

YModem Batch Analysis – Revision 3 – Xiphod, Jan. 2026

The follow is an annotated log of a YModem Batch transfer between ZOC9 on Windows 11 and TELIX on an MS-DOS '486 based system.

The test files were used, all selected and queued to be sent from ZOC9 to TELIX at 9600 baud.

```
TEST0.DAT 0 byte
TEST1.DAT 33 bytes
TEST2.DAT 132 bytes (slightly above 128-byte block threshold)
TEST3.DAT 528 bytes (slightly beyond 512-byte threshold)
TEST4.DAT 2112 bytes (slightly beyond 2048-byte threshold)
TEST5.DAT 8448 bytes
```

Transfer Setup Configuration

REMOTE

(receiver of the file)

486 PC

MS-DOS 6.22

TELIX

SENDER

(uploader of the batch of file)

i7 x86_64

Windows 11

ZOC 9

^ From log perspective, this node is
both the RECEIVER (RX) and
TRANSMITTER (TX)

These systems were connected via a null modem cable, using 8-N-1 at 9600 baud. A USB adapter was used on the i7 SENDER.

In an RS-232 connection, both nodes perform both TX (transmit) and RX (receive) functions. The logging was running on the faster, more performant system – the i7 laptop. So while the SENDER was the node uploading/sending the file, the REMOTE system also sends status responses (so it is not a RECEIVER-ONLY node).

Color Key

PURPLE	Sent by the REMOTE system, received by the SENDER.
BLUE	Sent by the SENDER, received by the REMOTE.
GREEN	Sent by the SENDER, the REMOTE should compute this same value.
RED	String translation of the hex sequence.
ORANGE	Same as BLUE, but unsure if part of official standard.

Begin Transfer Description via Example

All logged entries are in hex. e.g. #43 == 67 decimal == ASCII 'C'

This transfer describes the batch-sequence upload (via YModem) of the **TESTx.DAT** files described earlier. This intentionally includes a blank zero-byte file, then increasingly larger files thereafter. The "RX>" and "TX>" entries are prefixed with a timestamp for reference only (to see the approximate timing between events), they do not represent any data exchange over the wire.

RS-232 SERIAL PORT BYTE STREAM

DESCRIPTION / EXPLANATION

<20260124151805.949 RX>

#43 --> 'C'

Attention to START/CONTINUE

Optional, I typically see this as "CCC". It is sort of like ++, which switches a Hayes modem into command-mode. This rapid "CCC" should get interpreted as switching over into "file transfer mode" (with YModem). The sender can open file-select dialog, or start to send a pre-selected set of files.

At this point, the receiver waits and monitors for a #01 or #02.

And should be prepared to receive an entire header packet of at least 128+3 bytes.

<20260124151809.340 TX>

#01#00#FF

Header block, #01 == 128 byte block

NOTE: If the filename or its path are long (over 128 bytes), this could start with #02 to indicate a 1024-byte block instead.

```

---START CRC CONTENT [

```

#54#45#53#54#30#2E#44#41#54#00

"TEST0.DAT" <NUL>

[illegible]

```
> pad fill the block
```

```
#00#00#00#00#00#00]---END CRC CONTENT (128 bytes)
```

#00#47

(16-bit CRC)

```
<20260124151809.486 RX>
```

Before responding, the receiver verifies that it can create the requested file. This includes path and filename. If file already exist, it might auto choose an alternate name, or ask to overwrite. It could respond NAK or CAN (cancel), or ACK if all ok. The receiver does not yet know how many files, or what sizes.

#06#43

<ACK> 'C' (agree with the CRC, continue)

<20260124151809.608 TX>

#04

```
<EOT> Nothing left for this stream, all bytes sent.
```

<20260124151809.616 RX>

#06#43

<ACK> 'C' (acknowledge, continue for next file)

