```
; XMODEM/CRC Sender/Receiver for the Microtan 65
;
; Adapted by Alan Maunder from a version created by Daryl Rictor
;
; A simple file transfer program to allow transfers between the MT65 and a
; console device utilizing the X-modem/CRC transfer protocol.  Requires
; ~1330 bytes of either RAM or ROM, 132 bytes of RAM for the receive buffer,
; and 12 bytes ($50-$5B) of zero page RAM for variable storage.
;
;***************************************************************************
; This implementation of XMODEM/CRC does NOT conform strictly to the
; XMODEM protocol standard in that it (1) does not accurately time character
; reception or (2) fall back to the Checksum mode.

; (1) For timing, it uses a crude timing loop to provide approximate
; delays.
;
; (2) Most modern terminal programs support XMODEM/CRC which can detect a
; wider range of transmission errors so the fallback to the simple checksum
; calculation was not implemented to save space.
;***************************************************************************
;
; Files transferred via XMODEM-CRC will have the load address contained in
; the first two bytes in little-endian format:
;  FIRST BLOCK
;     offset(0) = lo(load start address),
;     offset(1) = hi(load start address)
;     offset(2) = data byte (0)
;     offset(n) = data byte (n-2)
;
; Subsequent blocks
;     offset(n) = data byte (n)
;
; Note, XMODEM send 128 byte blocks.  If the block of memory that
; you wish to save is smaller than the 128 byte block boundary, then
; the last block will be padded with zeros.  Upon reloading, the
; data will be written back to the original location.  In addition, the
; padded zeros WILL also be written into RAM, which could overwrite other
; data.
;
;-------------------------- The Code ----------------------------
;
; zero page variables
;
ICHAR       EPZ   $01          ; Keyboard Character
HXPKL       EPZ   $13          ; HEXPCK lo byte
HXPKH       EPZ   $14          ; HEXPCK hi byte
LASTBK      EPZ   $50          ; flag for last block
BLKNO       EPZ   $51          ; block number
ERRCNT      EPZ   $52          ; error counter 10 is the limit
BFLAG       EPZ   $53          ; block flag

CRC         EPZ   $54          ; CRC low byte  (two byte variable)
CRCH        EPZ   $55          ; CRC high byte

PTR         EPZ   $56          ; data pointer (two byte variable)
PTRH        EPZ   $57          ;  "    "

EOFP        EPZ   $58          ; end of file address pointer (2 bytes)
EOFPH       EPZ   $59          ;  "       "          "          "

RETRY       EPZ   $5A          ; RETRY counter
RETRY2      EPZ   $5B          ; 2nd counter
```

```
;
; non-zero page variables and buffers
;
RBUFF       EQU   $0200       ; temp 132 byte receive buffer
                              ; (upper screen memory area)
CRCLO       EQU   $EE00       ; Two 256-byte tables for quick lookup
CRCHI       EQU   $EF00       ; (should be page-aligned for speed)
                              ; Must be changed if code relocated
DATA        EQU   $BFD0       ;
STATUS      EQU   $BFD1       ;
COMMND      EQU   $BFD2       ;
CONTRL      EQU   $BFD3       ;
EXIT        EQU   $FC00       ;
HEXPCK      EQU   $FF28       ;
OUTCR       EQU   $FE73       ;
OPCHR       EQU   $FE75       ;
POLLKB      EQU   $FDFA       ;
;
;
; XMODEM Control Character Constants
;SOH        =     $01         ; start block
;EOT        =     $04         ; end of text marker
;ACK        =     $06         ; good block acknowledged
;NAK        =     $15         ; bad block acknowledged
;CAN        =     $18         ; cancel (not standard, not supported)
;CR         =     $0D         ; carriage return
;LF         =     $0A         ; line feed
;ESC        =     $1B         ; ESC to exit

;
;^^^^^^^^^^^^^^^^^^^^^^ Start of Program ^^^^^^^^^^^^^^^^^^^^^^
;
; Microtan 65 Xmodem/CRC transfer routine for Assember
; By Alan Maunder
;
; v2.0  released on 27 December 2011.
