AP-468

APPLICATION
BRIEF

# Serial Port Mode 0
# 8X9XBH/KB/KC/KD

**RICHARD N. EVANS**
APPLICATIONS ENGINEER

February 1995

Order Number: 272245-001

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

# Serial Port Mode 0
# 8X9XBH/KB/KC/KD

## CONTENTS



## CONTENTS

## ABSTRACT

This application brief explains how to program the MCS®-96 device to operate the serial port in synchronous mode. A 4-bit multiplier which utilizes mode 0 with a port expansion circuit is presented.

## METHOD OF OPERATION

The serial port can be operated in a synchronous mode. This mode was intended for port expansion using shift registers. For example, the TXD pin is used to clock both input and output data on RXD. The data is always one byte in length. Whenever a write to the serial port buffer (SBUF) is performed, a train of eight pulses is sent out TXD to clock the outgoing byte. Likewise, whenever SBUF is read, a train of eight pulses is sent out TXD to clock in the byte being read. See the synchronous serial mode timing diagram shown in Figure 1.

### Timing Considerations

All timings associated with the serial port are relative to $T_{OSC}$. Therefore, the timings are fixed whether XTAL1 or T2CLK clocks the baud rate generator.

**$T_{DVXH}$** Input Data Setup to Clock (TXD) Rising Edge. In other words, the data has to be valid at $T_{DVXH}$ before the next TXD pulse rises.

**$T_{QVXH}$** Output Data Setup to Clock (TXD) Rising Edge. The output bit will be valid before the rising edge of the next TXD pulse for $T_{QVXH}$.

**$T_{XLXH}$** Serial Port Clock Falling Edge to Rising Edge. The low period for the TXD clock cannot be changed. For the 8X9X, $T_{XLXH} = 4\ T_{OSC} \pm 50$. For KB, KC and KD, $T_{XLXH} = 4\ T_{OSC} \pm 50$ or $2\ T_{OSC} \pm 50$ depending on the baud rate register value.

**NOTE:**
See the A.C. Characteristics in the datasheets for the timing specifications.

### Baud Rate

Baud rate is a misused term. Baud rate is often used interchangeably with bits per second (BPS). This substitution is not always true though. Baud rate is the speed at which packets of information are passed per second. It just so happens that with the MCS-96 family the length of the information packet is 1 bit. Hence, the baud rate measurement is the same as the bits per second (BPS) for the MCS-96 serial port. The MCS-96 has a 15-bit baud rate generator. The most significant bit (bit 15) determines the clock source (XTAL1 or T2CLK). There is a baud rate register (location 0EH). This register is a byte wide. When loading the baud rate register it must be written twice: first, the least significant byte must be written to location 0EH, then the most significant byte. See equation 1 for the baud rate register formula.

$$\text{BAUD\_\_VALUE} = \frac{F_{OSC}}{\text{BAUD RATE} \times 2} - 1$$

OR

$$= \frac{\text{T2CLK}}{\text{BAUD RATE}}$$

**Equation 1: Serial Port Synchronous Mode 0 Baud Rate Register Equations**

Setting up the baud rate generator is easy. The example code (code 1) shows how to configure the baud rate generator to run at 9600 baud with a 16 MHz crystal.

