```
;RCA Microkit Firmware as found in Bill Degnan's unit.
;
;Rev 0: Microkit Firmware reverse engineered.  April 2015. Josh Bensadon, Toronto.
;               There are likely to be several errors in the comments.  A better understanding is possible with further
;               examination of the hardware.
;
;This unit uses the 1801 chip set, which is an early version of the 1802 where the CPU and Registers are in two
;separate chips working together.  It is assumed that the 1801 uses the same instruction set as the 1802.
;
;Bill Degnan was able to read the pair of 1702r EPROM chips.
;Two chips, marked "3" and "4" were read.  No reversed engineered schematics were available at this time.
;It is assumed that the chips occupy consecutive pages in the memory space.
;The code within the chips did not make any sense when looked at from the 1802 op codes, on a hunch, I figured
;that the codes may be inverted. Inverted data and address busses was not uncommon back then, the basic reason
;for this is the speed of logic gates and simplifying the LSI chips (which had very limited space on the die).
;Upon inverting the data, all the popular 1802 codes were clearly visible (F8 being one of the most popular).
;Codes shown in this file are inverted from the data given in the EPROM dumps.  Hence the first code shown
;00 is actually stored as FF on the EPROM.
;
;
;
;Firmware addresses indicate that these ROM's are mapped to pages 0x80xx and 0x81xx
;
;Firmware indicates that there is RAM at pages 0x82xx and at 0x00xx
;
;Firmware I/O indicates that RS-232 input is on EF4, and RS-232 output is on lsb of output port 7
;
;Firmware has 3 sections:
; 1. Save all Registers to RAM at 0x82xx
; 2. Auto detects the baud rate (I expect this to be either 110 or 300 baud)  May require stripping the 8th bit.
;    Auto baud also selects echo on/off, which characters to use will need to be explored further
; 3. Monitor program to display memory, edit memory and jump to execute code in memory (PC=R0)
;
;
;The character(s) used in autobaud and to select echo on/off needs to be explored further, having the exact
;clock frequency (unknown at this time) would be helpful. Microkit-autobaud.pdf will illustrate the autobaud.
;
;Another questionable aspect is the first instruction of 00, which is an IDLE instruction.  This instruction on
;the 1802 will halt the CPU until an interrupt or DMA occurs.  It is not certain that the 1801 will operate the
;same.  Since there is no schematic available, it's unclear if there's any device that can cause an interrupt.
;
;The code starts with R0 being the PC and immediately sets the A15 line high.  The schematic might reveal some
;circuitry to temporarily set A15 high until this code executes.  On a personal project, the VELF, I use a
;similar technique where the ROM is temporarily mapped to 0x0000 on reset, then this mapping gets undone on the
;first occurance of A15 being high (which occurs with a jump to 0x80xx)
;
;The rest of this file is the disassembled code, commented with purpose as best as possible.
;
```

```
          00       IDL                   ;Wait for interrupt
          F8 80    LDI        80
          B0       PHI        R0         ;Jumps PC to 8004,  Sets A15 high, perhaps it unlatches some kind of boot
strap bank switch like in my VELF?



                                         ;Save Registers F to 1 to RAM @ 821F - 8202
                                         ;This is done by creating and executing self modifying code that will cycle
through GLO and GHI instructions for 15 registers
                                         ;The self modifying code is a "Fetch Register" and "Return", upon Return, the
register is saved in the next RAM location
                                         ;
                                         ;     Iteration 0                 /  Iteration 1
                                         ;
                                         ;                                 821C  GHI RF  Fetch RF.1
                                         ;  821D  GLO RF  Fetch RF.0       821D  SEP R0  Return
                                         ;  821E  SEP R0  Return           821E  =RF.1   <-Save RF.1
                                         ;  821F  =RF.0   <-Save RF.0
                                         ;
                                         ;The last 2 iterations will look like this....
                                         ;
                                         ;                                 81FF  GLO R0  Fetch R0.0  <- This will fail the
verify test and exit the loop.
