Displaying a Numerical Value on HyperTerminal (or equ.)

(Here, only an integer value is discussed)

Objective

Sometimes, the result of a calculations or the value of a variable needs to be displayed on a terminal.

Points to Consider

- A numerical value is stored in MCU memory in binary
- For human interaction the value needs to be displayed in decimal. This requires a conversion from binary to decimal.
- The terminal requires that the decimal character to be displayed be sent as ASCII characters. This requires that the decimal characters (digits) be converted into equivalent ASCII
- If the value is negative, the ASCII character for negative sign (‘-‘) needs to be placed as the leading character.
- If the transmission of the character string is going to be managed by a function capable of transmitting ASCII character strings of varying length, a ‘Null’ marker (0x00) needs to be placed at the end of the converted ASCII strings.

Conversion Binary to (ASCII) Decimal

Example 1

int num = 12345;   // same as num = 0x3039 or num = 0b0011 0000 0011 1001

Let us assume that it is stored in PIC18F2520 data memory (starting) at address 0x100.

Since it is an integer,

- It will be stored in two (byte) locations
- PIC18F uses Little Endean format – low byte at lower address and high byte at higher address
- Thus, Low byte 0b00111001 (0x39) will be stored at 0x100, and
- The High byte 0b00110000 (0x30) will be stored at the next (higher) address 0x101

In order to display it on HyperTerminal in decimal, the above mentioned conversions need to be performed. After the appropriate conversion, the value that should be sent to HyperTerminal will be:

0x31 0x32 0x33 0x34 0x35

Let us assume that after conversion the required string (of ASCII characters) is stored in data memory starting at address 0x200.

Address	Data Byte	ASCII character
0x200	0x31	‘1’
0x201	0x32	‘2’
0x202	0x33	‘3’
0x203	0x34	‘4’
0x204	0x35	‘5’
0x205	..	
0x206	..	

It should be noted that the converted character string does not follow the little Endean format, as the first value that is sent to terminal for display is the most significant character (digit). Thus the converted string needs to be organized in the order, it is going to be displayed on terminal (the leading character first).

Example 2

int num2 = -12345;

Since it is a negative value it will be stored following signed binary convention (2's complement)

// same as num2 = 0xCFC7 or num2 = 0b1100 1111 1100 0111

Let us assume that it is stored in PIC18F2520 data memory (starting) at address 0x110.

Since it is an integer,

- It will be stored in two (byte) locations
- PIC18F uses Little Endean format – low byte at lower address and high byte at higher address
- Thus, Low byte 0b1100 0111 (0xC7) will be stored at 0x110, and
- The High byte 0b1100 1111 (0xCF) will be stored at the next (higher) address 0x111

In order to display it on HyperTerminal in decimal, the above mentioned conversions need to be performed, complete with the negative sign. After the conversion, the value that should be sent to HyperTerminal will be:

0x2D 0x31 0x32 0x33 0x34 0x35

Here 0x2D represents the ASCII for '-'(negative sign)

The following shows the converted value (character string) stored in data memory starting at address 0x210.

Address	Data Byte	ASCII character
0x210	0x2D	'-'
0x211	0x31	'1'
0x212	0x32	'2'
0x213	0x33	'3'
0x214	0x34	'4'
0x215	0x35	'5'
0x216	..	

Adding Null marker (character) at the end of character strings.

Once the starting address of a character string and the length of characters in the string are known, a Null character can be placed after the last character.

Fortunately, C18 provides a library function '**itoa**' that takes care of all conversion requirements, including the negative sign, if so required. The Null character is automatically placed after the last digit position (character).

The following information is taken from Microchip C18 Library Guide

itoa	
Function:	Convert a 16-bit signed integer to a string.
Include:	`stdlib.h`
Prototype:	**`char * itoa(int value, char * `*`string`*` );`**
Arguments:	***value*** A 16-bit signed integer. ***String*** **Pointer to ASCII string that will hold the result. `String` must be long enough to hold the ASCII representation, including the sign character for negative values and a trailing null character.**
Remarks:	This function converts the 16-bit signed integer in the argument **`value`** to a ASCII string representation.
	This function is an MPLAB C18 extension of the ANSI required libraries.
Return Value:	Pointer to the result **`string`**.
File Name:	`itoa.asm`

(Copyright Microchip Technology)

Note: the 'itoa' function works with signed character values as well.

