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 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 OF 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 OC hex to 43 hex (the ASCII code for the letter "C") the following happens: add 07 to OC 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


Figure 1. TEKHEX-format records used by BusMon loader program.
Figure 2. Sample TEKHEX file.

Figure 3. Z8000 loader outline.
APPENDIX A

LISTING 1

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

1 ; DATAC BUS MONITOR;
2 ; LOADER FOR Z8000 PROCESSOR INTERFACE TO DATAC SYSTEM
3 ; AUTHOR: S.M. NOVACKI 2 SEPT 83
4 ; REV 22 NOV 83: INCLUDES ERROR HANDLER- EXITS TO MONITOR
5 ; MACRO DEFINITIONS HERE:
6 7 MACRO NYBSUM
8 LDB RL2,RH2 ;TRANSPOSE HEX DIGITS
9 SRAB RL2,#04 ;MAKE HOO THE LOD
10 AND R2,#0FOFH ;MASK OFF HO BITS
11 ADDB RH2,RH2 ;ADD NYBBLES W/O CARRY
12 ;RH2 HOLDS NYBBLE CHECKSUM, TRANSFERRED TO RH7
13 ENDM
14 ;THIS MACRO PERFORMS A TEST FOR CHECKSUM ERRORS, IF >5 THE LOAD IS ABORTED
15 MACRO ERMSG
16 INC R13 ;COUNT NEW ERROR OCCURRENCE
17 CP R13,#5 ;REACH MAX# OF ERRORS?
18 JR UGT,ABRTLD ;TOO MANY ERRORS- RETURN TO MONITOR
19 SET R12,#01 ;SET 'OLD STRING, REPEAT' FLAG
20 LDB ACKBUF,#NAK ;READY BAD TX MSG
21 JR NEWSTR ;REQUEST REPEAT OF MSG AND CLEAR INBUF
22 ENDM
23
24 FE00 R ;ORG 0FE00
25 ;I/O STRING BUFFER DEFINITIONS, MUST BE ORG10 IN RAM
26 0000FE00 4 ACKBUF BLOCK 4 ;THREE BYTE BUFFER TO HANDSHAKE WITH 8550 DURING FILE TX
27 0000FE04 50 INBUF BLOCK 80 ;80 BYTE BUFFER FOR RECEIVING TEKHEX FILES
28 0000FE54 8 TKINAK BLOCK 8 ;I/O BLOK (WORKSPACE)
29 0000FE5C 8 TKinak BLOCK 8 ;I/O BLOK (WORKSPACE)
30
31 80E R ;ORG 080EH
32 ;CONSTANT DEFINITIONS:
33 3E PROMPT EQU 3EH ;8550 HANDSHAKE PROMPT CHAR
34 30 ACK EQU 30H ;MSG RECEIVED TOKEN
35 37 NAK EQU 37H ;MSG NOT RECEIVED TOKEN
36 00 RECEND EQU 00H ;OR USED TO TERMINATE PROMPT STRING
37 2F RECORD EQU 2FH ;'SLASH' CHAR USED TO DELIMIT TEKHEX RECORDS
38
39
40 ; BEGINNING OF LOADER ROUTINE;
41 ; CONSULT ZMON.DASSY AND .DLI_IP TO DETERMINE ACTUAL ADDRESSES
42 ; BEGINNING OF RCMABLE ROUTINES, ALL JUMPS RELATIVE, ONLY
43 ; RAM REFERENCES ARE ABSOLUTE FOR DURATION OF LOADER OPERATION
44
45 48 0000080E 53544420 "STK TEKHEX LOADER" ; NOTE# OF BYTES IN STRING MUST BE EVEN
46 54454848
47 4558204C
48 4F4A4445 5220
49
50 00000820 4D08FE00 R INTCOM CLR ACKBUF ;ONLY 3 OF 4 BYTES USED
51 00000828 4D08FE02 R CLR ACKBUF+2 ;IN HANDSHAKE SEQUENCE
52 00000828 4C05FE01 R LDB ACKBUF+1,RECEND ;READY STRING FOR
53 0000082E 4C05FE02 R 3E3E LDB ACKBUF+2,PRIMPT ;TEK HANDSHAKE
54 00000834 8DC8 CLR R12 ;FLAG: 0=NEW STRING, 1=REP'T OF LAST STRING
55 00000835 0F56 CLR SETIO ;SET UP FOR FOR INPUT OPERATIONS
56 0000083B 8CAB NEWSTR CLR RL2 ;(R2)=0 FOR ZAPPING
57 0000083E 210A0050 LD R10,#80 ;NUMBER OF BYTES TO BE ZAPPED
58 0000083E 2109FE04 R LD R0,#10NUF
59 00000842 7290AD00 ZAPWRD LDB R9(R10),RL2 ;ZERO OUT INBUF (I HOPE...)
60 00000846 ABAD DEC R10
61 00000848 EEFC JR NZ,ZAPWRD;
62 0000084A 2101FE5C R OUTMSG LD R1,#TKINAK ;SELECT SIGNAL MODE FOR TEK
63 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
; THE 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

