```
; PDP_MON.mac   by John Monahan (S100Computers.com)
;
; Assembled with AsmPDP.exe (windows) assembler.
; Note code cannot contain BASIC names like "PRINT" (Because the compiler was written in basic).
; Best to use lower case labels.
;
; Remember port I/O's are 8 bits wide. If you send a word to an odd port you will trigger a CPU error/exception.
; Use MOVB x,y and not MOV x,y
;
; V0.1          1/11/2017           ;In initial code
; V0.11           3/4/2017          ;Run on V0.8 prototype board
; V0.12             3/4/2017                ;Corrected byte span routines for M,V, F etc and speech synthesis.
; V0.13             3/14/2017               ;Added XModem file download capability
; V1.0          4/3/2017                ;Corrected Serial port initilization for speech synthesizer
;
;
ODT_CONIN_STAT:         equ   &3FFF70           ;&o177560
ODT_CONIN_DATA:         equ   &3FFF72           ;&o177562
ODT_CONOUT_STAT:  equ   &3FFF74           ;&o177564
ODT_CONOUT_DATA:  equ   &3FFF76           ;&o177566
BIT7:                   equ     &80             ;&o200
BIT4:                   equ     &10
BIT2:                   equ     &04
BIT1:                   equ     &02
BIT0:                   equ     &01

S100_CONIN_STAT:  equ   &E000       ;S100Computers Console IO Board In Status
S100_CONIN_DATA:  equ   &E001       ;S100Computers Console IO Board In Data
S100_CONOUT_STAT: equ   &E000       ;S100Computers Console IO Board Out Status
S100_CONOUT_DATA: equ   &E001       ;S100Computers Console IO Board Out Data

BCTL:                   equ     &E0A0           ;S100Computers Serial board speaker B CTL port (Zilog SCC Chip)
BDTA:                   equ     &E0A2           ;S100Computers Speaker B data port

SW86:                   equ     &E0ED           ;Input from this port switches the PDP back to the Z80 in hardware
SW86_TM:                equ     &E0EE           ;Output 00H switch teh PDP back to Z80 Hardware (on SMB V2,V3 boards)

IOBYTE:                 equ     &E0EF           ;S100Computers SMB IOBYTE Port

BP_SOH:                 EQU    &0
BP_BLK_NO:              EQU    &2              ;BP Offset for Recieved Sector Number for XModem
BP_INV_BLK_NO:          EQU    &4
BP_SECT_NO:             EQU    &6              ;BP Offset for CURRENT SECTOR NUMBER
BP_CKSUM:           EQU    &8
BP_TIMEOUT:             EQU    &A
```

```
FiveSeconds:              EQU    &10              ; Try Modem input for a max of 5 seconds

CR:               equ    &0D
LF:               equ    &0A
ESC:              equ    &1B
SPACE:                   equ     &20
SCROLL:                  EQU     &01             ; Set scrool direction UP.
BELL:                    EQU     &07
BS:                      EQU     &08
TAB:                     EQU     &09             ; TAB ACROSS (8 SPACES FOR SD-BOARD)
FF:                      EQU     &0C
DELETE_CHAR:             EQU     &7F
BACKS:                   EQU     &08

SOH:                     EQU     &1              ; For Modem etc.
EOT:                     EQU     &4
ACK:                     EQU     &6
NAK:                     EQU     &15


ROMS:                    equ     TRUE    ; Set to FALSE for test program running at 100H in RAM (400 Octal)
TEST:                    equ     FALSE   ; Normally FALSE, If TRUE just 3's are sent console

#if       ROMS
          ORG &C000                      ; Location of default onboard ROMS (&o140000)
#else
          ORG &100                       ; Locate at 400 Octal for testing (Above PDP11 traps etc)
#endif

#if     TEST
SSS:  MOV   #&33,R0                   ; Basic I/O test - only
      MOVB    R0,@#ODT_CONOUT_DATA    ; Send ASCII to ODT Console out port (Note, No status out check)
      MOVB    R0,@#S100_CONOUT_DATA   ; Send ASCII to S100Computers Console IO Board (Note, No status out check)
      BR    SSS
#endif

Start:  MOV     #&BFF0,SP               ; Setup stack at BFF0H (for now, below ROM ORG at C000H)
        MOV     #Signon,R5              ; Point to Signon Message
        JSR     PC,PrStr                ; Print string
        MOV     SP,R5                   ; Show current SP
        JSR     PC,PutWord_R5
        MOV     #Signon1,R5             ; Point to Signon Message finish
        JSR     PC,PrStr                ; Print string
        MOV     #MainMenu,R5            ; Point to Main Menu
        JSR     PC,PrStr                ; Print string
```

```
        JSR     PC,SetupIntVectors     ; Setup Trap vector pointers in low RAM (0-FFH)

Loop:   JSR     PC,ConCRLF             ; Show CR,LF
        MOV     #&3E,R0                ; Print '>'
        JSR     PC,CONSOLE_OUT
        JSR     PC,CONSOLE_IN        ; Get a menu character (WITH ECHO) to R0
        JSR     PC,ToUpper             ; a-z to A-Z
        JSR     PC,CONSOLE_OUT             ; Echo
        CMPB    #&41,R0
        BNE     NotA
        JMP     MemMap                 ; "A",  CR,LF,Memory map
NotA:   CMPB    #&43,R0
        BNE     NotC
        JMP     XModem                 ; "C",  File Download into RAM from PC
NotC:   CMPB    #&44,R0
        BNE     NotD
        JMP     DisplayRAM             ; "D",  Display RAM
NotD:   CMPB    #&45,R0
        BNE     NotE
        JMP     Echo                   ; "E",  Echo
NotE:   CMPB    #&46,R0
        BNE     NotF
        JMP     FillRAMB                ; "F",  Fill RAM (Bytes)
NotF:   CMPB    #&48,R0
        BNE     NotH
        JMP     FillRAMW                ; "H",  Fill RAM (Words)
