6502 Forth like tiny OS#

```
;
; LCD.AS
;
; Implements a "FORTH-like" operating system on a small 6502 SBC.
; The hardware includes a 6502 running at 1 Mhz, a 6522, 2 KB of RAM,
; 10 KB of EEPROM, a 4-line by 16-char LCD display, and a 22-key keyboard.
; The keyboard is scanned via the 6522 ports. Keys include hex chars 0 thru
; F, Space, <CR>, and <BS>. The keyboard is scanned via polling, whenever no
; other code is executing.
; The LCD display is mapped into memory.
; The SBC has other hardware features that are not currently used.
;
; All keystrokes are in hex. The OS identifies the purpose of a keyword
; based on it's length:
;   1-char symbols are simple functions (currently Examine and Deposit)
;   2-char symbols are byte values
;   3-char symbols are FORTH-like 'words'
;   4-char symbols are word values.
; Example: A3C2 5B F00
; The above line would do the following: push the word $A3C2 onto the stack,
; push the byte $5B onto the stack, then execute word 'F00'.
;
; 'Word' definitions are stored in an 8K EEPROM at $C000. The storage is
; divided into 32-byte blocks - all words start on a 32-byte boundary.
; The first byte of a block indicates the contents of the block:
;   0 - deleted or unused block
;   1 - block contains space-delimited tokens
;   2 - block contains executable code, ending with RTS
; The 2nd thru 4th bytes of a block contain the word's name, in ASCII.
; The remainder of the block contains either tokens or code.
;
; A number of words are defined so far:
;   E   examine memory (addr on top of stack)
;   D   deposit to memory (addr and data on stack)
;   CCC   compile a new word into EEPROM (finds an empty block)
;   001   POP - pop a byte from the stack
;   002   INCW - increment a word on the top of the stack
;   003   DUPW - duplicate a word
;   004   READ - read an addr - pop/read the addr, push the result byte
;   005   PRINTBNS - print a byte value
;   006   PRINTB - print a byte value plus a trailing space
;   007   PRINTW - print a word value plus a trailing space
;   008   DELAY - a short time delay - param is a byte
;   009   LDELAY - time delay, 1-second increments
;   00A   WRITE - write a location - addr word and data byte are on stack
;   00B   ADD - add 2 bytes on the stack, push the result
;   00C   SUB - subtract 2 bytes
;   00D   SHIFT-R - right-shift a byte
;   00E   SHIFT-L - left-shift a byte
;   00F   INC - increment a byte
;   010   DEC - decrement a byte
;   011   DO - the 'do' of a do-while loop
;   012   WHILE - the 'while' of a do-while loop
;
; The 'E' word examines/displays a memory location, but formats the
; output to be a subsequent 'D' command. For example, you enter:
```

```
;   0400 E
; then the next display line becomes:
;   0400 12 D
; where 12 is the current contents of address 0400. The cursor is
; sitting on the '1' in '12', so you can enter a new value easily
; if desired (just press Enter and the value is unchanged).
; After entering a new value, press Enter, and the display is now:
;   0401 AB D
; - note that the address incremented, and 'AB' is the current contents.
;
; The compile (CCC) word creates a new token-style word definition in
; the first available unused block (code-style words must be entered
; manually, using the 'D' command).
; Example:  CCC BAA 89 005
; This example compiles a new word, named 'BAA'. The function of BAA
; is to push $89 on the stack, then print that value.