                                         ;  8200  GHI R1  Fetch R1.1       8200  SEP R0  Return
                                         ;  8201  SEP R0  Return           8201  SEP R0
                                         ;  8202  =R1.1   <-Save R1.1
                                         ;
                                         ;
                                         ;
          F8 82    LDI        82         ;R1=821E (RAM) = Location of self modifying code & place-1 to save Register
          B1       PHI        R1
          F8 1E    LDI        1E
          A1       PLO        R1
          F8 A0    LDI        A0         ;R4.1=A0 = initial opcode for self modifing code that sequences a SAVE
Registers routine
          B4       PHI        R4
          E1       SEX        R1         ;X=R1

SAVE_LP   F8 D0    LDI        D0         ;Create a "SEP R0" instruction to Return after call to R1
          51       STR        R1         ;M(RX) = D ;Put Instruction into RAM @821E (next itterations = 821D, 821C,
821B... 8200)
                                         ;Test if we are still in RAM space
          F3       XOR                   ;D=D XOR M(RX) = 00 if saving to RAM
          3A 29    BNZ        L29        ;Jump if D <> byte saved (Confirms the instruction D0 is written)
          21       DEC        R1


                                         ;Create a "GLO Rx" / "GHI Rx" instruction
          94       GHI        R4         ;Gets the A0, then adds a modified increment to acheive the following pattern:
          FC 70    ADI        70         ;input: A0  8F  9F  8E 9E 8D 9D 8C 9C 8B 9B 8A 9A 89 99 88 98 87 97 86 96
```

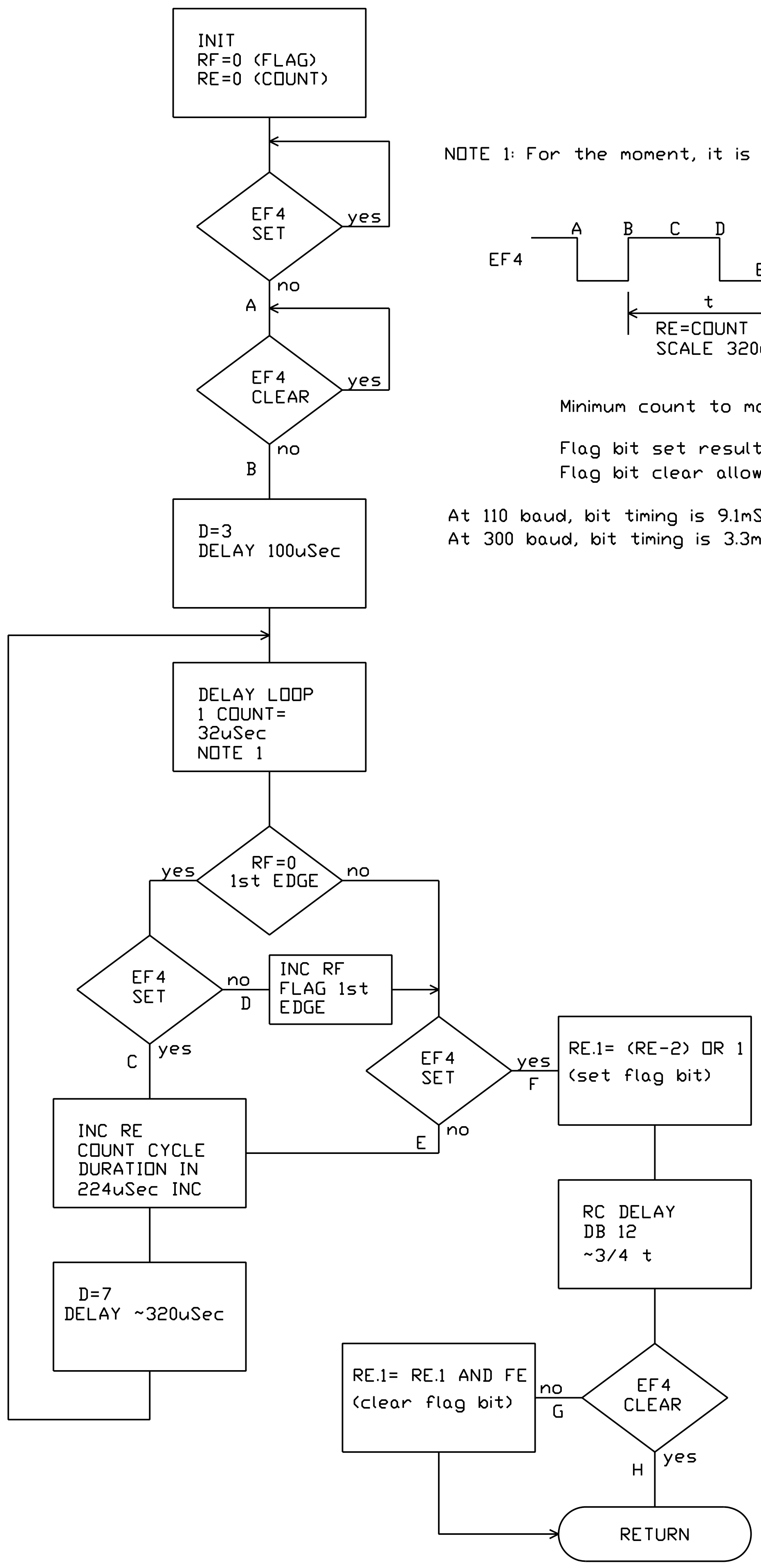
```
85 95 84 94 83 93 82 92 81
            33 1C       BDF         QNC         ;output:  8F  9F  8E 9E 8D 9D 8C 9C 8B 9B 8A 9A 89 99 88 98 87 97 86 96 85
95 84 94 83 93 82 92 81 91
            FC 21       ADI         21
QNC:        FC 7F       ADI         7F          ;
            B4          PHI         R4          ;Save instruction in R4.1 for next modification

            51          STR         R1          ;(821D)=8F  = GLO RF ;Put Insruction in RAM  @821D (next itterations =
821C, 821B, 821A... 81FF)
            F3          XOR                     ;D=D XOR M(RX) = 00 if saving to RAM (test for end of Register Saving)
            3A 29       BNZ         L29         ;Jump if R1=81FF = End of Register Saving (ROM)

            D1          SEP         R1          ;Call at 821D to "GLO RF"

            51          STR         R1          ;Save RF.0 at M(821F)
            21          DEC         R1          ;Decrement 2 bytes for next modified code call
            21          DEC         R1
            30 0E       BR          SAVE_LP


 L29:       90          GHI         R0          ;Setup Registers, Change PC to R5, X=5, R2=0000, PC to R3 (jump to ROM2)
  2A:       B5          PHI         R5
  2B:       B3          PHI         R3
  2C:       F8 30       LDI         PCR5