NotH:   CMPB    #&4D,R0
        BNE     NotM
        JMP     MoveRAM                 ; "M",  Move RAM
NotM:   CMPB    #&49,R0
        BNE     NotI
        JMP     IOByte                  ; "I",  IO Byte
NotI:   CMPB    #&56',R0
        BNE     NotV
        JMP     VerifyRAM               ; "V",  Verify RAM
NotV:   CMPB    #&53,R0
        BNE     NotS
        JMP     SubsRAMB                ; "Y",  Subs RAM (Byte)
NotS:   CMPB    #&59,R0
        BNE     NotY
        JMP     SubsRAMW                ; "Y",  Subs RAM (Word)
NotY:   CMPB    #&51,R0
        BNE     NotQ
        JMP     QPorts                  ; "Q",  Query Ports
```

```
NotQ:   CMPB    #&54,R0
        BNE     NotT
        JMP     ARAM                    ; "T",  Ascii in RAM
NotT:   CMPB    #&55,R0
        BNE     NotU
        JMP     Talk                    ; "U" speaker test
NotU:   CMPB    #&4B,R0
        BNE     NotK
        JMP     DisplayMenu             ; "K",  CR,LF,Display Menu
NotK:   CMPB    #&4C,R0
        BNE     NotL
        JMP     IO_Test             ; "L" Rapid port output test to console
NotL:   CMPB    #&5A,R0
        BNE     NotZ
        JMP     ReturnZ80               ; "Z",  CR,LF,Return to Z80
NotZ: MOV     #NotDoneMsg,R5           ; Point to CMD not done yet Message
        JSR     PC,PrStr                ; Print string
LoopDone:
        BR      Loop


;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> MENU COMMAND ROUTINES <<<<<<<<<<<<<<<<<<<<<<<<<<

MemMap:                             ; Menu A command
        MOV     #MM_Text,R5             ; Point to MemMap Message
        JSR     PC,PrStr                ; Print string
        MOV     #0,R5                   ; Set up start address
        MOV     #16,R4                  ; Characters across count
Map1:   JSR     PC,PutWord_R5           ; Print HEX word value in R5
Map2:   MOV     #SPACE,R0
        JSR     PC,CONSOLE_OUT          ; Send a space
        MOV     (R5),R0
        CMP     #&FFFF,R0           ; Is it RAM
        BNE     NotEmpty
        MOV     #&2E,R0                 ; No RAM or ROM
        JSR     PC,CONSOLE_OUT          ; Send a '.'
        BR      Map3
NotEmpty:
        MOV    (R5),R0                  ;Bring in the data
        MOV     R0,R1                   ;Store it
        SWAB    R0
        MOV     R0,(R5)
        CMP     R0,(R5)                 ;Did if flip
        BNE     IsROM                   ;Must be ROM
        MOV     R1,(R5)                 ;Put back the origional data
```

```
        MOV     #&52,R0
        JSR     PC,CONSOLE_OUT          ; Send a 'R'
        BR      Map3
IsROM:  MOV     #&70,R0
        JSR     PC,CONSOLE_OUT          ; Send a 'p'

Map3:   ADD     #&100,R5                ; Next 100H bytes
        DEC     R4                      ; Count characters across
        BNE     Map2
        JSR     PC,ConCRLF              ; Show CR,LF
        MOV     #16,R4                  ; Characters across count
        CMP     #0,R5                   ; Done yet, will wrap around
        BNE     Map1
      JMP   Loop

DisplayMenu:
        JSR     PC,ConCRLF              ; Show CR,LF
        MOV     #MainMenu,R5            ; Point to Main Menu
        JSR     PC,PrStr                ; Print string
      JMP   Loop

DisplayRAM:                             ; Menu D command
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     DisplayError            ; Abort if ESC returned
        MOV     R5,R3                   ; Store start location in R3
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     DisplayError            ; Abort if ESC returned
        CMPB    #CR,R0
        BNE     DisplayError            ; Abort if not CR set
        MOV     R5,R4                   ; Store end location in R4

        CMP     R3,R4
        BEQ     DisplayError            ; If the same abort
RAM1:   JSR     PC,ConCheckESC          ; Was the ESC key pressed
        CMPB    #ESC,R0
        BEQ     DisplayError
        JSR     PC,ConCRLF              ; Show CR,LF
        MOV     R3,R5
        JSR     PC,PutWord_R5
        JSR     PC,ConSPACE2          ; Send 2 SPACEs
        MOV     #16,R1                  ; Bytes across count
RAM2:   MOVB    (R3)+,R5
        JSR     PC,PutByte_R5
```

```
        JSR     PC,ConSPACE1          ; Send 1 SPACE
        DECB    R1
        CMP     #0,R1
        BNE     RAM2
        CMP     R4,R3               ;Are we there yet (note unsigned compare)
        BHI    RAM1
DisplayError:
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop

ARAM:                           ; Menu T command (Display ASCII in RAM)
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     AError                  ; Abort if ESC returned
        MOV     R5,R3                   ; Store start location in R3
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     AError                  ; Abort if ESC returned
        CMPB    #CR,R0
        BNE     AError                  ; Abort if not CR set
        MOV     R5,R4                   ; Store end location in R4

        CMP     R3,R4
        BEQ     AError                  ; If the same abort
ARAM1:  JSR     PC,ConCheckESC          ; Was the ESC key pressed
        CMPB    #ESC,R0
        BEQ     AError
        JSR     PC,ConCRLF              ; Show CR,LF
        MOV     R3,R5
        JSR     PC,PutWord_R5
        JSR     PC,ConSPACE2          ; Send 2 SPACEs
        MOV     #64,R1                  ; Bytes across count (This will be a minimum!)