;
; RAM is 0000-0800
; 6522 I/O is at "VIA" (2000-200F)
;   CA1 is for falling-edge priority interrupts
;   CA2
;   CB2 is SR out
;   CB1 is SR clock out
;   PA0-2 are inputs for keyboard scan
;     PA3
;     PA4-6 is interrupt priority
;     PA7
;   PB0-7 are outputs for keyboard scan

OUTBUF     EQU     $0040           ; transmit buffer
INBUFF     EQU     $0080           ; receiver buffer
ZPIND      EQU   $00FC      ; for indirect LDA/STA
DSTACK      EQU   $0700      ; bottom of Data Stack
VIA             EQU     $2000           ; address of 6522 VIA
ACIA            EQU     $4000           ; address of 6551 ACIA
LCD0      EQU   $6000      ; LCD display
LCD1      EQU   $6001
EE8K      EQU   $C000      ; FORTH 'word' storage, 8KB
EEPROM      EQU   $F800      ; boot and kernel code, 2KB

;
; 6522 VIA definitions
;
ORG     VIA
ORB     .DS     1               ;Output Register B
IRB     EQU     ORB             ;Input Register B
ORA     .DS     1               ;Output Register A
IRA     EQU     ORA             ;Input Register A
DDRB    .DS     1               ;Data Direction Register B
DDRA    .DS     1               ;Data Direction Register A
T1CL    .DS     1               ;read:  T1 counter, low-order
;write: T1 latches, low-order
T1CH    .DS     1               ;T1 counter, high-order
T1LL    .DS     1               ;T1 latches, low-order
T1LH    .DS     1               ;T1 latches, high-order
T2CL    .DS     1               ;read:  T2 counter, low-order
;write: T2 latches, low-order
```

```
T2CH    .DS     1               ;T2 counter, high-order
SR      .DS     1               ;Shift Register
ACR     .DS     1               ;Auxiliary Control Register
PCR     .DS     1               ;Peripheral Control Register
IFR     .DS     1               ;Interrupt Flag Register
IER     .DS     1               ;Interrupt Enable Register

;
; 6551 ACIA definitions
;
ORG     ACIA
ATXM    .DS     1               ;Transmitter Register (write only)
ARCX    EQU     ATXM            ;Receiver Register (read only)
ASTS    .DS     1               ;Status Register (read only)
ARES    EQU     ASTS            ;Soft Reset  (write only)
ACMD    .DS     1               ;Command Register
ACTL    .DS     1               ;Control Register

;
; page zero variable declarations
;
ORG     $0000
KBUFF   .DS   32   ; keyboard input buffer
ORACOP  .DS     1       ; copy of 6522 ORA
IFRCOP  .DS     1       ; copy of 6522 IFR
DLYCNT  .DS     1       ; delay counter
OUTNDX  .DS     1       ; output char index
UASTAT  .DS     1       ; UART status
FDA     .DS     1
CURSOR  .DS     1   ; LCD cursor location
TEMP    .DS     1
FRADLO  .DS     1
FRADHI  .DS     1
FCNT    .DS     1
FFDA    .DS     1
FRENLO  .DS     1
FRENHI  .DS     1
ECNTLO  .DS     1
ECNTHI  .DS     1
TOADLO  .DS     1
TOADHI  .DS     1
KEY   .DS   1
TOKEN1   .DS   1   ; ptr: beginning of token
TOKEN2   .DS   1   ; ptr: end of token
DSP   .DS   1   ; data stack pointer
ASCBYT   .DS   2
TKLEN   .DS   1   ; token length
EEPTR   .DS   2   ;
WLETT   .DS   1   ;
BTYPE   .DS   1   ; type: tokens, code, deleted
BUFFPTR   .DS   2   ; ptr to current token buffer
TKTMP   .DS   1
COMPTR   .DS   2   ; ptr used by Compile

ORG     INBUFF
INNDX   .DS     1       ; input buffer index
```

```
ORG   EEPROM   ; $F800

LCDBUSY:
; wait for the LCD to not be busy
PHA
L1:   LDA   LCD0
AND   #$80
BNE   L1
PLA
RTS

LCDINIT:
; init the LCD display
LDX   #$04      ; do it 4 times
L0:   LDA   #$38      ;
STA   LCD0
JSR   LCDBUSY
DEX
BNE   L0
LDA   #$06      ;
STA   LCD0
JSR   LCDBUSY
LDA   #$0E      ;
STA   LCD0
JSR   LCDBUSY
LDA   #$01      ; clear display
STA   LCD0
JSR   LCDBUSY
LDA   #$90      ; start on 3rd line
STA   CURSOR      ; init cursor location
STA   LCD0
JSR   LCDBUSY
RTS



ORG   $F850
SCROLL:
; Treat the 4 X 16 display as 2 lines of 32 char each.
