RAM loader program for 80C51 family applications AN440

Author: Greg Goodhue

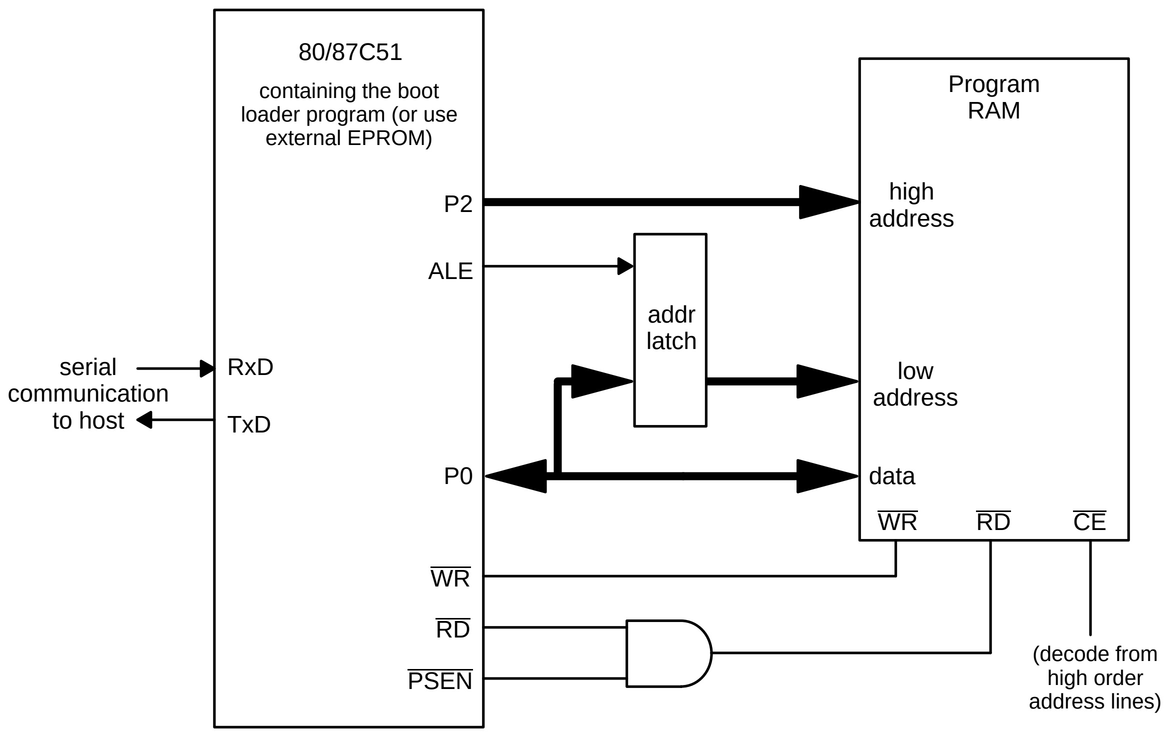
The following program allows an 80C51 family microcontroller to load most of its code into a RAM over a serial link after power up and execute out of the RAM for normal operation. This can allow a final product to have firmware updates done by a simple diskette mailing. Such a program is often called a "bootstrap loader".

For this example, it is assumed that the code download is done via a serial communication link, although the program could be adapted to other forms of download. The comments at the beginning of the listing are intended to document the program and its use completely.

An additional comment would be that any static routines (low level routines that are unlikely to change over time) can probably be put into the permanent program memory (on-chip or off-chip ROM or EPROM) along with the bootstrap loader to save program RAM space for other things.

The source code file for this program is available for downloading from the Philips computer bulletin board system. This system is open to all callers, operates 24 hours a day, and can be accessed with modems at 2400, 1200, and 300 baud. The telephone numbers for the BBS are: (800) 451-6644 (in the U.S. only) or (408) 991-2406.

