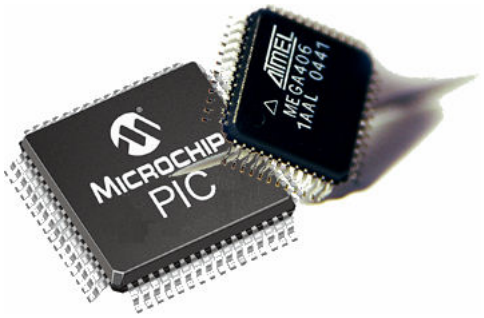
The 8051 Serial Port

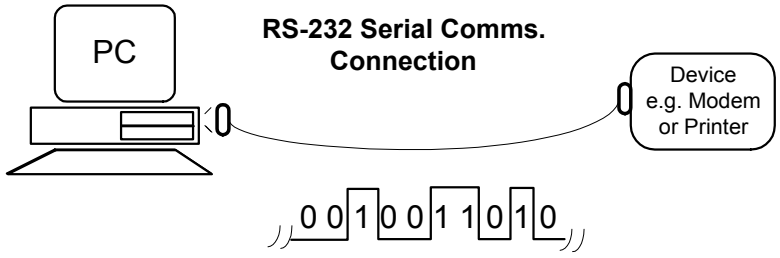


The 8051 Serial Port

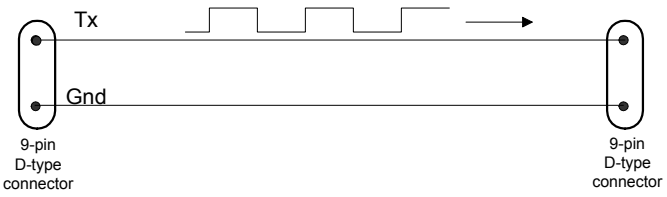
6.1 OVERVIEW OF ASYNCHRONOUS SERIAL COMMUNICATIONS

RS-232 Serial Communications

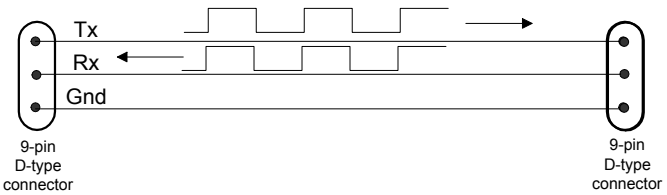
The EIA RS-232 serial communication standard is a universal standard, originally used to connect teletype terminals to modem devices. Figure 6.1(a) shows a PC connected to a device such as a modem or a serial printer using the RS-232 connection. In a modern PC the RS-232 interface is referred to as a COM port. The COM port uses a 9-pin D-type connector to attach to the RS-232 cable. The RS-232 standard defines a 25-pin D-type connector but IBM reduced this connector to a 9-pin device so as to reduce cost and size. Figure 6.1(b) shows a simple simplex serial communication link where data is being transmitted serially from left to right. A single Tx (transmit) wire is used for transmission and the return (Gnd) wire is required to complete the electrical circuit. Figure 6.1(c) shows the inclusion of another physical wire to support full-duplex (or half-duplex) serial communication. The RS-232 (COM port) standard includes additional signal wires for "hand-shake" purposes, but the fundamental serial communication can be achieved with just two or three wires as shown.



a) Serial communication link



b) Simple transmission using two wires



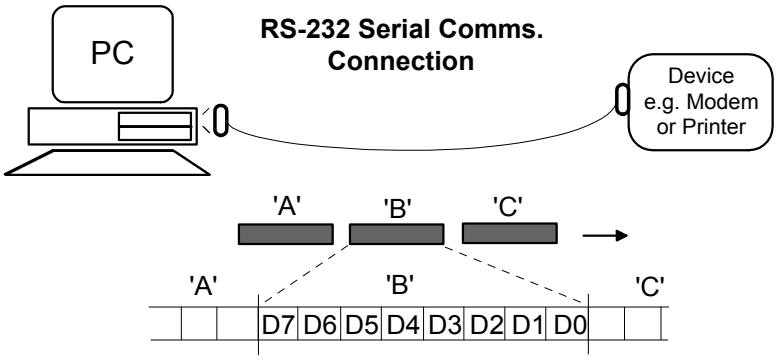
c) Two way communication using three wires