; scroll everything up one line
; leave cursor at start of 2nd line (3rd physical line)
LDX   #$00      ; index, beginning of kybd buffer
LDA   #$90      ; 3rd line
S6:   STA   LCD0      ; set cursor pos
JSR   LCDBUSY
S1:   LDA   LCD1      ; get char from display
STA   KBUFF,X      ; copy char to buffer
JSR   LCDBUSY
INX         ; next char
CPX   #$20      ; all done?
BEQ   S2      ; yes
CPX   #$10      ; end of line?
BNE   S1      ; no
LDA   #$D0      ; yes, goto 4th display line
BNE   S6
S2:   ; copy buffer to 1st line
LDX   #$00      ; beginning of buffer
LDA   #$80      ; 1st line
S5:   STA   LCD0      ; set cursor pos
```

```
JSR   LCDBUSY
S3:   LDA   KBUFF,X      ; get char from buffer
STA   LCD1      ; write to display
JSR   LCDBUSY
INX         ; next char
CPX   #$20      ; all done?
BEQ   S4      ; yes
CPX   #$10      ; end of line?
BNE   S3      ; no
LDA   #$C0      ; yes, goto 2nd line
BNE   S5
S4:   ; fill 3rd and 4th lines with spaces
LDA   #$90      ; goto 3rd line
JSR   BLNKLN
LDA   #$D0      ; goto 4th line
JSR   BLNKLN
; move cursor to start of 3rd physical line
LDA   #$90
STA   CURSOR
STA   LCD0
JSR   LCDBUSY
RTS

BLNKLN:
; fill a 16-char line with spaces
STA   LCD0
JSR   LCDBUSY
LDX   #$10      ; line len is 16.
BL1:   LDA   #$20      ; write a space
STA   LCD1
JSR   LCDBUSY
DEX
BNE   BL1
RTS


ORG   $F8E0

PRINTCH:
; Display a char at the current cursor location.
;   80...8F 1st line
;   C0...CF 2nd line
;   90...9F 3rd line
;   D0...DF 4th line
CMP   #$0D      ; <CR> ?
BNE   P1      ; no
JMP   SCROLL      ; yes, just scroll and return
P1:   CMP   #$08      ; backspace ?
BNE   P2      ; no
JMP   BACKSPACE   ; yes, just BS and return
P2:   STA   LCD1      ; display it
JSR   LCDBUSY
; decide where the next char should go
LDX   CURSOR      ; get cursor location
INX         ; move cursor forward
CPX   #$E0      ; end of 4th line?
BNE   P4      ; no
DEX         ; yes, stay where we were
P4:   CPX   #$A0      ; end of 3rd line?
```

```
BNE   P3      ; no
LDX   #$D0      ; yes, begin 4th line
P3:   STX   CURSOR      ; remember new location
STX   LCD0      ; set the new location
JSR   LCDBUSY
RTS


ORG   $F90F

BACKSPACE:
; backup the cursor one space
LDX   CURSOR
CPX   #$D0      ; beginning of 4th line?
BNE   BS1      ; no
LDX   #$A0      ; yes, goto end of 3rd line
BS1:   CPX   #$90      ; beginning of 3rd line?
