```
;
;------------------------------------------------------------
; Program      : 6502 Simple Monitor
; Written by    : John Monahan
; Date Created  : 1/30/2012
; Description   : Very basic monitor for 6502 S-100 board
;
;       V1.0   1/30/2011      First initial version
;
;
;       'A=Memmap D=Disp   E=Echo   F=Fill   G=Goto'
;       'H=Math   I=Time   K=Menu   M=Move   O=Z80'
;       'Q=Port   S=Subs   T=Type   V=Verify @=Flush Printer'
;------------------------------------------------------------
;Commands follow the usual "Zapple" like commands
;Display memory              D123,567
;Move memory          M123,1003,4567
;Fill memory          F1234,4567,00
;Output to a port     QO01,33
;Query a port         QI01
;Hex Math             H456,123

;The input character numbers can range 4,3,2 or 1 characters but a CR is always required to execute the
;command.  All values are in HEX (upper or lower case), For "continous/repeat/abort" commands,
;the ESC breaks to main menu. Note this monitor assumes you are using a 65C02 and not the older
;6502 (which is missing a few opcodes)

;The only important hardware ports use in this monotor are for the Console I/O. I use our Propeller
;Drive Console IO board (Ports 0,1). However you can just splice in code for CONIN, CONOUT,
;CONSTAT for your needs.

;----------------------------------------------------------------------------------------------------
BELL           .EQU     $07
BLANK          .EQU     $20
CR             .EQU     $0D
LF             .EQU     $0A
ESC            .EQU     $1B
SPACE          .EQU    $20

;Base page for I/O on the S100Computers/N8VEM Propeller driven Console I/O Card

io             .EQU    $F000          ;This is the default IO page for the S100Computers 6502 CPU Card

CONDATA        .EQU    io+$01         ;Console Data port     (S-100 Propeller Console IO Board)
CONSTATUS      .EQU    io+$00         ;Consol Status port    (S-100 Propeller Console IO Board)
IOBYTE         .EQU    io+$EF         ;IOBYTE (Front panel)

;Base page for I/O on the S100Computers/N8VEM  Serial I/O Card

; PORT ASSIGNEMENT FOR DLP-USB Controller chip
USBD           .EQU    io+$AC         ; Adjust as necessary, also update Signon MSG below
USBS           .EQU    io+$AA         ; Status port for USB port (Port C of 8255, bits 6,7)
USBREAD        .EQU    $80            ; If Bit 7 = 0, data available to recieve by S-100 Computer
USBSEND        .EQU    $40            ; If Bit 6 = 0 data CAN be written for transmission to PC

; PORT ASSIGNMENTS OF THE ZILOG SCC CHIP
BCTL           .EQU    io+$A0         ; CHANNEL B CONTROL ;<--- Adjust as necessary,
ACTL           .EQU    io+$A1         ; CHANNEL A CONTROL ; also update Signon MSG below
BDATA          .EQU    io+$A2         ; CHANNEL B DATA
ADATA          .EQU    io+$A3         ; CHANNEL A DATA

; PORT ASSIGNMENTS FOR THE 8255
PORTA          .EQU    io+$A8         ;A port of 8255 ;<--- Adjust as necessary
PORTB          .EQU    io+$A9         ;B port of 8255
PORTC          .EQU    io+$AA         ;C Port of 8255
PORTCT         .EQU    io+$AB         ;8255 configuration port
AIBO           .EQU    %10011000      ;Set 8255 ports:- A input, B output, C(bits 0-3) output, (bits 4-7)input
AOBI           .EQU    %10001010      ;Set 8255 ports:- A output, B input, C(bits 0-3) output, (bits 4-7)input)

; My S-100 System hardware equates

SW6502         .EQU    io+$ED         ;INPUT FROM THIS PORT SWITCHES THE 6502 BACK to the Z80 in hardware
IO_PAGE        .EQU    $F0            ;Page location for I/O ports (not currently used)



;-------------------------- 6502 BASE PAGE LOCATION EQUATES ---------------------------------------

TEMP1          .EQU    $30            ;Move RAM etc (Word)
```

```
TEMP2          .EQU   $32            ;Move RAM etc (Word)
TEMP3          .equ   $34            ;Move RAM etc (Word)
TEMP4          .equ   $36            ;various uses (Word)
RESULT         .equ   $38            ;Results Byte flag
PREVIOUS_CHAR  .equ   $39            ;Store of previous typed keyboard character (Byte)
STR_POINTER    .equ   $3A            ;Store for pointer for all PRINT_STRING calls (Word)

;--------------------------------------------------------------------------------------------------------

; Initialize the hardware we are going to use for I/O

        *=     $F000

IO_PORTS: .FILL $100,0        ;Set asid for hardware I/O (Later move to F800H)

ENTRY:  SEI                   ;Disable interrupts (Note Start Of ROM Code)
        LDX    #$FF           ;Set stack pointer
        TXS                   ;to 0FFH

        LDA    #0             ;Clear RAM at 0000H (Useful for debugging only)
        TAY                   ;Fill first page with 0's
CLEAR2: STA    $0000,Y        ;Set pointer Y -> 0
        INY
        BNE    CLEAR2
                              ;Initilize the S100Computers/N8VEM Serial I/O board
IN8255  LDA    #%10001010     ;Setup 8255 as:- A input, B output, C(bits 0-3) output, (bits 4-7)input
        STA    PORTCT         ;OUT (PortCtrl_8255),A ;Config 8255 chip, Mode 0

INSCC:  LDX    #0             ;Initilize the two Zilog SCC's
SCC1:   LDA    SCCINIT,X
        STA    ACTL
        INX
        CPX    #$0E
        BNE    SCC1

        LDX    #0
SCC2:   LDA    SCCINIT,X
        STA    BCTL
        INX
        CPX    #$0E
        BNE    SCC2
                              ;Clear any contents waiting in the input buffer
USBCLR: BIT    USBS           ;Bit 7,6 get loaded directly from the address, in this case 8255 Port C
        BMI    BEGIN          ;If Bit 7 = 1 the buffer is empty
        LDA    USBD           ;Get the actual character from the buffer
        JMP    USBCLR

                              ;We now have the Serial I/O board initilized.
