```
;   ***********************************************
;   *                                             *
;   *  MINMON - The Minimal 8051 Monitor Program  *
;   *                                             *
;   *  Portions of this program are courtesy of   *
;   *  Rigel Corporation, of Gainesville, Florida *
;   *                                             *
;   *  Modified for 6.115                         *
;   *  Massachusetts Institute of Technology      *
;   *  January, 2005  Steven B. Leeb              *
;   *                                             *
;   ***********************************************

stack  equ  2fh          ; bottom of stack
                         ; - stack starts at 30h -
errorf equ   0           ; bit 0 is error status

;=================================================================
; 8052 hardware vectors
;=================================================================
   org 00h               ; power up and reset vector
   ljmp start
   org 03h               ; interrupt 0 vector
   ljmp start
   org 0bh               ; timer 0 interrupt vector
   ljmp start
   org 13h               ; interrupt 1 vector
   ljmp start
   org 1bh               ; timer 1 interrupt vector
   ljmp start
   org 23h               ; serial port interrupt vector
   ljmp start
   org 2bh               ; 8052 extra interrupt vector
   ljmp start

;=================================================================
; begin main program
;=================================================================
   org     100h
start:
   clr     ea            ; disable interrupts
   lcall   init          ; initialize hardware
   lcall   print         ; print welcome message
   db 0ah, 0dh, "Welcome to 6.115!", 0ah, 0dh,"MINMON> ", 0h
monloop:
   mov     sp,#stack     ; reinitialize stack pointer
   clr     ea            ; disable all interrupts
   clr     errorf        ; clear the error flag
   lcall   print         ; print prompt
   db 0dh, 0ah,"*", 0h
   clr     ri            ; flush the serial input buffer
   lcall   getcmd        ; read the single-letter command
   mov     r2, a         ; put the command number in R2
   ljmp    nway          ; branch to a monitor routine
endloop:                 ; come here after command has finished
   sjmp monloop          ; loop forever in monitor loop
```

```
;=================================================================
; subroutine init
; this routine initializes the hardware
;=================================================================
init:
; set up serial port with a 11.0592 MHz crystal,
; use timer 1 for 9600 baud serial communications
   mov   tmod, #20h       ; set timer 1 for auto reload - mode 2
   mov   tcon, #41h       ; run timer 1 and set edge trig ints
   mov   th1,  #0fdh      ; set 9600 baud with xtal=11.059mhz
   mov   scon, #50h       ; set serial control reg for 8 bit data
                          ; and mode 1
   ret
;=================================================================
; monitor jump table
;=================================================================
jumtab:
   dw badcmd              ; command '@' 00
   dw badcmd              ; command 'a' 01
   dw badcmd              ; command 'b' 02
   dw badcmd              ; command 'c' 03
   dw downld              ; command 'd' 04 used
   dw badcmd              ; command 'e' 05
   dw badcmd              ; command 'f' 06
   dw goaddr              ; command 'g' 07 used
   dw badcmd              ; command 'h' 08
   dw badcmd              ; command 'i' 09
   dw badcmd              ; command 'j' 0a
   dw badcmd              ; command 'k' 0b
   dw badcmd              ; command 'l' 0c
   dw badcmd              ; command 'm' 0d
   dw badcmd              ; command 'n' 0e
   dw badcmd              ; command 'o' 0f
   dw badcmd              ; command 'p' 10
   dw badcmd              ; command 'q' 11
   dw badcmd              ; command 'r' 12
   dw badcmd              ; command 's' 13
   dw badcmd              ; command 't' 14
   dw badcmd              ; command 'u' 15
   dw badcmd              ; command 'v' 16
   dw badcmd              ; command 'w' 17
   dw badcmd              ; command 'x' 18
   dw badcmd              ; command 'y' 19
   dw badcmd              ; command 'z' 1a
```

db but for 2-byte objects

(db is for 1 byte)

