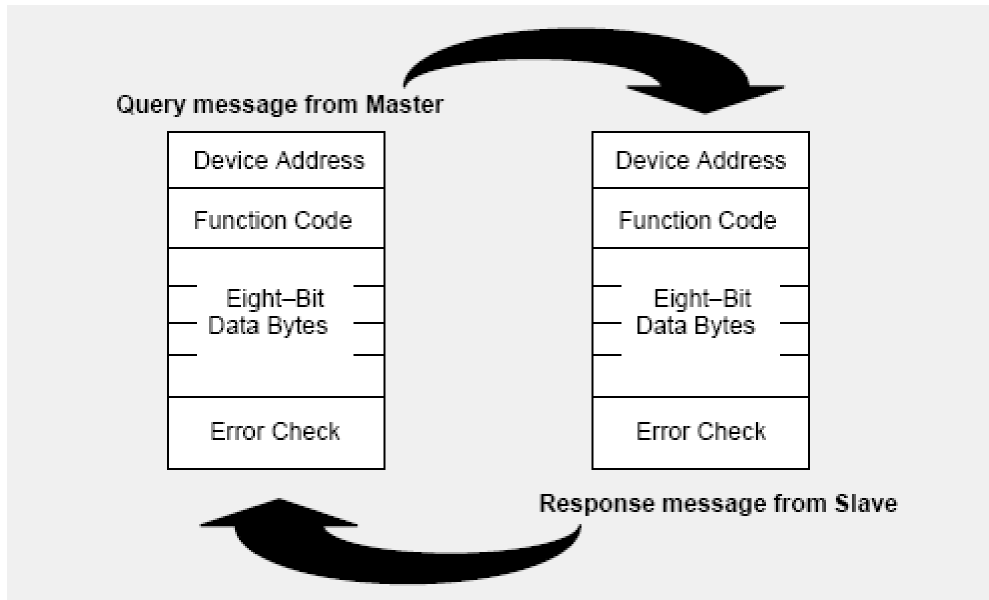
Appendix A. General MODBUS Protocol

본 기기의 "MODBUS Serial Master Driver"가 지원하는 MODBUS 프로토콜 명령어 및 디바이스에 대해 설명 합니다.

At the message level, the Modbus protocol still applies the master–slave principle even though the network communication method is peer–to–peer. If a controller originates a message, it does so as a master device, and expects a response from a slave device. Similarly, when a controller receives a message it constructs a slave response and returns it to the originating controller.



The Query: The function code in the query tells the addressed slave device what kind of action to perform. The data bytes contain any additional information that the slave will need to perform the function. For example, function code 03 will query the slave to read holding registers and respond with their contents. The data field must contain the information telling the slave which register to start at and how many registers to read. The error check field provides a method for the slave to validate the integrity of the message contents.

The Response: If the slave makes a normal response, the function code in the response is an echo of the function code in the query. The data bytes contain the data collected by the slave, such as register values or status. If an error occurs, the function code is modified to indicate that the response is an error response, and the data bytes contain a code that describes the error. The error check field allows the master to confirm that the message contents are valid.

A.1 "0" Device (Coil)

Read Single Coil : 01

MASTER 기기에서 Slave 기기 측(국번:17번)의 "000020–000056 Coil" 데이터를 읽어 오는 예제를 통해 "01"명령어 프레임을 설명 합니다.

■ RTU Mode

(Master → Slave : 요청 프레임)

Comment	Slave 국번	명령어	선두디바이스 H	선두디바이스 L	디바이스점수 H	디바이스점수 L	체크코드 (CRC) L	체크코드 (CRC) H
Hex	11	01	00	13	00	25	—	—

(Slave → Master : 응답 프레임)

Comment	Slave 국번	명령어	데이터 개수(byte)	데이터: Coils 27~20 (L)	Coils 35~28 (–)	Coils 43~36 (–)	Coils 51~44 (–)	Coils 56~52 (H)	체크코드 (CRC) L	체크코드 (CRC) H
Hex	11	01	05	CD	6B	B2	0E	1B	—	—

■ Coils 데이터 상태

Coils	27	26	25	24	23	22	21	20
on/off	1	1	0	0	1	1	0	1
Coils	35	34	33	32	31	30	29	28
on/off	0	1	1	0	1	0	1	1
Coils	43	42	41	40	39	38	37	36
on/off	1	0	1	1	0	0	1	0
Coils	51	50	49	48	47	46	45	44
on/off	0	0	0	0	1	1	1	0
Coils	59	58	57	56	55	54	53	52
on/off	–	–	–	1	1	0	1	1