;
;
            ORG   $1AD0       ; Start of program (adjust to your needs)
;
; Enter this routine with the beginning address stored in the zero page address
; pointed to by PTR & PTRH and the ending address stored in the zero page
; address pointed to by EOFP & EOFPH.
;
;
            JSR   INIT        ; initialise ACIA
            JSR   Q1MSG       ; CR, "S OR R?"
            CLI
POLKB1      JSR   POLLKB      ; Get response
            LDA   ICHAR
            CMP   #'S
            BEQ   XMODTX
            CMP   #'R
            BEQ   JXMODR
            CMP   #$1B        ; <ESC>?
            BNE   POLKB1
            RTS               ; exit program
JXMODR      JMP   XMODRX
XMODTX      JSR   OPCHR
            JSR   Q2MSG       ; CR, "START:"
            JSR   GETADD      ; get start address
            LDA   HXPKL
            STA   PTR
```

```
                LDA     HXPKH
                STA     PTRH
                JSR     Q3MSG           ; CR, “END    :”
                JSR     GETADD          ; get end address
                LDA     HXPKL
                STA     EOFP
                LDA     HXPKH
                STA     EOFPH
                JSR     OUTCR
                JSR     PRMSG           ; send prompt and info
                LDA     #$00            ;
                STA     ERRCNT          ; set error counter to 0
                STA     LASTBK          ; set flag to false
                LDA     #$01            ;
                STA     BLKNO           ; set block # to 1
WAITCR          LDA     #$FF            ; 3 seconds
                STA     RETRY2          ;
                JSR     GETBYT          ;
                BCC     WAITCR          ; wait for something to come in...
                CMP     #$43            ; is it the "C" to start a CRC transfer?
                BEQ     SETADD          ; yes
                CMP     #$1B            ; is it a cancel? <Esc> Key
                BNE     WAITCR          ; No, wait for another character
                JSR     PRTABT          ; Print abort MSG
                RTS                     ; and exit program
;
SETADD          LDA     #$01            ; manually load block number
                STA     RBUFF           ; into 1st byte
                LDA     #$FE            ; load 1's comp of block #
                STA     RBUFF+1         ; into 2nd byte
                LDA     PTR             ; load low byte of start address
                STA     RBUFF+2         ; into 3rd byte
                LDA     PTRH            ; load hi byte of start address
                STA     RBUFF+3         ; into 4th byte
                LDX     #$04            ;
                LDY     #$00            ; initialise data block offset to 0
                JMP     LDBUF1          ; jump into buffer load routine
;
LDBUFF          LDA     LASTBK          ; Was the last block sent?
                BEQ     LDBUF0          ; no, send the next one
                JMP     DONE            ; yes, we're DONE
LDBUF0          LDX     #$02            ; initialise pointers
                LDY     #$00            ;
                INC     BLKNO           ; INC block counter
                LDA     BLKNO           ;
                STA     RBUFF           ; save in 1st byte of buffer
                EOR     #$FF            ;
                STA     RBUFF+1         ; save 1's comp of BLKNO next
;
LDBUF1          LDA     (PTR),Y         ; save 128 bytes of data
                STA     RBUFF,X         ;
LDBUF2          SEC                     ;
                LDA     EOFP            ;
                SBC     PTR             ; Are we at the last address?
                BNE     LDBUF4          ; no, INC pointer and continue
                LDA     EOFPH           ;
                SBC     PTRH            ;
                BNE     LDBUF4          ;
                INC     LASTBK          ; Yes, Set last byte flag
LDBUF3          INX                     ;
                CPX     #$82            ; Are we at the end of the 128 byte block?