low at 3, high at 2

```
; monitor command routines
; goaddr 'g' - this routine branches to the 4 hex digit address which follows
;==================================================================
goaddr:
   lcall getbyt          ; get address high byte
   mov   r7, a           ; save in R7
   lcall prthex
   lcall getbyt          ; get address low byte
   push  acc             ; push lsb of jump address
   lcall prthex
   lcall crlf
   mov   a, r7           ; recall address high byte
   push  acc             ; push msb of jump address
   ret                   ; do jump by doing a ret
;==================================================================
; downld 'd' - this command reads in an Intel hex file from the serial port and stores it in external memory.
;==================================================================
downld:
   lcall crlf
   mov   a, #'>'         ; acknowledge by a '>'
   lcall sndchr
dl:
   lcall getchr          ; read in ':'
   cjne  a,  #':', dl
   lcall getbytx         ; get hex length byte
   jz    enddl           ; if length=0 then return
   mov   r0, a           ; save length in r0
   lcall getbytx         ; get msb of address
   setb  acc.7           ; make sure it is in RAM
   mov   dph, a          ; save in dph
   lcall getbytx         ; get lsb of address
   mov   dpl, a          ; save in dpl
   lcall getbytx         ; read in special purpose byte (ignore)
dloop:
   lcall getbytx         ; read in data byte
   movx  @dptr, a        ; save in ext mem
   inc   dptr            ; bump mem pointer
   djnz  r0, dloop       ; repeat for all data bytes in record
   lcall getbytx         ; read in checksum
   mov   a,  #'.'
   lcall sndchr          ; handshake '.'
   sjmp  dl              ; read in next record
enddl:
   lcall getbytx         ; read in remainder of the
   lcall getbytx         ; termination record
   lcall getbytx
   lcall getbytx
   mov   a,  #'.'
   lcall sndchr          ; handshake '.'
   ljmp  endloop         ; return
getbytx:
   lcall getbyt
   jb    errorf, gb_err
   ret
gb_err:
   ljmp badpar
```

```
;*****************************************************************
; monitor support routines
;*****************************************************************
badcmd:
   lcall print
   db 0dh, 0ah," bad command ", 0h
   ljmp endloop
badpar:
   lcall print
   db 0dh, 0ah," bad parameter ", 0h
   ljmp endloop
;================================================================
; subroutine getbyt
; this routine reads in an 2 digit ascii hex number from the
; serial port. the result is returned in the acc.
;================================================================
getbyt:
   lcall getchr          ; get msb ascii chr
   lcall ascbin          ; conv it to binary
   swap  a               ; move to most sig half of acc
   mov   b,  a           ; save in b
   lcall getchr          ; get lsb ascii chr
   lcall ascbin          ; conv it to binary
   orl   a,  b           ; combine two halves
   ret
;================================================================
; subroutine getcmd
; this routine gets the command line.  currently only a
; single-letter command is read - all command line parameters
; must be parsed by the individual routines.
;
;================================================================
getcmd:
   lcall getchr          ; get the single-letter command
   clr   acc.5           ; make UPPER case
   lcall sndchr          ; echo command
   clr   C               ; clear the carry flag
   subb  a, #'@'         ; convert to command number
   jnc   cmdok1          ; letter command must be above '@'
   lcall badpar
cmdok1:
   push  acc             ; save command number
   subb  a, #1Bh         ; command number must be 1Ah or less
   jc    cmdok2
   lcall badpar          ; no need to pop acc since badpar
                         ; initializes the system
cmdok2:
   pop   acc             ; recall command number
   ret
```

they're offset by 32

checks if negative via carry flag

wants between @ & beyond bound of Z

uppercase

```
;==============================================================
; subroutine nway
; this routine branches (jumps) to the appropriate monitor
; routine. the routine number is in r2
;==============================================================
nway:
   mov   dptr, #jumtab     ;point dptr at beginning of jump table
   mov   a, r2             ;load acc with monitor routine number
   rl    a                 ;multiply by two.
   inc   a                 ;load first vector onto stack
   movc  a, @a+dptr        ;         "          "
   push  acc               ;         "          "
   mov   a, r2             ;load acc with monitor routine number
   rl    a                 ;multiply by two
   movc  a, @a+dptr        ;load second vector onto stack
   push  acc               ;         "          "
   ret                     ;jump to start of monitor routine
```