  2E:       A5          PLO         R5          ;R5 = PCR5
  2F:       D5          SEP         R5          ;P=5   PC now equals R5
PCR5:       E5          SEX         R5          ;X=5
  31:       71          DIS                     ;Disable interrupts
  32:       55          DB          55          ;X,P = 5,5
  33:       60          IRX         60          ;Skip next instruction???
  34:       01          LDN         R1
  35:       F8 00       LDI          0          ;R2 = 0000
  37:       B2          PHI         R2
  38:       A2          PLO         R2
  39:       F8 FE       LDI         AUTOBR  ;Auto Baud Rate detect (Initializes delay counter and Echo Flag)
  3B:       A3          PLO         R3
  3C:       D3          SEP         R3          ;CALL TO PCR3 at 80FE (ADVANCES TO ROM 2 @8100), Fetchs the baud
rate from EF4
                                                ;lsb of RE.1 will indicate Echo ON/OFF

MAIN_LOOP
 L3D:       F8 9C       LDI         TX_CHAR;Subroutine in ROM2, TX_CHAR
  3F:       A3          PLO         R3

                                                ;Send <CR> <LF>
  40:       D3          SEP         R3          ;TX_Char
                        DB          0x0D

  42:       D3          SEP         R3          ;TX_Char
                        DB          0x0A

  44:       D3          SEP         R3
                        DB          '*'
```

```
REDO_INPUT
 L46:   92      GHI     R2      ;RD = 0000 = 16 bit HEX INPUT accumulator (HEX chars are shifted in from the
right)
  47:   AD      PLO     RD      ;(Clear HEX Input)
  48:   BD      PHI     RD

  49:   F8 3B   LDI     RX_HEX  ;Subroutine in ROM2, RX_HEX (ASCII)
  4B:   A3      PLO     R3
  4C:   D3      SEP     R3

  4D:   FB 24   XRI     '$'     ;Test incoming char for $
  4F:   32 D6   BZ      DOLLAR  ;Jump if $

  51:   FB 05   XRI     '$'^'!'
  53:   A1      PLO     R1      ;Save Cmd
  54:   32 5A   BZ      L5A     ;Jump if !
  56:   FB 1E   XRI     '!'^'?'
  58:   3A 46   BNZ     REDO_INPUT      ;Jump if NOT ?

CASE1                           ;If CMD is ! or ? then...       ?M xxxx xxxx<CR>   Dumps Memory @xxxx (1st
hex) for length xxxx (2nd hex)
 L5A:   D3      SEP     R3      ;Fetch next HEX char
  5B:   FB 4D   XRI     'M'
  5D:   3A CB   BNZ     INVALID ;Jump if not M

INPUT_ADDRESS
 L5F:   D3      SEP     R3      ;Fetch next HEX char
  60:   3B 5F   BNF     $-1     ;Loop back while NOT a HEX char (ie wait any number of characters after the
'M' to accept HEX input)
 L62:   D3      SEP     R3      ;Fetch next char after HEX input gets terminated by non-HEX char
  63:   33 62   BDF     $-1     ;Loop back if while HEX input
  65:   FB 20   XRI     ' '     ;Test for <SPACE>
  67:   3A CB   BNZ     INVALID ;Jump if NOT <SPACE>
  69:   9D      GHI     RD      ;R0 = RD (the HEX Address Entered)
  6A:   B0      PHI     R0
  6B:   8D      GLO     RD
  6C:   A0      PLO     R0
  6D:   81      GLO     R1      ;Fetch Cmd
  6E:   32 B5   BZ      MEMWR   ;Jump if Cmd '!'