                BEQ     SCALCR          ; Yes, calculate CRC
                LDA     #$00            ; Fill rest of 128 bytes with $00
```

```
                STA     RBUFF,X         ;
                BEQ     LDBUF3          ; Branch always
;
LDBUF4          INC     PTR             ; INC address pointer
                BNE     LDBUF5          ;
                INC     PTRH            ;
LDBUF5          INX                     ;
                CPX     #$82            ; last byte in block?
                BNE     LDBUF1          ; no, get the next
SCALCR          JSR     CALCRC
                LDA     CRCH            ; save high byte of CRC to buffer
                STA     RBUFF,Y         ;
                INY                     ;
                LDA     CRC             ; save low byte of CRC to buffer
                STA     RBUFF,Y         ;
RESEND          LDX     #$00            ;
                LDA     #$01
                JSR     PUTCHR          ; send SOH
SENDBK          LDA     RBUFF,X         ; Send 132 bytes in buffer to the console
                JSR     PUTCHR          ;
                INX                     ;
                CPX     #$84            ; last byte?
                BNE     SENDBK          ; no, get next
                LDA     #$FF            ; yes, set 3 second delay
                STA     RETRY2          ; and
                JSR     GETBYT          ; Wait for Ack/Nack
                BCC     SETERR          ; No char received after 3 seconds, RESEND
                CMP     #$06            ; Char received... is it:
                BEQ     LDBUFF          ; ACK, send next block
                CMP     #$15            ;
                BEQ     SETERR          ; NAK, INC errors and resend
                CMP     #$1B            ;
                BEQ     PRTABT          ; ESC pressed to abort
                                        ; fall through to error counter
SETERR          INC     ERRCNT          ; INC error counter
                LDA     ERRCNT          ;
                CMP     #$0A            ; are there 10 errors? (Xmodem spec for failure)
                BNE     RESEND          ; no, resend block
PRTABT          JSR     FLUSH           ; yes, too many errors, flush buffer,
                JSR     PRERR           ; print error message
                RTS                     ; and exit program

DONE            JSR     PRGOOD          ; All done..Print message
                RTS                     ; and exit program
;
;
;
XMODRX          JSR     OPCHR
                JSR     OUTCR
                JSR     PRMSG           ; send prompt and info
                LDA     #$01
                STA     BLKNO           ; set block # to 1
                STA     BFLAG           ; set flag to get address from block 1
CRCCHK          LDA     #$43            ; "C" start with CRC mode
                JSR     PUTCHR          ; send it
                LDA     #$FF
                STA     RETRY2          ; set loop counter for ~3 SEC delay
                LDA     #$00
                STA     CRC
                STA     CRCH            ; initialise CRC value
                JSR     GETBYT          ; wait for input
                BCS     GOTBYT          ; byte received, process it
                BCC     CRCCHK          ; RESEND "C"
```

```
STBLCK      LDA   #$FF        ;
            STA   RETRY2      ; set loop counter for ~3 SEC delay
            JSR   GETBYT      ; get first byte of block
            BCC   STBLCK      ; timed out, keep waiting...
GOTBYT      CMP   #$1B        ; quitting <ESC>?
            BNE   GOTBY1      ; no
;           LDA   #$FE        ; Error code in "A"
            BRK               ; YES - do BRK or change to RTS if desired
GOTBY1      CMP   #$01        ; start of block <SOH>?
            BEQ   BEGBLK      ; yes
            CMP   #$04        ;
            BNE   BADCRC      ; Not SOH or EOT, so flush buffer & send NAK
            JMP   RDONE       ; EOT - all done!
