A PROGRAM DOWNLOADER AND OTHER UTILITY SOFTWARE FOR THE DATAC BUS MONITOR UNIT*

N87-22618

Stanley M. Novacki, III
Avionics Engineering Center
Department of Electrical and Computer Engineering
Ohio University
Athens, Ohio

*Published as Ohio University Technical Memorandum 92, July 1984.

A set of programs designed to facilitate software testing on the DATAC Bus Monitor is described.

I. INTRODUCTION

The DATAC Bus Monitor Unit (BusMon) is a Z8000-based microcomputer system designed to receive, interpret, and display selected data items appearing on a DATAC Digital Data Bus. Software for the Bus Monitor Unit is developed on a Tektronix 8550 Microprocessor Development System (MDS). Once a program is written and compiled to object code, it may be tested using the in-circuit emulation and memory-partitioning capabilities of the 8550. The in-circuit emulator allows the MDS to imitate the Z8000 processor, giving the operator extensive control of the test system, while memory partitioning allows the prototype system to utilize memory in the 8550 as though it were part of the target system's memory. This is a great help in lab-testing of the prototype system because of the simplicity of loading and running the test software.

Because of the size of the Tektronix hardware, it is somewhat cumbersome to transport the entire MDS and the prototype system to a field installation simply to test programs in situ. To make on-site testing easier, a series of programs was developed to allow the Z8000 system, running in a standalone mode, to receive program code via its RS232C ports and ports on the host system, which stores the test program in a disk file. Once the program design is finalized, another utility program allows the Z8000 system to send the test software in ASCII form to a ProLog PROM programmer, eliminating the need for an integral PROM programmer on the MDS. These software tools are intended to simplify the development and testing of the data acquisition, reduction, and display routines planned for the DATAC Bus Monitor Unit.

II. IMPLEMENTATION

On the Tektronix 8550 MDS:

Once a program for the Z8000 system has been written and reduced to machine code, it can be transferred to a DOS/50 disk file. DOS/50 is the operating system currently in use on the MDS. The file format consists of lines of ASCII characters in a format called Standard TEKHEX (figs. 1, 2). There are two types of records in a TEKHEX file: data records and the "null" or terminator record. The format for a data record begins with the slash character "/" which denotes the start of a valid record. The slash is followed by 4 hex digits which specify the absolute loading address for the data contained in this record. Next are two hex digits which specify the number of bytes of data contained in the record. The following two digits form a nybble checksum of the load address and the datum count; that is, each digit of the load address and byte count are added together. This number, modulo 256, provides the first checksum. Following the checksum comes the data bytes representing the actual machine code of the program. After the data is the data nybble checksum. As with the first checksum, this is the sum of the individual hex digits of the data, modulo 256. Each record is terminated by an ASCII CR (0D hex). The last record in a TEKHEX file is the "null" record, that is, one with a datum count of zero. An address/byte-count checksum is still generated, usually with a zero value.

A file in this format can be sent to a slave system via RS232C communications ports on the slave and the MDS. The host system will read a record from the TEKHEX file, send it to the prototype system, and wait for a single ASCII token to indicate a good (ACK) or bad (NAK) reception. The 8550 uses the digits "0" as the ACK token and "7" as the NAK symbol. If the prototype system replies with an ACK, the MDS will send the next record, wait for the prompt for that record and so on until the entire file is sent. If the prototype system fails to reconstitute the same checksums sent in the TEKHEX record, it will reply with the NAK token. The 8550 will recognize this as a failed transmission and re-send the same record. The 8550 will continue to send the flagged record until the slave system elects to abort the load operation with an abort message, which appears on the 8550 console and halts the load operation, or the number of retries exceeds a limit set by the host system operator. After all data records are sent, followed by the null record, the 8550 exits from the load routines and and resumes terminal emulation. From this point, the MDS may simply be used as a console device to the prototype and the program is run on the prototype.

On The Bus Monitor Unit:

The loader program for the Z8000-based system (fig. 3; listing 1) is designed to accept serial ASCII data TEKHEX format, convert it to machine code, and store it in the prototype system memory. The processor monitor software for the Bus Monitor Unit provides serial I/O routines which allow it to transmit and receive blocks of ASCII data via serial port A, the default console port, by using the Z8000 System Call instruction, SC #0. The Z8000 loader program begins by sending the ACK token to the host system to indicate that it is ready to receive characters. The input operation of SC #0 returns a string in memory terminated by a carriage return. Once a string has been read, the loader routine scans the input buffer to find the "/" character to define the beginning of the record. If the slash does not occur in the first 80 bytes, it is assumed that part of the record was lost; TEKHEX records do not usually exceed 73 characters including the terminating carriage return. The loader routine sends a NAK token to request a re-send and waits for the next transmission.

Once a record has been received and the slash found, the load address and byte count are converted from ASCII representations to their actual hexadecimal values. This is done by shifting the seven-bit-code for the most-significant-digit of a data byte (i.e, a single ASCII character) to the left by 4 bits, producing a datum of the form "x0" from "zx" in hex. The next character ("zy"), the least-significant digit of the datum being reconstituted, is logically ANDed with 0F hex to zero the high order bits, leaving a "0y" pattern in hex. The loader then ORs the two patterns together, giving a byte of the form "xy". If the character being converted is a numeric, the binary-coded decimal (BCD) representation of the number and the least significant nybble match exactly and the conversion process may proceed. If the hex character is an alphabetic, A-F, some adjustment is needed because the 4 low-order bits of the ASCII characters A through F do not correspond to the hexadecimal values A through F (10 to 15 decimal). In fact, the low-order nybble of ASCII characters A-F has the values 1-6 in BCD. Because of the sequential value, we may correct these characters' codes to correspond to their actual value by adding 09 hex to the character code before the masking process. This addition bumps the low-order bits to a pattern corresponding to the binary representation of their namesakes. With this correction, the characters A-F can then be processed like the numerics 0-9. The alphabetic character adjustment is handled by subroutine TSTNUM and the ASCII-to-hexadecimal conversion is performed by ASCHEX.