  70:   92      GHI     R2      ;RD = 0000 (Clear HEX Input)
  71:   AD      PLO     RD
  72:   BD      PHI     RD
 L73:   D3      SEP     R3      ;Fetch HEX Input
  74:   33 73   BDF     $-1     ;Loop back while HEX Input
  76:   FB 0D   XRI     0Dh
  78:   3A CB   BNZ     INVALID ;Jump if input <> <CR>

  7A:   F8 9C   LDI     TX_CHAR;Output <LF>
  7C:   A3      PLO     R3

DUMP_NEW_LINE
 L7D:   D3      SEP     R3
```

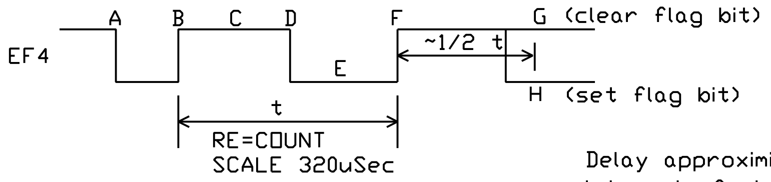
```
                DB      0Ah     ;<LF>

 7F:    90      GHI     R0      ;Output HEXBYTE number in R0.1
 80:    BF      PHI     RF
 81:    F8 AE   LDI     TX_BYTE
 83:    A3      PLO     R3
 84:    D3      SEP     R3

 85:    80      GLO     R0      ;Output HEXBYTE number in R0.0
 86:    BF      PHI     RF
 87:    F8 AE   LDI     TX_BYTE
 89:    A3      PLO     R3
 8A:    D3      SEP     R3

 8B:    D3      SEP     R3      ;Output <SPACE>
                DB      ' '

DUMP_NEXT_BYTE
 L8D:   40      LDA     R0      ;Fetch Byte at R0 (and advance R0)
 8E:    BF      PHI     RF
 8F:    F8 AE   LDI     TX_BYTE ;Output HEXBYTE
 91:    A3      PLO     R3
 92:    D3      SEP     R3

 93:    2D      DEC     RD      ;RD = RD - 1
 94:    8D      GLO     RD
 95:    3A 9A   BNZ     9A      ;Jump if RD.0 <> 0
 97:    9D      GHI     RD
 98:    32 3D   BZ      MAIN_LOOP       ;Jump back to MAIN_LOOP if RD = 0

 9A:    80      GLO     R0
 9B:    FA 0F   ANI      F
 9D:    3A A5   BNZ     A5

                                ;Every 16 bytes, Output a new line
 9F:    D3      SEP     R3      ;Output ";<CR>"
                DB      ';'
 A1:    D3      SEP     R3
                DB      0Dh

 A3:    30 7D   BR      DUMP_NEW_LINE   ;Loop back to dump more memory

 A5:    F6      SHR             ;Test for even/odd bytes
 A6:    33 8D   BDF     DUMP_NEXT_BYTE;Jump if odd

 A8:    D3      SEP     R3              'Output a <SPACE> every 2 bytes.
                DB      ' '
 AA:    30 8D   BR      DUMP_NEXT_BYTE

Continue_input
 LAC:   D3      SEP     R3      ;Get HEX
 AD:    3B AC   BNF     LAC     ;Jump Back if NOT HEX (ie ignore all non-hex chars until next HEX input = high
nibble)
```

```
 LAF:     D3       SEP     R3        ;Get HEX (low nibble)
  B0:     3B CB    BNF     INVALID   ;Jump if NOT HEX
  B2:     8D       GLO     RD        ;Fetch the Byte Entered
  B3:     50       STR     R0        ;Save it @R0
  B4:     10       INC     R0        ;Advance to next byte
                                     ;Repeat input, until <CR> or invalid HEX input....

MEMWR                                ;If CMD is ! then...       !M xxxx dd   (xxxx is already stored in R0)
 LB5:     D3       SEP     R3        ;Get HEX
  B6:     33 AF    BDF     LAF       ;Jump if HEX Char
  B8:     FB  D    XRI     D         ;Else, Test if ASCII <CR>
  BA:     32 3D    BZ      MAIN_LOOP

  BC:     FB 21    XRI     0Dh^','   ;Is input a comma?
  BE:     32 AC    BZ      LAC       ;Jump to continue input

  C0:     FB 17    XRI     ','^';'   ;Is input a semicolon?
  C2:     3A B5    BNZ     B5        ;Jump if not Semicolon...
