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 DATAAC Bus Monitor is described.

I. INTRODUCTION

The DATAAC Bus Monitor Unit (BusMon) is a Z8000-based microcomputer system designed to receive, interpret, and display selected data items appearing on a DATAAC 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 DATAAC 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 0C 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


<table>
<thead>
<tr>
<th>Record</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>/aaaaabbacbddd...dddc &lt;CR&gt;</td>
</tr>
<tr>
<td>Load Byte</td>
<td>ADDRESS                  BYTE COUNT CHKSUM DATA BYTES 2nd CHKSUM RECORD TERMINATOR</td>
</tr>
<tr>
<td>Terminator</td>
<td>/xxxxooaac</td>
</tr>
<tr>
<td>Load Zero</td>
<td>ADDRESS                  ZERO LENGTH CHKSUM RECORD</td>
</tr>
</tbody>
</table>

Abort Message Text

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
V01.01-01 (8550)  01-DEC-83/08:48:48

1  ; DATA 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,RL2  ;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 ERRMSG
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 FEO0 R  ;ORG OFE0M
25 ;I/O STRING BUFFER DEFINITIONS, MUST BE ORG'ED 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 TKNXIN BLOCK 8   ;TEKHEX BLOCK
29 0000FE5C 8 TKNBKC BLOCK 8   ;I/O BLOX (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 0D RECMD EQU 0DH   ;OR USED TO TERMINATE PROMPT STRING
37 2F RECMRK 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 ROMABLE ROUTINES, ALL JUMPS RELATIVE, ONLY
43 ; RAM REFERENCES ARE ABSOLUTE FOR DURATION OF LOADER OPERATION
44 ;
45 0000080E 53544420 ;TMMSG ASCII 'STD TEKHFX LOADER' ; NOTE# OF BYTES IN STRING MUST BE EVEN
46 54454848
47 4558204C
48 4F4A4445
49 5220
50
51 00000820 4D08FE00 R INTCOM CLR ACKBUF   ;ONLY 3 OF 4 BYTES USED
52 00000824 4D08FE02 R CLR ACKBUF+2   ;IN HANDSHAKE SEQUENCE
53 00000828 4C05FE01 R LDB ACKBUF+1,#RECMD ;READY STRING FOR
54 0000082E 4C05FE02 R 3E3E LDB ACKBUF+2,#PROMPT ;TEK HANDSHAKE
55
56 00000834 8D38 CLR R12 ;FLAG: 0=NEW STRING, 1=REP'T OF LAST STRING
57 00000836 0F56 CLR SET10 ;SET UP FOR INPUT OPERATIONS
58  ;SET UP PCB FOR OUTPUT OPERATIONS
59 00000838 8CA8 NEWSTR CLR R2   ;(R2)=0 FOR ZAPPING
60 0000083A 210A0050 LD R10,#80 ;NUMBER OF BYTES TO BE ZAPPED
61 0000083E 2109FE04 R LD R9,#100BF
62 00000842 729A0000 ZAPWRD LDB R9(R10),R2   ;ZERO OUT INBUF (I HOPE...)
63 00000846 8BA9 DEC R10
64 00000848 EEFC JR NZ,ZAPWRD;
65 0000084A 2101FE5C R OUTMSG LD R1,#TKINAK ;SELECT SIGNAL MODE FOR TEK
66 0000084E 7F00 SC #0 ;OUTPUT PROMPT VIA MONITOR ROUTINE

121
;HOPEFULLY WITH A SERIAL LINE DEDICATED TO THE Z8K-TEK INTERFACE
;THERE WOEN'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

00000850 2101FE54 R GETSTR LD R1,#THXIN ;SELECT HEX RECORD READ-MODE
00000854 7FOO 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
R7: CONTAINS THE NYBBLE CHECKSUMS
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/ZBK COMMUNICATION

0000085A 0808FE.54 R SEEK_CP R8,#INBUF+80 ;AT THE END OF THE INPUT BUFFER?
0000085E E605 JR EQ,STREQ ;IS THIS THE WHOLE RECORD JUNK, GET ANOTHER