Once the load address and byte count are reconstituted, the first checksum is generated. If the computed and transmitted checksums do not agree, a NAK token is sent and the Bus Monitor waits for a new transmission. Otherwise, the program reconstitutes the data stream using ASCHEX, stores it using the load address it generated earlier, and maintains a running checksum. After all data have been stored in the prototype's RAM, the data checksum is reconstituted from the string buffer and compared with the calculated value. If a mismatch occurs, a NAK token is sent and the Bus Monitor waits for the the same record to be retransmitted from the host. Otherwise, it issues an ACK, waits for the next record, and continues the load-and-store process until the entire file has been sent. In the event 5 successive checksum errors occur, the Bus Monitor will abort the load operation by sending an "Abort Load" record, whose message is displayed on the system console (line 198 of listing 1). When the null record is received, the Z8000 returns to the resident monitor via SC #3. No integrity check is performed on the checksum, since a transmission error at this point doesn't affect any data that has been stored.

On the CP/M-based Bus Monitor Console System:

In field experiments, a DEC VT-180 will be used as the host for the program down-loading in addition to being a data display/command input device. The file down-loader (listing 2) is written in the "C" language for the CP/M environment by Manx Software Systems. This loader contains two deviations from the 8550 down-load procedure: one is that the VT-180 itself counts errors and exits on 5 successive errors; the other is that on completion of file transmission, the loader is exited and the VT-180 returns to the CP/M command processor rather than to terminal mode.

Prolog PROM Programmer Support:

This utility can be thought of as a complement to the downloader program for the Z8000. The program (listing 3) sends machine code from the Bus Monitor Unit to a Prolog PROM Programmer equipped with an RS-232C serial port. Two factors complicate this seemingly simple task: one is that the serial port drivers for the PROM programmer expect to see only ASCII data. The other is that the memory for a Z8000 system is organized as 16-bit words. As yet, there are no 16-bit-wide memory devices being manufactured. The designers of these microcomputer systems routinely solve the latter problem by using 2 byte-wide RAMs or ROMs in parallel, one device located at an even byte address, the other at the subsequent odd address. The first "trick" is that we must read alternating memory locations (all even or all odd) addresses when sending data to the programmer.

We will solve the former problem by a procedure which complements the ASCHEX subroutine described earlier. The program produces two ASCII characters from one hexadecimal byte by splitting the byte into high and low-order nybbles and then shifting the high order nybble to the right 4 bit places. For example, byte "xy" becomes two bytes "0x" and "0y". For the hexadecimal digits 0-9, we simply add 30 hex to each byte and we have the ASCII character corresponding to the BCD digit. The hex digits A-F again pose another problem: the ASCII collating sequence has specified that the low-order nybbles of of the codes for the characters A-F are 1-6 decimal, not A-F hex. Further, the high order nybble of those letter digits is a hex 4, not a 3, as is the case for the numeric characters. To handle this case, the program tests the nybble being converted to see if it lies in the range of A-F. If so, an adjuster of 07 hex is added to the nybble first. This corrects the least significant digit to the proper value and puts a 1 in the most significant digit. For example, to turn 0C hex to 43 hex (the ASCII code for the letter "C") the following happens: add 07 to 0C giving 13 hex, then add 30 hex giving 43 hex, giving the desired character code.

The PROLOG utility is usually used with the 8550 running in processor emulation mode in the Bus Monitor system. A data rate of 2400 baud between the test system and the PROM programmer is assumed. The programmer support routine normally resides at address 4000 hex. If this conflicts with the intended load address of the program being sent to the PROM programmer, the support routine can be moved to another memory location. This is possible because the utility program uses only relative addresses, excepting the I/O port addresses which present no relocatability problems. Once the utility program and the application program have been loaded into Bus Monitor memory, the PROM programmer is set to receive the first block (even or odd) of data. Using the 8550 emulator or the Resident Monitor, the following CPU registers are initialized: R10 contains the address of the first byte if the program being sent to the programmer, R11 contains the address of the last byte to be programmed, and R12 contains a 0 if even-numbered bytes are being ROMmed, and a 1 if odd-numbered bytes are being sent to the programmer. Execution begins at the label GO; the "B" serial port on the serial I/O card is used to send data to the PROM programmer, R9 points to the machine code being processed. A pass is complete when R9 is greater than R11, the stop address. For convenience, a breakpoint can be set at GO + 4C hex, so that R12 can be toggled to send the second block of data bytes without having to reset R10 and R11. With R12 readied for the next series of data and the programmer fitted with a new chip, execution may be resumed with a "GO" command, completing the programming process.

III. SUMMARY

The software described in this paper will facilitate the design and testing of software for the DATAC Bus Monitor Unit. By providing a means to simplify program loading, firmware generation, and subsequent testing of programs, we can reduce the overhead involved in software evaluation and use that time more productively in performance, analysis and improvement of current software.

IV. ACKNOWLEDGMENTS

I would like to thank Mr. Kim Constantikes of Carnegie-Mellon University, Mr. John Simmons of Tektronix, Inc., and Mr. Jim Ramsay and Mr. Bill Lynn, both of Kentron International, for their support and patience during the development of these programs.

V. BIBLIOGRAPHY

[1] Kernighan, Brian W. and Ritchie, Dennis M., "The C Programming Language", Prentice-Hall, Inc.; 1978.

[2] Hancock, Les and Krieger, Morris, "The C Primer", McGraw-Hill Book Company, New York; 1982.

[3] "8550 Series B Z8001/Z8002 Assembler Specifics", Tektronix, Inc., Beaverton, OR.

[4] "ProLog Series 90 PROM Programmer Operating Manual", ProLog Corp., Monterey, CA.

[5] "Aztec C II Users' Guide," Manx Software Systems, Shrewsbury, NJ; 1981.