BEGIN:  LDA    MENU           ;<<<< Main Monitor Loop >>>>
        STA    STR_POINTER
        LDA    MENU+1
        STA    STR_POINTER+1
        JSR    PRINT_STRING   ;Print 0 terminated string
        JSR    CONIN
        JSR    TOUPPER        ;Convert to upper case
        JSR    CONOUT         ;Echo character
        SEC
        SBC    #'@'           ;Convert A,B,C.... to 0,1,2,3
        ASL    A              ;X2
        TAX                   ;Move to X
        LDA    MENU_TABLE,X
        STA    TEMP1
        INX
        LDA    MENU_TABLE,X
        STA    TEMP1+1
        JMP    (TEMP1)        ;<-- JUMP to Menu Routine Option. (Will always jump back to BEGIN)


;----------------------------- DISPLAY MEMORY MAP ----------------------------------------

RAM_MAP:JSR    CRLF           ;Print CR/LF ([A] is not destroyed)
        LDA    #0
        TAX                   ;Initialise the X count 0,1,2...255,0
        STA    TEMP1          ;Start at 0000H in RAM
        STA    TEMP1+1
        LDA    #16            ;16 Characrters per line
        STA    TEMP2
        JSR    SHOW_ADDRESS   ;Show Start Adderss (TEMP1)
```

```
MAP1:   LDA    (TEMP1)
        JSR    SHOW_TYPE      ;Show if RAM,Prom or Empty (R,P,.)
        INX                   ;Increase pointer for next time
        BEQ    MAP2           ;Loop back to zero, then done

        DEC    TEMP2          ;16 characters across
        BNE    NO_CRLF
        LDA    #CR
        JSR    CONOUT         ;Print CR/LF on Console
        LDA    #LF
        JSR    CONOUT         ;Print on Console
        LDA    #16            ;16 Characrters per line
        STA    TEMP2
        STX    TEMP1+1        ;Need to increment by 1
        JSR    SHOW_ADDRESS   ;Show Adderss (TEMP1)
        JMP    MAP1
NO_CRLF:STX    TEMP1+1        ;Increase TEMP1 pointer 255 bytes 0000H,0100H,0200H...FF00H
        JMP    MAP1
MAP2:   JMP    BEGIN          ;Back to main menu


SHOW_TYPE:                    ;Show if RAM,Prom or Empty (R,P,.)
        EOR    #$FF           ;Complement RAM value (6502 has no NOT opcode!)
        STA    (TEMP1)
        CMP    (TEMP1)        ;Did it flip
        BNE    NOT_RAM
        EOR    #$FF           ;Put back the origional RAM value
        STA    (TEMP1)
        LDA    #'R'
        JSR    CONOUT         ;Print on Console
        LDA    #' '
        JSR    CONOUT         ;Print on Console
        RTS
NOT_RAM:
        CMP    #$FF
        BNE    NOT_ROM        ;Assume if FF not RAM
        LDA    #'p'
        JSR    CONOUT         ;Print on Console
        LDA    #' '
        JSR    CONOUT
        RTS
NOT_ROM:
        LDA    #'.'
        JSR    CONOUT         ;Print on Console
        LDA    #' '
        JSR    CONOUT
        RTS


;------------------------- DISPLAY RAM (HEX or ASCII) -----------------------------------------
RAM_ASCII:
        LDA    #0
        STA    TEMP4          ;Flag to display ASCII
        BRA    DO_RAM
DISP_RAM:
        LDA    #$FF
        STA    TEMP4          ;Flag to display hex
DO_RAM: JSR    GET8DIGITS     ;Get 2X4 HEX digits and put in TEMP1 (start)& TEMP2 (end+1)
        JSR    WAIT_CR        ;Wait for a CR to be enterd
        INC    TEMP2          ;We need to go one past range for compare routine to work
        BNE    RAM3
        INC    TEMP2+1
RAM3:   JSR    CRLF
        JSR    CRLF
        JSR    SHOW_ADDRESS   ;Show Start Adderss (TEMP1)

        LDA    #32            ;32 Characrters per line
        STA    TEMP3
        LDA    TEMP1          ;May not be starting on an even boundry
        TAX                   ;Transfer count of bytes to display to X

RAM1:   LDA    TEMP4          ;Are we displaying Hex or ASCII values
        CMP    #$FF
        BEQ    RAM4
        LDA    (TEMP1)
        AND    #$7F
        CMP    #' '
        BCS    T33
T22:    LDA    #'.'