ARAM2:  MOVB    (R3)+,R0
        CMP     #&20,R0
        BGE     ARAM3
        CMP     #&7f,R0
        BEQ     ARAM3
ARAM5:  JSR     PC,CONSOLE_OUT
        BR      ARAM4
ARAM3:  MOVB    #&2E,R0                  ; Use '.' for non text characters
        BR      ARAM5
ARAM4:  DECB    R1
        CMP     #0,R1
        BNE     ARAM2
        CMP     R4,R3               ;Are we there yet (note unsigned compare)
```

```
        BHI     ARAM1
AError:
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop


Echo:                                   ; Menu E command
        MOV     #Echo_Text,R5           ; Point to MemMap Message
        JSR     PC,PrStr                ; Print string
Echo1:  JSR     PC,CONSOLE_IN
        CMPB    #ESC,R0                 ; Was an abort requested
        BEQ     EchoDone
        JSR     PC,CONSOLE_OUT          ; Echo data
        BR      Echo1
EchoDone:
        JMP     Loop


FillRAMW:                               ; Menu H command (Fill RAM, words)
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     FillError               ; Abort if ESC returned
        MOV     R5,R3               ; Store start location in R3
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     FillError               ; Abort if ESC returned
        MOV     R5,R4                   ; Store end location in R4
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     FillError               ; Abort if ESC returned
        MOV     R5,R2                   ; Store Fill word in R2

        CMP     R3,R4
        BEQ     FillError               ; If the same abort
Fill2:  MOV     R2,(R3)+                ; Fill one word (only)
        CMP     R4,R3               ;Are we there yet (note unsigned compare)
        BHI     Fill2
FillError:
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop

FillRAMB:                               ; Menu F command (Fill RAM, bytes)
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     FillError               ; Abort if ESC returned
```

```
        MOV     R5,R3               ; Store start location in R3
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     FillError               ; Abort if ESC returned
        MOV     R5,R4                   ; Store end location in R4
        JSR     PC,GetByte_R5
        CMPB    #ESC,R0
        BEQ     FillError               ; Abort if ESC returned
        MOVB    R5,R2                   ; Store Fill word in R2

        CMP     R3,R4
        BEQ     FillError               ; If the same abort
Fill3:  MOVB    R2,(R3)+            ; Fill one byte (only)
        CMP     R4,R3               ;Are we there yet (note unsigned compare)
        BHI     Fill3
        BR      FillError


MoveRAM:                                ; Menu M command
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     MoveError               ; Abort if ESC returned
        MOV     R5,R3               ; Store start location in R3
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     MoveError               ; Abort if ESC returned
        MOV     R5,R4                   ; Store end location in R4
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     MoveError               ; Abort if ESC returned
        MOV     R5,R2                   ; Store new location in R2

        CMP     R3,R4
        BEQ     MoveError               ; If the same abort
        CMP     R3,R2
        BEQ     MoveError               ; If the same abort
Move2:  MOVB    (R3)+,(R2)+             ;Move one byte at a time
        CMP     R4,R3               ;Are we there yet (note unsigned compare)
        BHI     Move2
MoveError:
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop

VerifyRAM:                              ; Menu M command
        JSR     PC,GetWord_R5
```

```
        CMPB    #ESC,R0
        BIC     #&01,R5                 ; Round  down to even boundry
        MOV     R5,R3                ; Store start location in R3
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     VerifyError             ; Abort if ESC returned
        MOV     R5,R4                   ; Store end location in R4
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     VerifyError             ; Abort if ESC returned
        MOV     R5,R2                   ; Store new location in R2

        CMP     R3,R4
        BEQ     VerifyError             ; If the same abort
        CMP     R3,R2
        BEQ     VerifyError             ; If the same abort
Ver2:   JSR     PC,ConCheckESC          ; Was the ESC key pressed
        CMPB    #ESC,R0
        BEQ     VerifyError
        CMPB    (R3)+,(R2)+
        BNE     MisMatch
Ver3:   CMP     R4,R3               ;Are we there yet (note unsigned compare)
        BHI     Ver2
VerifyError:
        JSR     PC,ConCRLF               ; Show CR,LF
        JMP     Loop

MisMatch:
        MOV     #VerMsg0,R5              ; Print "Mismatch found at "
        JSR     PC,PrStr                 ; Print string
        DEC     R3
        MOV     R3,R5
        INC     R3
        JSR     PC,PutWord_R5
        MOV     #VerMsg3,R5              ; Print "H,CR,LF"
        JSR     PC,PrStr                 ; Print string
        JMP     Ver3                 ; Go to next byte



SubsRAMB:                               ; Menu S command (Subs RAM Bytes)
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     SubsErrorB              ; Abort if ESC returned
        MOV     R5,R3                ; Store start location in R3
```

```
SubsB1: MOV     #&8,R4                  ; Store char count in R4
        JSR     PC,ConCRLF              ; Show CR,LF
        MOV     R3,R5
        JSR     PC,PutWord_R5       ; Show current location
        JSR     PC,ConSPACE1
SubsB2: MOVB    (R3),R5
        JSR     PC,PutByte_R5           ; Show current value
        JSR     PC,ConSPACE1
        JSR     PC,GetByte_R5           ; Get new value (Byte)
        CMPB    #ESC,R0
        BEQ     SubsErrorB              ; Abort if ESC returned
        CMPB    #SPACE,R0               ; Continue if a space
        BEQ     SubsB3
        CMPB    #CR,R0                  ; Continue if a CR
        BNE     SubsB4
SubsB3: JSR     PC,ConSPACE1
SubsB4: MOVB    R5,(R3)+
        DEC     R4
        BEQ     SubsB1
        BR      SubsB2
SubsErrorB:
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop



SubsRAMW:                               ; Menu S command (Subs RAM Words)
        JSR     PC,GetWord_R5
        CMPB    #ESC,R0
        BEQ     SubsErrorW              ; Abort if ESC returned
        MOV     R5,R3               ; Store start location in R3
SubsW1: MOV     #&8,R4                  ; Store char count in R4
        JSR     PC,ConCRLF              ; Show CR,LF
        MOV     R3,R5
        JSR     PC,PutWord_R5       ; Show current location
        JSR     PC,ConSPACE1
SubsW2: MOV     (R3),R5
        JSR     PC,PutWord_R5           ; Show current value
        JSR     PC,ConSPACE1
        JSR     PC,GetWord_R5           ; Get new value (word)
        CMPB    #ESC,R0
        BEQ     SubsErrorW              ; Abort if ESC returned
        CMPB    #SPACE,R0               ; Continue if a space
        BEQ     SubsW3
        CMPB    #CR,R0                  ; Continue if a CR
```

```
        BNE     SubsW3
        JSR     PC,ConSPACE1
SubsW3: MOV     R5,(R3)+
        DEC     R4
        BEQ     SubsW1
        BR      SubsW2
SubsErrorW:
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop


QPorts:                                 ; Menu Q command
        JSR     PC,CONSOLE_IN           ; Get a menu character (WITH ECHO) to R0
        JSR     PC,ToUpper              ; a-z to A-Z
        JSR     PC,CONSOLE_OUT          ; Echo
        CMPB    #&49,R0                 ; 'I'
        BEQ     InPort
        CMPB    #&4F,R0                 ; 'O'
        BEQ     OutPort
        MOV     #&3F,R0                 ; '?'