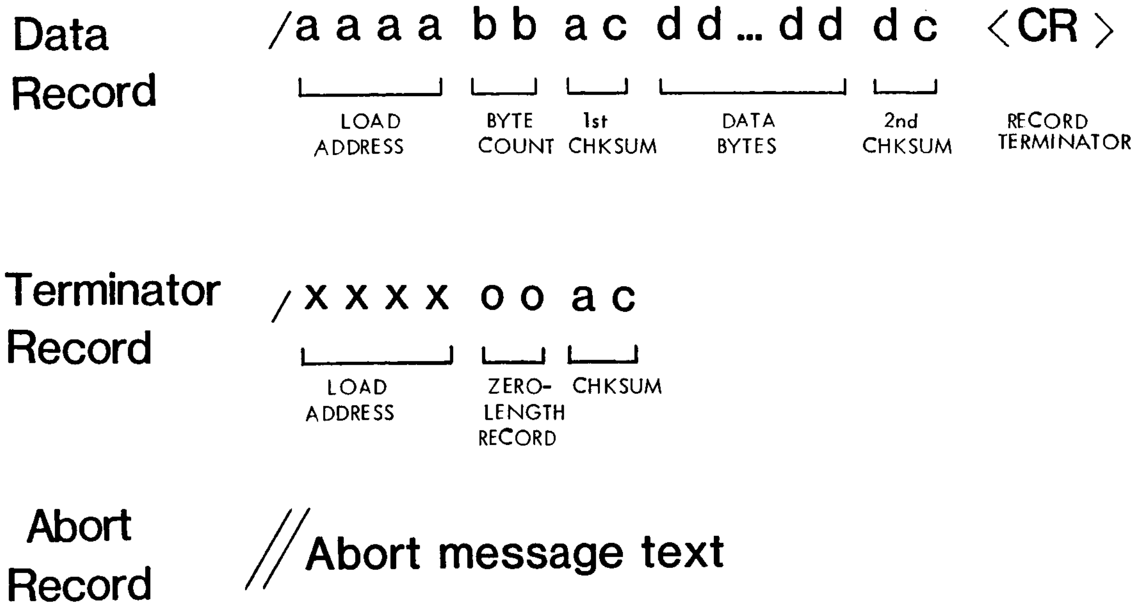

Figure 1. TEKHEX-format records used by BusMon loader program.

```
/1010080A21E462ABBC6E2F3270

/1018030D103FB220

/101B000D
```

Figure 2. Sample TEKHEX file.

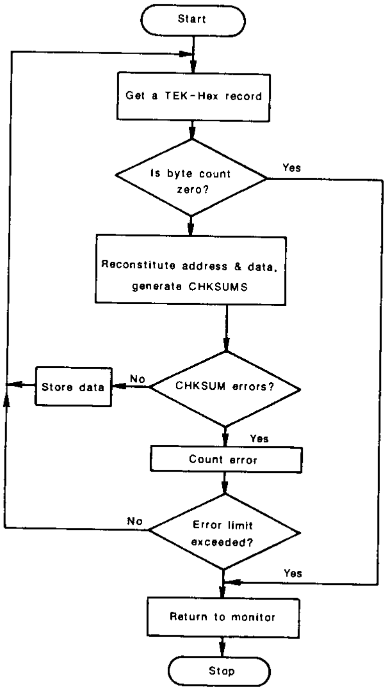

Figure 3. Z8000 loader outline.

