp          ;set it
;
_0040010:
        JSR     ascbin          ;evaluate bank number
        BCS     monasc04        ;conversion error
;
        BEQ     monasc04        ;nothing returned - error
;
        JSR     facasize        ;bank must be...
        CMP     #s_word         ;8 bits
        BCS     monasc04        ;it isn't - error
;
        LDA     faca            ;bank
        LDX     xrtemp          ;operand index
        STA     operand,x       ;store
        DEC     xrtemp          ;index=index-1
        BPL     _0040010        ;get destination bank
;
        JSR     getcharr        ;should be no more input
        BNE     monasc04        ;there is - error
;
        JMP     monasc08        ;finish MVN/MVP assembly
```

```
;
;
;       continue with normal assembly...
;
monasc02:
        SHORTA                  ;back to 8 bits
;
monasc03:
        JSR     getcharw        ;get next char
        BEQ     monasc06        ;EOI, no argument
;
        CMP     #amp_flim
        BNE     _0050010        ;no forced long immediate
;
        LDA     flimflag        ;FLIM already set?
        BNE     monasc04        ;yes - error
;
        LDA     #flimmask
        STA     flimflag        ;set flag &...
        BRA     monasc03        ;get next char
;
_0050010:
        CMP     #amp_imm        ;immediate mode?
        BEQ     _0050020        ;yes
;
        CMP     #amp_ind        ;indirect mode?
        BEQ     _0050020        ;yes
;
        CMP     #amp_indl       ;indirect long mode?
        BNE     _0050030        ;no
;
_0050020:
        STA     auxbuf          ;set addressing mode preamble
        INC     auxbufix        ;bump aux buffer index &...
        BRA     _0050040        ;evaluate operand
;
_0050030:
        DEC     ibufidx         ;position back to char
;
_0050040:
        JSR     ascbin          ;evaluate operand
        BNE     monasc05        ;evaluated
;
        BCS     monasc04        ;conversion error
;
        LDA     auxbufix        ;no operand...any preamble?
        BEQ     monasc06        ;no, syntax is okay so far
;
monasc04:
        JMP     monasc10        ;abort w/error
```

```
;
monasc05:
        JSR     facasize        ;size operand
        CMP     #s_dword        ;max is 24 bits
        BCS     monasc04        ;too big
;
        STA     eopsize         ;save operand size
        JSR     facaoper        ;store operand
;
monasc06:
        DEC     ibufidx         ;back to last char
        LDX     auxbufix        ;mode buffer index
        BNE     _0060010        ;preamble in buffer
;
        INX                     ;step past preamble position
;
_0060010:
        JSR     getcharc        ;get a char w/forced UC
        BEQ     _0060030        ;EOI
;
        CPX     #s_auxbuf       ;mode buffer full?
        BCS     monasc04        ;yes, too much input
;
_0060020:
        STA     auxbuf,x        ;store for comparison
        INX
        BNE     _0060010
;
;
;       evaluate mnemonic...
;
_0060030:
        LDX     #n_mnemon-1     ;starting mnemonic index
;
monasc07:
        TXA                     ;convert index...
        ASL                     ;to offset
        TAY                     ;now mnemonic table index
        SLONGA                  ;16 bit compare
        LDA     mnetab,y        ;get mnemonic from table
        CMP     mnepck          ;compare to entered mnemonic
        SHORTA                  ;back to 8 bits
        BEQ     _0070020        ;match
;
_0070010:
        DEX                     ;try next mnemonic
        BMI     monasc04        ;unknown mnemonic - error
;
        BRA     monasc07        ;keep going
;
```

```
_0070020:
        STX     mnepck          ;save mnemonic index
        TXA
        LDX     #0              ;trial opcode
;
_0070030:
        CMP     mnetabix,x      ;search index table...
        BEQ     _0070050        ;for a match
;
_0070040:
        INX                     ;keep going until we...
        BNE     _0070030        ;search entire table
;
        BRA     monasc04        ;this shouldn't happen!
;
;       ---------------------------------------------------------------
;       If the mnemonic index table search fails then there is a coding error
;       somewhere, as every entry in the mnemonic table is supposed to have a
;       matching cardinal index.
;       ---------------------------------------------------------------
;
;
;       evaluate addressing mode...
;
_0070050:
        STX     opcode          ;save trial opcode
        JSR     instdata        ;get related instruction data
        STA     vopsflag        ;save 8/16 or relative flag
        STX     iopsize         ;operand size
        INX
        STX     instsize        ;instruction size
        LDX     opcode          ;recover trial opcode
        TYA                     ;addressing mode
        ASL                     ;create table index
        TAY
        SLONGA
        LDA     ms_lutab,y      ;mode lookup table
        STA     addrb           ;set pointer
        SHORTA
        LDY     #0
;
_0070060:
        LDA     (addrb),y       ;table addressing mode
        CMP     auxbuf,y        ;entered addressing mode
        BEQ     _0070080        ;okay so far
;
_0070070:
        LDA     mnepck          ;reload mnemonic index
        BRA     _0070040        ;wrong opcode for addresing mode
;
```

```
_0070080:
        ORA     #0              ;last char the terminator?
        BEQ     _0070090        ;yes, evaluate operand
;
        INY
        BRA     _0070060        ;keep testing
;
;
;       evaluate operand...
;
_0070090:
        LDA     eopsize         ;entered operand size
        BNE     _0070100        ;non-zero
;
        ORA     iopsize         ;instruction operand size
        BNE     _0070070        ;wrong opcode - keep trying
;
        BRA     monasc08        ;assemble instruction
;
_0070100:
        BIT     vopsflag        ;is this a branch?
        BVS     _0070160        ;yes, evaluate
;
        LDA     iopsize         ;instruction operand size
        BIT     vopsflag        ;variable size operand allowed?
        BMI     _0070130        ;yes
;
        BIT     flimflag        ;was forced immediate set?
        BPL     _0070110        ;no
;
        JMP     monasc10        ;yes - error
;
_0070110:
        CMP     eopsize         ;entered operand size
        BCC     _0070070        ;operand too big
;
        STA     eopsize         ;new operand size
        BRA     monasc08        ;assemble, otherwise...
;
_0070120:
        CMP     eopsize         ;exact size match required
        BNE     _0070070        ;mismatch - wrong opcode
;
        BRA     monasc08        ;assemble
;
;
;       process variable size immediate mode operand...
;
_0070130:
        LDX     eopsize         ;entered operand size
```

```
        CPX     #s_xword        ;check size
        BCS     monasc10        ;too big - error
;
        BIT     flimflag        ;forced long immediate?
        BPL     _0070140        ;no
;
        LDX     #s_word         ;promote operand size to...
        STX     eopsize         ;16 bits
        BRA     _0070150
;
_0070140:
        CPX     #s_word         ;16 bits?
        BNE     _0070150        ;no
;
        LDY     #flimmask       ;yes so force long...
        STY     flimflag        ;immediate disassembly
;
_0070150:
        INA                     ;new instruction operand size
        CMP     eopsize         ;compare against operand size
        BCC     _0070070        ;mismatch - can't assemble
;
        BRA     monasc08        ;okay, assemble
;
;
;       process relative branch...
;
_0070160:
        JSR     targoff         ;compute branch offset
        BCS     monasc10        ;branch out of range
;
        STA     eopsize         ;effective operand size
;
;
;       assemble instruction...
;
monasc08:
        LDA     opcode          ;opcode
        STAIL   addra           ;store at assembly address
        LDX     eopsize         ;any operand to process?
        BEQ     _0080020        ;no
;
        TXY                     ;also storage offset
;
_0080010:
        DEX
        LDA     operand,x       ;get operand byte &...
        STAILY  addra           ;poke into memory
        DEY
        BNE     _0080010        ;next
```

```
;
_0080020:
        LDA     #a_cr
        JSR     putcha          ;return to left margin
        LDA     #asmprfx        ;assembly prefix
        JSR     dpycodaa        ;disassemble & display
;
;
;       prompt for next instruction...
;
monasc09:
        LDA     #a_blank
        LDX     #ascprmct-1
;
_0090010:
        STA     ibuffer,x       ;prepare buffer for...
        DEX                     ;next instruction
        BPL     _0090010
;
        LDA     #asmprfx        ;assemble code...
        STA     ibuffer         ;prompt prefix
        LDA     addra+s_word    ;next instruction address bank
        JSR     binhex          ;convert to ASCII
        STA     ibuffer+apadrbkh;store MSN in buffer
        STX     ibuffer+apadrbkl;store LSN in buffer
        LDA     addra+s_byte    ;next instruction address MSB
        JSR     binhex
        STA     ibuffer+apadrmbh
        STX     ibuffer+apadrmbl
        LDA     addra           ;next instruction address LSB
        JSR     binhex
        STA     ibuffer+apadrlbh
        STX     ibuffer+apadrlbl
        LDA     #ascprmct       ;effective input count
        JMP     moncea          ;reenter input loop
;
;
;       process assembly error...
;
monasc10:
        JSR     dpyerr          ;indicate error &...
        BRA     monasc09        ;prompt w/same assembly address
;
;================================================================================
;
;mondsc: DISASSEMBLE CODE
;
;       ----------------------------
;       syntax: D [<addr1> [<addr2>]]
;       ----------------------------
```

```
;
mondsc:
        BCS     _0100010        ;no parameters
;
        STZ     flimflag        ;reset to 8 bit mode
        JSR     facasize        ;check starting...
        CMP     #s_dword        ;address
        BCS     _0100050        ;out of range - error
;
        JSR     facaddra        ;copy starting address
        JSR     getparm         ;get ending address
        BCC     _0100020        ;gotten
;
_0100010:
        JSR     clrfaca         ;clear accumulator
        SLONGA
        CLC
        LDA     addra           ;starting address
        ADCW    n_dbytes        ;default bytes
        STA     faca            ;effective ending address
        SHORTA
        LDA     addra+s_word    ;starting bank
        ADC     #0
        STA     faca+s_word     ;effective ending bank
        BCS     _0100050        ;end address > $FFFFFF
;
_0100020:
        JSR     facasize        ;check ending...
        CMP     #s_dword        ;address
        BCS     _0100050        ;out of range - error
;
        JSR     facaddrb        ;set ending address
        JSR     getparm         ;check for excess input
        BCC     _0100050        ;present - error
;
        JSR     calccnt         ;calculate bytes
        BCC     _0100050        ;end < start
;
_0100030:
        JSR     teststop        ;test for display stop
        BCS     _0100040        ;stopped
;
        JSR     newline         ;next line
        JSR     dpycod          ;disassemble & display
        JSR     decdcnt         ;decrement byte count
        BCC     _0100030        ;not done
;
_0100040:
        JMP     monce           ;back to main loop
;
```

```
_0100050:
        JMP     monerr          ;address range error
;
;================================================================================
;
;monjmp: EXECUTE CODE
;
;       ---------------------------------------------------------
;       syntax: G [<dp>]
;
;       If no address is specified, the current values in the PB & PC
;       shadow registers are used.
;       ---------------------------------------------------------
;
monjmp:
        JSR     setxaddr        ;set execution address
        BCS     monjmpab        ;out of range - error
;
        JSR     getparm         ;check for excess input
        BCC     monjmpab        ;too much input - error
;
        SLONGA                  ;16 bit .A
        LDA     reg_spx
        TCS                     ;restore SP
;
monjmpaa:
        SHORTA
        LDA     reg_pbx
        PHA                     ;restore PB
        SLONGA
        LDA     reg_pcx
        PHA                     ;restore PC
        SHORTA
        LDA     reg_srx
        PHA                     ;restore SR
        LDA     reg_dbx
        PHA
        PLB                     ;restore DB
        LONGR
        LDA     reg_dpx
        TCD                     ;restore DP
        LDA     reg_ax          ;restore .C
        LDX     reg_xx          ;restore .X
        LDY     reg_yx          ;restore .Y
        RTI                     ;execute code
;
monjmpab:
        JMP     monerr          ;error
;
;================================================================================
```

```
;
;monjsr: EXECUTE CODE AS SUBROUTINE
;
;       ------------------------------------------------------------
;       syntax: J [<dp>]
;
;       If no address is specified the current values in the PB & PC
;       shadow registers are used.   An RTS at the end of the called
;       subroutine will return control to the monitor  provided  the
;       stack remains in balance.
;       ------------------------------------------------------------
;
monjsr:
        JSR     setxaddr        ;set execution address
        BCS     monjmpab        ;out of range - error
;
        JSR     getparm         ;check for excess input
        BCC     monjmpab        ;too much input - error
;
        SLONGA
        LDA     reg_spx
        TCS                     ;restore SP &...
        JSR     monjmpaa        ;call subroutine
        PHP                     ;push SR
        LONGR
        STA     reg_ax          ;save...
        STX     reg_xx          ;register...
        STY     reg_yx          ;returns
        SHORTX                  ;8 bit .X & .Y
        PLX                     ;get & save...
        STX     reg_srx         ;return SR
        TSC                     ;get & save...
        STA     reg_spx         ;return SP
        TDC                     ;get & save...
        STA     reg_dpx         ;DP pointer
        SHORTA                  ;8 bit .A
        PHK                     ;get &...
        PLA                     ;save...
        STA     reg_pbx         ;return PB
        PHB                     ;get &...
        PLA                     ;save...
        STA     reg_dbx         ;return DB
        PEA     mm_rts          ;"*RET"
        JMP     moncom          ;return to monitor
;
;================================================================================
;
;monchm: CHANGE and/or DUMP MEMORY
;
;       ----------------------------------------
```

```
;       syntax: > [<addr> <operand> [<operand>]...]
;
;       > <addr> without operands will dump 16 bytes
;       of memory, starting at <addr>.
;       ----------------------------------------
;
monchm:
        BCS     _0110030        ;no address given - quit
;
        JSR     facasize        ;size address
        CMP     #s_dword
        BCS     _0110040        ;address out of range - error
;
        JSR     facaddra        ;set starting address
        JSR     getpat          ;evaluate change pattern
        BCC     _0110010        ;entered
;
        BPL     _0110020        ;not entered
;
        BRA     _0110040        ;evaluation error
;
_0110010:
        DEY                     ;next byte
        BMI     _0110020        ;done
;
        LDA     auxbuf,y        ;write pattern...
        STAILY  addra           ;to memory
        BRA     _0110010        ;next
;
_0110020:
        JSR     newline         ;next line
        JSR     dpymem          ;regurgitate changes
;
_0110030:
        JMP     monce           ;back to command loop
;
_0110040:
        JMP     monerr          ;goto error handler
;
;=============================================================================
;
;moncmp: COMPARE MEMORY
;
;       ----------------------------
;       syntax: C <start> <end> <ref>
;       ----------------------------
;
moncmp:
        BCS     _0120030        ;start not given - quit
;
```

```
        JSR     enddest         ;get end & reference addresses
        BCS     _0120040        ;range or other error
;
        STZ     xrtemp          ;column counter
;
_0120010:
        JSR     teststop        ;check for stop
        BCS     _0120030        ;abort
;
        LDAIL   addra           ;get from reference location
        CMPIL   operand         ;test against compare location
        BEQ     _0120020        ;match, don't display address
;
        JSR     dpycaddr        ;display current location
;
_0120020:
        JSR     nxtaddra        ;next reference location
        BCS     _0120030        ;done
;
        SLONGA
        INC     operand         ;bump bits 0-15
        SHORTA
        BNE     _0120010
;
        INC     operand+s_word  ;bump bits 16-23
        BRA     _0120010
;
_0120030:
        JMP     monce           ;return to command exec
;
_0120040:
        JMP     monerr          ;goto error handler
;
;===============================================================================
;
;moncpy: COPY (transfer) MEMORY
;
;       -------------------------------
;       syntax: T <start> <end> <target>
;       -------------------------------
;
moncpy:
        BCS     _0130040        ;start not given - quit
;
        JSR     enddest         ;get end & target addresses
        BCS     _0130050        ;range or other error
;
        SLONGA
        SEC
        LDA     addrb           ;ending address
```