BEQ   BS2      ; yes, do nothing
DEX         ; no, backup one space
NOP
BS2:   STX   CURSOR      ; remember new location
STX   LCD0      ; set new location
JSR   LCDBUSY
LDA   #$20      ; print a space to overwrite old char
STA   LCD1
JSR   LCDBUSY
LDA   CURSOR      ; put cursor back on the space
STA   LCD0
JSR   LCDBUSY
RTS


ORG   $F950

DOKEY:
; handle an input char
PHA         ; save A
JSR   PRINTCH      ; print it
PLA         ; restore A
CMP   #$0D      ; <CR> ?
BNE   D1      ; no, we're done
JSR   PARSEK      ; yes, parse the input line
D1:   RTS

KBDINIT:
LDA     #$82            ; enable CA1 ints from 6522
; STA     IER
NOP
NOP
NOP
LDA     #$00
STA     DDRA            ; port A is input
STA     ORB             ; all 0's on output port
STA     INNDX           ; init inbuff index
LDA     #$FF
STA     DDRB            ; port B is output
RTS

CHKKBD:
```

```
; get the key (if pressed)
; the key is returned in A
; if no key pressed, return 00
LDY   #$FE      ; start with 11111110
C1:   STY   ORB      ; scan 1 of 8 rows
LDA   IRA      ; read the column
AND   #$07      ; lower 3 bits only
CMP   #$07      ; key ?
BNE   C2      ; yes, go decode it
TYA         ; no
BMI   C3      ; is it 01111111 yet?
LDA   #$00      ; yes, no key was pressed
RTS         ;   quit - return 00
C3:   SEC         ; no
ROL   A      ; shift left
TAY         ; put the pattern back in Y
CLC         ; go scan the next row
BCC   C1
C2:   STA   KEY      ; save the column info
TYA         ; get the row
LDY   #$00      ; init counter
C4:   INY         ; count
LSR   A      ; shift the row
BCS   C4      ; until the 0 falls out
TYA         ; get the row-count
ASL   A      ; move it to the higher bits
ASL   A
ASL   A
ORA   KEY      ; combine row and column
; decode the key value into ASCII
LDX   #$17
C6:   CMP   KEYTAB,X
BEQ   C5
DEX
BPL   C6
LDA   #$00
BEQ   C7
C5:   LDA   ASCTAB,X
C7:   STA   KEY      ; save the ASCII key
LDA   #$14      ; wait for debounce
JSR   DELAY
JSR   WAITNOKEY   ; wait for no key
LDA   #$30      ; wait for debounce
JSR   DELAY
JSR   WAITNOKEY   ; wait for no key
LDA   KEY      ; get the ASCII key char
RTS         ; return the char

WAITNOKEY:
; wait for no key being pressed
LDA   #$00
STA   ORB
W1:   LDA   IRA
AND   #$07
CMP   #$07
BNE   W1
RTS

KEYTAB:           ; keyboard codes
```

```
.BYTE   $0D
.BYTE   $3E
.BYTE   $3D
.BYTE   $26
.BYTE   $25
.BYTE   $36
.BYTE   $35
.BYTE   $2E
.BYTE   $2D
.BYTE   $46
.BYTE   $45
.BYTE   $16
.BYTE   $15
.BYTE   $1E
.BYTE   $1D
.BYTE   $0E
.BYTE   $23
.BYTE   $13
.BYTE   $2B
.BYTE   $0B
.BYTE   $33
.BYTE   $1B
.BYTE   $00   ; not used
.BYTE   $00   ; not used
ASCTAB:         ; ASCII codes
.BYTE   $30   ; 0
.BYTE   $31
.BYTE   $32
.BYTE   $33
.BYTE   $34
.BYTE   $35
.BYTE   $36
.BYTE   $37
.BYTE   $38
.BYTE   $39   ; 9
.BYTE   $41   ; A
.BYTE   $42
.BYTE   $43
.BYTE   $44
.BYTE   $45
.BYTE   $46   ; F
.BYTE   $00
.BYTE   $00
.BYTE   $00
.BYTE   $20   ; space
.BYTE   $08   ; backspace
.BYTE   $0D   ; <CR>
.BYTE   $00   ; not used
.BYTE   $00   ; not used



ORG   $FA30
PARSEK:
; Check the input line, see what to do.