Figure 6.1 Serial communications

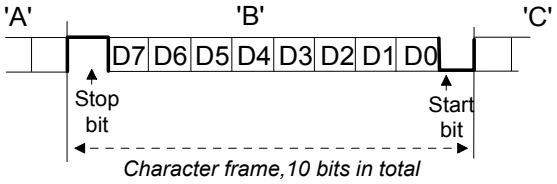
The serial data is transmitted at a predefined rate, referred to as the baud rate. The term baud rate refers to the number of state changes per second which is the same as the bit rate for this particular communication scheme. Typical baud rates are: 9600 bps; 19,200 bps; 56kbps etc.

Asynchronous Serial Communications

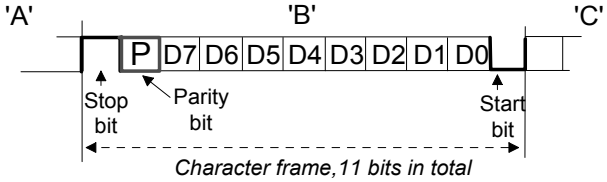
Since data is sent is a serial fashion, without any reference to a timing clock to help synchronise the receiver clock in terms of frequency and phase, the system is said to be non-synchronous, or asynchronous. The baud rate clocks at each end of the RS-232 link are set to the same frequency values but there is no mechanism to synchronise these clocks. Figure 6.2(a) shows three bytes transmitted by the PC. Assume the bytes are *ascii* coded to represent the characters A, B and C. The receiver needs to know exactly where each character starts and finishes. To achieve this the data character is framed with a start bit at the beginning of each character and a stop bit at the end of each character. Figure 6.2(b) shows the start bit as a low logic level and the stop bit as a high logic level. Thus the receiver can detect the start bit and it then clocks in the next eight



a) Sequence without framing



b) Framed data



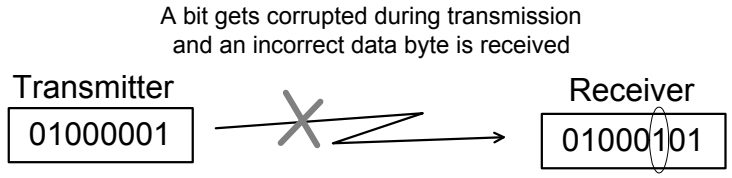
c) Framed data including a parity bit

Figure 6.2 Asynchronous transmission

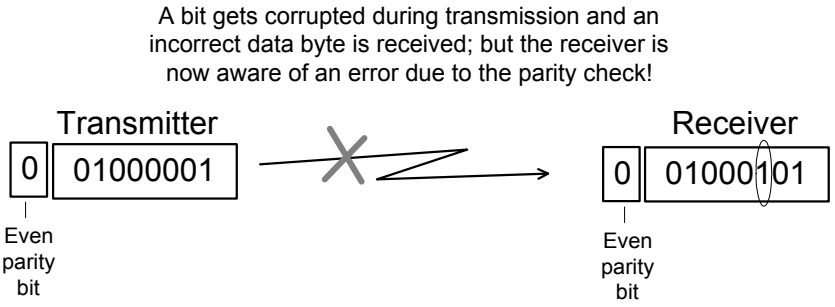
character bits. The receiver then expects to find the stop bit, existing as a logic high bit. This is a crude form of synchronisation applied to a system which is inherently non-synchronous. A high price is paid for this form of synchronisation in terms of bandwidth, as for every eight bits of data transmitted two bits are required to support the framing. Ten bits are transmitted to support eight bits of data thus the scheme is, at best, just eighty percent efficient. Figure 6.2(c) shows the inclusion of an additional parity bit for error control purposes.

