```
; ASSEM.DOC - dOcumentation for
;      ASSEM.ACT
;    NOT copyrighted at any time.
;
;        Programmed by
;        Allen D. Doum
;
;       Downloaded from
;     THE SOFTWARE CELLAR
;       (714) 772-9671
;
; ASSEM.ACT is a psuedo-assembler
; designed as a coding and
; documentation aid for machine
; language coding within the
; ACTION! programming language.
; Use of ASSEM will require
; knowledge of machine language and
; the ACTION! language.
; I recommend:
;    PROGRAMMING THE 6502
;       by Rodney Zaks (Sybex)

; IMPORTANT: The ACTION! compiler
; should be set to CASE SENSITIVE: Y
; ASSEM.ACT will NOT work otherwise.
; ACTION! compiler will flag
; incorrectly spelled op-codes and
; addess codes. It will NOT flag
; invalid combinations or addresses.

; "ASSEM.ACT" is the recommend file
; name for the psuedo-assembler.

; This documentation file contains
; "TWO56", an example program. Just
; compile this file and RUN. The
; START button will return you to the
; ACTION! moniter.

; For the instuctions below you
; should print a copy of ASSEM to
; refer to.

; Example #1:

;1 INCLUDE "D1:ASSEM.ACT"
;2 BYTE a,b,result

;3  CODE
;4     Lda AB a         ;3 1st addend
;5     Clc              ;1 clear carry
;6     Adc AB b         ;3 2nd addend
;7     Sta AB result    ;3 store sum
;8  ENDC

; IN example #1 line one loads ASSEM.
; The word "CODE" in line three
; starts ASSEM; the word "ENDC" in
; line eight stops it.
; Lines four thru seven contain
; psuedo-assember instuctions that
; the ACTION! compiler will convert
; to machine language.

; Sytax of ASSEM may be free-form.
; However, that misses the point.
; The syntax of the example is
;       should be indented within
;       a CODE - ENDC pair.
; (2)   Each line should have only
;       one instuction.
; (3)   Op-codes should appear in a
;       single column.
; (4)   Addessing codes (if required)
;       appear after the op code
;       separated by a space.
; (5)   Arguments follow op-codes and
;       addressing codes separated by
;       a space. Arguments can be
;       a variable name or a numeric
;       constant. HEX values may be
;       used, but negative numbers may
;       not. Arguments may take the
;       form "x+n" where x is a
;       numeric constant or a variable
;       and n is a numeric constant.
; (6)   Each instruction should have
;       a comment which begins with
;       the number of bytes used in
;       the instruction (1,2,3)
; (7)   Additional comments may (and
;       should) follow the byte count.

; ADDRESSING notes:

; For absolute, page zero, and all
; indexed and indirect modes an
; ACTION variable name or a numeric
; address may be used as an argument.
; POINTER values and array names
; should used with caution since the
; value of the pointer will be
; change, not where the pointer
; points. If an array is to be used
; by ASSEM (but not in the ACTION
; code itself) it may be defined as
; follows:

; Example #2

; BYTE xarray = A 0 1 2 3 4 5 6 0
; BYTE index

; CODE
;       Ldx AB index     ;3
;       Lda ABx xarray   ;3
; ENDC

; This will load the accumulator with
; value from "xarray" pointed to by
; "index" as will: Lda AB xarray+n,
; where n is a numeric constant.

; Relative addressing is harder to
; use. For forward branches the
; of bytes in the instuctions skipped
; (including neither the target
; nor the branch instruction itself)
; must be the argument. This is the
; reason the the byte count for each
; instruction should be included in
; the comments. Backwards branching
; should include both the target and
; branch instructions in its
; argument, however as negative
; numbers may not be used, this
; should be presented as the one byte
; PROGRAMMING NOTES

; Case sensitive is required since
; the word "AND" is both an ACTION!
; reserved word and an assembler
; instruction. Op-codes are all
; standard assembler instuctions
; with the 1st letter capitalized
; and the other two lower-case.
; Addessing codes are two upper-case
; letters that may be followed by
; a single lower-case modifier.
; Without this distinction the
; instruction "Increment X" (Inx)
; would conflict with the address
; code "Indirect X" (INx).

; Abberations:
; 1. The Ldx instruction has a
; seperate address code for "Absolute
; indexed Y" (ABi instead of ABy).
; 2. The address code "IMi" is used
; for immediate mode on four
; instructions (Ldx,Ldy,Cpx,Cpy).