```

```
T33:    CMP     #$7C
        BCS     T22
        JSR     CONOUT
        BRA     RAM5
RAM4:   LDA     (TEMP1)         ;Get RAM Byte
        JSR     HEXOUT          ;Display Hex in [A]
RAM5:   JSR     INC_COMPARE     ;Increase TEMP1, then see if we are done yet
        LDA     RESULT          ;If TEMP1 = TEMP2, RESULT = 0 so done
        CMP     #0
        BEQ     RAM2            ;RESULT = 0, then done

        DEC     TEMP3           ;32 characters across
        BNE     RAM1
        JSR     CRLF            ;Print CR/LF on Console
        LDA     #32             ;32 Characrters per line
        STA     TEMP3
        JSR     SHOW_ADDRESS    ;Show Adderss in (TEMP1)
        JSR     PAUSE_CHECK     ;Check for an abort
        JMP     RAM1
RAM2:   JMP     BEGIN           ;Back to main menu

;------------------------------------------------------------------------------------------------

ECHO:   JSR     CRLF            ;Keyboard input check, echo any character from keyboard on Console
        JSR     CONIN
        CMP     #ESC
        BEQ     ECHO1
        JSR     CONOUT          ;Print ASCII
        PHA
        LDA     #' '
        JSR     CONOUT          ;Space
        PLA
        JSR     HEXOUT          ;Print Hex value
        BRA     ECHO
ECHO1:  JMP     BEGIN

;------------------------------ FILL RAM -------------------------------------------------------

FILL_RAM:
        JSR     GET8DIGITS      ;Get 2X4 HEX digits and put in TEMP1 (start)& TEMP2 (end)
        INC     TEMP2           ;We need to go one past range for compare routine to work
        BNE     FILL4
        INC     TEMP2+1
FILL4:  LDA     TEMP1           ;We need TEMP1 for GET2DIGITS
        STA     TEMP4           ;Tempory store in TEMP4
        LDA     PREVIOUS_CHAR   ;Was it less than 8 characters entered above
        CMP     #CR
        BEQ     FILL2
        JSR     CICO            ;If note check we get a ','
        CMP     #','
        BNE     FILL3
FILL2:  JSR     GET2DIGITS      ;Get fill byte, (in TEMP1)
        JSR     WAIT_CR         ;Wait for a CR to be enterd
        LDA     TEMP1
        STA     TEMP3
        LDA     TEMP4
        STA     TEMP1           ;Get back origional TEMP1

FILL1:  LDA     TEMP3           ;Get above fill character
        STA     (TEMP1)         ;Put fill character in RAM
        JSR     INC_COMPARE     ;Increase TEMP1, then see if we are done yet
        LDA     RESULT          ;If TEMP1 = TEMP2, RESULT = 0 so done
        CMP     #0
        BNE     FILL1           ;RESULT |= 0, then not done yet
        JMP     BEGIN           ;Back to main menu
FILL3:  JMP     BAD_CHAR

;--------------------------- MOVE RAM ---------------------------------------------------------

MOVE_RAM:
        JSR     GET8DIGITS      ;Get 2X4 HEX digits and put in TEMP1 (start)& TEMP2 (end)
        INC     TEMP2           ;We need to go one past range for compare routine to work
        BNE     MOVE6
        INC     TEMP2+1
MOVE6:  LDA     TEMP1           ;We need TEMP1 for GET2DIGITS
        STA     TEMP4           ;Tempory store in TEMP4
        LDA     TEMP1+1
        STA     TEMP4+1
        LDA     PREVIOUS_CHAR   ;Was it less than 8 characters entered above
        CMP     #CR
        BEQ     MOVE2
        JSR     CICO            ;If note check we get a ','
        CMP     #','
        BNE     MOVE3
MOVE2:  JSR     GET4DIGITS      ;Get destination address, (in TEMP1 + TEMP1+1)
        JSR     WAIT_CR         ;Wait for a CR to be enterd
        LDA     TEMP1
        STA     TEMP3           ;srore it in TEMP3 & TEMP3+1
        LDA     TEMP1+1
        STA     TEMP3+1
        LDA     TEMP4           ;Get back origional TEMP1
        STA     TEMP1
        LDA     TEMP4+1
        STA     TEMP1+1         ;Get back origional TEMP1 & TEMP1+1

MOVE1:  LDA     (TEMP1)         ;Get byte
        STA     (TEMP3)         ;Put at new location in RAM
        INC     TEMP3           ;We need to increase the destination address for next loop
        BNE     MOVE5
        INC     TEMP3+1
MOVE5:                          ;Check if (TEMP1) address = (TEMP2)
        JSR     INC_COMPARE     ;Increase TEMP1, then see if we are done yet
        LDA     RESULT          ;If TEMP1 = TEMP2, RESULT = 0 so done
        CMP     #0
        BNE     MOVE1           ;RESULT |= 0, then not done yet
        JMP     BEGIN           ;Back to main menu
MOVE3:  JMP     BAD_CHAR



;------------------------------ VERIFY TWO RAM AREAS HAVE SAME DATA ----------------------------------
VERIFY:
        JSR     GET8DIGITS      ;Get 2X4 HEX digits and put in TEMP1 (start)& TEMP2 (end)
        INC     TEMP2           ;We need to go one past range for compare routine to work