0: OFF / 1:ON

■ ASCII Mode

(Master → Slave : 요청 프레임)

comment	Header	Slave 국번 H	Slave 국번 L	명령어 H	명령어 L	선두디바이스 H	선두디바이스 –	선두디바이스 –	선두디바이스 L	디바이스점수 H	디바이스점수 –	디바이스점수 –	디바이스점수 L	체크코드 (LRC) L	체크코드 (LRC) H	Tail	Tail
ASCII	:	1	1	0	1	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	31	33	30	30	32	35	—	—	0D	0A

(Slave → Master : 응답 프레임)

Comment	Header	Slave 국번	Slave 국번	명령어	명령어	데이터 개수(byte)	데이터 개수(byte)	데이터: Coils 27~20 H	Coils 27~20 L	Coils 35~28 H	Coils 35~28 L	Coils 43~36 H	Coils 43~36 L	Coils 51~44 H	Coils 51~44 L	Coils 56~52 H	Coils 56~52 L	체크코드 (LRC) L	체크코드 (LRC) H	Tail	Tail
ASCII	:	1	1	0	1	0	5	C	D	6	B	B	2	0	E	1	B			CR	LF
Hex	3A	31	31	30	31	30	35	43	44	36	42	42	32	30	45	31	42	—	—	0D	0A

Force Single Coil : 05

MASTER 기기에서 Slave 기기 측의 Coil 000173 에 FORCE "ON" 하는 예제를 통해 "05"명령어 프레임을 설명 합니다.

■ RTU Mode

(Master → Slave : 요청 프레임)

Comment	Slave 국번	명령어	선두디바이스		Force data		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	05	00	AC	FF	00	—	—

■ Force Data

	High	Low
Force ON	FF_H	00_H
Force OFF	00_H	00_H

(Slave → Master : 응답 프레임)

Comment	Slave 국번	명령어	선두디바이스		Force data		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	05	00	AC	FF	00	—	—

■ ASCII Mode

(Master → Slave : 요청 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				Force data				체크코드 (CRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	5	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	41	43	45	45	30	30	—	—	0D	0A

(Slave → Master : 응답 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				Force data				체크코드 (CRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	5	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	41	43	45	45	30	30	—	—	0D	0A

A.2 “1” Device (Discrete Input)

Read Input Status : 02

MASTER 기기에서 Slave 기기 측(국번:17번)의 “100197–100218 Input” 데이터를 읽어 오는 예제를 통해 “02”명령어 프레임을 설명합니다.

■ RTU Mode

(Master → Slave : 요청 프레임)

Comment	Slave 국번	명령어	선두디바이스		디바이스점수		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	02	00	C4	00	16	—	—

(Slave → Master : 응답 프레임)

Comment	Slave 국번	명령어	데이터 개수(byte)	데이터(Inputs)			체크코드 (CRC)	
				10204~10197	10212~10205	10218~10213	L	H
Hex	11	02	03	AC	DB	35	—	—

■ Coils 데이터 상태

Coils	204	203	202	201	200	199	198	197
on/off	1	0	1	0	1	1	0	0
Coils	212	211	210	209	208	207	206	205
on/off	1	1	0	1	1	0	1	1
Coils	220	219	218	217	216	215	214	213
on/off	–	–	1	1	0	1	0	1

0: OFF / 1:ON

■ ASCII Mode

(Master → Slave : 요청 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				디바이스점수				체크코드 (LRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	2	0	0	C	4	0	0	1	6			CR	LF
Hex	3A	31	31	30	32	30	30	43	34	30	30	31	36	—	—	0D	0A

(Slave → Master : 응답 프레임)

Comment	Header	Slave 국번		명령어		데이터 개수(byte)		데이터(Inputs)						체크코드 (LRC)		Tail	
								10204~10197		10212~10205		10218~10213					
								H	L	H	L	H	L	L	H		
ASCII	:	1	1	0	2	0	3	A	C	D	B	3	5			CR	LF
Hex	3A	31	31	30	31	30	35	41	43	44	42	33	35	—	—	0D	0A

A.3 "3" Device (Input Register)

Read Input Registers : 04

MASTER 기기에서 Slave 기기 측(국번:17번)의 "300009 Register" 데이터를 읽어 오는 예제를 통해 "03"명령어 프레임을 설명 합니다.