BEGBLK      LDX   #$00
GETBLK      LDA   #$FF        ; 3 SEC window to receive characters
            STA   RETRY2      ;
GETBK1      JSR   GETBYT      ; get next character
            BCC   BADCRC      ; char received error, flush and send NAK
GETBK2      STA   RBUFF,X     ; good char, save it in the received buffer
            INX               ; INC buffer pointer
            CPX   #$84        ; <01> <FE> <128 bytes> <CRCH> <CRCL>
            BNE   GETBLK      ; get 132 characters
            LDX   #$00        ;
            LDA   RBUFF,X     ; get block # from buffer
            CMP   BLKNO       ; compare to expected block #
            BEQ   GOODB1      ; matched!
            JSR   PRERR       ; Unexpected block number - abort
            JSR   FLUSH       ; mismatched - flush buffer and then do BRK
            LDA   #$FD        ; put error code in "A" if desired
            BRK               ; unexpected block # - fatal error - BRK or RTS
GOODB1      EOR   #$FF        ; 1's comp of block #
            INX               ;
            CMP   RBUFF,X     ; compare with expected 1's comp of block #
            BEQ   GOODB2      ; matched!
            JSR   PRERR       ; Unexpected block number - abort
            JSR   FLUSH       ; mismatched - flush buffer and then do BRK
            LDA   #$FC        ; put error code in "A" if desired
            BRK               ; bad 1's comp of block#
GOODB2      JSR   CALCRC      ; calculate CRC
            LDA   RBUFF,Y     ; get hi CRC from buffer
            CMP   CRCH        ; compare to calculated hi CRC
            BNE   BADCRC      ; bad CRC, send NAK
            INY               ;
            LDA   RBUFF,Y     ; get lo CRC from buffer
            CMP   CRC         ; compare to calculated lo CRC
            BEQ   GOODCR      ; good CRC
BADCRC      JSR   FLUSH       ; FLUSH the input port
            LDA   #$15        ;
            JSR   PUTCHR      ; send NAK to RESEND block
            JMP   STBLCK      ; start over, get the block again
GOODCR      LDX   #$02        ;
            LDA   BLKNO       ; get the block number
            CMP   #$01        ; 1st block?
            BNE   COPYBK      ; no, copy all 128 bytes
            LDA   BFLAG       ; is it really block 1, not block 257, 513 etc.
            BEQ   COPYBK      ; no, copy all 128 bytes
            LDA   RBUFF,X     ; get target address from 1st 2 bytes of block 1
            STA   PTR         ; save low address
            INX               ;
            LDA   RBUFF,X     ; get high address
            STA   PTR+1       ; save it
            INX               ; point to first byte of data
```

```
                DEC     BFLAG           ; set the flag so we won't get another address
COPYBK          LDY     #$00            ; set offset to zero
COPYB3          LDA     RBUFF,X         ; get data byte from buffer
                STA     (PTR),Y         ; save to target
                INC     PTR             ; point to next address
                BNE     COPYB4          ; did it step over page boundary?
                INC     PTR+1           ; adjust high address for page crossing
COPYB4          INX                     ; point to next data byte
                CPX     #$82            ; is it the last byte
                BNE     COPYB3          ; no, get the next one
INCBLK          INC     BLKNO           ; DONE.  INC the block #
                LDA     #$06            ; send ACK
                JSR     PUTCHR          ;
                JMP     STBLCK          ; get next block
;
RDONE           LDA     #$06            ; last block, send ACK
                JSR     PUTCHR          ;
                JSR     FLUSH           ; get leftover characters, if any
                JSR     PRGOOD          ;
                RTS                     ; and exit program
;
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
;=========================================================================
; Subroutines
;
;I/O Device Routines for the 6551 ACIA.
;
;
INIT            LDA     #$1F            ; 19.2K/8/N/1
                STA     CONTRL          ; control reg
                LDA     #$0B            ; N parity/echo off/rx int off/ dtr active low
                STA     COMMND          ; command reg
                RTS                     ; done
;
; "GETCHR" routine will scan the input port for a character.  It will
; return without waiting with the Carry flag clear if no character is
; present or return with the Carry flag set and the character in the "A"
; register if one was present.