APPENDIX A

LISTING 1

```
ASM      Z8001/Z8002
V01.01-01  (8550)                                          01-DEC-83/08:48:48

                           ;       DATAC BUS MONITOR:
                           ;       LOADER FOR Z8000 PROCESSOR INTERFACE TO DATAC SYSTEM
                           ;       AUTHOR:  S.M. NOVACKI      2 SEPT 83
                           ;       REV 22 NOV 83: INCLUDES ERROR HANDLER- EXITS TO MONITOR
                           ;
                           ; MACRO DEFINITIONS HERE:
                                   MACRO   NYBSUM
                                   LDB     RL2,RH2         ;TRANSPOSE HEX DIGITS
                                   SRAB    RL2,#04         ;MAKE HOD THE LOD
                                   AND     R2,#0F0FH       ;MASK OFF HO BITS
                                   ADDB    RH2,RL2         ;ADD NYBBLES W/O CARRY
                           ;RH2 HOLDS NYBBLE CHECKSUM, TRANSFERRED TO RH7
                                   ENDM
                           ;THIS MACRO PERFORMS A TEST FOR CHECKSUM ERRORS, IF >5 THE LOAD IS ABORTED
                                   MACRO   ERRMSG
                                   INC     R13             ;COUNT NEW ERROR OCCURRENCE
                                   CP      R13,#5          ;REACH MAX# OF ERRORS?
                                   JR      UGT,ABRTLD      ;TOO MANY ERRORS- RETURN TO MONITOR
                                   SET     R12,#01         ;SET 'OLD STRING, REPEAT' FLAG
                                   LDB     ACKBUF,#NAK     ;READY BAD TX MSG
                                   JR      NEWSTR          ;REQUEST REPEAT OF MSG AND CLEAR INBUF
                                   ENDM
                           ;
              FE00 R               ORG     0FE00H
                           ;I/O STRING BUFFER DEFINITIONS, MUST BE ORG'D IN RAM
0000FE00        4          ACKBUF BLOCK   4       ;THREE BYTE BUFFER TO HANDSHAKE WITH 8550 DURING FILE TX
0000FE04       50          INBUF  BLOCK   80      ;80 BYTE BUFFER FOR RECEIVING TEKHEX FILES
0000FE54        8          TKHXIN BLOCK   8
0000FE5C        8          TKINAK BLOCK   8       ;I/O FC BLOX (WORKSPACE)
                           ;
               80E R                ORG     080EH
                           ;CONSTANT DEFINITIONS:
                3E         PROMPT EQU     3EH     ;8550 HANDSHAKE PROMPT CHAR
                30         ACK    EQU     30H     ;MSG RECEIVED TOKEN
                37         NAK    EQU     37H     ;MSG NOT RECEIVED TOKEN
                0D         RECEND EQU     0DH     ;CR USED TO TERMINATE PROMPT STRING
                2F         RECMRK EQU     2FH     ;'SLASH' CHAR USED TO DELIMIT TEKHEX RECORDS
                           ;


                           ;
                           ; BEGINNING OF LOADER ROUTINE;
                           ; CONSULT ZMON.DASSY AND .DUMP TO DETERMINE ACTUAL ADDRESSES
                           ; BEGINNING OF ROMABLE ROUTINES, ALL JUMPS RELATIVE, ONLY
                           ; RAM REFERENCES ARE ABSOLUTE FOR DURATION OF LOADER OPERATION
                           ;
0000080E 53544420          TMMSG   ASCII     'STD TEKHEX LOADER '     ; NOTE# OF BYTES IN STRING MUST BE EVEN
         54454B48
         4558204C
         4F414445
         5220
                           ;
00000820 4D08FE00 R        INTCOM  CLR     ACKBUF          ;ONLY 3 OF 4 BYTES USED
00000824 4D08FE02 R                CLR     ACKBUF+2        ;IN HANDSHAKE SEQUENCE
00000828 4C05FE01 R                LDB     ACKBUF+1,#RECEND        ;READY STRING FOR
         0D0D
0000082E 4C05FE02 R                LDB     ACKBUF+2,#PROMPT      ;TEK HANDSHAKE
         3E3E
00000834 8DC8                      CLR     R12             ;FLAG: 0=NEW STRING, 1=REP'T OF LAST STRING
00000836 DF56                      CALR    SETIO           ;SET UP FCB FOR INPUT OPERATIONS
                                                           ;SET UP FCB FOR OUTPUT OPERATIONS
00000838 8CA8              NEWSTR  CLRB    RL2             ;(R2)=0 FOR ZAPPING
0000083A 210A0050                  LD      R10,#80         ;NUMBER OF BYTES TO BE ZAPPED
0000083E 2109FE04 R                LD      R9,#INBUF
00000842 729A0A00          ZAPWRD  LDB     R9(R10),RL2     ;ZERO OUT INBUF (I HOPE..)
00000846 ABA0                      DEC     R10
00000848 EEFC                      JR      NZ,ZAPWRD;
0000084A 2101FE5C R        OUTMSG  LD      R1,#TKINAK      ;SELECT SIGNAL MODE FOR TEK
0000084E 7F00                      SC      #0              ;OUTPUT PROMPT VIA MONITOR ROUTINE
```

```
                                 ;
                                 ;HOPEFULLY WITH A SERIAL LINE DEDICATED TO THE Z8K-TEK INTERFACE
                                 ;THERE WON'T BE ANY JUNK BEFORE THE PROMPT AND THE FIRST HEX RECORD.
                                 ;UNTIL THAT SERIAL LINE IS ESTABLISHED, WE'LL SHARE THE ONE WITH
                                 ;Z8K CONSOLE DEVICE AND PROVIDE FOR GETTING RID OF ANY BAD DATA
                                 ;WE MAY HAPPEN TO READ.  ONCE A SEPARATE SERIAL LINE IS AVAILABLE, WE CAN
                                 ;DISCARD THE 'FIND START-OF-RECORD' ROUTINE
                                 ;
                                 ;IDLE 8550 BEGINS TO TX AFTER THE PROMPT SENT BY OUTMSG
                                 ;
00000850 2101FE54 R              GETSTR LD      R1,#TKHXIN      ;SELECT HEX RECORD READ-MODE
00000854 7F00                           SC      #0              ;GET HEX RECORD AND SAVE IT AT INBUF
00000856 7608FE04 R                     LDA     R8,INBUF        ;SET BASE ADDRESS OF HEX STRING
                                 ;AT THIS POINT, WE SHOULD HAVE ONE COMPLETE TEKHEX RECORD FOR PROCESSING
                                 ;REGISTER ASSIGNMENTS FOR REDUCING THE ASCII STRING
                                 ;      R1:     TRANSIENT AREA FOR CONSOLE I/O
                                 ;      R2:     WORK AREAS FOR CHECKSUM COMPUTATION
                                 ;      R3,R4:  WORK AREAS FOR ASCII HEX CONVERSION
                                 ;      R5:     WORKSPACE FOR FINDING INCOMING ASCII STRING
                                 ;      R6:     CONTAINS THE LOAD ADDRESS OF THE DATA
                                 ;      RH7:    CONTAINS THE NYBBLE CHECKSUMS
                                 ;      RL7:    CONTAINS THE# OF DATA BYTES IN THE RECORD
                                 ;      R8:     POINTER INTO ASCII STRING FOR HEX GENERATION


                                 ;      R13:    CONTAINS CHECKSUM ERROR COUNT
                                 ;
                                 ;FIRST WE'LL SCAN FOR JUNK THAT THE Z8K MAY HAVE READ BEFORE
                                 ;THE 8550 STARTED TX OF THE HEX FILE; THIS SECTION CAN BE
                                 ;DELETED IF WE DEDICATE A SERIAL PORT FOR 8550/Z8K COMMUNICATION
                                 ; 'SLASH' CHR DELIMITS START OF DATA
0000085A 0B08FE54 R              SEEK   CP      R8,#INBUF+80    ;AT THE END OF THE INPUT BUFFER?
0000085E E605                           JR      EQ,STREQ        ;IF SO, THE WHOLE RECORD WAS JUNK,GET ANOTHER