; Process each token found on the line.
LDA   #$00      ; beginning of input line
STA   BUFFPTR      ; point to kybd buffer
STA   BUFFPTR+1
```

```
STA   TOKEN1
PK1:   JSR   GETTOKEN   ; get the next token
JSR   DOTOKEN      ; process it
BNE   PK1      ; more tokens?
RTS

NOP
NOP

NOP
NOP
NOP
NOP

GETTOKEN:
; identify the next token in the current buffer
; BUFFPTR points to the buffer
; max buffer length is 32.
; TOKEN1 points to first char in token.
; TOKEN2 points to first space after token.
LDY   TOKEN1      ; start from here
GTLOOP:   LDA   (BUFFPTR),Y   ; get a char
CMP   #$00      ; end of buffer marker?
BEQ   G3      ; yes - quit
CMP   #$20      ; space?
BNE   SYM      ; no - it's a token
INY         ; yes - skip it
CPY   #$20      ; end of buffer?
BNE   GTLOOP      ; no, loop
G3:   LDA   #$00      ; yes, report no more tokens
STA   TOKEN2
GTDONE:   RTS         ; return
SYM:   STY   TOKEN1      ; point to beginning of token
GTL2:   INY         ; next char
CPY   #$20      ; end of buffer?
BNE   G1      ; no
G2:   STY   TOKEN2      ; yes, point to end of token
RTS
G1:   LDA   (BUFFPTR),Y   ; get next char
CMP   #$20      ; space?
BNE   GTL2      ; no, loop
BEQ   G2      ; yes, quit

NOP
NOP
NOP
NOP
NOP
NOP

DOTOKEN:
; Process a token from the current buffer.
; Process each token based on it's length.
; Len of 1 is a 'kernel' command.
; Len of 3 is a command word.
; Len of 2 is a byte value.
; Len of 4 is a word value.
LDA   TOKEN2      ; was a token found?
BEQ   DT1      ; no, quit
```

```
SEC         ; yes, determine it's length
SBC   TOKEN1      ; subtract
STA   TKLEN      ; save the length
CMP   #$01
BNE   DT2
JSR   ONE
CLC
BCC   DT1
DT2:   LDA   TKLEN
CMP   #$02
BNE   DT3
JSR   TWO
CLC
BCC   DT1
DT3:   LDA   TKLEN
CMP   #$03
BNE   DT4
JSR   THREE
CLC
BCC   DT1
DT4:   LDA   TKLEN
CMP   #$04
BNE   DT5
JSR   FOUR
CLC
BCC   DT1
DT5:   LDA   #$00      ; unsupported token length
STA   TOKEN2
DT1:   LDA   TOKEN2
STA   TOKEN1
RTS

ONE:
; Is it a 'D'?
LDY   TOKEN1
LDA   (BUFFPTR),Y   ; get the letter
CMP   #'D
BEQ   DEPOSIT
CMP   #'E
BEQ   EXAMINE
RTS

TWO:
; push a byte onto the dstack
; convert ASCII byte to binary
LDY   TOKEN1      ; point to hi-order char
JSR   CVTBYT
JMP   PUSH      ; push onto dstack

THREE:
; It's a command 'word'. Execute it.
JMP   DOWORD

FOUR:
; Push a word onto the dstack. Do it hi-byte first,
; so that it can be used as a pointer directly.
LDY   TOKEN1
JSR   CVTBYT
NOP
```

```
NOP
NOP
NOP
JSR   PUSH
NOP
NOP
JSR   CVTBYT
JMP   PUSH      ; push lo byte

DEPOSIT:
JSR   POP
JSR   DUPW
JSR   WRITEA
LDA   #$01
JSR   DELAY
JSR   INCW
EXAMINE:
JSR   DUPW
JSR   PRINTW
JSR   READ
JSR   PRINTB
LDA   #'D
JSR   PRINTCH
; backup cursor 4 spaces
LDX   CURSOR
DEX
DEX
DEX
DEX
STX   CURSOR      ; remember new location
STX   LCD0      ; set new location
JSR   LCDBUSY
RTS

CVTBYT:
; Get an ASCII byte from the buffer, convert to binary.