■ RTU Mode

(Master → Slave : 요청 프레임)

Comment	Slave 국번	명령어	선두디바이스		디바이스점수 (Word Count)		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	04	00	08	00	01	—	—

(Slave → Master : 응답 프레임)

Comment	Slave 국번	명령어	데이터 개수(byte)	데이터 30009 Register		체크코드 (CRC)	
				H	L	L	H
Hex	11	04	02	00	0A	—	—

■ ASCII Mode

(Master → Slave : 요청 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				디바이스점수 (Word)				체크코드 (LRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	1	0	0	0	8	0	0	0	1			CR	LF
Hex	3A	31	31	30	31	30	30	30	38	30	30	30	31	—	—	0D	0A

(Slave → Master : 응답 프레임)

Comment	Header	Slave 국번		명령어		데이터 개수(byte)		데이터 40108 Register				체크코드 (LRC)		Tail	
								H	–	–	L	L	H		
ASCII	:	1	1	0	4	0	2	0	0	0	A			CR	LF
Hex	3A	31	31	30	31	30	35	30	30	30	41	—	—	0D	0A

A.4 "4" Device (Holding Register)

Read Holding Registers : 03

MASTER 기기에서 Slave 기기 측(국번:17)의 "400108 – 400110 Register" 데이터를 읽어 오는 예제를 통해 "03"명령어 프레임을 설명 합니다.

■ RTU Mode

(Master → Slave : 요청 프레임)

Comment	Slave 국번	명령어	선두디바이스		디바이스점수		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	03	00	6B	00	03	—	—

(Slave → Master : 응답 프레임)

Comment	Slave 국번	명령어	데이터 개수(byte)	데이터						체크코드 (CRC)	
				40108 Register		40109 Register		40110 Register			
				H	L	H	L	H	L	L	H
Hex	11	03	06	02	2B	00	00	00	64	—	—

■ ASCII Mode

(Master → Slave : 요청 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				디바이스점수 (Word)				체크코드 (LRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	1	0	0	1	3	0	0	2	5			CR	LF
Hex	3A	31	31	30	31	30	30	31	33	30	30	32	35	—	—	0D	0A

(Slave → Master : 응답 프레임)

Comment	Header	Slave 국번		명령어		데이터 개수(byte)		데이터												체크코드 (LRC)		Tail	
								40108 Register				40109 Register				40110 Register							
								H	–	–	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	3	0	6	0	2	2	B	0	0	0	0	0	0	6	4			CR	LF
Hex	3A	31	31	30	31	30	35	30	32	32	42	30	30	30	30	30	30	36	34	—	—	0D	0A

Preset Single Register : 06

Slave 기기 측의 400002 Register 에 00 03 (hex) 데이터를 입력 하는 예제를 통해 "06"명령어 프레임을 설명 합니다.

■ RTU Mode

(Master → Slave : 요청 프레임)

Comment	Slave 국번	명령어	선두디바이스		Preset data		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	06	00	01	00	03	—	—

(Slave → Master : 응답 프레임)

Comment	Slave 국번	명령어	선두디바이스		Preset data		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	06	00	01	00	03	—	—

■ ASCII Mode

(Master → Slave : 요청 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				Preset data				체크코드 (LRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	6	0	0	0	1	0	0	0	3			CR	LF
Hex	3A	31	31	30	36	30	30	30	31	30	30	30	33	—	—	0D	0A

(Slave → Master : 응답 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				Preset data				체크코드 (LRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	0	6	0	0	0	1	0	0	0	3			CR	LF
Hex	3A	31	31	30	36	30	30	30	31	30	30	30	33	—	—	0D	0A

Preset Multiple Register : 10

Slave 기기 측의 400002 Register 에 "00 0A (hex)", "01 02 (hex)" 연속한 두 개의 데이터를 입력 하는 예제를 통해 "10"명령어 프레임을 설명 합니다. (Error Code : 90_H)

■ RTU Mode

(Master → Slave : 요청 프레임)

Comment	Slave 국번	명령어	선두디바이스		Quantity of Register (Word Count)		데이터 개수(Byte)	데이터 Register 40002		데이터 Register 40003		체크코드 (CRC)	
			H	L	H	L		H	L	H	L	L	H
Hex	11	10	00	01	00	02	04	00	0A	01	02	—	—

(Slave → Master : 응답 프레임)

Comment	Slave 국번	명령어	선두디바이스		Quantity of Register (Word Count)		체크코드 (CRC)	
			H	L	H	L	L	H
Hex	11	10	00	01	00	02	—	—

■ ASCII Mode

(Master → Slave : 요청 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				Quantity of Register (Word Count)				데이터 개수(Byte)		데이터 Register 40002				데이터 Register 40003			
		H	L	H	L	H	–	–	L	H	–	–	L	–	L	H	–	–	L	H	–	–	L
ASCII	:	1	1	1	0	0	0	0	1	0	0	0	2	0	4	0	0	0	A	0	1	0	2
Hex	3A	31	31	31	30	30	30	41	43	30	30	30	32	30	34	30	30	30	41	30	31	30	32

→ 계속...