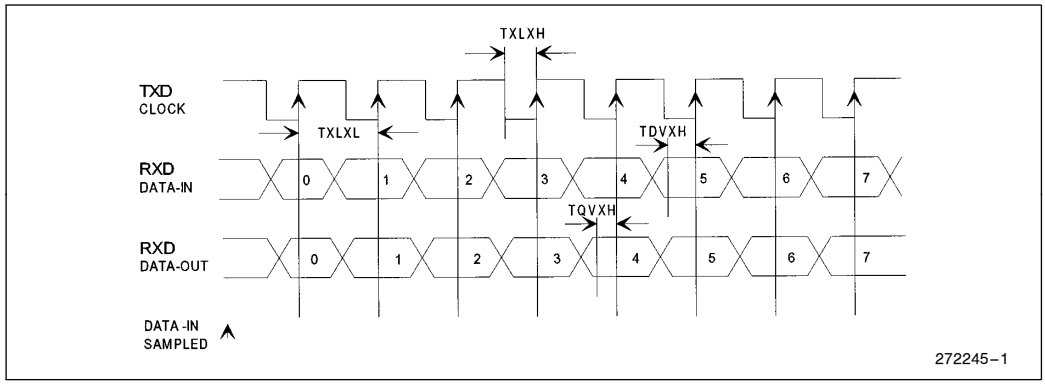



**Figure 1. Important Timings for Serial Port Mode 0**

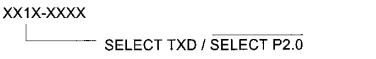
```
BAUD_VALUE  EQU   0EH            ; BAUD RATE REGISTER

CSEG AT 2080H
LDB BAUD_VALUE,40H               ; SET UP BAUD RATE GENERATOR FOR
LDB BAUD_VALUE,83H               ; FOR 9600 BAUD AT A 16MHz CRYSTAL FREQUENCY
```

**Code 1: Setting up Baud Rate Generator in Mode 0**

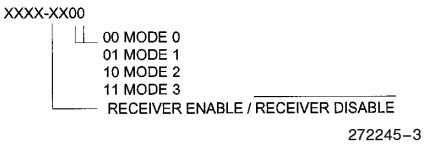
## SETTING UP THE CONTROL REGISTERS

### (16H) IOC1



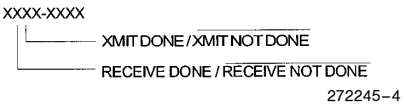

There are a few control registers that need to be utilized for mode 0 operation. First, since TXD is shared on the same pin as P2.0, we need to select the TXD function of that pin. This is accomplished by setting bit 5 in IOC1 (16H).

### (11H) SP__CON




In order to set the serial port to operate in mode 0, the serial port control register (SP__CON 11H) needs to be initialized. Bits 0 to 1 set the mode. Hence, setting them to zero enables mode 0. Also, in the SP__CON is the receiver enable bit. Setting this bit (bit 3) enables the receiver (see RECEIVE).

### (11H) SP__STAT




The serial port status register (SP__STAT 11H) is located at the same address as SP__CON. Writing to address 11H loads the serial port control register. Reading from 11H will read from the serial port status register (SP__STAT). Two status bits of importance are RI and TI. When set they indicate a receive completion or a transmit completion respectively. The RI and TI bits are cleared by reading SP__STAT.

### (09H) INT__MASK


(200CH) SERIAL PORT interrupt vector location.

**NOTE:**
This interrupt is available on the 8X9X, KB, KC and KD.

There are two ways to monitor the status of the receiver and/or the transmitter. One is by polling the SP__STAT register (specifically RI and TI), the other is by using interrupts. RI is set whenever the receiver is done receiving one byte in mode 0. Likewise, TI is set whenever the transmitter has sent out one byte in mode 0. If the SERIAL PORT interrupt is masked in, then a rising edge on RI or TI causes the SERIAL PORT interrupt to be taken. The SERIAL PORT interrupt bit in INT__MASK (09H) is the inclusive OR of RI and TI. Hence, either RI or TI can cause a SERIAL PORT interrupt. Therefore, once the interrupt routine is entered, SP__STAT has to be tested to determine which interrupt (RI or TI) occurred.

### (12H) INT__MASK1


(2030H) TI vector location
(2032H) RI vector location

**NOTE:**
These interrupts are available on the KB, KC, and KD—**not** on the 8X9X.

Additional interrupt vectors exist on the KB, KC and KD which make it easier to write code for the serial port. To interrupt on just the receive completion, the RI interrupt vector can be masked in. Similarly, the TI interrupt has a separate vector for transmit completion.

