AVR Microcontroller

Microprocessor Course

Chapter 11

AVR SERIAL PORT PROGRAMMING IN ASSEMBLY AND C

Bahman1397 (Version 1.2)

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.1 BASIC OF SERIAL COMMUNICATION

When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks. For some devices, such as printers, the information is simply grabbed from the 8-bit data bus and presented to the 8-bit data bus of the device. This can work only if the cable is not too long, because long cables diminish and even distort signals. Furthermore, an 8-bit data path is expensive. For these reasons, serial communication is used for transferring data between two systems located at distances of hundreds of feet to millions of miles apart.

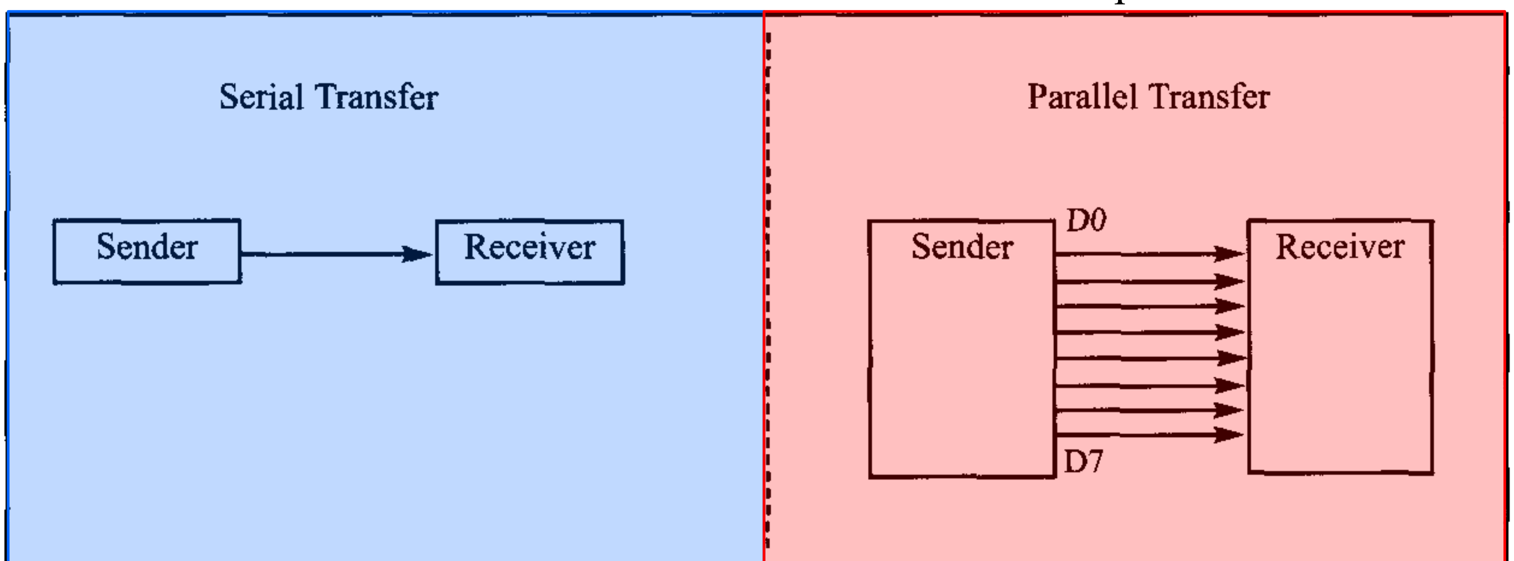


Figure 11-1. Serial versus Parallel Data Transfer

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.1 BASIC OF SERIAL COMMUNICATION

Serial communication enables two computers located in two different cities to communicate over the telephone.

For serial data communication to work, the byte of data must be converted to serial bits using a parallel-in-serial-out shift register; then it can be transmitted over a single data line. This also means that at the receiving end there must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte. Of course, if data is to be transmitted on the telephone line, it must be converted from 0s and 1s to audio tones, which are sinusoidal signals. This conversion is performed by a peripheral device called a ***modem,*** which stands for "***MOdulator/DEModulator***".

When the distance is short, the digital signal can be transmitted as it is on a simple wire and requires no modulation. For long-distance data transfers using communication lines such as a telephone, however, serial data communication requires a modem to ***modulate*** (convert from 0s and 1 s to audio tones) ***and demodulate*** (convert from audio tones to 0s and 1s).

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.1 BASIC OF SERIAL COMMUNICATION

Serial data communication uses two methods

1. The synchronous method transfers a block of data (characters) at a time
2. The asynchronous method transfers a single byte at a time.

It is possible to write software to use either of these methods, but the programs can be tedious and long. For this reason, special IC chips are made by many manufacturers for serial data communications.

These chips are commonly referred to as

UART (universal asynchronous receiver-transmitter) and
USART (universal synchronous/asynchronous receiver-transmitter).

The AVR chip has a built-in USART.

Half- and full-duplex transmission

Duplex transmissions can be half or full duplex, depending on whether or not the data transfer can be simultaneous.

If data is transmitted one way at a time, it is referred to as ***half duplex***.
If the data can go both ways at the same time, it is ***full duplex***.

Of course, full duplex requires two wire conductors for the data lines (in addition to the signal ground), one for transmission and one for reception, in order to transfer and receive data simultaneously.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.1 BASIC OF SERIAL COMMUNICATION

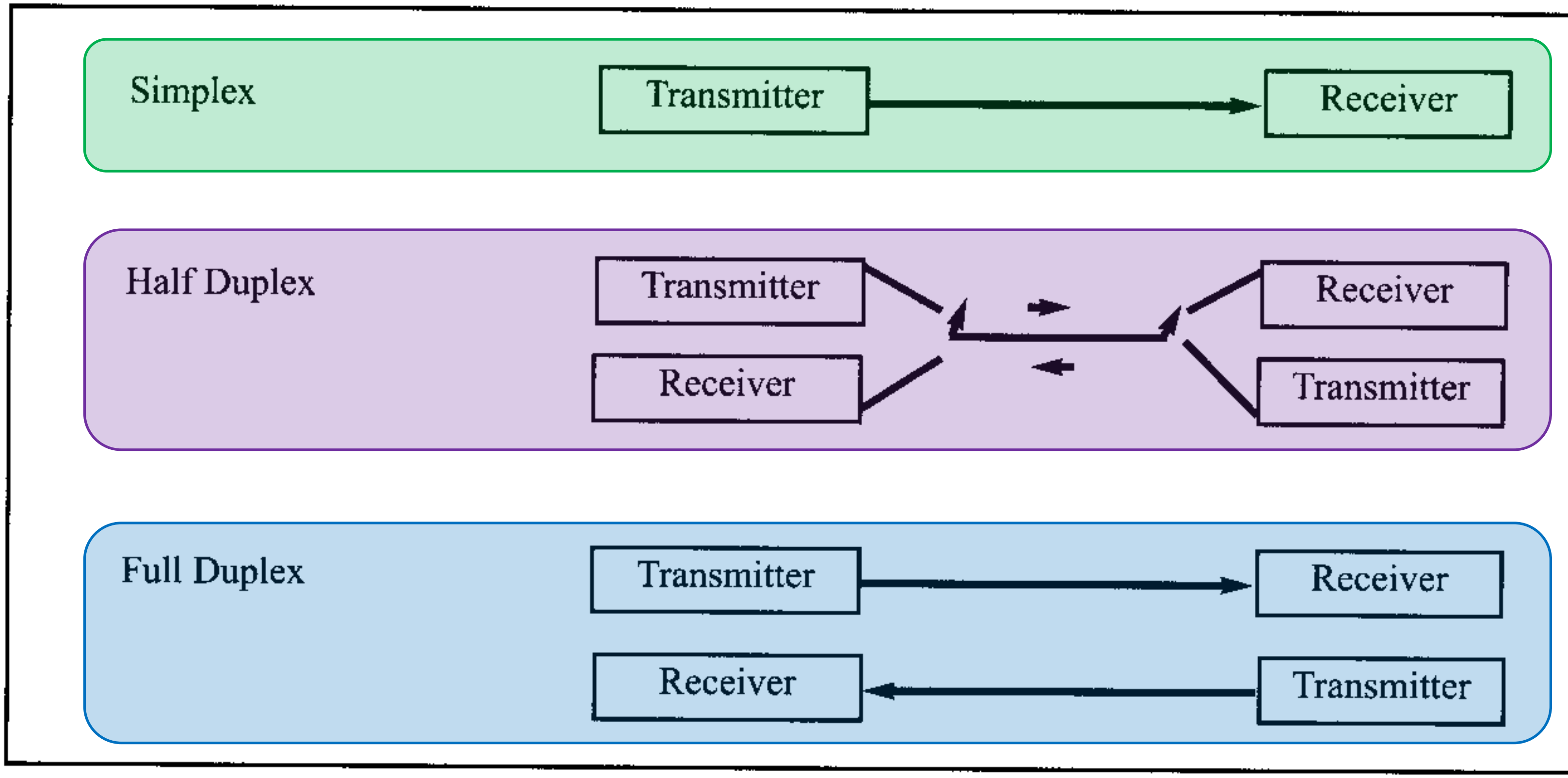


Figure 11-2. Simplex, Half-, and Full-Duplex Transfers

Asynchronous serial communication and data framing

Asynchronous serial data communication is widely used for character-oriented transmissions. In this method, each character is placed between start and stop bits. This is called ***framing.*** In data framing for asynchronous communications, the data, such as ASCII characters, are packed between a start bit and a stop bit. The start bit is always a 0 (low), and the stop bit(s) is 1 (high).