;
GETCHR          CLC                     ; no char present
                LDA     STATUS          ; get Serial port status
                AND     #$08            ; mask rcvr full bit
                BEQ     GETCH2          ; if not char, DONE
                LDA     DATA            ; else get char
                SEC                     ; and set the Carry Flag
GETCH2          RTS                     ; done
;
; "PUTCHR" routine will write one byte to the output port.  It is all right
; if this routine waits for the port to be ready.  It is assumed that the
; character was send upon return from this routine.
;
;
PUTCHR          PHA                     ; save registers
PUTCH1          LDA     STATUS          ; serial port status
                AND     #$10            ; is tx buffer empty
                BEQ     PUTCH1          ; no, go back AND test it again
                PLA                     ; yes, get char to send
                STA     DATA            ; put character to Port
                RTS                     ; done

; Subroutines
;
;
```

```
GETBYT      LDA   #$00        ; wait for char input AND cycle timing loop
            STA   RETRY       ; set low value of timing loop
CKLOOP      JSR   GETCHR      ; get char from serial port, don't wait
            BCS   GETBY1      ; got one, so exit
            DEC   RETRY       ; no character received, so DEC counter
            BNE   CKLOOP      ;
            DEC   RETRY2      ; DEC hi byte of counter
            BNE   CKLOOP      ; look for character again
            CLC               ; if loop times out, CLC, else SEC AND return
GETBY1      RTS               ; with character in "A"
;
;
FLUSH       LDA   #$70        ; flush receive buffer
            STA   RETRY2      ; flush until empty for ~1 SEC.
FLUSH1      JSR   GETBYT      ; read the port
            BCS   FLUSH       ; if char received, wait for another
            RTS               ; else done
;
GETADD      JSR   POLLKB      ; wait for character
            LDA   ICHAR       ; get it from ICHAR
            CMP   #$0D        ; carriage return?
            BEQ   GOPACK      ; yes – pack it
            JSR   OPCHR       ; else display it
            JMP   GETADD      ; get next character
GOPACK      LDY   #$FF        ; set to get first character
            JSR   HEXPCK      ; pack it
            RTS
;
;Text Messages
;
Q1MSG       LDX   #$00        ;
Q1MSG1      LDA   Q1TXT,X
            BEQ   Q1MSG2
            JSR   OPCHR
            INX
            BNE   Q1MSG1
Q1TXT       BYT   13, 'S,' ,'O
            BYT   'R,' ' ,'R,'?
            BYT   0
Q1MSG2      RTS
;
Q2MSG       LDX   #$00        ;
Q2MSG1      LDA   Q2TXT,X
            BEQ   Q2MSG2
            JSR   OPCHR
            INX
            BNE   Q2MSG1
Q2TXT       BYT   13,'S,'T,'A
            BYT   'R,'T,':
            BYT   0
Q2MSG2      RTS
;
Q3MSG       LDX   #$00        ;
Q3MSG1      LDA   Q3TXT,X
            BEQ   Q3MSG2
            JSR   OPCHR
            INX
            BNE   Q3MSG1
Q3TXT       BYT   13,'E,'N,'D
            BYT   ' ,' ,':,0
Q3MSG2      RTS
;
PRMSG       LDX   #$00        ; print starting message
```

```
PRMSG1      LDA   MSG,X
            BEQ   PRMSG2
            JSR   PUTCHR
            INX
            BNE   PRMSG1
PRMSG2      RTS
MSG         BYT   "Begin XMODEM transfer.  Press <Esc> to abort..."
            BYT   13, 10          ;CR, LF,0
;
PRERR       LDX   #$00            ; print Error message
PRERR1      LDA   ERRMSG,X
            BEQ   PRERR2
            JSR   PUTCHR
            INX
            BNE   PRERR1
PRERR2      RTS
ERRMSG      BYT   "Transfer Error!"