## RECEIVE

Reading the SP_STAT register always clears the RI bit and TI bit. If the RI bit is cleared while the RECEIVER ENABLE bit (bit 3 in SP_CON) is high, then another reception is started. Hence, it is possible to start another reception and overwrite the previous one. Therefore, don't poll SP_STAT to monitor the receiver. Use the serial port interrupt, the receive interrupt vector, or INT_PEND1 (KB, KC and KD) to test the RI bit for receive completion.

It is a good programming practice to use the serial port interrupt or the RI interrupt for testing the RI bit. First, load the interrupt vector location with the appropriate ISR routine address. Next, enable the interrupt using either INT_MASK or INT_MASK1 depending on which interrupt is chosen. Now, enable interrupts using the EI instruction. Then, disable the receiver by clearing bit 3 in SP_CON. Now the receiver is in a known state. To start a reception initiate a rising edge on the receiver enable bit (set bit 3 in SP_CON). When the service routine is entered, disable interrupts (i.e., PUSHF or PUSHA) and read SBUF (07H) to obtain the received byte. To start another reception, clear the RI bit by reading SP_STAT. Then, enable interrupts (i.e., POPF or POPA), and return from the interrupt service routine. Clearing the RI bit while the receiver is enabled starts a reception and allows another serial port or receive interrupt to occur. To disable the receiver simply clear the RECEIVER ENABLE bit in SP_CON. See the programming example in the following pages.

## TRANSMIT

Transmitting a byte is much more straightforward. First, load SBUF (07H) with the byte to be transmitted. Two methods can be used to detect when transmit completion occurs: polling TI in SP_STAT, or using the serial port interrupt or TI interrupt (KB, KC and KD). Once again, using an interrupt to detect transmit complete is good programming practice.

To set up the transmit interrupt service routine, load the address of the ISR into either the serial port interrupt vector (200CH) or the TI interrupt vector (2030H). As with receive, mask in the appropriate interrupt using either INT_MASK or INT_MASK1. Enable interrupts with EI and load SBUF with the byte to transmit.

When the interrupt service routine is entered disable interrupts (i.e., PUSHF or PUSHA). After the routine is executed another transmit can be started by loading SBUF again. Clear the TI bit in SP_STAT by reading SP_STAT. This action allows another transmit or serial interrupt to occur. Enable interrupts before returning from the service routine (i.e., POPF or POPA). When the next transmit is done, another interrupt occurs (serial or transmit).

## EXAMPLE USING MODE 0

A programming example is included to demonstrate most of the above procedures for implementing mode 0. An evaluation board was used in conjunction with an I/O port expansion circuit to test out the following code. The program reads in one byte from an external shift register. Then it multiplies the lower nibble by the upper nibble. The product is transmitted to another external shift register and is displayed on LEDs. The largest product is 0E1H which is 0FH x 0FH.

## HARDWARE

The schematic for this example is pictured in Figure 2. The data-in byte is generated by a DIP switch attached to a parallel-in serial-out shift register (74LS165). The output display is simply eight LEDs. The clock (TXD) is used to clock the parallel-in serial-out shift register (74LS165). This (74LS165) register has two modes: a shift mode and a load mode. When the transmit part of the circuit is activated, the 74LS165 is put into LOAD mode so the transmit shift register is not interfered with. To enable the transmit circuit, the TXD clock is gated to the 74LS164 (serial-in parallel-out). The transmit circuit is enabled by the active low signal $\overline{\text{ENABLE}}$. The RXD line is used for receiving and transmitting.

### Inverter

The inverter (74LS05) has an open-collector output. A weak (15K) pull-up is used at the output. The purpose of the weak pull-up is so the RXD (when used as an output) can drive the data on RXD high or low. If a regular inverter were used, then contention would exist between RXD (when used as an output) and the inverter output. Notice that the input to the inverter is $\overline{\text{QH}}$, hence the output of the inverter is the actual data QH.

