```
;;;                                                     ;;;
;;; simple Z80 monitor                                  ;;; simple Z80 monitor
;;;                                                     ;;;
                                                      > ;;; h                          print help
;;; d <addr> <count>           dump memory              ;;; d <addr> <count>           dump memory
;;; e <addr> <dd> <dd>...      edit up to 16 bytes in memory ;;; e <addr> <dd> <dd>...  edit up to 16 bytes in memory
;;; o <port> <val>             output <val> to <port>   ;;; o <port> <val>           output <val> to <port>
;;; z <val>                    set port zero value bits 0-6 <
;;; i <port>                   input from <port> and display ;;; i <port>              input from <port> and display
;;; g <addr>                   goto addr                | ;;; g <addr>              goto address (restore regs)
;;; b <addr>                   set breakpoint (currently 3-by ;;; b <addr>           set breakpoint (currently 3-by
;;; a <val1> <val2>            hex Arithmetic           ;;; a <val1> <val2>        hex Arithmetic
;;; f <addr>                   dump HP registers from <addr> <
;;; c                          continue from breakpoint ;;; c                      continue from breakpoint
;;; c <addr>                   continue, set new breakpoint ;;; c <addr>           continue, set new breakpoint
;;; m <start> <end> <size>     memory region compare    ;;; m <start> <end> <size> memory region compare
;;; p <start> <end> <size>     memory region copy       ;;; p <start> <end> <size> memory region copy
;;; l                          binary load              ;;; l                      binary load
;;; r                          repeat last command      | ;;; f <n>               restore regs, call function <n
;;; calculator hardware                                 | ;;; r                   display stored regs
;;; k                          scan keyboard            | ;;; r <reg> <val>       edit stored regs
;;; 7 <addr>                   update 7-segment display from | ;;; x <lba_h> <lba_l> <addr> read sector
;;; V <addr>                   update VFD display from <addr> | ;;; y <lba_h> <lba_l> <addr> write sector
                                                      | ;;; w <start> <size> <patt> zero or pattern-fill memory (s
                                                      > ;;;
                                                      > ;;; gs [<lba_h> <lba_l> <addr> <count>]  GETSYS load CP/M
                                                      > ;;;                          defaults to 0 0 E000 40
                                                      >
                                                      >       extern  sec128          ;use 128 byte disk sectors

;       org     08100H                                  ;       org     08100H
        org     UMON_ORIGIN                                     org     UMON_ORIGIN

                                                      > ;;; -------------------- CP/M parameters --------------------
                                                      > ;;; (currently used only for GS command)
                                                      > MEM     EQU     63            ;63K to match cbios_hd
                                                      >
                                                      > ccp:    equ     (MEM-7)*1024
                                                      > bdos:   equ     ccp+0806h
                                                      > bios:   equ     ccp+1600h
                                                      >
                                                      > nsect:  equ     40h           ;sectors to load
                                                      > ;;; ---------------------------------------------------------
                                                      >
stak:   equ     $               ;stack grows down from start    stak:   equ     $               ;stack grows down from start

;;; jump table for useful entry points                  ;;; jump table for useful entry points
        jmp     main            ;0000  cold start       | jump_table:
        jmp     savestate       ;0003  save state (breakpoint |         ;;                    fn_no   offset    description
        jmp     getc            ;0006  read serial input to A |         jmp     main            ;00   0000      cold start
        jmp     putc            ;0009  output serial from A   |         jmp     save_state      ;01   0003      save state (b
        jmp     crlf            ;000c  output CR/LF           |         jmp     getc            ;02   0006      read serial i
        jmp     puts            ;000f  output string from HL  |         jmp     putc            ;03   0009      output serial
        jmp     phex2           ;0012  output hex byte from A |         jmp     crlf            ;04   000c      output CR/LF
        jmp     phex4           ;0015  output hex word from H |         jmp     puts            ;05   000f      output string
```

```
                                                        >         jmp     phex2           ;06   0012      output hex by
                                                        >         jmp     phex4           ;07   0015      output hex wo
                                                        >
                                                        >         jmp IDE_Initialize      ;08   0018      setup PPI, re
                                                        >         jmp IDE_Byte_Read       ;09   001b      read byte fro
                                                        >         jmp IDE_Word_Read       ;0a   001e      read word fro
                                                        >         jmp IDE_Byte_Write      ;0b   0021      write data in
                                                        >         jmp IDE_Word_Write      ;0c   0024      write data in
                                                        >         jmp IDE_Get_Status      ;0d   0027      read status r
                                                        >         jmp IDE_Wait_Ready      ;0e   002a      wait for busy
                                                        >         jmp IDE_Wait_DRQ        ;0f   002d      wait for DRQ=
                                                        >         jmp IDE_Do_Cmd          ;10   0030      issue a comma
                                                        >         jmp IDE_Setup_LBA       ;11   0033      set LBA from
                                                        >         jmp IDE_Read_ID         ;12   0036      Read 512 byte
                                                        >         jmp IDE_Read_Sector     ;13   0039      Read sector f
                                                        >         jmp IDE_Write_Sector    ;14   003c      Write sector

;;; ---- data area ----                                  ;;; ---- data area ----

;;; save CPU state coming from extrnal prog             ;;; save CPU state coming from extrnal prog
savein: db      0,0,0           ;instruction overwritten by b   savein: db      0,0,0           ;instruction overwritten by b
savead: dw      0               ;address of breakpoint          savead: dw      0               ;address of breakpoint
savsp:  dw      0               ;caller's stack pointer       | savesp: dw      0               ;caller's stack pointer

;;; saved registers                                     ;;; saved registers
saviy:  dw      0                                       saviy:  dw      0
savix:  dw      0                                       savix:  dw      0

savhlp: dw      0                                       savhlp: dw      0
savdep: dw      0                                       savdep: dw      0
savbcp: dw      0                                       savbcp: dw      0
savafp: dw      0                                       savafp: dw      0

savhl:  dw      0                                       savhl:  dw      0
savde:  dw      0                                       savde:  dw      0
savbc:  dw      0                                       savbc:  dw      0
savaf:  dw      0                                       savaf:  dw      0

savetop: equ    $                                       savetop: equ    $

regnam: db      'HL', 0, 'DE', 0, 'BC', 0, 'AF', 0      <
                                                        <
pzero:  db      0               ;port 0 value bits 0-6  <
lastc:  db      0               ;last command byte              lastc:  db      0               ;last command byte

buff:   rept    60                                      <
        db      0                                       <
        endm                                            <
                                                        <
bend:   equ     $               ;mark the end           <
                                                        <
maxarg: equ     18              ;maximum number of arguments    maxarg: equ     18              ;maximum number of arguments
argc:   db      0                                       argc:   db      0

argv:   rept    maxarg*2                                argv:   rept    maxarg*2
```

```
        dw      0                                                       dw      0
        endm                                                            endm

iargv:  rept    maxarg*2                                        iargv:  rept    maxarg*2
        dw      0                                                       dw      0
        endm                                                            endm

;       INCLUDE "s1200.asm"                                     |       secct:  db      0               ;sector count for getsys/puts
        INCLUDE "s19200.asm"                                    |       sadd:   dw      0               ;hex load sector count