LDA   (BUFFPTR),Y
JSR   BINBYT      ; convert to binary
ASL   A      ; move to upper nibble
ASL   A
ASL   A
ASL   A
STA   TEMP      ; save it
INY         ; get lo-order char
LDA   (BUFFPTR),Y
JSR   BINBYT      ; convert to binary
CLC         ; add upper nibble
ADC   TEMP
INY         ; point to next char
RTS

NOP
NOP
NOP
NOP
NOP
NOP
```

```
ORG   $FB80

PUSH:
; push A onto the dstack
LDX   DSP
STA   DSTACK,X
DEC   DSP
RTS

POP:
; pop A from the dstack
INC   DSP
LDX   DSP
LDA   DSTACK,X
RTS

INCW:
; Increment the word on the top of the stack,
; but don't pop it.
LDX   DSP
INX
INC   DSTACK,X
BNE   IW1
INX
INC   DSTACK,X
IW1:   RTS

WRITEA:
; Write A to the location indicated by the word
; on the top of the stack.
; Preserve A, pop the address.
PHA
LDA   #$8D
STA   ZPIND
JSR   POP
STA   ZPIND+1
JSR   POP
STA   ZPIND+2
LDA   #$60
STA   ZPIND+3
PLA
JSR   ZPIND
RTS

DUPW:
; Push a duplicate of the word that is already
; on the top of the stack.
; Preserve A.
PHA
LDY   DSP
INY
INY
LDA   DSTACK,Y
JSR   PUSH
DEY
LDA   DSTACK,Y
JSR   PUSH
PLA
RTS
```

```
READ:
; Read a memory location, specified by the addr
; on the stack. Pop the address, and push the
; result. Also return the result in A.
LDA   #$AD
STA   ZPIND
JSR   POP
STA   ZPIND+1
JSR   POP
STA   ZPIND+2
LDA   #$60
STA   ZPIND+3
JSR   ZPIND
JSR   PUSH
RTS

PRINTBNS:
; Print a byte value in hex at the
; current cursor location. Do NOT include a space.
JSR   POP
JSR   BYTEHEX
LDA   ASCBYT
JSR   PRINTCH
LDA   ASCBYT+1
JSR   PRINTCH
RTS

PRINTB:
; Print a byte value in hex at the
; current cursor location. Include a space.
JSR   PRINTBNS
LDA   #$20
JSR   PRINTCH
RTS

PRINTW:
; Print a word value in hex at the
; current cursor location. Include a space.
JSR   POP
PHA
JSR   PRINTBNS
PLA
JSR   PUSH
JSR   PRINTB
RTS

DOWORD:
; process a 3-letter word from the current buffer
; save the current BUFFPTR and TOKEN2
LDA   BUFFPTR
PHA
LDA   BUFFPTR+1
PHA
LDA   TOKEN2
PHA
; find the word's definition block
LDA   #$00      ; point to start of word defs
STA   EEPTR
```

```
LDA   #$C0
STA   EEPTR+1
DW3:   LDA   TOKEN1
STA   TKTMP
LDY   #$00      ; check block type
LDA   (EEPTR),Y
STA   BTYPE
BNE   DW1      ; deleted?
DW2:   LDA   EEPTR      ; yes - goto next block
CLC
ADC   #$20      ; blk size is 32.
STA   EEPTR
BCC   DW5
INC   EEPTR+1
LDA   EEPTR+1
CMP   #$D8      ; end of EE ?