```
        SBC     addra           ;starting address
        BCC     _0130050        ;start > end - error
;
        STA     facb            ;bytes to copy
        SHORTA
        LONGX
        LDA     operand+s_word  ;target bank
        LDY     operand         ;target address
        CMP     addra+s_word    ;source bank
        SLONGA
        BNE     _0130020        ;can use forward copy
;
        CPY     addra           ;source address
        BCC     _0130020        ;can use forward copy
;
        BNE     _0130010        ;must use reverse copy
;
        BRA     _0130050        ;copy in place - error
;
_0130010:
        LDA     facb            ;get bytes to copy
        PHA                     ;protect
        JSR     lodbnk          ;load banks
        JSR     cprvsup         ;do reverse copy setup
        PLA                     ;get bytes to copy
        TAX                     ;save a copy
        CLC
        ADC     operand         ;change target to...
        TAY                     ;target end
        TXA                     ;recover bytes to copy
        LDX     addrb           ;source end
        BRA     _0130030
;
_0130020:
        LDA     facb            ;get bytes to copy
        PHA                     ;protect
        JSR     lodbnk          ;load banks
        JSR     cpfwsup         ;do forward copy setup
        PLA                     ;get bytes to copy
        LDX     addra           ;source start
;
_0130030:
        JMP     mcftwork        ;copy memory
;
_0130040:
        JMP     monce           ;back to executive
;
_0130050:
        JMP     monerr          ;error
;
```

```
;================================================================================
;
;mondmp: DISPLAY MEMORY RANGE
;
;       ----------------------------
;       syntax: M [<addr1> [<addr2>]]
;       ----------------------------
;
mondmp:
        BCS     _0140010        ;no parameters
;
        JSR     facasize        ;check address...
        CMP     #s_dword        ;range
        BCS     _0140050        ;address out of range
;
        JSR     facaddra        ;copy starting address
        JSR     getparm         ;get ending address
        BCC     _0140020        ;gotten
;
_0140010:
        JSR     clrfaca         ;clear accumulator
        SLONGA
        CLC
        LDA     addra           ;starting address
        ADCW    n_mbytes        ;default bytes
        STA     faca            ;effective ending address
        SHORTA
        LDA     addra+s_word    ;starting bank
        ADC     #0
        STA     faca+s_word     ;effective ending bank
        BCS     _0140050        ;end address > $FFFFFF
;
_0140020:
        JSR     facasize        ;check ending address...
        CMP     #s_dword        ;range
        BCS     _0140050        ;out of range - error
;
        JSR     facaddrb        ;copy ending address
        JSR     getparm         ;check for excess input
        BCC     _0140050        ;error
;
        JSR     calccnt         ;calculate bytes to dump
        BCC     _0140050        ;end < start
;
_0140030:
        JSR     teststop        ;test for display stop
        BCS     _0140040        ;stopped
;
        JSR     newline         ;next line
        JSR     dpymem          ;display
```

```
        JSR     decdcnt         ;decrement byte count
        BCC     _0140030        ;not done
;
_0140040:
        JMP     monce           ;back to main loop
;
_0140050:
        JMP     monerr          ;address range error
;
;================================================================================
;
;monfil: FILL MEMORY
;
;       ---------------------------------------
;       syntax: F <start> <end> <fill>
;
;       <start> & <end> must be in the same bank.
;       ---------------------------------------
;
monfil:
        BCS     _0150010        ;start not given - quit
;
        JSR     facasize        ;check size
        CMP     #s_dword
        BCS     _0150020        ;out of range - error...
;
        JSR     facaddra        ;store start
        JSR     getparm         ;evaluate end
        BCS     _0150020        ;not entered - error
;
        JSR     facasize        ;check size
        CMP     #s_dword
        BCS     _0150020        ;out of range - error
;
        LDA     faca+s_word     ;end bank
        CMP     addra+s_word    ;start bank
        BNE     _0150020        ;not same - error
;
        JSR     facaddrb        ;store <end>
        SLONGA
        SEC
        LDA     addrb           ;ending address
        SBC     addra           ;starting address
        BCC     _0150020        ;start > end - error
;
        STA     facb            ;bytes to copy
        SHORTA
        JSR     getparm         ;evaluate <fill>
        BCS     _0150020        ;not entered - error
;
```

```
        JSR     facasize        ;<fill> should be...
        CMP     #s_word         ;8 bits
        BCS     _0150020        ;it isn't - error
;
        JSR     facaoper        ;store <fill>
        JSR     getparm         ;should be no more parameters
        BCC     _0150020        ;there are - error
;
        LDA     operand         ;<fill>
        STAIL   addra           ;fill 1st location
        LONGR                   ;16 bit operations
        LDA     facb            ;get byte count
        BEQ     _0150010        ;only 1 location - finished
;
        DEA                     ;zero align &...
        PHA                     ;protect
        SHORTA
        LDA     addra+s_word    ;start bank
        XBA
        LDA     addrb+s_word    ;end bank
        JSR     cpfwsup         ;do forward copy setup
        PLA                     ;recover fill count
        LDX     addra           ;fill-from starting location
        TXY
        INY                     ;fill-to starting location
        JMP     mcftwork        ;fill memory
;
_0150010:
        JMP     monce           ;goto command executive
;
_0150020:
        JMP     monerr          ;goto error handler
;
;================================================================================
;
;monhnt: SEARCH (hunt) MEMORY
;
;       ----------------------------------
;       syntax: H <addr1> <addr2> <pattern>
;       ----------------------------------
;
monhnt:
        BCS     _0160050        ;no start address
;
        JSR     facasize        ;size starting address
        CMP     #s_dword
        BCS     _0160060        ;address out of range - error
;
        JSR     facaddra        ;store starting address
        JSR     getparm         ;evaluate ending address
```

```
        BCS     _0160060        ;no address - error
;
        JSR     facasize        ;size ending address
        CMP     #s_dword
        BCS     _0160060        ;address out of range - error
;
        JSR     facaddrb        ;store ending address
        JSR     calccnt         ;calculate byte range
        BCC     _0160060        ;end < start
;
        JSR     getpat          ;evaluate search pattern
        BCS     _0160060        ;error
;
        STZ     xrtemp          ;clear column counter
;
_0160010:
        JSR     teststop        ;check for stop
        BCS     _0160050        ;abort
;
        LDY     auxbufix        ;pattern index
;
_0160020:
        DEY
        BMI     _0160030        ;pattern match
;
        LDAILY  addra           ;get from memory
        CMP     auxbuf,y        ;test against pattern
        BNE     _0160040        ;mismatch, next location
;
        BEQ     _0160020        ;match, keep testing
;
_0160030:
        JSR     dpycaddr        ;display current location
;
_0160040:
        JSR     nxtaddra        ;next location
        BCC     _0160010        ;not done
;
_0160050:
        JMP     monce           ;back to executive
;
_0160060:
        JMP     monerr          ;goto error handler
;
;===============================================================================
;
;monenv: CONVERT NUMERIC VALUE
;
;       ----------------------
;       syntax: <radix><value>
```

```
;       ----------------------
;
monenv:
        JSR     getparmr        ;reread & evaluate parameter
        BCS     _0170020        ;none entered
;
        LDX     #0              ;radix index
        LDY     #n_radix        ;number of radices
;
_0170010:
        PHY                     ;save counter
        PHX                     ;save radix index
        JSR     newline         ;next line &...
        JSR     clearlin        ;clear it
        LDA     #a_blank
        LDX     #halftab
        JSR     multspc         ;indent 1/2 tab
        PLX                     ;get radix index but...
        PHX                     ;put it back
        LDA     radxtab,x       ;get radix
        JSR     binasc          ;convert to ASCII
        PHY                     ;string address MSB
        PHX                     ;string address LSB
        JSR     sprint          ;print
        PLX                     ;get index again
        PLY                     ;get counter
        INX
        DEY                     ;all radices handled?
        BNE     _0170010        ;no

_0170020:
        JMP     monce           ;back to command exec
;
;===============================================================================
;
;monchr: CHANGE REGISTERS
;
;       ------------------------------------------------------
;       syntax: ; [PB [PC [.S [.C [.X [.Y [SP [DP [DB]]]]]]]]]
;
;       ; with no parameters is the same as the R command.
;       ------------------------------------------------------
;
monchr:
        BCS     _0570040        ;dump registers & quit
;
        LDY     #0              ;register counter
        STY     facc            ;initialize register index
;
_0570010:
```

```
        JSR     facasize        ;get parameter size
        CMP     rcvltab,y       ;check against size table
        BCS     _0570050        ;out of range
;
        LDA     rcvltab,y       ;determine number of bytes...
        CMP     #s_word+1       ;to store
        ROR     facc+s_byte     ;condition flag
        BPL     _0570020        ;8 bit register size
;
        SLONGA                  ;16 bit register size
;
_0570020:
        LDX     facc            ;get register index
        LDA     faca            ;get parm
        STA     reg_pbx,x       ;put in shadow storage
        SHORTA
        ASL     facc+s_byte     ;mode flag to carry
        TXA                     ;register index
        ADC     #s_byte         ;at least 1 byte stored
        STA     facc            ;save new index
        JSR     getparm         ;get a parameter
        BCS     _0570040        ;EOI
;
        INY                     ;bump register count
        CPY     #n_regchv       ;all registers processed?
        BNE     _0570010        ;no, keep going
;
_0570030:
        JSR     alert           ;excessive input
;
_0570040:
        JMP     monreg          ;display changes
;
_0570050:
        JMP     monerr          ;goto error handler
;
;================================================================================
;
;monxit: EXIT TO OPERATING ENVIRONMENT
;
;       ---------
;       syntax: X
;       ---------
;
;monxit   bcc _0180020          ;no parameters allowed
;
;         slonga
;        lda vecbrki           ;BRK indirect vector
;         cmpw monbrk           ;we intercept it?
;         bne _0180010          ;no, don't change it
```

```
;
;         lda vecbrkia          ;old vector
;         sta vecbrki           ;restore it
;         stz vecbrkia          ;invalidate old vector
;
;_0180010 shortr
;         jml vecexit           ;long jump to exit
;
;_0180020 jmp monerr            ;goto error handler
;
; * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * *
; * *                                             * *
; * * S T A R T   o f   S U B R O U T I N E S * *
; * *                                             * *
; * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * *
;
;dpycaddr: DISPLAY CURRENT ADDRESS IN COLUMNS
;
dpycaddr:
        LDX     xrtemp          ;column count
        BNE     _0190010        ;not at right side
;
        JSR     newline         ;next row
        LDX     #n_hccols       ;max columns
;
_0190010:
        CPX     #n_hccols       ;max columns
        BEQ     _0190020        ;at left margin
;
        LDA     #a_ht
        JSR     putcha          ;tab a column
;
_0190020:
        DEX                     ;one less column
        STX     xrtemp          ;save column counter
        JMP     prntladr        ;print reference address
;
;================================================================================
;
;dpycod: DISASSEMBLE & DISPLAY CODE
;
;       ------------------------------------------------------------------------
;       This function disassembles & displays the machine code at  the  location
;       pointed to by ADDRA.  Upon return, ADDRA will point to the opcode of the
;       next instruction.   The entry point at DPYCODAA  should be called with a
;       disassembly prefix character loaded in .A.   If entered  at  DPYCOD, the
;       default character will be display at the beginning of each  disassembled
;       instruction.
```

```
;
;       The disassembly of immediate mode instructions that can take an 8 or  16
;       bit operand is affected by the bit pattern that is  stored  in  FLIMFLAG
;       upon entry to this function:
;
;           FLIMFLAG: xx000000
;                     ||
;                     |+---------> 0:  8 bit .X or .Y operand
;                     |            1: 16 bit .X or .Y operand
;                     +----------> 0:  8 bit .A or BIT # operand
;                                  1: 16 bit .A or BIT # operand
;
;       FLIMFLAG is conditioned according to the operand of  the  most  recently
;       disassembled REP or SEP instruction.   Hence repetitive  calls  to  this
;       subroutine will usually result in the correct disassembly of 16 bit imm-
;       ediate mode instructions.
;       ------------------------------------------------------------------------
;
dpycod:
        LDA     #disprfx        ;default prefix
;
;
;       alternate prefix display entry point...
;
dpycodaa:
        JSR     putcha          ;print prefix
        JSR     printspc        ;space
        JSR     prntladr        ;print long address
        JSR     printspc        ;space to opcode field
        JSR     getbyte         ;get opcode
        STA     opcode          ;save &...
        JSR     printbyt        ;display as hex
;
;
;       decode menmonic & addressing info...
;
        LDX     opcode          ;current mnemonic
        LDA     mnetabix,x      ;get mnemonic index
        ASL                     ;double for...
        TAY                     ;mnemonic table offset
        SLONGA                  ;16 bit load
        LDA     mnetab,y        ;copy encoded mnemonic to...
        STA     mnepck          ;working storage
        SHORTA                  ;back to 8 bits
        JSR     instdata        ;extract mode & size data
        STA     vopsflag        ;save mode flags
        STY     admodidx        ;save mode index
        ASL                     ;variable immediate instruction?
        BCC     dpycod01        ;no, effective operand size in .X
;
```

```
;
;       determine immediate mode operand size...
;
        LDA     opcode          ;current opcode
        BIT     flimflag        ;operand display mode
        BPL     _0200010        ;8 bit .A & BIT immediate mode
;
        AND     #aimmaska       ;determine if...
        CMP     #aimmaskb       ;.A or BIT immediate
        BEQ     _0200030        ;display 16 bit operand
;
        LDA     opcode          ;not .A or BIT immediate
;
_0200010:
        BVC     dpycod01        ;8 bit .X/.Y immediate mode
;
        LDY     #n_vopidx-1     ;opcodes to test
;
_0200020:
        CMP     vopidx,y        ;looking for LDX #, CPY #, etc.
        BEQ     _0200040        ;disassemble a 16 bit operand
;
        DEY
        BPL     _0200020        ;keep trying
;
        BRA     dpycod01        ;not .X or .Y immediate
;
_0200030:
        LDA     opcode          ;reload
;
_0200040:
        INX                     ;16 bit operand
;
;
;       get & display operand bytes...
;
dpycod01:
        STX     iopsize         ;operand size...
        INX                     ;plus opcode becomes...
        STX     instsize        ;instruction size
        STX     charcnt         ;total bytes to process
        LDA     #n_opcols+2     ;total operand columns plus WS
        STA     xrtemp          ;initialize counter
        JSR     clroper         ;clear operand
        LDY     iopsize         ;operand size
        BEQ     _0210020        ;no operand
;
        LDX     #0              ;operand index
;
_0210010:
```

```
        JSR     getbyte         ;get operand byte
        STA     operand,x       ;save
        PHX                     ;protect operand index
        JSR     printbyt        ;print operand byte
        DEC     xrtemp          ;3 columns used, 2 for...
        DEC     xrtemp          ;operand nybbles &...
        DEC     xrtemp          ;1 for whitespace
        PLX                     ;get operand index
        INX                     ;bump it
        DEY
        BNE     _0210010        ;next
;
_0210020:
        LDX     xrtemp          ;operand columns remaining
        JSR     multspc         ;space to mnemonic field
;
;
;       display mnemonic...
;
        LDY     #s_mnemon       ;size of ASCII mnemonic
;
_0210030:
        LDA     #0              ;initialize char
        LDX     #n_shfenc       ;shifts to execute
;
_0210040:
        ASL     mnepck          ;shift encoded mnemonic
        ROL     mnepck+s_byte
        ROL
        DEX
        BNE     _0210040
;
        ADC     #a_mnecvt       ;convert to ASCII &...
        PHA                     ;stash
        DEY
        BNE     _0210030        ;continue with mnemonic
;
        LDY     #s_mnemon
;
_0210050:
        PLA                     ;get mnenmonic byte
        JSR     putcha          ;print it
        DEY
        BNE     _0210050
;
;
;       display operand...
;
        LDA     iopsize         ;operand size
        BEQ     clearlin        ;zero, disassembly finished
```