                                                                >
                                                                >               INCLUDE "serial.asm"
        INCLUDE "console.asm"                                           INCLUDE "console.asm"
        INCLUDE "hex.asm"                                               INCLUDE "hex.asm"
        INCLUDE "strings.asm"                                           INCLUDE "strings.asm"
        INCLUDE "diskey.asm"                                    |       ;       INCLUDE "c-link.asm"
        INCLUDE "c-link.asm"                                    |               INCLUDE "disk_ide.asm"
        INCLUDE "vfd.asm"                                       |               INCLUDE "disk_ide_extras.asm"

banner: db      "UMON v0.7 ORG ",0                              |       banner: db      "UMON-P v0.8 ORG ",0
error:  db      "ERROR",0                                               error:  db      "ERROR",0
                                                                >       secmsg: db      " SECTORS LOADED.  START=",0

usage:  db      "h                      print this help", 13,           usage:  db      "h                      print this help", 13,
        db      "d <addr> <count", 13, 10                       <
        db      "d <addr> <count>       dump memory", 13, 10                    db      "d <addr> <count>       dump memory", 13, 10
        db      "e <addr> <dd> <dd>... edit up to 16 bytes in                   db      "e <addr> <dd> <dd>... edit up to 16 bytes in
        db      "o <addr> <val>         output <val> to port <                  db      "o <addr> <val>         output <val> to port <
        db      "z <val>                set port zero value bi <
        db      "i <addr>               input from <addr> and                   db      "i <addr>               input from <addr> and
        db      "g <addr>               goto addr", 13, 10                      db      "g <addr>               goto addr", 13, 10
        db      "b <addr>               set breakpoint (curren                  db      "b <addr>               set breakpoint (curren
        db      "a <val1> <val2>        hex Arithmetic", 13, 1                  db      "a <val1> <val2>        hex Arithmetic", 13, 1
        db      "g <addr>               dump HP registers from <
        db      "m <ad1> <ad2> <n>      memory compare", 13, 1                  db      "m <ad1> <ad2> <n>      memory compare", 13, 1
        db      "p <ad1> <ad2> <n>      memory copy", 13, 10                    db      "p <ad1> <ad2> <n>      memory copy", 13, 10
        db      "c                      continue from breakpoi                  db      "c                      continue from breakpoi
        db      "l                      binary load", 13, 10                    db      "l                      binary load", 13, 10
        db      "r                      repeat last command",   |               db      "f <n>                  call function", 13, 10
        db      "k                      scan keyboard", 13, 10  |               db      "r [<reg> <val>]        display/edit regs",13,
        db      "7 <addr>               update LED display fro  |               db      "w <addr> <count> <p>   fill memory with wordz
        db      "V <addr>               update VFD display (0=  |               db      "x <LH> <LL> <adr> [n] read disk sector(s)",1
                                                                >               db      "y <LH> <LL> <adr> [n] write disk sector",13,
                                                                >               db      "gs [<LH> <LL> <adr> n] get CP/M sectors",13,
        db      0                                                       db      0

main:   ld      sp,stak                                         main:   ld      sp,stak
                                                                >               ;; set RC2014 memory map to all RAM
                                                                >               out     (30h),a         ;reset memory page thing (bac
                                                                >               out     (38h),a         ;increment memory page thing
                                                                >               call    io_init
        ld      hl,banner                                               ld      hl,banner
        call    puts                                                    call    puts
        ld      hl,stak                                                 ld      hl,stak
        call    phex4                                                   call    phex4
        call    space                                                   call    space
```

```
        ld      hl,umontop                                      ld      hl,umontop
        call    phex4                                           call    phex4
        call    crlf                                            call    crlf

loop:   ld      a,'>'           ;prompt                 loop:   ld      a,'>'           ;prompt
        call    putc                                            call    putc

        ld      hl,buff                                         ld      hl,buff
        ld      bc,bend-buff    ; maximum size                  ld      bc,bend-buff    ; maximum size
        call    gets                                            call    gets

        ;; check for 'R'                                |      ;;; commands here which don't want tokenizer to be run
        ld      a,(buff)                                <
        cp      a,'R'                                   <
        jr      nz,not_r                                <
                                                        <
        ;; restore last command byte                    <
        ld      a,(lastc)                               <
        ld      (buff),a                                <

        ;; parse string into tokens at argc / argv     |      ;;; parse string into tokens at argc / argv
not_r:  ld      hl,buff                                 |               ld      hl,buff
        ld      de,argv                                                 ld      de,argv
        ld      b,maxarg                                                ld      b,maxarg
        call    strtok                                                  call    strtok
        ld      a,c                                                     ld      a,c
        ld      (argc),a                                                ld      (argc),a

        ;; convert tokens to integers                                   ;; convert tokens to integers
        ld      hl,argv                                                 ld      hl,argv
        ld      de,iargv                                                ld      de,iargv
        ld      b,a                                                     ld      b,a
        call    cvint                                                   call    cvint

        ld      hl,buff         ;parse command character                ld      hl,buff         ;parse command character
        ld      a,(hl)                                                  ld      a,(hl)
        ld      (lastc),a       ;save for possible repeat               ld      (lastc),a       ;save for possible repeat

        cp      a,'D'           ;dump memory                            cp      a,'D'           ;dump memory
        jz      dump                                                    jz      dump

        cp      a,'H'                                                   cp      a,'H'
        jz      help                                                    jz      help

        cp      a,'F'                                   <
        jz      hpdump                                  <
                                                        <
        cp      a,'A'                                                   cp      a,'A'
        jz      arith                                                   jz      arith

        cp      a,'E'                                                   cp      a,'E'
        jz      edit                                                    jz      edit

        cp      a,'B'                                                   cp      a,'B'
        jz      brkpt                                                   jz      brkpt
```

```
        cp      a,'C'                                   cp      a,'C'
        jz      continu                                 jz      continu

        cp      a,'G'                                   cp      a,'G'
        jz      goto                            |       jz      cmd_g

        cp      a,'L'                                   cp      a,'L'
        jz      binary                          |       jz      cmd_l

        cp      a,'O'                                   cp      a,'O'
        jz      output                                  jz      output

        cp      a,'I'                                   cp      a,'I'
        jz      input                                   jz      input

        cp      a,'Z'                           <
        jz      zero                            <
                                                <
        cp      a,'K'                           <
        jz      kbtest                          <
                                                <
        cp      a,'7'                           <
        jz      dptest                          <
                                                <
        cp      a,'V'                           <
        jz      vfdtest                         <
                                                <
        cp      a,'M'                                   cp      a,'M'
        jz      memcmp                                  jz      memcmp

        cp      a,'P'                                   cp      a,'P'