00000860 0CB12F2F CPB @R8,#RECMRK ;SCAN INBUF FOR THE 'SLASH' CHARACTER
00000864 E606 JR EQ,TSTSTR ;FOUND IT!
00000866 A980 INC R8 ;ON TO THE NEXT CHAR
00000868 EEFE JR NE,SEEK ;HEADER NOT FOUND, TRY AGAIN

0000086A 4D05FE00 R STREQ LD AOXBUF,#NAK ;BAD TX, ASK FOR REPEAT OF STRING

00000870 EBE3 JR NEWSTR ;DO THE ASKIN'

;END OF SOM-SCANNER ROUTINE

;WE'LL ASSUME THAT A VALID RECORD HAS BEEN READ

00000872 B0C4 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 BD08 CLR R13 ;ZERO OUT CKSUM ERROR ACCUMULATOR
00000878 08F4 OLDSTR CLR CHKTRM ;SEE IF THE RECORD IS THE ZERO-LENGTH TERMINATOR
0000087A A980 INC R8 ;MOVE POINTER PAST HEADER TO FIRST ASCII CHARACTER

0000087C 9800 (R8)+ADDRESS OF FIRST CHAR IN HEX STRING
0000087E C9FE CALR ASCHEX ;GET 1ST BYTE OF ADDRESS

00000880 A042 LDB RH2,RH4 ;1ST BYTE TO CKSUM ACCUMULATOR

00000882 5000 MYSUM ATL RH2,RH4 ;ADD IT TO ACCUM

00000884 02A M LDB RL2,RH2 ;TRANSPOSE HEX DIGITS

00000886 B2AFCFCM SRAB RL2,#0A ;MAKE MOD THE LOO
00000888 07020F0F M ADD RH2,RL2 ;ADD NYBBLES W/O CARRY
122 0000088C A027 LDB RH7,RH2 ;TO CHECKSUM ACCUMULATOR
123 0000088E A046 LDB RH6,RH4 ;HOBYTE OF ADDRESS TO R6
124 00000890 A980 INC R8 ;NEXT DIGIT
125 00000892 C9FE CALR ASCHEX ;GET SECOND BYTE OF LOAD ADDRESS
126 00000894 A042 LDB RH2,RH4 ;2ND BYTE TO CKSUM ACCUMULATOR
127 00000896 5000 MYSUM ATL RH2,RH4
128 00000898 02A M LDB RL2,RH2 ;TRANSPOSE HEX DIGITS
129 0000089A B2AFCFCM SRAB RL2,#0A ;MAKE MOD THE LOO
130 0000089C 07020F0F M ADD RH2,RL2 ;ADD NYBBLES W/O CARRY
131 000008A0 A027 LDB RH7,RH2 ;TO CHECKSUM ACCUMULATOR
132 000008A2 8027 ADD RH7,RH2 ;ADD IT TO ACCUM
133 000008A4 A04E LDB RL6,RH4 ;LOBYTE TO R6; LOAD ADDRESS IS NOW COMPLETE
136
137 000008A6 A980 INC R8 ; ON TO THE BYTE COUNT
138 000008A8 D4A0 CALL ASCHEX ; GET # OF BYTES IN MSG
139 000008AA A042 LDB RH2,RH4 ; ADD IT TO CHKSUM
140 M NYBSUM
141 000008AC A02A M LDB RL2,RL2 ; TRANSPOSE HEX DIGITS
142 000008AE 78ACF3C M SRAB RL2,#04 ; MAKE HOB THE LOB
143 000008B0 07020FO0 M AND R2,#FOFH ; MASK OFF HO BITS
144 000008B6 80A2 M ADDB RH2,RL2 ; ADD NYBBLES W/O CARRY
145 000008BB 8027 ADDB RH7,RL2 ; ADD RUNNING NYBBLE CHECKSUM
146 000008BA A04F M LDB RL7,RL4 ; SAVE # OF DATA BYTES IN HEX FOR RAM LOAD
147 000008BD A980 INC R8 ; SET CHAR CNT FROM STRING
148 000008BE D4AA CALL CHKSUM ; TEST 1ST BYTE-CHECKSUM
149 000008C0 E609 JR EO,SUMOK ; NO PROBS, GO ON
150 M ERRMSG
151 000008C2 A900 M INC R13 ; COUNT NEW ERROR OCCURRENCE
152 000008C4 08000005 M CP R13,#5 ; REACH MAX # OF ERRORS?