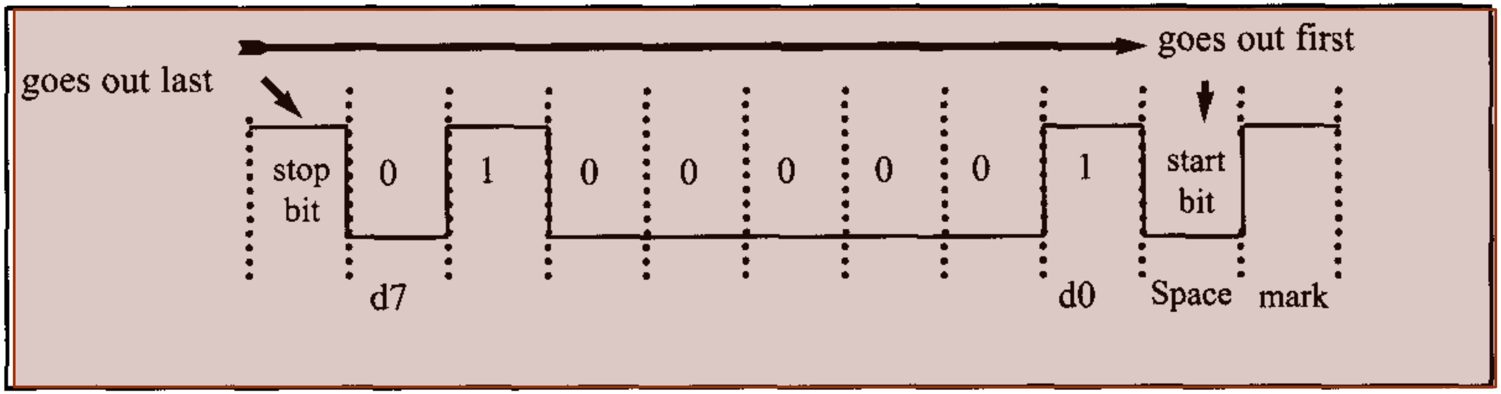


Figure 11-3. Framing ASCII 'A' (41H)

Notice that the LSB is sent out first.

Data transfer rate

The rate of data transfer in serial data communication is stated in **bps** (bits per second). Another widely used terminology for bps is **baud rate**.

However, the baud and bps rates are not necessarily equal. As far as the conductor wire is concerned, the baud rate and bps are the same.

The data transfer rate of a given computer system depends on communication ports incorporated into that system. Notice that in asynchronous serial data communication, the baud rate is generally limited to 100,000 bps.

RS232 standards

Today, RD232 is one of the most widely used serial I/O interfacing standards. This standard is used in PCs and numerous types of equipment. Because the standard was set long before the advent of the TTL logic family, however, its input and output voltage levels are not TTL compatible.

In RS232, a 1 is represented by -3 to -25 V, while a 0 bit is +3 to +25 volts, making -3 to +3 undefined. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic levels to the RS232 voltage levels, and vice versa.

MAX232 IC chips are commonly referred to as **line drivers**.

RS232 pins

Table 11-1 shows the pins for the original RS232 cable and their labels, commonly referred to as the DB-25 connector. In labeling, DB-25P refers to the plug connector (male), and DB-25s is for the socket connector (female).

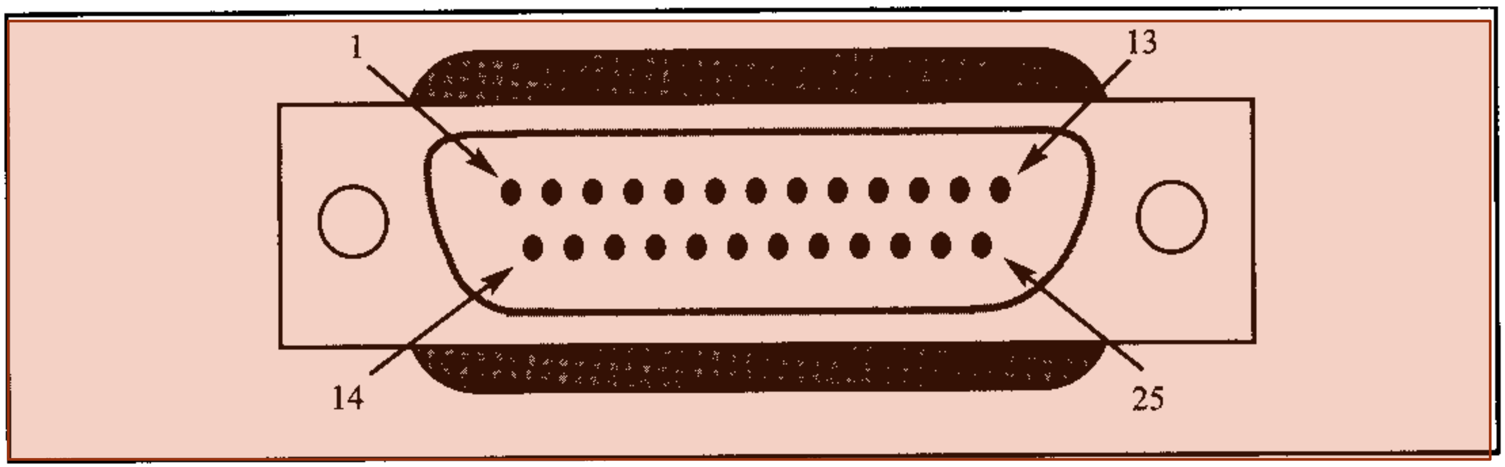


Figure 11-4. The Original RS232 Connector DB-25 (No longer in use)

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.1 BASIC OF SERIAL COMMUNICATION

Because not all the pins were used in PC cables, IBM introduced the DB-9 version of the serial I/O standard, which uses only 9 pins, as shown in Table 11-2. The DB-9 pins are shown in Figure 11-5.

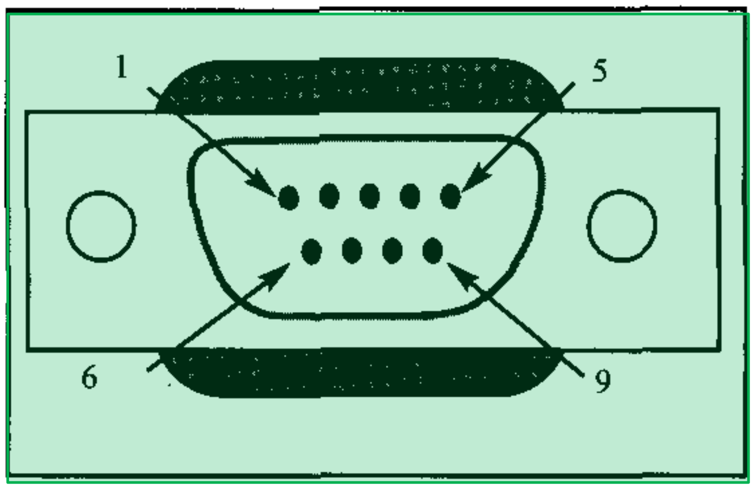


Figure 11-5. 9-Pin Connector for DB-9

Table 11-2: IBM PC DB-9 Signals

Pin	Description
1	Data carrier detect (DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (DSR)
7	Request to send (RTS)
8	Clear to send (CTS)
9	Ring indicator (RI)

mashhoun@iust.ac.ir Iran Univ of Science & Tech

Data communication classification

Current terminology classifies data communication equipment as DTE (data terminal equipment) or DCE (data communication equipment).

- DTE refers to terminals and computers that send and receive data, while
- DCE refers to communication equipment, such as modems, that are responsible for transferring the data.