GETSTR LD R1,#THXIN ; SELECT HEX RECORD READ-MODE
SC #0 ; GET HEX RECORD AND SAVE IT AT INBUF
LDA R8,INBUF ; SET BASE ADDRESS OF HEX STRING

; AT THIS POINT, WE SHOULD HAVE ONE COMPLETE TEXHEX 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
; R7: CONTAINS THE NYBBLE CHECKSUMS
; 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
SEEK CP R8,#INBUF+80 ; AT THE END OF THE INPUT BUFFER?
JR EQ,STREQ
C=I
JR R8,#RECMRK
JR EQ,TSTSTR
INC R8
JR NE,SEEK
STREQ LD AC,<BUF,#NAK

; AT THE END OF THE INPUT BUFFER?
; IF SO, THE WHOLE RECORD WAS JUNK, GET ANOTHER
; SCAN FOR THE 'SLASH' CHARACTER
; FOUND IT!
; ON TO THE NEXT CHAR
; HEADER NOT FOUND, TRY AGAIN
; BAD TX, ASK FOR REPEAT OF STRING
JR NEWS'FR ; DO THE ASKING

; END OF SOM-SCANNER ROUTINE
; WE'LL ASSUME THAT A VALID RECORD HAS BEEN READ

; WE'LL ASSUME THAT A VALID RECORD HAS BEEN READ
TSTSTR TEST R12 ; IS THIS NEW OR OLD DATA?
JR NZ,OLDSTR
CLR R13 ; ZERO OUT CKSUM ERROR ACCUMULATOR
OLDSTR CALR CHKTRM ; SEE IF THE RECORD IS THE ZERO-LENGTH TERMINATOR
IF TERM RECORD IS FOUND, RETURN TO MONITOR
INC R8 ; MOVE POINTER PAST HEADER TO FIRST ASCII CHARACTER

; (R8) ADDRESS OF FIRST CHAR IN HEX STRING
CALR ASCHEX ; GET 1ST BYTE OF ADDRESS
LOD RH2,RH4 ; 1ST BYTE TO CKSUM ACCUMULATOR

; TRANSPOSE HEX DIGITS
; MAKE MOD THE LOB
; MASK OFF HO BITS
; ADD NYBBLES W/O CARRY
; TO CHECKSUM ACCUMULATOR
; HOBYTE OF ADDRESS TO R6
; NEXT DIGIT
; ADD IT TO ACCUM