Single Bit Parity for Error Checking

All communication systems are prone to errors. An RS-232 communication system is susceptible to bit errors as data bits can become corrupted (bit changes from 1 to 0 or from 0 to 1). Such corruption is often caused by unwanted electrical noise coupled into the wiring. Figure 6.3(a) shows an example where an 8-bit data character is transmitted and a single bit becomes corrupted during transmission. The receiver gets the wrong data. The receiver cannot known that the received data contains an error. Figure 6.2(b) show a single bit parity scheme where the parity bit is calculated at the transmitter and this parity bit is sent along with the eight data bits. The receiver can apply a test on the received data to establish whether or not an error exists in the



a) Undetected error



b) Presence of error is detected

Figure 6.3 Single bit parity error detection

received data. In this example *even* parity is used. The parity bit is calculated at the transmitter so that all of the bits, including the parity bit add up to an even number of ones. Thus, in the example, the parity bit is set to 0 so that an even number of ones (two ones) exists across the 9 bits. The receiver checks the received data for *even* parity and in this case finds that the parity test fails. The receiver now knows that an error exists and it is up to a higher layer protocol to act on the error. Note, the receiver does not know which bit is in error, it is simply aware than an error exist in the received data. If the parity bit itself had been corrupted the same parity test would detect this error also. If any

odd number of bits (1, 3, 5, 7 or 9 bits) are in error the simple parity test will detect the error. However, if an even number of bits are in error (2, 4, 6 or 8 bits) then such errors will go unnoticed in the parity test. Since the majority of errors in communication systems are single bit errors then the simple single bit parity scheme is worthwhile. There are more complex techniques used to provide more rigorous error checking and error correction.

6.2 THE 8051 UART

The 8051 includes a hardware UART to support serial asynchronous communications so that, typically, the product can support RS-232 standard communication. The UART (**U**niversal **A**synchronous **R**eveiver and **T**ransmitter) block diagram is shown in figure 6.4. In our examples the BAUD clocks are, in fact, a single clock source provided by Timer/Counter 1.