Notice that all the RS232 pin function definitions of Tables 11-1 and 11-2 are from the DTE point of view.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.1 BASIC OF SERIAL COMMUNICATION

The simplest connection between a PC and a microcontroller requires a minimum of three pins, TX, RX, and ground.

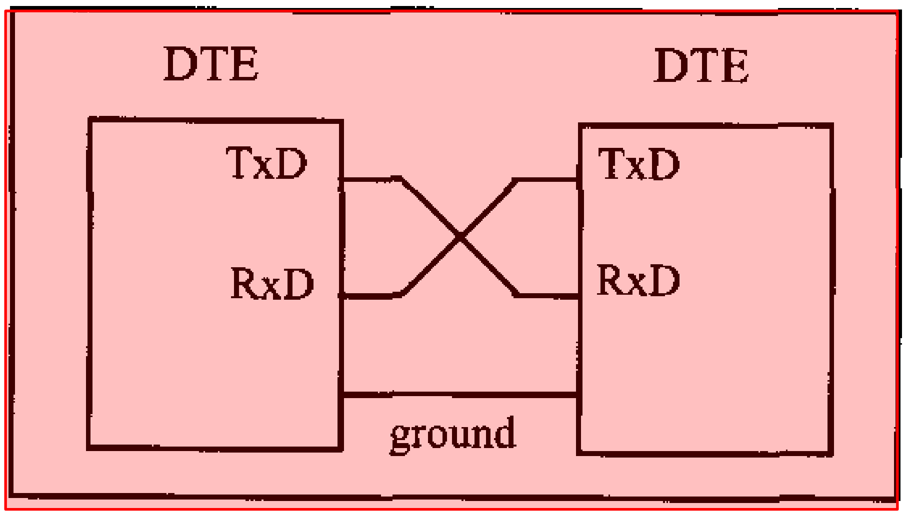


Figure 11-6. Null Modem Connection

mashhoun@iust.ac.ir

Examining RS232 handshaking signals

To ensure fast and reliable data transmission between two devices, the data transfer must be coordinated. There must be a way to inform the sender to stop sending data. Many of the pins of the RS-232 connector are used for handshaking signals. Their description is provided below only as a reference, and they can be bypassed

1. DTR (data terminal ready). When the terminal (or a PC COM port) is turned on, after going through a self-test, it sends out signal DTR to indicate that it is ready for communication.
2. DSR (data set ready). When the DCE (modem) is turned on and has gone through the self-test, it asserts DSR to indicate that it is ready to communicate.
3. RTS (request to send). When the DTE device (such as a PC) has a byte to transmit, it asserts RTS to signal the modem that it has a byte of data to transmit. RTS is an active-LOW output from the DTE and an input to the modem.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.1 BASIC OF SERIAL COMMUNICATION

4. CTS (clear to send). In response to RTS, when the modem has room to store the data it is to receive, it sends out signal CTS to the DTE (PC) to indicate that it can receive the data now. This input signal to the DTE is used by the DTE to start transmission.
5. DCD (data carier detect). The modem asserts signal DCD to inform the DTE (PC) that a valid carrier has been detected and that contact between it and the other modem is established. Therefore, DCD is an output from the modem and an input to the PC (DTE).
6. RI (ring indicator). An output from the modem (DCE) and an input to a PC (DTE) indicates that the telephone is ringing. RI goes on and off in synchronization with the ringing sound. Of the six handshake signals, this is the least often used because modems take care of answering the phone.

x86 PC COM ports

The x86 PCs used to have two COM ports. Both COM ports were RS232-type connectors. The COM ports were designated as COM1 and COM2.

In recent years, one of these has been replaced with the USB port, and COM1 is the only serial port available, if any. We can connect the AVR serial port to the COM1 port of a PC for serial communication experiments. In the absence of a COM port, we can use a COM-to-USB converter module.

11.2 ATMEGA32 CONNECTION TO RS232

RX and TX pins in the ATmega32

The ATmega32 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TX and RX and are part of the Port D group (PD0 and PD1) of the 40-pin package. These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible. One such line driver is the MAX232 chip.

MAX232

The MAX232 converts from RS232 voltage levels to TTL voltage levels, and vice versa. One advantage of the MAX232 chip is that it uses a +5 V power source, which is the same as the source voltage for the AVR. In other words, with a single +5 V power supply we can power both the AVR and MAX232.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.2 ATMEGA32 CONNECTION TO RS232

The MAX232 has two sets of line drivers for transferring and receiving data, as shown in Figure 11-7. The line drivers used for TX are called T1 and T2,

