```
; Simple Z80 Boot Loader routine
; to be built with ZCC (version 3.09) tools
; available here: http://www.z80.info/z80sdt.htm
; Note: tools will only run in 16bit windows (ex: XP)
;
; build using the following commands:
;     set path=%path%;c:\z80\zcc\exe
;     asz80 -l bootrom.asm
;     aslink -i bootrom.rel
;     del bootrom.hex
;     ren a.ihx bootrom.hex


    .include ".\Hardware.asm"

    .area   RAM (REL)

    .globl      RAMLowAddress, RAMHighAddress, OSEntryPoint

.if SUPPORT_BREAKPOINTS
    .globl      BrkPtFlag

BrkPtFlag:      .db     0x00            ;this should be the first byte in
                                        ;hardware debug tools - such as th
                                        ;0x00 allows the breakpoint to be
                                        ;!0x00 causes the breakpoint to sp
                .db     0               ;need spare byte after for front p
.endif

RAMLowAddress:  .ds     2               ;start of RAM found (first actual
RAMHighAddress: .ds     2               ;end of RAM found (last actual wor

    .area   VECTOR   (ABS)               ;program area VECTOR is absolute



;--------------------------------------------------------------
;
; Reset and Interrupt Vectors
;
;--------------------------------------------------------------

;Reset & RST00 Vector
    .org    0x0000
    jp  PowerUp
```

```
;RST08 Vector
    .org    0x0008          ;OS Entry point
    jp      OSEntryPoint
    ret

;RST10 Vector
    .org    0x0010
    ret

;RST18 Vector
    .org    0x0018
    ret

;RST20 Vector
    .org    0x0020
    ret

;RST28 Vector
    .org    0x0028
    ret

;RST30 Vector
    .org    0x0030
    ret

;RST38 Vector & Interrupt Vector in Interrupt Mode 1
    .org    0x0038
    di
    reti

;-------------------------------------------------------------------------
;Put these strings here to use the empty space between the interrupt vecto
NoRAMMsg:
    .ascii  "No RAM found! halting CPU."
    .db     0
RAMMsg:
    .ascii  "RAM found from "
    .db     0
;-------------------------------------------------------------------------

;NMI Vector
    .org    0x0066
    reti
```

```
;---------------------------------------------------------------
;
; End of Vector section, Start of Relocatable Code
;
;---------------------------------------------------------------


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


.if SUPPORT_BREAKPOINTS
    .globl      BreakPoint
.endif


.if     USE_IM2

;Default ISR
DISR:
    di
    reti

;---------------------------------------------------------------
;
; ISR table to use if using interrupt mode 2
;
;---------------------------------------------------------------

;Replace default vectors with actual interrupt vectors
ISR_TABLE:
    .org    0x0100
    .dw     DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DI
    .dw     DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DI
    .dw     DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DI
    .dw     DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DI
    .dw     DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DI
    .dw     DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DI
    .dw     DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DISR, DI

.endif


BootMsg:
```

```
    ;Keep this at 64 characters so cursor ends up in column 1 of the next
    .ascii  "Digital Group Restoration Project, Copyright Alec Ginsburg 20
    .db     0

.if USE_CMD_DISP
.else
EOLMsg:
    .ascii  "                                               "
    .db     0
.endif


.if SUPPORT_BREAKPOINTS
    .include    ".\BreakPoint.asm"
.endif

;-----------------------------------------------------------------
;
; Power up logic - no calls allowed until RAM found and stack
;                  setup.
;
;-----------------------------------------------------------------

PowerUp:

    di                                  ;disable interrupts incase we came

    ld      a, #LED_RESET               ;write status pattern to LED port
    out     (LEDPORT), a


    ;--------------------------------------
    ;Clear the screen and write out logo
    ;--------------------------------------

    ld      bc, #((DISPCHARS * DISPLINES) + 16)
clsloop:                                ;write blanks to all display posit
    ld      a, #(0x20 + DISPWRBIT)
    out     (DISPPORT), a
    xor     a
    out     (DISPPORT), a
    dec     bc
    ld      a, b
    or      c
    jr      nz, clsloop
```

```
    ld      a, #(DISPHOME + DISPWRBIT)  ;move the cursor to the home posit
    out     (DISPPORT), a
    xor     a
    out     (DISPPORT), a

    ld      hl, #BootMsg                ;write the BootMsg string to the c
writeloop:
    ld      a,(hl)
    or      a
    jr      z, writedone
    inc     hl
    or      #DISPWRBIT
    out     (DISPPORT), a
    xor     a
    out     (DISPPORT), a

    jp      writeloop

writedone:


    ;--------------------------------------
    ;Determine usable RAM block area
    ;
    ;NOTE: this is a quick RAM check and
    ;      does NOT fully exercise the
    ;      RAM before declaring it usable.
    ;      The tests in RAMTest.asm should
    ;      be used after power up to see
    ;      if the RAM is truely stable.