  C4:     D3       SEP     R3        ;Get HEX
  C5:     FB  D    XRI     D         ;Test if ASCII <CR>
  C7:     3A C4    BNZ     C4        ;Loop back until <CR> is entered
  C9:     30 5F    BR      INPUT_ADDRESS  ;Go back to fetch next address


INVALID
 LCB:     F8 9C    LDI     TX_CHAR
  CD:     A3       PLO     R3        ;OUTPUT <CR><LF>?
  CE:     D3       SEP     R3
                   DB      0Dh
  D0:     D3       SEP     R3
                   DB      0Ah
  D2:     D3       SEP     R3
                   DB      '?'
  D4:     30 3D    BR      MAIN_LOOP

DOLLAR                               ;DOLLAR COMMAND  $P xxxx<CR>  Execute at xxxx
 LD6:     D3       SEP     R3        ;Get Hex
  D7:     FB 50    XRI     'P'       ;
  D9:     3A CB    BNZ     INVALID
 LDB:     D3       SEP     R3        ;Get Hex
  DC:     33 DB    BDF     LDB       ;Loop back while HEX
  DE:     FB 0D    XRI     D         ;Test for <CR>
  E0:     3A CB    BNZ     INVALID
  E2:     9D       GHI     RD        ;R0 = RD (Input)
  E3:     B0       PHI     R0
  E4:     8D       GLO     RD
  E5:     A0       PLO     R0

  E6:     F8 9C    LDI     TX_CHAR;Output <LF>
  E8:     A3       PLO     R3
  E9:     D3       SEP     R3
                   DB      0Ah
  EB:      E5       SEX      R5       ;X = R5 (=PC)
  EC:      70       RET
                   DB       0        ;X=0, P=0 Jump to address at R0

S1RET     D3       SEP      R3
SUB1      9E       GHI      RE       ;Called routine from PCR3.  DELAY RE.1 less (shift) a bit
  F0:      F6       SHR               ;Remove lsb from delay count
  F1:      AE       PLO      RE       ;Save RE.0 for outter loop
S1LOOP    2E       DEC      RE
  F3:      43       LDA      R3       ;Fetch Delay Value (parameter following call to SUB1)
  F4:      FF 01    SMI      1        ;4 cycle delay
  F6:      3A F4    BNZ      $-1
  F8:      8E       GLO      RE       ;Repeat until RE.0 = 0
  F9:      32 EE    BZ       S1RET
  FB:      23       DEC      R3       ;Point RETURN back to parameter
  FC:      30 F2    BR       S1LOOP   ;and repeat delay


                                     ;        Branch here to get into ROM2 (PC=R3, X=R5).  These are the last 2
instructions of ROM1 & execution continues into ROM2
AUTOBR    93       GHI      R3       ;D=R3.1=80
          BC       PHI      RC       ;RC.1=80

                                     ;================================ ROM 2
=========================================================
 100:     F8 00    LDI       0
 102:     AE       PLO      RE       ;RE.0=0
 103:     AF       PLO      RF       ;RF.0=0
 104:     F8 EF    LDI      SUB1     ;Address to SUB1
 106:     AC       PLO      RC

                                     ;        HOLD HERE UNTIL EF4 INPUT IS TOGGLED CLEAR/SET ONCE
                                     ;        WAIT UNTIL START BIT.
                                     ;
          37 07    B4       $        ;Loop while EF4 SET
          3F 09    BN4      $        ;Loop while EF4 CLEAR

 10B:     F8 03    LDI       3       ;D=3   (DELAY 3 itterations)  100uSec
TPLOOP    FF 01    SMI      1        ;D=D-1
 10F:     3A 0D    MBNZ     $-1      ;LOOP BACK UNTIL D=0      Delay 4 cycles or 32 Clocks per itteration.  =
32uSec @ 1Mhz
 111:     8F       GLO      RF       ;RF is 0 until EF4 returns to a clear state
 112:     3A 17    BNZ      L117
 114:     37 19    B4       L119
 116:     1F       INC      RF       ;RF flag set when EF4 is CLEAR
L117:     37 1E    B4       TPEXIT   ;Exit when EF4 SET's again
L119:     1E       INC      RE       ;RE COUNTS HOW LONG EF4 IS SET & CLEAR in 224uSec increments
 11A:     F8 07    LDI       7       ;Delay 224uSec
 11C:     30 0D    BR       TPLOOP

TPEXIT    2E       DEC      RE       ;RE=RE-2     Now delay for time that is less than the cycle just timed.
 11F:     2E       DEC      RE       ; Say the cycle just timed was 2mSec, RE=8. The delay to follow is
3*384=1.1mSec
```

```
 120:     8E       GLO     RE
 121:     F9 01    ORI      1       ;Assume a "1" bit?