```
;
        JSR     printspc        ;space to operand field
        BIT     vopsflag        ;check mode flags
        BVC     dpycod02        ;not a branch
;
        JSR     offtarg         ;compute branch target
        LDX     instsize        ;effective instruction size
        DEX
        STX     iopsize         ;effective operand size
;
dpycod02:
        STZ     vopsflag        ;clear
        LDA     admodidx        ;instruction addressing mode
        CMP     #am_move        ;block move instruction?
        BNE     _0220010        ;no
;
        ROR     vopsflag        ;yes
;
_0220010:
        ASL                     ;convert addressing mode to...
        TAX                     ;symbology table index
        SLONGA                  ;do a 16 bit load
        LDA     ms_lutab,x      ;addressing symbol pointer
        PHA
        SHORTA                  ;back to 8 bit loads
        LDY     #0
        LDASI   1               ;get 1st char
        CMP     #a_blank
        BEQ     _0220020        ;no addresing mode preamble
;
        JSR     putcha          ;print preamble
;
_0220020:
        LDA     #c_hex
        JSR     putcha          ;operand displayed as hex
        LDY     iopsize         ;operand size = index
;
_0220030:
        DEY
        BMI     _0220040        ;done with operand
;
        LDA     operand,y       ;get operand byte
        JSR     dpyhex          ;print operand byte
        BIT     vopsflag        ;block move?
        BPL     _0220030        ;no
;
        STZ     vopsflag        ;reset
        PHY                     ;protect operand index
        PEA     ms_move
        JSR     sprint          ;display MVN/MVP operand separator
```

```
        PLY                     ;recover operand index again
        BRA     _0220030        ;continue
;
_0220040:
        PLX                     ;symbology LSB
        PLY                     ;symbology MSB
        INX                     ;move past preamble
        BNE     _0220050
;
        INY
;
_0220050:
        PHY
        PHX
        JSR     sprint          ;print postamble, if any
;
;
;       condition immediate mode display format...
;
dpycod03:
        LDA     operand         ;operand LSB
        AND     #pfmxmask       ;isolate M & X bits
        ASL                     ;shift to match...
        ASL                     ;FLIMFLAG alignment
        LDX     opcode          ;current instruction
        CPX     #opc_rep        ;was it REP?
        BNE     _0230010        ;no
;
        TSB     flimflag        ;set flag bits as required
        BRA     clearlin
;
_0230010:
        CPX     #opc_sep        ;was it SEP?
        BNE     clearlin        ;no, just exit
;
        TRB     flimflag        ;clear flag bits as required
;
;================================================================================
;
;clearlin: CLEAR DISPLAY LINE
;
clearlin:
        RTS
;
;================================================================================
;
;dpyibuf: DISPLAY MONITOR INPUT BUFFER CONTENTS
;
dpyibuf:
        PEA     ibuffer
```

```
        BRA     dpyerraa
;
;===============================================================================
;
;dpymem: DISPLAY MEMORY
;
;       ---------------------------------------------------------
;       This function displays 16 bytes of memory as hex values & as
;       ASCII equivalents.  The starting address for the display is
;       in ADDRA & is expected to be a 24 bit address.  Upon return,
;       ADDRA will point to the start of the next 16 bytes.
;       ---------------------------------------------------------
;
dpymem:
        SHORTR
        STZ     charcnt         ;reset
;         lda #memprfx
;         jsr putcha            ;display prefix
        JSR     prntladr        ;print 24 bit address
        LDX     #0              ;string buffer index
        LDY     #n_dump         ;bytes per line
;
_0240010:
        JSR     getbyte         ;get from RAM, also...
        PHA                     ;save for decoding
        PHX                     ;save string index
        JSR     printbyt        ;display as hex ASCII
        INC     charcnt         ;bytes displayed +1
        PLX                     ;recover string index &...
        PLA                     ;byte
        CMP     #a_blank        ;printable?
        BCC     _0240020        ;no
;
        CMP     #a_del
        BCC     _0240030        ;is printable
;
_0240020:
        LDA     #memsubch       ;substitute character
;
_0240030:
        STA     ibuffer,x       ;save char
        INX                     ;bump index
        DEY                     ;byte count -= 1
        BNE     _0240010        ;not done
;
        STZ     ibuffer,x       ;terminate ASCII string
        LDA     #memsepch
        JSR     putcha          ;separate ASCII from bytes
        JSR     dpyibuf         ;display ASCII equivalents
        RTS
```

```
;
;================================================================================
;
;dpyerr: DISPLAY ERROR SIGNAL
;
dpyerr:
        PEA     mm_err          ;"*ERR"
;
dpyerraa:
        JSR     sprint
        RTS
;
;================================================================================
;
;gendbs: GENERATE DESTRUCTIVE BACKSPACE
;
gendbs:
        PEA     dc_bs           ;destructive backspace
        BRA     dpyerraa
;
;================================================================================
;
;prntladr: PRINT 24 BIT CURRENT ADDRESS
;
prntladr:
        PHP                     ;protect register sizes
        SHORTA
        LDA     addra+s_word    ;get bank byte &...
        JSR     dpyhex          ;display it
        SLONGA
        LDA     addra           ;get 16 bit address
        PLP                     ;restore register sizes
;
;================================================================================
;
;dpyhexw: DISPLAY BINARY WORD AS HEX ASCII
;
;       -----------------------------------
;       Preparatory Ops: .C: word to display
;
;       Returned Values: .C: used
;                        .X: used
;                        .Y: entry value
;       -----------------------------------
;
dpyhexw:
        PHP                     ;save register sizes
        SLONGA
        PHA                     ;protect value
        SHORTA
```

```
        XBA                     ;get MSB &...
        JSR     dpyhex          ;display
        SLONGA
        PLA                     ;recover value
        SHORTA                  ;only LSB visible
        PLP                     ;reset register sizes
;
;===============================================================================
;
;dpyhex: DISPLAY BINARY BYTE AS HEX ASCII
;
;       ----------------------------------
;       Preparatory Ops: .A: byte to display
;
;       Returned Values: .A: used
;                        .X: used
;                        .Y: entry value
;       ----------------------------------
;
dpyhex:
        JSR     binhex          ;convert to hex ASCII
        JSR     putcha          ;print MSN
        TXA
        JMP     putcha          ;print LSN
;
;===============================================================================
;
;multspc: PRINT MULTIPLE BLANKS
;
;       ------------------------------------------------
;       Preparatory Ops : .X: number of blanks to print
;
;       Register Returns: none
;
;       Calling Example : ldx #3
;                         jsr multspc    ;print 3 spaces
;
;       Notes: This sub will print 1 blank if .X=0.
;       ------------------------------------------------
;

multspc:
        TXA
        BNE     _0250010        ;blank count specified
;
        INX                     ;default to 1 blank
;
_0250010:
        JSR     printspc
        DEX
```

```
        BNE     _0250010
;
        RTS

;
;===============================================================================
;
;newline: PRINT NEWLINE (CRLF)
;
newline:
        PEA     dc_lf
        BRA     dpyerraa
;
;===============================================================================
;
;printbyt: PRINT A BYTE WITH LEADING SPACE
;
printbyt:
        PHA                     ;protect byte
        JSR     printspc        ;print leading space
        PLA                     ;restore &...
        BRA     dpyhex          ;print byte
;
;===============================================================================
;
;alert: ALERT USER w/TERMINAL BELL
;
alert:
        LDA     #a_bel
        BRA     printcmn
;
;===============================================================================
;
;printspc: PRINT A SPACE
;
printspc:
        LDA     #a_blank
;
printcmn:
        JMP     putcha
;
;===============================================================================
;
;sprint: PRINT NULL-TERMINATED CHARACTER STRING
;
;       ---------------------------------------------------
;       Preparatory Ops : SP+1: string address LSB
;                         SP+2: string address MSB
;
;       Register Returns: .A: used
```

```
;                       .B: entry value
;                       .X: used
;                       .Y: used
;
;       MPU Flags: NVmxDIZC
;                  ||||||||
;                  |||||||+---> 0: okay
;                  |||||||      1: string too long (1)
;                  ||||+++----> not defined
;                  |||+-------> 1
;                  ||+--------> 1
;                  ++---------> not defined
;
;       Example: PER STRING
;                JSR SPRINT
;                BCS TOOLONG
;
;       Notes: 1) Maximum permissible string length including the
;                 terminator is 32,767 bytes.
;              2) All registers are forced to 8 bits.
;              3) DO NOT JUMP OR BRANCH INTO THIS FUNCTION!
;       ------------------------------------------------------------
;
sprint:
        SHORTA                  ;8 bit accumulator
        LONGX                   ;16 bit index
;
;------------------------------------------------------------
_retaddr        = 1             ;return address
_src            = _retaddr+s_word;string address stack offset
;------------------------------------------------------------
;
        LDYW    0
        CLC                     ;no initial error
;
_0260010:
        LDASI   _src            ;get a byte
        BEQ     _0260020        ;done
;
        JSR     putcha          ;write to console port
        INY
        BPL     _0260010        ;next
;
        SEC                     ;string too long
;
_0260020:
        PLX                     ;pull RTS address
        PLY                     ;clear string pointer
        PHX                     ;replace RTS
        SHORTX
```

```
        RTS
;
;=============================================================================
;
;ascbin: CONVERT NULL-TERMINATED ASCII NUMBER STRING TO BINARY
;
;       ------------------------------------------------
;       Preparatory Ops: ASCII number string in IBUFFER
;
;       Returned Values: FACA: converted parameter
;                        .A: used
;                        .X: used
;                        .Y: used
;                        .C: 1 = conversion error
;                        .Z: 1 = nothing to convert
;
;       Notes: 1) Conversion stops when a non-numeric char-
;                 acter is encountered.
;              2) Radix symbols are as follows:
;
;                 % binary
;                 % octal
;                 + decimal
;                 $ hexadecimal
;
;                 Hex is the default if no radix is speci-
;                 fied in the 1st character of the string.
;       ------------------------------------------------
;
ascbin:
        SHORTR
        JSR     clrfaca         ;clear accumulator
        STZ     charcnt         ;zero char count
        STZ     radix           ;initialize
;
;
;       process radix if present...
;
        JSR     getcharw        ;get next non-WS char
        BNE     _0270010        ;got something
;
        CLC                     ;no more input
        RTS
;
_0270010:
        LDX     #n_radix-1      ;number of radices
;
_0270020:
        CMP     radxtab,x       ;recognized radix?
        BEQ     _0270030        ;yes
```

```
;
        DEX
        BPL     _0270020        ;try next
;
        DEC     ibufidx         ;reposition to previous char
        INX                     ;not recognized, assume hex
;
_0270030:
        CMP     #c_dec          ;decimal radix?
        BNE     _0270040        ;not decimal
;
        ROR     radix           ;flag decimal conversion
;
_0270040:
        LDA     basetab,x       ;number bases table
        STA     range           ;set valid numeral range
        LDA     bitsdtab,x      ;get bits per digit
        STA     bitsdig         ;store
;
;
;       process numerals...
;
ascbin01:
        JSR     getchar         ;get next char
        BNE     _TMP0001        ;not EOI
        JMP     ascbin03        ;EOI
;
_TMP0001:
        CMP     #' '
        BEQ     ascbin03        ;blank - EOF
;
        CMP     #','
        BEQ     ascbin03        ;comma - EOF
;
        CMP     #a_ht
        BEQ     ascbin03        ;tab - EOF
;
        JSR     nybtobin        ;change to binary
        BCS     ascbin04        ;not a recognized numeral
;
        CMP     range           ;check range
        BCS     ascbin04        ;not valid for base
;
        STA     numeral         ;save processed numeral
        INC     charcnt         ;bump numeral count
        BIT     radix           ;working in base 10?
        BPL     _1570030        ;no
;
;
;       compute N*2 for decimal conversion...
```

```
;
        LDX     #0              ;accumulator index
        LDY     #s_pfac/2       ;iterations
        SLONGA
        CLC
;
_1570020:
        LDA     faca,x          ;N
        ROL                     ;N=N*2
        STA     facb,x
        INX
        INX
        DEY
        BNE     _1570020
;
        BCS     ascbin04        ;overflow - error
;
        SHORTA
;
;
;       compute N*base for binary, octal or hex...
;       or N*8 for decimal...
;
_1570030:
        LDX     bitsdig         ;bits per digit
        SLONGA                  ;16 bit shifts
;
_1570040:
        ASL     faca
        ROL     faca+s_word
        BCS     ascbin04        ;overflow - error
;
        DEX
        BNE     _1570040        ;next shift
;
        SHORTA                  ;back to 8 bits
        BIT     radix           ;check base
        BPL     ascbin02        ;not decimal
;
;
;       compute N*10 for decimal (N*8 + N*2)...
;
        LDY     #s_pfac
        SLONGA
;
_1570050:
        LDA     faca,x          ;N*8
        ADC     facb,x          ;N*2
        STA     faca,x          ;now N*10
        INX
```

```
        INX
        DEY
        BNE     _1570050
;
        BCS     ascbin04        ;overflow - error
;
        SHORTA
;
;
;       add current numeral to partial result...
;
ascbin02:
        LDA     faca            ;N
        ADC     numeral         ;N=N+D
        STA     faca
        LDX     #1
        LDY     #s_pfac-1
;
_0280010:
        LDA     faca,x
        ADC     #0              ;account for carry
        STA     faca,x
        INX
        DEY
        BNE     _0280010
;
        BCC     _0280020        ;next if no overflow
;
        BCS     ascbin04        ;overflow - error
;
;
;       finish up...
;
ascbin03:
        CLC                     ;no error
;
ascbin04:
        SHORTA                  ;reset if necessary
        LDA     charcnt         ;load char count
        RTS                     ;done
_0280020:
        JMP     ascbin01        ;next if no overflow
;
;===============================================================================
;
;bcdasc: CONVERT BCD DIGIT TO ASCII
;
;       -------------------------------------
;       Preparatory Ops: .A: BCD digit, $00-$99
;
```

```
;       Returned Values: .A: ASCII MSD
;                        .X: ASCII LSD
;                        .Y: entry value
;       ---------------------------------------
;
bcdasc:
        JSR     bintonyb        ;extract nybbles
        PHA                     ;save tens
        TXA
        ORA     #btoamask       ;change units to ASCII
        TAX                     ;store
        PLA                     ;get tens
        ORA     #btoamask       ;change to ASCII
        RTS
;
;=============================================================================
;
;bintonyb: EXTRACT BINARY NYBBLES
;
;       --------------------------------
;       Preparatory Ops: .A: binary value
;
;       Returned Values: .A: MSN
;                        .X: LSN
;                        .Y: entry value
;       --------------------------------
;
bintonyb:
        PHA                     ;save
        AND     #bcdumask       ;extract LSN
        TAX                     ;save it
        PLA
;         .rept s_bnybbl        ;extract MSN
        LSR
        LSR
        LSR
        LSR
;         .endr
        RTS
;
;=============================================================================
;
;binasc: CONVERT 32-BIT BINARY TO NULL-TERMINATED ASCII NUMBER STRING
;
;       ----------------------------------------------------
;       Preparatory Ops: FACA: 32-bit operand
;                         .A: radix character, w/bit 7 set to
;                             suppress radix symbol in the
;                             conversion string
;
```

```
;       Returned Values: ibuffer: conversion string
;                              .A: string length
;                              .X: string address LSB
;                              .Y: string address MSB
;
;       Execution Notes: ibufidx & instsize are overwritten.
;       ----------------------------------------------------
;
binasc:
        STZ     ibufidx         ;initialize string index
        STZ     instsize        ;clear format flag
;
;
;       evaluate radix...
;
        ASL                     ;extract format flag &...
        ROR     instsize        ;save it
        LSR                     ;extract radix character
        LDX     #n_radix-1      ;total radices
;
_0290010:
        CMP     radxtab,x       ;recognized radix?
        BEQ     _0290020        ;yes
;
        DEX
        BPL     _0290010        ;try next
;
        INX                     ;assume hex
;
_0290020:
        STX     radix           ;save radix index for later
        BIT     instsize
        BMI     _0290030        ;no radix symbol wanted
;
        LDA     radxtab,x       ;radix table
        STA     ibuffer         ;prepend to string
        INC     ibufidx         ;bump string index
;
_0290030:
        CMP     #c_dec          ;converting to decimal?
        BNE     _0290040        ;no
;
        JSR     facabcd         ;convert operand to BCD
        LDA     #0
        BRA     _0290070        ;skip binary stuff
;
;
;       prepare for binary, octal or hex conversion...
;
_0290040:
```