    ;--------------------------------------

    ld      hl, #0x2000                 ;BootROM occupies 0x0000-0x1FFF so
    ld      b, #0x00                    ;default to no RAM found yet
    ld      de, #0x2000                 ;default to RAM start right after
RAMLOOP:                                ;Repeat

    ld      c,(hl)                      ;  save the old RAM value in C reg

    ld      a, #0x55                    ;
    ld      (hl), a                     ;  write 0x55 to the location
    cp      (hl)                        ;  check to see if the value can b
    jr      nz, NOMATCH                 ;
    ld      a, #0xaa                    ;
```

```
    ld      (hl), a                     ;  write 0xaa to the location
    cp      (hl)                        ;  check to see if the value can b
    jr      nz, NOMATCH                 ;
MATCH:                                  ;  if (this address has functionin
    ld      (hl),c                      ;    restore old RAM value
    ld      a, b                        ;
    or      a                           ;
    jr      nz, NOTFIRST                ;    if (this is the first functio
    ld      d, h                        ;
    ld      e, l                        ;      save the first RAM location
    ld      b, #0x01                    ;      indicate the first byte of
NOTFIRST:                               ;    endif
    jr  CHKDONE                         ;
NOMATCH:                                ;  else
    ld      (hl),c                      ;    restore old RAM value
    ld      a, b                        ;
    or      a                           ;
    jr      nz, ENDFOUND                ;    *abort if (we had previously
CHKDONE:                                ;  endif
    inc     hl                          ;  check for end of possible memor
    ld      a, h
    or      l
    jr      nz, RAMLOOP                 ;Until all of memory has been chec

ENDFOUND:                               ;if (B != 0) then de=start of RAM,
    dec     hl                          ;adjust HL back from (last valid R

    ;--------------------------------------
    ;Halt the processor if no RAM exists
    ;--------------------------------------

    ld      a, b                        ;
    or      a                           ;
    jr      nz, RAMEXISTS               ;if (RAM was NOT found) then

    ld      a, #LED_NORAM               ;  write status pattern to LED por
    out     (LEDPORT), a

    ld      hl, #NoRAMMsg               ;  write the 'No RAM' string to th
writeloop1:
    ld      a,(hl)
    or      a
    jr      z, writedone1
    inc     hl
    or      #DISPWRBIT
```

```
    out     (DISPPORT), a
    xor     a
    out     (DISPPORT), a

    jp      writeloop1

writedone1:

    halt                                ;  halt the processor, nothing le


RAMEXISTS:                              ;endif

    ld      (RAMLowAddress),de          ;store start of RAM address
    ld      (RAMHighAddress),hl         ;store end of RAM address

    ;--------------------------------------
    ; DE=start of RAM, and HL=end of RAM
    ;--------------------------------------

    ld      a, #LED_RAM                 ;write status pattern to LED port
    out     (LEDPORT), a


.if SHOW_RAM_LOCATION_WITHOUT_CALLS
    .include    ".\DispHLDE.asm"
.endif

    ;--------------------------------------
    ;Configure stack and interrupts
    ;--------------------------------------

    ld      sp, hl                      ;setup the stack pointer to point


.if     USE_IM2
    ld      a,(ISR_TABLE)               ;load ISR table into 'i'
    ld      i,a
    im      2                           ;use interrupt mode 2
.else
    im      1                           ;use interrupt mode 1
.endif

    ;--------------------------------------
    ;Ensure interrupts are disabled
```

```
    ;
    ;There are no valid interrupt sources
    ;so keep disabled to prevent spurious
    ;interrupts from causing havoc.
    ;-------------------------------------

    di

    ;-------------------------------------
    ;start program
    ;
    ;From this point on it is safe to call
    ;subroutines now that we know that
    ;RAM exists and we have a stack.
    ;-------------------------------------

    push    hl                          ;save end address on stack
    push    de                          ;save start address on stack

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

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

    call    InitMonitor                 ;Initialize the Monitor

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

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

    ld      hl,#BootMsg                 ;re-display the Copyright message
    call    DisplayString

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

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

    ;--------------------------------------
    ;Display usable RAM to user
    ;DE=start of RAM, and HL=end of RAM
    ;--------------------------------------

    ld      hl,#RAMMsg
    call    DisplayString               ;display RAM found message

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

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

    pop     de                          ;pop start address from stack
    call    DisplayDE                   ;display start of RAM address

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

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

    ld      a,#0x20
.if USE_CMD_DISP
    call    CmdWRChar
.else
    call    DisplayA                    ;display a space
.endif

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

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

    pop     de                          ;pop end address from stack
    call    DisplayDE                   ;display end of RAM address

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

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

.if USE_CMD_DISP
    call    CmdNewLine                  ;display spaces until end of line
.else
    ld      hl, #EOLMsg
    call    DisplayString               ;display spaces until end of line
.endif

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

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

    jp      MainLoop

    .include        ".\OSIFCode.asm"
```