	체크코드 (LRC)		Tail	
	L	H		
ASCII			CR	LF
Hex	—	—	0D	0A

(Slave → Master : 응답 프레임)

comment	Header	Slave 국번		명령어		선두디바이스				Quantity of Register (Word Count)				체크코드 (LRC)		Tail	
		H	L	H	L	H	–	–	L	H	–	–	L	L	H		
ASCII	:	1	1	1	0	0	0	0	1	0	0	0	2			CR	LF
Hex	3A	31	31	30	31	30	30	30	31	30	30	30	32	—	—	0D	0A

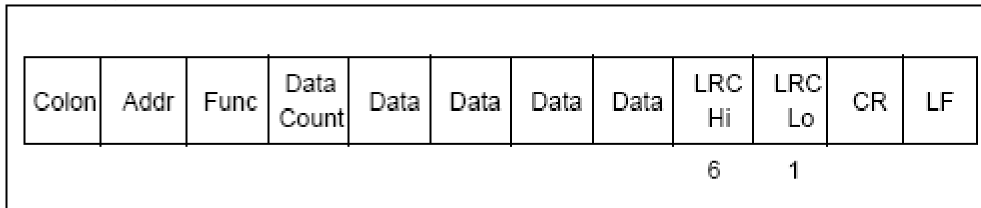
A.5 LRC/CRC Generation

(1) LRC Generation

The Longitudinal Redundancy Check (LRC) field is one byte, containing an 8–bit binary value. The LRC value is calculated by the transmitting device, which appends the LRC to the message. The receiving device recalculates an LRC during receipt of the message, and compares the calculated value to the actual value it received in the LRC field. If the two values are not equal, an error results.
The LRC is calculated by adding together successive 8–bit bytes in the message, discarding any carries, and then two's complementing the result. The LRC is an 8–bit field, therefore each new addition of a character that would result in a value higher than 255 decimal simply 'rolls over' the field's value through zero. Because there is no ninth bit, the carry is discarded automatically.
A procedure for generating an LRC is:

1. Add all bytes in the message, excluding the starting 'colon' and ending CRLF. Add them into an 8–bit field, so that carries will be discarded. 2. Subtract the final field value from FF hex (all 1's), to produce the ones–complement. 3. Add 1 to produce the twos–complement.

– Placing the LRC into the Message
When the 8–bit LRC (2 ASCII characters) is transmitted in the message, the high–order character will be transmitted first, followed by the low–order character.
For example, if the LRC value is 61 hex (0110 0001):



Colon	Addr	Func	Data Count	Data	Data	Data	Data	LRC Hi	LRC Lo	CR	LF
								6	1		

– Example
An example of a C language function performing LRC generation is shown below.
The function takes two arguments:

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

The function returns the LRC as a type unsigned char.

– LRC Generation Function

```
static unsigned char LRC(auchMsg, usDataLen)
unsigned char *auchMsg ;                    /* message to calculate LRC upon */
unsigned short usDataLen ;                  /* quantity of bytes in message */
{
    unsigned char uchLRC = 0 ;              /* LRC char initialized */
    while (usDataLen––)                     /* pass through message buffer */
    uchLRC += *auchMsg++ ;                  /* add buffer byte without carry */
    return ((unsigned char)(–((char)uchLRC))) ;  /* return twos complement */
}
```

(2) CRC Generation

The Cyclical Redundancy Check (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 receiving device 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 contents of the register, after all the characters of the message have been applied, is the CRC value.

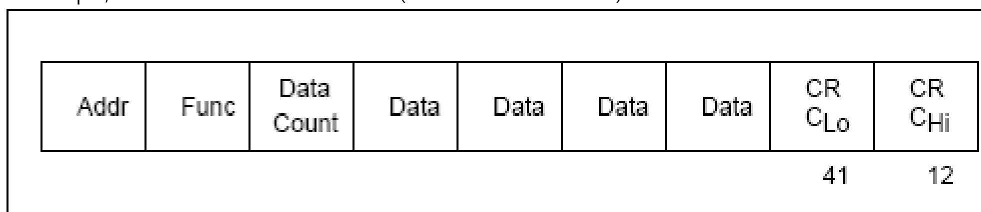
A procedure for generating a CRC is:

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 A001 hex (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 contents 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):



– 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.
```

The function returns the CRC as a type unsigned short.

– CRC Generation Function

```
unsigned short CRC16(puchMsg, usDataLen)
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 uIndex ;                           /* will index into CRC lookup table */
    while (usDataLen—)                          /* pass through message buffer */
    {
        uIndex = uchCRCHi ^ *puchMsgg++ ;       /* calculate the CRC */
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex} ;
        uchCRCLo = auchCRCLo[uIndex] ;
    }
    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,
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, 0x00, 0xC1, 0x81, 0x40, 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
} ;
```