        JSR     PC,CONSOLE_OUT
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop

InPort:
        JSR     PC,GetByte_R5           ; Get port number
        CMPB    #ESC,R0
        BEQ     PortError               ; Abort if ESC returned
        ADD     #&E000,R5           ; Move up to I/O space
        MOVB    (R5),R4                 ; In Port to R4 reg
        MOV     #VerMsg2,R5             ; Print " = "
        JSR     PC,PrStr                ; Print string
        MOV     R4,R5               ; Get back port #
        JSR     PC,PutByte_R5
        MOV     #VerMsg3,R5             ; Print "H ",CR,LF
        JSR     PC,PrStr                ; Print string
        JMP     Loop
PortError:
        JSR     PC,ConCRLF              ; Show CR,LF
        JMP     Loop

OutPort:
        JSR     PC,GetByte_R5           ; Get port number
        CMPB    #ESC,R0
        BEQ     PortError               ; Abort if ESC returned
```

```
        ADD     #&E000,R5           ; Move up to I/O space
        MOV     R5,R4                   ; Store Out Port in R4
        MOV     #&2C,R0                 ; ','
        JSR     PC,CONSOLE_OUT
        JSR     PC,GetByte_R5           ; Get Byte value to be sent to port (R4)
        CMPB    #ESC,R0
        BEQ     PortError               ; Abort if ESC returned
        MOVB    R5,(R4)
        JMP     Loop


Talk:                                   ; Menu 'U' command (Speaker systhesis test)
        MOV     #WillSpeak,R5        ; Show speech string to be sent
        JSR     PC,PrStr                ; Print string
        JSR     PC,InitSerialB          ; Initilize serial port B
        MOV     #BCTL,R5             ; Note all ports are E000H + Port#
        MOVB    (R5),R0
;       CMPB    #&C5,R0              ; Check port is valid (Possibly other values are also fine, you can comment out if
necessary)
;       BNE     TalkError
        MOV     #TestSpeak,R5        ; Pick up test speech string in R5
        JSR     PC,SpeakStr             ; Talk string
        JMP     Loop
TalkError:
        MOV     #BadPort,R5             ; Print "Port Inactive "
        JSR     PC,PrStr                ; Print string
        JMP     Loop

ReturnZ80:                      ; Menu 'Z' Command. (Return to Z80)
        MOV     #Z80Msg,R5          ; Show going back to Z80
        JSR     PC,PrStr                ; Print string
        MOVB    @#(SW86),R0         ; This switches control back over to Z80 (Old SMB)
        MOVB    #0,R0                 ; Or reset TMA-O back to Z80 control. (New V3 SMB)
        MOVB    R0,@#(SW86_TM)
        MOV     #10000,R0
Z80A:   DEC     R0                  ; Slight delay -- just in case
        CMPB    R0,0
        BNE     Z80A
        JMP     Loop

IOByte:                                 ; Menu 'I' Command (Display IOByte)
        MOV     #IOByteMsg,R5       ; Show IO Byte Msg
        JSR     PC,PrStr                ; Print string
        MOVB    @#(IOBYTE),R5
        JSR     PC,PutBits_R5
```

```
        JSR     PC,ConCRLF
        JMP     Loop


IO_Test:                                ; Menu 'L' Command (Send ASCII cgaracters to S100 Console)
        MOV     #IOTestMsg,R5       ; "Send character test string to port 01H, 80 times"
        JSR     PC,PrStr                ; Print string

        MOV     #IOTestMsg,R5       ; "Send character test string to port 01H, 80 times"
        JSR     PC,S100_PrStr           ; Print string on S100 console
        MOV     #81,R4              ; Do 80 lines
IO_Test1:
        DEC     R4
        BEQ     IO_Test_Done
        MOV     #TestStr,R5         ; Point to string
IO_Test2:
        MOVB    (R5)+,R0
        CMPB    R0,#0
        BEQ     IO_Test1
        JSR     PC,S100_CONSOLE_OUT ; Send directly to S100 Propeller Console I/O board (character in R0)
      JMP   IO_Test2
IO_Test_Done:
        MOV     #CMD_Done,R5        ; "Test Done"
        JSR     PC,PrStr                ; Print string
      JMP   Loop

; ----------------------------- XMODRM FILE DOWNLOAD -----------------------------------------------

XModem:                                 ; Note this is the converted 8086 Monitor code
        MOV     #XModemMsg,R5       ; "Load a File from a PC...."
        JSR     PC,PrStr                ; Print string
        MOV     SP,R4               ; R4 will be the Base Pointer (BP of 8086 code)
        SUB     #100,R4                 ; Drop it well below the stack for now
        MOVB    #1,BP_SECT_NO(R4)       ; Current sector/Blk

        MOV     #RAMStart,R5        ; "Enter destination in RAM for data (up to 4 digits):"
        JSR     PC,PrStr                ; Print string
        JSR     PC,GetWord_R5           ; Put Start Address in R3
        MOV     R5,R3                   ; Store in R3
        CMPB    #ESC,R0
        BNE     XModem1                 ; Abort if ESC returned
        JMP     Loop

ShowBlkInfo:
        MOV     #RMSGMsg,R5              ; "WAITING FOR SECTOR #
```

```
        JSR     PC,PrStr                ; Print string
        MOVB    BP_SECT_NO(R4),R5       ; Current sector/Blk#
        JSR     PC,PutByte_R5
        MOV     #RAMMsg,R5              ; "H.  If OK will write to RAM location "
        JSR     PC,PrStr                ; Print string
        MOV     R3,R5
        JSR     PC,PutWord_R5           ; Load Address in R3