```
        LDX     #0              ;operand index
        LDY     #s_sfac-1       ;workspace index
;
_0290050:
        LDA     faca,x          ;copy operand to...
        STA     facb,y          ;workspace in...
        DEY                     ;big-endian order
        INX
        CPX     #s_pfac
        BNE     _0290050
;
        LDA     #0
        TYX
;
_0290060:
        STA     facb,x          ;pad workspace
        DEX
        BPL     _0290060
;
;
;       set up conversion parameters...
;
_0290070:
        STA     facc            ;initialize byte counter
        LDY     radix           ;radix index
        LDA     numstab,y       ;numerals in string
        STA     facc+s_byte     ;set remaining numeral count
        LDA     bitsntab,y      ;bits per numeral
        STA     facc+s_word     ;set
        LDA     lzsttab,y       ;leading zero threshold
        STA     facc+s_xword    ;set
;
;
;       generate conversion string...
;
_0290080:
        LDA     #0
        LDY     facc+s_word     ;bits per numeral
;
_0290090:
        LDX     #s_sfac-1       ;workspace size
        CLC                     ;avoid starting carry
;
_0290100:
        ROL     facb,x          ;shift out a bit...
        DEX                     ;from the operand or...
        BPL     _0290100        ;BCD conversion result
;
        ROL                     ;bit to .A
        DEY
```

```
        BNE     _0290090        ;more bits to grab
;
        TAY                     ;if numeral isn't zero...
        BNE     _0290110        ;skip leading zero tests
;
        LDX     facc+s_byte     ;remaining numerals
        CPX     facc+s_xword    ;leading zero threshold
        BCC     _0290110        ;below it, must convert
;
        LDX     facc            ;processed byte count
        BEQ     _0290130        ;discard leading zero
;
_0290110:
        CMP     #10             ;check range
        BCC     _0290120        ;is 0-9
;
        ADC     #a_hexdec       ;apply hex adjust
;
_0290120:
        ADC     #'0'            ;change to ASCII
        LDY     ibufidx         ;string index
        STA     ibuffer,y       ;save numeral in buffer
        INC     ibufidx         ;next buffer position
        INC     facc            ;bytes=bytes+1
;
_0290130:
        DEC     facc+s_byte     ;numerals=numerals-1
        BNE     _0290080        ;not done
;
;
;       terminate string & exit...
;
        LDX     ibufidx         ;printable string length
        STZ     ibuffer,x       ;terminate string
        TXA
        LDX     #<ibuffer       ;converted string
        LDY     #>ibuffer
        CLC                     ;all okay
        RTS
;
;===============================================================================
;
;binhex: CONVERT BINARY BYTE TO HEX ASCII CHARS
;
;       ----------------------------------------
;       Preparatory Ops: .A: byte to convert
;
;       Returned Values: .A: MSN ASCII char
;                        .X: LSN ASCII char
;                        .Y: entry value
```

```
;       ----------------------------------------
;
binhex:
        JSR     bintonyb        ;generate binary values
        PHA                     ;save MSN
        TXA
        JSR     _0300010        ;generate ASCII LSN
        TAX                     ;save
        PLA                     ;get input
;
;
;       convert nybble to hex ASCII equivalent...
;
_0300010:
        CMP     #10
        BCC     _0300020        ;in decimal range
;
        ADC     #k_hex          ;hex compensate
;
_0300020:
        EOR     #'0'            ;finalize nybble
        RTS                     ;done
;
;===============================================================================
;
;clrfaca: CLEAR FLOATING ACCUMULATOR A
;
clrfaca:
        PHP
        SLONGA
        STZ     faca
        STZ     faca+s_word
        PLP
        RTS
;
;===============================================================================
;
;clrfacb: CLEAR FLOATING ACCUMULATOR B
;
clrfacb:
        PHP
        SLONGA
        STZ     facb
        STZ     facb+s_word
        PLP
        RTS
;
;===============================================================================
;
;facabcd: CONVERT FACA INTO BCD
```

```
;
facabcd:
        LDX     #s_pfac-1       ;primary accumulator size -1
;
_1300010:
        LDA     faca,x          ;value to be converted
        PHA                     ;preserve
        DEX
        BPL     _1300010        ;next
;
        LDX     #s_sfac-1       ;workspace size
;
_1300020:
        STZ     facb,x          ;clear final result
        STZ     facc,x          ;clear scratchpad
        DEX
        BPL     _1300020
;
        INC     facc+s_sfac-s_byte
        SED                     ;select decimal mode
        LDY     #m_bits-1       ;bits to convert -1
;
_1300030:
        LDX     #s_pfac-1       ;operand size
        CLC                     ;no carry at start
;
_1300040:
        ROR     faca,x          ;grab LS bit in operand
        DEX
        BPL     _1300040
;
        BCC     _1300060        ;LS bit clear
;
        CLC
        LDX     #s_sfac-1
;
_1300050:
        LDA     facb,x          ;partial result
        ADC     facc,x          ;scratchpad
        STA     facb,x          ;new partial result
        DEX
        BPL     _1300050
;
        CLC
;
_1300060:
        LDX     #s_sfac-1
;
_1300070:
        LDA     facc,x          ;scratchpad
```

```
        ADC     facc,x          ;double &...
        STA     facc,x          ;save
        DEX
        BPL     _1300070
;
        DEY
        BPL     _1300030        ;next operand bit
;
        CLD
        LDX     #0
        LDY     #s_pfac
;
_1300080:
        PLA                     ;operand
        STA     faca,x          ;restore
        INX
        DEY
        BNE     _1300080        ;next
;
        RTS
;
;=============================================================================
;
;nybtobin: CONVERT ASCII NYBBLE TO BINARY
;
nybtobin:
        JSR     toupper         ;convert case if necessary
        SEC
        SBC     #'0'            ;change to binary
        BCC     _0310020        ;not a numeral - error
;
        CMP     #10
        BCC     _0310010        ;numeral is 0-9
;
        SBC     #a_hexdec+1     ;10-15 --> A-F
        CLC                     ;no conversion error
;
_0310010:
        RTS
;
_0310020:
        SEC                     ;conversion error
        RTS
;
;=============================================================================
;
;calccnt: COMPUTE BYTE COUNT FROM ADDRESS RANGE
;
calccnt:
        JSR     clrfacb         ;clear accumulator
```

```
        SLONGA
        SEC
        LDA     addrb           ;ending address
        SBC     addra           ;starting address
        STA     facb            ;byte count
        SHORTA
        LDA     addrb+s_word    ;handle banks
        SBC     addra+s_word
        STA     facb+s_word
        RTS
;
;===============================================================================
;
;clroper: CLEAR OPERAND
;
clroper:
        PHX
        LDX     #s_oper-1
;
_0320010:
        STZ     operand,x
        DEX
        BPL     _0320010
;
        STZ     eopsize
        PLX
        RTS
;
;===============================================================================
;
;cpfwsup: FOWARD COPY MEMORY SETUP
;
cpfwsup:
        LONGR
        LDXW    opc_mvn         ;"move next" opcode
        BRA     cpsup
;
;===============================================================================
;
;cprvsup: REVERSE COPY MEMORY SETUP
;
cprvsup:
        LONGR
        LDXW    opc_mvp         ;"move previous" opcode
;
;===============================================================================
;
;cpsup: COPY MEMORY SETUP
;
cpsup:
```

```
        PHA                     ;save banks
        TXA                     ;protect...
        XBA                     ;opcode
        SHORTA
        LDXW    copylen
;
_1320010:
        LDALX   cpcode          ;transfer copy code to...
        STA     mcftwork,x      ;to workspace
        DEX
        BPL     _1320010
;
        XBA                     ;recover opcode &...
        STA     mcftopc         ;set it
        SLONGA
        PLA                     ;get banks &...
        STA     mcftbnk         ;set them
        RTS
;
;================================================================================
;
;decdcnt: DECREMENT DUMP COUNT
;
;       ----------------------------------------
;       Preparatory Ops: bytes to process in FACB
;                        bytes processed in CHARCNT
;
;       Returned Values: .A: used
;                        .X: entry value
;                        .Y: entry value
;                        .C: 1 = count = zero
;       ----------------------------------------
;
decdcnt:
        SHORTA
        LDA     #0
        XBA                     ;clear .B
        LDA     facb+s_word     ;count MSW
        SLONGA
        SEC
        ORA     facb            ;count LSW
        BEQ     _0330020        ;zero, just exit
;
        LDA     facb
        SBC     charcnt         ;bytes processed
        STA     facb
        SHORTA
        LDA     facb+s_word
        SBC     #0              ;handle borrow
        BCC     _0330010        ;underflow
```

```
;
        STA     facb+s_word
        CLC                     ;count > 0
        RTS
;
_0330010:
        SEC
;
_0330020:
        SHORTA
        RTS
;
;================================================================================
;
;enddest: GET 2ND & 3RD ADDRESSES FOR COMPARE & TRANSFER
;
enddest:
        JSR     facasize        ;check start...
        CMP     #s_dword        ;for range
        BCS     _0340010        ;out of range - error
;
        JSR     facaddra        ;store start
        JSR     getparm         ;get end
        BCS     _0340010        ;not entered - error
;
        JSR     facasize        ;check end...
        CMP     #s_dword        ;for range
        BCS     _0340010        ;out of range - error
;
        JSR     facaddrb        ;store end
        JSR     getparm         ;get destination
        BCS     _0340010        ;not entered - error
;
        JSR     facasize        ;check destination...
        CMP     #s_dword        ;for range
        BCC     facaoper        ;store dest address
;
_0340010:
        RTS                     ;exit w/error
;
;================================================================================
;
;facaddra: COPY FACA TO ADDRA
;
facaddra:
        LDX     #s_xword-1
;
_0350010:
        LDA     faca,x
        STA     addra,x
```

```
        DEX
        BPL     _0350010
;
        RTS
;
;===============================================================================
;
;facaddrb: COPY FACA TO ADDRB
;
facaddrb:
        LDX     #s_xword-1
;
_1350010:
        LDA     faca,x
        STA     addrb,x
        DEX
        BPL     _1350010
;
        RTS
;
;===============================================================================
;
;facaoper: COPY FACA TO OPERAND
;
facaoper:
        LDX     #s_oper-1
;
_0360010:
        LDA     faca,x
        STA     operand,x
        DEX
        BPL     _0360010
;
        RTS
;
;===============================================================================
;
;facasize: REPORT OPERAND SIZE IN FACA
;
;       ----------------------------------------
;       Preparatory Ops: operand in FACA
;
;       Returned Values: .A: s_byte  (1)
;                            s_word  (2)
;                            s_xword (3)
;                            s_dword (4)
;
;       Notes: 1) This function will always report
;                 a non-zero result.
;       ----------------------------------------
```

```
;
facasize:
        SHORTR
        LDX     #s_dword-1
;
_0370010:
        LDA     faca,x          ;get byte
        BNE     _0370020        ;done
;
        DEX
        BNE     _0370010        ;next byte
;
_0370020:
        INX                     ;count=index+1
        TXA
        RTS
;
;===============================================================================
;
;getparm: GET A PARAMETER
;
;       ------------------------------------------------
;       Preparatory Ops: null-terminated input in IBUFFER
;
;       Returned Values: .A: chars in converted parameter
;                        .X: used
;                        .Y: entry value
;                        .C: 1 = no parameter entered
;       ------------------------------------------------
;
getparmr:
        DEC     ibufidx         ;reread previous char
;
getparm:
        PHY                     ;preserve
        JSR     ascbin          ;convert parameter to binary
        BCS     _0380040        ;conversion error
;
        JSR     getcharr        ;reread last char
        BNE     _0380010        ;not end-of-input
;
        DEC     ibufidx         ;reindex to terminator
        LDA     charcnt         ;get chars processed so far
        BEQ     _0380030        ;none
;
        BNE     _0380020        ;some
;
_0380010:
        CMP     #a_blank        ;recognized delimiter
        BEQ     _0380020        ;end of parameter
```

```
;
        CMP     #','            ;recognized delimiter
        BNE     _0380040        ;unknown delimter
;
_0380020:
        CLC
        .BYTE   bitzp           ;skip SEC below
;
_0380030:
        SEC
        PLY                     ;restore
        LDA     charcnt         ;get count
        RTS                     ;done
;
_0380040:                       ;.rept 3               ;clean up stack
        PLA
        PLA
        PLA
;.endr
        JMP     monerr          ;abort w/error
;
;================================================================================
;
;nxtaddra: TEST & INCREMENT WORKING ADDRESS 'A'
;
;       ------------------------------------------------
;       Calling syntax: JSR NXTADDRA
;
;       Exit registers: .A: used
;                       .B: used
;                       .X: entry value
;                       .Y: entry value
;                      DB: entry value
;                      DP: entry value
;                      PB: entry value
;                      SR: NVmxDIZC
;                          ||||||||
;                          |||||||+---> 0: ADDRA < ADDRB
;                          |||||||      1: ADDRA >= ADDRB
;                          ||||||+----> undefined
;                          |||+++-----> entry value
;                          ||+--------> 1
;                          ++---------> undefined
;       ------------------------------------------------
;
nxtaddra:
        SHORTA
        LDA     addra+s_word    ;bits 16-23
        CMP     addrb+s_word
        BCC     incaddra        ;increment
```

```
;
        BNE     _0390010        ;don't increment
;
        SLONGA
        LDA     addra           ;bits 0-15
        CMP     addrb           ;condition flags
        SHORTA
        BCC     incaddra        ;increment
;
_0390010:
        RTS
;
;================================================================================
;
;getbyte: GET A BYTE FROM MEMORY
;
getbyte:
        LDAIL   addra           ;get a byte
;
;================================================================================
;
;incaddra: INCREMENT WORKING ADDRESS 'A'
;
;       ------------------------------------------------
;       Calling syntax: JSR INCADDRA
;
;       Exit registers: .A: entry value
;                       .B: entry value
;                       .X: entry value
;                       .Y: entry value
;                      DB: entry value
;                      DP: entry value
;                      PB: entry value
;                      SR: NVmxDIZC
;                          ||||||||
;                          ++++++++---> entry value
;       ------------------------------------------------
;
incaddra:
        PHP
        SLONGA
        INC     addra           ;bump bits 0-15
        BNE     _0400010
;
        SHORTA
        INC     addra+s_word    ;bump bits 16-23
;
_0400010:
        PLP
        RTS
```

```
;
;================================================================================
;
;incoper: INCREMENT OPERAND ADDRESS
;
incoper:
        CLC
        PHP
        LONGR
        PHA
        INC     operand         ;handle base address
        BNE     _0410010
;
        SHORTA
        INC     operand+s_word  ;handle bank
        SLONGA
;
_0410010:
        PLA
        PLP
        RTS
;
;================================================================================
;
;instdata: GET INSTRUCTION SIZE & ADDRESSING MODE DATA
;
;       ---------------------------------
;       Preparatory Ops: .X: 65C816 opcode
;
;       Returned Values: .A: mode flags
;                        .X: operand size
;                        .Y: mode index
;       ---------------------------------
;
instdata:
        SHORTR
        LDA     mnetabam,x      ;addressing mode data
        PHA                     ;save mode flag bits
        PHA                     ;save size data
        AND     #amodmask       ;extract mode index &...
        TAY                     ;save
        PLA                     ;recover data
        AND     #opsmask        ;mask mode fields &...
;         .rept n_opslsr        ;extract operand size
        LSR
        LSR
        LSR
        LSR
;         .endr
        TAX                     ;operand size
```

```
        PLA                     ;recover mode flags
        AND     #vopsmask       ;discard mode & size fields
        RTS
;
;================================================================================
;
;offtarg: CONVERT BRANCH OFFSET TO TARGET ADDRESS
;
;       ------------------------------------------------
;       Preparatory Ops:    ADDRA: base address
;                         INSTSIZE: instruction size
;                          OPERAND: offset
;
;       Returned Values:  OPERAND: target address (L/H)
;                              .A: used
;                              .X: entry value
;                              .Y: entry value
;       ------------------------------------------------
;
offtarg:
        SLONGA
        LDA     addra           ;base address
        SHORTA
        LSR     instsize        ;bit 0 will be set if...
        BCS     _0420010        ;a long branch
;
        BIT     operand         ;short forward or backward?
        BPL     _0420010        ;forward
;
        XBA                     ;expose address MSB
        DEA                     ;back a page
        XBA                     ;expose address LSB
;
_0420010:
        SLONGA
        CLC
        ADC     operand         ;calculate target address
        STA     operand         ;new operand
        SHORTA
        LDA     #s_xword
        STA     instsize        ;effective instruction size
        RTS
;
;================================================================================
;
;setxaddr: SET EXECUTION ADDRESS
;
setxaddr:
        BCS     _0430010        ;no address given
;
```