Requirements:

- Inlcude stdlib (#include <stdlib.h>
- Create a RAM character array of sufficient size to hold converted string, including sign and Null character
- Call itoa () function and pass the name of the integer variable and pointer to string that will hold the converted ASCII characters.

Example 3 (Example 1 revisited)

int num1 = 12345;

Here is the final output after the conversion

Address	Data Byte	ASCII character
0x200	0x31	‘1’
0x201	0x32	‘2’
0x202	0x33	‘3’
0x203	0x34	‘4’
0x204	0x35	‘5’
0x205	0x00	Null character
0x206	..	

Since the numerical value is positive, there is no ‘+’ character placed as the start, as is conventionally done. The total number of characters required is now 6.

Example 4 (Example 2 revisited)

int num1 = -12345;

Here is the final output after the conversion

Address	Data Byte	ASCII character
0x200	0x2D	‘-‘
0x201	0x31	‘1’
0x202	0x32	‘2’
0x203	0x33	‘3’
0x204	0x34	‘4’
0x205	0x35	‘5’
0x206	0x00	Null character

Since the numerical value is negative, a ‘-’ character placed as the start. The total number of characters required is now 7.

Procedure

- Get the numerical value
- Type cast it to an integer, if so required
- Convert the numerical value to equivalent text string (consisting of ASCII characters) – use itoa() function
- Use appropriate function to transmit the message – use tx_ram_msg () or equivalent function

Example 5

The following shows a code fragment that converts integer and char (byte) values and transmits those.

```
// SG
// March 28, 2010
// itoa_1.c
// converting integers and a byte values to ASCII

#include <P18F2520.h>
#include <stdlib.h>

// Prototypes
void set_uart(void);
void tx_char(char);
char rx_char(void);
void tx_cr_lf(void);
void tx_ram_msg(char *);
void tx_rom_msg(rom char *);


// ==================== Configuration bits ================
#pragma config WDT = OFF
#pragma config OSC = INTIO67
#pragma config PWRT = ON
#pragma config LVP = OFF
#pragma config MCLRE = ON
#pragma config BOREN = ON
// ======================================================

#define BRG_VAL          12                             // For a baud rate of 19200 at 32 MHz

char msg1[7];                                  // create ram buffers
char msg2[7];
```

```
//          ------------------- Start of main program ----------------------------
void main(void)
{
            int num1 = 12345;                   // assign some values
            int num2 = -12345;
            char cnum1 = 123;
            char cnum2 = -123;

            OSCCON = 0b01100000;
            while (OSCCONbits.IOFS != 1);       // wait for frequency to stabilize

            itoa(num1, msg1);                   // convert integer value and store equivalent string
            tx_ram_msg(msg1);                   // display value of num1 on terminal

            itoa(num2, msg2);
            tx_ram_msg(msg2);                   // display value of num1 on terminal

            itoa(cnum1, msg1);                  // convert integer value and store equivalent string
            tx_ram_msg(msg1);                   // send cnum1 value
            itoa(cnum2, msg2);
            tx_ram_msg(msg2);                   // send cnum2 value

            while(1);
}
// ****************************************************************************

//          ----------------------------------------------------------------------
//                                  set_uart()
//          ----------------------------------------------------------------------
// Set up Uart
void set_uart(void)
{
            TRISCbits.TRISC7 = 1;               // Set Rx pin as an input
            TRISCbits.TRISC6 = 0;               // Set Tx pin as an output

            TXSTA = 0b10100100;                         // initialize USART 8-bit, Async, High Speed
            SPBRG = BRG_VAL;                            // set the baud rate
            RCSTA = 0b10010000;
}
//          -----------------------------------------------------------------------
```

```
//          ------------------------------------------------------------------------
//                                  tx_ram_msg()
//          ------------------------------------------------------------------------
// Send a message contained in RAM
void tx_ram_msg(char * cp)
{
            while(*cp!= 0x00)
                        {
                        tx_char(*cp);     // read char and tranmit
                        cp++;                       // bump up the pointer
                        }
}
//          ------------------------------------------------------------------------

//          ------------------------------------------------------------------------
//                                  tx_rom_msg()
//          ------------------------------------------------------------------------
// Send a message contained in FLASH
void tx_rom_msg(rom char * cp)
{
            while(*cp!= 0x00)
                        {
                        tx_char(*cp);     // read char and tranmit
                        cp++;                       // bump up the pointer
                        }
}
//          ------------------------------------------------------------------------

//          ------------------------------------------------------------------------
//                                  tx_char()
//          ------------------------------------------------------------------------
void tx_char(char c)
{
// Write a byte to the serial port.

            while (PIR1bits.TXIF != 1);           // wait for the flag to be set
            TXREG = c;
}
//          ------------------------------------------------------------------------
//      -----------------------------------------------------------------------
//                      rx_char()
//      -----------------------------------------------------------------------
char rx_char(void)
{
// Get a character from the serial port.
        char c;
        while (PIR1bits.RCIF != 1);     // wait for receive flag

        c = RCREG;                              // read the char
        return (c);
}
//      ------------------------------------------------------------------------

//      -----------------------------------------------------------------------
//                      tx_cr_lf()
//      -----------------------------------------------------------------------
void tx_cr_lf(void)
{
// Send a CR and LF
        tx_char(0x0A);                  // send LF
        tx_char(0x0D);                  // send CR
}
//      -----------------------------------------------------------------------
```