Basic Diagram of RAM Connections for the Boot Loader



RAM loader program for 80C51 family applications   AN440

```
;=============================================================================

;                   Bootstrap Loader for Hexadecimal Files
;                    written by G. Goodhue, Philips Electronics

; This program allows downloading a hexadecimal program file over an
; asynchronous serial link to a code RAM in an 80C51 system. The downloaded
; code may then be executed as the main program for the system. This technique
; may be used in a system that normally connects to a host PC so that the code
; may come from a disk and thus be easily updated. The system RAM must be
; wired to the 80C51 system so that it appears as both data and program memory
; (wire the RAM normally, but use the logical AND of RD and PSEN for the
; output enable.)

; To use the bootstrap program, an Intel Hex file is sent through the serial
; port in 8–N–1 format at 9600 baud. The baud rate and format may be altered
; by making small changes in the serial port setup routine (SerStart).

; Note that there is no hardware handshaking (e.g. RTS/CTS or XON/XOFF)
; implemented between the host and the bootstrap system. This was done to keep
; the protocol between the two systems as simple as possible.

; Since the bootstrap program does not echo the data file, there is no chance
; of an overrun unless the 80C51 is running very slowly and/or the
; communication is very fast. An 80C51 running at 11.0592 MHz (the most
; commonly used frequency in systems with serial communication) will be able
; to easily keep up with 38.4K baud communication without handshaking.
;=============================================================================
; The download protocol for this program is as follows:

;  – When the bootstrap program starts up, it sends a prompt character (”=”)
;    up the serial link to the host.

;  – The host may then send the hexadecimal program file down the serial link.
;    At any time, the host may send an escape character (1B hex) to abort and
;    restart the download process from scratch, beginning from the ”=” prompt.
;    This procedure may be used to restart if a download error occurs.

;  – At the end of a hex file download, a colon (”:”) prompt is returned. If
;    an error or other suspicious circumstance occurred, a flag value will
;    also be returned as shown below. The flag is a bit map of possible
;    conditions and so may represent more than one problem. If an error
;    occurs, the bootstrap program will refuse to execute the downloaded
;    program.

;  Exception codes:
;    01 – non–hexadecimal characters found embedded in a data line.
;    02 – bad record type found.
;    04 – incorrect line checksum found.
;    08 – no data found.
;    10 – incremented address overflowed back to zero.
;    20 – RAM data write did not verify correctly.

;  – If a download error occurs, the download may be retried by first sending
;    an escape character. Until the escape is received, the bootstrap program
;    will refuse to accept any data and will echo a question mark (”?”) for
;    any character sent.

;  – After a valid file download, the bootstrap program will send a message
;    containing the file checksum. This is the arithmetic sum of all of the
;    DATA bytes (not addresses, record types, etc.) in the file, truncated to
;    16 bits. This checksum appears in parentheses: ”(abcd)”. Program
;    execution may then be started by telling the bootstrap program the
;    correct starting address. The format for this is to send a slash (”/”)
;    followed by the address in ASCII hexadecimal, followed by a carriage
;    return. Example: ”/8A31<CR>”
;  – If the address is accepted, an at sign (”@”) is returned before executing
;    the jump to the downloaded file.

; The bootstrap loader can be configured to re–map interrupt vectors to the
; downloaded program if jumps to the correct addresses are set up. For
; instance, if the program RAM in the system where this program is to be used
; starts at 8000 hexadecimal, the re–mapped interrupts may begin at 8003 for
; external interrupt 0, etc.

;=============================================================================


$Title(Bootstrap Loader for Hexadecimal Files)
$Date(04–13–92)
$MOD51


;=============================================================================
;                                  Definitions
;=============================================================================


LF          EQU     0Ah                ; Line Feed character.
CR          EQU     0Dh                ; Carriage Return character.
ESC         EQU     1Bh                ; Escape character.
StartChar   EQU     ’:’                ; Line start character for hex file.
