```
    .include ".\Hardware.asm"


    .area   RAM    (REL)                ;RAM area is relative

InitPrompt: .db     0           ;initial prompt needed
CmdLinePtr: .dw     0           ;ptr to command line that is ready to proc
FlashCursor:.db     0           ;set if cursor should flash

;User variables on the command line are parsed into ParseData for use by t
ParsePtr:   .dw     0           ;ptr to current entry in ParseData
ParseData:  .ds     (2*4)       ;array of 4x 2 byte user input data fields

    .globl  FlashCursor, ParseData

    .area   CODE   (REL)                ;program area CODE is relative

    .globl Monitor, InitMonitor

PromptMsg:  .ascii  "c:\\>"
            .db     0
PromptLen = 4

BadFmtMsg:  .ascii      "invalid command format"
            .db         0x00

CmdTable:
            .ascii      "d"                 ;dump memory
            .db         4
            .db         4
            .db         0
            .db         0
            .dw         DumpMemory
            .ascii      "f"                 ;fill memory
            .db         4
            .db         4
            .db         2
            .db         0
            .dw         FillMemory
            .ascii      "i"                 ;input port
            .db         2
            .db         0
            .db         0
            .db         0
            .dw         InputPort
```

```
            .ascii      "o"                 ;output port
            .db         2
            .db         2
            .db         0
            .db         0
            .dw         OutputPort
            .ascii      "c"                 ;toggle cursor
            .db         0
            .db         0
            .db         0
            .db         0
            .dw         ToggleCursor
            .ascii      "r"                 ;run program
            .db         4
            .db         0
            .db         0
            .db         0
            .dw         RunProgram
.if SUPPORT_HARD_DISK
            .ascii      "s"                 ;sys hard drive
            .db         4
            .db         0
            .db         0
            .db         0
            .dw         SysDrive
.endif
            .ascii      "t"                 ;test RAM
            .db         0
            .db         0
            .db         0
            .db         0
            .dw         TestRAM

            .db         0x00                ;end of command table

; Length of each command table entry
CTELen = 7
    CTEOffCmd = 0       ;offset to command
    CTEOffP1 = 1        ;offset to input definition field 1
    CTEOffP2 = 2        ;offset to input definition field 2
    CTEOffP3 = 3        ;offset to input definition field 3
    CTEOffP4 = 4        ;offset to input definition field 4
    CTEOffFcn = 5       ;offset to function to invoke
```

```
DumpMemory:
    ld      hl,(ParseData+2)            ;get high address
    ld      de,(ParseData)              ;get low address
    xor     a                           ;clear c flag
    sbc     hl,de                       ;calculate offset requested
    ld      a,h                         ;
    or      a                           ;
    jr      nz,DMGT255                  ;if (delta < 256)
    ld      a,l                         ;  use lsb as count
    jr      DMLCDone                    ;
DMGT255:                                ;else
    xor     a                           ;  use 256 as count
DMLCDone:                               ;endif
    ld      b,a                         ;set loop counter for #of byte

    ld      hl,(ParseData)              ;get low address
DMLoop:                                 ;repeat

    ld      a,(hl)                      ;  get next byte
    call    HexDispA                    ;  display it as a hex byte
    ld      a,#0x20                     ;  get a space
    call    CmdWRA                      ;  display it
    inc     hl                          ;  adjust pointer up
    djnz    DMLoop                      ;until (all bytes displayed)
    ret

FillMemory:
    ld      hl,(ParseData+2)            ;get high address
    ld      de,(ParseData)              ;get low address
    xor     a                           ;clear c flag
    sbc     hl,de                       ;calculate offset requested
    inc     hl                          ;adjust up by one to include t
    ex      de,hl                       ;de = #of bytes to fill
    ld      hl,(ParseData)              ;get low address
FMLoop:                                 ;repeat
    ld      a,(ParseData+4)             ;  get fill byte
    ld      (hl),a                      ;  store it
    inc     hl                          ;  adjust pointer up
    dec     de                          ;  decrement count
    ld      a,d                         ;
    or      e                           ;
    jr      nz,FMLoop                   ;until (all bytes filled)
    ret
```

```
InputPort:
    ld      a,(ParseData)                   ;get port number
    ld      c,a                             ;
    in      a,(c)                           ;
    call    HexDispA                        ;display it as a hex byte
    ret

OutputPort:
    ld      a,(ParseData)                   ;get port number
    ld      c,a                             ;
    ld      a,(ParseData+2)                 ;get byte to output