```
        JSR     facasize        ;check address...
        CMP     #s_dword        ;range
        BCS     _0430020        ;out of range
;
        SLONGA
        LDA     faca            ;execution address
        STA     reg_pcx         ;set new PC value
        SHORTA
        LDA     faca+s_word
        STA     reg_pbx         ;set new PB value
;
_0430010:
        CLC                     ;no error
;
_0430020:
        RTS
;
;================================================================================
;
;targoff: CONVERT BRANCH TARGET ADDRESS TO BRANCH OFFSET
;
;       ------------------------------------------------
;       Preparatory Ops:   ADDRA: instruction address
;                        OPERAND: target address
;
;       Returned Values: OPERAND: computed offset
;                              .A: effective operand size
;                              .X: entry value
;                              .Y: entry value
;                              .C: 1 = branch out of range
;
;       Execution notes: ADDRB is set to the branch base
;                        address.
;       ------------------------------------------------
;
targoff:
        STZ     instsize+s_byte ;always zero
        LDA     instsize        ;instruction size will tell...
        LSR                     ;if long or short branch
;
;-------------------------------------------------
_btype          = facc+5        ;branch type flag
;-------------------------------------------------
;
        ROR     _btype          ;set branch type...
;
;       x0000000
;       |
;       +----------> 0: short
;                    1: long
```

```
;
        SLONGA
        CLC
        LDA     addra           ;instruction address
        ADC     instsize        ;instruction size
        STA     addrb           ;base address
        SEC
        LDA     operand         ;target address
        SBC     addrb           ;base address
        STA     operand         ;offset
        SHORTA
        BCC     _0440040        ;backward branch
;
        BIT     _btype          ;check branch range
        BMI     _0440020        ;long
;
;
;       process short forward branch...
;
        XBA                     ;offset MSB should be zero
        BNE     _0440060        ;it isn't - out of range
;
        XBA                     ;offset LSB should be $00-$7F
        BMI     _0440060        ;it isn't - out of range
;
_0440010:
        LDA     #s_byte         ;final instruction size
        CLC                     ;branch in range
        RTS
;
;
;       process long forward branch...
;
_0440020:
        XBA                     ;offset MSB should be positive
        BMI     _0440060        ;it isn't - branch out of range
;
_0440030:
        LDA     #s_word
        CLC
        RTS
;
;
;       process backward branch...
;
_0440040:
        BIT     _btype          ;long or short?
        BMI     _0440050        ;long
;
;
```

```
;       process short backward branch...
;
        XBA                     ;offset MSB should be negative
        BPL     _0440060        ;it isn't - out of range
;
        EOR     #%11111111      ;complement offset MSB 2s
        BNE     _0440060        ;out of range
;
        XBA                     ;offset LSB should be $80-$FF
        BMI     _0440010        ;it is - branch in range
;
        BRA     _0440060        ;branch out of range
;
;
;       process long backward branch...
;
_0440050:
        XBA                     ;offset MSB should be negative
        BMI     _0440030        ;it is - branch in range
;
_0440060:
        SEC                     ;range error
        RTS
;
;===============================================================================
;
;getcharr: GET PREVIOUS INPUT BUFFER CHARACTER
;
getcharr:
        DEC     ibufidx         ;move back a char
;
;===============================================================================
;
;getchar: GET A CHARACTER FROM INPUT BUFFER
;
;       ---------------------------------------------
;       Preparatory Ops : none
;
;       Register Returns: .A: character or <NUL>
;                         .B: entry value
;                         .X: entry value
;                         .Y: entry value
;
;       MPU Flags: NVmxDIZC
;                  ||||||||
;                  |||||||+---> entry value
;                  ||||||+----> 1: <NUL> gotten
;                  |||||+-----> entry value
;                  ||||+------> entry value
;                  |||+-------> entry value
```

```
;                  ||+--------> entry value
;                  |+---------> not defined
;                  +----------> not defined
;       ------------------------------------------
;
getchar:
        PHX
        PHY
        PHP                     ;save register sizes
        SHORTR                  ;force 8 bits
        LDX     ibufidx         ;buffer index
        LDA     ibuffer,x       ;get char
        INC     ibufidx         ;bump index
        PLP                     ;restore register widths
        PLY
        PLX
        XBA                     ;condition...
        XBA                     ;.Z
        RTS
;
;================================================================================
;
;getpat: GET PATTERN FOR MEMORY CHANGE or SEARCH
;
;       ------------------------------------------------------
;       Preparatory Ops: Null-terminated pattern in IBUFFER.
;
;       Returned Values: .A: used
;                        .X: used
;                        .Y: pattern length if entered
;                        .C: 0 = pattern valid
;                            1 = exception:
;                        .N  0 = no pattern entered
;                            1 = evaluation error
;
;       Notes: 1) If pattern is preceded by "'" the following
;                 characters are interpreted as ASCII.
;              2) A maximum of 32 bytes or characters is
;                 accepted.  Excess input will be discarded.
;       ------------------------------------------------------
;
getpat:
        STZ     status          ;clear pattern type indicator
        LDY     #0              ;pattern index
        JSR     getcharr        ;get last char
        BEQ     _0450070        ;EOS
;
        LDX     ibufidx         ;current buffer index
        JSR     getcharw        ;get next
        BEQ     _0450070        ;EOS
```

```
;
        CMP     #$27            ; single quote
        BNE     _0450010        ;not ASCII input
;
        ROR     status          ;condition flag
        BRA     _0450030        ;balance of input is ASCII
;
_0450010:
        STX     ibufidx         ;restore buffer index
;
_0450020:
        JSR     getparm         ;evaluate numeric pattern
        BCS     _0450060        ;done w/pattern
;
        JSR     facasize        ;size
        CMP     #s_word
        BCS     _0450070        ;not a byte - error
;
        LDA     faca            ;get byte &...
        BRA     _0450040        ;store
;
_0450030:
        JSR     getchar         ;get ASCII char
        BEQ     _0450060        ;done w/pattern
;
_0450040:
        CPY     #s_auxbuf       ;pattern buffer full?
        BEQ     _0450050        ;yes
;
        STA     auxbuf,y        ;store pattern
        INY
        BIT     status
        BPL     _0450020        ;get next numeric value
;
        BRA     _0450030        ;get next ASCII char
;
_0450050:
        JSR     alert           ;excess input
;
_0450060:
        STY     auxbufix        ;save pattern size
        TYA                     ;condition .Z
        CLC                     ;pattern valid
        RTS
;
;
;       no pattern entered...
;
_0450070:
        REP     #%10000000
```

```
        SEC
        RTS
;
;
;       evaluation error...
;
_0450080:
        SEP     #%10000001
        RTS
;
;================================================================================
;
;getcharw: GET FROM INPUT BUFFER, DISCARDING WHITESPACE
;
;       ------------------------------------------------
;       Preparatory Ops: Null-terminated input in IBUFFER.
;
;       Returned Values: .A: char or null
;                        .X: entry value
;                        .Y: entry value
;                        .Z: 1 = null terminator detected
;
;       Notes: Whitespace is defined as a blank ($20) or a
;              horizontal tab ($09).
;       ------------------------------------------------
;
getcharw:
        JSR     getchar         ;get from buffer
        BEQ     _0460010        ;EOI
;
        CMP     #a_blank
        BEQ     getcharw        ;discard whitespace
;
        CMP     #a_ht           ;also whitespace
        BEQ     getcharw
;
_0460010:
        CLC
        RTS
;
;================================================================================
;
;input: INTERACTIVE INPUT FROM CONSOLE CHANNEL
;
;       ------------------------------------------------------------
;       Preparatory Ops: Zero IBUFIDX or load IBUFFER with default
;                        input & set IBUFIDX to the number of chars
;                        loaded into the buffer.
;
;       Returned Values: .A: used
```

```
;                       .X: characters entered
;                       .Y: used
;
;       Example: STZ IBUFIDX
;                JSR INPUT
;
;       Notes: Input is collected in IBUFFER & is null-terminated.
;              IBUFIDX is reset to zero upon exit.
;       ------------------------------------------------------------
;
input:
        LDX     ibufidx
        STZ     ibuffer,x       ;be sure buffer is terminated
        JSR     dpyibuf         ;print default input if any

        LDX     ibufidx         ;starting buffer index
;
;
;       main input loop...
;
_0470010:
        JSR     CURSOR
_047001A:
        JSR     getcha          ;poll for input
        BCC     _0470020        ;got something
;
;         wai                   ;wait 'til any IRQ &...
        BRA     _047001A        ;try again
;
_0470020:
        CMP     #a_del          ;above ASCII range?
        BCS     _047001A        ;yes, ignore

        JSR     UNCURSOR
;
        CMP     #a_ht           ;horizontal tab?
        BNE     _0470030        ;no
;
        LDA     #a_blank        ;replace <HT> w/blank
;
_0470030:
        CMP     #a_blank        ;control char?
        BCC     _0470050        ;yes
;
;
;       process QWERTY character...
;
        CPX     #s_ibuf         ;room in buffer?
        BCS     _0470040        ;no
;
```

```
        STA     ibuffer,x       ;store char
        INX                     ;bump index
        .BYTE   bitabs          ;echo char
;
_0470040:
        LDA     #a_bel          ;alert user
        JSR     putcha
        BRA     _0470010        ;get some more
;
;
;       process carriage return...
;
_0470050:
        CMP     #a_cr           ;carriage return?
        BNE     _0470060        ;no
;
;         phx                   ;protect input count
;         pea dc_co
;         jsr sprint            ;cursor off
;         plx                   ;recover input count
        STZ     ibuffer,x       ;terminate input &...
        STZ     ibufidx         ;reset buffer index
        RTS                     ;done
;
;
;       process backspace...
;
_0470060:
        CMP     #a_bs           ;backspace?
        BNE     _0470010        ;no
;
        TXA
        BEQ     _0470010        ;no input, ignore <BS>
;
        DEX                     ;1 less char
        PHX                     ;preserve count
        JSR     gendbs          ;destructive backspace
        PLX                     ;restore count
        BRA     _0470010        ;get more input
;
;===============================================================================
;
;lodbnk: LOAD SOURCE & DESTINATION BANKS
;
lodbnk:
        SHORTA
        LDA     operand+s_word  ;destination bank
        XBA                     ;make it MSB
        LDA     addra+s_word    ;source bank is LSB
        RTS
```

```
;
;================================================================================
;
;getcharc: GET A CHARACTER FROM INPUT BUFFER & CONVERT CASE
;
;       ------------------------------------------------
;       Preparatory Ops: Null-terminated input in IBUFFER.
;
;       Returned Values: .A: char or null
;                        .X: entry value
;                        .Y: entry value
;                        .Z: 1 = null terminator detected
;       ------------------------------------------------
;
getcharc:
        JSR     getchar         ;get from buffer
;
;================================================================================
;
;toupper: FORCE CHARACTER TO UPPER CASE
;
;       ------------------------------------------------
;       Preparatory Ops : .A: 8 bit character to convert
;
;       Register Returns: .A: converted character
;                         .B: entry value
;                         .X: entry value
;                         .Y: entry value
;
;       MPU Flags: no change
;
;       Notes: 1) This subroutine has no effect on char-
;                 acters that are not alpha.
;       ------------------------------------------------
;
toupper:
        PHP                     ;protect flags
        CMP     #a_asclcl       ;check char range
        BCC     _0480010        ;not LC alpha
;
        CMP     #a_asclch+s_byte
        BCS     _0480010        ;not LC alpha
;
        AND     #a_lctouc       ;force to UC
;
_0480010:
        PLP                     ;restore flags
;
touppera:
        RTS
```

```
;
;================================================================================
;
;teststop: TEST FOR STOP KEY
;
;       ---------------------------------------------
;       Preparatory Ops: none
;
;       Returned Values: .A: detected keypress, if any
;                        .X: entry value
;                        .Y: entry value
;
;       MPU Flags: NVmxDIZC
;                  ||||||||
;                  |||||||+---> 0: normal key detected
;                  |||||||      1: <STOP> detected
;                  +++++++----> not defined
;
;       Example: jsr teststop
;                bcs stopped
;
;       Notes: The symbol STOPKEY defines the ASCII
;              value of the "stop key."
;       ---------------------------------------------
;
teststop:
        JSR     getcha          ;poll console
        BCS     _0490010        ;no input
;
        CMP     #stopkey        ;stop key pressed?
        BEQ     _0490020        ;yes
;
_0490010:
        CLC
;
_0490020:
        RTS


;__LOAD_____________________________________________________________________
; LOAD A MOTOROLA FORMATTED HEX FILE (S28)
;
;____________________________________________________________________________
LOADS19:
        PHP
        SHORTR
        PEA     mm_S19_prmpt
        JSR     sprint          ;display input prompt
```

```
LOADS19_1:
        JSR     getc            ;
        CMP     #'S'            ;
        BNE     LOADS19_1       ; FIRST CHAR NOT (S)
        JSR     getc            ; READ CHAR
        CMP     #'8'            ;
        BEQ     LOAD21          ;
        CMP     #'2'            ;
        BNE     LOADS19_1       ; SECOND CHAR NOT (2)
        LDA     #$00            ;
        STA     faca            ; ZERO CHECKSUM

        JSR     GETBYTE         ; READ BYTE
        SBC     #$02            ;
        STA     facb            ; BYTE COUNT
; BUILD ADDRESS
        JSR     GETBYTE         ; READ 2 FRAMES
        STA     addra+2         ;
        JSR     GETBYTE         ; READ 2 FRAMES
        STA     addra+1         ;
        JSR     GETBYTE         ;
        STA     addra           ;

        LDY     #$00            ;
LOAD11:
        JSR     GETBYTE         ;
        DEC     facb            ;
        BEQ     LOAD15          ; ZERO BYTE COUNT
        STA     [addra],Y       ; STORE DATA
        SLONGA
        INC     addra           ;
        CMPW    $0000
        BNE     LOAD11A
        SHORTA
        INC     addra+2         ;
LOAD11A:
        SHORTA
        JMP     LOAD11          ;

LOAD15:
        INC     faca            ;
        BEQ     LOADS19_1       ;
LOAD19:
        LDA     #'?'            ;
        JSR     putcha          ;
LOAD21:
        PLP
        JMP     monce           ;back to executive
GETBYTE:
        JSR     INHEX           ; GET HEX CHAR
```

```
        ASL     A               ;
        ASL     A               ;
        ASL     A               ;
        ASL     A               ;
        STA     numeral         ;
        JSR     INHEX           ;
        AND     #$0F            ; MASK TO 4 BITS
        ORA     numeral         ;
        PHA                     ;
        CLC                     ;
        ADC     faca            ;
        STA     faca            ;
        PLA                     ;
        RTS                     ;
; INPUT HEX CHAR
INHEX:
        JSR     getc            ;
        CMP     #$3A            ; LESS THAN 9?
        BCS     INHEX_BIG       ; NO, SKIP NEXT
        SBC     #$2F            ; CONVERT 0-9
INHEX_BIG:
        CMP     #$41            ; A OR MORE?
        BCC     INHEX_SMALL     ; NO, SKIP NEXT
        SBC     #$37            ; CONVERT A-F
INHEX_SMALL:
        RTS                     ;
getc:
        JSR     getcha          ;poll for input
        BCC     getcd           ;got something
        BRA     getc            ;try again
getcd:
        PHA                     ;
        JSR     putcha          ;
        PLA                     ;
        RTS



;
;cpcode: COPY MEMORY CODE
;
;       ----------------------------------------
;       This code is transfered to workspace when a
;       copy or fill operation is to be performed.
;       ----------------------------------------
;
cpcode:
        PHB                     ;must preserve data bank
;.rept s_mvinst
        NOP                     ;placeholder
```