Slash       EQU     ’/’                ; Go command character.
Skip        EQU     13                 ; Value for ”Skip” state.

Ch          DATA     0Fh               ; Last character received.
State       DATA     10h               ; Identifies the state in process.
DataByte    DATA     11h               ; Last data byte received.
ByteCount   DATA     12h               ; Data byte count from current line.
HighAddr    DATA     13h               ; High and low address bytes from the
LowAddr     DATA     14h               ;   current data line.
RecType     DATA     15h               ; Line record type for this line.
ChkSum      DATA     16h               ; Calculated checksum received.
HASave      DATA     17h               ; Saves the high and low address bytes
LASave      DATA     18h               ;   from the last data line.
FilChkHi    DATA     19h               ; File checksum high byte.
FilChkLo    DATA     1Ah               ; File checksum low byte.

Flags       DATA     20h               ; State condition flags.
HexFlag     BIT      Flags.0           ; Hex character found.
EndFlag     BIT      Flags.1           ; End record found.
DoneFlag    BIT      Flags.2           ; Processing done (end record or some
                                       ;   kind of error.

EFlags      DATA     21h               ; Exception flags.
ErrFlag1    BIT      EFlags.0          ;   Non–hex character embedded in data.
ErrFlag2    BIT      EFlags.1          ;   Bad record type.
ErrFlag3    BIT      EFlags.2          ;   Bad line checksum.
ErrFlag4    BIT      EFlags.3          ;   No data found.
ErrFlag5    BIT      EFlags.4          ;   Incremented address overflow.
ErrFlag6    BIT      EFlags.5          ;   Data storage verify error.
DatSkipFlag BIT      Flags.3           ; Any data found should be ignored.
```

```
;===============================================================================
;                           Reset and Interrupt Vectors
;===============================================================================

; The following are dummy labels for re–mapped interrupt vectors. The
; addresses should be changed to match the memory map of the target system.

ExInt0      EQU     8003h               ; Remap address for ext interrupt 0.
T0Int       EQU     800Bh               ; Timer 0 interrupt.
ExInt1      EQU     8013h               ; External interrupt 1.
T1Int       EQU     801Bh               ; Timer 1 interrupt.
SerInt      EQU     8023h               ; Serial port interrupt.

            ORG      0000h
            LJMP     Start              ; Go to the downloader program.

; The following are intended to allow re–mapping the interrupt vectors to the
; users downloaded program. The jump addresses should be adjusted to reflect
; the memory mapping used in the actual application.

; Other (or different) interrupt vectors may need to be added if the target
; processor is not an 80C51.

            ORG      0003h
;            LJMP     ExInt0            ; External interrupt 0.
            RETI

            ORG      000Bh
;            LJMP     T0Int             ; Timer 0 interrupt.
            RETI

            ORG      0013h
;            LJMP     ExInt1            ; External interrupt 1.
            RETI

            ORG      001Bh
;            LJMP     T1Int             ; Timer 1 interrupt.
            RETI

            ORG      0023h
;            LJMP     SerInt            ; Serial port interrupt.
            RETI

;===============================================================================
;                           Reset and Interrupt Vectors
;===============================================================================

Start:      MOV      IE,#0              ; Turn off all interrupts.
            MOV      SP,#5Fh            ; Start stack near top of ’51 RAM.
            ACALL    SerStart           ; Setup and start serial port.
            ACALL    CRLF               ; Send a prompt that we are here.
            MOV      A,#’=’             ;   ”<CRLF> =”
            ACALL    PutChar
            ACALL    HexIn              ; Try to read hex file from serial port.

            ACALL    ErrPrt             ; Send a message for any errors or
                                        ;   warnings that were noted.
            MOV      A,EFlags           ; We want to get stuck if a fatal
            JZ       HexOK              ;   error occurred.