Example 6

Refer to the code given in Example 5. Modify the main program so that current values of variables num1 and num2 (as initialized in the program are displayed as follows:

Screen display

```
The value of num1: 12345

The value of num2: -12345

Program Terminated
```

Solution

The following additional constant strings (character arrays) need to be created to display the messages. These strings can be declared globally (outside of main).

```
rom near *rom_msg1 [] = {“The value of num1: “};

rom near *rom_msg2 [] = {“The value of num2: “};

rom near *rom_msg3 [] = {“Program Terminated”};
```

The new main function will be

```
Void main (void)
{
        OSCCON = 0b01100000;
        while (OSCCONbits.IOFS != 1);   // wait for frequency to stabilize

        tx_rom_msg(rom_msg1);           // show The value of num1: on screen
                                        // cursor is poised at the end of the message
        itoa(num1, msg1);               // convert integer value and store equivalent string
        tx_ram_msg(msg1);               // display value of num1 on screen
        tx_cr_lf();                     // insert  <CR and LF>

        tx_rom_msg(rom_msg2);           // show The value of num2: on screen
        itoa(num2, msg2);
        tx_ram_msg(msg2);               // display value of num1 on screen
        tx_cr_lf();                     // insert  <CR and LF>

        tx_rom_msg(rom_msg3);           // show Program Terminated on screen

        while(1);
}
```

Example 7

Refer to Examples 5 and 6. A global variable **voltage** holds a value of 3550 representing a voltage reading of 3.550 volts (or 3550 mV). Display the reading on the screen with an appropriate information.

Solution

A possible screen output for this reading may be as follows:

(Screen display)

The voltage reading in mV is: 3550

The following additional constant string need to be created to display the leading message. Declaring this string globally

rom near *rom_msg4 [] = {" The voltage reading in mV is: "};

The new main function will be modified as follows

```
Void main (void)
{         .....
          tx_rom_msg(rom_msg4);         // show The voltage reading in mV is: on screen
          itoa(voltage, msg1);
          tx_ram_msg(msg1);             // display value of voltage on screen
          .....
          while(1);
}
```

Example 8

Refer to Example 7. Modify the code so to the information is displayed as shown below:

(Screen display)

The voltage reading is: 3550 mV

Solution

The line of information can be divided in three parts:

- First part will display the leading message **The voltage reading is:**
- The second part will be to display the numerical value **3550**
- The last part will display the trailing character string **mV.** It should be noted that a leading space character is present before **mV**, so that there is a space after the numerical value is displayed information on screen.

Declaring these strings

```
rom near *rom_msg5 [] = {“ The voltage reading  mV is: “};       // there is a trailing space character

rom near *rom_msg6 [] = {“ mV“};                                 // there is a leading space character
```

The new main function will be modified as follows

```
Void main (void)
{         …..
          tx_rom_msg(rom_msg5);        // show The voltage reading in mV is: on screen
          itoa(voltage, msg1);
          tx_ram_msg(msg1);            // display value of voltage on screen

          tx_rom_msg(rom_msg6);        // show mV on screen
          while(1);
}
```