            BYT   13,10           ;CR, LF
            BYT   0
;
;
PRGOOD      LDX   #$00            ; print Good Transfer message
PGOOD1      LDA   GDMSG,X
            BEQ   PGOOD2
            JSR   PUTCHR
            INX
            BNE   PGOOD1
PGOOD2      RTS
GDMSG       BYT                   ;EOT,CR,LF,EOT,CR,LF,EOT,CR,LF,CR,LF
            BYT   "Transfer Successful!"
            BYT   13,10           ;CR, LF
            BYT   0
;
;  CRC subroutines
;
CALCRC      LDA   #$00            ; yes, calculate the CRC for the 128 bytes
            STA   CRC             ;
            STA   CRCH            ;
            LDY   #$02            ;
CALCR1      LDA   RBUFF,Y         ;
            EOR   CRC+1           ; Quick CRC computation with lookup tables
            TAX                   ; updates the two bytes at CRC & CRC+1
            LDA   CRC             ; with the byte send in the "A" register
            EOR   CRCHI,X
            STA   CRC+1
            LDA   CRCLO,X
            STA   CRC
            INY                   ;
            CPY   #$82            ; done yet?
            BNE   CALCR1          ; no, get next
            RTS                   ; y=82 on exit
;
;
;================================================================
;
; The following tables are used to calculate the CRC for the 128 bytes
; in the xmodem data blocks.  You can use these tables if you plan to
; store this program in ROM.  If you choose to build them at run-time,
; then just delete them and define the two labels: CRCLO & CRCHI.
;
; low byte CRC lookup table (should be page aligned)
CRCLO
 BYT $00,$21,$42,$63,$84,$A5,$C6,$E7,$08,$29,$4A,$6B,$8C,$AD,$CE,$EF
```

```
 BYT $31,$10,$73,$52,$B5,$94,$F7,$D6,$39,$18,$7B,$5A,$BD,$9C,$FF,$DE
 BYT $62,$43,$20,$01,$E6,$C7,$A4,$85,$6A,$4B,$28,$09,$EE,$CF,$AC,$8D
 BYT $53,$72,$11,$30,$D7,$F6,$95,$B4,$5B,$7A,$19,$38,$DF,$FE,$9D,$BC
 BYT $C4,$E5,$86,$A7,$40,$61,$02,$23,$CC,$ED,$8E,$AF,$48,$69,$0A,$2B
 BYT $F5,$D4,$B7,$96,$71,$50,$33,$12,$FD,$DC,$BF,$9E,$79,$58,$3B,$1A
 BYT $A6,$87,$E4,$C5,$22,$03,$60,$41,$AE,$8F,$EC,$CD,$2A,$0B,$68,$49
 BYT $97,$B6,$D5,$F4,$13,$32,$51,$70,$9F,$BE,$DD,$FC,$1B,$3A,$59,$78
 BYT $88,$A9,$CA,$EB,$0C,$2D,$4E,$6F,$80,$A1,$C2,$E3,$04,$25,$46,$67
 BYT $B9,$98,$FB,$DA,$3D,$1C,$7F,$5E,$B1,$90,$F3,$D2,$35,$14,$77,$56
 BYT $EA,$CB,$A8,$89,$6E,$4F,$2C,$0D,$E2,$C3,$A0,$81,$66,$47,$24,$05
 BYT $DB,$FA,$99,$B8,$5F,$7E,$1D,$3C,$D3,$F2,$91,$B0,$57,$76,$15,$34
 BYT $4C,$6D,$0E,$2F,$C8,$E9,$8A,$AB,$44,$65,$06,$27,$C0,$E1,$82,$A3