; These abberations are due to the
; nature of the 6502 machine
; language and of the
; psuedo-assembler itself.
; A single mode (PZi) was included
; for page zero indexed since it
; covered all uses of that mode.
; PZx and PZy have been included for
; completeness.

; One final note. ASSEM is not by any
; means, and is not ment to be, a
; full assembler. It is an aid to
; documenting and coding of code
; blocks. Hopefully future versions
; of ASSEM (and of ACTION!) will
; allow expanded use.

; Example #3 - Compile this file
; and RUN.

; TW056 - program to output 256 colors
;          to the screen at one time.
;     NOT copyrighted at any time by
;      Allen D. Doum or ACAOC.

INCLUDE "D1:ASSEM.ACT"  ;psuedo-assembler

MODULE

BYTE dlic=$F0,         ;DLI counter
     wsync=$D40A,      ;DLI line sync
     sdmctl=$22F,      ;DMA control (shadow)
     dmactl=$D400,     ;DMA control (hardware)
     nmien=$D40E,      ;NMI enable
     consol=$D01F,     ;console buttons
     colbk=$D01A,      ;backgroud color
     gprior=$26F       ;priority & GTIA modes

CARD vdslst=$200,      ;DLI vector
     vvblkd=$224,      ;VBI deferred vector
     sdlstl=$230,      ;display list pointer (shadow)
     dlist=$D402,      ;display list pointer (hardware)
     savvbd,           ;save area VBI vector
     savdl             ;save area display list pointer
```

```
BYTE POINTER dl=$8028      ;display list

PROC builddl () ; build display list & screen memory
    CARD POINTER c
    BYTE POINTER b
    BYTE i,i1,j
;screen memory (40 bytes!)
    b=screen
    FOR i=0 TO 14 STEP 2 DO
       j=i+i LSH 4
       b^=j
       b==+1
       b^=j
       b==+1
       i1=i+1
       j=i1+i LSH 4
       b^=j
       b==+1
       j=i1+i1 LSH 4
       b^=j
       b==+1
       b^=j
       b==+1
    OD
;display list  ;(582 bytes!!)
    b=dl
    b^=$70    ;24 blank lines
    b==+1
    b^=$70
    b==+1
    b^=$70
    b==+1
FOR i=0 TO 15 DO
  FOR j=0 TO 10 DO
    b^=$4F    ;screen memory LMS
    c=b+1
    c^=screen
    b=c+2
  OD
    b^=$CF    ;interupt call + LMS
    c=b+1
    c^=screen
    b=c+2
OD
    b^=$41    ;VB jump
    c=b+1
    c^=dl
RETURN

PROC VBI()   ;vertical blank interrupt
CODE
   Lda IM 0         ;2 reset DLI counter
   Sta PZ dlic      ;2
   Jmp IN savvbd    ;3 exit VBI
ENDC

PROC DLI()   ;display list interrupt
CODE
   Pha              ;1 save accum.
   Inc PZ dlic      ;2 increment counter
   Lda PZ dlic      ;2
   Asl AC           ;1 multiply * 16
   Asl AC           ;1
   Asl AC           ;1
   Asl AC           ;1
   Sta AB wsync     ;3 wait end of line
   Sta AB colbk     ;3 change color
   Pla              ;1 restore accum.
   Rti              ;1 exit DLI
ENDC
```

```
   sdmctl=0       ;turn off screen
   dmactl=0
   savvbd=vvblkd  ;save VBI address
   savdl=sdlstl   ;save display list
   gprior=64      ;GTIA mode
   nmien=192      ;enable interrupts
   sdlstl=dl      ;display list
   dlist=dl
   vdslst=DLI     ;display list interrupt
   vvblkd=VBI     ;vertical blank interrupt
   sdmctl=$22     ;turn on screen
   dmactl=$22
RETURN

PROC noscreen()  ;restore system screen
   sdmctl=0       ;turn of screen
   dmactl=0
   gprior=0       ;stop GTIA mode
   nmien=64       ;stop DLI
   sdlstl=savdl   ;restore display list
   dlist=savdl
   vvblkd=savvbd  ;vertical blank interrupt
   sdmctl=$22     ;turn on screen
   dmactl=$22
RETURN

PROC two56()  ;main proc
   builddl()
   goscreen()
   DO
       consol=8
       UNTIL consol#7
   OD
   noscreen()
RETURN
```