Acknowledges that the file has been written (or for 0 bytes, maybe discarded). Asks to


```

                                (#02 at end = number of 128-byte data blocks)
#67#1D                                (16-bit CRC)
<20260124151813.089 RX>
#06#43                                <ACK> 'C' (agree with the CRC, continue)
<20260124151813.212 TX>
#01#01#FE                                Data block 1 (#01 == 128 byte block)
---START CRC CONTENT[
#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31 [16 bytes]
#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [16 bytes]
#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19 [16 bytes]
#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [16 bytes]
#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18 [16 bytes]
#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01 [16 bytes]
#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17 [16 bytes]
#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01 [16 bytes, 128 bytes so far for file]
]---END CRC CONTENT (128 bytes)
#07#C1                                (16-bit CRC)
<20260124151813.356 RX>
#06                                <ACK> (agree with the CRC, no retransmit - continue implied)
<20260124151813.360 TX>
#01#02#FD                                Data block 2 (#01 == 128 byte block)
---START CRC CONTENT[
#01#01#01#19#00#00#00#00#00#00#00#00#00#00#00#00 [4 bytes used, reached 132 byte total expected]
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 > pad fill the block
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
]---END CRC CONTENT (128 bytes)
#46#9E                                (16-bit CRC)
<20260124151813.496 RX>
#06                                <ACK> (agree with the CRC, no retransmit - continue implied)
<20260124151813.496 TX>
#04                                <EOT> Nothing left for this stream, all bytes sent.
<20260124151813.514 RX>
#06#43                                <ACK> 'C' (acknowledge, continue for next file)

<20260124151815.096 TX>
#01#00#FF                                Header block (#01 == 128 byte block)
---START CRC CONTENT[
#54#45#53#54#33#2E#44#41#54#00                                "TEST3.DAT" <NUL>
#35#32#38#20                                "528" <SPACE> (bytes)
#31#35#31#32#33#36#31#37#35#32#32#20                                "15123617522" <SPACE> (time modified, in octal)
#30#20                                "0" <SPACE> (mode)
#30#20                                "0" <SPACE> (serial_number)
#33#20                                "3" <SPACE> (other)
#31#31#30#38#38                                "11088" (??? by ZOC)
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 > pad fill the block
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#05]---END CRC CONTENT (128 bytes)
                                (#05 at end = number of 128-byte data blocks)
#D2#44                                (16-bit CRC)
<20260124151815.241 RX>
#06#43                                <ACK> 'C' (agree with the CRC, continue)

```

```

<20260124151815.361 TX>
#01#01#FE                                     Data block 1 (#01 == 128 byte block)
---START CRC CONTENT[
#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31 [16 bytes]
#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [16 bytes]
#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19 [16 bytes]
#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [16 bytes]
#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18 [16 bytes]
#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01 [16 bytes]
#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17 [16 bytes]
#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01 [16 bytes, 128 bytes so far for file]
]---END CRC CONTENT (128 bytes)
#07#C1                                         (16-bit CRC)
<20260124151815.503 RX>
#06                                             <ACK> (agree with the CRC, no retransmit - continue implied)
<20260124151815.503 TX>
#01#02#FD                                     Data block 2 (#01 == 128 byte block)
---START CRC CONTENT[
#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16 [16 bytes]
#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01 [16 bytes]
#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12 [16 bytes]
#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01 [16 bytes]
#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11 [16 bytes]
#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20 [16 bytes]
#01#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F [16 bytes]
#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30 [16 bytes, 256 bytes so far for file]
]---END CRC CONTENT (128 bytes)
#7D#0C                                         (16-bit CRC)
<20260124151815.652 RX>
#06                                             <ACK> (agree with the CRC, no retransmit - continue implied)
<20260124151815.652 TX>
#01#03#FC                                     Data block 3 (#01 == 128 byte block)
---START CRC CONTENT[
#20#01#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E [16 bytes]
#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39 [16 bytes]
#30#20#01#01#01#01#01#01#19#01#02#04#05#07#0B#0C [16 bytes]
#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38 [16 bytes]
#39#30#20#01#01#01#01#01#01#01#19#01#02#04#05#07#0B [16 bytes]
#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37 [16 bytes]
#38#39#30#20#01#01#01#01#01#01#19#01#02#04#05#07 [16 bytes]
#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36 [16 bytes, 384 bytes so far for file]
]---END CRC CONTENT (128 bytes)
#33#77                                         (16-bit CRC)
<20260124151815.791 RX>
#06                                             <ACK> (agree with the CRC, no retransmit - continue implied)
<20260124151815.791 TX>
#01#04#FB                                     Data block 4 (#01 == 128 byte block)
---START CRC CONTENT[
#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05 [16 bytes]
#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35 [16 bytes]
#36#37#38#39#30#20#01#01#01#01#01#01#19#01#02#04 [16 bytes]
#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34 [16 bytes]
#35#36#37#38#39#30#20#01#01#01#01#01#01#19#01#02 [16 bytes]
#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33 [16 bytes]
#34#35#36#37#38#39#30#20#01#01#01#01#01#01#19#01 [16 bytes]
#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32 [16 bytes, 512 bytes so far for file]
]---END CRC CONTENT (128 bytes)
#65#AE                                         (16-bit CRC)
<20260124151815.948 RX>
#06                                             <ACK> (agree with the CRC, no retransmit - continue implied)
<20260124151815.948 TX>

```

```

#01#05#FA                                     Data block 5 (#01 == 128 byte block)
---START CRC CONTENT [
#33#34#35#36#37#38#39#30#20#01#01#01#01#01#01#19 [16 bytes, 528 bytes of total file size reached]
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 > pad fill the block
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
]---END CRC CONTENT (128 bytes)
#6C#D3                                     (16-bit CRC)
<20260124151816.091 RX>
#06                                     <ACK> (agree with the CRC, no retransmit - continue implied)
<20260124151816.091 TX>
#04                                     <EOT> Nothing left for this stream, all bytes sent.
<20260124151816.091 RX>
#06#43                                     <ACK> 'C' (acknowledge, continue for next file)