 123:     BE       PHI     RE       ;RE.1 = RE.0 OR 1
 124:     DC       SEP     RC       ;CALL SUB1  (big delay)
 125:     0C       DB      0Ch      ;Delay value for loop in SUB1, 12*32 = 384uSec

 126:     3F 2C    BN4     L12C     ;If Clear jump
 128:     9E       GHI     RE       ;If EF4 still set after delay, clear the bit to "0"
 129:     FA FE    ANI     FE
 12B:     BE       PHI     RE
L12C:     DC       SEP     RC       ;Repeat Delay, but at count * 38*32 = Count * 1,216uSec
 12D:     26       DB      26h
 12E:     D5       SEP     R5       ;Return to calling routine


L12F:     FC 07    ADI      7
 131:     33 37    BDF     L137     ;Jump if input char is less than 'A' but greater than '9' (ie invalid HEX)
 133:     FC 0A    ADI      A
 135:     33 77    BDF     L177     ;Jump if input char is between '9' and '0' (ie valid HEX)....
                                    ;Else...
                                    ;EXIT - INVALID HEX INPUT
L137:     FC 00    ADI      0       ;Clear DF
L139:     9F       GHI     RF       ;Return with the ASCII CHAR

                                    ;Recieve a serial character?
 13A:     D5       SEP     R5
RX_HEX             equ     3Bh
 13B:     F8 00    LDI      0       ;RF.0 = 0 (HEX input, DF is set if HEX input)
 13D:     38       SKP
RX_CHAR
 13E:     93       GHI     R3       ;RF.0 = 81h (ASCII input only)
 13F:     AF       PLO     RF       ;
L140:     F8 80    LDI     80       ;RF.1 = 80h  (Preload the RX Data register with a flag bit for testing)
 142:     BF       PHI     RF
 143:     E2       SEX     R2
L144:     3F 44    BN4     L144     ;Wait for start bit
L146:     37 46    B4      L146
 148:     DC       SEP     RC       ;Delay, at count * 2*32 = Count * 64uSec (Delay 1/4 bit time?)
                   DB      2

L14A:     F8 00    LDI      0       ;M(RX) = 00
 14C:     52       STR     R2
L14D:     9E       GHI     RE
 14E:     FA 01    ANI      1       ;Fetch flag BIT
 150:     F1       OR               ;OR it with RX Byte
 151:     52       STR     R2       ;Save M(RX)

 152:     67       OUT     PIN7     ;Echo the bit (if flag=0)
 153:     22       DEC     R2       ;

                                    ;136uSec overhead time
 154:     DC       SEP     RC       ;Delay, count * 7*32 = Count * 224uSec (1 BIT Time)
                   DB      7
 156:     F8 01    LDI      1       ;M(RX)=1 (assume a 1 bit for next echo)
 158:     52       STR      R2

 159:     9F       GHI      RF      ;RF.1 = RF.1 >> 1
 15A:     F6       SHR
 15B:     BF       PHI      RF

 15C:     33 65    BDF      L165    ;JUMP DF=1  (Exit after 7 bits shifted in)
 15E:     F9 80    ORI      80      ;D = RF.1 OR 80 (assume a 1 bit)
 160:     3F 4A    BN4      L14A    ;Sample RX input, Jump if zero (and don't update RF.1 with 1 bit)
 162:     BF       PHI      RF      ;Save the 1 bit
 163:     30 4D    BR       L14D    ;Jump back to echo the 1 bit.

L165:     67       OUT      PIN7    ;Send Stop Bit (might occur at 8th bit time, TTY must strip high bit)
 166:     22       DEC      R2
 167:     32 40    BZ       L140    ;If incoming byte is <NULL> then branch back to get another char (ie
ignore/reject NULL's)

 169:     8F       GLO      RF      ;Test type of input desired
 16A:     3A 39    BNZ      L139    ;Branch to EXIT if ASCII input desired

                                    ;Process HEX input
 16C:     9F       GHI      RF      ;Fetch incoming ASCII char
 16D:     FF 41    SMI      'A'     ; D = D - 'A',  DF=1 if D>='A'
 16F:     3B 2F    BNF      L12F    ;Jump if < 'A' and test if input is correct digits 0 to 9
 171:     FF 06    SMI      6       ; Lets see if char was > 'F'
 173:     33 37    BDF      L137    ;Jump if char >'F' to EXIT-INVALID-HEX-INPUT
 175:     FC 10    ADI      10      ;Add correct offset for 'A', so D=10 if char='A'

                                    ;If input is a valid HEX character (0-9 or A-F), save the converted nibble here...