ErrLoop:    MOV      A,#’?’             ; Send a prompt to confirm that we
            ACALL    PutChar            ;   are ’stuck’.  ” ? ”
            ACALL    GetChar            ; Wait for escape char to flag reload.
            SJMP     ErrLoop

HexOK:      MOV      EFlags,#0          ; Clear errors flag in case we re–try.
            ACALL    GetChar            ; Look for GO command.
            CJNE     A,#Slash,HexOK     ; Ignore other characters received.

            ACALL    GetByte            ; Get the GO high address byte.
            JB       ErrFlag1,HexOK     ; If non–hex char found, try again.
            MOV      HighAddr,DataByte  ; Save upper GO address byte.

            ACALL    GetByte            ; Get the GO low address byte.
            JB       ErrFlag1,HexOK     ; If non–hex char found, try again.
            MOV      LowAddr,DataByte   ; Save the lower GO address byte.

            ACALL    GetChar            ; Look for CR.
            CJNE     A,#CR,HexOK        ; Re–try if CR not there.
```

```
; All conditions are met, so hope the data file and the GO address are all
;   correct, because now we're committed.

            MOV      A,#'@'             ; Send confirmation to GO. " @ "
            ACALL    PutChar
            JNB      TI,$               ; Wait for completion before GOing.
            PUSH     LowAddr            ; Put the GO address on the stack,
            PUSH     HighAddr           ;   so we can Return to it.
            RET                         ; Finally, go execute the user program!

;==============================================================================
;                          Hexadecimal File Input Routine
;==============================================================================

HexIn:      CLR      A                  ; Clear out some variables.
            MOV      State,A
            MOV      Flags,A
            MOV      HighAddr,A
            MOV      LowAddr,A
            MOV      HASave,A
            MOV      LASave,A
            MOV      ChkSum,A
            MOV      FilChkHi,A
            MOV      FilChkLo,A
            MOV      EFlags,A
            SETB     ErrFlag4           ; Start with a 'no data' condition.

StateLoop:  ACALL    GetChar            ; Get a character for processing.
            ACALL    AscHex             ; Convert ASCII–hex character to hex.
            MOV      Ch,A               ; Save result for later.
            ACALL    GoState            ; Go find the next state based on
                                        ;   this char.
            JNB      DoneFlag,StateLoop ; Repeat until done or terminated.

            ACALL    PutChar            ; Send the file checksum back as
            MOV      A,#'('             ;   confirmation. " (abcd) "
            ACALL    PutChar
            MOV      A,FilChkHi
            ACALL    PrByte
            MOV      A,FilChkLo
            ACALL    PrByte
            MOV      A,#')'
            ACALL    PutChar
            ACALL    CRLF
            RET                         ; Exit to main program.

; Find and execute the state routine pointed to by "State".

GoState:    MOV      A,State            ; Get current state.
            ANL      A,#0Fh             ; Insure branch is within table range.
            RL       A                  ; Adjust offset for 2 byte insts.
            MOV      DPTR,#StateTable
            JMP      @A+DPTR            ; Go to appropriate state.

StateTable: AJMP     StWait             ;  0 – Wait for start.
            AJMP     StLeft             ;  1 – First nibble of count.
            AJMP     StGetCnt           ;  2 – Get count.
            AJMP     StLeft             ;  3 – First nibble of address byte 1.
            AJMP     StGetAd1           ;  4 – Get address byte 1.
            AJMP     StLeft             ;  5 – First nibble of address byte 2.
            AJMP     StGetAd2           ;  6 – Get address byte 2.
            AJMP     StLeft             ;  7 – First nibble of record type.
            AJMP     StGetRec           ;  8 – Get record type.
            AJMP     StLeft             ;  9 – First nibble of data byte.
            AJMP     StGetDat           ; 10 – Get data byte.
            AJMP     StLeft             ; 11 – First nibble of checksum.
            AJMP     StGetChk           ; 12 – Get checksum.
            AJMP     StSkip             ; 13 – Skip data after error condition.