        RTS     PC                      ; Return

XModem1:
        JSR     PC,ConCRLF
        JSR     PC,ShowBlkInfo
XModemLoop:
        MOVB    #0,BP_SOH(R4)           ; SOH store
        MOVB    #0,BP_BLK_NO(R4)        ; Current Block store
        MOVB    #0,BP_INV_BLK_NO(R4)    ; Inverted Current Block store
        MOVB    #0,BP_CKSUM(R4)
        MOVB    #&80,R2                 ; 128 Byte Blocks

        JSR     PC,XMODEM_GET_CHARACTER
        BCS     XModemTimeout
        CMPB    #SOH,R0
        BEQ     GOT_SOH
        CMPB    #EOT,R0
        BNE     BAD_SOH
        JMP     GOT_EOT                 ;All Done get out of loop
GOT_SOH:
        JSR     PC,XMODEM_GET_CHARACTER;
        MOVB    R0,BP_BLK_NO(R4)        ; Store BLK#
        JSR     PC,XMODEM_GET_CHARACTER;
        MOVB    R0,BP_INV_BLK_NO(R4)    ; Store Inverted BLK#
 XBlock:
        JSR     PC,XMODEM_GET_CHARACTER  ; Get 128 byte Block
        MOVB    R0,(R3)+
        DECB    R2
        BNE     XBlock

        JSR     PC,XMODEM_GET_CHKSUM    ; Get File Checksum
        CMPB    R0,BP_CKSUM(R4)
        BNE     BAD_CKSUM

        MOVB    BP_INV_BLK_NO(R4),R0
        MOVB    BP_BLK_NO(R4),R1
        ADD     R1,R0
        CMPB    #&FF,R0
```

```
        BNE     BAD_BLK

        INCB    BP_SECT_NO(R4)          ; Point to next BLK#
        JSR PC,ShowBlkInfo          ; Update Screen Info

        MOV     #ACK,R0
        JSR     PC,XMODEM_SEND_CHARACTER ; Send Back Block OK Acknowledge
        JMP     XModemLoop              ; Get next block until EOT signal

XModemTimeout:
        MOV     #TimeOutMsg,R5          ; "Timeout"
        JSR     PC,PrStr                ; Print string
        JMP     Loop

BAD_SOH:
        MOV     #NOSOH,R5               ; "Did not get SOH"
        JSR     PC,PrStr                ; Print string
        JMP     Loop

BAD_BLK:
        MOV     #XERR2,R5               ; "Bad BLK # in Header"
        JSR     PC,PrStr                ; Print string
        JMP     Loop

BAD_CKSUM:
        MOV     #XERR3,R5               ; "Bad Checksum for Sector"
        JSR     PC,PrStr                ; Print string
        JMP     Loop

GOT_EOT:
        MOV     #ACK,R0
        JSR     PC,XMODEM_SEND_CHARACTER ; Send Back Block OK Acknowledge to close out sender
        MOV     #TRANS_DONE,R5          ; "Data Transfer Is Complete",CR,LF,LF,0
        JSR     PC,PrStr                 ; Print string
        JMP     Loop


          ;-------------------------------------------------
          ;   XMODEM SERIAL PORT GET CHARACTER ROUTINE
          ;-------------------------------------------------

XMODEM_GET_CHARACTER:
        MOV     #FiveSeconds,R1
        MOV     #0,R0
XMODEM_IN1:
```

```
        BITB    #BIT7,@#ODT_CONIN_STAT      ; Check bit-7/ready of xmt status reg
        BNE     XMODEM_IN2
        DEC     R0
        BNE     XMODEM_IN1
        DEC     R1
        BNE     XMODEM_IN1
        SEC                                 ; Timeout Set Carry
        RTS     PC                          ; Return

XMODEM_IN2:
        MOVB    @#ODT_CONIN_DATA,R0      ; ASCII to R0 reg
        ADD     R0,BP_CKSUM(R4)             ; Add in to checksum
        CLC                                 ; Carry clear indicating all OK
        RTS     PC

XMODEM_GET_CHKSUM:
        MOV     #FiveSeconds,R1
        MOV     #0,R0
XMODEM_IN3:
        BITB    #BIT7,@#ODT_CONIN_STAT      ; Check bit-7/ready of xmt status reg
        BNE     XMODEM_IN4
        DEC R0
        BNE     XMODEM_IN3
        DEC     R1
        BNE     XMODEM_IN3
        SEC                                 ; Timeout Set Carry
        RTS     PC                          ; Return

XMODEM_IN4:
        MOVB    @#ODT_CONIN_DATA,R0      ; Checksuum in R0 reg
        CLC                                 ; Carry clear indicating all OK
        RTS     PC

          ;-----------------------------------------------------
          ;   XMODEM SERIAL PORT SEND CHARACTER ROUTINE
          ;-----------------------------------------------------

XMODEM_SEND_CHARACTER:                  ;CHECK IF MONITORING OUTPUT
        ADD     R0,BP_CKSUM(R4)         ;CALC CKSUM NO MATTER WHAT
        JSR     PC,ODT_CONSOLE_OUT
        RTS     PC


;>>>>>>>>>>>>>>>>>>> HIGH LEVEL SUPPORT ROUTINES <<<<<<<<<<<<<<
```

```
SpeakStr:                   ;Send speech data (in R5) to port up to end with '0'
        MOV     #BDTA,R3            ;Point to serial data port
        MOV     #BCTL,R4            ;Point to serial status port
Speak0: MOV     #100,R1         ; Retry count
Speak1: MOVB    (R4),R0
        BIT     #BIT2,R0          ; Test bit-2/ready
        BNE     Speak2              ; busy-loop while bit-7 is 0
        DEC     R1
        BEQ     TalkError1
        BR      Speak1          ;Try 100 times
Speak2: MOVB    (R5)+,R0            ;get a character
        CMPB    #0,R0           ;Finish if terminating 0
        BEQ     SpeakDone
        CMPB    #&20,R0         ;Skip < SPACE
        BGT     Speak2
        CMPB    #&7F,R0         ;Skip < DEL
        BEQ     Speak2
        MOV     #5000,R1        ;Not clear why I need this delay. I think my code above for
Speak3: DEC     R1              ;checking the status port is incorrect (Note also had a problem with this
        BNE     Speak3          ;in the Z80 monitor. (See SPEAKER_STS: in that code).