### OR Gate

The OR gate is a switch for the TXD clock. TXD is at one input. The other input is the $\overline{\text{ENABLE}}$ line (from P2.6). When the $\overline{\text{ENABLE}}$ line is low, TXD passes freely through the OR gate. However, when $\overline{\text{ENABLE}}$ is high, the output of the OR gate is always high. As a result there are no transistions at the output of the OR gate and the 74LS164 is not clocked.

### DIP Switch

The DIP switch is weakly pulled high (switch off) and strongly driven low (switch on).

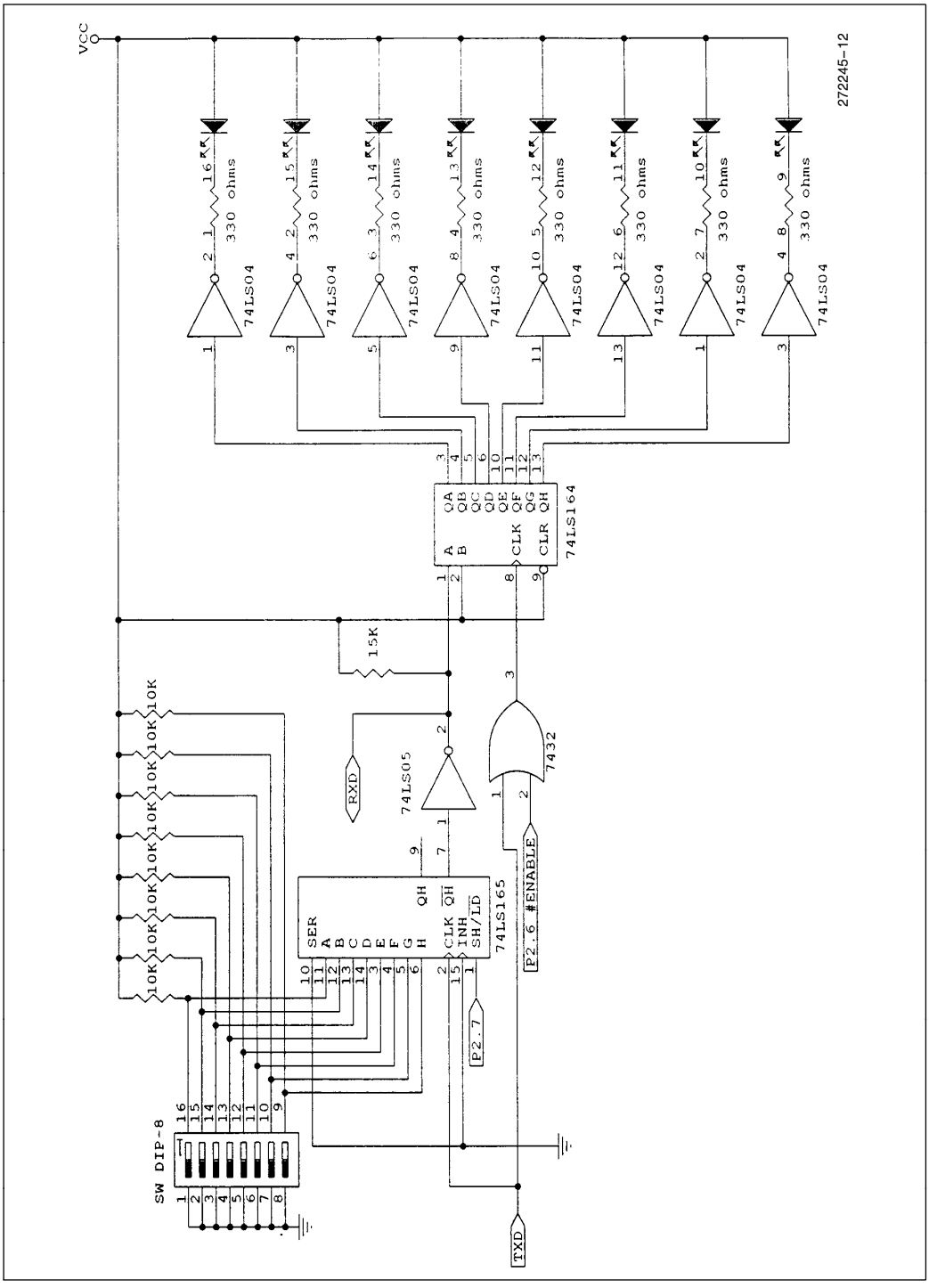



**Figure 2. I/O Port Expansion and Example Schematic**

## SOFTWARE

See the program listing for the software part of this example. Only the serial port interrupt is used in this example. Hence, this program is compatible with 8X9XBH, KB, KC and KD. The flow chart in Figure 3 illustrates the algorithm.