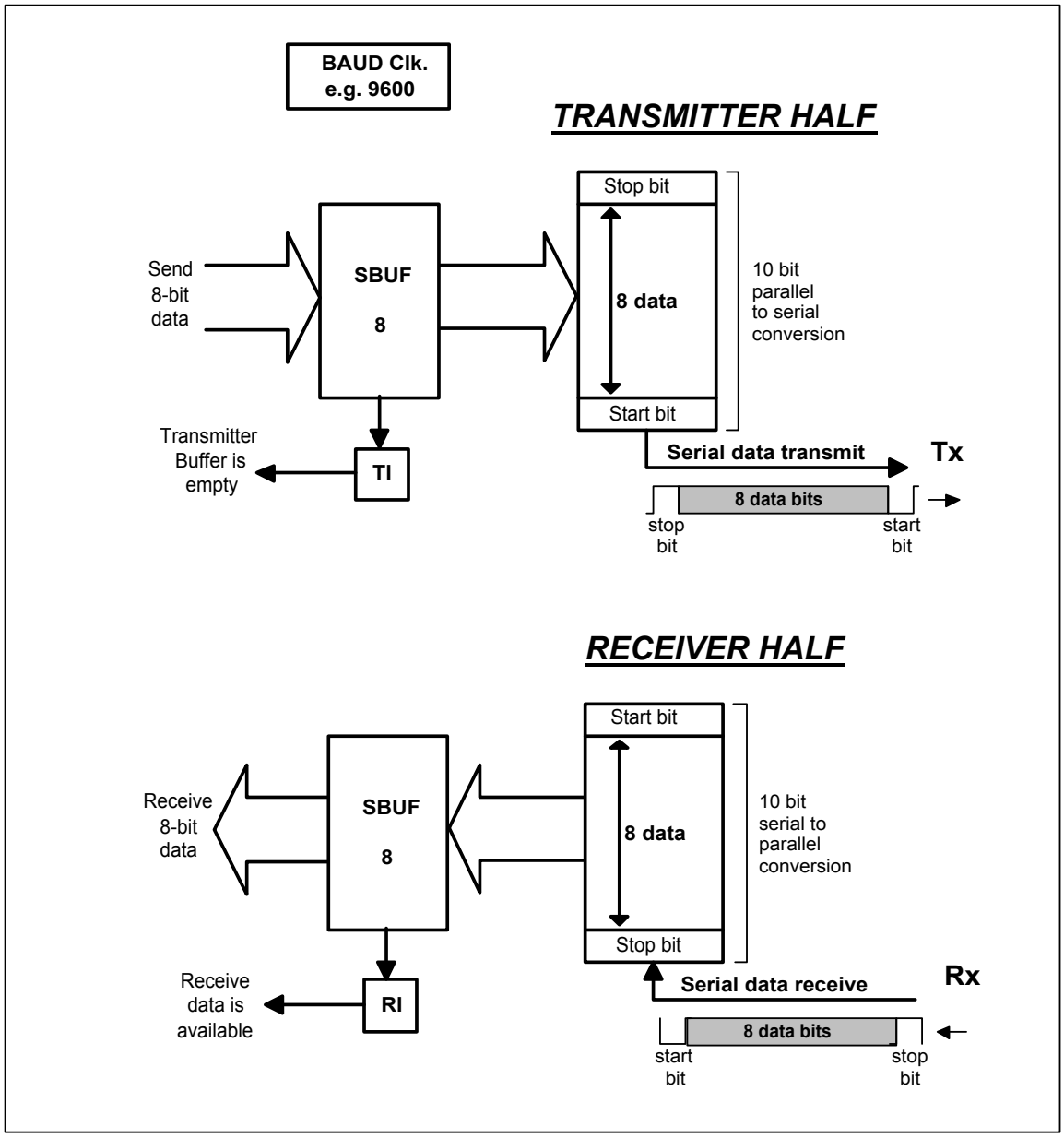


Figure 6.4 UART block diagram

The UART can be configured for 9-bit data transmission and reception. Here 8 bits represent the data byte (or character) and the ninth bit is the parity bit. Figure 6.5 shows a block diagram for the UART transmitter, where the ninth bit is used as the parity bit.

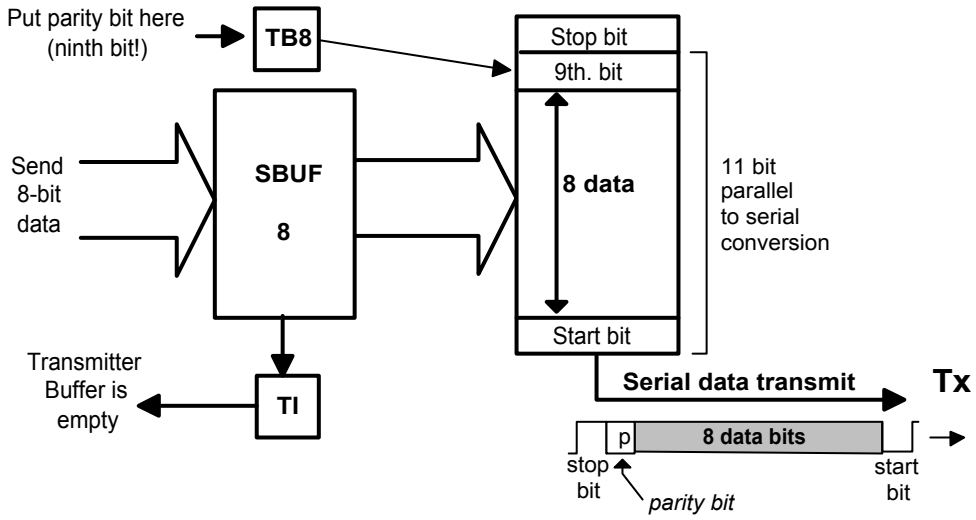


Figure 6.5 Block diagram of UART transmitter, using the 9th. bit

Figure 6.6 shows a block diagram for the UART receiver, where the ninth bit is used as the parity bit.