        MOVB    R0,(R3)         ;Send Character to talker data port
        BR      Speak0          ;Next Character
SpeakDone:
        MOVB    #&0D,(R3)       ;Must end with a 0D Character to data port for speaker to actully speak
        RTS     PC              ;Routine return

TalkError1:
        MOV     #TimeoutPort,R5         ;Print "Talk Port Timeout "
        JSR     PC,PrStr            ;Print string
        RTS     PC              ;Routine return


InitSerialB:                        ;Initilize the S100 Computers serial board speaker port
        MOVB    #BCTL,R1            ;Serial board speaker CTL port (Zilog SCC Chip)
        MOVB    #&04,(R1)           ;Point to WR4
        MOVB    #&44,(R1)           ;X16 clock,1 Stop,NP
        MOVB    #&03,(R1)           ;Point to WR3
        MOVB    #&C1,(R1)           ;Enable reciever, Auto Enable, Recieve 8 bits
        MOVB    #&05,(R1)           ;Point to WR5
        MOVB    #&EA,(R1)           ;Enable, Transmit 8 bits
        MOVB    #&0B,(R1)           ;Set RTS,DTR, Enable. Point to WR11
        MOVB    #&56,(R1)           ;Recieve/transmit clock = BRG
        MOVB    #&0C,(R1)           ;Point to WR12
;       MOVB    #&02,(R1)           ;Low byte 38,400 Baud <<<<<<<<<<< for XModem I/O (Port A)
      MOVB    #&06,(R1)       ;Low byte 19,200 Baud <<<<<<<<<<< Speech chip speed
```

```
        MOVB    #&0D,(R1)               ;Point to WR13
        MOVB    #&00,(R1)               ;High byte for Baud
        MOVB    #&0E,(R1)               ;Point to WR14
        MOVB    #&01,(R1)               ;Use 4.9152 MHz Clock.
        MOVB    #&0F,(R1)               ;Point to WR15
        MOVB    #&00,(R1)               ;Generate Int. with CTS going high
        RTS     PC                  ;Return

                                ;Note currently Serial port A is NOT initilized.

SetupIntVectors:                        ; Setup low RAM interrupt/trap vector pointers
      MOV   0,R5                    ; Setup low RAM Interrupt & Trap vectors
Trap1:      MOV   #CatchAllRoutine,(R5)+
        CMP     #&100,R5                ; Fill 0-FFH with catch-all trap routine
        BHI     Trap1

        MOV     #&4,R5                  ; Go back and fill in special cases
        MOV     #AbortRoutine,(R5)
        MOV     #&8,R5
        MOV     #IllegalOpcodeRoutine,(R5)
        MOV     #&1C,R5
        MOV     #TrapRoutine,(R5)
        MOV     #&40,R5
        MOV     #EventRoutine,(R5)
        MOV     #&A0,R5
        MOV     #PIRRoutine,(R5)
        MOV     #&A4,R5
        MOV     #FPERoutine,(R5)
        RTS     PC

CatchAllRoutine:
        MOV     R5,-(SP)
        MOV     R0,-(SP)
        MOV     #CatchAllMsg,R5         ; Point to Catch All Message
        JSR     PC,PrStr                ; Print string
        MOV     (SP)+,R0
        MOV     (SP)+,R5
        RTI

AbortRoutine:
        MOV     R5,-(SP)
        MOV     R0,-(SP)
        MOV     #AbortMsg,R5            ; Point to Abort Message
        JSR     PC,PrStr                ; Print string
        MOV     (SP)+,R0
```

```
        MOV     (SP)+,R5
        RTI
IllegalOpcodeRoutine:
        MOV     R5,-(SP)
        MOV     R0,-(SP)
        MOV     #OPCodeMsg,R5          ; Point to Abort Message
        JSR     PC,PrStr               ; Print string
        MOV     (SP)+,R0
        MOV     (SP)+,R5
        RTI
TrapRoutine:
        MOV     R5,-(SP)
        MOV     R0,-(SP)
        MOV     #TrapMsg,R5            ; Point to Abort Message
        JSR     PC,PrStr               ; Print string
        MOV     (SP)+,R0
        MOV     (SP)+,R5
        RTI
EventRoutine:
        MOV     R5,-(SP)
        MOV     R0,-(SP)
        MOV     #EventMsg,R5           ; Point to Abort Message
        JSR     PC,PrStr               ; Print string
        MOV     (SP)+,R0
        MOV     (SP)+,R5
        RTI
PIRRoutine:
        MOV     R5,-(SP)
        MOV     R0,-(SP)
        MOV     #PIRMsg,R5             ; Point to Abort Message
        JSR     PC,PrStr               ; Print string
        MOV     (SP)+,R0
        MOV     (SP)+,R5
        RTI
FPERoutine:
        MOV     R5,-(SP)
        MOV     R0,-(SP)
        MOV     #FPEMsg,R5             ; Point to Abort Message
        JSR     PC,PrStr               ; Print string
        MOV     (SP)+,R0
        MOV     (SP)+,R5
        RTI


;--------------------------- ROUTINES TO GET AND PUT AND SHOW BYTE/WORD VALUES FOR REGESTERS  -------------------
```

```
GetWord_R5:                 ;Get a WORD HEX value from Console to R5
        CLR     R0
        CLR     R5
GetWord1:
        JSR     PC,GetNibble_R5
        CMPB    #CR,R0
        BEQ     GetWordDone
        CMPB    #SPACE,R0
        BEQ     GetWordDone
        CMPB    #ESC,R0
        BEQ     GetWordDone
        BR      GetWord1            ; Note will return the last 4 valid hex characters
GetWordDone:                        ; up to a CR, ',' or SPACE
        RTS     PC                ; 0 in R0 if all OK, ESC if invalid character


GetByte_R5:                 ;Get a BYTE HEX value from Console to R5
        CLR     R0
        CLR     R5
        JSR     PC,GetNibble_R5
        CMPB    #CR,R0
        BEQ     GetByteDone
        CMPB    #SPACE,R0
        BEQ     GetByteDone
        CMPB    #ESC,R0
        BEQ     GetByteDone
        JSR     PC,GetNibble_R5
        CMPB    #ESC,R0
        BEQ     GetByteDone         ; Abort if ESC returned
        JSR     PC,CONSOLE_IN       ; Character to R0
GetByteDone:
        RTS     PC                ;N clear if all OK, N set if invalid character



GetNibble_R5:                       ;Get a NIBBLE HEX CHARACTER from Console to R5
        CLR     R0
        JSR     PC,CONSOLE_IN
        CMPB    #ESC,R0         ;Was an abort requested
        BEQ     NibbleError
        CMPB    #CR,R0
        BEQ     NibbleCR
        CMPB    #SPACE,R0
        BEQ     NibbleSPACE
```

```
        CMPB    #&2C,R0                 ;Check for ','
        BEQ     NibbleCOMMA
        JSR     PC,ToUpper
        JSR     PC,CONSOLE_OUT          ;Echo data
        SUB     #&30,R0                 ;'0' to 0, '1' to 1...