122
000008A6 A980 ; INC R8 ; ON TO THE BYTE COUNT
000008A8 D4A0 ; CLR ASCHEX ; GET # OF BYTES IN MSG
000008AA A042 ; LDB RH2,RH4 ; ADD IT TO CHKSUM
140   M    NYBSUM
141 000008AC A02A ; LDB RL2,RL2 ; TRANPOSE HEX DIGITS
142 000008AE 8A5F ; SRAB R2,#04 ; MAKE HO THE LOD
143 000008B2 07020FOF M ; AND R2,FOFOF M ; MASK OFF HO BITS
144 000008B8 80A2 ; ADDOB RH2,RL2 ; ADD NYBBLES W/O CARRY
147 000008BA 8027 ; ADDOB RH7,RL2 ; ADD RUNNING NYBBLE CHECKSUM
148 000008BC A04F ; LDB RL7,RL4 ; SAVE # OF DATA BYTES IN HEX FOR RAM LOAD
150 000008B0 8800 ; INC RB ; SET CHAR CNT FROM STRING
151 000008B6 DF6A ; CALR CHKSUM ; TEST 1ST BYTE-CHECKSUM
152   M    ERRMSG
153 000008C2 A9D0 ; INC R13 ; COUNT NEW ERROR OCCURRENCE
154 000008C4 08D00005 M ; CP R13,#5 ; REACH MAX # OF ERRORS?
155 000008CB E825 ; JR UGT,ABRTLD ; TOO MANY ERRORS- RETURN TO MONITOR
156 000008CA 45C1 ; SET R12,#01 ; SET 'OLD STRING, REPEAT' FLAG
157 000008CC 4C05FEO0 MR ; LDB ACKBUF,#NAK ; READY BAD TX MSG
158   M    ERRMSG
159 000008E8 E822 ; JR NEWSTR ; REQUEST REPEAT OF MSG AND CLEAR INBUF
160 000008D4 8C78 ; SUMOK CLR RH7 ; RESET ACCUMULATOR FOR FOR SECOND CHECKSUM
161 000008D6 A9B0 ; HXLOAD INC RB ; NEX CHR
162 000008D8 DF65 ; CALR ASCHEX ; FORM DATA BYTE
163 000008DA A042 ; LDB RH2,RL4 ; SEND TO CHKSUM ACCUM
164   M    NYBSUM
165 000008DC A02A ; LDB RL2,RL2 ; TRANPOSE HEX DIGITS
166 000008DE B29FCFC M ; SRAB RL2,#04 ; MAKE HO THE LOD
167 000008F2 07020FOF M ; AND R2,FOFOF M ; MASK OFF HO BITS
168 000008F0 80A2 ; ADDOB RH2,RL2 ; ADD NYBBLES W/O CARRY
169 000008F8 8273 ; ADDOB RH7,RL2 ; ADD RUNNING NYBBLE CHECKSUM
170 000008FA 2E64 ; LDB #R6,RL4 ; STORE MACHINE CODE
171 000008EA A960 ; INC RB ; NEXT RAM LOCATION...
172 000008EC 45C1 ; SET R6 ; NEXT ERROR COUNT
173 000008F0 EF62 ; DCSB RL7 ; ONE LESS BYTE TO STORE
174 000008F2 A980 ; INC RB ; RECORD LOAD COMPLETE
175   M    ERRMSG
176 000008F4 DF65 ; CALR CHKSUM ; PRODUCE AND COMPARE SECOND BYTE-CHECKSUM
177 000008F6 E609 ; JR EQ,GOODRX ; NO ERRORS
178   M    ERRMSG
179 000008F8 A9D0 ; INC R13 ; COUNT NEW ERROR OCCURRENCE
180   M    ERRMSG
181 000008FA 08000005 M ; CP R13,#5 ; REACH MAX # OF ERRORS?
182 000008FE EB60 ; JR UGT,ABRTLD ; TOO MANY ERRORS- RETURN TO MONITOR
183 00000900 A5C1 ; SET R12,#01 ; SET 'OLD STRING, REPEAT' FLAG
184 00000902 4C05FEO0 MR ; LDB ACKBUF,#NAK ; READY BAD TX MSG
185   M    ERRMSG
186 00000908 E897 ; JR NEWSTR ; REQUEST REPEAT OF MSG AND CLEAR INBUF
187 0000090A 4C05FEOO R ; GOODRX LDB ACKBUF,#ACK ; SET ACKNOWLEDGE TOKEN
188   M    ERRMSG
189 00000910 80C8 ; CLR R12 ; CLEAR FLAG FOR A NEW STRING
190 00000912 E892 ; JR NEWSTR ; SEND IT TO THE 8550
191 00000914 2101091C R ; ABRTLD LD R1,#MSGBLK ; READY ERROR MSG FOR TX TO TEX CONSOLE
192 00000918 7FO0 ; SC #0 ; SEND IT OUT
193 0000091A 7F03 ; SC #3 ; RETURN TO Z8000 MONITOR
194 0000091C 0200 ; MSGBLK WORD 0200H ; TX MODE FOR SC#0
195 0000091E 0000 ; WORD 0000H ; NOT USED
196 00000920 929A ; WORD ENDMSG ; ADDRESS OF ERROR MSG
197 00000922 0025 ; WORD LSTCHR-ENDMSG ; # OF CHAR'S IN STRING TO BE TX'D
198 00000924 2F2F2020 ; ENDMOD ASCII " ERROR LIMIT EXCEEDED, LOAD IS ABORTED" ; SELF-EXPLANATORY