            AJMP     BadState           ; 14 – Should never get here.
            AJMP     BadState           ; 15 –   "      "    "    "
```

```
; This state is used to wait for a line start character. Any other characters
;   received prior to the line start are simply ignored.

StWait:     MOV      A,Ch                ; Retrieve input character.
            CJNE     A,#StartChar,SWEX  ; Check for line start.
            INC      State               ; Received line start.
SWEX:       RET

; Process the first nibble of any hex byte.

StLeft:     MOV      A,Ch                ; Retrieve input character.
            JNB      HexFlag,SLERR       ; Check for hex character.
            ANL      A,#0Fh              ; Isolate one nibble.
            SWAP     A                   ; Move nibble too upper location.
            MOV      DataByte,A          ; Save left/upper nibble.
            INC      State               ; Go to next state.
            RET                          ; Return to state loop.

SLERR:      SETB     ErrFlag1            ; Error – non–hex character found.
            SETB     DoneFlag            ; File considered corrupt. Tell main.
            RET

; Process the second nibble of any hex byte.

StRight:    MOV      A,Ch                ; Retrieve input character.
            JNB      HexFlag,SRERR       ; Check for hex character.
            ANL      A,#0Fh              ; Isolate one nibble.
            ORL      A,DataByte          ; Complete one byte.
            MOV      DataByte,A          ; Save data byte.
            ADD      A,ChkSum            ; Update line checksum,
            MOV      ChkSum,A            ;   and save.
            RET                          ; Return to state loop.

SRERR:      SETB     ErrFlag1            ; Error – non–hex character found.
            SETB     DoneFlag            ; File considered corrupt. Tell main.
            RET

; Get data byte count for line.

StGetCnt:   ACALL    StRight             ; Complete the data count byte.
            MOV      A,DataByte
            MOV      ByteCount,A
            INC      State               ; Go to next state.
            RET                          ; Return to state loop.

; Get upper address byte for line.

StGetAd1:   ACALL    StRight             ; Complete the upper address byte.
            MOV      A,DataByte
            MOV      HighAddr,A          ; Save new high address.
            INC      State               ; Go to next state.
            RET                          ; Return to state loop.

; Get lower address byte for line.

StGetAd2:   ACALL    StRight             ; Complete the lower address byte.
            MOV      A,DataByte
            MOV      LowAddr,A           ; Save new low address.
            INC      State               ; Go to next state.
            RET                          ; Return to state loop.

; Get record type for line.

StGetRec:   ACALL    StRight             ; Complete the record type byte.
            MOV      A,DataByte
            MOV      RecType,A           ; Get record type.
            JZ       SGRDat              ; This is a data record.
            CJNE     A,#1,SGRErr         ; Check for end record.
            SETB     EndFlag             ; This is an end record.
            SETB     DatSkipFlag         ; Ignore data embedded in end record.
            MOV      State,#11           ; Go to checksum for end record.
            SJMP     SGREX
```

```
SGRDat:     INC      State              ; Go to next state.
SGREX:      RET                         ; Return to state loop.

SGRErr:     SETB     ErrFlag2           ; Error, bad record type.
            SETB     DoneFlag           ; File considered corrupt. Tell main.
            RET

; Get a data byte.

StGetDat:   ACALL    StRight            ; Complete the data byte.
            JB       DatSkipFlag,SGD1   ; Don’t process the data if the skip
                                        ;   flag is on.
            ACALL    Store              ; Store data byte in memory.

            MOV      A,DataByte         ; Update the file checksum,
            ADD      A,FilChkLo         ;   which is a two–byte summation of
            MOV      FilChkLo,A         ;   all data bytes.
            CLR      A
            ADDC     A,FilChkHi
            MOV      FilChkHi,A
            MOV      A,DataByte
SGD1:       DJNZ     ByteCount,SGDEX    ; Last data byte?
