

COMMUNICATION PROTOCOL

📖 INTERFACE

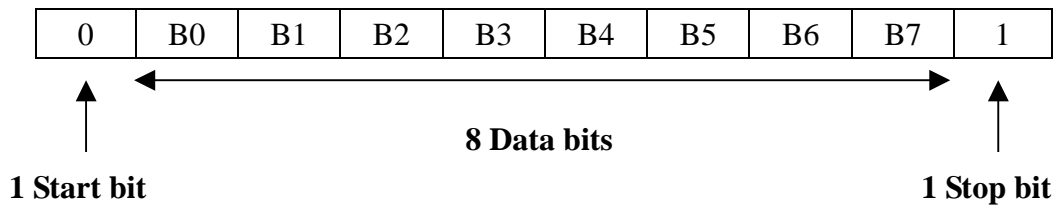
RS-485, RS-232

📖 BAUD RATE

2400, 4800, 9600, 19200, 38400 bps

📖 DATA FRAME

Data Bits = 8 , Parity = None , Start bit = 1 , Stop bit = 1



📖 DATA FORMAT

ModBus Protocol RTU Mode

RTU Request : Read command

| | | | | | | | |
|-----------------------|------------------|--------------------------|---|------------------------|---|------------------------|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 0x03 | | | | | | |
| Station Number | Func-tion | Address (MSB LSB) | | Count (MSB LSB) | | CRC16 (LSB MSB) | |

Station Number: 00H~1FH

Address: 0000H~0100H

Count: Number of Data

CRC16: Cyclical Redundancy Check

RTU Response : Read command

| | | | | | | | | |
|-----------------------|------------------|-------------------|------------------------|---|--------------------------|---|------------------------|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | 0x03 | | | | | | | |
| Station Number | Func-tion | Byte Count | Data1 (MSB LSB) | | Data2.. (MSB LSB) | | CRC16 (LSB MSB) | |

Station Number: 00H~1FH

Address: 0000H~0100H

Byte Count: Number of Data Bytes

CRC16: Cyclical Redundancy Check

RTU Request : Write command

| | | | | | | | |
|-----------------------|------------------|--------------------------|---|-----------------------|---|------------------------|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 0x06 | | | | | | |
| Station Number | Func-tion | Address (MSB LSB) | | Data (MSB LSB) | | CRC16 (LSB MSB) | |

Station Number: 00H~1FH

Address: 0000H~00B6H

CRC16: Cyclical Redundancy Check

RTU Response : Write command

| | | | | | | | |
|-----------------------|------------------|--------------------------|---|-----------------------|---|------------------------|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 0x06 | | | | | | |
| Station Number | Func-tion | Address (MSB LSB) | | Data (MSB LSB) | | CRC16 (LSB MSB) | |

Station Number: 00H~1FH

Address: 0000H~00B6H

CRC16: Cyclical Redundancy Check

 **COMMUNICATION EXAMPLES****RTU Request : Read command** Read PV from station 1

☞ Station number: 01H

☞ Function: 03H

☞ Address MSB: 01H

☞ Address LSB: 00H

☞ Count MSB: 00H

☞ Count LSB: 01H

☞ CRC16 LSB: 85H

☞ CRC16 MSB: F6H

| | | | | | | | |
|-----------------------|------------------|--------------------------|------|------------------------|------|------------------------|------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0x01 | 0x03 | 0x01 | 0x00 | 0x00 | 0x01 | 0x85 | 0xF6 |
| Station Number | Func-tion | Address (MSB LSB) | | Count (MSB LSB) | | CRC16 (LSB MSB) | |

RTU Response : Read command

📁 Response PV 100.0°C from MC-5 Series station 1

☞ Station number: 01H

☞ Function: 03H

☞ Byte count: 02H

☞ Data MSB: 03H

☞ Data LSB: E8H

☞ CRC16 LSB: B8H

☞ CRC16 MSB: FAH

| | | | | | | |
|-----------------------|------------------|-------------------|------------------------|------|------------------------|------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0x01 | 0x03 | 0x02 | 0x03 | 0xE8 | 0xB8 | 0xFA |
| Station Number | Func-tion | Byte Count | Data1 (MSB LSB) | | CRC16 (LSB MSB) | |