        BNE     VER6
        INC     TEMP2+1
VER6:   LDA     TEMP1           ;We need TEMP1 for GET2DIGITS
        STA     TEMP4           ;Tempory store in TEMP4
        LDA     TEMP1+1
        STA     TEMP4+1
        LDA     PREVIOUS_CHAR   ;Was it less than 8 characters entered above
        CMP     #CR
        BEQ     VER2
        JSR     CICO            ;If note check we get a ','
        CMP     #','
        BNE     VER3
VER2:   JSR     GET4DIGITS      ;Get Second start address, (in TEMP1 + TEMP1+1)
        JSR     WAIT_CR         ;Wait for a CR to be enterd
        LDA     TEMP1
        STA     TEMP3           ;srore it in TEMP3 & TEMP3+1
        LDA     TEMP1+1
        STA     TEMP3+1
        LDA     TEMP4           ;Get back origional TEMP1
        STA     TEMP1
        LDA     TEMP4+1
        STA     TEMP1+1         ;Get back origional TEMP1 & TEMP1+1
        JSR     CRLF

VER1:   LDA     (TEMP1)         ;Get byte
        CMP     (TEMP3)         ;Is it the same as the second location in RAM
        BNE     VER_ERROR
VER0:   INC     TEMP3           ;We need to increase the destination address for next loop
        BNE     VER5
        INC     TEMP3+1
                                ;Check if (TEMP1) address = (TEMP2)
VER5:   JSR     INC_COMPARE     ;Increase TEMP1, then see if we are done yet
        LDA     RESULT          ;If TEMP1 = TEMP2, RESULT = 0 so done
        CMP     #0
        BNE     VER1            ;RESULT != 0, then not done yet
        JMP     BEGIN           ;Back to main menu

VER3:   JMP     BAD_CHAR


VER_ERROR:
        LDA     V_ERR_MSG       ;"Error at location:'
        STA     STR_POINTER
        LDA     V_ERR_MSG+1
        STA     STR_POINTER+1
        JSR     PRINT_STRING

        JSR     SHOW_ADDRESS   ;Show Start Adderss (TEMP1)
        LDA     (TEMP1)
        JSR     HEXOUT
        LDA     #'h'
        JSR     CONOUT
        LDA     #'='
        JSR     CONOUT
        LDA     (TEMP1)
        JSR     BINOUT

        LDA     #'b'
        JSR     CONOUT
        LDA     #' '
        JSR     CONOUT
        LDA     #' '
        JSR     CONOUT
        LDA     #' '
        JSR     CONOUT

        LDA     TEMP1
        STA     TEMP4          ;Tempory Store here
        LDA     TEMP1+1
        STA     TEMP4+1

        LDA     TEMP3          ;Move TEMP3 address to TEMP1 for SHOW_ADDRESS
        STA     TEMP1
        LDA     TEMP3+1
        STA     TEMP1+1
        JSR     SHOW_ADDRESS   ;Show Start Adderss (TEMP1)
        LDA     (TEMP1)
        JSR     HEXOUT
        LDA     #'h'
        JSR     CONOUT
        LDA     #'='
        JSR     CONOUT
        LDA     (TEMP1)
        JSR     BINOUT
        LDA     #'b'
        JSR     CONOUT

        LDA     TEMP4          ;Restore start address
        STA     TEMP1
        LDA     TEMP4+1
        STA     TEMP1+1
        JMP     VER0



;--------------------------- SUBSTITUTE RAM -------------------------------------------------------

SUBSTITUTE:                    ;Substitute RAM values
        JSR     GET4DIGITS     ;Get the two hex numbers in (TEMP1,TEMP1+1)
SUBS3:  JSR     CRLF
        JSR     SHOW_ADDRESS   ;Show Start Adderss (TEMP1+1,TEMP1)
        LDA     #' '
        JSR     CONOUT
        LDA     (TEMP1)
        JSR     HEXOUT         ;show on console
        LDA     TEMP1          ;Tempory store TEMP1
        STA     TEMP3          ;in TEMP3
        LDA     #'-'
        JSR     CONOUT
        JSR     GET2DIGITS
        LDA     TEMP1          ;Put new byte in TEMP2
        STA     TEMP2
        LDA     TEMP3          ;Get back pointer TEMP1
        STA     TEMP1
        LDA     PREVIOUS_CHAR  ;Flag for less than 2 characters entered above
        CMP     #CR
        BEQ     SUBS1          ;If CR, then skip substitution
        LDA     TEMP2          ;Get new value
        STA     (TEMP1)        ;add the new value
SUBS1:  INC     TEMP1          ;Point to next RAM location
        BNE     SUBS2
        INC     TEMP1+1
SUBS2:  JMP     SUBS3          ;Continue until ESC is entered
```

```
;----------------------------- GOTO --------------------------------------------------------------

GOTO:   JSR     GET4DIGITS     ;Get address
        JSR     WAIT_CR        ;Wait for a CR to be enterd
        JMP     (TEMP1)

;--------------------------- 16 bit HEX MATH -----------------------------------------------------

MATH:   JSR     GET8DIGITS     ; and  (TEMP2,TEMP2+1)
        JSR     WAIT_CR        ;Wait for a CR to be enterd

        LDA     SUM_MSG        ;Pick up msg character (DIFF_MSG + 0 offset)
        STA     STR_POINTER
        LDA     SUM_MSG+1