            INC      State              ; Done with data, go to next state.
            SJMP     SGDEX2

SGDEX:      DEC      State              ; Set up state for next data byte.
SGDEX2:     RET                         ; Return to state loop.

; Get checksum.

StGetChk:   ACALL    StRight            ; Complete the checksum byte.
            JNB      EndFlag,SGC1       ; Check for an end record.
            SETB     DoneFlag           ; If this was an end record,
            SJMP     SGCEX              ;  we are done.

SGC1:       MOV      A,ChkSum           ; Get calculated checksum.
            JNZ      SGCErr             ; Result should be zero.
            MOV      ChkSum,#0          ; Preset checksum for next line.
            MOV      State,#0           ; Line done, go back to wait state.
            MOV      LASave,LowAddr     ; Save address byte from this line for
            MOV      HASave,HighAddr    ;   later check.
SGCEX:      RET                         ; Return to state loop.

SGCErr:     SETB     ErrFlag3           ; Line checksum error.
            SETB     DoneFlag           ; File considered corrupt. Tell main.
            RET

; This state used to skip through any additional data sent, ignoring it.

StSkip:     RET                         ; Return to state loop.

; A place to go if an illegal state comes up somehow.

BadState:   MOV      State,#Skip        ; If we get here, something very bad
            RET                         ;   happened, so return to state loop.

; Store – Save data byte in external RAM at specified address.

Store:      MOV      DPH,HighAddr       ; Set up external RAM address in DPTR.
            MOV      DPL,LowAddr
            MOV      A,DataByte
            MOVX     @DPTR,A            ; Store the data.

            MOVX     A,@DPTR            ; Read back data for integrity check.
            CJNE     A,DataByte,StoreErr ; Is read back OK?

            CLR      ErrFlag4           ; Show that we’ve found some data.
            INC      DPTR               ; Advance to the next addr in sequence.
            MOV      HighAddr,DPH       ; Save the new address
            MOV      LowAddr,DPL
            CLR      A
            CJNE     A,HighAddr,StoreEx ; Check for address overflow
            CJNE     A,LowAddr,StoreEx ;   (both bytes are 0).
            SETB     ErrFlag5           ; Set warning for address overflow.
```

```
StoreEx:    RET

StoreErr:   SETB     ErrFlag6          ; Data storage verify error.
            SETB     DoneFlag          ; File considered corrupt. Tell main.
            RET


;===============================================================================
;                                  Subroutines
;===============================================================================

; Subroutine summary:

; SerStart – Serial port setup and start.
; GetChar  – Get a character from the serial port for processing.
; GetByte  – Get a hex byte from the serial port for processing.
; PutChar  – Output a character to the serial port.
; AscHex   – See if char in ACC is ASCII–hex and if so convert to hex nibble.
; HexAsc   – Convert a hexadecimal nibble to its ASCII character equivalent.
; ErrPrt   – Return any error codes to our host.
; CRLF     – output a carriage return / line feed pair to the serial port.
; PrByte   – Send a byte out the serial port in ASCII hexadecimal format.

; SerStart – Serial port setup and start.

SerStart:   MOV      A,PCON            ; Make sure SMOD is off.
            CLR      ACC.7
            MOV      PCON,A
            MOV      TH1,#0FDh         ; Set up timer 1.
            MOV      TL0,#0FDh
            MOV      TMOD,#20h
            MOV      TCON,#40h
            MOV      SCON,#52h         ; Set up serial port.
            RET

; GetByte – Get a hex byte from the serial port for processing.

GetByte:    ACALL    GetChar           ; Get first character of byte.
            ACALL    AscHex            ; Convert to hex.
            MOV      Ch,A              ; Save result for later.
            ACALL    StLeft            ; Process as top nibble of a hex byte.
            ACALL    GetChar           ; Get second character of byte.
            ACALL    AscHex            ; Convert to hex.