        jz      memcpy                                  jz      memcpy
                                                |
                                                >       cp      a,'R'
                                                >       jz      edit_regs
                                                >
                                                >       cp      a,'F'
                                                >       jz      call_func
                                                >
                                                >       cp      a,'X'
                                                >       jz      read_sect
                                                >
                                                >       cp      a,'Y'
                                                >       jz      writ_sect
                                                >
                                                >       cp      a,'W'
                                                >       jz      memzer
                                                >
errz:   ld      hl,error                          errz: ld      hl,error
        call    puts                                    call    puts
        call    crlf                                    call    crlf

        jp      loop                                    jp      loop
```

```
quit:   jp      0

kbtest: call    kbscan
        call    phex4
        call    crlf
        jp      loop

dptest: ld      hl,(iargv+2)
        call    display

        jp      loop

vfdtest: call   vfd_init        ;initialize (and blank) VFD d
        ld      hl,(iargv+2)    ;get address to display from
        ld      a,h             ;check for zero (blank)
        or      l
        jp      z,loop          ;zero, leave display blanked
```

```
quit:   jp      0

help:   ld      hl,usage
        call    puts
<
        jp      loop

;;; memory pattern/zero
memzer:
>       ld      hl,(iargv+2)    ;address
>       ld      bc,(iargv+4)    ;count
>       ld      de,(iargv+6)    ;pattern
>       ;; check for letter 'P' for pattern
>       ld      ix,(argv+6)     ;pointer to pattern
>       ld      a,(ix)
>       cp      a,'P'
>       jr      z,mempat
>       ;; just fill with value in DE
> memfil: push  hl              ;save start address
>       ld      (hl),e          ;store 16-bit value in first
>       inc     hl
>       ld      (hl),d
>       inc     hl
>       pop     de              ;restore start address
>       ex      de,hl           ;now hl=start, de=next
>       ldir
        jp      loop
>
>       ;; fill memory with counter value
> mempat: ld    de,0
> memp1:  ld    (hl),e
>       inc     hl
>       ld      (hl),d
>       inc     hl
>       inc     de
>       dec     bc
>       ld      a,b
>       or      c
>       jr      nz,memp1

        jp      loop


;;; write sector(s)
> writ_sect:
>       ld      a,(argc)
>       ld      b,1             ;default count
>       cp      4               ;count = 1
>       jr      z,wsrun
>       cp      5               ;count specified
>       jp      nz,errz
>       ld      a,(iargv+8)     ;count
>       ld      b,a
>
```

```
                                                        >         ;; write B sectors, incrementing LBA in DEHL
                                                        > wsrun:
                                                        >         ld      de,(iargv+2)
                                                        >         ld      hl,(iargv+4)
                                                        >         ld      ix,(iargv+6)
                                                        >
                                                        > wsloup:
                                                        >         push    hl
                                                        >         push    bc
                                                        >         push    de
                                                        >
                                                        >         call    IDE_Write_Sector
                                                        >         push    de              ;after call next locn is in D
                                                        >         pop     ix              ;copy to IX

        call    vfd_display     ;else update the display  |         pop     de
                                                        >         pop     bc
                                                        >         pop     hl
                                                        >
                                                        >         ;; increment lba
                                                        >         ld      a,h             ;check for HL=FFFF
                                                        >         and     l
                                                        >         cp      0ffh
                                                        >         inc     hl
                                                        >         jr      nz,nolbcw
                                                        >
                                                        >         inc     de
                                                        >
                                                        >         ;; delay
                                                        >         push    hl
                                                        >         ld      hl,0
                                                        > dilly:  dec     hl
                                                        >         ld      a,h
                                                        >         or      l
                                                        >         jr      nz,dilly
                                                        >         pop     hl
                                                        >
                                                        > nolbcw: djnz    wsloup
                                                        >
        jp      loop                                            jp      loop

;;; set port zero value                                |
zero:   ld      a,(iargv+2)                             | ;;; read sector(s)
        ld      (pzero),a                               | read_sect:
                                                        >         ld      a,(argc)
                                                        >         ld      b,1             ;default count
                                                        >         cp      4               ;count = 1
                                                        >         jr      z,rsrun
                                                        >         cp      5               ;count specified
                                                        >         jp      nz,errz
                                                        >         ld      a,(iargv+8)     ;count
                                                        >         ld      b,a
                                                        >
                                                        >         ;; read B sectors, incrementing LBA in DEHL
                                                        > rsrun:
```

```
                                                        >         ld      de,(iargv+2)
                                                        >         ld      hl,(iargv+4)
                                                        >         ld      ix,(iargv+6)
                                                        >
                                                        > rsloup:
                                                        >         push    hl
                                                        >         push    bc
                                                        >         push    de
                                                        >
                                                        >         call    IDE_Read_Sector
                                                        >         push    de              ;after call next locn is in D
                                                        >         pop     ix              ;copy to IX
                                                        >
                                                        >         pop     de
                                                        >         pop     bc
                                                        >         pop     hl
                                                        >
                                                        >         ;; increment lba
                                                        >         ld      a,h             ;check for HL=FFFF
                                                        >         and     l
                                                        >         cp      0ffh
                                                        >         inc     hl
                                                        >         jr      nz,nolbcy
                                                        >
                                                        >         inc     de
                                                        > nolbcy: djnz    rsloup
                                                        >
        jp      loop                                            jp      loop

;;; alternative subr to do this from outside            | ;;; call function from jump table, first restoring regs
setpzero: ld (pzero),a                                  | call_func:
        ret                                             |         ld      a,(argc)        ;check for value
                                                        >         cp      2