        BLT     NibbleError
        CMPB    #&09,R0
        BGE     NibbleDone
        SUB     #&7,R0
        CMPB    #&0F,R0                 ;Greater than 'F' in invalid
        BLT     NibbleError
NibbleDone:
        ASL     R5                  ;Shift R5 up a nibble
        ASL     R5
        ASL     R5
        ASL     R5
        BIS     R0,R5                   ;OR in the high nibble of what will be the high byte
        CLR     R0
        RTS     PC                  ;Return with Nibble in R5, 0 in R0

NibbleCR:
        MOV     #CR,R0
        RTS     PC                  ;Return with CR in R0

NibbleSPACE:
        JSR     PC,CONSOLE_OUT      ;Echo data
        MOV     #SPACE,R0
        RTS     PC                  ;Return with SPACE in R0

NibbleCOMMA:
        JSR     PC,CONSOLE_OUT      ;Echo data
        MOV     #CR,R0
        RTS     PC                  ;Return with CR in R0

NibbleError:
        MOV     #&3F,R0                 ;Send "?'
        JSR     PC,CONSOLE_OUT
        MOV     #ESC,R0
        RTS     PC                  ;Return with ESC in R0


PutWord_R5:                             ;Print HEX Word value in R5
        SWAB    R5                      ;Value in R5 is retained
        JSR     PC,PutByte_R5           ;Print HEX byte value in R5
        SWAB    R5
```

```
        JSR     PC,PutByte_R5
        RTS     PC

PutByte_R5:                             ;Print HEX Byte value in R5
        CLR     R0                      ;Clear R0
        MOVB    R5,R0                   ;Origional number to R0
        ROR     R0                      ;Shift upper byte to lower 4 bits
        ROR     R0
        ROR     R0
        ROR     R0                      ;Upper nibble is now lower nibble
        JSR     PC,HexOut
        MOV     R5,R0
        JSR     PC,HexOut
        RTS     PC

HexOut:                                 ;Print Hex Byte value in R0
        BICB    #&F0,R0                 ;Clear upper nibble
        ADD     #&30,R0                 ;CONVERT TO ASCII
        CMPB    #&39,R0                 ;See if > 9
        BGE     HexOK
        ADD     #&07,R0                 ;Add to make 10=A, 11=B...
HexOK:  JSR     PC,CONSOLE_OUT
        RTS     PC


PutBits_R5:                     ;Print 8 bits in R5.
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R4,-(SP)
        MOVB    #8,R2                   ;8 bits across
        MOVB    #&80,R3                 ;test bit
        MOVB    R5,R4
BitTst: BITB    R3,R4
        BEQ     Bit8L
        MOVB    #&31,R0                     ;'1'
        BR      Bits8
Bit8L:  MOVB    #&30,R0                     ;'0'
Bits8:  JSR     PC,CONSOLE_OUT
        CLC
        RORB    R3                      ;Shift test bit right one bit
        DECB    R2
        BNE     BitTst
        MOV     (SP)+,R4
        MOV     (SP)+,R3
        MOV     (SP)+,R2
```

```
        RTS     PC


ToUpper:                                ;>>> ;a-z to A-Z
        CMP     #&61,R0                 ; less than 'a'
        BGT     SkipToU
        CMP     #&7A,R0                 ; Greater tha 'z'
        BLT     SkipToU
        SUB     #&20,R0                 ; Adjust
SkipToU:
        RTS     PC


; >>>>>>>>>>>>>>>>>>>> ODT & S100 BUS CONSOLE I/O ROUTINES <<<<<<<<<<<<<<<<<<<<<<<

PrStr:
        MOVB    (R5)+,R0                ; >>> Print String Routine to ODT (or S100 Console)
        BEQ     PrStr1
        JSR     PC,CONSOLE_OUT
        BR      PrStr
PrStr1: RTS     PC                      ; return

S100_PrStr:
        MOVB    (R5)+,R0                ; >>> Print String Routine direct to S100 Console
        BEQ     PrStr2
        JSR     PC,S100_CONSOLE_OUT
        BR  S100_PrStr
PrStr2: RTS     PC                      ; return


ConSPACE2:                              ; >>> Send SPACE+SPACE to ODT or S100 console
        MOVB    #SPACE,R0
        JSR     PC,CONSOLE_OUT
ConSPACE1:
        MOVB    #SPACE,R0
        JSR     PC,CONSOLE_OUT
        RTS     PC                      ; Note R0 contains ASCII ctaracter (as a Byte)


ConCRLF:                                ; >>> Send CR+LF+> to ODT or S100 console
        MOVB    #CR,R0
        JSR     PC,CONSOLE_OUT
ConCRLF1:
        MOVB    #LF,R0
        JSR     PC,CONSOLE_OUT
```

```
        RTS     PC                      ; Note R0 contains ASCII ctaracter (as a Byte)

;-----------------------------------------------------------------------------------------------------
ConCheckESC:                            ; Check if ESC key was pressed on console
        BITB    #BIT0,@#(ODT_CONOUT_STAT)   ; See if output goes to S100 Bus
        BEQ     S100_ConCheckESC
        BR      ODT_ConCheckESC             ; if not send to the default ODT routine in CPU

S100_ConCheckESC:
        BITB    #BIT1,@#S100_CONIN_STAT     ; Check bit-1/ready of Propeller board Console In port (0H)
        BEQ     S100_NoESC                  ; Nothing there while bit-1 is 0
        MOVB    @#S100_CONIN_DATA,R0        ; ASCII to R0 reg
        CMPB    #ESC,R0
        BNE     S100_NoESC
        SEC                         ; Set Carry Flag if ESC
        RTS     PC                          ; Return with ESC in R0
S100_NoESC:
        CLC                         ; Return with Carry flag cleared
        RTS     PC                          ; Return

ODT_ConCheckESC:
        BITB    #BIT7,@#ODT_CONIN_STAT        ; Check bit-7/ready of xmt status reg