L177:     AE       PLO      RE      ;Save the HEX value of the input

                                    ;Shift the new HEX input into the 16 bit RD register.
 178:     9D       GHI      RD      ;Fetch RD.1
 179:     52       STR      R2      ;Shift RD.1 << 4  (not sure why they just didn't do a SHL???)
 17A:     F4       ADD
 17B:     52       STR      R2
 17C:     F4       ADD
 17D:     52       STR      R2
 17E:     F4       ADD
 17F:     52       STR      R2
 180:     F4       ADD
 181:     52       STR      R2
 182:     8D       GLO      RD      ;Fetch RD.0
 183:     F6       SHR              ; >> 4 to put high nibble to low position
 184:     F6       SHR
 185:     F6       SHR
 186:     F6       SHR
 187:     F1       OR               ;OR it with RD.1 << 4
 188:     BD       PHI      RD      ;Update RD.1
 189:     8D       GLO      RD
 18A:     52       STR      R2
 18B:     F4       ADD
 18C:     52       STR      R2
 18D:     F4       ADD
 18E:     52       STR      R2
 18F:     F4       ADD
 190:     52       STR      R2
 191:     F4       ADD
 192:     52       STR      R2
 193:     8E       GLO      RE
 194:     FA 0F    ANI       F
 196:     F1       OR
 197:     AD       PLO      RD
 198:     FF 00    SMI      0        ;SET DF Flag
 19A:     30 39    BR       L139     ;Exit with D= the last 2 HEX chars (RD.0)


TX_CHAR            equ      9Ch      ;Send next inline byte
 19C:     DC       SEP      RC       ;Repeat Delay, but at count * 23*32 = Count * 736uSec (Character Delay)
                   DB       17h
 19E:     38       SKP               ;Skip the Return on entering
L19F:     D5       SEP      R5       ;Return with a "Fetch Next Char"
 1A0:     45       LDA      R5       ;Fetch Paramater
 1A1:     38       SKP               ;Skip the alternate Parameter Fetch?
 1A2:     46       LDA      R6
 1A3:     38       SKP               ;Skip the 3rd alternate parameter
 1A4:     9F       GHI      RF
 1A5:     AE       PLO      RE       ;Save in RE.0
 1A6:     FB 0A    XRI      A        ;Test if char = <LF>  (0x0A)
 1A8:     3A C0    BNZ      L1C0     ;Jump to Send Character with a "Fetch Next Char"

 1AA:     F8 8B    LDI      8B       ;Flag=8, Bit Counter = 11
 1AC:     30 C2    BR       L1C2     ;Jump to Send Character <LF> with 5 NULL's added to the output

TX_BYTE            equ      0AEh
 1AE:     9F       GHI      RF       ;Send BYTE in RF.1
 1AF:     F6       SHR               ;Convert High Nibble of RF.1 to ASCII (0 to F)
 1B0:     F6       SHR
 1B1:     F6       SHR
 1B2:     F6       SHR
 1B3:     FC F6    ADI      F6
 1B5:     3B B9    BNF      L1B9
 1B7:     FC 07    ADI       7
L1B9:     FF C6    SMI      C6
 1BB:     AE       PLO      RE
 1BC:     F8 1B    LDI      1B       ;Flag=1, Bit Counter =11
 1BE:     30 C2    BR       L1C2     ;Jump to Send with flag to send Low Nibble of RF.1

L1C0:     F8 0B    LDI       B
L1C2:     AF       PLO      RF       ;RF=Bit Counter = 11 (lower nibble).  Upper Nibble = Flag (0, 1 or 8).  0B, 1B,
8B
 1C3:     E2       SEX      R2       ;R2 = 0000 = RAM?
L1C4:     F8 00    LDI       0       ;D=0
 1C6:     52       STR      R2       ;M(R2)=0
 1C7:    67      OUT     PIN7     ;OUTPUT PORT7 = 0 (Start Bit)
 1C8:    22      DEC     R2       ;Undo the RX Increment by the OUT command
 1C9:    8E      GLO     RE       ;Fetch Character to send

TXLOOP
L1CA:    52      STR     R2       ;Save in M(R2)
                                  ;288uSec over head in loop
 1CB:    DC      SEP     RC       ;Delay, count * 7*32 = Count * 236uSec  (1 BIT TIME)
                 DB      7
 1CD:    2F      DEC     RF       ;Dec Bit Counter
 1CE:    F0      LDX              ;Fetch Character
 1CF:    AE      PLO     RE       ;Save in RE.0
 1D0:    FA 01   ANI      1       ;Mask the bit to be sent

 1D2:    52      STR     R2       ;Output bit to Port7
 1D3:    67      OUT     PIN7
 1D4:    22      DEC     R2

 1D5:    8F      GLO     RF       ;Test Bit Counter
 1D6:    FA 0F   ANI      F       ;Just the bit counter (lower nibble)
 1D8:    32 E2   BZ      TXEXIT   ;Jump if Zero

 1DA:    8E      GLO     RE       ;Fetch RE.0
 1DB:    8E      GLO     RE       ;twice???