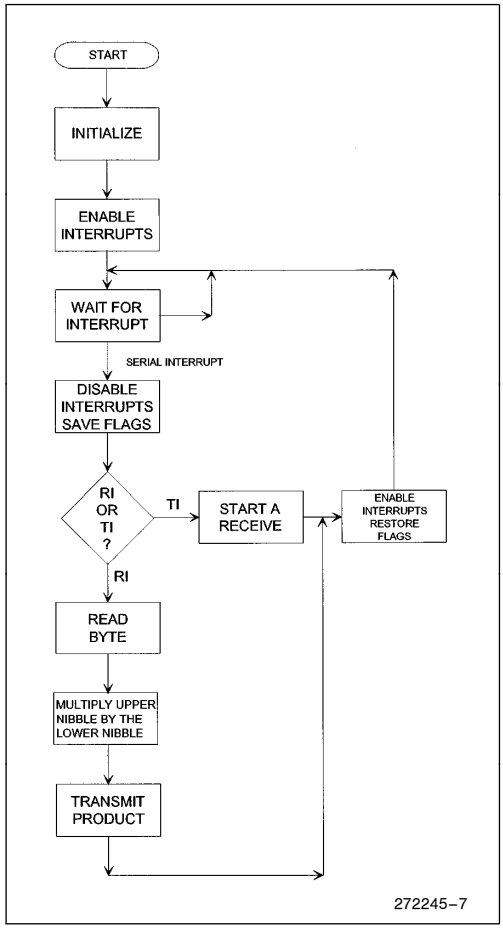
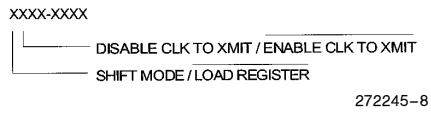



**Figure 3. Flowchart of Example Mode 0 Program**

During the initialize procedure the control registers are set to a known state. The serial port interrupt is masked in. Then the TXD function is enabled on its respective pin. Next, the baud rate is set. The baud rate generator can be clocked by either XTAL1 or T2CLK.

Now, the first receive is started. The shift register must be loaded with the DIP switch byte. Hence, a procedure is called to load the shift register and to set the register to shift mode. Port 2 is used to output control signals to the shift register and the transmit enable gate. Clearing P2.7 loads the shift register. Setting P2.7 puts the register in shift mode. Furthermore, to allow TXD to clock the transmit shift register P2.6 must be cleared. Setting P2.6 disables the TXD clock to the transmit shift register. Port 2 is diagrammed below:

### (10H) PORT 2




The next step in starting a receive in this example is to disable the clock to the transmit circuit (see above). P2.6 is set by performing a logical OR. Next, mode 0 is selected and the receiver is disabled by clearing all bits in SP__CON. Now, a rising edge on the REN bit is initiated by setting bit 3 in SP__CON. Finally, SP__STAT is cleared by reading it. Note that a special procedure was used to clear SP__STAT. This routine only needs to be called for the 8X9XBH (see techbit MC3391). The KB, KC and KD can simply do a LDB temp, SP__STAT.

The foreground loop is entered until an interrupt occurs. There is only one interrupt routine—the serial port interrupt service routine. The first step in an interrupt routine is to disable other interrupts and save the flags. Next, the receiver is disabled by clearing SP__CON. Then, the TXD clock to the transmit circuitry is disabled by setting P2.6. The completion bits RI and TI are cleared (call SP__STAT__rd) to allow for the next interrupt. SP__IMAGE is returned from the SP__STAT__rd procedure. SP__IMAGE contains the status of the serial port upon entry into the service routine. The RI bit is tested in SP__IMAGE for receive completion.