; ON TO THE BYTE COUNT
; GET # OF BYTES IN MSG
; ADD IT TO CHKSUM
; COUNT NEW ERROR OCCURRENCE
; REACH MAX # OF ERRORS?
; TOO MANY ERRORS- RETURN TO MONITOR
; SET 'OLD STRING, REPEAT' FLAG
; READY BAD TX MSG
; REQUEST REPEAT OF MSG AND CLEAR INBUF
; PRODUCE AND COMPARE SECOND BYTE-CHECKSUM
; NO ERRORS
; COUNT NEW ERROR OCCURRENCE
; REACH MAX # OF ERRORS?
; TOO MANY ERRORS- RETURN TO MONITOR
; READY BAD TX MSG
; REQUEST REPEAT OF MSG AND CLEAR INBUF
; SET ACKNOWLEDGE TOKEN
; CLEAR FLAG FOR A NEW STRING
; SEND IT TO THE 8550
; READY ERROR MSG FOR TX TO TEX CONSOLE
; SEND IT OUT
; RETURN TO Z8000 MONITOR
; TX MODE FOR SC#0
; NOT USED
; ADDRESS OF ERROR MSG
; # OF CHAR'S IN STRING TO BE TX'D
; ERROR LIMIT EXCEEDED, LOAD IS ABORTED" ; SELF-EXPLANATORY

123
; 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.

208 00000950 208C ASCHEX LOD R4,#R8 ;GET 1ST ASCII CHARACTER
209 00000952 0FD3 CALR TSTNUM ;ADJUST ASCII IF CHR IS A-F
210 00000954 060C0F0F ANDB RL4,#OFH ;MASK OFF ZONE BITS
211 00000958 82C90404 SLAB RL4,#04 ;LSBITS BECOME MSBITS
212 0000095C A0C4 LOD RH4,R4 ;READY FOR NEXT DIGIT
213 0000095E A980 INC R8 ;NEXT DIGIT
214 00000960 208C LOD RL4,#R8 ;GET IT
215 00000962 DFDB CALR TSTNUM ;ADJUST ASCII IF CHR IS A-F
216 00000964 060C0F0F ANDB RL4,#OFH ;PROCESS IT
217 00000968 84C4 D0B RH4,R4 ;FORM COMPLETE BYTE OF DATA
218 0000096A 9E08 RET ;GO HOME

;CHECKSUM: COMPARE THE COMPUTED CHECKSUM WITH THE VALUE CONTAINED IN THE
;STRING TRANSMITTED FORM THE 8550. RUNNING CHECKSUM IS MAINTAINED IN
;R7. THIS ROUTINE CALLS ASCHEX TO READ THE ASCII STRING AND GEN THE
;TX CHECKSUM.

220 0000096C 000F CHKSUM CALR ASCHEX ;GET 1ST BYTE-CHECKSUM
221 0000096E 8A47 CPB RH7,RH4 ;COMPARE CALCULATED AND GIVEN CHECKSUMS
222 00000970 9E08 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

229 00000972 A18A CHKTRM LD RIO,R8 ;SAVE CURRENT POSITION IN STRING
230 00000974 A984 INC R8,#1 ;AIM AT 1ST CHR OF BYTE COUNT
231 00000976 0014 CALR TSTNUM ;FORM BYTE COUNT
232 00000978 A1A8 LD R8,R10 ;RECOVER ORIGINAL POINTER
233 0000097A 8C44 TESTB RH4 ;IS DATA STRING LENGTH ZERO?
234 0000097C 9E08 RET NE ;NO, GO BACK AND FINISH PROCESSING

;AT THIS POINT, WHO CARES ABOUT A BIT-ERROR?

235 0000097E 4D05FEOO R0030 LD ACKBUF,#ACK ;SIGNAL THE END
236 00000984 2101FESC R LD R1,#'rKINAK ;READY THE MSG
237 00000988 7F00 CPB RH7,RH4 ;COMPARE TRANSFER END TO HOST COMPUTER
238 0000098A 7F03 CPB RIO,RH4 ;LOAD COMPLETED, RETURN TO MONITOR

;SETIO: USED TO RESET FC8 FOR SC#0

239 0000098C 210AF534 R SETIO LD R10,#'KXXHIN ;DEST FOR MOVE
240 00000990 2108099E R LD R11,#'OBKL ;SOURCE FOR MOVE
241 00000994 21090008 R LD R9,#08H ;# OF WORDS TO MOVE
242 00000998 B8B10940 WMOVE LD R11,#R11,R9 ;GO IT!
243 0000099C 9E08 RET ;GO HOME...