153 000008C8 EB25 M JR UGT,ABRTLD ; TOO MANY ERRORS- RETURN TO MONITOR
154 000008CA A5C1 M SET R12,#01 ; SET 'OLD STRING, REPEAT' FLAG
155 000008CC 4C05FEO0 MR LDB ACKBUF,#NAK ; READY BAD TX MSG
156 M 3737
157 000008D0 E8B2 M JR NEWSR ; REQUEST REPEAT OF MSG AND CLEAR INBUF
158 000008D4 8C78 SUMOK CLR RH7 ; RESET ACCUMULATOR FOR FOR SECOND CHECKSUM
159 000008D6 A980 HXLOAD INC R8 ; NXT CHR
160 000008D8 DFC5 CALL ASCHEX ; FORM DATA BYTE
161 000008DA A042 M LDB RH2,RH4 ; SENT TO CHKSUM ACCUM
162 M NYBSUM
163 000008DC A02A M LDB RL2,RL2 ; TRANSPOSE HEX DIGITS
164 000008DE B2A9FCFC M SRAB RL2,#04 ; MAKE HOB THE LOB
165 000008F0 07020FO0 M AND R2,#FOFH ; MASK OFF HO BITS
166 000008F6 80A2 M ADDB RH2,RL2 ; ADD NYBBLES W/O CARRY
167 000008FB 8027 ADDB RH7,RL2 ; ANOTHER DIGIT TO BE SUMMED
168 000008FE 2E64 M LDB #R6,RL4 ; STORE MACHINE CODE
169 00000900 A960 INC R6 ; NEXT RAM LOCATION...
170 00000904 A5C0 M SET R12,#01 ; SET 'OLD STRING, REPEAT' FLAG
171 00000908 E827 M JR NEWSR ; UNTIL (RL7)=0, STORE THEM BYTES!
172 M 3737
173 0000090E EEP2 M JR RECORD LOAD COMPLETE ; PRODUCE AND COMPARE SECOND BYTE-CHECKSUM
174 00000910 A980 INC R8
175 00000914 DFC5 CALL CHKSUM ; PRODUCE AND COMPARE SECOND BYTE-CHECKSUM
176 00000916 E609 JR EQ,GOODRX ; NO ERRORS
177 M ERRMSG
178 00000918 A900 M INC R13 ; COUNT NEW ERROR OCCURRENCE
179 0000091C EB25 M JR UGT,ABRTLD ; TOO MANY ERRORS- RETURN TO MONITOR
180 00000920 A5C1 M SET R12,#01 ; SET 'OLD STRING, REPEAT' FLAG
181 00000924 4C05FEO0 MR LDB ACKBUF,#NAK ; READY BAD TX MSG
182 M 3737
183 00000928 E897 M JR NEWSR ; REQUEST REPEAT OF MSG AND CLEAR INBUF
184 0000092A 4C05FEO0 R GOODRX LDB ACKBUF,#ACK ; SET ACKNOWLEDGE TOKEN
185 M 3030
186 00000930 80C8 CLR R12 ; CLEAR FLAG FOR A NEW STRING
187 00000932 E92 JR NEWSR ; SEND IT TO THE 8550
188 00000934 2101091C R ABRTLD LD R1,#MSGBLK ; READY ERROR MSG FOR TX TO TX CONSOLE
189 00000938 7F00 SC #0 ; SEND IT OUT
190 0000093A 7F03 SC #3 ; RETURN TO 8550 MONITOR
191 0000093C 0200 MSGBLK WORD 0200H ; TX MODE FOR SC#0
192 0000093E 0000 WORD 0000H ; NOT USED
193 00000940 092A R WORD ENDMSEG ; ADDRESS OF ERROR MSG
194 00000942 0229 WORD LSTCHR-ENDMSEG ; # OF CHARS IN STRING TO BE TX'D
195 00000944 2820 ENDMSG ASCII "ERROR LIMIT EXCEEDED, LOAD IS ABORTED' ; SELF-EXPLANATORY
196 4552524F
197 52204C49
198 40495420
199 45544544
200 2C0204CF
201 41442049
202 53204142
203 4F525445
204 44
205 0000094A 000A CR/LF BYTE 0DH,0AH
206 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 R4.