        STA     STR_POINTER+1
        JSR     PRINT_STRING

        CLC                    ;Ensure carry is clear
         LDA    TEMP1          ;Add the two least significant bytes
         ADC    TEMP2
         STA    TEMP3
         LDA    TEMP1+1        ;Add the two most significant bytes
         ADC    TEMP2+1        ;... and any propagated carry bit
         STA    TEMP3+1
         JSR    HEXOUT         ;... and store the result
         LDA    TEMP3
         JSR    HEXOUT         ;... and show the result

        LDA     DIFF_MSG       ;Pick up msg character (DIFF_MSG + 0 offset)
        STA     STR_POINTER
        LDA     DIFF_MSG+1
        STA     STR_POINTER+1
        JSR     PRINT_STRING

        SEC                    ;Ensure carry is set
         LDA    TEMP1          ;Subtract the two least significant bytes
         SBC    TEMP2
         STA    TEMP3
         LDA    TEMP1+1        ;Subtract the two most significant bytes
         SBC    TEMP2+1        ;... and any propagated borrow bit
         JSR    HEXOUT         ;... and show the result
         LDA    TEMP3
         JSR    HEXOUT         ;... and show the result
        JMP     BEGIN

;------------- SWITCH CONTROL BACK TO Z80 (Master) ----------------------------------------------

Z80:    LDA     SW6502         ;This switches control back over to Z80
        nop
        nop
        nop
        nop
        nop
        JMP     BEGIN

;-------------------------------- QUERY I/O PORTS -------------------------------------------------

QUERY_PORTS:
        JSR     CONIN          ;Must be "I" or "O"
        JSR     TOUPPER        ;Convert to upper case
        JSR     CONOUT         ;Echo character
        CMP     #'I'
        BEQ     IN_PORTS       ;Query an input port
        CMP     #'O'
        BEQ     OUT_PORTS      ;Send data to Out Port
BAD_PORT:
        JMP     BAD_CHAR

OUT_PORTS:
        LDA     #IO_PAGE       ;I/O addresses are F800H-F8FFH
        STA     TEMP4+1
        JSR     GET2DIGITS     ;Get port number in TEMP1+1, value in TEMP
        LDA     TEMP1
        STA     TEMP4          ;Remember LSB then MSB for addressing
        LDA     PREVIOUS_CHAR  ;Was it less than 2 characters entered above
        CMP     #CR
        BEQ     OUT1
        JSR    CICO           ;If note check we get a ','
        CMP    #','
        BNE    BAD_PORT
OUT1:   JSR    GET2DIGITS     ;Get value in TEMP1
        JSR    WAIT_CR        ;Wait for a CR to be enterd
        LDA    TEMP1          ;Get value
        STA    (TEMP4)        ;Send to port
        JMP    BEGIN

IN_PORTS:
        LDA    #IO_PAGE       ;I/O addresses are F800H-F8FFH
        STA    TEMP4+1
        JSR    GET2DIGITS     ;Get port number in TEMP1+1, value in TEMP
        JSR    WAIT_CR        ;Wait for a CR to be enterd
        LDA    TEMP1
        STA    TEMP4          ;Remember LSB then MSB for addressing
IN1:    JSR    CRLF
        LDA    (TEMP4)        ;Get data from port
        JSR    HEXOUT         ;Write out hex value of port
        LDA    #' '
        JSR    CONOUT         ;Add two spaces
        LDA    #' '
        JSR    CONOUT
        LDA    (TEMP4)        ;Get data from port
        JSR    BINOUT         ;Write out binary data
        JMP    BEGIN

;------------------------------- K Command ------------------------------------------

KCMD:   LDA    SP_MENU        ;If speech synthesizer is active speak signon
        STA    STR_POINTER
        LDA    SP_MENU+1
        STA    STR_POINTER+1
        JSR    SPEAK_STRING   ;Speak 0 terminated string
        JMP    BEGIN



;-------------------------------------------------------------------------------------
NMI_VECTOR:                   ;Come here if an NMI interrupt
        PHA
        PHX
        PHY
        LDA    NMI_MSG        ;Pick up first character (NMI_MSG + 0 offset)
        STA    STR_POINTER
        LDA    NMI_MSG+1
        STA    STR_POINTER+1
        JSR    PRINT_STRING   ;Print 0 terminated string
        PLY
        PLX
        PLA
        RTI

;--------------------------------------------------------------------------------------
IRQ_VECTOR
        PHA                   ;Come here if an normal INT interrupt
        PHX
        PHY
        LDA    IRQ_MSG        ;Pick up first character (IRQ_MSG + 0 offset)
        STA    STR_POINTER
        LDA    IRQ_MSG+1
        STA    STR_POINTER+1
        JSR    PRINT_STRING   ;Print 0 terminated string
        PLY
        PLX
        PLA
        RTI


RAW_GETTIME:                  ;Not done yet, fall through
        NOP

TBD:    LDA    TBD_MSG        ;Pick up first character (MENU + 0 offset)
        STA    STR_POINTER
        LDA    TBD_MSG+1