BEQ   DWDONE
DW5:   CLC         ; loop back
BCC   DW3
DW1:   INY         ; check the symbol
LDA   (EEPTR),Y
STA   WLETT
TYA         ; save Y
TAX
LDY   TKTMP      ; get token letter
LDA   (BUFFPTR),Y
CMP   WLETT      ; letter match?
BNE   DW2      ; no - goto next block
TXA         ; restore Y
TAY
INC   TKTMP      ; yes - next letter
CPY   #$03      ; done?
BNE   DW1      ; no - next letter
; found matching word block!
LDA   BTYPE      ; check blk type
CMP   #$01      ; tokens?
BEQ   DW7      ; yes
CMP   #$02      ; code?
BEQ   DW4      ; yes
BNE   DWDONE      ; unknown - quit

NOP
NOP
DW7:
; found token block - point to 1st token
INY         ; point to space before token
INY         ; point to first letter of token
; process tokens
STY   TOKEN1
STY   TOKEN2
; set the BUFFPTR to match EEPTR
LDA   EEPTR
STA   BUFFPTR
LDA   EEPTR+1
STA   BUFFPTR+1
; recurse
DW6:   JSR   GETTOKEN   ; get a token
JSR   DOTOKEN      ; process it
BNE   DW6      ; more tokens?
```

```
DWDONE:
; restore the BUFFPTR and TOKEN1
PLA
STA   TOKEN2
STA   TOKEN1
PLA
STA   BUFFPTR+1
PLA
STA   BUFFPTR
RTS
DW4:   ; process code
LDA   #$4C      ; make a JMP in 0-page
STA   ZPIND
LDA   EEPTR      ; point to the block
CLC
ADC   #$04      ; the 4th byte is the start
STA   ZPIND+1
LDA   EEPTR+1
STA   ZPIND+2
JSR   ZPIND      ; make the call
CLC         ; done
BCC DWDONE

NOP
NOP
NOP
NOP
NOP
NOP

COMPILE:
; Compile a new word into EE. The definition is
; in (BUFFPTR)+TOKEN2 .
; find an unused or deleted block
LDA   #$00      ; EE starts at $C000
STA   COMPTR
LDA   #$C0
STA   COMPTR+1
COM1:   LDY   #$00      ; 1st byte of blk is type
LDA   (COMPTR),Y   ; get block type
CMP   #$01      ; token block?
BEQ   USEDBL      ; yes - find another
CMP   #$02      ; code block?
BEQ   USEDBL      ; yes - find another
; this block is free!
; (BUFFPTR)+TOKEN2 points to space before 1st token
LDA   #$00
STA   TKTMP
; (COMPTR)+TKTMP now points to block type
LDA   #$01      ; write the block type
JSR   EEWRT
COM2:   INC   TKTMP      ; next output index
INC   TOKEN2      ; next input index
LDY   TOKEN2      ; beyond end of input buffer?
CPY   #$20
BEQ   DONULL      ; yes
LDA   (BUFFPTR),Y   ; get a char from buffer
JSR   EEWRT      ; write the char
CLC         ; do next char
```

```
BCC   COM2
DONULL:   LDA   #$00      ; write a null terminator
JSR   EEWRT
COMDONE:
LDA   #$00      ; don't process rest of buffer
STA   TOKEN2
RTS         ; return
USEDBL:   LDA   COMPTR      ; goto next block
CLC
ADC   #$20      ; blk size is 32.
STA   COMPTR
BCC   COM1
INC   COMPTR+1
LDA   COMPTR+1
CMP   #$D8      ; limit of blocks ?
BEQ   COMDONE      ; yes - quit
BNE   COM1      ; no - try this block
EEWRT:   LDY   TKTMP      ; get index
STA   (COMPTR),Y   ; write the char
LDA   #$0D      ; wait 50ms for EE to finish
JSR   DELAY
RTS

ORG   $FDA0

BOOT:
;
; The Reset vector (FFFC) points here.