If a receive just finished, then a transmit is initiated. First, the received byte is read in. Then, the nibbles are multiplied. The TXD clock to the transmit circuitry is enabled and the transmit is initiated.

Now, if a transmit caused the interrupt, then a receive is started. First, the external shift register is loaded by calling "load__shift__reg". Then, the receiver is enabled. A rising edge on REN starts another reception. The RI bit has already been cleared because SP__STAT__rd was called. Hence, when the interrupt service routine is exited, the POPF enables interrupts and allows for the receive interrupt to occur.

Once again, the foreground loop is entered to await another interrupt.

```
$debug
;*******************************************************************************************
;* TITLE:       Mode 0 demonstration                                                       *
;* AUTHOR:      Richard N. Evans                                                           *
;* DATE:        March 11, 1992                                                             *
;* DESCRIPTION:                                                                            *
;*      This program demonstrates the receive and transmit                                 *
;* functions of mode 0.  The following program is tailored for                             *
;* the 8X9XBH,JF and is upward compatable with the KB and KC.                              *
;* Testing of this program was done on an MCS-96 Eval Board                                *
;* rev 3.1 with a 8096BH, 80C196KB, and 80C196KC running at                                *
;* 12MHz.                                                                                  *
;*      The code continually reads a byte from the receive                                 *
;* buffer.   It then multiplies the upper nibble by the lower                              *
;* nibble.  Then, it outputs the product via the transmit                                  *
;* buffer.  The product is one byte in length.                                             *
;*      One interrupt routine is necessary.  The serial port                               *
;* interrupt is used.  Once entered, either a receive or                                   *
;* or transmit is executed.  If a receive done caused the                                  *
;* interrupt, then the nibbles are multiplied and the product                              *
;* is trasmitted.  However, if a transmit done caused the                                  *
;* interrupt, then a receive is initiated.                                                 *
;*******************************************************************************************

$include (sfrs.equu)
;*******************************************************************************************

;                       76543210
INT_MASK_MSK    EQU     01000000B
;                       |||||||+------- TIMER OVERFLOW
;                       ||||||+-------- A/D CONVERSION COMPLETE
;                       |||||+--------- HSI DATA AVAILABLE
;                       ||||+---------- HIGH SPEED OUTPUTS
;                       |||+----------- HSI.0 PIN
;                       ||+------------ SOFTWARE TIMER
;                       |+------------- SERIAL PORT
;                       +-------------- EXTERNAL INTERRUPT
;                                       (EXTINT OR P0.7 PIN)

IOC1_MSK        EQU     00100000B
;                       |||||||+------- SELECT PWM / #SELECT P2.5
;                       ||||||+-------- EXTERNAL INTERRUPT ACH7 / #EXTINT
;                       |||||+--------- TIMER 1 OVERFLOW INTERRUPT ENABLE / #DISABLE
;                       ||||+---------- TIMER 2 OVERFLOW INTERRUPT ENABLE / #DISABLE
;                       |||+----------- HSO.4 OUTPUT ENABLE / #DISABLE
;                       ||+------------ SELECT TXD / #SELECT P2.0
;                       |+------------- HSO.5 OUTPUT ENABLE / #DISABLE
;                       +-------------- HSI INTERRUPT FIFO FULL / #HOLDING REGISTER
;                                       LOADED

XMIT_OFF        EQU     01000000B       ;TRANSMIT OFF, USE: "OR"
;                       |+-------------  DISABLE CLOCK TO XMIT CIRCUITRY / #ENABLE
;                       +--------------  SHIFT / #LOAD FOR 74LS165
XMIT_ON         EQU     10111111B       ;TRANSMIT ON, "AND" MASK
;                        +-------------  DISABLE CLOCK TO XMIT CIRCUITRY / #ENABLE
BAUD_RATE_LO    EQU     08H             ;FASTEST RATE FOR MODE 0 -> 1.5MBAUD BH,JF
                                        ;AND 3MBAUD FOR KB,KC AT 12 MHZ
BAUD_RATE_HI    EQU     80H             ;USE XTAL1 TO CLOCK BAUD RATE GENERATOR
LOAD_SR_MSK     EQU     01111111B       ;LOAD SHIFT REGISTER
;                       +--------------  SHIFT / #LOAD FOR 74LS165
SHIFT           EQU     10000000B       ;PUT SHIFT REG IN SHIFT MODE, USE "OR"
;                       +--------------  SHIFT / #LOAD FOR 74LS165
RB8RPE_MSK      EQU     01111111B       ;MASK FOR SP_STAT RB8/RPE BIT
;                       +--------------  RB8/RPE IN SP_STAT
RI_BNO          EQU     06H             ;BIT NUMBER OF RI IN SP_STAT
TI_BNO          EQU     05H             ;BIT NUMBER OF TI IN SP_STAT
RECV_ENABLE     EQU     00001000B       ;SP_CON
;                          +----------- RECEIVER ENABLE
NIBBLE_SIZE     EQU     04H             ;THE LENGTH OF A NIBBLE IN BITS
UP_NIBBLE_MSK   EQU     00001111B       ;MASK OFF UPPER NIBBLE OF A BYTE
CODE_START      EQU     2080H           ;STARTING ADDRESS OF CODE
TOP_STACK       EQU     0F0H            ;TOP OF STACK ADDRESS
REGISTER_START  EQU     01AH            ;START OF USER REGISTER SPACE
SP_IRPT_VECTOR  EQU     200CH           ;SERIAL PORT INTERRUPT VECTOR
```