00000860 0C812F2F                       CPB     @R8,#RECMRK     ;SCAN INBUF FOR THE 'SLASH' CHARACTER
00000864 E606                           JR      EQ,TSTSTR       ;FOUNDIT!
00000866 A980                           INC     R8              ;ON TO THE NEXT CHAR
00000868 EEF8                           JR      NE,SEEK         ;HEADER NOT FOUND,TRY AGAIN
0000086A 4D05FE00 R              STREQ  LD      ACKBUF,#NAK     ;BAD TX,ASK FOR REPEAT OF STRING
         0037
00000870 E8E3                           JR      NEWSTR          ;DO THE ASKIN'
                                 ;END OF SOH-SCANNER ROUTINE
                                 ;
                                 ;WE'LL ASSUME THAT A VALID RECORD HAS BEEN READ
                                 ;
00000872 8DC4                    TSTSTR TEST    R12             ;IS THIS NEW OR OLD DATA?
00000874 EE01                           JR      NZ,OLDSTR       ;DON'T RESET ERROR ACCUM IF THIS IS A REPEAT
00000876 8DD8                           CLR     R13             ;ZERO OUT CKSUM ERROR ACCUMULATOR
00000878 DF84                    OLDSTR CALR    CHKTRM          ;SEE IF THE RECORD IS  THE ZERO-LENGTH TERMINATOR
                                 ;IF TERM RECORD IS FOUND, RETURN TO MONITOR
0000087A A980                           INC     R8              ;MOVE POINTER PAST HEADER TO FIRST ASCII CHARACTER
                                 ;(R8)=ADDRESS OF FIRST CHAR IN HEX STRING
0000087C DF97                           CALR    ASCHEX          ;GET 1ST BYTE OF ADDRESS
0000087E A042                           LDB     RH2,RH4         ;1ST BYTE TO CKSUM ACCUMULATOR
                  M                     NYBSUM
00000880 A02A     M                     LDB     RL2,RH2         ;TRANSPOSE HEX DIGITS
00000882 B2A9FCFC M                     SRAB    RL2,#04         ;MAKE HOD THE LOD
00000886 07020F0F M                     AND     R2,#0F0FH       ;MASK OFF HO BITS
0000088A 80A2     M                     ADDB    RH2,RL2         ;ADD NYBBLES W/O CARRY
0000088C A027                           LDB     RH7,RH2         ;TO CHECKSUM ACCUMULATOR
0000088E A046                           LDB     RH6,RH4         ;HOBYTE OF ADDRESS TO R6
00000890 A980                           INC     R8              ;NEXT DIGIT
00000892 DFA2                           CALR    ASCHEX          ;GET SECOND BYTE OF LOAD ADDRESS
00000894 A042                           LDB     RH2,RH4         ;2ND BYTE TO CKSUM ACCUMULATOR
                  M                     NYBSUM
00000896 A02A     M                     LDB     RL2,RH2         ;TRANSPOSE HEX DIGITS
00000898 B2A9FCFC M                     SRAB    RL2,#04         ;MAKE HOD THE LOD
0000089C 07020F0F M                     AND     R2,#0F0FH       ;MASK OFF HO BITS
000008A0 80A2     M                     ADDB    RH2,RL2         ;ADD NYBBLES W/O CARRY
000008A2 8027                           ADDB    RH7,RH2         ;ADD IT TO ACCUM
000008A4 A04E                           LDB     RL6,RH4         ;LOBYTE TO R6; LOAD ADDRESS IS NOW COMPLETE
```

```
                                 ;
000008A6 A980                           INC    R8                 ;ON TO THE BYTE COUNT
000008A8 DFAD                           CALR   ASCHEX             ;GET# OF BYTES IN MSG
000008AA A042                           LDB    RH2,RH4            ;ADD IT TO CHKSUM
                       M                NYBSUM
000008AC A02A          M                LDB    RL2,RH2            ;TRANSPOSE HEX DIGITS
000008AE B2A9FCFC      M                SRAB   RL2,#04            ;MAKE HOD THE LOD
000008B2 07020F0F      M                AND    R2,#0F0FH          ;MASK OFF HO BITS
000008B6 80A2          M                ADDB   RH2,RL2            ;ADD NYBBLES W/O CARRY
000008B8 8027                           ADDB   RH7,RH2            ;ADD RUNNING NYBBLE CHECKSUM
000008BA A04F                           LDB    RL7,RH4            ;SAVE# OF DATA BYTES IN HEX FOR RAM LOAD
000008BC A980                           INC    R8                 ;GET CHAR CNT FROM STRING
000008BE DFAA                           CALR   CHKSUM             ;TEST 1ST BYTE-CHECKSUM
000008C0 E609                           JR     EQ,SUMOK           ;NO PROBS,GO ON
                       M                ERRMSG
000008C2 A9D0          M                INC    R13                ;COUNT NEW ERROR OCCURRENCE
000008C4 0B0D0005      M                CP     R13,#5             ;REACH MAX# OF ERRORS?
000008C8 EB25          M                JR     UGT,ABRTLD         ;TOO MANY ERRORS- RETURN TO MONITOR
000008CA A5C1          M                SET    R12,#01            ;SET 'OLD STRING, REPEAT' FLAG
000008CC 4C05FE00      MR               LDB    ACKBUF,#NAK        ;READY BAD TX MSG
         3737          M
000008D2 E8B2          M                JR     NEWSTR             ;REQUEST REPEAT OF MSG AND CLEAR INBUF
000008D4 8C78                    SUMOK  CLRB   RH7                ;RESET ACCUMULATOR FOR FOR SECOND CHECKSUM
000008D6 A980                    HXLOAD INC    R8                 ;NXT CHR
000008D8 DFC5                           CALR   ASCHEX             ;FORM DATA BYTE
000008DA A042                           LDB    RH2,RH4            ;SENT TO CKSUM ACCUM
                       M                NYBSUM
000008DC A02A          M                LDB    RL2,RH2            ;TRANSPOSE HEX DIGITS
000008DE B2A9FCFC      M                SRAB   RL2,#04            ;MAKE HOD THE LOD
000008E2 07020F0F      M                AND    R2,#0F0FH          ;MASK OFF HO BITS
000008E6 80A2          M                ADDB   RH2,RL2            ;ADD NYBBLES W/O CARRY
000008E8 8027                           ADDB   RH7,RH2            ;ANOTHER DIGIT TO BE SUMMED
000008EA 2E64                           LDB    @R6,RH4            ;STORE MACHINE CODE
000008EC A960                           INC    R6                 ;NEXT RAM LOCATION...
000008EE AAF0                           DECB   RL7                ;ONE LESS BYTE TO STORE
000008F0 EEF2                           JR     NE,HXLOAD          ;UNTIL (RL7)=0, STORE THEM BYTES!
                                 ;RECORD LOAD COMPLETE
000008F2 A980                           INC    R8
000008F4 DFC5                           CALR   CHKSUM             ;PRODUCE AND COMPARE SECOND BYTE-CHECKSUM
000008F6 E609                           JR     EQ,GOODRX          ;NO ERRORS
                       M                ERRMSG
000008F8 A9D0          M                INC    R13                ;COUNT NEW ERROR OCCURRENCE
000008FA 0B0D0005      M                CP     R13,#5             ;REACH MAX# OF ERRORS?
000008FE EB0A          M                JR     UGT,ABRTLD         ;TOO MANY ERRORS- RETURN TO MONITOR
00000900 A5C1          M                SET    R12,#01            ;SET 'OLD STRING, REPEAT' FLAG
00000902 4C05FE00      MR               LDB    ACKBUF,#NAK        ;READY BAD TX MSG
         3737          M
00000908 E897          M                JR     NEWSTR             ;REQUEST REPEAT OF MSG AND CLEAR INBUF
0000090A 4C05FE00      R         GOODRX LDB    ACKBUF,#ACK        ;SET ACKNOWLEGE TOKEN
         3030
00000910 8DC8                           CLR    R12                ;CLEAR FLAG FOR A NEW STRING