RTU Request: Write command

📁 Write SV 100.0°C to MC-5 Series station 1

☞ Station number: 01H

☞ Function: 06H

☞ Address MSB: 00H

☞ Address LSB: 02H

☞ Data MSB: 03H

☞ Data LSB: E8H

☞ CRC16 LSB: 28H

☞ CRC16 MSB: B4H

| | | | | | | | |
|-----------------------|------------------|--------------------------|------|-----------------------|------|------------------------|------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0x01 | 0x06 | 0x00 | 0x02 | 0x03 | 0xE8 | 0x28 | 0xB4 |
| Station Number | Func-tion | Address (MSB LSB) | | Data (MSB LSB) | | CRC16 (LSB MSB) | |

RTU Response: Write command

📁 Response request command from MC-5 Series station 1

☞ Station number: 01H

☞ Function: 06H

☞ Address MSB: 00H

☞ Address LSB: 02H

☞ Data MSB: 03H

☞ Data LSB: E8H

☞ CRC16 LSB: 28H

☞ CRC16 MSB: B4H

| | | | | | | | |
|---------------------------|-----------------------|------------------------------|------|---------------------------|------|----------------------------|------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0x01 | 0x06 | 0x00 | 0x02 | 0x03 | 0xE8 | 0x28 | 0xB4 |
| Station Number | Func- tion | Address (MSB LSB) | | Data (MSB LSB) | | CRC16 (LSB MSB) | |

 ADDRESS INDEX

| PARA | ADDR | PARA | ADDR | PARA | ADDR |
|------|------|------|------|------|------|
| LEvL | 00H | P2 | 20H | AO | 40H |
| LoCK | 01H | i2 | 21H | O2LS | 41H |
| Sv | 02H | d2 | 22H | O2HS | 42H |
| OutL | 03H | Ct2 | 23H | t1SS | 43H |
| At | 04H | HSt2 | 24H | t1On | 44H |
| mAn | 05H | db | 25H | t1ES | 45H |
| AL1S | 06H | SSv | 26H | t1oF | 46H |
| AL1L | 07H | Sout | 27H | t2SS | 47H |
| AL1U | 08H | StmE | 28H | t2On | 48H |
| AL2S | 09H | rUCy | 29H | t2ES | 49H |
| AL2L | 0AH | rPt | 2AH | t2oF | 4AH |
| AL2U | 0BH | StAt | 2BH | inP1 | 4BH |
| AL3S | 0CH | PvSt | 2CH | LoSP | 4CH |
| AL3L | 0DH | wAit | 2DH | HiSP | 4DH |
| AL3U | 0EH | Pid | 2EH | LoAn | 4EH |
| SOAK | 0FH | EndP | 2FH | HiAn | 4FH |
| rAmP | 10H | AL1F | 30H | A1LS | 50H |
| PvoF | 11H | AL1H | 31H | A1HS | 51H |
| Pvrr | 12H | AL1t | 32H | unit | 52H |
| SvoF | 13H | AL1m | 33H | dP | 53H |
| Ct | 14H | AL2F | 34H | FiLt | 54H |
| HbA | 15H | AL2H | 35H | inP2 | 55H |
| LbA | 16H | AL2t | 36H | A2LS | 56H |
| Lbd | 17H | AL2m | 37H | A2HS | 57H |
| rPtm | 18H | AL3F | 38H | | |
| P1 | 19H | AL3H | 39H | bAud | 59H |
| il | 1AH | AL3t | 3AH | Addr | 5AH |
| d1 | 1BH | AL3m | 3BH | LEv1 | 5BH |
| Ct1 | 1CH | Act | 3CH | LEv2 | 5CH |
| HSt1 | 1DH | Outm | 3DH | LEv3 | 5DH |
| AtoF | 1EH | O1LS | 3EH | LvSL | 5EH |
| Ar | 1FH | O1HS | 3FH | L1P1 | 5FH |