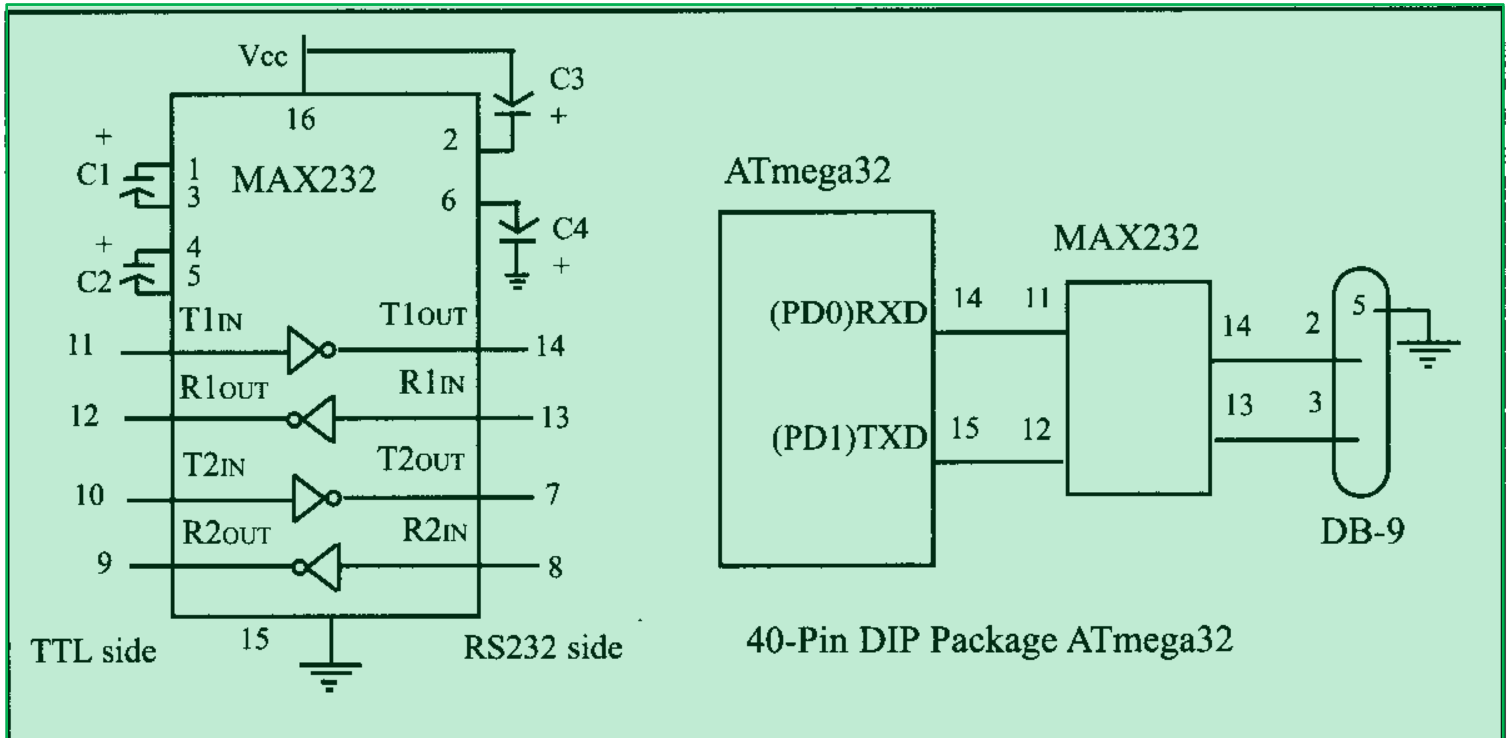


Figure 11-7. (a) Inside MAX232 and (b) Its Connection to the ATmega32 (Null

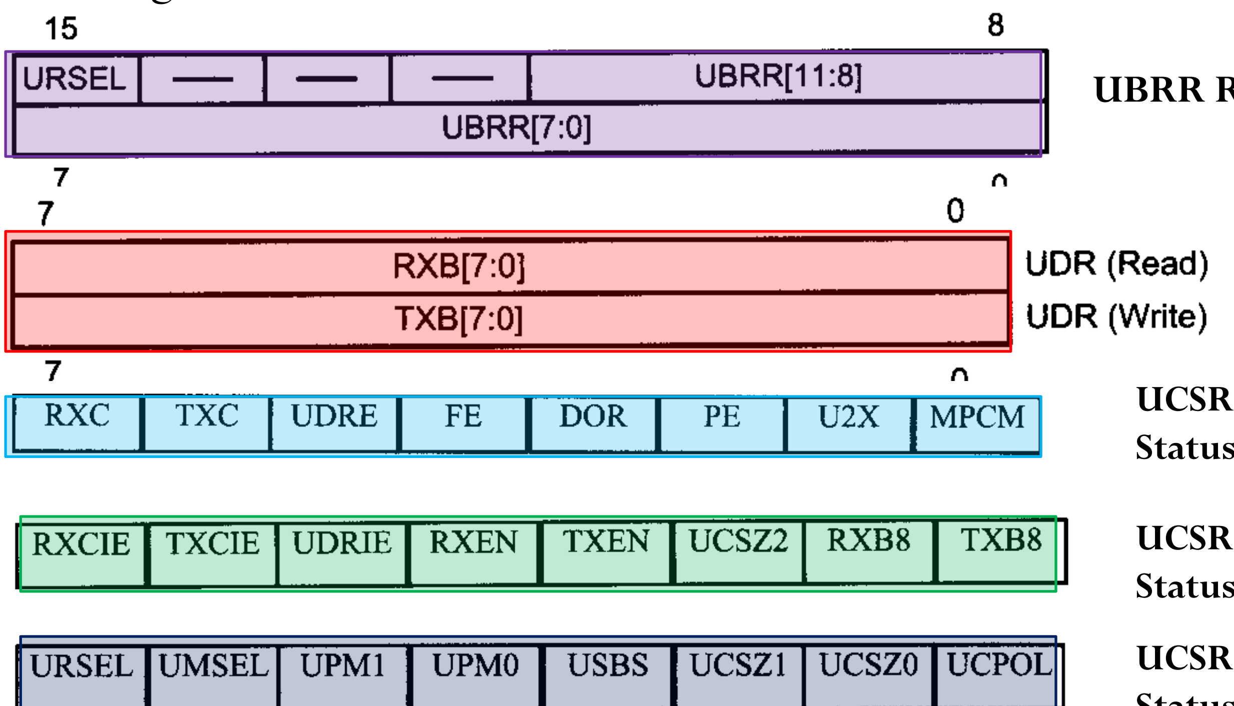
AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

The USART (Universal Synchronous Asynchronous Receiver Transmitter) in the AVR has normal asynchronous, double-speed asynchronous, master synchronous, and slave synchronous mode features.

- The synchronous mode can be used to transfer data between the AVR and external peripherals such as ADC and EEPROMs.
- The asynchronous mode is the one we will use to connect the AVR-based system to the x86 PC serial port for the purpose of full-duplex serial data transfer.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

In the AVR microcontroller five registers are associated with the USART that we deal with in this chapter. They are UDR (USART Data Register), UCSRA, UCSRB, UCSRC (USART Control Status Register), and UBRR (USART Baud Rate Register).



UBRR Register

UDR Register

UCSRA: USART Control and Status Register A

UCSRB: USART Control and Status Register B

UCSRC: USART Control and Status Register C

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

UBRR register and baud rate in the AVR

Because the x86 PCs are so widely used to communicate with AVR-based systems, we will emphasize serial communications of the AVR with the COM port of the x86 PC.

The AVR transfers and receives data serially at many different baud rates. The baud rate in the AVR is programmable. This is done with the help of the 8-bit register called UBRR. For a given crystal frequency, the value loaded into the UBRR decides the baud rate. The relation between the value loaded into UBBR and the Fosc (frequency of oscillator connected to the XTALl and XTAL2 pins) is dictated by the following formula:

Desired Baud Rate = Fosc/ (16(X + 1))

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Assuming that Fosc = 8 MHz, we have the following:

Desired Baud Rate = Fosc/ (16(X + 1)) = 8 MHz/16(X + 1) = 500 kHz/(X + 1)

X = (500 kHz/ Desired Baud Rate) – 1

Table 11-4: UBRR Values for Various Baud Rates (Fosc = 8 MHz, U2X = 0)

Baud Rate	UBRR (Decimal Value)	UBRR (Hex Value)
38400	12	C
19200	25	19
9600	51	33
4800	103	67
2400	207	CF
1200	415	19F

Note: For Fosc = 8 MHz we have UBRR = (500000/BaudRate) – 1

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Another way to understand the UBRR values listed in Table 11-4 is to look at Figure 11-9. The UBRR is connected to a down-counter, which functions as a programmable prescaler to generate baud rate. The system clock (Fosc) is the clock input to the down-counter. The down-counter is loaded with the UBRR value each time it counts down to zero. When the counter reaches zero, a clock is generated. This makes a frequency divider that divides the OSC frequency by UBRR + 1. Then the frequency is divided by 2, 4, and 2.

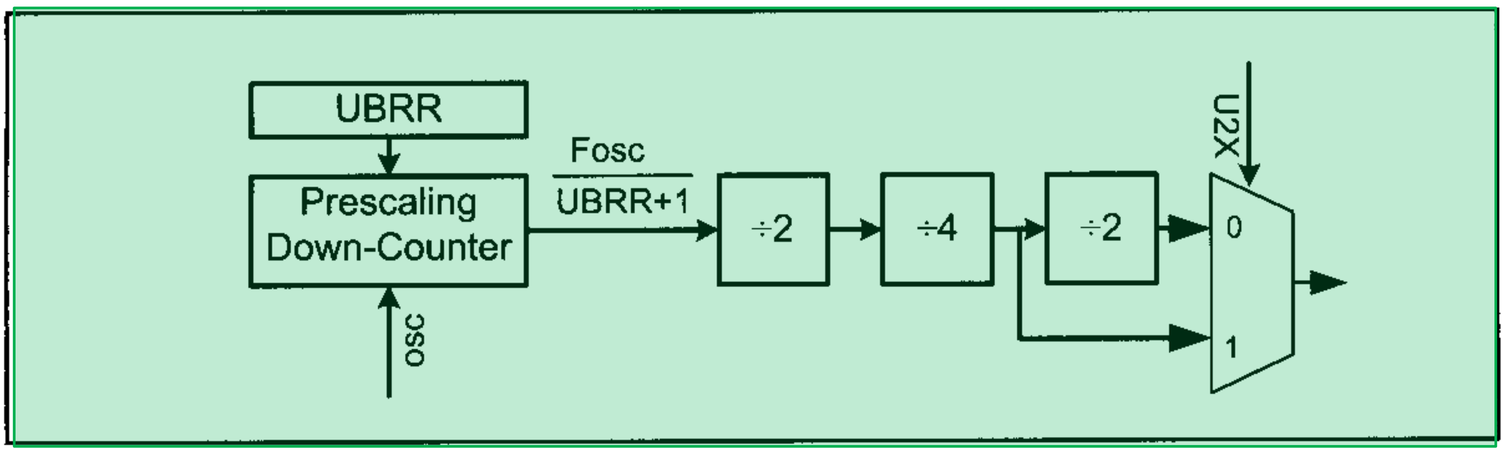


Figure 11-9. Baud Rate Generation Block Diagram

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-1

With Fosc = 8 MHz, find the UBRR value needed to have the following baud rates:

(a) 9600 (b) 4800 (c) 2400 (d) 1200

Solution:

Fosc = 8 MHz => X = (8 MHz/16(Desired Baud Rate)) – 1
=> X = (500 kHz/(Desired Baud Rate)) – 1

(a) (500 kHz/ 9600) – 1 = 52.08 – 1 = 51.08 = 51 = 33 (hex) is loaded into UBRR
(b) (500 kHz/ 4800) – 1 = 104.16 – 1 = 103.16 = 103 = 67 (hex) is loaded into UBRR
(c) (500 kHz/ 2400) – 1 = 208.33 – 1 = 207.33 = 207 = CF (hex) is loaded into UBRR
(d) (500 kHz/ 1200) – 1 = 416.66 – 1 = 415.66 = 415 = 19F (hex) is loaded into UBRR

Notice that dividing the output of the prescaling down-counter by 16 is the default setting upon Reset. We can get a higher baud rate with the same crystal by changing this default setting. This is explained at the end of this section.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

As you see in Figure 11-10, UBRR is a 16-bit register but only 12 bits of it are used to set the USART baud rate. Bit 15 is URSEL and, as we will see in the next section, selects between accessing the UBRRH or the UCSRC register. The other bits are reserved.