;I=OBKL WORD 0100H ;BLOCK RECEIVE MODE OF MONITOR CONSOLE HANDLER

245 000009AC 0003 WORD 0000H ;NOT USED
246 000009AD 0200 WORD 0200H ;BLOCK TRANSMIT MODE FOR SYSTEM CALL #0

;MANIPULATION TO HEX FORM

247 000009AE 0A0C3599 TSTNUM CPB RL4,#35H
248 000009B2 E202 JR LE;ISNUM ;IF 0-9, NO CORRECTION NEEDED
249 000009B4 000C0099 ADDB RL4,#9 ;ELSE ADD OFFSET OF 9 TO PRODUCE USEABLE LO NYBBLE
250 000009B6 9E08 ISNUM RET ;BACK TO ASCHEX

;end of loader and subroutines

261 00000820 END INTCOM; PROGRAM START ADDRESS FOR ASSEMBLER
## Symbol Table

### Scalars
- **ACK**: 00000030
- **NAK**: 00000057
- **PROMPT**: 0000003E
- **RECO**: 0000003E
- **RECEO**: 000000D

### Strings & Macros
- **ERRMSG**: M
- **NYSUM**: M

### Section Information
- **Section**: "$BMLLOA£), Inpage Relocatable, Size = 0000FE64"

### Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABRTLO</td>
<td>00000914</td>
</tr>
<tr>
<td>CHKTRM</td>
<td>00000972</td>
</tr>
<tr>
<td>GETSTR</td>
<td>00000850</td>
</tr>
<tr>
<td>INTCOM</td>
<td>00000820</td>
</tr>
<tr>
<td>MGBLX</td>
<td>0000091C</td>
</tr>
<tr>
<td>SEEK</td>
<td>0000085A</td>
</tr>
<tr>
<td>TKHIN</td>
<td>0000FE54</td>
</tr>
<tr>
<td>TSTSTR</td>
<td>00000872</td>
</tr>
<tr>
<td>ACKBUF</td>
<td>0000FE00</td>
</tr>
<tr>
<td>CRLF</td>
<td>0000094D</td>
</tr>
<tr>
<td>GOODRX</td>
<td>0000090A</td>
</tr>
<tr>
<td>IOBLK</td>
<td>0000099E</td>
</tr>
<tr>
<td>NEWSTR</td>
<td>00000838</td>
</tr>
<tr>
<td>TKINAK</td>
<td>0000FE5C</td>
</tr>
<tr>
<td>WMOVE</td>
<td>00000998</td>
</tr>
<tr>
<td>ASCHEX</td>
<td>00000950</td>
</tr>
<tr>
<td>ENDMSC</td>
<td>00000924</td>
</tr>
<tr>
<td>HXLOAD</td>
<td>00000806</td>
</tr>
<tr>
<td>ISNUM</td>
<td>00000988</td>
</tr>
<tr>
<td>OLOSTR</td>
<td>00000878</td>
</tr>
<tr>
<td>TMMSG</td>
<td>00000804</td>
</tr>
<tr>
<td>TSTNUM</td>
<td>000009AE</td>
</tr>
</tbody>
</table>