 1DC:    F6      SHR              ;Shift the bits down
 1DD:    F9 80   ORI     80       ;Set the 9th+ bit to 1 (ie, Stop bits)
 1DF:    52      STR     R2       ;Save the character at M(RX)
 1E0:    30 CA   BR      TXLOOP   ;Jump to send next bit

TXEXIT
L1E2:    8F      GLO     RF       ;Test Flag
 1E3:    FC FB   ADI     FB       ;Flag=Flag-1, Counter=11
 1E5:    AF      PLO     RF       ;Resave Flag/Counter
 1E6:    3B 9F   BNF     L19F     ;Jump if RF = 0x0B (Fetch Next Char to send)
 1E8:    FF 1B   SMI     1B
 1EA:    32 9F   BZ      L19F     ;Jump if RF = 0x2B (Fetch Next Char to send)
 1EC:    3B F2   BNF     L1F2     ;Jump if RF = 0x1B (Send Low Nibble of RF.1)
                                  ;Else RF > 0x2B (=0x8B) (Sent the <LF>)

                                  ;While Flag > 2, Send NULL's  (ie, send 5 NULL's, then fetch next char)
 1EE:    F8 00   LDI      0       ;Send a <NULL> character after <LF>
 1F0:    30 FD   BR      L1FD

L1F2:    9F      GHI     RF       ;Fetch HEX Char in RF.1 (lower nibble)
 1F3:    FA 0F   ANI      F       ;Convert HEX Char to ASCII
 1F5:    FC F6   ADI     F6
 1F7:    3B FB   BNF     L1FB
 1F9:    FC 07   ADI      7
L1FB:    FF C6   SMI     C6

L1FD:    AE      PLO     RE       ;Save Character to send
 1FE:    30 C4   BR      L1C4     ;Repeat Send Loop
```



NOTE 1: For the moment, it is assumed the clock is 1Mhz.

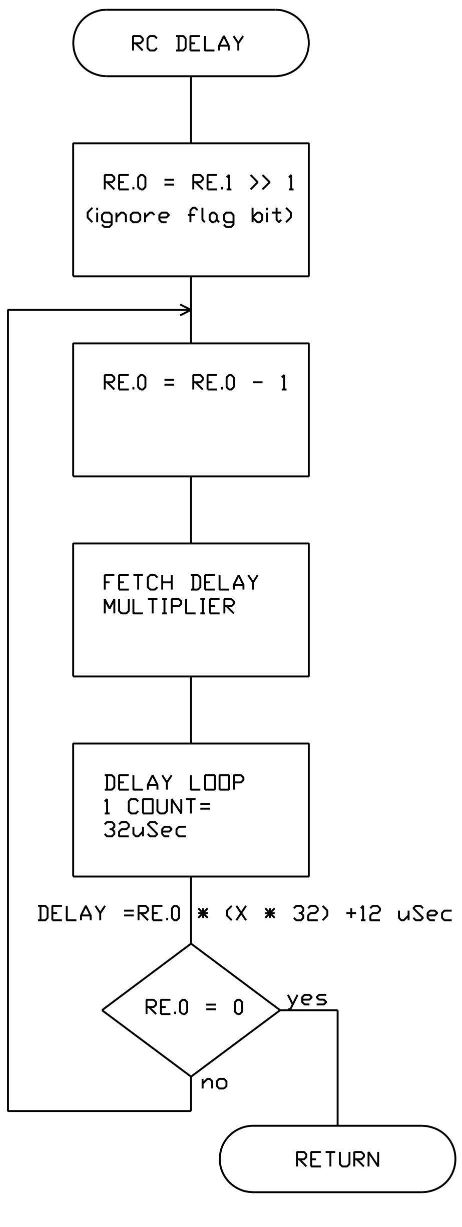


Delay approximiately an
interval of about 1/2 t
then sample input

Minimum count to make sense is 16

Flag bit set results in NO ECHO
Flag bit clear allows echo

At 110 baud, bit timing is 9.1mSec, RE=56? =74
At 300 baud, bit timing is 3.3mSec, RE=20? =26


RC DELAY
RE.0 = RE.1 >> 1
(ignore flag bit)
RE.0 = RE.0 - 1
FETCH DELAY
MULTIPLIER
DELAY LOOP
1 COUNT=
32uSec
DELAY =RE.0 * (X * 32) +12 uSec
RE.0 = 0
yes
no
RETURN