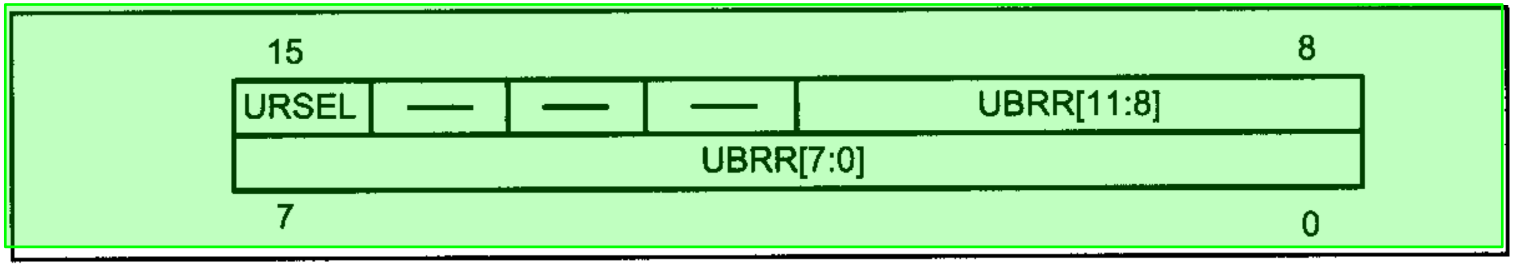


Figure 11-10. UBRR Register

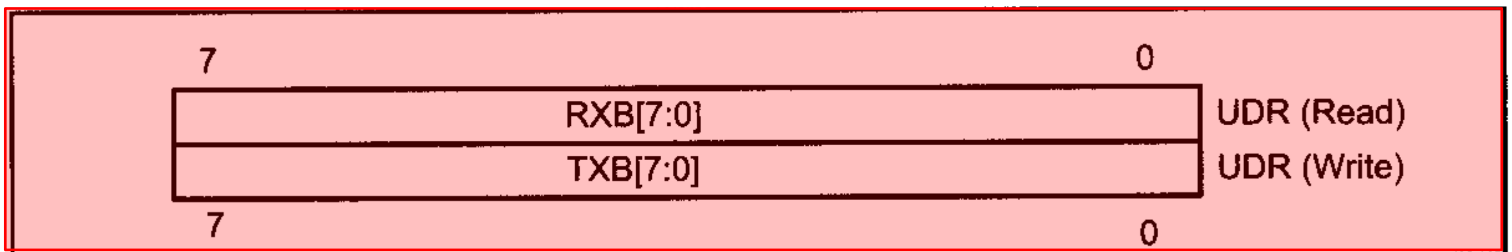


Figure 11-11. UDR Register

mashhoun@iust.ac.ir Iran Univ of Science & Tech

UDR registers and USART data *I/O in the AVR*

In the AVR, to provide a full-duplex serial communication, there are two shift registers referred to as *Transmit Shift Register and Receive Shift Register.*

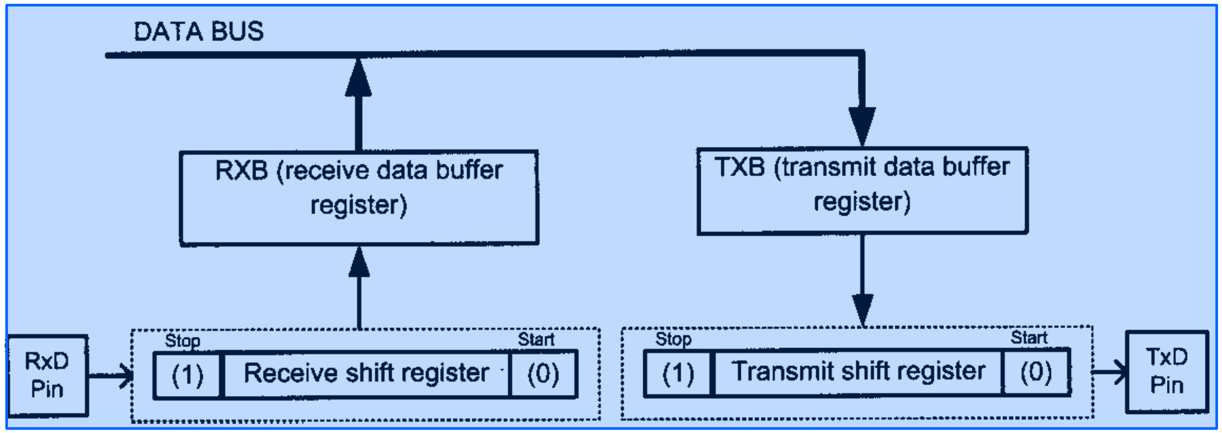


Figure 11-12. Simplified USART Transmit Block Diagram

Each shift register has a buffer that is connected to it directly. These buffers are called ***Transmit Data Buffer Register*** *and* ***Receive Data Buffer Register***. *The* USART Transmit Data Buffer Register and USART Receive Data Buffer Register share the same *I/O address, which is called USART Data Register or UDR.*

UCSR registers and USART configurations in the AVR

UCSRs are 8-bit control registers used for controlling serial communication in the AVR. **There are three USART Control Status Registers** in the AVR. They are UCSRA, UCSRB, and UCSRC.

UCSRA: USART Control and Status RegisterA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

RXC (Bit 7): USART Receive Complete

This flag bit is set when there are new data in the receive buffer that are not read yet. It is cleared when the receive buffer is empty. It also can be used to generate a receive complete interrupt

TXC (Bit 6): USART Transmit Complete

This flag bit is set when the entire frame in the transmit shift register has been transmitted and there are no new data available in the transmit data buffer register (TXB). It can be cleared by writing a one to its bit location.

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

UDRE (Bit 5): USART Data Register Empty

This flag is set when the transmit data buffer is empty and it is ready to receive new data. If this bit is cleared you should not write to UDR because it overrides your last data. The UDRE flag can generate a data register empty interrupt.

FE (Bit 4): Frame Error

This bit is set if a frame error has occurred in receiving the next character in the receive buffer. A frame error is detected when the first stop hit of the next character in the receive buffer is zero.

DOR (Bit 3): Data OverRun

This bit is set if a data overrun is detected. A data overrun occurs when the receive data buffer and receive shift register are full, and a new start bit is detected.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

PE (Bit 2): Parity Error

This bit is set if parity checking was enabled (UPM1 = 1) and the next character in the receive buffer had a parity error when received.

U2X (Bit 1): Double the USART Transmission Speed

Setting this bit will double the transfer rate for asynchronous communication.

MPCM (Bit 0): Multi-processor Communication Mode

This bit enables the multi-processor communication mode. The MPCM feature is not discussed in this book.

Notice that FE, DOR, and PE are valid until the receive buffer (UDR) is read. Always set these bits to zero when writing to UCSRA.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

UCSRB: USART Control and Status Register B

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

RXCIE (Bit 7): Receive Complete Interrupt Enable

To enable the interrupt on the RXC flag in UCSRA you should set this bit to one.

TXCIE (Bit 6):Transmit Complete Interrupt Enable

To enable the interrupt on the TXC flag in UCSRA you should set this bit to one.

UDRIE (Bit 5): USART Data Register Empty Interrupt Enable

To enable the interrupt on the UDRE flag in UCSRA you should set this bit to one.

RXEN (Bit 4): Receive Enable

To enable the USART receiver you should set this bit to one.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

TXEN (Bit 3): Transmit Enable

To enable the USART transmitter you should set this bit to one.

UCSZ2 (Bit 2): Character Size

This bit combined with the UCSZ1:O bits in UCSRC sets the number of data bits (character size) in a frame.

RXBS (Bit 1): Receive data bit 8

This is the ninth data bit of the received character when using serial frames with nine data bits. This bit is not used in this book.

TXBS (Bit 0): Transmit data bit 8

This is the ninth data bit of the transmitted character when using serial frames with nine data bits. This bit is not used in this book.

USART Control and Status Register C

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

URSEL (Bit 7): Register Select

This bit selects to access either the UCSRC or the UBRRH register and will be discussed more in this section.

UMSEL (Bit 6): USART Mode Select

This bit selects to operate in either the asynchronous or synchronous mode of operation.

0 = Asynchronous operation

1 = Synchronous operation

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

UPM1:0 (Bit 5:4): Parity Mode

These bits disable or enable and set the type of parity generation and check.

00 = Disabled 0 l = Reserved 10 = Even Parity 11 = Odd Parity

USBS (Bit 3): Stop Bit Select

This bit selects the number of stop bits to be transmitted.

0 = 1 bit 1 = 2 bits

UCSZ1:0 (Bit 2:l): Character Size

These bits combined with the UCSZ2 bit in UCSRB set the character size in a frame and will be discussed more in this section.