    out     (c),a                           ;
    ret

ToggleCursor:
    ld      a,(FlashCursor)                 ;
    cpl                                     ;invert the flash cursor flag
    ld      (FlashCursor),a                 ;
    ret

RunProgram:
    ld      hl,(ParseData)                  ;get address of program to run
    jp      (hl)                            ;jump to it
    ret

    ;RAMTest logic is extensive and has been placed in it's own file for c
    ;Note: it is included here (rather than linked) due to the linker's in
    ;      to support commands from a file and 132 character command line

    .include ".\RAMTest.asm"


;-----------------------------------------------------------------------
;CmdHelp
;
;Display command help based on command table
;
;all registers affected
;-----------------------------------------------------------------------
CmdHelp:                                ;
    ld      ix,#CmdTable                ;setup structure pointer to comman
CHelpLoop:                              ;Repeat
    ld      a,CTEOffCmd(ix)             ;  get the next command
    or      a                           ;
    jr      z,CHelpDone                 ;  *abort if end of table
```

```
    call    CmdWRA                      ;  display the command
    ld      a,#0x20                     ;
    call    CmdWRA                      ;  display space
    ld      b,#0x04                     ;  set for a maximum of 4 user par
    push    ix                          ;
CHPLoop:                                ;  repeat
    ld      a,CTEOffP1(ix)              ;    get size of user parameter in
    or      a                           ;
    jr      z,CHelpNext                 ;    *if found zero byte input fie
    push    bc                          ;
    ld      b,a                         ;    setup to display the specifie
CHP1Loop:                               ;    repeat
    ld      a,#0x58                     ;
    call    CmdWRA                      ;      display one more 'X'
    djnz    CHP1Loop                    ;    until (all 'X' displayed)
    ld      a,#0x20                     ;
    call    CmdWRA                      ;    display a space
    pop     bc                          ;
    inc     ix                          ;    increment pointer
    djnz    CHPLoop                     ;  until (all parameters have beer
CHelpNext:                              ;
    call    CmdNewLine                  ;  issue a new line
    pop     ix                          ;
    ld      de,#CTELen                  ;  get structure entry size
    add     ix,de                       ;  adjust structure pointer
    jr      CHelpLoop                   ;Until (end of table)
CHelpDone:
    ret


;-------------------------------------------------------------------------
;ParseInput
;
;Parse input field
;
;Input: HL - ptr to user command string past command that contains input v
;       A - size of input field (in bytes) only 2, and 4 currently support
;
;Output: A - 0 = field processed, !0 = field error
;        HL - adjusted past processed field
;
;A, HL, affected
;-------------------------------------------------------------------------

ParseInput:                             ;
```

```
    push    bc                          ;
    push    de
    ld      b,a                         ;save field size in B
    ld      de,#0000                    ;initialize our summation field fo
    call    SkipWhite                   ;skip whitespace
    or      a                           ;
    jr      z,PIError                   ;*abort with error if end of strin
PILoop:                                 ;repeat
    ld      a,(hl)                      ;  get the next character
    inc     hl                          ;  adjust command string pointer p
    call    HexInput                    ;  add it into the summation field
    or      a                           ;  check for errors
    jr      nz,PIError                  ;  *abort if error
    djnz    PILoop                      ;until (all field bytes scanned)
                                        ;if (no errors occured)
    push    hl                          ;  DE = field value
    ld      hl,(ParsePtr)               ;
    ld      a,e                         ;
    ld      (hl),a                      ;
    inc     hl                          ;
    ld      a,d                         ;
    ld      (hl),a                      ;  store DE in the word pointed to
    inc     hl                          ;
    ld      (ParsePtr),hl               ;  update ParsePtr to point to nex
    pop     hl                          ;
    ld      a,#0                        ;  indicate success
    jr      PIDone                      ;
PIError:                                ;else
    ld      a,#1                        ;  indicate an error occured
PIDone:                                 ;endif
    pop     de                          ;
    pop     bc                          ;
    ret


;-------------------------------------------------------------------------
;ParseCmds
;
;Parse user commands
;
;Input: HL - ptr to user command string
;
;  logic will parse string using CmdTable invoking the appropriate subrout
;  after populating the ParseData global array
;
```

```
;all registers destroyed
;-----------------------------------------------------------------------

ParseCmds:
    ld      de,#ParseData
    ld      (ParsePtr),de               ;initialize parse pointer to start

    ld      ix,#CmdTable                ;setup structure pointer to comman
PCCLoop:                                ;Repeat
    ld      a,CTEOffCmd(ix)             ;  get the next command
    or      a                           ;
    jr      z,PCPError                  ;  *abort if end of table
    cp      a,(hl)                      ;  check to see if this is the use
    jr      nz,PCCNext                  ;  if (command entry found)
    inc     hl                          ;    adjust command string pointer
    ld      b,#0x04                     ;    set for a maximum of 4 user p
PCPLoop:                                ;    repeat
    ld      a,CTEOffP1(ix)              ;      get size of user parameter
    or      a                           ;
    jr      z,PCNoParm                  ;      if (parameter length != 0)
    call    ParseInput                  ;        parse the specified user
    or      a                           ;
    jr      nz,PCPError                 ;        *if parse error occured a
PCNoParm:                               ;      endif
    inc     ix                          ;      increment pointer
    djnz    PCPLoop                     ;    until (all parameters have be
    dec     ix                          ;
    dec     ix                          ;
    dec     ix                          ;
    dec     ix                          ;
PCReady:                                ;    Done parsing input fields
    ld      a,CTEOffFcn(ix)             ;
    ld      l,a                         ;
    ld      a,CTEOffFcn+1(ix)           ;
    ld      h,a                         ;    HL = ptr to command handler f
    jp      (hl)                        ;    *jump directly to handler (wh
                                        ;     and NOT fall thru to the log
                                        ;
PCCNext:                                ;  endif
    ld      de,#CTELen                  ;  get structure entry size
    add     ix,de                       ;  adjust structure pointer
    jr      PCCLoop                     ;Until (end of table, or command f

PCPError:                               ;if (parse error occured)
    ld      hl,#BadFmtMsg               ;
```

```
    call    CmdWRString                 ;  display invalid command message
    call    CmdNewLine                  ;
    call    CmdHelp                     ;  display list of valid commands
    ret                                 ;endif

;------------------------------------------------------------------------
;Monitor
;
;This should be called once before the program attempts to display informa
;
;all registers destroyed
;------------------------------------------------------------------------

InitMonitor:                            ;One time monitor initialization
    xor     a                           ;important - this must be either 0
    ld      (FlashCursor),a             ;default to no cursor flashing

.if SUPPORT_BREAKPOINTS
    call    BreakPoint                  ;set a breakpoint here
.endif

    call    InitCmdLine                 ;Initialize the Command Line logic

.if SUPPORT_BREAKPOINTS
    call    BreakPoint                  ;set a breakpoint here
.endif

    ld      a,#1                        ;
    ld      (InitPrompt),a              ;indicate the initial prompt is ne


; *** do not issue a prompt here as power-up messages can still come in **
    ret

;------------------------------------------------------------------------
;Monitor
;
;This should be called frequently from the main loop so that the monitor l
;process user input from the keyboard
;
;all registers destroyed
;------------------------------------------------------------------------

Monitor:
    ld      a,(InitPrompt)              ;
```

```
    or      a                       ;
    jr      z,MPDone                ;if (initial prompt needs to be di
    xor     a                       ;
    ld      (InitPrompt),a          ;  clear out trap door flag

.if     USE_PU_LED_CODES
    ld      a,#LEDMONI1
    out     (LEDPORT), a
.endif

    ld      hl,#PromptMsg           ;
    call    CmdWRPrompt             ;  display prompt message

.if     USE_PU_LED_CODES
    ld      a,#LEDMONI2
    out     (LEDPORT), a
.endif

MPDone:                             ;endif

.if     USE_PU_LED_CODES
    ld      a,#LEDMONL1
    out     (LEDPORT), a
.endif

    call    CmdLineScan             ;process command line input (HL po
    or      a                       ;
    jr      z,MonDone               ;if (command is ready to be proces

.if     USE_PU_LED_CODES
    ld      a,#LEDMONL2
    out     (LEDPORT), a
.endif

    inc     hl
    inc     hl
    inc     hl
    inc     hl

    call    ParseCmds               ;  parse and execute user command

.if     USE_PU_LED_CODES
    ld      a,#LEDMONL3
    out     (LEDPORT), a
.endif
```

```
    call    CmdNewLine                  ;  issue a new line to ensure the
    ld      hl,#PromptMsg               ;    after any command logic may h
    call    CmdWRPrompt                 ;  display prompt message

.if     USE_PU_LED_CODES
    ld      a,#LEDMONL4
    out     (LEDPORT), a
.endif

MonDone:                                ;endif

.if     USE_PU_LED_CODES
    ld      a,#LEDMONL5
    out     (LEDPORT), a
.endif

    ret
```