00000912 E892                           JR     NEWSTR             ;SEND IT TO THE 8550
00000914 2101091C      R         ABRTLD LD     R1,#MSGBLK         ;READY ERROR MSG FOR TX TO TEK CONSOLE
00000918 7F00                           SC     #0                 ;SEND IT OUT
0000091A 7F03                           SC     #3                 ;RETURN TO Z8000 MONITOR
0000091C 0200                    MSGBLK WORD   0200H              ;TX MODE FOR SC#0
0000091E 0000                           WORD   0000H              ;NOT USED
00000920 0924          R                WORD   ENDMSG             ;ADDRESS OF ERROR MSG
00000922 002B                           WORD   LSTCHR-ENDMSG      ;# OF CHARS IN STRING TO BE TX'D
00000924 2F2F2020                ENDMSG ASCII  '//  ERROR LIMIT EXCEEDED, LOAD IS ABORTED'    ;SELF-EXPLANATORY
         4552524F
         52204C49
         4D495420
         45584345
         45444544
         2C204C4F
         41442049
         53204142
         4F525445
         44
0000094D 0D0A                    CRLF   BYTE   0DH,0AH
0000094F 00                      LSTCHR BYTE   0
```

```
                      ;END OF MAIN ROUTINE; HERE ARE THE SUBROUTINES...
                      ;
                      ;ASCHEX: THE ASCII CHARACTERS WHOSE ADDRESSES ARE (R8) AND (R8)+1 ARE
                      ;CONSOLIDATED TO FORM ONE HEXADECIMAL BYTE. R3 AND R4 ARE THE WORK SPACE WITH
                      ;THE FORMED HEX BYTE LEFT IN RH4.
                      ;
00000950 208C         ASCHEX LDB     RL4,@R8         ;GET 1ST ASCII CHARACTER
00000952 DFD3                CALR    TSTNUM          ;ADJUST ASCII IF CHR IS A-F
00000954 060C0F0F            ANDB    RL4,#0FH        ;MASK OFF ZONE BITS
00000958 B2C90404            SLAB    RL4,#04         ;LSBITS BECOME MSBITS
0000095C A0C4                LDB     RH4,RL4         ;READY FOR NXT DIGIT
0000095E A980                INC     R8              ;NEXT DIGIT
00000960 208C                LDB     RL4,@R8         ;GET IT
00000962 DFDB                CALR    TSTNUM          ;ADJUST ASCII IF CHR IS A-F
00000964 060C0F0F            ANDB    RL4,#0FH        ;PROCESS IT
00000968 84C4                ORB     RH4,RL4         ;FORM COMPLETE BYTE OF DATA
0000096A 9E08                RET                     ;GO HOME
                      ;
                      ;CHKSUM: COMPARE THE COMPUTED CHECKSUM WITH THE VALUE CONTAINED IN THE
                      ;STRING TRANSMITTED FORM THE 8550. RUNNING CHECKSUM IS MAINTAINED IN
                      ;RH7.  THIS ROUTINE CALLS ASCHEX TO READ THE ASCII STRING AND GEN THE
                      ;TX CHECKSUM.
                      ;
0000096C D00F         CHKSUM CALR    ASCHEX          ;GET 1ST BYTE-CHECKSUM
0000096E 8A47                CPB     RH7,RH4         ;COMPARE CALCULATED AND GIVEN CHECKSUMS
00000970 9E08         EXIT   RET                     ;REQUEST ANOTHER TX OF THE STRING IF NEEDED
                      ;
                      ;CHKTRM:  SCANS THE INPUT BUFFER FOR A BYTE COUNT OF ZERO.  USES ASCHEX
                      ;TRANSLATE THE TWO ASCII CHARACTERS OF THE DATA COUNT TO HEX.  IF THE
                      ;BYTE COUNT IS ZERO,THE LOAD IS CONCLUDED WITHOUT A CHECKSUM SCAN AND CONTROL
                      ;IS RETURNED TO THE MONITOR
                      ;ENTER WITH (R8)= LOCATION OF 1ST CHAR IN LOAD ADDRESS
                      ;
00000972 A18A         CHKTRM LD      R10,R8          ;SAVE CURRENT POSITION IN STRING
00000974 A984                INC     R8,#5           ;AIM AT 1ST CHR OF BYTE COUNT
00000976 D014                CALR    ASCHEX          ;FORM BYTE COUNT
00000978 A1A8                LD      R8,R10          ;RECOVER ORIGINAL POINTER
0000097A 8C44                TESTB   RH4             ;IS DATA STRING LENGTH ZERO?
0000097C 9E0E                RET     NE              ;NO, GO BACK AND FINISH PROCESSING
                      ;AT THIS POINT, WHO CARES ABOUT A BIT-ERROR?
0000097E 4D05FE00 R          LD      ACKBUF,#ACK     ;SIGNAL THE END
         0030
00000984 2101FE5C R          LD      R1,#TKINAK      ;READY THE MSG
00000988 7F00                SC      #0              ;SIGNAL TRANSFER END TO HOST COMPUTER
0000098A 7F03                SC      #3              ;LOAD COMPLETED, RETURN TO MONITOR
                      ;SETIO: USED TO RESET FCB FOR SC#0
0000098C 210AFE54 R   SETIO  LD      R10,#TKHXIN     ;DEST FOR MOVE
00000990 210B099E R          LD      R11,#IOBLK      ;SOURCE FOR MOVE
00000994 21090008            LD      R9,#08H         ; # OF WORDS TO MOVE
00000998 BBB109A0     WMOVE  LDIR    @R10,@R11,R9    ;DO IT!
0000099C 9E08                RET                     ;GO HOME..
0000099E 0100         IOBLK  WORD    0100H   ;BLOCK RECEIVE MODE  OF MONITOR CONSOLE HANDLER
000009A0 0000                WORD    0000H   ;NOT USED
000009A2 FE04     R          WORD    INBUF   ;INBUF BUFFER LOCATION
000009A4 0050                WORD    0050H   ;STRING LENGTH IS 80 DECIMAL BYTES TO ALLOW FOR JUNK
                      ;
000009A6 0200                WORD    0200H   ;BLOCK TRANSMIT MODE FOR SYSTEM CALL #0
000009A8 0000                WORD    0000H   ;NOT USED
000009AA FE00     R          WORD    ACKBUF  ;START ADDRESS OF PROMPT-ACKNOWLEGE BUFFER
000009AC 0003                WORD    0003H   ;ONE BYTE FOR PROMPT,ONE FOR ACK-NAK TOKEN,ONE FOR EOL TOKEN
                      ; TSTNUM: CORRECTS ASCII CHARACTERS FROM A TO F TO ALLOW FOR SIMPLE
                      ; MANIPULATION TO HEX FORM
000009AE 0A0C3939     TSTNUM CPB     RL4,#39H
000009B2 E202                JR      LE,ISNUM ;IF 0-9, NO CORRECTION NEEDED
000009B4 000C0909            ADDB    RL4,#9  ;ELSE ADD OFFSET OF 9 TO PRODUCE USEABLE LO NYBBLE
000009B8 9E08         ISNUM  RET             ;BACK TO ASCHEX
                      ;end of loader and subroutines
         00000820            END     INTCOM; PROGRAM START ADDRESS FOR ASSEMBLER
```

```
ASM     Z8001/Z8002 SYMBOL TABLE
V01.01-01  (8550)                                 01-DEC-83/08:48:48

Scalars
ACK--------------00000030   NAK--------------00000037   PROMPT-----------0000003E   RECEND-----------0000000D
RECMRK-----------0000002F

Strings & Macros

ERRMSG------------------ M   NYBSUM------------------ M

Section = %BMLLOAD, Inpage Relocatable, Size = 0000FE64

ABRTLD-----------00000914   ACKBUF-----------0000FE00   ASCHEX-----------00000950   CHKSUM-----------0000096C
CHKTRM-----------00000972   CRLF-------------0000094D   ENDMSG-----------00000924   EXIT-------------00000970
GETSTR-----------00000850   GOODRX-----------0000090A   HXLOAD-----------00000806   INBUF------------0000FE04
INTCOM-----------00000820   IOBLK------------0000099E   ISNUM------------000009B8   LSTCHR-----------0000094F
MSGBLK-----------0000091C   NEWSTR-----------00000838   OLDSTR-----------00000878   OUTMSG-----------0000084A
SEEK-------------0000085A   SETIO------------0000098C   STREQ------------0000086A   SUMOK------------000008D4
TKHXIN-----------0000FE54   TKINAK-----------0000FE5C   TMMSG------------0000080E   TSTNUM-----------000009AE
TSTSTR-----------00000872   WMOVE------------00000998   ZAPWRD-----------00000842

230 Lines Read
268 Lines Processed
0 Errors
```