UCPOL (Bit 2): Clock Polarity

This bit is used for synchronous mode only and will not be covered in this section.

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Table 11-5 shows the values of UCSZ2, UCSZl, and UCSZ0 for different character sizes. In this book we use the 8-bit character size because it is the most common in x86 serial communications. If you want to use 9-bit data, you have to use the RXB8 and TXB8 bits in UCSRB as the 9th bit of UDR (USART Data Registers).

Table 11-5: Values of UCSZ2:0 for Different Character Sizes

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	1	1	9

Note: Other values are reserved. Also notice that UCSZ0 and UCSZ1 belong to UCSRC and UCSZ2 belongs to UCSRB

mashhoun@iust.ac.ir

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Because of some technical considerations, the UCSRC register shares the same I/O location as the UBRRH, and therefore some care must be taken when accessing these I/O locations.

When you write to UCSRC or UBRRH, the high bit of the written value (URSEL) controls which of the two registers will be the target of the write operation.

If URSEL is zero during a write operation, the UBRRH value will be updated; otherwise, UCSRC will be updated.

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-2

(a) What are the values of UCSRB and UCSRC needed to configure USART for asynchronous operating mode, 8 data bits (character size), no parity, and 1 stop bit? Enable both receive and transmit.

(b) Write a program for the AVR to set the values of UCSRB and UCSRC for this configuration.

Solution:

(a) RXEN and TXEN have to be 1 to enable receive and transmit. UCSZ2:0 should be 011 for 8-bit data, UMSEL should be 0 for asynchronous operating mode, UPM 1:0 have to be 00 for no parity, and USBS should be 0 for one stop bit.

(b) Refer to the program in the next page

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Solution:

(a) RXEN and TXEN have to be 1 to enable receive and transmit. UCSZ2:0 should be 011 for 8-bit data, UMSEL should be 0 for asynchronous operating mode, UPM 1:0 have to be 00 for no parity, and USBS should be 0 for one stop bit.

(b) Refer to the following program

```
.INCLUDE "M32DEF.INC"
        LDI     R16,(1<<RXEN)|(1<<TXEN)
        OUT     UCSRB,R16
;In the next line URSEL = 1 to access UCSRC. Note that instead
;of using shift operator, you can write "
        LDI     R16,0b10000110"
        LDI     R16,(1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL)
        OUT     UCSRC,R16
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-3

In Example 11-2, set the baud rate to 1200 and write a program for the AVR to set up the values of UCSRB, UCSRC, and UBRR. (Fosc = 8 MHz)

Solution:

```
.INCLUDE "M32DEF.INC"
        LDI     R16,(1<<RXEN)|(1<<TXEN)
        OUT     UCSRB,R16
;In the next line URSEL = 1 to access UCSRC. Note that instead
;of using shift operator, you can write "LDI  R16,0B10000110"
        LDI     R16,(1<<UCSZ1)|<<UCSZ0)|(1<<DRSEL)
        OUT     UCSRC,R16                       ;move R16 to UCSRC
        LDI     R16,0x9F                        ;see Table 11-4
        OUT     UBRRL,R16                       ;1200 baud rate
        LDI     R16,0x1                         ;URSEL= 0 to
        OUT     UBRRH,R16                       ;access UBRRH
```

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

FE and PE flag bits

When the AVR USART receives a byte,

- we can check the parity bit and stop bit. If the parity bit is not correct, the AVR will set PE to one, indicating that an parity error has occurred.

- We can also check the stop bit. As we mentioned before, the stop bit must be one, otherwise the AVR would generate a stop bit error and set the FE flag bit to one, indicating that a stop bit error has occurred.

- We can check these flags to see if the received data is valid and correct.

- Notice that FE and PE are valid until the receive buffer (UDR) is read. So we have to read FE and PE bits before reading UDR.

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Programming the AVR to transfer data serially

In programming the AVR to transfer character bytes serially, the following steps must be taken:

1. The UCSRB register is loaded with the value 08H, enabling the USART transmitter. The transmitter will override normal port operation for the TxD pin when enabled.
2. The UCSRC register is loaded with the value 06H, indicating asynchronous mode with 8-bit data frame, no parity, and one stop bit.
3. The UBRR is loaded with one of the values in Table 11-4 (if Fosc = 8 MHz) to set the baud rate for serial data transfer.
4. The character byte to be transmitted serially is written into the UDR register.
5. Monitor the UDRE bit of the UCSRA register to make sure UDR is ready for the next byte.
6. To transmit the next character, go to Step 4.

Importance of monitoring the UDRE flag

By monitoring the UDRE flag, we make sure that we are not overloading the UDR register. If we write another byte into the UDR register before it is empty, the old byte could be lost before it is transmitted.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-4

Write a program for the AVR to transfer the letter 'G' serially at 9600 baud, continuously. Assume XTAL = 8 MHz.

Solution:

```
.INCLUDE "M32DEF.INC"
        LDI     R16,(1<<TXEN)                       ;enable transmitter
        OUT     UCSRB,R16
        LDI     R16,(1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);8-bit data
        OUT     UCSRC,R16                           ;no parity, 1 stop bit
        LDI     R16,0x33                            ;9600 baud rate
        OUT     UBRRL,R16                           ;for XTAL = 8 MHz
AGAIN:  SBIS    UCSRA,UDRE                          ;is UDR empty
        RJMP    AGAIN                               ;wait more
        LDI     R16,'G'                             ;send 'G'
        OUT     UDR,R16                             ;to UDR
        RJMP    AGAIN                               ;do it again
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-5

Write a program to transmit the message "YES " serially at 9600 baud, 8-bit data, and 1 stop bit. Do this forever.

```
.INCLUDE "M32DEF.INC"
        LDI     R21,HIGH(RAMEND)                    ;initialize high
        OUT     SPH,R21                             ;byte of SP
        LDI     R21,LOW(RAMEND)                     ;initialize low
        OUT     SPL,R21                             ;byte of SP
        LDI     R16,(1<<TXEN)                       ;enable transmitter
        OUT     UCSRB,R16
        LDI     R16,(1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);8-bit data
        OUT     UCSRC,R16                           ;no parity, 1 stop bit
        LDI     R16,0x33                            ;9600 baud rate
        OUT     UBRRL,R16
AGAIN:  LDI     R17,'Y'                             ;move 'Y' to R17
        CALL    TRNSMT                              ;transmit r17 to TxD
        LDI     R17,'E'                             ;move 'E' to R17
        CALL    TRNSMT                              ;transmit r17 to TxD
        LDI     R17,'S'                             ;move 'S' to R17
        CALL    TRNSMT                              ;transmit r17 to TxD
        LDI     R17,' '                             ;move ' ' to R17
        CALL    TRNSMT                              ;transmit space to TxD
        RJMP    AGAIN                               ;do it again
TRNSMT: SBIS    UCSRA,UDRE                          ;is UDR empty?
        RJMP    TRNSMT                              ;wait more
        OUT     UDR,R17                             ;send R17 to UDR
        RET
```

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Programming the AVR to receive data serially

In programming the AVR to receive character bytes serially, the following steps must be taken:

1. The UCSRB register is loaded with the value 10H, enabling the USART receiver. The receiver will override normal port operation for the RxD pin when enabled.
2. The UCSRC register is loaded with the value 06H, indicating asynchronous mode with 8-bit data frame, no parity, and one stop bit.
3. The UBRR is loaded with one of the values in Table 11-4 (if Fosc = 8 MHz) to set the baud rate for serial data transfer.
4. The RXC flag bit of the UCSRA register is monitored for a HIGH to see if an entire character has been received yet.
5. When RXC is raised, the UDR register has the byte. Its contents are moved into a safe place.
6. To receive the next character, go to Step 5.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-6

Program the ATmega32 to receive bytes of data serially and put them on PortB. Set the baud rate at 9600, 8-bit data, and 1 stop bit.

Solution:

```
.INCLUDE "M32DEF.INC"
        LDI     R16,(1<<RXEN)                   ;enable receiver
        OUT     UCSRB,R16
        LDI     R16,(1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);8-bit data
        OUT     UCSRC,R16                       ;no parity, 1 stop bit
        LDI     R16,0x33                        ;9600 baud rate
        OUT     UBRRL,R16
        LDI     R16,0xFF                        ;Port B is output
        OUT     DDRB,R16
RCVE:
        SBIS    UCSRA,RXC                       ;is any byte in UDR?
        RJMP    RCVE                            ;wait more
        IN      R17,UDR                         ;send UDR to R17
        OUT     PORTB,R17                       ;send R17 to PORTB
        RJMP    RCVE                            ;do it again
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-7

Write an AVR program with the following parts:

(a) send the message "YES" once to the PC screen,

(b) get data from switches on Port A and transmit it via the serial port to the PC's screen, and

(c) receive any key press sent by HyperTerminal and put it on LEDs. The programs must do parts (b) and (c) repeatedly.