```
        NOP                     ;placeholder
        NOP                     ;placeholder
;.endr
        PLB                     ;restore data bank
        JML     monce           ;return to command executive
cpcodeee:                       ;placeholder - do not delete
copylen         = cpcodeee-cpcode-1
;
;================================================================================
;
;COMMAND PROCESSING DATA TABLES
;
;
;       monitor commands...
;
mpctab:
        .BYTE   "A"             ;assemble code
        .BYTE   "C"             ;compare memory ranges
        .BYTE   "D"             ;disassemble code
        .BYTE   "F"             ;fill memory
        .BYTE   "G"             ;execute code
        .BYTE   "H"             ;search memory
        .BYTE   "J"             ;execute code as subroutine
        .BYTE   "L"             ;load S19 file
        .BYTE   "M"             ;dump memory range
        .BYTE   "R"             ;dump registers
        .BYTE   "T"             ;copy memory range
;         .BYTE "X"             ;exit from monitor
        .BYTE   ">"             ;change memory
        .BYTE   ";"             ;change registers
n_mpctab        = *-mpctab      ;entries in above table
;
;
;       monitor command jump table...
;
mpcextab:
        .WORD   monasc-s_byte   ; A  assemble code
        .WORD   moncmp-s_byte   ; C  compare memory ranges
        .WORD   mondsc-s_byte   ; D  disassemble code
        .WORD   monfil-s_byte   ; F  fill memory
        .WORD   monjmp-s_byte   ; G  execute code
        .WORD   monhnt-s_byte   ; H  search memory
        .WORD   monjsr-s_byte   ; J  execute code as subroutine
        .WORD   LOADS19-s_byte  ; L  Load S19 File
        .WORD   mondmp-s_byte   ; M  dump memory range
        .WORD   monreg-s_byte   ; R  dump registers
        .WORD   moncpy-s_byte   ; T  copy memory range
;         .WORD monxit-s_byte   ; X  exit from monitor
        .WORD   monchm-s_byte   ; >  change memory
        .WORD   monchr-s_byte   ; ;  change registers
```

```
;
;
;       number conversion...
;
basetab:
        .BYTE   16,10,8,2       ;supported number bases
bitsdtab:
        .BYTE   4,3,3,1         ;bits per binary digit
bitsntab:
        .BYTE   4,4,3,1         ;bits per ASCII character
lzsttab:
        .BYTE   3,2,9,2         ;leading zero suppression thresholds
numstab:
        .BYTE   12,12,16,48     ;bin to ASCII conversion numerals
radxtab:
        .BYTE   c_hex           ;hexadecimal radix
        .BYTE   c_dec           ;decimal radix
        .BYTE   c_oct           ;octal radix
        .BYTE   c_bin           ;binary radix
n_radix         = *-radxtab     ;number of recognized radices
;
;
;       shadow MPU register sizes...
;
rcvltab:
        .BYTE   s_mpupbx+s_byte ; PB
        .BYTE   s_mpupcx+s_byte ; PC
        .BYTE   s_mpusrx+s_byte ; SR
        .BYTE   s_word+s_byte   ; .C
        .BYTE   s_word+s_byte   ; .X
        .BYTE   s_word+s_byte   ; .Y
        .BYTE   s_mpuspx+s_byte ; SP
        .BYTE   s_mpudpx+s_byte ; DP
        .BYTE   s_mpudbx+s_byte ; DB
n_regchv        = *-rcvltab     ;total shadow registers
;
;===============================================================================
;
;ASSEMBLER/DISASSEMBLER DATA TABLES
;
;
;       numerically sorted & encoded W65C816S mnemonics...
;
mnetab:
        .WORD   mne_xba         ;  0 - XBA
        .WORD   mne_lda         ;  1 - LDA
        .WORD   mne_pea         ;  2 - PEA
        .WORD   mne_pha         ;  3 - PHA
        .WORD   mne_pla         ;  4 - PLA
        .WORD   mne_bra         ;  5 - BRA
```

```
.WORD   mne_ora         ;  6 - ORA
.WORD   mne_sta         ;  7 - STA
.WORD   mne_txa         ;  8 - TXA
.WORD   mne_tya         ;  9 - TYA
.WORD   mne_phb         ; 10 - PHB
.WORD   mne_plb         ; 11 - PLB
.WORD   mne_trb         ; 12 - TRB
.WORD   mne_tsb         ; 13 - TSB
.WORD   mne_sbc         ; 14 - SBC
.WORD   mne_bcc         ; 15 - BCC
.WORD   mne_adc         ; 16 - ADC
.WORD   mne_tdc         ; 17 - TDC
.WORD   mne_dec         ; 18 - DEC
.WORD   mne_sec         ; 19 - SEC
.WORD   mne_clc         ; 20 - CLC
.WORD   mne_inc         ; 21 - INC
.WORD   mne_tsc         ; 22 - TSC
.WORD   mne_bvc         ; 23 - BVC
.WORD   mne_tcd         ; 24 - TCD
.WORD   mne_sed         ; 25 - SED
.WORD   mne_phd         ; 26 - PHD
.WORD   mne_cld         ; 27 - CLD
.WORD   mne_pld         ; 28 - PLD
.WORD   mne_and         ; 29 - AND
.WORD   mne_xce         ; 30 - XCE
.WORD   mne_bne         ; 31 - BNE
.WORD   mne_wai         ; 32 - WAI
.WORD   mne_pei         ; 33 - PEI
.WORD   mne_sei         ; 34 - SEI
.WORD   mne_cli         ; 35 - CLI
.WORD   mne_bmi         ; 36 - BMI
.WORD   mne_rti         ; 37 - RTI
.WORD   mne_phk         ; 38 - PHK
.WORD   mne_brk         ; 39 - BRK
.WORD   mne_jml         ; 40 - JML
.WORD   mne_rol         ; 41 - ROL
.WORD   mne_bpl         ; 42 - BPL
.WORD   mne_brl         ; 43 - BRL
.WORD   mne_asl         ; 44 - ASL
.WORD   mne_jsl         ; 45 - JSL
.WORD   mne_rtl         ; 46 - RTL
.WORD   mne_wdm         ; 47 - WDM
.WORD   mne_mvn         ; 48 - MVN
.WORD   mne_rep         ; 49 - REP
.WORD   mne_sep         ; 50 - SEP
.WORD   mne_php         ; 51 - PHP
.WORD   mne_plp         ; 52 - PLP
.WORD   mne_cmp         ; 53 - CMP
.WORD   mne_jmp         ; 54 - JMP
.WORD   mne_cop         ; 55 - COP
```

```
        .WORD   mne_nop         ; 56 - NOP
        .WORD   mne_stp         ; 57 - STP
        .WORD   mne_mvp         ; 58 - MVP
        .WORD   mne_beq         ; 59 - BEQ
        .WORD   mne_per         ; 60 - PER
        .WORD   mne_eor         ; 61 - EOR
        .WORD   mne_ror         ; 62 - ROR
        .WORD   mne_jsr         ; 63 - JSR
        .WORD   mne_lsr         ; 64 - LSR
        .WORD   mne_bcs         ; 65 - BCS
        .WORD   mne_tcs         ; 66 - TCS
        .WORD   mne_rts         ; 67 - RTS
        .WORD   mne_bvs         ; 68 - BVS
        .WORD   mne_txs         ; 69 - TXS
        .WORD   mne_bit         ; 70 - BIT
        .WORD   mne_clv         ; 71 - CLV
        .WORD   mne_tax         ; 72 - TAX
        .WORD   mne_ldx         ; 73 - LDX
        .WORD   mne_dex         ; 74 - DEX
        .WORD   mne_phx         ; 75 - PHX
        .WORD   mne_plx         ; 76 - PLX
        .WORD   mne_inx         ; 77 - INX
        .WORD   mne_cpx         ; 78 - CPX
        .WORD   mne_tsx         ; 79 - TSX
        .WORD   mne_stx         ; 80 - STX
        .WORD   mne_tyx         ; 81 - TYX
        .WORD   mne_tay         ; 82 - TAY
        .WORD   mne_ldy         ; 83 - LDY
        .WORD   mne_dey         ; 84 - DEY
        .WORD   mne_phy         ; 85 - PHY
        .WORD   mne_ply         ; 86 - PLY
        .WORD   mne_iny         ; 87 - INY
        .WORD   mne_cpy         ; 88 - CPY
        .WORD   mne_sty         ; 89 - STY
        .WORD   mne_txy         ; 90 - TXY
        .WORD   mne_stz         ; 91 - STZ
;
s_mnetab        = *-mnetab      ;mnemonic table size
n_mnemon        = s_mnetab/s_word;total mnemonics
;
;
;       mnemonic lookup indices in opcode order...
;
mnetabix:
        .BYTE   mne_brkx        ; $00  BRK
        .BYTE   mne_orax        ; $01  ORA (dp,X)
        .BYTE   mne_copx        ; $02  COP
        .BYTE   mne_orax        ; $03  ORA <offset>,S
        .BYTE   mne_tsbx        ; $04  TSB dp
        .BYTE   mne_orax        ; $05  ORA dp
```

```
        .BYTE   mne_aslx        ; $06  ASL dp
        .BYTE   mne_orax        ; $07  ORA [dp]
        .BYTE   mne_phpx        ; $08  PHP
        .BYTE   mne_orax        ; $09  ORA #
        .BYTE   mne_aslx        ; $0A  ASL A
        .BYTE   mne_phdx        ; $0B  PHD
        .BYTE   mne_tsbx        ; $0C  TSB abs
        .BYTE   mne_orax        ; $0D  ORA abs
        .BYTE   mne_aslx        ; $0E  ASL abs
        .BYTE   mne_orax        ; $0F  ORA absl
;
        .BYTE   mne_bplx        ; $10  BPL abs
        .BYTE   mne_orax        ; $11  ORA (<dp>),Y
        .BYTE   mne_orax        ; $12  ORA (dp)
        .BYTE   mne_orax        ; $13  ORA (<offset>,S),Y
        .BYTE   mne_trbx        ; $14  TRB dp
        .BYTE   mne_orax        ; $15  ORA dp,X
        .BYTE   mne_aslx        ; $16  ASL dp,X
        .BYTE   mne_orax        ; $17  ORA [dp],Y
        .BYTE   mne_clcx        ; $18  CLC
        .BYTE   mne_orax        ; $19  ORA abs
        .BYTE   mne_incx        ; $1A  INC A
        .BYTE   mne_tcsx        ; $1B  TCS
        .BYTE   mne_trbx        ; $1C  TRB abs
        .BYTE   mne_orax        ; $1D  ORA abs,X
        .BYTE   mne_aslx        ; $1E  ASL abs,X
        .BYTE   mne_orax        ; $1F  ORA absl,X
;
        .BYTE   mne_jsrx        ; $20  JSR abs
        .BYTE   mne_andx        ; $21  AND (dp,X)
        .BYTE   mne_jslx        ; $22  JSL absl
        .BYTE   mne_andx        ; $23  AND <offset>,S
        .BYTE   mne_bitx        ; $24  BIT dp
        .BYTE   mne_andx        ; $25  AND dp
        .BYTE   mne_rolx        ; $26  ROL dp
        .BYTE   mne_andx        ; $27  AND [dp]
        .BYTE   mne_plpx        ; $28  PLP
        .BYTE   mne_andx        ; $29  AND #
        .BYTE   mne_rolx        ; $2A  ROL A
        .BYTE   mne_pldx        ; $2B  PLD
        .BYTE   mne_bitx        ; $2C  BIT abs
        .BYTE   mne_andx        ; $2D  AND abs
        .BYTE   mne_rolx        ; $2E  ROL abs
        .BYTE   mne_andx        ; $2F  AND absl
;
        .BYTE   mne_bmix        ; $30  BMI abs
        .BYTE   mne_andx        ; $31  AND (<dp>),Y
        .BYTE   mne_andx        ; $32  AND (dp)
        .BYTE   mne_andx        ; $33  AND (<offset>,S),Y
        .BYTE   mne_bitx        ; $34  BIT dp,X
```

```
        .BYTE   mne_andx        ; $35  AND dp,X
        .BYTE   mne_rolx        ; $36  ROL dp,X
        .BYTE   mne_andx        ; $37  AND [dp],Y
        .BYTE   mne_secx        ; $38  SEC
        .BYTE   mne_andx        ; $39  AND abs,Y
        .BYTE   mne_decx        ; $3A  DEC A
        .BYTE   mne_tscx        ; $3B  TSC
        .BYTE   mne_bitx        ; $3C  BIT abs,X
        .BYTE   mne_andx        ; $3D  AND abs,X
        .BYTE   mne_rolx        ; $3E  ROL abs,X
        .BYTE   mne_andx        ; $3F  AND absl,X
;
        .BYTE   mne_rtix        ; $40  RTI
        .BYTE   mne_eorx        ; $41  EOR (dp,X)
        .BYTE   mne_wdmx        ; $42  WDM
        .BYTE   mne_eorx        ; $43  EOR <offset>,S
        .BYTE   mne_mvpx        ; $44  MVP sb,db
        .BYTE   mne_eorx        ; $45  EOR dp
        .BYTE   mne_lsrx        ; $46  LSR dp
        .BYTE   mne_eorx        ; $47  EOR [dp]
        .BYTE   mne_phax        ; $48  PHA
        .BYTE   mne_eorx        ; $49  EOR #
        .BYTE   mne_lsrx        ; $4A  LSR A
        .BYTE   mne_phkx        ; $4B  PHK
        .BYTE   mne_jmpx        ; $4C  JMP abs
        .BYTE   mne_eorx        ; $4D  EOR abs
        .BYTE   mne_lsrx        ; $4E  LSR abs
        .BYTE   mne_eorx        ; $4F  EOR absl
;
        .BYTE   mne_bvcx        ; $50  BVC abs
        .BYTE   mne_eorx        ; $51  EOR (<dp>),Y
        .BYTE   mne_eorx        ; $52  EOR (dp)
        .BYTE   mne_eorx        ; $53  EOR (<offset>,S),Y
        .BYTE   mne_mvnx        ; $54  MVN sb,db
        .BYTE   mne_eorx        ; $55  EOR dp,X
        .BYTE   mne_lsrx        ; $56  LSR dp,X
        .BYTE   mne_eorx        ; $57  EOR [dp],Y
        .BYTE   mne_clix        ; $58  CLI
        .BYTE   mne_eorx        ; $59  EOR abs,Y
        .BYTE   mne_phyx        ; $5A  PHY
        .BYTE   mne_tcdx        ; $5B  TCD
        .BYTE   mne_jmlx        ; $5C  JML absl
        .BYTE   mne_eorx        ; $5D  EOR abs,X
        .BYTE   mne_lsrx        ; $5E  LSR abs,X
        .BYTE   mne_eorx        ; $5F  EOR absl,X
;
        .BYTE   mne_rtsx        ; $60  RTS
        .BYTE   mne_adcx        ; $61  ADC (dp,X)
        .BYTE   mne_perx        ; $62  PER
        .BYTE   mne_adcx        ; $63  ADC <offset>,S
```

```
        .BYTE   mne_stzx        ; $64  STZ dp
        .BYTE   mne_adcx        ; $65  ADC dp
        .BYTE   mne_rorx        ; $66  ROR dp
        .BYTE   mne_adcx        ; $67  ADC [dp]
        .BYTE   mne_plax        ; $68  PLA
        .BYTE   mne_adcx        ; $69  ADC #
        .BYTE   mne_rorx        ; $6A  ROR A
        .BYTE   mne_rtlx        ; $6B  RTL
        .BYTE   mne_jmpx        ; $6C  JMP (abs)
        .BYTE   mne_adcx        ; $6D  ADC abs
        .BYTE   mne_rorx        ; $6E  ROR abs
        .BYTE   mne_adcx        ; $6F  ADC absl
;
        .BYTE   mne_bvsx        ; $70  BVS abs
        .BYTE   mne_adcx        ; $71  ADC (<dp>),Y
        .BYTE   mne_adcx        ; $72  ADC (dp)
        .BYTE   mne_adcx        ; $73  ADC (<offset>,S),Y
        .BYTE   mne_stzx        ; $74  STZ dp,X
        .BYTE   mne_adcx        ; $75  ADC dp,X
        .BYTE   mne_rorx        ; $76  ROR dp,X
        .BYTE   mne_adcx        ; $77  ADC [dp],Y
        .BYTE   mne_seix        ; $78  SEI
        .BYTE   mne_adcx        ; $79  ADC abs,Y
        .BYTE   mne_plyx        ; $7A  PLY
        .BYTE   mne_tdcx        ; $7B  TDC
        .BYTE   mne_jmpx        ; $7C  JMP (abs,X)
        .BYTE   mne_adcx        ; $7D  ADC abs,X
        .BYTE   mne_rorx        ; $7E  ROR abs,X
        .BYTE   mne_adcx        ; $7F  ADC absl,X
;
        .BYTE   mne_brax        ; $80  BRA abs
        .BYTE   mne_stax        ; $81  STA (dp,X)
        .BYTE   mne_brlx        ; $82  BRL abs
        .BYTE   mne_stax        ; $83  STA <offset>,S
        .BYTE   mne_styx        ; $84  STY dp
        .BYTE   mne_stax        ; $85  STA dp
        .BYTE   mne_stxx        ; $86  STX dp
        .BYTE   mne_stax        ; $87  STA [dp]
        .BYTE   mne_deyx        ; $88  DEY
        .BYTE   mne_bitx        ; $89  BIT #
        .BYTE   mne_txax        ; $8A  TXA
        .BYTE   mne_phbx        ; $8B  PHB
        .BYTE   mne_styx        ; $8C  STY abs
        .BYTE   mne_stax        ; $8D  STA abs
        .BYTE   mne_stxx        ; $8E  STX abs
        .BYTE   mne_stax        ; $8F  STA absl
;
        .BYTE   mne_bccx        ; $90  BCC abs
        .BYTE   mne_stax        ; $91  STA (<dp>),Y
        .BYTE   mne_stax        ; $92  STA (dp)
```