APPENDIX B

LISTING 2

```
/*
-
- BUSLODR.C:     8550 DOWNLOAD EMULATOR FOR DEC VT-180
-                WRITTEN IN AZTEC C FOR THE CP/M ENVIRONMENT
-
- AUTHOR: S. NOVACKI
- CREATED: JULY, 1984
-
*/

#include "b:stdio.h"       /* standard I/O used for file handling   */
#define ACK '0'            /* definitions of: the ACK token         */
#define NAK '7'            /*                 the NAK token         */
#define CR 13              /*                 end-of-line flag      */
#define TX_RDY 0x01        /* UART transmitter ready flag bit       */
#define RX_RDY 0x02        /* receiver ready bit                    */
#define COMM_DATA 0x58     /* UART data register port number        */
#define COMM_STAT 0x59     /* status register port number           */

/*
infile:
       pointer for source file (from disk)
numchar:
       subscript for reading characters from disk file into buffer vector
outptr:
       subscript for sending buffer characters to UART
argc:
       command line argument count, used by "C" console processor
errcount:
       number of consecutive reception errors
iolinebuffer:
       array used in moving characters from disk file using standard
       I/O to UART using system-specific hardware
reply:
       token read from BusMon system to indicate quality of message
tx_stat, rx_stat:
       UART register statuses used during character-send procedure

*/

FILE *infile,*fopen();
int numchar,outptr,argc,errcount = 0;
char iolinebuffer[80],reply,tx_stat,rx_stat;

/***************************************************************************/

main(argc,argv)
char *argv[];

{

/*
```

```
   open disk file to be sent to the BUSMON system
   if a NULL is returned, OPEN has failed, exit to CP/M
*/
if ((infile = fopen(*++argv, "r")) == NULL) {
       printf("open failure on file %s\n", *argv); exit(99);
       }

       while ()   {      /* a DO-ALWAYS loop, a la BASIC                    */
get_reply();     /* get first ACK to commence file transmission         */
get_line();      /* read a line from the TEKHEX disk file               */

#ASM
                 /* after reading a line from the disk file, kill IRQs for */
    DI           /* polled serial I/O for both the record output        */
                 /* and the REPLY input                                 */
#ENDASM

tx_line();       /* send record to waiting BusMon unit                  */
get_reply();
errcount == 0;   /* zero error count for each record being sent         */
while (reply != ACK)  {  /* if NAK is received:                         */
      retrans_record();
      get_reply();
      }
}

#ASM

    EI             /* bring back IRQs for BDOS/BIOS disk I/O routines */

#ENDASM

}

/*************************************************************************/

get_line()
/* function to read <=80 character from the TEKHEX disk file             */
{
for (numchar =1; numchar <= 80; ++numchar) {   /* for numchar = 1 to 80    */
    iolinebuffer[numchar] = getc(infile);      /* read from infile to
                                                  the line buffer          */
    if (iolinebuffer[numchar] == EOF) {        /*have we reached the end?  */
       fclose(infile);                  /* if so, close the disk file      */
       exit(0);                         /* and back to CP/M...             */
       }
    if (iolinebuffer[numchar] == CR) break;    /* if a CR, exit from the read
     }                                            routine and move on      */
}

/*************************************************************************/