                                                        >         jp      nz,errz
                                                        >         ld      hl,save_state   ;user returns to here
                                                        >         push    hl
                                                        >         ld      a,(iargv+2)     ;get function code
                                                        >         ld      e,a             ;to DE
                                                        >         ld      d,0
                                                        >         ld      hl,jump_table
                                                        >         add     hl,de           ;multiply by 3
                                                        >         add     hl,de
                                                        >         add     hl,de
                                                        >         jp      gother          ;go restore state, call (hl)

help:   ld      hl,usage                                | ;;; edit/display registers in memory (at 'saveiy')
                                                        > ;;; no args = display all regs
                                                        > ;;; first arg is register name:  AF, BC, DE, HL, A', B', D',
                                                        > ;;; second arg is 16-bit value
                                                        > edit_regs:
                                                        >         ld      a,(argc)        ;get for two args
                                                        >         cp      3
                                                        >         jp      nz,view_regs    ;nope, just display regs
                                                        >         ;; search for register name as 2nd argument
                                                        >         ld      hl,(argv+2)     ;pointer to register name fro
```

```
                                                > 	ld	d,(hl)		;get first char
                                                > 	inc	hl
                                                > 	ld	e,(hl)		;get 2nd char
                                                > 	ld	hl,reg_names	;list of names for compare
                                                > 	ld	b,10		;number of register name byte
                                                > 
                                                > 	;; lookup 16-bit register name from list
                                                > regnam:
                                                > 	ld	a,d		;get first char
                                                > 	cp	(hl)		;compare it
                                                > 	inc	hl		;advance ptr (no flags change
                                                > 	jr	nz,regnm	;didn't match
                                                > 
                                                > 	ld	a,e		;get 2nd char
                                                > 	cp	(hl)		;compare
                                                > 	inc	hl
                                                > 	jr	z,regfound
                                                > 
                                                > regnxt: djnz	regnam
                                                > 
                                                > 	jp	badreg
                                                > 
                                                > regnm:  inc	hl		;skip 2nd char
                                                > 	jr	regnxt		;go loop
                                                > 
                                                > 	;; reg name not found
                                                > 	;; display it
                                                > badreg: ld	hl,badreg_msg
	call	puts                                    	call	puts
                                                > 	ld	hl,(argv+2)
                                                > 	ld	a,''''
                                                > 	call	putc
                                                > 	ld	a,(hl)
                                                > 	call	putc
                                                > 	inc	hl
                                                > 	ld	a,(hl)
                                                > 	call	putc
                                                > 	ld	a,''''
                                                > 	call	putc
                                                > 	call	crlf
	jp	loop                                    	jp	loop

                                                > badreg_msg:	db 'BAD REG',13,10,0
                                                > 
                                                > view_regs:
                                                > 	call	display_regs
                                                > 	jp	loop
                                                > 
                                                > ;;; register name at HL-2 matches
                                                > regfound:
                                                > 	dec	hl
                                                > 	dec	hl
                                                > 	or	a		;clear CY
                                                > 	ld	de,reg_names
                                                > 	sbc	hl,de		;now we have offset into regi
```

```
>         ld      de,saviy        ;offset by two since HL point
>         add     hl,de           ;point to stored reg value
>         ld      de,(iargv+4)    ;get new value
>
>         ld      (hl),e          ;change the stored value
>         inc     hl
>         ld      (hl),d
>
>         jp      loop
>
> reg_names:
>         db      'IY', 'IX', 'H'', 'D'', 'B'', 'A'', 'HL',
>
```

```
;;; output to port
output: ld      a,(iargv+2)
        ld      c,a
        ld      a,(iargv+4)
        out     (c),a
        jp      loop

;;; input from port
input:  ld      a,(iargv+2)
        ld      c,a
        in      a,(c)
        call    phex2
        call    crlf
        jp      loop

        ;; continue after breakpoint
        ;; check if there is one first
        ;; clears breakpoint as part of the process
continu: ld     hl,(savead)
        ld      a,l
        or      h
        jp      z,nobrk         ;go if not set
        ld      de,0
        ld      (savead),de     ;mark as cleared

        ex      de,hl           ;address to HL

        ;; first restore the instruction saved
        ld      hl,savein
        ldi
        ldi
        ldi

        ;; optionally, set a new breakpoint
        ld      a,(argc)
        cp      2
        jr      nz,nonew

        ;; set new breakpoint
        ld      hl,(iargv+2)
        call    brkat
```

```
  ;;; output to port
  output: ld      a,(iargv+2)
          ld      c,a
          ld      a,(iargv+4)
          out     (c),a
          jp      loop

  ;;; input from port
  input:  ld      a,(iargv+2)
          ld      c,a
          in      a,(c)
          call    phex2
          call    crlf
          jp      loop

          ;; continue after breakpoint
          ;; check if there is one first
          ;; clears breakpoint as part of the process
  continu: ld     hl,(savead)
          ld      a,l
          or      h
          jp      z,nobrk         ;go if not set
          ld      de,0
          ld      (savead),de     ;mark as cleared

          ex      de,hl           ;address to HL

          ;; first restore the instruction saved
          ld      hl,savein
          ldi
          ldi
          ldi

          ;; optionally, set a new breakpoint
          ld      a,(argc)
          cp      2
|         jr      nz,restore_state

          ;; set new breakpoint
          ld      hl,(iargv+2)
          call    brkat
```

```
        call    phex4                                           call    phex4
        ld      hl,msgset                                       ld      hl,msgset
        call    puts                                            call    puts

        ;; restore the machine state                            ;; restore the machine state
nonew:  ld      sp,saviy                                |       ;; restore stack from (savesp) and ret
                                                        > restore_state:
                                                        >       ld      sp,saviy

        pop     iy                                              pop     iy
        pop     ix                                              pop     ix

        pop     hl                                              pop     hl
        pop     de                                              pop     de
        pop     bc                                              pop     bc
        pop     af                                              pop     af
        exx                                                     exx
        ex      af,af'                                          ex      af,af'

        pop     hl                                              pop     hl
        pop     de                                              pop     de
        pop     bc                                              pop     bc
        pop     af                                              pop     af

        ld      sp,(savsp)      ;get back caller's stack |      ld      sp,(savesp)     ;get back caller's stack
        ret                     ;should to back to BP locn      ret                     ;should to back to BP locn


;;; set breakpoint                                              ;;; set breakpoint
brkpt:  ld      a,(argc)                                        brkpt:  ld      a,(argc)
        cp      2               ;single numeric argument?               cp      2               ;single numeric argument?