| PARA | ADDR | PARA | ADDR | PARA | ADDR |
|-------------|-------------|-------------|-------------|-------------|-------------|
| L1i1 | 60H | tS1 | 80H | 1-12 | A0H |
| L1d1 | 61H | Sv2 | 81H | 1-13 | A1H |
| L1Ar | 62H | tP2 | 82H | 1-14 | A2H |
| L1P2 | 63H | tS2 | 83H | 1-15 | A3H |
| L1i2 | 64H | Sv3 | 84H | 1-16 | A4H |
| L1d2 | 65H | tP3 | 85H | 1-17 | A5H |
| L2P1 | 66H | tS3 | 86H | 1-18 | A6H |
| L2i1 | 67H | Sv4 | 87H | 1-19 | A7H |
| L2d1 | 68H | tP4 | 88H | 1-20 | A8H |
| L2Ar | 69H | tS4 | 89H | 1-21 | A9H |
| L2P2 | 6AH | Sv5 | 8AH | 1-22 | AAH |
| L2i2 | 6BH | tP5 | 8BH | 2-14 | ABH |
| L2d2 | 6CH | tS5 | 8CH | 2-15 | ACH |
| L3P1 | 6DH | Sv6 | 8DH | 2-16 | ADH |
| L3i1 | 6EH | tP6 | 8EH | 2-17 | AEH |
| L3d1 | 6FH | tS6 | 8FH | 3-20 | AFH |
| L3Ar | 70H | Sv7 | 90H | 3-21 | B0H |
| L3P2 | 71H | tP7 | 91H | 3-22 | B1H |
| L3i2 | 72H | tS7 | 92H | 3-23 | B2H |
| L3d2 | 73H | Sv8 | 93H | 3-24 | B3H |
| L4P1 | 74H | tP8 | 94H | 3-25 | B4H |
| L4i1 | 75H | tS8 | 95H | 3-26 | B5H |
| L4d1 | 76H | 1-2 | 96H | 3-27 | B6H |
| L4Ar | 77H | 1-3 | 97H | | |
| L4P2 | 78H | 1-4 | 98H | | |
| L4i2 | 79H | 1-5 | 99H | | |
| L4d2 | 7AH | 1-6 | 9AH | | |
| SEG | 7BH | 1-7 | 9BH | | |
| TimE | 7CH | 1-8 | 9CH | | |
| EndS | 7DH | 1-9 | 9DH | | |
| Sv1 | 7EH | 1-10 | 9EH | | |
| tP1 | 7FH | 1-11 | 9FH | Pv | 100H |

6.2.2 CRC Generation

The Cyclical Redundancy Checking (CRC) field is two bytes, containing a 16-bit binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The device that receives recalculates a CRC during receipt of the message, and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results.

The CRC is started by first preloading a 16-bit register to all 1's. Then a process begins of applying successive 8-bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and stop bits and the parity bit, do not apply to the CRC.

During generation of the CRC, each 8-bit character is exclusive ORed with the register contents. Then the result is shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position. The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value. If the LSB was a 0, no exclusive OR takes place.

This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next 8-bit character is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final content of the register, after all the characters of the message have been applied, is the CRC value.

A procedure for generating a CRC is:

1. Load a 16-bit register with FFFF hex (all 1's). Call this the CRC register.
2. Exclusive OR the first 8-bit byte of the message with the low-order byte of the 16-bit CRC register, putting the result in the CRC register.
3. Shift the CRC register one bit to the right (toward the LSB), zero-filling the MSB. Extract and examine the LSB.
4. (If the LSB was 0): Repeat Step 3 (another shift).
(If the LSB was 1): Exclusive OR the CRC register with the polynomial value 0xA001 (1010 0000 0000 0001).
5. Repeat Steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8-bit byte will have been processed.
6. Repeat Steps 2 through 5 for the next 8-bit byte of the message. Continue doing this until all bytes have been processed.
7. The final content of the CRC register is the CRC value.
8. When the CRC is placed into the message, its upper and lower bytes must be swapped as described below.