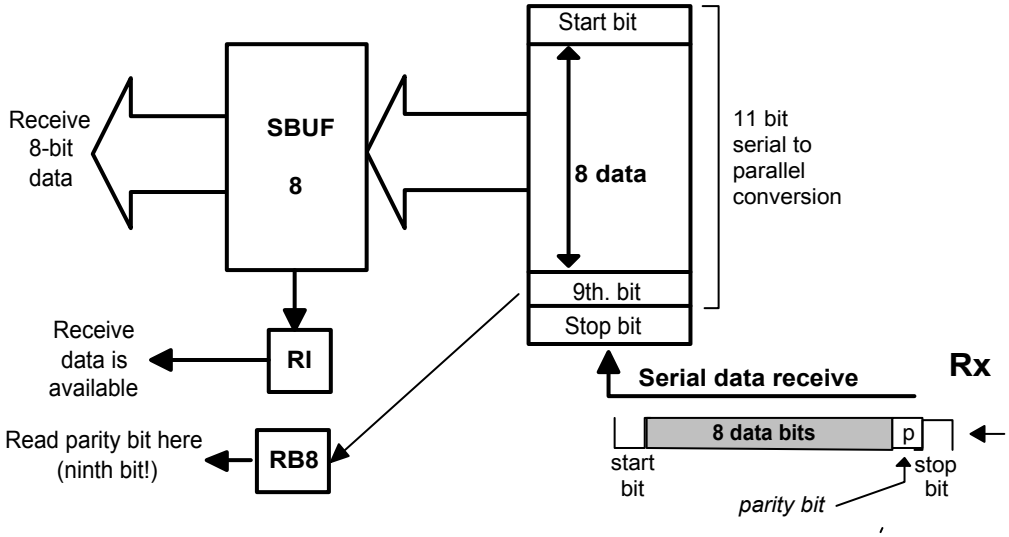


Figure 6.6 Block diagram of UART receiver, using the 9th. bit

SBUF is an SFR register which can be written to, so as to hold the next data byte to be transmitted. Also it can be read from to get the latest data byte received by the serial port. SBUF is thus effectively two registers: one for transmitting and one for receiving.

The SCON (**S**erial **Co**ntrol) register is an SFR register, used for configuring and monitoring the serial port status.

SCON register

SM0 *msb*	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI** *lsb*

SM0, SM1 bits define the mode of operation, such as the number of data bits (8 or 9), the clock source etc. Our examples will use *mode 3,* which specifies 9 data bits (8 data plus a parity bit) with the clock source being Timer/Counter 1.

SM2 is set to 0 for normal operation

REN is set to 1 to enable reception, 0 to disable reception

TB8 is the ninth bit (parity bit) to be transmitted

RB8 is the ninth bit received (parity bit)

TI Transmit Interrupt flag. A logic 1 indicates that transmit buffer (SBUF) is empty. This flag must be cleared by software.

RI Receive Interrupt flag. A logic 1 indicates that data has been received in the receive buffer (SBUF). This flag must be cleared by software.

In the example programs the serial port is initialised for *mode 3* operation with the receiver enabled using the following instruction:

```
MOV  SCON,  #11010000B
```

SETTING THE BAUD RATE

Timer/Counter 1 (in SCON mode 3) provides the serial port baud rate clock. Usually the 8-bit auto reload operation (Timer/Counter mode 2) is used. The table shows some values defined for the TH1 register to achieve some of the more common baud rates. The values shown assume a processor clock rate of 11.059MHz. This is a common crystal value for 8051 based designs as it divides down to provide accurate baud rates.

Baud rate	**Timer/Counter1 TH1 value**	**PCON.7 SMOD**	**8051clock frequency**
300	A0h	0	11.059MHz.
1,200	D0h	0	11.059MHz.

2,400	FAh	0	11.059MHz.
9,600	FDh	0	11.059MHz.