```
        .BYTE   mne_stax        ; $93  STA (<offset>,S),Y
        .BYTE   mne_styx        ; $94  STY dp,X
        .BYTE   mne_stax        ; $95  STA dp,X
        .BYTE   mne_stxx        ; $96  STX dp,Y
        .BYTE   mne_stax        ; $97  STA [dp],Y
        .BYTE   mne_tyax        ; $98  TYA
        .BYTE   mne_stax        ; $99  STA abs,Y
        .BYTE   mne_txsx        ; $9A  TXS
        .BYTE   mne_txyx        ; $9B  TXY
        .BYTE   mne_stzx        ; $9C  STZ abs
        .BYTE   mne_stax        ; $9D  STA abs,X
        .BYTE   mne_stzx        ; $9E  STZ abs,X
        .BYTE   mne_stax        ; $9F  STA absl,X
;
        .BYTE   mne_ldyx        ; $A0  LDY #
        .BYTE   mne_ldax        ; $A1  LDA (dp,X)
        .BYTE   mne_ldxx        ; $A2  LDX #
        .BYTE   mne_ldax        ; $A3  LDA <offset>,S
        .BYTE   mne_ldyx        ; $A4  LDY dp
        .BYTE   mne_ldax        ; $A5  LDA dp
        .BYTE   mne_ldxx        ; $A6  LDX dp
        .BYTE   mne_ldax        ; $A7  LDA [dp]
        .BYTE   mne_tayx        ; $A8  TAY
        .BYTE   mne_ldax        ; $A9  LDA #
        .BYTE   mne_taxx        ; $AA  TAX
        .BYTE   mne_plbx        ; $AB  PLB
        .BYTE   mne_ldyx        ; $AC  LDY abs
        .BYTE   mne_ldax        ; $AD  LDA abs
        .BYTE   mne_ldxx        ; $AE  LDX abs
        .BYTE   mne_ldax        ; $AF  LDA absl
;
        .BYTE   mne_bcsx        ; $B0  BCS abs
        .BYTE   mne_ldax        ; $B1  LDA (<dp>),Y
        .BYTE   mne_ldax        ; $B2  LDA (dp)
        .BYTE   mne_ldax        ; $B3  LDA (<offset>,S),Y
        .BYTE   mne_ldyx        ; $B4  LDY dp,X
        .BYTE   mne_ldax        ; $B5  LDA dp,X
        .BYTE   mne_ldxx        ; $B6  LDX dp,Y
        .BYTE   mne_ldax        ; $B7  LDA [dp],Y
        .BYTE   mne_clvx        ; $B8  CLV
        .BYTE   mne_ldax        ; $B9  LDA abs,Y
        .BYTE   mne_tsxx        ; $BA  TSX
        .BYTE   mne_tyxx        ; $BB  TYX
        .BYTE   mne_ldyx        ; $BC  LDY abs,X
        .BYTE   mne_ldax        ; $BD  LDA abs,X
        .BYTE   mne_ldxx        ; $BE  LDX abs,Y
        .BYTE   mne_ldax        ; $BF  LDA absl,X
;
        .BYTE   mne_cpyx        ; $C0  CPY #
        .BYTE   mne_cmpx        ; $C1  CMP (dp,X)
```

```
        .BYTE   mne_repx        ; $C2  REP #
        .BYTE   mne_cmpx        ; $C3  CMP <offset>,S
        .BYTE   mne_cpyx        ; $C4  CPY dp
        .BYTE   mne_cmpx        ; $C5  CMP dp
        .BYTE   mne_decx        ; $C6  DEC dp
        .BYTE   mne_cmpx        ; $C7  CMP [dp]
        .BYTE   mne_inyx        ; $C8  INY
        .BYTE   mne_cmpx        ; $C9  CMP #
        .BYTE   mne_dexx        ; $CA  DEX
        .BYTE   mne_waix        ; $CB  WAI
        .BYTE   mne_cpyx        ; $CC  CPY abs
        .BYTE   mne_cmpx        ; $CD  CMP abs
        .BYTE   mne_decx        ; $CE  DEC abs
        .BYTE   mne_cmpx        ; $CF  CMP absl
;
        .BYTE   mne_bnex        ; $D0  BNE abs
        .BYTE   mne_cmpx        ; $D1  CMP (<dp>),Y
        .BYTE   mne_cmpx        ; $D2  CMP (dp)
        .BYTE   mne_cmpx        ; $D3  CMP (<offset>,S),Y
        .BYTE   mne_peix        ; $D4  PEI dp
        .BYTE   mne_cmpx        ; $D5  CMP dp,X
        .BYTE   mne_decx        ; $D6  DEC dp,X
        .BYTE   mne_cmpx        ; $D7  CMP [dp],Y
        .BYTE   mne_cldx        ; $D8  CLD
        .BYTE   mne_cmpx        ; $D9  CMP abs,Y
        .BYTE   mne_phxx        ; $DA  PHX
        .BYTE   mne_stpx        ; $DB  STP
        .BYTE   mne_jmpx        ; $DC  JMP [abs]
        .BYTE   mne_cmpx        ; $DD  CMP abs,X
        .BYTE   mne_decx        ; $DE  DEC abs,X
        .BYTE   mne_cmpx        ; $DF  CMP absl,X
;
        .BYTE   mne_cpxx        ; $E0  CPX #
        .BYTE   mne_sbcx        ; $E1  SBC (dp,X)
        .BYTE   mne_sepx        ; $E2  SEP #
        .BYTE   mne_sbcx        ; $E3  SBC <offset>,S
        .BYTE   mne_cpxx        ; $E4  CPX dp
        .BYTE   mne_sbcx        ; $E5  SBC dp
        .BYTE   mne_incx        ; $E6  INC dp
        .BYTE   mne_sbcx        ; $E7  SBC [dp]
        .BYTE   mne_inxx        ; $E8  INX
        .BYTE   mne_sbcx        ; $E9  SBC #
        .BYTE   mne_nopx        ; $EA  NOP
        .BYTE   mne_xbax        ; $EB  XBA
        .BYTE   mne_cpxx        ; $EC  CPX abs
        .BYTE   mne_sbcx        ; $ED  SBC abs
        .BYTE   mne_incx        ; $EE  INC abs
        .BYTE   mne_sbcx        ; $EF  SBC absl
;
        .BYTE   mne_beqx        ; $F0  BEQ abs
```

```
        .BYTE   mne_sbcx        ; $F1  SBC (<dp>),Y
        .BYTE   mne_sbcx        ; $F2  SBC (dp)
        .BYTE   mne_sbcx        ; $F3  SBC (<offset>,S),Y
        .BYTE   mne_peax        ; $F4  PEA #
        .BYTE   mne_sbcx        ; $F5  SBC dp,X
        .BYTE   mne_incx        ; $F6  INC dp,X
        .BYTE   mne_sbcx        ; $F7  SBC [dp],Y
        .BYTE   mne_sedx        ; $F8  SED
        .BYTE   mne_sbcx        ; $F9  SBC abs,Y
        .BYTE   mne_plxx        ; $FA  PLX
        .BYTE   mne_xcex        ; $FB  XCE
        .BYTE   mne_jsrx        ; $FC  JSR (abs,X)
        .BYTE   mne_sbcx        ; $FD  SBC abs,X
        .BYTE   mne_incx        ; $FE  INC abs,X
        .BYTE   mne_sbcx        ; $FF  SBC absl,X
;
;
;       instruction addressing modes & sizes in opcode order...
;
;           xxxxxxxx
;           ||||||||
;           ||||++++---> Addressing Mode
;           ||||         ---------------------------------
;           ||||          0000  dp, abs, absl, implied or A
;           ||||          0001  #
;           ||||          0010  dp,X, abs,X or absl,X
;           ||||          0011  dp,Y or abs,Y
;           ||||          0100  (dp) or (abs)
;           ||||          0101  [dp] or [abs]
;           ||||          0110  [dp],Y
;           ||||          0111  (dp,X) or (abs,X)
;           ||||          1000  (<dp>),Y
;           ||||          1001  <offset>,S
;           ||||          1010  (<offset>,S),Y
;           ||||          1011  sbnk,dbnk (MVN or MVP)
;           ||||          ---------------------------------
;           ||||           #    = immediate
;           ||||           A    = accumulator
;           ||||           abs  = absolute
;           ||||           absl = absolute long
;           ||||           dbnk = destination bank
;           ||||           dp   = direct (zero) page
;           ||||           S    = stack relative
;           ||||           sbnk = source bank
;           ||||         ---------------------------------
;           ||||
;           ||++-------> binary-encoded operand size
;           |+---------> 1: relative branch instruction
;           +----------> 1: variable operand size...
;
```

```
;           ------------------------------------------------------------
;           Variable operand size refers to an immediate mode instruction
;           that can accept either an 8 or 16 bit operand.  During instr-
;           uction assembly, an 8 bit operand can be forced to 16 bits by
;           preceding the operand field with !,  e.g.,  LDA !#$01,  which
;           will assemble as $A9 $01 $00.
;           ------------------------------------------------------------
;
mnetabam:
        .BYTE   ops0|am_nam     ; $00  BRK
        .BYTE   ops1|am_indx    ; $01  ORA (dp,X)
        .BYTE   ops1|am_nam     ; $02  COP
        .BYTE   ops1|am_stk     ; $03  ORA <offset>,S
        .BYTE   ops1|am_nam     ; $04  TSB dp
        .BYTE   ops1|am_nam     ; $05  ORA dp
        .BYTE   ops1|am_nam     ; $06  ASL dp
        .BYTE   ops1|am_indl    ; $07  ORA [dp]
        .BYTE   ops0|am_nam     ; $08  PHP
        .BYTE   vops|am_imm     ; $09  ORA #
        .BYTE   ops0|am_nam     ; $0A  ASL A
        .BYTE   ops0|am_nam     ; $0B  PHD
        .BYTE   ops2|am_nam     ; $0C  TSB abs
        .BYTE   ops2|am_nam     ; $0D  ORA abs
        .BYTE   ops2|am_nam     ; $0E  ASL abs
        .BYTE   ops3|am_nam     ; $0F  ORA absl
;
        .BYTE   bop1|am_nam     ; $10  BPL abs
        .BYTE   ops1|am_indy    ; $11  ORA (<dp>),Y
        .BYTE   ops1|am_ind     ; $12  ORA (dp)
        .BYTE   ops1|am_stky    ; $13  ORA (<offset>,S),Y
        .BYTE   ops1|am_nam     ; $14  TRB dp
        .BYTE   ops1|am_adrx    ; $15  ORA dp,X
        .BYTE   ops1|am_adrx    ; $16  ASL dp,X
        .BYTE   ops1|am_indly   ; $17  ORA [dp],Y
        .BYTE   ops0|am_nam     ; $18  CLC
        .BYTE   ops2|am_nam     ; $19  ORA abs
        .BYTE   ops0|am_nam     ; $1A  INC A
        .BYTE   ops0|am_nam     ; $1B  TCS
        .BYTE   ops2|am_nam     ; $1C  TRB abs
        .BYTE   ops2|am_adrx    ; $1D  ORA abs,X
        .BYTE   ops2|am_adrx    ; $1E  ASL abs,X
        .BYTE   ops3|am_adrx    ; $1F  ORA absl,X
;
        .BYTE   ops2|am_nam     ; $20  JSR abs
        .BYTE   ops1|am_indx    ; $21  AND (dp,X)
        .BYTE   ops3|am_nam     ; $22  JSL absl
        .BYTE   ops1|am_stk     ; $23  AND <offset>,S
        .BYTE   ops1|am_nam     ; $24  BIT dp
        .BYTE   ops1|am_nam     ; $25  AND dp
        .BYTE   ops1|am_nam     ; $26  ROL dp
```

```
        .BYTE   ops1|am_indl    ; $27  AND [dp]
        .BYTE   ops0|am_nam     ; $28  PLP
        .BYTE   vops|am_imm     ; $29  AND #
        .BYTE   ops0|am_nam     ; $2A  ROL A
        .BYTE   ops0|am_nam     ; $2B  PLD
        .BYTE   ops2|am_nam     ; $2C  BIT abs
        .BYTE   ops2|am_nam     ; $2D  AND abs
        .BYTE   ops2|am_nam     ; $2E  ROL abs
        .BYTE   ops3|am_nam     ; $2F  AND absl
;
        .BYTE   bop1|am_nam     ; $30  BMI abs
        .BYTE   ops1|am_indy    ; $31  AND (<dp>),Y
        .BYTE   ops1|am_ind     ; $32  AND (dp)
        .BYTE   ops1|am_stky    ; $33  AND (<offset>,S),Y
        .BYTE   ops1|am_adrx    ; $34  BIT dp,X
        .BYTE   ops1|am_adrx    ; $35  AND dp,X
        .BYTE   ops1|am_adrx    ; $36  ROL dp,X
        .BYTE   ops1|am_indly   ; $37  AND [dp],Y
        .BYTE   ops0|am_nam     ; $38  SEC
        .BYTE   ops2|am_adry    ; $39  AND abs,Y
        .BYTE   ops0|am_nam     ; $3A  DEC A
        .BYTE   ops0|am_nam     ; $3B  TSC
        .BYTE   ops2|am_adrx    ; $3C  BIT abs,X
        .BYTE   ops2|am_adrx    ; $3D  AND abs,X
        .BYTE   ops2|am_adrx    ; $3E  ROL abs,X
        .BYTE   ops3|am_adrx    ; $3F  AND absl,X
;
        .BYTE   ops0|am_nam     ; $40  RTI
        .BYTE   ops1|am_indx    ; $41  EOR (dp,X)
        .BYTE   ops0|am_nam     ; $42  WDM
        .BYTE   ops1|am_stk     ; $43  EOR <offset>,S
        .BYTE   ops2|am_move    ; $44  MVP sb,db
        .BYTE   ops1|am_nam     ; $45  EOR dp
        .BYTE   ops1|am_nam     ; $46  LSR dp
        .BYTE   ops1|am_indl    ; $47  EOR [dp]
        .BYTE   ops0|am_nam     ; $48  PHA
        .BYTE   vops|am_imm     ; $49  EOR #
        .BYTE   ops0|am_nam     ; $4A  LSR A
        .BYTE   ops0|am_nam     ; $4B  PHK
        .BYTE   ops2|am_nam     ; $4C  JMP abs
        .BYTE   ops2|am_nam     ; $4D  EOR abs
        .BYTE   ops2|am_nam     ; $4E  LSR abs
        .BYTE   ops3|am_nam     ; $4F  EOR absl
;
        .BYTE   bop1|am_nam     ; $50  BVC abs
        .BYTE   ops1|am_indy    ; $51  EOR (<dp>),Y
        .BYTE   ops1|am_ind     ; $52  EOR (dp)
        .BYTE   ops1|am_stky    ; $53  EOR (<offset>,S),Y
        .BYTE   ops2|am_move    ; $54  MVN sb,db
        .BYTE   ops1|am_adrx    ; $55  EOR dp,X
```