```
.INCLUDE "M32DEF.INC"
        LDI     R21,0x00
        OUT     DDRA,R21                            ;Port A is input
        LDI     R21,0xFF
        OUT     DDRB, R21                           ;Port B is output
        LDI     R21,HIGH(RAMEND)                    ;initialize high
        OUT     SPH,R21                             ;byte of SP
        LDI     R21,LOW(RAMEND)                     ;initialize low
        OUT     SPL,R21                             ;byte of SP
        LDI     R16,(1<<TXEN)|(1<<RXEN)             ;enable transmitter
        OUT     UCSRB,R16                           ;and receiver
        LDI     R16,(1<<UCSZ1)||(1<<UCSZ0)|(1<<URSEL);8-bit data
        OUT     UCSRC,R16                           ;no parity, 1 stop bit
        LDI     R16,0x33                            ;9600 baud rate
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

```
        OUT     UBRRL,R16

        LDI     R17,'Y'                     ;move 'Y' to R17
        CALL    TRNSMT                      ;transmit r17 to TxD
        LDI     R17,'E'                     ;move 'E' to R17
        CALL    TRNSMT                      ;transmit r17 to TxD
        LDI     R17,'S'                     ;move 'S' to R17
        CALL    TRNSMT                      ;transmit r17 to TxD
AGAIN:
        SBIS    UCSRA,RXC                   ;is there new data?
        RJMP    SKIP_RX                     ;skip receive cmnds
        IN      R17,UDR                     ;move UDR to R17
        OUT     PORTB,R17                   ;move R17 TO PORTB
SKIP_RX:
        SBIS    UCSRA,UDRE                  ;is UDR empty?
        RJMP    SKIP_TX                     ;skip transmit cmnds
        IN      R17,PINA                    ;move Port A to R17
        OUT     UDR,R17                     ;send R17 to UDR
SKIP_TX:
        RJMP    AGAIN                       ;do it again
TRNSMT:
        SBIS    UCSRA,UDRE                  ;is UDR empty?
        RJMP    TRNSMT                      ;wait more
        OUT     UDR,R17                     ;send R17 to UDR
        RET
```

Doubling the baud rate in the AVR

There are two ways to increase the baud rate of data transfer in the AVR:

1. Use a higher-frequency crystal.
2. Change a bit in the UCSRA register, as shown below.

Option 1 is not feasible in many situations because the system crystal is fixed. Therefore, we will explore option 2. There is a software way to double the baud rate of the AVR while the crystal frequency stays the same. This is done with the U2X bit of the UCSRA register.

If we set the U2X bit to HIGH, the third frequency divider (in Figure 11.9) will be bypassed. In the case of XTAL = 8 MHz and U2X bit set to HIGH, we would have:

$$\textbf{Desired Baud Rate} = \text{Fosc} / (8\,(X + 1)) = 8\text{ MHz} / 8\,(X + 1) = 1\text{ MHz} / (X + 1)$$

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

To get the X value for different baud rates we can solve the equation as follows:

$$\mathbf{X = (1\ kHz\ /\ Desired\ Baud\ Rate) - 1}$$

In Table 11-6 you can see values of UBRR in hex and decimal for different baud rates for U2X = 0 and U2X = 1. (XTAL = 8 MHz).

Table 11-6: UBRR Values for Various Baud Rates (XTAL = 8 MHz)

	U2X = 0		U2X = 1	
Baud Rate	**UBRR**	**UBR (HEX)**	**UBRR**	**UBR (HEX)**
38400	12	C	25	19
19200	25	19	51	33
9,600	51	33	103	67
4,800	103	67	207	CF

UBRR = (500 kHz / Baud rate) – 1   *UBRR = (1 kHz / Baud rate) – 1*

11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Baud rate error calculation

In calculating the baud rate we have used the integer number for the UBRR register values because AVR microcontrollers can only use integer values. By dropping the decimal portion of the calculated values we run the risk of introducing error into the baud rate. There are several ways to calculate this error. One way would be to use the following formula.

Error = (Calculated value for the UBRR – Integer part) / Integer part

For example, with XTAL = 8 MHz and U2X = 0 we have the following for the 9600 baud rate:

UBRR value = (500,000/ 9600) – 1 = 52.08 – 1 = 51.08 = 51
Error = (51.08 – 51)/ 51 = 0.16%

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Another way to calculate the error rate is as follows:

Error = (Calculated baud rate – desired baud rate) / desired baud rate

Where the desired baud rate is calculated using X = (Fosc / (16(Desired Baud rate))) - 1, and then the integer X (value loaded into UBRR reg) is used for the calculated baud rate as follows:

Calculated baud rate = Fosc / (16(X + 1))   (for U2X = 0)

For XTAL = 8 MHz and 9600 baud rate, we got X = 51. Therefore, we get the calculated baud rate of 8 MHz/(16(51 + 1)) = 9765. Now the error is calculated as follows:

Error = (9615 – 9600) / 9600 = 0.16%

which is the same as what we got earlier using the other method.

mashhoun@iust.ac.ir

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Table 11-7 provides the error rates for UBRR values of 8 MHz crystal frequencies.

Table 11-7: UBRR Values for Various Baud Rates (XTAL = 8 MHz)

	U2X = 0		U2X = 1	
Baud Rate	**UBRR**	**Error**	**UBRR**	**Error**
38400	12	0.2%	25	0.2%
19200	25	0.2%	51	0.2%
9,600	51	0.2%	103	0.2%
4,800	103	0.2%	207	0.2%
UBRR = (500,000 / Baud rate) – 1			*UBRR = (1,000,000 / Baud rate) – 1*	

In some applications we need very accurate baud rate generation. In these cases we can use a 7.3728 MHz or 11.0592 MHz crystal. As you can see in Table 11-8, the error is 0% if we use a 7.3728 MHz crystal. In the table there are values of UBRR for different baud rates for U2X = 0 and U2X = 1.

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Table 11-8: UBRR Values for Various Baud Rates (XTAL = 7.3728 MHz)

	U2X = 0		U2X = 1	
Baud Rate	**UBRR**	**Error**	**UBRR**	**Error**
38400	11	0%	23	0%
19200	23	0%	47	0%
9,600	47	0%	95	0%
4,800	95	0%	191	0%
UBRR = (460,800 / Baud rate) – 1			*UBRR = (921,600 / Baud rate) – 1*	

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.3 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY

Example 11-8

Assuming XTAL = 10 MHz, calculate the baud rate error for each of the following:

(a) 2400 (b) 9600 (c) 19,200 (d) 57,600

Use the U2X = 0 mode.

Solution:

UBRR Value = (Fosc / 16(baud rate)) - 1 , Fosc = 10 MHz ⇒

(a) UBRR Value = (625,000 / 2400) - 1 = 260.41 - 1 = 259.41 = 259

Error = (259.41 - 259) / 259 = 0.158%

(b) UBRR Value (625,000 / 9600) -1 = 65,104 - 1 = 64.104 = 64

Error = (64.104 - 64)/64 = 0.162%

(c) UBRR Value (625,000 / 19,200) - 1 = 32.55 - 1 = 31.55 = 31

Error = (31.55 - 31) / 31 = 1.77%

(d) UBRR Value (625,000 / 57,600) - 1 = 10.85 - 1 = 9.85 = 9

Error = (9.85 - 9) / 9 = 9.4%

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.4 AVR SERIAL PORT PROGRAMMING IN C

Examples 11-9 through 11-14 show how to program the serial port in C. Connect your AVR trainer to the PC's COM port and use HyperTerminal to test the operation of these examples.

Example 11-9

Write a C function to initialize the USART to work at 9600 baud, 8-bit data, and 1 stop bit. Assume XTAL = 8 MHz.