Note. The most significant bit of the PCON register is assumed to be at 0. If this were set to 1 the baud rate value would be doubled.

Based on the above we could set up the timer for 9,600 baud operation using the following code:

```
MOV TMOD, #00100000B   ; timer/counter 1 set for mode 2, 8-bit TIMER operation
MOV TH1, #0FDh         ; timer/counter 1 is timed for 9600 baud
SETB TR1               ; timer/counter 1 is enabled and will just free run now
```

Some sample programs using the serial port are listed below as follows:

SEND_1.A51
This program continuously transmits the *ascii* 'A' character. The 9th. bit exists but is ignored. Listing 6.1 shows the source code listing.

SEND_2.A51
This program example is similar to SEND_1.A51 above but puts an *even* parity bit into the ninth bit. Listing 6.2 shows the source code listing.

READ_1.A51
This program reads a character from the serial port and stores the character in R7. The parity bit is ignored. Listing 6.3 shows the source code listing.

READ_2.A51
This program is an interrupt driven version of the READ_1.A51 program. Listing 6.4 shows the source code listing.

SEND_3.A51
This program sends a block of 100 characters from external memory out through the serial port. Listing 6.5 shows the source code listing.

```
;=====================================================================
; SEND_1.A51
; Transmits the ascii 'A' character continuously using the 8051 serial port.
; Uses 9 bit data at 9600 baud. Parity bit exists but is not calculated.
; Uses POLLED operation, not interrupt driven
;;=====================================================================

ORG    0000h                ; entry address for 8051 RESET
       LJMP MAIN            ; MAIN starts beyond interrupt vector space


       ORG 0100h
MAIN:

; set up timer/counter 1 to drive 9600 baudrate
       MOV TMOD, #00100000B     ; timer/counter 1 is set for mode 2 8-bit TIMER
       MOV TH1, #0FDh           ; timer/counter 1 is timed for 9600 baud
       SETB TR1                 ; timer/counter 1 is enabled and will free run

; Initialise serial port for mode 3: operation
       MOV SCON, #11010000B

SEND:
       MOV SBUF, #41h           ; acsii 'A' to SBUF

LOOP:
       JNB TI, LOOP         ; loop testing TI to know when data is sent
       CLR TI               ; clear TI
       LJMP SEND            ; back to send 'A' again


END
```

Listing 6.1 SEND_1.A51

```
;=========================================================
;SEND_2.A51
;
; Like SEND_1.A51 above but parity bit is calculated and used. EVEN parity.
; The program transmits the ascii 'A' character continuously, using the 8051
; serial port. It uses 9 bit data at 9600 baud. Uses POLLED operation,
; Not interrupt driven
;
;;=========================================================

        ORG 0000h           ; entry address for 8051 RESET
        LJMP MAIN           ; MAIN starts beyond interrupt vector space

        ORG 0100h
MAIN:

;set up timer/counter 1 to drive 9600 baudrate
        MOV TMOD, #00100000B    ; timer/counter 1 is set for mode 2, 8-bit TIMER
        MOV TH1, #0FDh          ; timer/counter 1 is timed for 9600 baud
        SETB TR1                ; timer/counter 1 is enabled and will free run

; Initialise serial port for mode 3: operation
        MOV SCON, #11010000B


; Move Ascii ‘A’ to SBUF via the accumulator so that parity bit is calculated

SEND:  MOV A, #41h
        MOV SBUF, A             ; acsii 'A' to SBUF

        MOV C, P                ; the parity flag in the PSW is moved to carry flag
        MOV TB8, C              ; the carry flag is move to TB8

        LOOP:
        JNB TI, LOOP            ; loop testing TI to know when data is sent
        CLR TI                  ; clear TI
        LJMP SEND               ; back to send 'A' again

END
```