        BEQ     ODT_NoESC                   ; busy-loop while bit-7 is 0
        MOVB    @#ODT_CONIN_DATA,R0         ; ASCII to R0 reg
        CMPB    #ESC,R0
        BNE     ODT_NoESC
        SEC                         ; Set Carry Flag if ESC
        RTS     PC                          ; Return with ESC in R0
ODT_NoESC:
        CLC                         ; Return with Carry flag cleared
        RTS     PC                          ; Return

;-----------------------------------------------------------------------------------------------------
CONSOLE_OUT:                            ; >>> MAIN Console output routine <<<<
        BITB    #BIT0,@#(ODT_CONOUT_STAT)   ; See if output goes to S100 Bus
        BEQ     S100_CONSOLE_OUT
        BR      ODT_CONSOLE_OUT             ; if not send to the default ODT routine in CPU

S100_CONSOLE_OUT:                       ; S100 Bus Console output routine <<<<
        BITB    #BIT2,@#S100_CONOUT_STAT    ; Check bit-2/ready of Propeller board Console Out port (0H)
        BEQ     S100_CONSOLE_OUT        ; busy-loop while bit-2 is 0
        MOVB    R0,@#S100_CONOUT_DATA       ; Send ASCII to Propeller board Console Out port (01H)
        RTS     PC                          ; Note R0 contains ASCII ctaracter (as a Byte)

ODT_CONSOLE_OUT:                        ; ODT Console Out Routine
```

```
        BITB    #BIT7,@#ODT_CONOUT_STAT       ; Check bit-7/ready of xmt status reg
        BEQ     ODT_CONSOLE_OUT             ; busy-loop while bit-7 is 0
        MOVB    R0,@#ODT_CONOUT_DATA        ; send ASCII to xmt data reg
        RTS     PC                          ; Note R0 is still valid (as a Byte)


;-----------------------------------------------------------------------------------------------
CONSOLE_IN:                                 ; >>> MAIN Console input routine <<<<
        BITB    #BIT0,@#ODT_CONOUT_STAT   ; See if output goes to S100 Bus
        BEQ     S100_CONSOLE_IN
        BR      ODT_CONSOLE_IN              ; if not send to the default ODT routine in CPU

S100_CONSOLE_IN:                            ; S100 Bus Console input routine <<<<
        BITB    #BIT1,@#S100_CONIN_STAT     ; Check bit-1/ready of Propeller board Console In port (0H)
        BEQ     S100_CONSOLE_IN             ; Nothing there while bit-1 is 0
        MOVB    @#S100_CONIN_DATA,R0        ; Get ASCII from Propeller board Console In port (01H)
        RTS     PC                          ; Note R0 contains ASCII ctaracter (as a Byte)

ODT_CONSOLE_IN:                             ; ODT Console In Routine
        BITB    #BIT7,@#ODT_CONIN_STAT        ; Check bit-7/ready of xmt status reg
        BEQ     ODT_CONSOLE_IN              ; Nothing there while bit-7 is 0
        MOVB    @#ODT_CONIN_DATA,R0         ; ASCII to R0 reg
        RTS     PC                          ; Return

;-----------------------------------------------------------------------------------------------
MainMenu:
equs    CR,LF
equs   "A=Memmap    C=XMODEM         D=Disp RAM  E=Echo           F=Fill RAM(Byte)",CR,LF
equs   "G=Goto RAM  H=Fill RAM(Word) I=IOBYTE    K=Menu           L=IO Test",CR,LF
equs   "M=Move RAM  N=               QI,O=Port   U=Speech Test    S=Subs RAM (Byte)",CR,LF
equs   "T=ASCII RAM V=Verify RAM     X=Interrupt Y=Subs RAM(Word) Z=Back to Z80",CR,LF,LF,0

Signon:      equs   CR,LF,"S100 Bus PDP-11 Monitor V1.00 John Monahan 4/3/2017.  (SP = ",0
Signon1:     equs   "H)",CR,LF,0
MM_Text:     equs   CR,LF,"Memory Map (64K)",CR,LF,0
Echo_Text:   equs   CR,LF,"Enter Characters (ESC to abort)",0
VerMsg0:     equs   CR,LF,"Mismatch found at ",0
VerMsg1:     equs   "H = ",0
VerMsg2:     equs   " = ",0
VerMsg3:     equs   "H ",0
WillSpeak:   equs   CR,LF,"Will speak string--> "
TestSpeak:   equs   "1 2 3 4 5 6 7 8 9",CR,0
BadPort:     equs   CR,LF,"Inactive Talk Port",0
TimeoutPort: equs   CR,LF,"Talk Port Timeout",0
Z80Msg:      equs   CR,LF,"Back To Z80",CR,LF,0
```

```
IOByteMsg:    equs   CR,LF,"IOBYTE = ",0
NotDoneMsg:   equs   CR,LF,"Command code not yet done!",0
IOTestMsg:    equs   CR,LF,"Send character test string to port 01H, 80 times",CR,LF,0
CMD_Done:     equs   CR,LF,"Test Finished",CR,LF,0
TestStr:      equs   CR,LF," !#$%&'()*+,-./0123456789@ABCDEFGHIJKLMNOPQRST abcdefghijklmnoqrstuvwxyz",CR,LF,0
XModemMsg:    equs   CR,LF,"Load a File from a PC into RAM via the PDP-11 UART",CR,LF,0
RAMStart:     equs   CR,LF,"Enter destination in RAM for data (up to 4 digits): ",0
StartMsg:     equs   CR,LF,"Will load data starting at RAM location ",0
HCRLFMsg:     equs   "H",CR,LF,0
RMSGMsg:      equs   CR,LF,"Waiting for Block# ",0
RAMMsg:       equs   "H.  If OK will write to RAM location ",0
NOSOH:        equs   CR,LF,"Did not get SOH",CR,LF,0
XERR2:        equs   CR,LF,"Bad Block# in Header",CR,LF,0
XERR3:        equs   CR,LF,"Bad Checksum for Block",CR,LF,0
TRANS_DONE:   equs   CR,LF,LF,"Data Transfer Is Complete",CR,LF,LF,0
TimeOutMsg:   equs   CR,LF,"Timeout.",0

CatchAllMsg: equs   CR,LF,"Undefined Interrupt detected",CR,LF,0
AbortMsg:     equs   CR,LF,"Abort Interrupt detected. (Check for word access to Odd Port)",CR,LF,0
OPCodeMsg:    equs   CR,LF,"Illegal Opcode detected",CR,LF,0
TrapMsg:      equs   CR,LF,"Trap Instruction detected",CR,LF,0
EventMsg:     equs   CR,LF,"Event Interrupt detected",CR,LF,0
PIRMsg:       equs   CR,LF,"PIR Interrupt detected",CR,LF,0
FPEMsg:       equs   CR,LF,"FP Error detected",CR,LF,0
```