        jr      z,brkset                                                jr      z,brkset

        ;; clear breakpoint if set                                      ;; clear breakpoint if set
brkclr: ld      hl,(savead)                                     brkclr: ld      hl,(savead)
        ld      a,l                                                     ld      a,l
        or      h                                                       or      h
        jr      z,nobrk         ;go if not set                          jr      z,nobrk         ;go if not set

        call    phex4                                                   call    phex4
        ld      hl,msgclr                                               ld      hl,msgclr
        call    puts                                                    call    puts

        ld      hl,(savead)                                             ld      hl,(savead)
        ex      de,hl                                                   ex      de,hl
        ld      hl,savein                                               ld      hl,savein
        ldi                                                             ldi
        ldi                                                             ldi
        ldi                                                             ldi

        ld      hl,0                                                    ld      hl,0
        ld      (savead),hl     ;erase saved address                    ld      (savead),hl     ;erase saved address

        jp      loop                                                    jp      loop
```

```
        ;; check for breakpoint already set
brkset: ld      hl,(savead)
        ld      a,h
        or      l
        jr      nz,brkovr       ;attempt to overwrite breakpo

        ld      hl,(iargv+2)    ;breakpoint goes here
        call    brkat
        call    phex4
        ld      hl,msgset
        call    puts

        jp      loop

brkat:
        push    hl              ;save locn
        ld      (savead),hl     ;set brkpt locn
        ld      de,savein       ;save area for 3-byte instruc
        ldi
        ldi
        ldi                     ;copy 3 bytes
        pop     hl              ;get locn back
        push    hl
        ld      (hl),0cdh       ;CALL
        inc     hl
        ld      de,savestate    ;target for call
        ld      (hl),e
        inc     hl
        ld      (hl),d
        pop     hl
        ret

brkovr: ld      hl,msgovr
        call    puts
        jp      loop

nobrk:  ld      hl,msgno
        call    puts
        jp      loop

msgset: db      ' SET', 13, 10, 0
msgno:  db      'NO BKPT', 13, 10, 0
msgclr: db      ' CLEARED', 13, 10, 0
msgovr: db      'BKPT ALREADY SET CLEAR FIRST', 13, 10, 0
```

```
        ;; check for breakpoint already set
brkset: ld      hl,(savead)
        ld      a,h
        or      l
        jr      nz,brkovr       ;attempt to overwrite breakpo

        ld      hl,(iargv+2)    ;breakpoint goes here
        call    brkat
        call    phex4
        ld      hl,msgset
        call    puts

        jp      loop

brkat:
        push    hl              ;save locn
        ld      (savead),hl     ;set brkpt locn
        ld      de,savein       ;save area for 3-byte instruc
        ldi
        ldi
        ldi                     ;copy 3 bytes
        pop     hl              ;get locn back
        push    hl
        ld      (hl),0cdh       ;CALL
        inc     hl
        ld      de,save_state   ;target for call
        ld      (hl),e
        inc     hl
        ld      (hl),d
        pop     hl
        ret

brkovr: ld      hl,msgovr
        call    puts
        jp      loop

nobrk:  ld      hl,msgno
        call    puts
        jp      loop

msgset: db      ' SET', 13, 10, 0
msgno:  db      'NO BKPT', 13, 10, 0
msgclr: db      ' CLEARED', 13, 10, 0
msgovr: db      'BKPT ALREADY SET CLEAR FIRST', 13, 10, 0


>       ;; check for "LH" for hex
> cmd_l: inc     hl
>       ld      a,(hl)
>       cp      'H'
>       jr      nz,binary
>
> ;;; hex loader on SIO port B
> line:  ld      a,'+'
> prom:  call    putc_B
```

```
>
>         ld      hl,buff
>         ld      bc,bend-buff
>         call    gets_B
>
>         ld      hl,buff
>         ld      a,(hl)
>         cp      a,':'
>         jr      z,lode
>         cp      a,'/'
>         jp      z,0
>         jr      err
>
> lode:   inc     hl
>         call    ghex2           ; get record size to A
>         ld      b,a             ; size to b
>         call    ghex4           ; get load address to DE
>
>         ;; if we don't have a load address, store it now at (
>         ld      a,(sadd)
>         or      a
>         jr      nz,noja
>         ld      a,(sadd+1)
>         or      a
>         jr      nz,noja
>
>         ld      (sadd),de
>
> noja:   call    ghex2           ; get record type to A
>         or      a
>         jr      z,datt          ; zero, get data
>         dec     a
>         jr      nz,line
>         ;; type = 01, we're done
>         ld      hl,(sadd)
>
>         call    phex4
>         call    crlf
>
>         jp      main
>
> err:    ld      a,'#'
>         jr      prom
>
>         ;; parse and store data
> datt:   call    ghex2
>         ld      (de),a
>         inc     de
>         djnz    datt
>         jr      line
>
>
>
  ;;; start binary loader
  ;;; expect binary words (LSB first):
```

```
;;; start binary loader
;;; expect binary words (LSB first):
```

```
;;;    0x5791, <addr>, <count>
;;; then <count> data bytes
;;; does not jump after, just returns to prompt
;;; if header not seen after a few bytes, bail out
;;;
;;; echo back all received
;;;
binary: ld      b,5             ;max bad bytes
        ;; read chars until 5 received or 0x91 seen
bin1:   call    getc
        call    putc
        cp      0x91            ;first magic byte?