```

```
tx_line()
/* function to send a character at a time to the 8251A UART                */
{
/* send all the chars in the line buffer to the 8251A                      */
for (outptr =1; outptr <= numchar; ++outptr) {
/* idle until UART transmitter is ready */
   while (((tx_stat = in(COMM_STAT)) && TX_RDY) != TX_RDY) {}
   out(COMM_DATA,iolinebuffer[outptr]);        /* send out the character   */
   }
}

/**************************************************************************/

get_reply()
/* receives reply token from the BusMon unit after tx_line is performed    */
{
while (((rx_stat = in(COMM_STAT)) && RX_RDY) != RX_RDY) {}
/* idle until UART receiver is ready */
reply = in(COMM_DATA);                  /* get ACK/NAK token */
if (reply != ACK) {
   if (++errcount > 5) load_error(); /* if too many errors, exit */
   }
}

/**************************************************************************/

retrans_record()
/* tx_line by another name, done for improved legibility
/* since numchar is not destroyed by tx_line, this offers a very convenient
/* way to retransmit the same line of characters                           */
{
tx_line();
}

/**************************************************************************/

load_error()
/* only if five successive load errors are reported by the BusMon          */
{

/* EI          /* restore IRQs for standard I/O functions */

printf("error limit exceeded, load operation aborted\n");
fclose(infile); /* close the disk file                                     */
exit(88);       /* return to CP/M with error code 88                       */
}

```

APPENDIX C

LISTING 3

```
ASM      Z8001/Z8002                                         Page     1
V01.01-01   (8550)                                     30-NOV-83/12:00:49

                    4000 R                ORG      4000H
     00004000 21007A3A          GO        LD       R0,#7A3AH         ;SET UP UART FOR 2400 BAUD,
     00004004 3A060006                    OUTB     0006H,RH0         ;EVEN PARITY, 1 STOP BIT
     00004008 3A860006                    OUTB     0006H,RL0         ;7 DATA BITS ON 6SIO
     0000400C C827                        LDB      RL0,#27H          ;'B' SERIAL PORT TO DUMP
     0000400E 3A860007                    OUTB     0007H,RL0         ;BYTES TO THE PROLOG
                                ;
                                ; R10:    START ADDRESS (BYTE BOUNDARY) OF PROGRAM TO BE SENT TO PROLOG
                                ; R11:    END ADDRESS (BYTE BOUNDARY) OF PROGRAM
                                ; R12:    0=FOR EVEN NUMBERED BYTES, 1 FOR ODD NUMBERED BYTES
                                ;
                                ; NOTE: PLEASE RECALL THAT THE EVEN BYTES ARE LOW ORDER ADDRESSES BUT
                                ;         ARE ACTUALLY THE HIGH ORDER DATA BYTE. PLEASE REMEMBER THIS WHEN
                                ;         YOU USE THE NOTATION 'HIGH ORDER BYTE' WHEN DETERMINING WHICH
                                ;         PROM YOU ARE PROGRAMMING
                                ;
     00004012 A1A9              INIT      LD       R9,R10            ;USE R9 AS WORKSPACE, SAVE R10 FOR NXT LOAD
     00004014 81C9                        ADD      R9,R12            ;SET EVEN/ODD ADDRESSES TO BE DUMPED
     00004016 209B              MOVE      LDB      RL3,@R9           ;GET DATUM
     00004018 A0B3                        LDB      RH3,RL3           ;COPY DATUM TO WORK ON EACH NYBBLE
     0000401A 0703F00F                    AND      R3,#0F00FH        ;ISOLATE EACH NYBBLE
     0000401E B231FCFC                    SRLB     RH3,#4            ;REDUCE HO DIGIT TO HEX DIGIT
     00004022 0A030909                    CPB      RH3,#9            ;IS DIGIT DECIMAL OR HEX??
     00004026 E302                        JR       ULE,NOTHX         ;IF DECIMAL, NO OFFSET NEEDED
     00004028 00030707                    ADDB     RH3,#7            ;IF HEX, ADD 7 TO PUSH ASCII CODE TO ALPHA
     0000402C 00033030          NOTHX     ADDB     RH3,#30H          ;IN ANY EVENT, ADD ZONE BITS TO MAKE ASCII CHAR
     00004030 A03C                        LDB      RL4,RH3           ;MOVE FOR OUTPUT TO PROLOG
     00004032 DFF3                        CALR     PUTCHR            ;SEND IT OUT
     00004034 0A0B0909                    CPB      RL3,#9            ;SAME AS ABOVE
     00004038 E302                        JR       ULE,NOTHX2        ;THIS TIME FOR LO NYBBLE
     0000403A 000B0707                    ADDB     RL3,#7            ;SAME OFFSET
     0000403E 000B3030          NOTHX2    ADDB     RL3,#30H          ;SAME ZONE BITS
     00004042 A0BC                        LDB      RL4,RL3           ;PUT LETTER IN THE MAILBOX
     00004044 DFFC                        CALR     PUTCHR            ;HERE COMES THE POSTMAN
     00004046 A991                        INC      R9,#2             ;MOVE TO NEXT BYTE OF THE PROGRAM
     00004048 8BB9                        CP       R9,R11            ;AT THE END OF THE PROGRAM?
     0000404A E3E5                        JR       ULE,MOVE          ;IF NOT, GET ANOTHER BYTE!!
     0000404C E8E2                        JR       INIT              ;BREAKPOINT SET TO STALL HERE, THEN
                                                                     ; GO TO INIT FOR NEXT PROM
     0000404E 3AE40005          PUTCHR    INB      RL6,0005H         ;GET STATUS BITS
     00004052 A760                        BIT      R6,#0             ;IS UART STILL BUSY?
     00004054 E6FC                        JR       Z,PUTCHR          ;IF SO, WAIT UNTIL CHAR IS SENT...
     00004056 3AC60004                    OUTB     0004H,RL4         ;SEND DATUM TO THE B-PORT
     0000405A 9E08                        RET                        ;BACK TO MAIN PROG
                    4000                  END      GO                ;THAT'S ALL FOLKS!!!


ASM      Z8001/Z8002 SYMBOL TABLE
V01.01-01   (8550)                                     30-NOV-83/12:00:49

Section = $PROLOADLOAD, Inpage Relocatable, Size = 0000405C

GO----------------00004000   INIT--------------00004012   MOVE--------------00004016   NOTHX-------------0000402C
NOTHX2------------0000403E   PUTCHR------------0000404E

   45 Lines Read
   45 Lines Processed
    0 Errors
```