272245-9

```
;*******************************************************************************************
rseg     at     REGISTER_START

temp:           dsb     1               ;a temporary register
mltplier:       dsb     1               ;multiplier (upper nibble in byte read in)
                                        ;lower nibble in mltplier byte register
mltplicand:     dsb     1               ;multiplicand (lower nibble in byte read in)
                                        ;lower nibble in mltplicand register
product:        dsw     1               ;lower byte contains product
sp_image:       dsb     1               ;contains serial port status

;*******************************************************************************************
cseg    at      SP_IRPT_VECTOR
        dcw     serial_isr

;*******************************************************************************************
cseg    at      CODE_START
        ld      sp,#TOP_STACK           ;set the stack pointer to the top of the stack
        di
        call    init                    ;initialize registers and start reception
        ei
foreground:
        br      foreground              ;wait for interrupt

;*******************************************************************************************
;* serial_isr                                                                              *
;*      This routine services the serial port interrupt.  If                               *
;* a receive or transmit is done, then the RI and TI bits get                              *
;* set and this routine is vectored to.                                                    *
;*      If a receive caused the interrupt, then the byte is                                *
;* read.  The lower and upper nibbles are multiplied together.                             *
;* Finally, the product is written to the serial buffer which                              *
;* initiates a transmit.                                                                   *
;*      If a transmit caused the interrupt.  Then that means a                             *
;* product was just written out the RXD pin.  So, a receive is                             *
;* initiated to get the next byte.                                                         *
;*                                                                                         *
;* INPUT:       sbuf                                                                       *
;* OUTPUT:      product, mltplier, mltplicand, port2, sbuf, sp_image                       *
;* CHANGED:     temp, sp_con, sp_stat, (plus OUTPUT)                                       *
;*******************************************************************************************
serial_isr:
        pushf                           ;save flags, disable irpts
        clrb    sp_con                  ;disable the receiver
        ldb     port2,#XMIT_OFF         ;disable the transmit circuitry
        call    sp_stat_rd              ;get sp_image
        jbs     sp_image,RI_BNO,get_byte ;if receive irpt, goto receive routine
        call    load_shift_reg          ;load shift register, put in shift
        ldb     sp_con,#RECV_ENABLE     ;transmit done so enable receiver
        br      serial_isr_end          ;exit isr

  get_byte:
        ldb     temp,sbuf               ;read the received byte
        ldb     mltplier,temp           ;multiplier is the upper nibble in byte recvd
        shrb    mltplier,#NIBBLE_SIZE   ;shift out the lower nibble and keep upper
        andb    mltplier,#UP_NIBBLE_MSK ;preserve multiplier nibble
        ldb     mltplicand,temp         ;get multiplicand nibble from byte
        andb    mltplicand,#UP_NIBBLE_MSK ;mask off multiplier nibble

        mulb    product,mltplier,mltplicand ;multiply and get product
        andb    port2,#XMIT_ON          ;enable clock to transmit circuit, put shift reg
                                        ;in load mode.
        ldb     sbuf,product            ;start transmission of product

serial_isr_end: popf                    ;restore flags, and enable irpts
        ret                             ;return to calling procedure

;*******************************************************************************************
;* load_shift_reg                                                                          *
;*      This procedure will load the external shift register.  The shift register is a     *
;* 74LS165.  A high to low transition on the shift/#load pin, loads the shift register.    *
;* Hence, this routine outputs a high to low transition to port 2.  The other bits in      *
;* port 2 are preserved.                                                                   *
;*                                                                                         *
;* INPUT:                                                                                  *
```