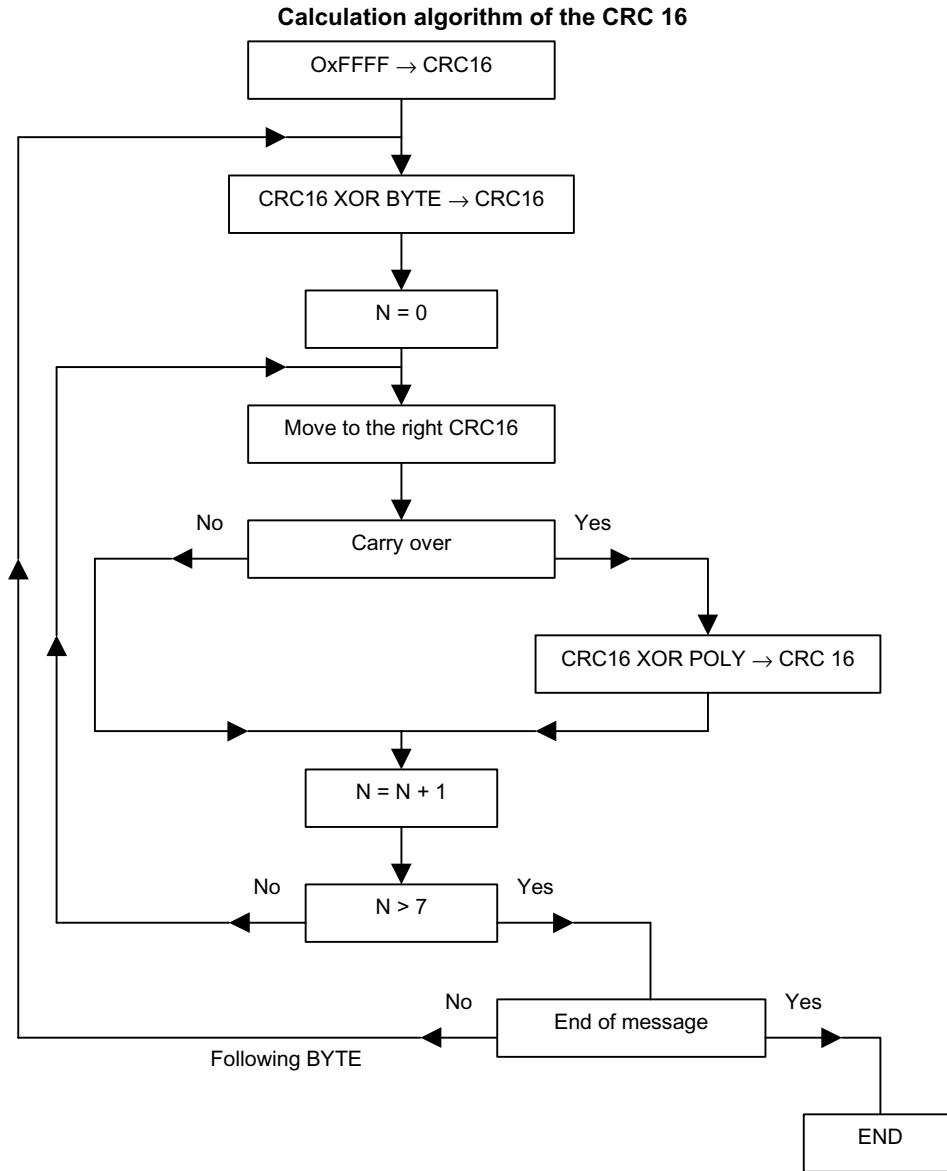
Placing the CRC into the Message

When the 16-bit CRC (two 8-bit bytes) is transmitted in the message, the low-order byte will be transmitted first, followed by the high-order byte.

For example, if the CRC value is 1241 hex (0001 0010 0100 0001):



Figure 30: CRC Byte Sequence



XOR = exclusive or

N = number of information bits

POLY = calculation polynomial of the CRC 16 = 1010 0000 0000 0001

(Generating polynomial = $1 + x_2 + x_{15} + x_{16}$)

In the CRC 16, the 1st byte transmitted is the least significant one.