;
SEI                     ; no IRQ interrupts
CLD                     ; no Decimal mode
JSR   LCDINIT
JSR   KBDINIT
LDA   #$C0
STA     ARES            ; ACIA soft reset
LDA     #$1A            ; 8-N-1, 2400 baud
STA     ACTL
LDA   #$FF
STA   DSP
; LDA     #$89            ; enable ACIA xmtr/rcvr
; STA     ACMD
NOP
NOP
NOP
NOP
NOP
CLI         ; enable ints
; print "OK"
LDA   #$4F      ; 'O'
JSR   PRINTCH
LDA   #$4B      ; 'K'
JSR   PRINTCH
LDA   #$0D      ; <CR>
JSR   PRINTCH

; Go into the idle loop.
; Check the kbd, about 60 times/second.
; Any other work must be interrupt-driven.
IDLE:   JSR   CHKKBD
```

```
BEQ   IDLE      ; no key
JSR   DOKEY      ; handle the key
LDA   #$04      ; delay
JSR   DELAY
CLC         ; loop
BCC   IDLE



ORG   $FE00
BINBYT:
; Convert a single hex-ascii char to binary.
CMP     #$60
BMI     BB2
AND     #$DF
BB2:
SEC
SBC     #$30
CMP     #$11
BMI     BB1
SEC
SBC     #$07
BB1:
RTS

; Convert the value in A into two hex-ascii bytes, and store
; the two bytes in ASCBYT.
BYTEHEX:
PHA                     ; save the byte
LSR     A               ; do the high nibble first
LSR     A
LSR     A
LSR     A
CLC                     ; make it ascii
ADC     #$30
CMP     #$3A            ; if it's a letter, add a little more
BMI     BH3
CLC
ADC     #$07
BH3:
STA     ASCBYT        ; store the ascii byte
PLA                     ; get the original byte
AND     #$0F            ; do the low nibble
CLC                     ; make it ascii
ADC     #$30
CMP     #$3A
BMI     BH4
CLC
ADC     #$07
BH4:
STA     ASCBYT+1        ; store the 2nd ascii byte
RTS

; Time delay.
; Put the delay value in A. A value of 00 is 256 loops (max delay).
; With 256 loops, the number of machine cycles is: 329479+131072N,
; where N is the number of NOP instructions. If N is 5, and the
; clock is 1 Mhz, then the max delay is 0.985 seconds, or 3.85 ms
; per loop. If N is 12, and the clock is 1.8432 Mhz, then the max
```

```
; delay is 1.032 seconds, or 4.03 ms per loop.
; X and Y are preserved, Status is not.
; Location DLYCNT is used for temp storage.
DELAY:
STA DLYCNT
TXA             ; save X
PHA
LDX #$0 ; init X
DLOOP:
INX             ; waste time
NOP
NOP
NOP
NOP
NOP
BNE DLOOP       ; did X wrap?
DEC DLYCNT      ; yes, decrement counter
BNE DLOOP       ; are we done? No - loop again.
PLA             ; we're done - restore X
TAX
RTS             ; return

; Do longer delays, in multiples of about 1 second.
; Put the desired delay in A.
; A value of 00 will return immediately.
LDELAY:
BEQ LDELAY2     ; if 00, return right away
TAX             ; save the counter in X
LDELAY1:
LDA #$00        ; do a 1-second delay
JSR DELAY
DEX             ; DEC the counter
BNE LDELAY1     ; done?
LDELAY2:
RTS             ; yes - return

;
; NMI entry point.
;
NMI:
RTI

;
; IRQ entry point.
;
IRQ:
RTI         ; return from interrupt


; INTERRUPT AND RESTART VECTORS
ORG     $FFFA
.WORD   NMI             ; NMI vector
.WORD   BOOT            ; Cold start & reset vector
.WORD   IRQ             ; IRQ vector (and BRK vector)
.END    BOOT
```