        STA    STR_POINTER+1
        JSR    PRINT_STRING   ;Print 0 terminated string
        JMP    BEGIN
```

```
;-------------------------- SUPPORT ROUTINES -------------------------------------------------------
                              ;Print Character on Console
CONOUT: PHA                   ;Save character
CONOUT1:LDA    #%00000100
        AND    CONSTATUS      ;are we ready to output data
        BEQ    CONOUT1
        PLA                   ;get character
        STA    CONDATA        ;Send Character to port 01
        RTS

                              ;Get character from console
CONIN:  LDA    #%00000010
        AND    CONSTATUS      ;are we ready to input data
        BEQ    CONIN
        LDA    CONDATA        ;Get Character TEMP1 from port 01
        STA    PREVIOUS_CHAR  ;Several routines need to know this
        RTS

TOUPPER:CMP    #$40           ;LC->UC
        BCC    UPPER_DONE
        CMP    #$7B
        BCS    UPPER_DONE
        AND    #$5F
UPPER_DONE:
        RTS

                              ;Get console status
CONSTAT:LDA    #%00000010     ;Console Status, 0 = empty, FF = full
        AND    CONSTATUS      ;are we ready to input data
        BEQ    CON_EMPTY
        LDA    #$FF
        RTS
CON_EMPTY
        LDA    #$0
        RTS

                              ;Wait until a CR is enterd
WAIT_CR:LDA    PREVIOUS_CHAR  ;Was a CR previously entered
        CMP    #CR
        BEQ    CR_DONE
        JSR    CICO           ;If note check we get a ','
        CMP    #CR
        BEQ    CR_DONE
        JMP    BAD_CHAR
CR_DONE:RTS



SPEAKOUT:                     ;Speak text via Serial I/O board (if present)
        LDY    #0             ;Will try 256 times, then timeout
        PHA                   ;Save value in [A]
SPXXX:  LDA    BCTL           ;(A0), Is SCC TX Buffer empty
        AND    #$04
        BNE    SENDS          ;NZ if ready to recieve character
        DEY                   ;try 255 times
        BNE    SPXXX
        PLA
        RTS                   ;return if timeout
SENDS:  PLA
        STA    BDATA          ;(A2), Send it
        RTS


PRINT_STRING:                 ;Print string on console
STR2:   LDA    (STR_POINTER)  ;Pick up first character (String pointer)
        CMP    #0
        BEQ    STRING_DONE
        JSR    CONOUT
        INC    (STR_POINTER)
        BNE    STR2
        INC    (STR_POINTER+1)
        JMP    STR2
STRING_DONE:
        RTS


;SPEAKTOMM THIS IS A ROUTINE TO SEND A STRING TO TALKER [HL] AT STRING
SPEAK_STRING:
        PHX
        LDX    #0
SP2:    LDA    (SP_MENU),X    ;Pick up first character (SP_MENU + 0 offset)
        CMP    #0
        BEQ    SP1
        JSR    SPEAKOUT       ;Try sending to speaker
        INX
        JMP    SP2
SP1:    PLX
        RTS


CICO:   JSR    CONIN          ;CONSOLE INPUT WITH ECHO ON CONSOLE
        AND    #7FH           ;Characters 0-9, A-F, a-f
        BEQ    BAD_CHAR       ;No Nulls
        CMP    #','           ;Allow "," character
        BEQ    CIC1
        CMP    #CR            ;ACCEPT ONLY CR,LF,SP
        BEQ    CIC1
        CMP    #ESC           ;Also ESC
        BEQ    ENTRY1         ;Abort

        CMP    #'0'
        BCC    BAD_CHAR
        CMP    #':'           ;Allow 0-9
        BCC    CIC1
        CMP    #'A'
        BCC    BAD_CHAR       ;do not allow : to @
        CMP    #'G'           ;Is upper case A to F
        BCC    CIC1
        CMP    #'a'
        BCC    BAD_CHAR
        CMP    #'g'
        BCC    CIC1
        JMP    BAD_CHAR
CIC1:   JSR    CONOUT
        RTS

BAD_CHAR:
        LDA    BELL           ;SEND BELL TO INDICATE BAD DATA
        JSR    CONOUT
        LDA    #'?'           ;SEND ? TO INDICATE BAD DATA
        JSR    CONOUT
        JSR    CRLF
ENTRY1: JMP    ENTRY


                              ;Send CRLF to Console
CRLF:   PHA                   ;Save what is in [A]
        LDA    #CR
        JSR    CONOUT
        LDA    #LF
        JSR    CONOUT
        PLA
        RTS

HEXOUT: PHA                   ;SAVE ACC FOR USE LATER ON
         LSR   A              ;SHIFT H.O. NIBBLE
         LSR   A              ;DOWN TO THE L.O. NIBBLE
         LSR   A              ;CLEARING THE H.O. NIBBLE
         LSR   A
        AND    #$0F           ;PRINT L.O. NIBBLE AS A DIGIT
        JSR    HEX2ASC
         PLA                  ;GET ORIGINAL VALUE BACK
        AND    #$0F           ;PRINT L.O. NIBBLE AS A DIGIT
        JSR    HEX2ASC
        RTS
                              ;Convert a hex digit ($00-$0F) to ASCII ('0'-'9' or 'A'-'F')
HEX2ASC:ORA    #$30           ;Form the basic character code
         CMP   #$3A           ;Does the result need adjustment?