```

```

<20260124151817.607 TX>
#01#00#FF                                     Header block (#01 == 128 byte block)
---START CRC CONTENT [
#54#45#53#54#34#2E#44#41#54#00                                     "TEST4.DAT" <NUL>
#32#31#31#32#20                                     "2112" <SPACE> (bytes)
#31#35#31#32#33#36#31#37#35#37#34#20                                     "15123617574" <SPACE> (time modified, octal)
#30#20                                     "0" <SPACE> (mode)
#30#20                                     "0" <SPACE> (serial_number)
#32#20                                     "2" <SPACE> (other)
#31#30#35#36#30                                     "1056" (??? by ZOC)
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 \
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 > pad fill the block
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00#00 /
]---END CRC CONTENT (128 bytes)
                                     (#11 at end = number of 128-byte data blocks)
#FC#43                                     (16-bit CRC)
<20260124151817.755 RX>
#06#43                                     <ACK> 'C' (agree with the CRC, continue)
<20260124151818.405 TX>

```

```

#02#01#FE                                     Data block 1 (#02 == 1024 byte block)
---START CRC CONTENT [
#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [32 bytes]
#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [32 bytes]
#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [32 bytes]
#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [32 bytes]
#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01 [32 bytes]
#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01 [32 bytes]
#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20 [32 bytes]
#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30 [32 bytes]
#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39 [32 bytes]
#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38 [32 bytes]
#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37 [32 bytes]
#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36 [32 bytes]
#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35 [32 bytes]
#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34 [32 bytes]
#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33 [32 bytes]
#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32 [32 bytes]
#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31 [32 bytes]
#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19 [32 bytes]
#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17#18 [32 bytes]
#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16#17 [32 bytes]
#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12#16 [32 bytes]
#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05#07#0B#0C#0E#0F#11#12 [32 bytes]

```


[illegible]

[illegible][illegible]

[illegible][illegible]

```

---START CRC CONTENT[
#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05
#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02#04#05
#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02
#04#05#07#0B#0C#0E#0F#11#12#16#17#18#19#31#32#33#34#35#36#37#38#39#30#20#01#01#01#01#01#19#01#02

```


[illegible]

NOTES

For the header blocks, the number-of-blocks at the end is optional (likely just a debug-aid used by ZOC). Note that it assumes the use of 128-byte blocks. This impacts the computed CRC – a valid client/receiver would match the same CRC computation, but would otherwise ignore this piece of data.

ZOC automatically switch to 128-byte blocks when the bytes remaining goes below a certain threshold (about 512-bytes). This is prudent, especially at low baud rates, so that the last block doesn't have to be a full 1KB length.

Header blocks can switch to 1024-byte block if the filename is long (or contains a long path). It remains ambiguous on what constraints the path has – relative, absolute, upper/lower case, and direction of slashes. It is implied that the convention used must be understood and supported by the receiving node.

The “(??? by ZOC)” parts of the Block Header also seem optional.

If the filesize is large (255 blocks * 1024 bytes, so over 256KB), my understanding is the block increment counter rolls over from FF back to 01 (to avoid being confused with the 00 header block). Likewise, the 3rd byte rolls from 00 to FE. [this is not yet confirmed]

The very last “all empty” block seems to be the way for the SENDER to indicate that no more additional files are coming, and that the batch xfer is concluded.

Sample C Implementation for 16-bit CRC

```
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>

uint16_t calculate_xmodem_crc(const unsigned char* data, size_t len)
{
    uint16_t crc = 0x0000; // Initial value (other CRC might start with 0xFFFF)
    int16_t signed_ref;
    uint8_t i;

    while (len-->0) {
        // XOR with the current byte shifted into the high byte
        printf("%c(%d) %X ", *data, *data, crc);
        printf(" ^ (%X)", (uint16_t)(*data) << 8);
        crc ^= (uint16_t)(*data++) << 8;
        signed_ref = crc;
        printf("= %X(%d)\n", crc, signed_ref);
        for (i = 0; i < 8; ++i) {
            printf("  %d %X --> ", i, crc);
            if (crc & 0x8000) {
                // If the most significant bit is set, shift left
                // and XOR with the polynomial
                crc = (crc << 1) ^ 0x1021; // The XMODEM polynomial
            } else {
                // Otherwise, just shift left
                crc <<= 1;
            }
            signed_ref = crc;
            printf("%X(%d)\n", crc, signed_ref);
        }
    }

    return crc; // The result is used directly without final XOR
}

int main()
{
    uint16_t crc_result;
    int16_t unsigned_ref;
    char test_str[128] = {
        0x01, 0x02, 0x04, 0x05, 0x07, 0x0B, 0x0C, 0x0E, 0x0F, 0x11, 0x12, 0x16, 0x17, 0x18, 0x19, 0x31,
        0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x20, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
        0x19, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };

    crc_result = calculate_xmodem_crc(test_str, 128);
    unsigned_ref = crc_result;
    printf("%X(%d)", crc_result, unsigned_ref);

    return 0;
}
```