        jr      z,bin1a
        djnz    bin1
        jp      errz

        ;; read chars, skipping repeat 0x91, wait for 0x57
bin1a:  call    getc
        call    putc
        cp      0x91
        jr      z,bin1a
        cp      0x57
        jp      nz,errz

        ;; get address to hl
        call    getc
        call    putc
        ld      l,a
        call    getc
        call    putc
        ld      h,a
        ld      (iargv),hl
        ;; get count to bc
        call    getc
        call    putc
        ld      c,a
        call    getc
        call    putc
        ld      b,a
        ld      (iargv+2),bc
        ;; read and store data
bin2:   call    getc
        call    putc
        ld      (hl),a
        inc     hl
        dec     bc
        ld      a,b
        or      c
        jr      nz,bin2

        ;; leave addr, count in iargv+2, iargv+4
        ;; display them too
        ld      hl,(iargv)
        call    phex4
        call    crlf
```

```
;;;    0x5791, <addr>, <count>
;;; then <count> data bytes
;;; does not jump after, just returns to prompt
;;; if header not seen after a few bytes, bail out
;;;
;;; echo back all received
;;;
binary: ld      b,5             ;max bad bytes
        ;; read chars until 5 received or 0x91 seen
bin1:   call    getc
        call    putc
        cp      0x91            ;first magic byte?
        jr      z,bin1a
        djnz    bin1
        jp      errz

        ;; read chars, skipping repeat 0x91, wait for 0x57
bin1a:  call    getc
        call    putc
        cp      0x91
        jr      z,bin1a
        cp      0x57
        jp      nz,errz

        ;; get address to hl
        call    getc
        call    putc
        ld      l,a
        call    getc
        call    putc
        ld      h,a
        ld      (iargv),hl
        ;; get count to bc
        call    getc
        call    putc
        ld      c,a
        call    getc
        call    putc
        ld      b,a
        ld      (iargv+2),bc
        ;; read and store data
bin2:   call    getc
        call    putc
        ld      (hl),a
        inc     hl
        dec     bc
        ld      a,b
        or      c
        jr      nz,bin2

        ;; leave addr, count in iargv+2, iargv+4
        ;; display them too
        ld      hl,(iargv)
        call    phex4
        call    crlf
```

```
        ld      bc,(iargv+2)
        add     hl,bc
        call    phex4
        call    crlf
        jp      loop

;;; jump to 1st arg
goto:   cp      2
```

```
        ld      bc,(iargv+2)
        add     hl,bc
        call    phex4
        call    crlf
        jp      loop

        |
        | ;;; check for "GS"
> cmd_g:  inc     hl
>         ld      a,(hl)
>         cp      'S'
>         jr      nz, goto
>
> ;;; GETSYS
> ;;; either zero or four arguments
>         ld      a,(argc)
>         cp      5               ; all arguments specified?
>         jr      z,gsall
>         ;; set defaults
>         ld      de,0
>         ld      hl,0
>         ld      iy,ccp
>         ld      a,nsect         ;default sector count
>         ld      (secct),a
>
>         jr      getsys
> ;;;
> gsall:
>         ;; read B sectors, incrementing LBA in DEHL
>         ld      de,(iargv+2)
>         ld      hl,(iargv+4)
>         ld      iy,(iargv+6)
>         ld      a,(iargv+8)     ;count
>         ld      (secct),a
>
>         ;; load from lba=dehl (secct) sectors to iy
> getsys:
>         push    hl
>         push    de
>
>         ld      ix,dskbuf
>         call    IDE_Read_Sector
>
>         ;; copy 80h bytes from buff to IY, increment IY
>         ld      hl,dskbuf       ;source for copy
>         push    iy
>         pop     de              ;dest for copy
>         ld      bc,80h          ;count for copy
>         add     iy,bc           ;nudge iy
>         ldir                    ;do the copy
>
>         pop     de
>         pop     hl
>
>         inc     hl
```

```
                                                        >
                                                        >         ld      a,(secct)
                                                        >         dec     a
                                                        >         ld      (secct),a
                                                        >
                                                        >         jr      nz,getsys
                                                        >
                                                        >         ;; print message
                                                        >         ld      a,nsect
                                                        >         call    phex2
                                                        >         ld      hl,secmsg
                                                        >         call    puts
                                                        >         ld      hl,bios
                                                        >         call    phex4
                                                        >         call    crlf
                                                        >
                                                        >         jp      main
                                                        >
                                                        >
                                                        > ;;; jump to 1st arg
                                                        > goto:   ld      a,(argc)
                                                        >         cp      2
        jp      c,errz                                            jp      c,errz
        ld      hl,(iargv+2)                                      ld      hl,(iargv+2)
        jp      (hl)                                    |         ld      de,showstate
                                                        >         ;; set up the stack:  first, a return address in case
                                                        >         push    de
                                                        >         ;; now the user specified address from the command li
                                                        > gother: push    hl
                                                        >         ld      (savesp),sp     ;save caller's stack pointer
                                                        >         jp      restore_state

;;; edit values into memory                                       ;;; edit values into memory
edit:   ld      a,(argc)                                        edit:   ld      a,(argc)
        cp      a,3             ;need at least 3 args                   cp      a,3             ;need at least 3 args
        jp      c,errz                                                  jp      c,errz

        sub     a,2                                                     sub     a,2
        ld      b,a             ;count of bytes to store                ld      b,a             ;count of bytes to store

        ld      hl,(iargv+2)    ;get address                            ld      hl,(iargv+2)    ;get address
        ld      ix,iargv+4      ;pointer to first data item             ld      ix,iargv+4      ;pointer to first data item

eloop:  ld      a,(ix)                                          eloop:  ld      a,(ix)
        ld      (hl),a                                                  ld      (hl),a
        inc     hl                                                      inc     hl
        inc     ix                                                      inc     ix
        inc     ix                                                      inc     ix
        djnz    eloop                                                   djnz    eloop
        jp      loop                                                    jp      loop

;;; dump some memory                                              ;;; dump some memory
dump:   ld      hl,(iargv+2)    ;first arg              |       dump:   ld      a,(argc)
                                                        >         ld      b,0             ;default count = 0/100
                                                        >         cp      a,3             ;
```

```
                                                        >         jr      c,dump1
                                                        >
                                                        >         ;; else > 1 arg, so count specified