         BCC   HEX2A
         ADC   #$06           ;Add 6 (5 and the carry) if needed
HEX2A:   JSR   CONOUT
        RTS


BINOUT: PHA                   ;Print Binary bits in [A]
        PHX
        LDX    #8
```

```
BIN1:   ASL     A
        BCC     BIN2
        PHA
        LDA     #'1'
        JSR     CONOUT
        PLA
        BRA     BIN3
BIN2:   PHA
        LDA     #'0'
        JSR     CONOUT
        PLA
BIN3:   DEX
        BNE     BIN1
        PLX
        PLA
        RTS


SHOW_ADDRESS:                   ;Show 4 digit HEX value in TEMP1+1,TEMP1
        LDA     TEMP1+1
        JSR     HEXOUT
        LDA     TEMP1
        JSR     HEXOUT
        LDA     #' '
        JSR     CONOUT          ;Print on Console
        LDA     #' '
        JSR     CONOUT
        RTS


INC_COMPARE:                    ;First Increase TEMP1, then check if TEMP1 address = TEMP2
        INC     TEMP1
        BNE     COMPARE
        INC     TEMP1+1
COMPARE:                        ;Check if (TEMP1) address = (TEMP2)
        LDA     TEMP1+1
        CMP     TEMP2+1
        BNE     NO_MATCH
        LDA     TEMP1
        CMP     TEMP2
        BNE     NO_MATCH
        LDA     #$00
        STA     RESULT
        RTS
NO_MATCH:
        LDA     #$FF
        STA     RESULT
        RTS


PAUSE_CHECK:                    ;Check for an abort or pause on long window display
        JSR     CONSTAT         ;ESC aborts, any other key holds until another keypress
        CMP     #0
        BEQ     CHECK1          ;NZ if nothing
        LDA     CONDATA         ;Get keyboard character
        CMP     #ESC            ;ESC to abort
        BEQ     CHECK2
        JSR     CONIN
CHECK1: RTS                     ;Just Return
CHECK2: JMP     BEGIN

GET8DIGITS:                     ;Get start to (TEMP1,TEMP1+1) and finish (TEMP2,TEMP2+1) address
        JSR     GET4DIGITS
        LDA     TEMP1           ;Store in TEMP3
        STA     TEMP3
        LDA     TEMP1+1
        STA     TEMP3+1
        LDA     PREVIOUS_CHAR   ;Was less than 4 characters entered above
        CMP     #CR
        BEQ     RANGE1
        JSR     CICO            ;If note check we get a ','
        CMP     #','
        BNE     BAD_CHAR1
RANGE1: JSR     GET4DIGITS
        LDA     TEMP1           ;Store in TEMP2
        STA     TEMP2
        LDA     TEMP1+1
        STA     TEMP2+1
        LDA     TEMP3           ;TEMP3 -> TEMP1
        STA     TEMP1
        LDA     TEMP3+1
        STA     TEMP1+1
        RTS
BAD_CHAR1
        JMP     BAD_CHAR


GET4DIGITS:                     ;Get 0,1,2,3,4 HEX digits and put in (LSB)TEMP1 + (MSB)TEMP1+1
        LDA     #0
        STA     TEMP1           ;Default = 0,0
        STA     TEMP1+1         ;High byte in TEMP1+1

        JSR     CICO            ;Get First High Byte
        CMP     #','            ;Allow "," return with 0
        BEQ     GET4_ABORT
        CMP     #CR             ;Accept only CR, if CR return with 0
        BEQ     GET4_ABORT
        JSR     A2HEX
        STA     TEMP1           ;Remember MSB is last (done below)

        JSR     CICO            ;get second character/digit
        CMP     #','            ;Allow "," character
        BEQ     GET4_ABORT
        CMP     #CR             ;Accept only CR or ','
        BEQ     GET4_ABORT
        JSR     A2HEX
        ASL     TEMP1
        ASL     TEMP1
        ASL     TEMP1
        ASL     TEMP1           ;First digit is now shifted up
        ORA     TEMP1
        STA     TEMP1+1         ;Store it in TEMP+1

        JSR     CICO            ;Now second LOW byte
        CMP     #','            ;Allow "," return with 0
        BEQ     GET4_ABORT1
        CMP     #CR             ;Accept only CR, if CR return with 0
        BEQ     GET4_ABORT1
        JSR     A2HEX
        STA     TEMP1           ;Remember MSB is last (done below)

        JSR     CICO            ;Get second character/digit
        CMP     #','            ;Allow "," character
        BEQ     GET4_ABORT2
        CMP     #CR             ;Accept only CR or ','
        BEQ     GET4_ABORT2
        JSR     A2HEX
        ASL     TEMP1
        ASL     TEMP1
        ASL     TEMP1
        ASL     TEMP1           ;First digit is now shifted up
        ORA     TEMP1
        STA     TEMP1           ;Store it in TEMP+1
        RTS
GET4_ABORT:                     ;If CR etc. entered after 0 or 1 character
        LDA     #CR
        STA     PREVIOUS_CHAR   ;Flag for less than 4 characters entered above
        RTS
GET4_ABORT1:
        LDA     TEMP1+1         ;If CR etc. at this stage (2 digits) then shift to LSB
        STA     TEMP1
        LDA     #0
        STA     TEMP1+1
        LDA     #CR
        STA     PREVIOUS_CHAR   ;Flag for less than 4 characters entered above
        RTS                     ;Return with 00,xx
GET4_ABORT2:                    ;Abort after 3 digits need to adjust things
        LDA     TEMP1+1
        ASL     A               ;Need to shift things down one nibble
        ASL     A
        ASL     A
        ASL     A
        ORA     TEMP1
        STA     TEMP1
        LSR     TEMP1+1
        LSR     TEMP1+1
        LSR     TEMP1+1
        LSR     TEMP1+1
        LDA    #CR
        STA    PREVIOUS_CHAR  ;Flag for less than 4 characters entered above
        RTS                   ;Return with 0x,xx


GET2DIGITS:                   ;Get 0,1,2 HEX digits and put in TEMP1
        LDA    #0
        STA    TEMP1          ;Default = 0
        JSR    CICO
        CMP    #','           ;Allow "," return with 0
        BEQ    GET2_ABORT
        CMP    #CR            ;Accept only CR, if CR return with 0
        BEQ    GET2_ABORT
        JSR    A2HEX
        STA    TEMP1          ;Remember MSB is last (done below)

        JSR    CICO           ;get second character/digit
        CMP    #','           ;Allow "," character
        BEQ    GET2_ABORT
        CMP    #CR            ;Accept only CR or ','