Sample BASIC Implementation for 16-bit CRC

Style is X16 BASLOAD, but dialect is very similar to CBM BASIC V2 (just add line numbers).

```
REM XIPHOD - DEC 2025

MAGIC.VALUE% = $1021
REM THESE ONLY WORK WHEN X <= 32767 AND X >= -32768 (SPECIFICALLY THE "OR" AND "AND" KEYWORDS)
REM X XOR Y = (X OR Y) AND NOT (X AND Y)
REM ALTERNATIVE: (X+Y) - 2*(X AND Y)
DEF FN XM(X) = (X + XV) - 2*(X AND XV)

DIM CRC.DATA%(1024) : REM CRC.DATA%(0) CONTAINS LENGTH

GOSUB INIT.CRC.DATA

GOSUB CALC.CRC

GOSUB PRINT.CRC.AS.HEX

END

REM INTERNAL EXAMPLE OF RECEIVING SOME DATA THAT WE WANT TO
REM THEN MAINTAIN A CRC OF. THIS "SIMULATES" A FILE LOAD OR
REM SERIAL DATA RECEIVE.
INIT.CRC.DATA:
  I% = 0
INIT.CRC.NEXT:
  READ A% : REM SIGNED 16-BIT
  IF A% < 0 THEN CRC.DATA%(0) = I% : RETURN
  I% = I% + 1 : CRC.DATA%(I%) = A%
  GOTO INIT.CRC.NEXT

CALC.CRC:
  CRC.RESULT = $0 : REM INITIAL VALUE (XMODEM INIT 0, FOR IBM 3740 - START WITH $FFFF)
  DATA.IDX% = 1 : REM DATA STARTS AT INDEX 1

NEXT.BYTE:
  IF DATA.IDX% > CRC.DATA%(0) THEN RETURN : REM CRC.RESULT COMPUTED

  REM CHR$(80) ENABLES A "DON'T INTERPRET" SIGNAL, SO PETSCII SYMBOL
  REM OF SUBSEQUENT OUTPUT IS SHOWN INSTEAD OF CRT-EFFECT APPLIED.
  PRINT CHR$(80);CHR$(CRC.DATA%(DATA.IDX%));"(";CRC.DATA%(DATA.IDX%);") ";CRC.RESULT;" ";

  XV = CRC.DATA%(DATA.IDX%) * 256 : REM (CRC.DATA AT CURRENT INDEX) << 8
  REM THE FOLLOWING IS NECESSARY SO WE CAN USE "XV" LATER IN THE DO.XOR SUBROUTINE
  IF XV > 32767 THEN XV = XV - 65536
  IF XV < -32768 THEN XV = XV + 65536

  CRC.TEMP = CRC.RESULT
  CRC.TEMP = FN XM(CRC.TEMP) : REM XOR'D WITH XV
  CRC.RESULT = CRC.TEMP
  GOSUB PRINT.CRC.AS.HEX : PRINT : REM PRINT HEX$(CRC.RESULT)

  XV = MAGIC.VALUE% : REM PREPARE THAT WE WILL XOR BY THE XMODEM POLYNOMIAL
  FOR I = 0 TO 7

    REM CAN'T USE "AND" RELIABLY ON SIGNED 16-BIT, BUT WE JUST NEED
    REM TO CHECK IF THE HIGH BIT (MSB) IS SET. SO JUST LOOK AT THE HIGH-BYTE
    REM BY DIVIDING THE VALUE BY 256 (SAME AS SHR >> 8).
    CRC.LOW% = CRC.RESULT / 256
    IF CRC.LOW% AND $80 THEN GOTO HANDLE.HIGH.BIT.CASE

    REM ELSE...
    CRC.RESULT = CRC.RESULT * 2 : REM (CRC << 1)
  NEXT I

  HIGH.BIT.WAS.SET:
    IF CRC.RESULT > 32767 THEN CRC.RESULT = CRC.RESULT - 65536
    IF CRC.RESULT < -32768 THEN CRC.RESULT = CRC.RESULT + 65536
    PRINT " ";I;" ";GOSUB PRINT.CRC.AS.HEX : PRINT : REM PRINT HEX$(CRC.RESULT)
  NEXT I

  DATA.IDX% = DATA.IDX% + 1
  GOTO NEXT.BYTE

HANDLE.HIGH.BIT.CASE:
```


[illegible]

End.