;R7, THIS ROUTINE CALLS ASCHEX TO READ THE ASCII STRING AND GEN THE
;TX CHECKSUM.

;CHKSUM: COMPARE THE COMPUTED CHECKSUM WITH THE VALUE CONTAINED IN THE
;STRING TRANSMITTED FROM THE 8550. RUNNING CHECKSUM IS MAINTAINED IN
;RH7. THIS ROUTINE CALLS ASCHEX TO READ THE ASCII STRING AND GEN THE
;TX CHECKSUM.

;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

;set IO: used to reset fcb for sc40

;MANIPULATION TO HEX FORM

;End of loader and subroutines

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

#### Scalars
- **ACK**
  - Value: 00000030
- **NAK**
  - Value: 00000057
- **PROMPT**
  - Value: 0000003E
- **RECEND**
  - Value: 0000000D

#### Strings & Macros
- **ERRMSG**
  - Value: M
- **NYBSUM**
  - Value: M

#### Section
- Section: $BMLOAD, Inpage Relocatable, Size = 0000FE64

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABRTLD</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>MSGBLK</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>0000FED0</td>
</tr>
<tr>
<td>CRFLF</td>
<td>0000094D</td>
</tr>
<tr>
<td>GOODRX</td>
<td>0000090A</td>
</tr>
<tr>
<td>IOBLK</td>
<td>000009F8</td>
</tr>
<tr>
<td>MSGBLK</td>
<td>00000838</td>
</tr>
<tr>
<td>ODLSTR</td>
<td>00000878</td>
</tr>
<tr>
<td>SETIO</td>
<td>0000098C</td>
</tr>
<tr>
<td>TKNAK</td>
<td>0000FE5C</td>
</tr>
<tr>
<td>TKINAK</td>
<td>0000FE5C</td>
</tr>
<tr>
<td>TINAK</td>
<td>0000FE5C</td>
</tr>
<tr>
<td>TKXIN</td>
<td>0000FE54</td>
</tr>
<tr>
<td>TKIN</td>
<td>0000FE5C</td>
</tr>
<tr>
<td>TMMSG</td>
<td>0000080E</td>
</tr>
<tr>
<td>TSTNUM</td>
<td>000009AE</td>
</tr>
<tr>
<td>TSTSTR</td>
<td>000009AE</td>
</tr>
<tr>
<td>WHOVE</td>
<td>00000998</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: 
11: #include "b:stdio.h" /* standard I/O used for file handling */
12: #define ACK '0' /* definitions of: the ACK token */
13: #define NAK '7' /* the NAK token */
14: #define CR 13 /* end-of-line flag */
15: #define TX_RDY 0x01 /* UART transmitter ready flag bit */
16: #define RX_RDY 0x02 /* receiver ready bit */
17: #define COMM_DATA 0x58 /* UART data register port number */
18: #define COMM_STAT 0x59 /* status register port number */
19: 
20: /*
21: infile: pointer for source file (from disk)
22: numchar:
23: subscript for reading characters from disk file into buffer vector
24: outptr:
25: subscript for sending buffer characters to UART
26: argc:
27: command line argument count, used by "C" console processor
28: errcount:
29: number of consecutive reception errors
30: iolinebuffer:
31: array used in moving characters from disk file using standard
32: I/O to UART using system-specific hardware
33: reply:
34: token read from BusMon system to indicate quality of message
35: tx_stat, rx_stat:
36: UART register statuses used during character-send procedure
37: */
38: 
39: FILE *infile,*fopen();
40: int numchar,outptr,argc,errcount = 0;
41: char iolinebuffer[80],reply,tx_stat,rx_stat;
42: 
43: ********************
44: main(argc,argv)
45: char *argv[];
46: 
47: { 
48: }
49: 
50: /*
51: 
52: */
53: open disk file to be sent to the BUSMON system
54: if a NULL is returned, OPEN has failed, exit to CP/M
55: */
56: if ((infile = fopen(*++argv, "r")) == NULL) {
57:    printf("open failure on file %s\n", *argv); exit(99);
58: }
59:
60:   while () {    /* a DO-ALWAYS loop, a la BASIC */
61:        get_reply();    /* get first ACK to commence file transmission */
62:        get_line();    /* read a line from the TEKHEX disk file */
63:
64: #ASM
65:    /* after reading a line from the disk file, kill IRQs for */
66: DI    /* polled serial I/O for both the record output */
67: /* and the REPLY input */
68: #ENDASM
69:
70: tx_line();    /* send record to waiting BusMon unit */
71: get_reply();
72: errcount == 0;    /* zero error count for each record being sent */
73: while (reply != ACK) {    /* if NAK is received: */
74:        retrans_record();
75:        get_reply();
76: }
77: }
78: }
79: #ASM
80:
81: EI    /* bring back IRQs for BDOS/BIOS disk I/O routines */
82:
83: #ENDASM
84:
85: }
86:
87: /*********************************************************************************/
88: get_line()
89: /* function to read <=80 character from the TEKHEX disk file */
90: {
91:     for (numchar =1; numchar <= 80; ++numchar) {    /* for numchar = 1 to 80 */
92:        iolinebuffer[numchar] = getc(infile);    /* read from infile to */
93:          the line buffer */
94:     if (iolinebuffer[numchar] == EOF) {    /*have we reached the end? */
95:        fclose(infile);    /* if so, close the disk file */
96:          exit(0);    /* and back to CP/M... */
97:     }
98:     if (iolinebuffer[numchar] == CR) break;    /* if a CR, exit from the read */
99: }    /* routine and move on */
100: }
101: }
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) {