        ld      a,(iargv+4)     ;second arg                       ld      a,(iargv+4)     ;second arg
                                                        >         ld      b,a
                                                        >
                                                        > dump1:  ld      hl,(iargv+2)    ;first arg

        ld      b,a             ;count                  | dump0:  call    hdump
        call    hdump                                   |
                                                        >         ld      (iargv+2),hl    ;save addr after
                                                        >
        jp      loop                                              jp      loop

;;; hex dump B bytes from HL                              ;;; hex dump B bytes from HL
        ;; see if we need to print the address                    ;; see if we need to print the address
        ;; either on 16-byte boundary, or first address           ;; either on 16-byte boundary, or first address
hdump:  call    haddr           ;always print first address |   ;; each time we print the address, copy to IX for asc
                                                        > hdump:  call    hex_addr        ;always print first address
                                                        >         push    hl
                                                        >         pop     ix              ;address to IX
        jr      bite            ;skip the 16-byte test            jr      bite            ;skip the 16-byte test

hdump2: ld      a,l                                       hdump2: ld      a,l
        and     0xf                                               and     0xf
        call    z,haddr                                 |         jr      nz,bite
                                                        >
                                                        >         call    hex_ascii

bite:   ld      a,(hl)                                    bite:   ld      a,(hl)
        inc     hl                                                inc     hl
        call    phex2                                             call    phex2
        call    space                                             call    space

noadr:  djnz    hdump2                                    noadr:  djnz    hdump2
                                                        >
        call    crlf                                              call    crlf
        ret                                                       ret

                                                        > ;;; print 16 bytes ascii from IX
                                                        > hex_ascii:
                                                        >         call    space
                                                        >         push    bc
                                                        >         ld      b,10h
                                                        > hexal:  ld      a,(ix)
                                                        >         inc     ix
                                                        >         cp      20h             ;control char?
                                                        >         jr      c,hexdot        ;yes, print a dot
                                                        >         cp      80h             ;high bit set?
                                                        >         jr      c,hexput        ;no, print the char
                                                        >
                                                        > hexdot: ld      a,'.'
                                                        >
                                                        > hexput: call    putc
                                                        >         djnz    hexal
```

```
                                                        >         pop     bc
                                                        >
;;; print address in HL                                   ;;; print address in HL
haddr:  push    hl                                    | hex_addr:
                                                        >         push    hl
        call    crlf                                              call    crlf
        call    phex4                                             call    phex4
        ld      a,':'                                             ld      a,':'
        call    putc                                              call    putc
        call    space                                             call    space
        pop     hl                                                pop     hl
        ret                                                       ret

;;; do hex arithmetic                                   ;;; do hex arithmetic
arith:  ld      hl,(iargv+2)    ;first arg            arith:  ld      hl,(iargv+2)    ;first arg
        ld      de,(iargv+4)    ;second arg                   ld      de,(iargv+4)    ;second arg

        add     hl,de                                         add     hl,de
        call    phex4                                         call    phex4
        call    space                                         call    space
        ld      hl,(iargv+2)    ;first arg                    ld      hl,(iargv+2)    ;first arg
        ld      de,(iargv+4)    ;second arg                   ld      de,(iargv+4)    ;second arg
        or      a               ;clear CY                     or      a               ;clear CY
        sbc     hl,de                                         sbc     hl,de
        call    phex4                                         call    phex4
        call    crlf                                          call    crlf
        jp      loop                                          jp      loop

;;; HP calculator word size                           <
wsize:  equ     14                                    <
;;; register names                                    <
hpregs: db      "ABXYZT12", 0                         <
                                                      <
;;; dump HP registers                                 <
;;; all 14 nibbles:  A, B, X, Y, Z, T, M1, M2         <
hpdump: ld      hl,(iargv+2)    ;first arg            <
        ld      de,wsize                              <
        ld      ix,hpregs                             <
                                                      <
hpd1:   ld      a,(ix)          ;get register name    <
        inc     ix                                    <
        or      a               ;Z=done               <
        jp      z,loop                                <
                                                      <
        call    putc            ;display reg name     <
        call    space           ;space                <
        call    hpreg           ;display reg          <
        jr      hpd1                                  <
                                                      <
;;; display HP register from (HL)                     <
;;; de must be WSIZE (14)                             <
;;; save bc, advance HL past reg                      <
hpreg:  push    hl                                    <
        push    bc                                    <
        ld      b,wsize                               <
```

```
        add     hl,de           ;point to next reg          <
hpr1:   dec     hl                                          <
        ld      a,(hl)                                      <
        call    phex1                                       <
        djnz    hpr1                                        <
        pop     bc                                          <
        pop     hl                                          <
        add     hl,de                                       <
        call    crlf                                        <
        ret                                                 <
                                                            <
altban: db      "BREAK ",0                                    altban: db      "BREAK ",0

;;; ---------- breakpoint entry ----------                    ;;; ---------- breakpoint entry ----------
;;; Save machine state and display it                         ;;; Save machine state and display it