 BYT $7D,$5C,$3F,$1E,$F9,$D8,$BB,$9A,$75,$54,$37,$16,$F1,$D0,$B3,$92
 BYT $2E,$0F,$6C,$4D,$AA,$8B,$E8,$C9,$26,$07,$64,$45,$A2,$83,$E0,$C1
 BYT $1F,$3E,$5D,$7C,$9B,$BA,$D9,$F8,$17,$36,$55,$74,$93,$B2,$D1,$F0

; hi byte CRC lookup table (should be page aligned)
CRCHI
 BYT $00,$10,$20,$30,$40,$50,$60,$70,$81,$91,$A1,$B1,$C1,$D1,$E1,$F1
 BYT $12,$02,$32,$22,$52,$42,$72,$62,$93,$83,$B3,$A3,$D3,$C3,$F3,$E3
 BYT $24,$34,$04,$14,$64,$74,$44,$54,$A5,$B5,$85,$95,$E5,$F5,$C5,$D5
 BYT $36,$26,$16,$06,$76,$66,$56,$46,$B7,$A7,$97,$87,$F7,$E7,$D7,$C7
 BYT $48,$58,$68,$78,$08,$18,$28,$38,$C9,$D9,$E9,$F9,$89,$99,$A9,$B9
 BYT $5A,$4A,$7A,$6A,$1A,$0A,$3A,$2A,$DB,$CB,$FB,$EB,$9B,$8B,$BB,$AB
 BYT $6C,$7C,$4C,$5C,$2C,$3C,$0C,$1C,$ED,$FD,$CD,$DD,$AD,$BD,$8D,$9D
 BYT $7E,$6E,$5E,$4E,$3E,$2E,$1E,$0E,$FF,$EF,$DF,$CF,$BF,$AF,$9F,$8F
 BYT $91,$81,$B1,$A1,$D1,$C1,$F1,$E1,$10,$00,$30,$20,$50,$40,$70,$60
 BYT $83,$93,$A3,$B3,$C3,$D3,$E3,$F3,$02,$12,$22,$32,$42,$52,$62,$72
 BYT $B5,$A5,$95,$85,$F5,$E5,$D5,$C5,$34,$24,$14,$04,$74,$64,$54,$44
 BYT $A7,$B7,$87,$97,$E7,$F7,$C7,$D7,$26,$36,$06,$16,$66,$76,$46,$56
 BYT $D9,$C9,$F9,$E9,$99,$89,$B9,$A9,$58,$48,$78,$68,$18,$08,$38,$28
 BYT $CB,$DB,$EB,$FB,$8B,$9B,$AB,$BB,$4A,$5A,$6A,$7A,$0A,$1A,$2A,$3A
 BYT $FD,$ED,$DD,$CD,$BD,$AD,$9D,$8D,$7C,$6C,$5C,$4C,$3C,$2C,$1C,$0C
 BYT $EF,$FF,$CF,$DF,$AF,$BF,$8F,$9F,$6E,$7E,$4E,$5E,$2E,$3E,$0E,$1E
;
;
; End of File
;
;=====================================================================
;
; Alternate solution is to build the two lookup tables at run-time.  This might
; be desirable if the program is running from ram to reduce binary upload time.
; The following code generates the data for the lookup tables.  You would need to
; call this routine to build the tables before calling the main program.
; This can also be used to generate the lookup tables to embody in the overall program.
;
;CRCTAB      LDX   #$00
;           LDA   #$00
;ZLOOP       STA   CRCLO,X
;           STA   CRCHI,x
;           INX
;           BNE   ZLOOP
;           LDX   #$00
;FETCH       TXA
;           EOR   CRCHI,x
;           STA   CRCHI,x
;           LDY   #$08
;FETCH1      ASL   CRCLO,x
;           ROL   CRCHI,x
;           BCC   FETCH2
;           LDA   CRCHI,x
;           EOR   #$10
;           STA   CRCHI,x
```

```
;               LDA     CRCLO,x
;               EOR     #$21
;               STA     CRCLO,x
;FETCH2         DEY
;               BNE     FETCH1
;               INX
;               BNE     FETCH
;               RTS
;
```