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 */

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
V01.01-01 (8550) 30-NOV-83/12:00:49

ORG 4000H
LD R0,#7A3AH ;SET UP UART FOR 2400 BAUD,
OUTB 0006H,RH0 ;EVEN PARITY, 1 STOP BIT
OUTB 0006H,RL0 ;7 DATA BITS ON 6510
LD R0,#27H ;'B' SERIAL PORT TO DUMP
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

INIT
ADD R9,R10 ;USE R9 AS WORKSPACE, SAVE R10 FOR NEXT LOAD
ADD R9,R12 ;SET EVEN/ODD ADDRESSES TO BE DUMPED
MOVE LDB RL3,#69 ;GET DATUM
LD RH3,RLO ;COPY DATUM TO WORK ON EACH NYBBLE
AND R5,#FF00FH ;ISOLATE EACH NYBBLE
SRLB RH3,#4 ;REDUCE HIGH DIGIT TO HEX DIGIT
CPB RH3,#9 ;IS DIGIT DECIMAL OR HEX??
JNZ 0000402C ;IF DECIMAL, NO OFFSET NEEDED
ADD RHO,#7 ;IF HEX, ADD 7 TO PUSH ASCI1 CODE TO ALPHA
ADD RHO,#30H ;IN ANY EVENT, ADD ZONE BITS TO MAKE ASCII CHAR
LDB RL4,RL3 ;MOVE FOR OUTPUT TO PROLOG
CALR PUTCRL ;SEND IT OUT
CPB RLS,#9 ;SAME AS ABOVE
JNZ 0000402C ;IF NOT, GET ANOTHER BYTE!!
ADD RLS,#30H ;SAME OFFSET
LDB RL4,RL3 ;SAME ZONE BITS
CALR PUTCRL ;PUT LETTER IN THE MAILBOX
INC R9,#2 ;MOVE TO NEXT BYTE OF THE PROGRAM
OUTB 0004H,RL4 ;SEND DATUM TO THE B-PORT
JR ULE,NOTHX2 ;BACK TO MAIN PROG
JR ULE,NOTHX ;AT THE END OF THE PROGRAM?
JR ULE,MOVE ;IF NOT, GET ANOTHER BYTE!!
JR INIT ;BREAKPOINT SET TO STALL HERE, THEN
JNZ 0000402C ;GO TO INIT FOR NEXT PROM

PUTCHR INB RL6,0005H ;GET STATUS BITS
BIT R6,#0 ;IS UART STILL BUSY?
JNZ 0000402C ;IF SO, WAIT UNTIL CHAR IS SENT...
OUTB 0004H,RL4 ;SEND DATUM TO THE B-PORT
CALR PUTCRL ;BACK TO MAIN PROG

END GO ;THAT'S ALL FOLKS!!!