```
        .BYTE   ops1|am_adrx    ; $56  LSR dp,X
        .BYTE   ops1|am_indly   ; $57  EOR [dp],Y
        .BYTE   ops0|am_nam     ; $58  CLI
        .BYTE   ops2|am_adry    ; $59  EOR abs,Y
        .BYTE   ops0|am_nam     ; $5A  PHY
        .BYTE   ops0|am_nam     ; $5B  TCD
        .BYTE   ops3|am_nam     ; $5C  JML absl
        .BYTE   ops2|am_adrx    ; $5D  EOR abs,X
        .BYTE   ops2|am_adrx    ; $5E  LSR abs,X
        .BYTE   ops3|am_adrx    ; $5F  EOR absl,X
;
        .BYTE   ops0|am_nam     ; $60  RTS
        .BYTE   ops1|am_indx    ; $61  ADC (dp,X)
        .BYTE   bop2|am_nam     ; $62  PER
        .BYTE   ops1|am_stk     ; $63  ADC <offset>,S
        .BYTE   ops1|am_nam     ; $64  STZ dp
        .BYTE   ops1|am_nam     ; $65  ADC dp
        .BYTE   ops1|am_nam     ; $66  ROR dp
        .BYTE   ops1|am_indl    ; $67  ADC [dp]
        .BYTE   ops0|am_nam     ; $68  PLA
        .BYTE   vops|am_imm     ; $69  ADC #
        .BYTE   ops0|am_nam     ; $6A  ROR A
        .BYTE   ops0|am_nam     ; $6B  RTL
        .BYTE   ops2|am_ind     ; $6C  JMP (abs)
        .BYTE   ops2|am_nam     ; $6D  ADC abs
        .BYTE   ops2|am_nam     ; $6E  ROR abs
        .BYTE   ops3|am_nam     ; $6F  ADC absl
;
        .BYTE   bop1|am_nam     ; $70  BVS abs
        .BYTE   ops1|am_indy    ; $71  ADC (<dp>),Y
        .BYTE   ops1|am_ind     ; $72  ADC (dp)
        .BYTE   ops1|am_stky    ; $73  ADC (<offset>,S),Y
        .BYTE   ops1|am_adrx    ; $74  STZ dp,X
        .BYTE   ops1|am_adrx    ; $75  ADC dp,X
        .BYTE   ops1|am_adrx    ; $76  ROR dp,X
        .BYTE   ops1|am_indly   ; $77  ADC [dp],Y
        .BYTE   ops0|am_nam     ; $78  SEI
        .BYTE   ops2|am_adry    ; $79  ADC abs,Y
        .BYTE   ops0|am_nam     ; $7A  PLY
        .BYTE   ops0|am_nam     ; $7B  TDC
        .BYTE   ops2|am_indx    ; $7C  JMP (abs,X)
        .BYTE   ops2|am_adrx    ; $7D  ADC abs,X
        .BYTE   ops2|am_adrx    ; $7E  ROR abs,X
        .BYTE   ops3|am_adrx    ; $7F  ADC absl,X
;
        .BYTE   bop1|am_nam     ; $80  BRA abs
        .BYTE   ops1|am_indx    ; $81  STA (dp,X)
        .BYTE   bop2|am_nam     ; $82  BRL abs
        .BYTE   ops1|am_stk     ; $83  STA <offset>,S
        .BYTE   ops1|am_nam     ; $84  STY dp
```

```
        .BYTE   ops1|am_nam     ; $85  STA dp
        .BYTE   ops1|am_nam     ; $86  STX dp
        .BYTE   ops1|am_indl    ; $87  STA [dp]
        .BYTE   ops0|am_nam     ; $88  DEY
        .BYTE   vops|am_imm     ; $89  BIT #
        .BYTE   ops0|am_nam     ; $8A  TXA
        .BYTE   ops0|am_nam     ; $8B  PHB
        .BYTE   ops2|am_nam     ; $8C  STY abs
        .BYTE   ops2|am_nam     ; $8D  STA abs
        .BYTE   ops2|am_nam     ; $8E  STX abs
        .BYTE   ops3|am_nam     ; $8F  STA absl
;
        .BYTE   bop1|am_nam     ; $90  BCC abs
        .BYTE   ops1|am_indy    ; $91  STA (<dp>),Y
        .BYTE   ops1|am_ind     ; $92  STA (dp)
        .BYTE   ops1|am_stky    ; $93  STA (<offset>,S),Y
        .BYTE   ops1|am_adrx    ; $94  STY dp,X
        .BYTE   ops1|am_adrx    ; $95  STA dp,X
        .BYTE   ops1|am_adry    ; $96  STX dp,Y
        .BYTE   ops1|am_indly   ; $97  STA [dp],Y
        .BYTE   ops0|am_nam     ; $98  TYA
        .BYTE   ops2|am_adry    ; $99  STA abs,Y
        .BYTE   ops0|am_nam     ; $9A  TXS
        .BYTE   ops0|am_nam     ; $9B  TXY
        .BYTE   ops2|am_nam     ; $9C  STZ abs
        .BYTE   ops2|am_adrx    ; $9D  STA abs,X
        .BYTE   ops2|am_adrx    ; $9E  STZ abs,X
        .BYTE   ops3|am_adrx    ; $9F  STA absl,X
;
        .BYTE   vops|am_imm     ; $A0  LDY #
        .BYTE   ops1|am_indx    ; $A1  LDA (dp,X)
        .BYTE   vops|am_imm     ; $A2  LDX #
        .BYTE   ops1|am_stk     ; $A3  LDA <offset>,S
        .BYTE   ops1|am_nam     ; $A4  LDY dp
        .BYTE   ops1|am_nam     ; $A5  LDA dp
        .BYTE   ops1|am_nam     ; $A6  LDX dp
        .BYTE   ops1|am_indl    ; $A7  LDA [dp]
        .BYTE   ops0|am_nam     ; $A8  TAY
        .BYTE   vops|am_imm     ; $A9  LDA #
        .BYTE   ops0|am_nam     ; $AA  TAX
        .BYTE   ops0|am_nam     ; $AB  PLB
        .BYTE   ops2|am_nam     ; $AC  LDY abs
        .BYTE   ops2|am_nam     ; $AD  LDA abs
        .BYTE   ops2|am_nam     ; $AE  LDX abs
        .BYTE   ops3|am_nam     ; $AF  LDA absl
;
        .BYTE   bop1|am_nam     ; $B0  BCS abs
        .BYTE   ops1|am_indy    ; $B1  LDA (<dp>),Y
        .BYTE   ops1|am_ind     ; $B2  LDA (dp)
        .BYTE   ops1|am_stky    ; $B3  LDA (<offset>,S),Y
```

```
        .BYTE   ops1|am_adrx    ; $B4  LDY dp,X
        .BYTE   ops1|am_adrx    ; $B5  LDA dp,X
        .BYTE   ops1|am_adry    ; $B6  LDX dp,Y
        .BYTE   ops1|am_indly   ; $B7  LDA [dp],Y
        .BYTE   ops0|am_nam     ; $B8  CLV
        .BYTE   ops2|am_adry    ; $B9  LDA abs,Y
        .BYTE   ops0|am_nam     ; $BA  TSX
        .BYTE   ops0|am_nam     ; $BB  TYX
        .BYTE   ops2|am_adrx    ; $BC  LDY abs,X
        .BYTE   ops2|am_adrx    ; $BD  LDA abs,X
        .BYTE   ops2|am_adry    ; $BE  LDX abs,Y
        .BYTE   ops3|am_adrx    ; $BF  LDA absl,X
;
        .BYTE   vops|am_imm     ; $C0  CPY #
        .BYTE   ops1|am_indx    ; $C1  CMP (dp,X)
        .BYTE   ops1|am_imm     ; $C2  REP #
        .BYTE   ops1|am_stk     ; $C3  CMP <offset>,S
        .BYTE   ops1|am_nam     ; $C4  CPY dp
        .BYTE   ops1|am_nam     ; $C5  CMP dp
        .BYTE   ops1|am_nam     ; $C6  DEC dp
        .BYTE   ops1|am_indl    ; $C7  CMP [dp]
        .BYTE   ops0|am_nam     ; $C8  INY
        .BYTE   vops|am_imm     ; $C9  CMP #
        .BYTE   ops0|am_nam     ; $CA  DEX
        .BYTE   ops0|am_nam     ; $CB  WAI
        .BYTE   ops2|am_nam     ; $CC  CPY abs
        .BYTE   ops2|am_nam     ; $CD  CMP abs
        .BYTE   ops2|am_nam     ; $CE  DEC abs
        .BYTE   ops3|am_nam     ; $CF  CMP absl
;
        .BYTE   bop1|am_nam     ; $D0  BNE abs
        .BYTE   ops1|am_indy    ; $D1  CMP (<dp>),Y
        .BYTE   ops1|am_ind     ; $D2  CMP (dp)
        .BYTE   ops1|am_stky    ; $D3  CMP (<offset>,S),Y
        .BYTE   ops1|am_nam     ; $D4  PEI dp
        .BYTE   ops1|am_adrx    ; $D5  CMP dp,X
        .BYTE   ops1|am_adrx    ; $D6  DEC dp,X
        .BYTE   ops1|am_indly   ; $D7  CMP [dp],Y
        .BYTE   ops0|am_nam     ; $D8  CLD
        .BYTE   ops2|am_adry    ; $D9  CMP abs,Y
        .BYTE   ops0|am_nam     ; $DA  PHX
        .BYTE   ops0|am_nam     ; $DB  STP
        .BYTE   ops2|am_indl    ; $DC  JMP [abs]
        .BYTE   ops2|am_adrx    ; $DD  CMP abs,X
        .BYTE   ops2|am_adrx    ; $DE  DEC abs,X
        .BYTE   ops3|am_adrx    ; $DF  CMP absl,X
;
        .BYTE   vops|am_imm     ; $E0  CPX #
        .BYTE   ops1|am_indx    ; $E1  SBC (dp,X)
        .BYTE   ops1|am_imm     ; $E2  SEP #
```

```
        .BYTE   ops1|am_stk     ; $E3  SBC <offset>,S
        .BYTE   ops1|am_nam     ; $E4  CPX dp
        .BYTE   ops1|am_nam     ; $E5  SBC dp
        .BYTE   ops1|am_nam     ; $E6  INC dp
        .BYTE   ops1|am_indl    ; $E7  SBC [dp]
        .BYTE   ops0|am_nam     ; $E8  INX
        .BYTE   vops|am_imm     ; $E9  SBC #
        .BYTE   ops0|am_nam     ; $EA  NOP
        .BYTE   ops0|am_nam     ; $EB  XBA
        .BYTE   ops2|am_nam     ; $EC  CPX abs
        .BYTE   ops2|am_nam     ; $ED  SBC abs
        .BYTE   ops2|am_nam     ; $EE  INC abs
        .BYTE   ops3|am_nam     ; $EF  SBC absl
;
        .BYTE   bop1|am_nam     ; $F0  BEQ abs
        .BYTE   ops1|am_indy    ; $F1  SBC (<dp>),Y
        .BYTE   ops1|am_ind     ; $F2  SBC (dp)
        .BYTE   ops1|am_stky    ; $F3  SBC (<offset>,S),Y
        .BYTE   ops2|am_imm     ; $F4  PEA #
        .BYTE   ops1|am_adrx    ; $F5  SBC dp,X
        .BYTE   ops1|am_adrx    ; $F6  INC dp,X
        .BYTE   ops1|am_indly   ; $F7  SBC [dp],Y
        .BYTE   ops0|am_nam     ; $F8  SED
        .BYTE   ops2|am_adry    ; $F9  SBC abs,Y
        .BYTE   ops0|am_nam     ; $FA  PLX
        .BYTE   ops0|am_nam     ; $FB  XCE
        .BYTE   ops2|am_indx    ; $FC  JSR (abs,X)
        .BYTE   ops2|am_adrx    ; $FD  SBC abs,X
        .BYTE   ops2|am_adrx    ; $FE  INC abs,X
        .BYTE   ops3|am_adrx    ; $FF  SBC absl,X
;
;
;       .X & .Y immediate mode opcodes...
;
vopidx:
        .BYTE   opc_cpxi        ;CPX #
        .BYTE   opc_cpyi        ;CPY #
        .BYTE   opc_ldxi        ;LDX #
        .BYTE   opc_ldyi        ;LDY #
n_vopidx        = *-vopidx      ;number of opcodes
;
;
;       addressing mode symbology lookup...
;
ms_lutab:
        .WORD   ms_nam          ;no symbol
        .WORD   ms_imm          ;#
        .WORD   ms_addrx        ;<addr>,X
        .WORD   ms_addry        ;<addr>,Y
        .WORD   ms_ind          ;(<addr>)
```

```
        .WORD   ms_indl         ;[<dp>]
        .WORD   ms_indly        ;[<dp>],Y
        .WORD   ms_indx         ;(<addr>,X)
        .WORD   ms_indy         ;(<dp>),Y
        .WORD   ms_stk          ;<offset>,S
        .WORD   ms_stky         ;(<offset>,S),Y
        .WORD   ms_nam          ;<sbnk>,<dbnk>
;
;
;       addressing mode symbology strings...
;
ms_nam:
        .BYTE   " ",0           ;no symbol
ms_addrx:
        .BYTE   " ,X",0         ;<addr>,X
ms_addry:
        .BYTE   " ,Y",0         ;<addr>,Y
ms_imm:
        .BYTE   "#",0           ;#
ms_ind:
        .BYTE   "()",0          ;(<addr>)
ms_indl:
        .BYTE   "[]",0          ;[<dp>]
ms_indly:
        .BYTE   "[],Y",0        ;[<dp>],Y
ms_indx:
        .BYTE   "(,X)",0        ;(<addr>,X)
ms_indy:
        .BYTE   "(),Y",0        ;(<dp>),Y
ms_move:
        .BYTE   ",$",0          ;<sbnk>,<dbnk>
ms_stk:
        .BYTE   " ,S",0         ;<offset>,S
ms_stky:
        .BYTE   "(,S),Y",0      ;(<offset>,S),Y
;
;===============================================================================
;
;CONSOLE DISPLAY CONTROL STRINGS
;
dc_lf:
        LF                      ;newline
        .BYTE   0
;
dc_bs:  ;destructive backspace
        .BYTE   a_bs
        .BYTE   $20
        .BYTE   a_bs
        .BYTE   0
;
```

```
dc_cl_DUMB:                     ;clear to end of line
        .BYTE   $0d,$0a
        .BYTE   0
dc_cl_ANSI:                     ;clear to end of line
        .BYTE   a_esc,"[K"
        .BYTE   0
dc_cl_WYSE:                     ;clear to end of line
        .BYTE   a_esc,"T"
        .BYTE   0

;
;

;
;================================================================================
;
;TEXT STRINGS
;
mm_brk:
        RB
        LF
        .BYTE   "*BRK"
        LF
        .BYTE   0
;
mm_entry:
        LF
        .BYTE   "Supermon 816 "
        SOFTVERS
        .BYTE   " "
        LF
        .BYTE   0
;
mm_err:
        .BYTE   " *ERR ",0
;
mm_prmpt:
        LF
        .BYTE   ".",0
;
mm_regs:
        LF
        .BYTE   "PB  PC   NVmxDIZC  .C   .X   .Y   SP"
        LF
        .BYTE   0
mm_regs1:
        LF
        .BYTE   " DP  DB"
        LF
        .BYTE   0
```

```
;
mm_rts:
        RB
        LF
        .BYTE   "*RTS"
        LF
        .BYTE   0


mm_S19_prmpt:
        LF
        .BYTE   "Begin sending S28 encoded file. . ."
        LF
        .BYTE   0

;
ALIVEM:
        .BYTE   $0D,$0A
        .BYTE   $0D,$0A
        .BYTE   "    __ _____  ___  __   __",$0D,$0A
        .BYTE   "  / /| ____|/ _ \/_ | / /",$0D,$0A
        .BYTE   " / /_| |__ | (_) || |/ /_",$0D,$0A
        .BYTE   "|  _ \___ \ > _ < | |  _ \",$0D,$0A
        .BYTE   "| (_) |__) | (_) || | (_) |",$0D,$0A
        .BYTE   " \___/____/ \___/ |_|\___/ ",$0D,$0A
        .BYTE   $0D,$0A
        .BYTE   "65c816 BIOS (NATIVE MODE)",$0D,$0A
        .BYTE   "DUODYNE v1 9/30/2022 - D.WERNER",$0D,$0A
        .BYTE   "-----------------------------------",$0D,$0A
        .BYTE   $0D,$0A,0
;


_txtend_        = *             ;end of program text
;
;================================================================================
```