;;; Restore code at breakpoint                                ;;; Restore code at breakpoint
;;; ------------------------------------                      ;;; ------------------------------------
savestate:                                                  | save_state:
        ;; get the return address and save it                       ;; get the return address and save it
        ex      (sp),hl         ;return address to HL             ex      (sp),hl         ;return address to HL
        dec     hl                                                dec     hl
        dec     hl                                                dec     hl
        dec     hl              ;back up over breakpoint call     dec     hl              ;back up over breakpoint call
        ex      (sp),hl         ;put back on caller's stack       ex      (sp),hl         ;put back on caller's stack
        ld      (savsp),sp      ;save caller's SP           | showstate:
                                                            >           ld      (savesp),sp     ;save caller's SP
        ;; reset the stack to the save area                         ;; reset the stack to the save area
        ld      sp,savetop                                        ld      sp,savetop
        ;; save primary regs                                      ;; save primary regs
        push    af                                                push    af
        push    bc                                                push    bc
        push    de                                                push    de
        push    hl                                                push    hl
        ;; save alternate regs                                    ;; save alternate regs
        exx                                                       exx
        ex      af,af'                                            ex      af,af'
        push    af                                                push    af
        push    bc                                                push    bc
        push    de                                                push    de
        push    hl                                                push    hl
        ;; save IX, IY                                           ;; save IX, IY
        push    ix                                                push    ix
        push    iy                                                push    iy

;;; cold start with alternate banner, display struct address    ;;; cold start with alternate banner, display struct address
        ld      sp,stak         ;restore UMON stack                ld      sp,stak         ;restore UMON stack
        ld      hl,altban                                          ld      hl,altban
        call    puts                                               call    puts
        ;; display breakpoint location                           ;; display breakpoint location
        ld      hl,(savead)                                        ld      hl,(savead)
        call    phex4                                              call    phex4
        call    crlf                                               call    crlf
        call    pstate          ;display regs from state    |
                                                            >           call    display_regs            ;display regs from st
        jp      loop                                               jp      loop
```

```
;;; display machine state from stored values
pstate:
        ld      hl,(savaf)
        ld      de,'AF'
        call    pregn

        ld      hl,(savbc)
        ld      de,'BC'
        call    pregn

        ld      hl,(savde)
        ld      de,'DE'
        call    pregn

        ld      hl,(savhl)
        ld      de,'HL'
        call    pregn

        call    crlf

        ;; alternate regs
        ld      hl,(savafp)
        ld      de,'A'''
        call    pregn

        ld      hl,(savbcp)
        ld      de,'B'''
        call    pregn

        ld      hl,(savdep)
        ld      de,'D'''
        call    pregn

        ld      hl,(savhlp)
        ld      de,'H'''
        call    pregn
        call    crlf

        ld      hl,(savix)
        ld      de,'IX'
        call    pregn

        ld      hl,(saviy)
        ld      de,'IY'
        call    pregn

        ld      hl,(savsp)
        ld      de,'SP'
        call    pregn

        call    crlf

        jp      loop
```

```
;;; display machine state from stored values
display_regs:
        ld      hl,(savaf)
        ld      de,'AF'
        call    pregn

        ld      hl,(savbc)
        ld      de,'BC'
        call    pregn

        ld      hl,(savde)
        ld      de,'DE'
        call    pregn

        ld      hl,(savhl)
        ld      de,'HL'
        call    pregn

        call    crlf

        ;; alternate regs
        ld      hl,(savafp)
        ld      de,'A'''
        call    pregn

        ld      hl,(savbcp)
        ld      de,'B'''
        call    pregn

        ld      hl,(savdep)
        ld      de,'D'''
        call    pregn

        ld      hl,(savhlp)
        ld      de,'H'''
        call    pregn
        call    crlf

        ld      hl,(savix)
        ld      de,'IX'
        call    pregn

        ld      hl,(saviy)
        ld      de,'IY'
        call    pregn

        ld      hl,(savesp)
        ld      de,'SP'
        call    pregn

        call    crlf

        ex      af,af'          ;back to primary regs
        exx
```

```
>
>         ret

;;; copy memory
memcpy: call    load3w
        ldir
        jp      loop

;;; compare memory
;;; up to 16 errors then stop
memcmp: call    load3w
memcp1: ld      a,(de)
        cp      a,(hl)
        jr      nz,nocmp
memnxt: inc     hl
        inc     de
        dec     bc
        ld      a,b
        or      c
        jr      nz,memcp1
        jp      loop

        ;; display memory mismatch
nocmp:  call    phex4           ;display hl
        call    space
        ld      a,(hl)
        call    phex2
        call    space
        ex      de,hl
        call    phex4           ;display hl
        call    space
        ld      a,(hl)
        call    phex2
        ex      de,hl
        call    crlf
        jr      memnxt

;;; load hl, de, bc from 3 arguments for compare/copy
load3w: ld      hl,(iargv+2)    ;source
        ld      de,(iargv+4)    ;dest
        ld      bc,(iargv+6)    ;count
        ret

;;; display reg name from de, value from hl
pregn:
        call    space
        ld      a,e
        call    putc
        ld      a,d
        call    putc
        ld      a,':'
        call    putc
        call    phex4
        call    space
        ret
```

```
;;; copy memory
memcpy: call    load3w
        ldir
        jp      loop

;;; compare memory
;;; up to 16 errors then stop
memcmp: call    load3w
memcp1: ld      a,(de)
        cp      a,(hl)
        jr      nz,nocmp
memnxt: inc     hl
        inc     de
        dec     bc
        ld      a,b
        or      c
        jr      nz,memcp1
        jp      loop

        ;; display memory mismatch
nocmp:  call    phex4           ;display hl
        call    space
        ld      a,(hl)
        call    phex2
        call    space
        ex      de,hl
        call    phex4           ;display hl
        call    space
        ld      a,(hl)
        call    phex2
        ex      de,hl
        call    crlf
        jr      memnxt

;;; load hl, de, bc from 3 arguments for compare/copy
load3w: ld      hl,(iargv+2)    ;source
        ld      de,(iargv+4)    ;dest
        ld      bc,(iargv+6)    ;count
        ret

;;; display reg name from de, value from hl
pregn:
        call    space
        ld      a,e
        call    putc
        ld      a,d
        call    putc
        ld      a,':'
        call    putc
        call    phex4
        call    space
        ret
```

```
                                                  | ;;; input buffer (tokens zero-terminated after parsing)
                                                  > buff:   ds      100H
                                                  > bend:   equ     $               ;mark the end
                                                  >
                                                  > dskbuf: ds      200H
                                                  >
umontop:        equ     $                           umontop:        equ     $

        .end                                                .end
```