230 Lines Read
268 Lines Processed
0 Errors
APPENDIX B
LISTING 2

1: /*
2: -
3: - BUSLODR.C: 8550 DOWNLOAD EMULATOR FOR DEC VT-180
4: - WRITTEN IN AZTEC C FOR THE CP/M ENVIRONMENT
5: -
6: - AUTHOR: S. NOVACKI
7: - CREATED: JULY, 1984
8: -
9: */
10: #include "b:stdio.h" /* standard I/O used for file handling */
11: #define ACK '0' /* definitions of: the ACK token */
12: #define NAK '7' /* the NAK token */
13: #define CR 13 /* end-of-line flag */
14: #define TX_RDY 0x01 /* UART transmitter ready flag bit */
15: #define RX_RDY 0x02 /* receiver ready bit */
16: #define COMM_DATA 0x58 /* UART data register port number */
17: #define COMM_STAT 0x59 /* status register port number */
18: /
19: /
20: /*
21: infile:
22:     pointer for source file (from disk)
23: numchar:
24:     subscript for reading characters from disk file into buffer vector
25: outptr:
26:     subscript for sending buffer characters to UART
27: argc:
28:     command line argument count, used by "C" console processor
29: errcount:
30:     number of consecutive reception errors
31: iolinebuffer:
32:     array used in moving characters from disk file using standard
33: I/O to UART using system-specific hardware
34: reply:
35:     token read from BusMon system to indicate quality of message
36: tx_stat, rx_stat:
37:     UART-register statuses used during character-send procedure
38: */
39: /
40: FILE *infile,*fopen();
41: int numchar,outptr,argc,errcount = 0;
42: char iolinebuffer[80],reply,tx_stat,rx_stat;
43: /
44: /*****************************************************************************/
45: main(argc,argv)
46: char *argv[];
47: {
48: char *argv[];
49: {*/
50: */
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
    /* bring back IRQs for BDOS/BIOS disk I/O routines */
    EI
    #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 */
}

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 */
}

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 */
}
105: tx_line()
106: /* function to send a character at a time to the 8251A UART */
107: {
108: /* send all the chars in the line buffer to the 8251A */
109: for (outptr = 1; outptr < numchar; ++outptr) {
110: /* idle until UART transmitter is ready */
111: while (((tx_stat = in(COMM_STAT)) && TX_RDY) != TX_RDY) {}
112: out(COMM_DATA, iolinebuffer[outptr]); /* send out the character */
113: }
114: }
115: /****************************************************************************/
116: 117: get_reply()
118: /* receives reply token from the BusMon unit after tx_line is performed */
119: {
120: while (((rx_stat = in(COMM_STAT)) && RX_RDY) != RX_RDY) {}
121: /* idle until UART receiver is ready */
122: reply = in(COMM_DATA); /* get ACK/NAK token */
123: if (reply != ACK) {
124: if (++errcount > 5) load_error(); /* if too many errors, exit */
125: }
126: }
127: /****************************************************************************/
128: 129: retrans_record()
130: /* tx_line by another name, done for improved legibility */
131: /* since numchar is not destroyed by tx_line, this offers a very convenient */
132: /* way to retransmit the same line of characters */
133: {
134: tx_line();
135: }
136: /****************************************************************************/
137: 138: load_error()
139: /* only if five successive load errors are reported by the BusMon */
140: {
141: /* EI */ /* restore IRQs for standard I/O functions */
142: printf("error limit exceeded, load operation aborted\n");
143: fclose(infile); /* close the disk file */
144: exit(88); /* return to CP/M with error code 88 */
145: }
146: /****************************************************************************/
147: 148: 149: 150: 151:
APPENDIX C
LISTING 3