Example of CRC calculation (frame 02 07)

| | | | | | |
|-----------------------------|--------|-------|------|------|--------|
| CRC register initialization | | 1111 | 1111 | 1111 | 1111 |
| XOR 1st character | | 0000 | 0000 | 0000 | 0000 |
| | | <hr/> | | | |
| | Move 1 | 1111 | 1111 | 1111 | 1101 |
| | | 0111 | 1111 | 1111 | 1110 1 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| Flag to 1, XOR polynomial | | 1101 | 1111 | 1111 | 1111 |
| | Move 2 | 0110 | 1111 | 1111 | 1111 1 |
| Flag to 1, XOR polynomial | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1100 | 1111 | 1111 | 1110 |
| | Move 3 | 0110 | 0111 | 1111 | 1110 0 |
| | Move 4 | 0011 | 0011 | 1111 | 1111 1 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1001 | 0011 | 1111 | 1110 |
| | Move 5 | 0100 | 1001 | 1111 | 1111 0 |
| | Move 6 | 0010 | 0100 | 1111 | 1111 1 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1000 | 0100 | 1111 | 1110 |
| | Move 7 | 0100 | 0010 | 0111 | 1111 0 |
| | Move 8 | 0010 | 0001 | 0011 | 1111 0 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1000 | 0001 | 0011 | 1110 |
| | | 0000 | 0000 | 0000 | 0111 |
| | | <hr/> | | | |
| XOR 2nd character | | 1000 | 0001 | 0011 | 1001 |
| | Move 1 | 0100 | 0000 | 1001 | 1100 1 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1110 | 0000 | 1001 | 1101 |
| | Move 2 | 0111 | 0000 | 0100 | 1110 1 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1101 | 0000 | 0100 | 1111 |
| | Move 3 | 0110 | 1000 | 0010 | 0111 1 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1100 | 1000 | 0010 | 0110 |
| | Move 4 | 0110 | 0100 | 0001 | 0011 0 |
| | Move 5 | 0011 | 0010 | 0000 | 1001 1 |
| | | 1010 | 0000 | 0000 | 0001 |
| | | <hr/> | | | |
| | | 1001 | 0010 | 0000 | 1000 |
| | Move 6 | 0100 | 1001 | 0000 | 0100 0 |
| | Move 7 | 0010 | 0100 | 1000 | 0010 0 |
| | Move 8 | 0001 | 0010 | 0100 | 0001 0 |



The CRC 16 of the frame is then: 4112

Example

An example of a C language function performing CRC generation is shown on the following pages. All of the possible CRC values are preloaded into two arrays, which are simply indexed as the function increments through the message buffer. One array contains all of the 256 possible CRC values for the high byte of the 16-bit CRC field, and the other array contains all of the values for the low byte.

Indexing the CRC in this way provides faster execution than would be achieved by calculating a new CRC value with each new character from the message buffer.

Note: This function performs the swapping of the high/low CRC bytes internally. The bytes are already swapped in the CRC value that is returned from the function.

Therefore the CRC value returned from the function can be directly placed into the message for transmission.

The function takes two arguments:

unsigned char *puchMsg; A pointer to the message buffer containing binary data to be used for generating the CRC
unsigned short usDataLen; The quantity of bytes in the message buffer.

CRC Generation Function

```
unsigned short CRC16 ( puchMsg, usDataLen )                 /* The function returns the CRC as a unsigned short type */  
unsigned char *puchMsg ;                                     /* message to calculate CRC upon                             */  
unsigned short usDataLen ;                                   /* quantity of bytes in message                                 */  
{  
    unsigned char uchCRChi = 0xFF ;                         /* high byte of CRC initialized                                 */  
    unsigned char uchCRCLo = 0xFF ;                         /* low byte of CRC initialized                                    */  
    unsigned ulIndex ;                                        /* will index into CRC lookup table                             */  
  
    while (usDataLen--)  
    {                                                             /* pass through message buffer                                 */  
        ulIndex = uchCRCLo ^ *puchMsgg++ ;                 /* calculate the CRC                                             */  
        uchCRCLo = uchCRChi ^ auchCRChi[ulIndex] ;  
        uchCRChi = auchCRCLo[ulIndex] ;  
    }  
    return (uchCRChi << 8 | uchCRCLo) ;  
}
```

High-Order Byte Table

/* Table of CRC values for high-order byte */

```
static unsigned char auchCRCHI[] = {
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
    0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40
};
```

Low-Order Byte Table

/* Table of CRC values for low-order byte */

```
static char auchCRCLo[] = {
    0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
    0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
    0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
    0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
    0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
    0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
    0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
    0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
    0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
    0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
    0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
    0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
    0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
    0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
    0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
    0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
    0x40
};
```