load pointer of a [redacted] subroutine

rotate-left

low-byte

high-byte

```
;*************************************************************
; general purpose routines
;*************************************************************
;==============================================================
; subroutine sndchr
; this routine takes the chr in the acc and sends it out the
; serial port.
;==============================================================
sndchr:
   clr  scon.1             ; clear the tx complete flag
   mov  sbuf,a             ; put chr in sbuf
txloop:
   jnb  scon.1, txloop     ; wait till chr is sent
   ret
;==============================================================
; subroutine getchr
; this routine reads in a chr from the serial port and saves it
; in the accumulator.
;==============================================================
getchr:
   jnb  ri, getchr         ; wait till character received
   mov  a,  sbuf           ; get character
   anl  a,  #7fh           ; mask off 8th bit
   clr  ri                 ; clear serial status bit
   ret
```

```
;==============================================================
; subroutine print
; print takes the string immediately following the call and
; sends it out the serial port.  the string must be terminated
; with a null. this routine will ret to the instruction
; immediately following the string.
;==============================================================
print:
   pop   dph                ; put return address in dptr
   pop   dpl
prtstr:
   clr   a                  ; set offset = 0
   movc a,  @a+dptr         ; get chr from code memory
   cjne a,  #0h, mchrok     ; if termination chr, then return
   sjmp prtdone
mchrok:
   lcall sndchr             ; send character
   inc   dptr               ; point at next character
   sjmp  prtstr             ; loop till end of string
prtdone:
   mov   a,  #1h            ; point to instruction after string
   jmp   @a+dptr            ; return
;==============================================================
; subroutine crlf
; crlf sends a carriage return line feed out the serial port
;==============================================================
crlf:
   mov   a,  #0ah           ; print lf
   lcall sndchr
cret:
   mov   a,  #0dh           ; print cr
   lcall sndchr
   ret
;==============================================================
; subroutine prthex
; this routine takes the contents of the acc and prints it out
; as a 2 digit ascii hex number.
;==============================================================
prthex:
   push acc
   lcall binasc             ; convert acc to ascii
   lcall sndchr             ; print first ascii hex digit
   mov   a,  r2             ; get second ascii hex digit
   lcall sndchr             ; print it
   pop acc
   ret
```

← little endian, high byte first

dptr = dph dpl

← works w/ movx too, program and ext. memory are all the same

← doesn't use lcall to escape

```
;==============================================================
; subroutine binasc
; binasc takes the contents of the accumulator and converts it
; into two ascii hex numbers.  the result is returned in the
; accumulator and r2.
;==============================================================
binasc:
   mov   r2, a            ; save in r2
   anl   a,  #0fh         ; convert least sig digit.
   add   a,  #0f6h        ; adjust it
   jnc   noadj1           ; if a-f then readjust
   add   a,  #07h
noadj1:
   add   a,  #3ah         ; make ascii
   xch   a,  r2           ; put result in reg 2
   swap  a                ; convert most sig digit
   anl   a,  #0fh         ; look at least sig half of acc
   add   a,  #0f6h        ; adjust it
   jnc   noadj2           ; if a-f then re-adjust
   add   a,  #07h
noadj2:
   add   a,  #3ah         ; make ascii
   ret
;==============================================================
; subroutine ascbin
; this routine takes the ascii character passed to it in the
; acc and converts it to a 4 bit binary number which is returned
; in the acc.
;==============================================================
ascbin:
   clr   errorf
   add   a,  #0d0h        ; if chr < 30 then error
   jnc   notnum
   clr   c                ; check if chr is 0-9
   add   a,  #0f6h        ; adjust it
   jc    hextry           ; jmp if chr not 0-9
   add   a,  #0ah         ; if it is then adjust it
   ret
hextry:
   clr   acc.5            ; convert to upper
   clr   c                ; check if chr is a-f
   add   a,  #0f9h        ; adjust it
   jnc   notnum           ; if not a-f then error
   clr   c                ; see if char is 46 or less.
   add   a,  #0fah        ; adjust acc
   jc    notnum           ; if carry then not hex
   anl   a,  #0fh         ; clear unused bits
   ret
notnum:
   setb  errorf           ; if not a valid digit
   ljmp  endloop
;==============================================================
; mon_return is not a subroutine.  It simply jumps to address 0 which resets the
; system and invokes the monitor program.
;==============================================================
mon_return:
   ljmp  0 ; end of MINMON
```