Listing 6.2 SEND_2.A51

```
;=====================================================================
; READ_1.A51
; Program to receive a character from serial port and save this character
; in R7. Uses 9 bit data at 9600 baud. Parity bit exists but is ignored.
; Uses POLLED operation, not interrupt driven.
;
;
;=====================================================================

        ORG 0000h           ; entry address for 8051 RESET
        LJMP MAIN           ; MAIN starts beyond interrupt vector space

        ORG 0100h
MAIN:

;set up timer/counter 1 to drive 9600 baudrate
        MOV TMOD, #00100000B    ; timer/counter 1 is set for mode 2 8-bit TIMER
        MOV TH1, #0FDh          ; timer/counter 1 is timed for 9600 baud
        SETB TR1                ; timer/counter 1 is enabled and will free run

; initialise serial port for mode 3: operation
        MOV SCON, #11010000B

INCHAR:

LOOP:  JNB RI, LOOP         ; loop testing RI to know when data is received
        CLR RI              ; clear RI
        MOV R7, SBUF        ; read data to R7

END
```

Listing 6.3 READ_1.A51

```
;===============================================================
; READ_2.A51
; Like READ_1.A51 program but this is an interrupt program. When a character is
; received from serial port RI interrupt ISR saves the received character in R7.
; Uses 9 bit data at 9600 baud. Parity bit exists but is ignored.
;
;
;===============================================================

        ORG 0000h           ; entry address for 8051 RESET
        LJMP MAIN           ; MAIN starts beyond interrupt vector space

        ORG 23h             ; vector address serial port interrupt
        LJMP ISR_SERIAL

        ORG 0100h
MAIN:

;set up timer/counter 1 to drive 9600 baudrate
        MOV TMOD, #00100000B    ; timer/counter 1 is set for mode 2 8-bit TIMER
        MOV TH1, #0FDh          ; timer/counter 1 is timed for 9600 baud
        SETB TR1                ; timer/counter 1 is enabled and will free run

; initialise serial port for mode 3: operation
        MOV SCON, #11010000B

; enable the serial port interrupt
        MOV IE, #10010000B


LOOP:  LJMP LOOP            ; Main just loops around doing nothing!


;===============================================================
; ISR_SERIAL
; TI or RI will cause a serial port interrupt. This routine ignores TI
; but on RI it reads the received character to R7
;===============================================================
ISR_SERIAL:

        JNB RI, RETURN      ; return if RI is not set (TI caused int.)
        MOV R7, SBUF        ; read data to R7
        CLR RI              ; clear RI


RETURN:
        RETI                ; return from interrupt
END
```

Listing 6.4 READ_2.A51

```
;===============================================================
; SEND_3.A51
; Program sends block of 100 characters to serial port from XDATA RAM
; starting at location 2000h.
; Uses 9 bit data at 9600 baud. Parity bit exist but is ignored.
; Uses POLLED operation, not interrupt driven.
;
;;===============================================================

        ORG 0000h           ; entry address for 8051 RESET
        LJMP MAIN           ; MAIN starts beyond interrupt vector space

        ORG 0100h
MAIN:

;set up timer/counter 1 to drive 9600 baudrate
        MOV TMOD, #00100000B    ; timer/counter 1 is set for mode 2 8-bit TIMER
        MOV TH1, #0FDh          ; timer/counter 1 is timed for 9600 baud
        SETB TR1                ; timer/counter 1 is enabled and will free run

; initialise serial port for mode 3: operation
        MOV SCON, #11010000B

;Initialise DPTR as memory pointer, starting at 2000
;and initialise R6 as send character counter with value 100d
        MOV DPTR, #2000h
        MOV R6, #100d


SEND_BLOCK:
        LCALL SEND_CHAR         ;send a character
        INC DPTR                ;increment DPTR memory pointer
        DJNZ R6, SEND_BLOCK     ; decrement R6 and loop back if not zero

STOP:  LJMP STOP                ; program is finished and stops


SEND_CHAR:
        MOVX A, @ DPTR          ; Memory data to SBUF, via Acc
        MOV SBUF, A

LOOP:
        JNB TI, LOOP            ; loop testing TI to know when data is sent
        CLR TI                  ; clear TI
        RET

END
```

Listing 6.5 SEND_3.A51