272245-10

```
;* OUTPUT:      port2                                                              *
;* CHANGED:     port2                                                              *
;************************************************************************************
load_shift_reg:
        orb     port2,#SHIFT            ;make shift/#load high
        andb    port2,#LOAD_SR_MSK      ;shift/#load goes low
        orb     port2,#SHIFT            ;put in shift mode
        ret                             ;return to calling procedure

;************************************************************************************
;* init                                                                             *
;*      This procedure initializes some control registers and starts a receive.     *
;*                                                                                  *
;* INPUT:       sp_stat                                                             *
;* OUTPUT:      port2                                                               *
;* CHANGED:     int_mask, ioc1, baud_reg, sp_con, sp_stat, port2, temp, sp_image    *
;************************************************************************************
init:
        ldb     int_mask,#INT_MASK_MSK  ;allow serial port irpt
        ldb     ioc1,#IOC1_MSK          ;enable txd
        ldb     baud_reg,#BAUD_RATE_LO  ;set the baud rate generator
        ldb     baud_reg,#BAUD_RATE_HI
        call    load_shift_reg          ;load the shift register and put in shift mode
        orb     port2,#XMIT_OFF         ;disable the clock to the transmit circuitry
        clrb    sp_con                  ;Disable receiver
        ldb     sp_con,#RECV_ENABLE     ;enable receiver and put in mode 0, start recept
        call    sp_stat_rd              ;clear RI bit
        ret                             ;return to calling program

;************************************************************************************
;* sp_stat_rd                                                                       *
;* This subroutine will clear RI, TI, and RB8 in sp_stat.  The other status bits are*
;* preserved and returned in sp_image.  This subroutine is meant to replace the    *
;* instruction  "ldb   sp_image,sp_stat".                                           *
;*                                                                                  *
;* INPUT:       sp_stat                                                             *
;* OUTPUT:      sp_image                                                            *
;* CHANGED:     temp, sp_stat, sp_image                                             *
;************************************************************************************
sp_stat_rd:
        clrb    sp_image                ;clear status bits

get_status:
        ldb     temp,sp_stat            ;get current status
        orb     sp_image,temp           ;accumulate status bits

        jbs     temp,RI_BNO,get_status  ;if ti is set, then read again
        jbs     temp,TI_BNO,get_status  ;if ri is set, then read again, otherwise ti
                                        ;and ri are clear

        andb    sp_image,#RB8RPE_MSK    ;clear out the RB8/RPE bit
        orb     sp_image,temp           ;include the most recent copy of RB8/RPE

        ret                             ;return to calling procedure

        end
```

272245-11