            MOV      Ch,A              ; Save result for later.
            ACALL    StRight           ; Process as bottom nibble of hex byte.
            RET

; GetChar – Get a character from the serial port for processing.

GetChar:    JNB      RI,$              ; Wait for receiver flag.
            CLR      RI                ; Clear receiver flag.
            MOV      A,SBUF            ; Read character.
            CJNE     A,#ESC,GCEX       ; Re–start immediately if Escape char.
            LJMP     Start
GCEX:       RET

; PutChar – Output a character to the serial port.

PutChar:    JNB      TI,$              ; Wait for transmitter flag.
            CLR      TI                ; Clear transmitter flag.
            MOV      SBUF,A            ; Send character.
            RET

; AscHex – See if char in ACC is ASCII–hex and if so convert to a hex nibble.
;   Returns nibble in A, HexFlag tells if char was really hex. The ACC is not
;   altered if the character is not ASCII hex. Upper and lower case letters
;   are recognized.
```

```
AscHex:     CJNE     A,#'0',AH1         ; Test for ASCII numbers.
AH1:        JC       AHBad              ; Is character is less than a '0'?
            CJNE     A,#'9'+1,AH2       ; Test value range.
AH2:        JC       AHVal09            ; Is character is between '0' and '9'?

            CJNE     A,#'A',AH3         ; Test for upper case hex letters.
AH3:        JC       AHBad              ; Is character is less than an 'A'?
            CJNE     A,#'F'+1,AH4       ; Test value range.
AH4:        JC       AHValAF            ; Is character is between 'A' and 'F'?

            CJNE     A,#'a',AH5         ; Test for lower case hex letters.
AH5:        JC       AHBad              ; Is character is less than an 'a'?
            CJNE     A,#'f'+1,AH6       ; Test value range.
AH6:        JNC      AHBad              ; Is character is between 'a' and 'f'?
            CLR      C
            SUBB     A,#27h             ; Pre-adjust character to get a value.
            SJMP     AHVal09            ; Now treat as a number.

AHBad:      CLR      HexFlag            ; Flag char as non-hex, don't alter.
            SJMP     AHEX               ; Exit
AHValAF:    CLR      C
            SUBB     A,#7               ; Pre-adjust character to get a value.
AHVal09:    CLR      C
            SUBB     A,#'0'             ; Adjust character to get a value.
            SETB     HexFlag            ; Flag character as 'good' hex.
AHEX:       RET


; HexAsc - Convert a hexadecimal nibble to its ASCII character equivalent.

HexAsc:     ANL      A,#0Fh             ; Make sure we're working with only
                                        ;   one nibble.
            CJNE      A,#0Ah,HA1        ; Test value range.
HA1:        JC        HAVal09           ; Value is 0 to 9.
            ADD       A,#7              ; Value is A to F, extra adjustment.
HAVal09:    ADD       A,#'0'            ; Adjust value to ASCII hex.
            RET

; ErrPrt - Return an error code to our host.

ErrPrt:     MOV       A,#':'            ; First, send a prompt that we are
            CALL      PutChar           ;   still here.
            MOV       A,EFlags          ; Next, print the error flag value if
            JZ        ErrPrtEx          ;   it is not 0.
            CALL      PrByte
ErrPrtEx:   RET

; CRLF - output a carriage return / line feed pair to the serial port.

CRLF:       MOV       A,#CR
            CALL      PutChar
            MOV       A,#LF
            CALL      PutChar
            RET

; PrByte - Send a byte out the serial port in ASCII hexadecimal format.

PrByte:     PUSH      ACC               ; Print ACC contents as ASCII hex.
            SWAP      A
            CALL      HexAsc            ; Print upper nibble.
            CALL      PutChar
            POP       ACC
            CALL      HexAsc            ; Print lower nibble.
            CALL      PutChar
            RET


;============================================================================

            END
```