        BEQ    GET2_ABORT
        JSR    A2HEX
        ASL    TEMP1
        ASL    TEMP1
        ASL    TEMP1
        ASL    TEMP1          ;First digit is now shifted up
        ORA    TEMP1
        STA    TEMP1
        RTS
GET2_ABORT:                   ;If CR etc. entered after 0 or 1 character
        LDA    #CR
        STA    PREVIOUS_CHAR  ;Flag for less than 2 characters entered above
        RTS

A2HEX:  SEC                   ;Convert ASCII to BIN
        SBC    #'0'
        CMP    #10
        BCC    A2HEX1
        SBC    #7
A2HEX1: RTS



;-------------------------- DATA AREA ------------------------------------------------------------

MENU           .WORD  $+2
               .byte   CR,LF,LF
               .byte   "S-100 6502 Monitor Version 1.0 (1/30/2011)"
               .byte  CR,LF
               .byte  "A=Memmap   D=Disp RAM E=Echo     F=Fill RAM   G=Goto RAM Address"
               .byte  CR,LF
               .byte  "H=Math     I=Time     K=Menu     M=Move RAM   O=Z80"
               .byte  CR,LF
               .byte  "Q=Port I/O S=Subs RAM T=Type RAM V=Verify RAM @=Flush Printer"
PROMPT         .WORD  $+2
               .byte  CR,LF,LF,'>',0
SP_MENU        .WORD  $+2
               .byte  "6502 Monitor Version 1.0",CR,0
SUM_MSG        .WORD  $+2
               .byte  CR,LF," Sum=",0
DIFF_MSG       .WORD  $+2
               .byte  " Difference=",0
IRQ_MSG        .WORD  $+2
               .byte  CR,LF,"IRQ",CR,LF,0
NMI_MSG        .WORD  $+2
               .byte  CR,LF,"NMI",CR,LF,0
V_ERR_MSG      .WORD  $+2
               .byte  CR,LF,"Error at: ",0
TBD_MSG        .WORD  $+2
               .byte  CR,LF,"Code not yet done",0

MENU_TABLE     .EQU   $
               .word  TBD            ;@             ;Flush Printer
               .word  RAM_MAP        ;A             ;Display Memory Map
               .word  TBD            ;B             ;Set Console output to Propeller or CGA/VGA Video board
               .word  TBD            ;C             ;
               .word  DISP_RAM        ;D            ;Display Memory contents (Read RAM in Bytes)
               .word  ECHO           ;E             ;Show keyboard character typed
               .word  FILL_RAM       ;F             ;Fill memory contents
               .word   GOTO           ;G             ;Jump to an address location
               .word   MATH           ;H             ;Add & Subtract two Hex numbers
               .word   RAW_GETTIME    ;I             ;Put CMOS-RTC Time & Date on CRT
               .word   TBD            ;J             ;Test RAM
               .word   KCMD           ;K             ;Display this menu & speech
               .word   TBD            ;L             ;
               .word   MOVE_RAM       ;M             ;Move memory
               .word   TBD            ;N             ;Sub-menu to test/diagnose IDE Board
               .word   Z80            ;O             :Return back to Z80 master
               .word   TBD            ;P             ;LOAD OS from HDISK
               .word   QUERY_PORTS    ;Q             ;Query In or Out to a port
               .word   TBD            ;R             ;Display all active 6502 INPUT ports
               .word   SUBSTITUTE     ;S             ;Substitute byte values in RAM
               .word   RAM_ASCII      ;T             ;Display Memory contents in ASCII
               .word   TBD            ;U             ;
               .word   VERIFY         ;V             ;Verify two memory regions are the same
               .word   TBD            ;W             ;Jump to exactly 500H in RAM
               .word   TBD            ;X             ;
               .word   TBD            ;Y             ;
               .word   TBD            ;Z             ;

;Table of values to initilize the two Zilog SCC. Note the SCC is set here for 9600 BAUD

SCCINIT        .byte   $04            ;DB 04H ;Point to WR4
               .byte   $44            ;DB 44H ;X16 clock,1 Stop,NP
;
               .byte   $03            ;DB 03H ;Point to WR3
               .byte   $C1            ;DB 0C1H ;Enable reciever, No Auto Enable, Recieve 8 bits
;              .byte   $E1            ;DB 0E1H ;Enable reciever, Auto Enable, Recieve 8 bits (for CTS bit)
;
               .byte   $05            ;DB 05H ;Point to WR5
               .byte   $EA            ;DB 0EAH ;Enable, Transmit 8 bits
;
;                                     ;Set RTS,DTR, Enable
;
               .byte   $0B            ;DB 0BH ;Point to WR11
               .byte   $56            ;DB 56H ;Recieve/transmit clock = BRG

               .byte   $0C            ;DB 0CH ;Point to WR12
;              .byte   $40            ;DB 40H ;Low Byte 2400 Baud
;              .byte   $1E            ;DB 1EH ;Low Byte 4800 Baud
;              .byte   $0E            ;DB 0EH ;Low Byte 9600 Baud
               .byte   $06            ;DB 06H ;Low byte 19,200 Baud
;              .byte   $02            ;DB 02H ;Low byte 38,400 Baud
;              .byte   $00            ;DB 00H ;Low byte 76,800 Baud
;
               .byte   $0D            ;DB 0DH ;Point to WR13
               .byte   $00            ;DB 00H ;High byte for Baud
;
               .byte   $0E            ;DB 0EH ;Point to WR14
               .byte   $01            ;DB 01H ;Use 4.9152 MHz Clock. Note SD Systems uses a 2.4576 MHz clock, enable
BRG
;
               .byte   $0F            ;DB 0FH ;Point to WR15
               .byte   $00            ;DB 00H ;Generate Int with CTS going high

;-----------------------------------------------------------------------------------------------
; Set the Reset vectors for 6502 system
       *=      $FFFA
       .word   ENTRY   ;NMI_VECTOR (board giving false ints!)       ;FFFA   (NMI)
       .word   ENTRY                                         ;FFFC  (Reset)
       .word   ENTRY   ;IRQ_VECTOR                           ;FFFE  (IRQ)
       .end
```