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4000 R</td>
</tr>
<tr>
<td>2</td>
<td>00004000 21007A3A GO LD R0,#7A3AH ;SET UP UART FOR 2400 BAUD,</td>
</tr>
<tr>
<td>3</td>
<td>00004004 3A060006 OUTB 0006H,RO ;EVEN PARITY, 1 STOP BIT</td>
</tr>
<tr>
<td>4</td>
<td>00004008 3A860006 OUTB 0006H,RO ;6 DATA BITS ON 6510</td>
</tr>
<tr>
<td>5</td>
<td>0000400C CB27 LD R0,#27H ;'B' SERIAL PORT TO DUMP</td>
</tr>
<tr>
<td>6</td>
<td>0000400E 3A860007 OUTB 0007H,RO ;BYTES TO THE PROLOG</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>00004012 A1A9 INIT LD R9,R10 ;USE R9 AS WORKSPACE, SAVE R10 FOR NEXT LOAD</td>
</tr>
<tr>
<td>18</td>
<td>00004014 81C9 ADD R9,R12 ;SET EVEN/ODD ADDRESSES TO BE DUMPED</td>
</tr>
<tr>
<td>19</td>
<td>00004016 2098 MOVE LD R3,#89 ;GET DATUM</td>
</tr>
<tr>
<td>20</td>
<td>00004018 A0B3 LDB RH3,R3 ;COPY DATUM TO WORK ON EACH NYBBLE</td>
</tr>
<tr>
<td>21</td>
<td>0000401A 0703F00F AND R3,#F00FH ;ISOLATE EACH NYBBLE</td>
</tr>
<tr>
<td>22</td>
<td>0000401C 231FFC SRLB RH3,#4 ;REDUCE NO DIGIT TO HEX DIGIT</td>
</tr>
<tr>
<td>23</td>
<td>00004020 00309099 CPB RH3,#9 ;IS DIGIT DECIMAL OR HEX??</td>
</tr>
<tr>
<td>24</td>
<td>00004022 00085030 E302 JR ULE,NOTHX ;IF DECIMAL, NO OFFSET NEEDED</td>
</tr>
<tr>
<td>25</td>
<td>00004024 00030707 00030707 ADDB RH3,#7 ;IF HEX, ADD 7 TO PUSH ASCII CODE TO ALPHA</td>
</tr>
<tr>
<td>26</td>
<td>00004026 A03C NOTHX ADDB RH3,#30H ;IN ANY EVENT, ADD ZONE BITS TO MAKE ASCII CHAR</td>
</tr>
<tr>
<td>27</td>
<td>00004028 0A3E NOTHX LDB RL4,R4 ;MOVE FOR OUTPUT TO PROLOG</td>
</tr>
<tr>
<td>28</td>
<td>00004030 DFF3 CALR PUTCHR ;SEND IT OUT</td>
</tr>
<tr>
<td>29</td>
<td>00004032 000B0909 CPB RL3,#9 ;SAME AS ABOVE</td>
</tr>
<tr>
<td>30</td>
<td>00004034 E302 JR ULE,NOTHX2 ;THIS TIME FOR LO NYBBLE</td>
</tr>
<tr>
<td>31</td>
<td>00004036 000B0707 ADDB RL3,#7 ;SAME OFFSET</td>
</tr>
<tr>
<td>32</td>
<td>00004038 000B3030 NOTHX2 ADDB RL3,#30H ;SAME ZONE BITS</td>
</tr>
<tr>
<td>33</td>
<td>0000403A 000B3030 NOTHX ADDB RL4,R4 ;PUT LETTER IN THE MAILBOX</td>
</tr>
<tr>
<td>34</td>
<td>0000403C A08C CALR PUTCHR ;HERE COMES THE POSTMAN</td>
</tr>
<tr>
<td>35</td>
<td>00004040 A991 INC R9,#2 ;MOVE TO NEXT BYTE OF THE PROGRAM</td>
</tr>
<tr>
<td>36</td>
<td>00004042 000B0889 CP R9,R11 ;AT THE END OF THE PROGRAM?</td>
</tr>
<tr>
<td>37</td>
<td>00004044 E3E5 JR ULE,MOVE ;IF NOT, GET ANOTHER BYTE!!</td>
</tr>
<tr>
<td>38</td>
<td>00004046 000E82E JR INIT ;BREAKPOINT SET TO STALL HERE, THEN</td>
</tr>
<tr>
<td>39</td>
<td>00004048 000E82E CALR ;GO TO INIT FOR NEXT PROM</td>
</tr>
<tr>
<td>40</td>
<td>0000404A 3A4E0005 PUTCHR INB RL6,0005H ;GET STATUS BITS</td>
</tr>
<tr>
<td>41</td>
<td>0000404C A760 BIT R6,#0 ;IS UART STILL BUSY?</td>
</tr>
<tr>
<td>42</td>
<td>0000404E 004054 E6FC JR Z,PUTCHR ;IF SO, WAIT UNTIL CHAR IS SENT...</td>
</tr>
<tr>
<td>43</td>
<td>00004056 3AC60004 OUTB 0004H,RL4 ;SEND DATUM TO THE B-PORT</td>
</tr>
<tr>
<td>44</td>
<td>00004058 9E08 RET ;BACK TO MAIN PROG</td>
</tr>
<tr>
<td>45</td>
<td>0000405A 4000 END GO ;THAT'S ALL FOLKS!!!</td>
</tr>
</tbody>
</table>

ASM Z8001/Z8002 SYMBOL TABLE
V0.01-01 (8550) 30-NOV-83/12:00:49

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

GO-----------------------------00004000 INIT-----------------------------00004012 MOVE-----------------------------00004016 NOTHX-----------------------------00004020

NOTHX2-------------------------00004035 PUTCHR-------------------------00004040

45 Lines Read
45 Lines Processed
0 Errors