Solution:

```c
void usart_init (void)
{
        UCSRB = (1<<TXEN);
        UCSRC = (1<< UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
        UBRRL = 0x33;
}
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.4 AVR SERIAL PORT PROGRAMMING IN C

Example 11-10 (C Version of Example 11-4)

Write a C program for the AVR to transfer the letter `G' serially at 9600 baud, continuously. Use 8-bit data and 1 stop bit. Assume XTAL = 8 MHz.

Solution:

```
#include <avr/io.h>                    //standard AVR header void usart init (void)
{
        UCSRB = (1<<TXEN);
        UCSRC = (1<< UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
        UBRRL = 0x33;
}
void usart_send (unsigned char ch)
{                                      //wait until UDR
        while (! (UCSRA & (1<<UDRE))); //is empty
        UDR = ch;                      //transmit 'G'
}
int main (void)
{
        usart init();                  //initialize the USART
        while(1)                       //do forever
            usart send ( 'G');         //transmit 'G' letter
        return 0;
}
```

11.4 AVR SERIAL PORT PROGRAMMING IN C

Example 11-11

Write a program to send the message "The Earth is but One Country. " to the serial port continuously. Using the settings in the last example.

Solution:

```c
#include <avr/io.h>                      //standard AVR header
void usart_init(void)
{
        UCSRB = (1<<TXEN);
        UCSRC = (1<< UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
        UBRRL = 0x33;
}
void usart_send(unsigned char ch)
{
        while (! (UCSRA & (1<<UDRE)));
        UDR = ch;
}
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.4 AVR SERIAL PORT PROGRAMMING IN C

```c
int main(void)
{
        unsigned char str[30]= "The Earth is but One Country. ";
        unsigned char strLenght = 30;
        unsigned char i = 0;
        usart_init();
        while (1)
        {
            usart_send(str(i+4]);
            if (i >= strLenght);
            i = 0;
        }
        return 0;
```


mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.4 AVR SERIAL PORT PROGRAMMING IN C

Example 11-12

Program the AVR in C to receive bytes of data serially and put them on Port A. Set the baud rate at 9600, 8-bit data, and 1 stop bit.

Solution:

```c
#include <avr/io.h>                          //standard AVR header
int main(void)
{
        DDRA = 0xFF;                         //Port A is input
        UCSRB (1<<RXEN);                     //initialize USART
        UCSRC = (1<< UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
        UBRRL = 0x33;
        while (1)
        {
            while (! (UCSRA & (1<<RXC)));    //wait until new data
                PORTA = UDR;
        }
        return 0;
}
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C
11.4 AVR SERIAL PORT PROGRAMMING IN C

Example 11-13

Write an AVR C program to receive a character from the serial port. If it is 'a' -'z' change it to capital letters and transmit it back. Use the settings in the last example.

Solution:

```c
#include <avr/io.h>                    //standard AVR header
void transmit(unsigned char data);
int main(void)
{
            // initialize USART transmitter and receiver
    UCSRB = (1<<TXEN)|(1<<RXEN);
    UCSRC = (1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
    UBRRL = 0x33;
    unsigned char ch;
    while(1)
    {
        while(MUCSRA&(1<<RXC));        //while new data received
            ch = UDR;
        if (ch >= 'a' && ch<='z')
        {
            ch += ('A'-;a');
            while (! (UCSRA & (1<<UDRE)));
            UDR = ch;
        }
    }
```

11.4 AVR SERIAL PORT PROGRAMMING IN C

Example 11-14

In the last five examples, what is the baud rate error?

Solution:

According to Table 11-8, for 9600 baud rate and XTAL = 8 MHz, the baud rate error is about 2%.

Interrupt-based data receive

To program the serial port to receive data using the interrupt method, we need to set HIGH the Receive Complete Interrupt Enable (RXCIE) bit in UCSRB. Program 11-15 shows how to receive data using interrupts.

Example 11-15

Program the ATmega32 to receive bytes of data serially and put them on Port B. Set the baud rate at 9600, 8-bit data, and 1 stop bit. Use Receive Complete Interrupt instead of the polling method.

Solution:

```
.INCLUDE "M32DEF.INC"
.CSEG                                    ;put in code segment
        RJMP    MAIN                     ;jump main after reset
.ORG URXCaddr                            ;int-vector of URXC int.
        RJMP    URXC_INT_HANDLER         ;jump to URXC_INT_HANDLER
.ORG 40                                  ;start main after
                                         ;interrupt vector
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C

11.4 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY AND C USING INTERRUPTS

```
MAIN:     LDI     R16,HIGH(RAMEND)                   ;initialize high byte of
          OUT     SPH,R16                            ;stack pointer
          LDI     R16,LOW(RAMEND)                    ;initialize low byte of
          OUT     SPL,R16                            ;stack pointer
          LDI     R16,(1<<RXEN)|(1<<RXCIE)           ;enable receiver
          OUT     UCSRB,R16                          ;and RXC interrupt
          LDI     R16,(1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);sync,8-bit data
          OUT     UCSRC,R16                          ;no parity, 1 stop bit
          LDI     R16,0x33                           ;9600 baud rate
          OUT     UBRRL,R16
          LDI     R16,0xFF                           ;set Port B as an
          OUT     DDRB,R16                           ;input SEI
          SEI                                        ;enable interrupts
WAIT_HERE:
          RJMP    WAIT_HERE                          ;stay here

URXC_INT_HANDLER:
          IN      R17,UDR                            ;send UDR to R17
          OUT     PORTB,R17                          ;send R17 to PORTB
          RETI
```


mashhoun@iust.ac.ir

Interrupt-based data transmit

To program the serial port to transmit data using the interrupt method, we need to set HIGH the USART Data Register Empty Interrupt Enable (UDRIE) bit in UCSRB. Setting this bit enables the interrupt on the UDRE flag in UCSRA. When the UDR register is ready to accept new data, the UDRE (USART Data Register Empty flag) becomes HIGH. If UDRIE = 1, changing UDRE to one will force the CPU to jump to the interrupt vector.

Example 11-16 shows how to transmit data using interrupts. To transmit data using the interrupt method, there is another source of interrupt; it is Transmit Complete Interrupt. Try to clarify the difference between these two interrupts for yourself. Can you provide some example that the two interrupts can be used interchangeably?

Example 11-16

Write a program for the AVR to transmit the letter `G' serially at 9600 baud, continuously. Assume XTAL = 8 MHz. Use interrupts instead of the polling method.

```
.INCLUDE "M32DEF.INC"
.CSEG                                              ;put in code segment
        RJMP    MAIN                               ;jump main after reset
.ORG UDREaddr                                      ;int. vector of UDRE int.
        RJMP    UDRE_INT_HANDLER                   ;jump to UDRE_INT_HANDLER
.ORG 40                                            ;start main after
;****************************                      ;interrupt vector
MAIN:   LDI     R16,HIGH(RAMEND)                   ;initialize high byte of
        OUT     SPH,R16                            ;stack pointer
        LDI     R16,LOW(RAMEND)                    ;initialize low byte of
        OUT     SPL,R16                            ;stack pointer
        LDI     R16,(1<<TXEN)|(1<<UDRIE)           ;enable transmitter
        OUT     UCSRB,R16                          ;and UDRE interrupt
        LDI     R16,(1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL) ;sync., 8-bit
        OUT     UCSRC,R16                          ;data no parity, 1 stop bit
        LDI     R16,0x33                           ;9600 baud rate
        OUT     UBRRL,R16
        SEI                                        ;enable interrupts
WAIT_HERE:
        RJMP    WAIT_HERE                          ;stay here
;****************************
UDRE_INT_HANDLER:
        LDI     R26,'G'
        OUT     UDR,R26                            ;send 'G'
        RETI                                       ;to UDR
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C

11.4 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY AND C USING INTERRUPTS

Examples 11-17 and 11-18 are the C versions of Examples 11-15 and 11-16, respectively.

Example 11-17

Write a C program to receive bytes of data serially and put them on Port B. Use Receive Complete Interrupt instead of the polling method.

Solution:

```c
#include <avr\io.h>
#include <avr\interrupt.h>
ISR(USART_RXC_vect)
{
        PORTB = UDR;
}
int main(void)
{
        DDRB = 0xFF;                            //make Port B an output
        UCSRB = (1<<RXEN)|(1<<RXCIE);           //enable receive and RXC int.
        UCSRC = (1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
        UBRRL = 0x33;
        sei();                                  //enable interrupts
        while (1);                              //wait forever
        return 0;
}
```

AVR SERIAL PROGRAMMING In ASSEMBLY AND C

11.4 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY AND C USING INTERRUPTS

Example 11-17

Write a C program to receive bytes of data serially and put them on Port B. Use Receive Complete Interrupt instead of the polling method.

Solution:

mashhoun@iust.ac.ir

11.4 AVR SERIAL PORT PROGRAMMING IN ASSEMBLY AND C USING INTERRUPTS

Example 11-18

Write a C program to transmit the letter `G.' serially at 9600 baud, continuously. Assume XTAL = 8 MHz. Use interrupts instead of the polling method.

Solution:

```c
#include <avr\io.h>
#include <avr\interrupt.h>
ISR(USART_UDRE_vect)
{
        UDR = 'G';
}
int main(void)
{
        UCSRB = (1<<TXEN)|(1<<UDRIE);
        UCSRC = (1<<UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
        UBRRL = 0x33;
        sei();                              //enable interrupts
        while(1);                           //wait forever
        return 0;
}
```