A PROGRAM DOWNLOADER AND OTHER UTILITY SOFTWARE
FOR THE DATA C 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 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.
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 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.

Data Record
/aaaaabbbacdd...dddc<br>
LOAD ADDRESS BYTE COUNT CHKSUM DATA BYTES 2nd CHKSUM RECORD TERMINATOR

Terminator Record
/xxxxxooac<br>
LOAD ADDRESS ZERO-CHKSUM LENGTH RECORD

Abort Record
//Abort message text
Figure 2. Sample TEKHEX file.

Figure 3. Z8000 loader outline.
APPENDIX A

LISTING 1

ASM  Z8001/Z8002
V01.01-01 (8550)

<table>
<thead>
<tr>
<th>No.</th>
<th>ASSEMBLY Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>; DATA Bus Monitor:</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>; Loader for Z8000 Processor Interface to Data Bus System</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>; Author: S.M. Novack 2 Sept 83</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>; REV 22 Nov 83: Includes Error Handler- Exits to Monitor</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>; Macro Definitions Here:</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>MACRO NYBSUM</td>
</tr>
<tr>
<td>7</td>
<td>LDB RL2, RH2 ; Transpose Hex Digits</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SRA RL2, #04 ; Make HOD the LOD</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>AND R2, #0FOH ; Mask Off HO Bits</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ADOB RH2, RL2 ; Add Nybbles W/O Carry</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>; RH2 Holds Nybble Checksum, Transferred to RH7</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ENDM</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>; This Macro Performs a Test for Checksum Errors, If &gt;5 the Load is Aborted</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Macro ERRMSG</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>INC R13       ; Count New Error Occurrence</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>CP R13, #5 ; Reach Max# of Errors?</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>JR UGT, ABRTLD ; Too Many Errors- Return to Monitor</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>SET R12, #01 ; Set 'Old String, Repeat' Flag</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>LDB ACKBUF, NAK ; Ready Bad TX MSG</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>JR NEWSTR     ; Request Repeat of MSG and Clear INBUF</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>ENDM</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>; I/O String Buffer Definitions, Must Be Ordered in Ram</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>FEOO R        ; ORG ofEOOM</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>; I/O String Buffer Definitions, Must Be Ordered in Ram</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0000FE00 4    ; Three Byte Buffer to Handshake With 8550 During File TX</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0000FE04 50   ; 80 Byte Buffer for Receiving Tekhex Files</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>0000FE54 8    ; Tekhex Block 8</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0000FE5C 8    ; Teknak Block 8 ; I/O FC Block (Workspace)</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>; Constant Definitions:</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>80E R         ; ORG ofEOH</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>3E PROMPT EQU 3EH ; 8550 Handshake Prompt Char</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>30 ACK EQU 30H ; MSG Received Token</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>37 NAK EQU 37H ; MSG Not Received Token</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>2F RECEND EQU 2FH ; Slash Char Used to Delimit Tekhex Records</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>; Count New Error Occurrence</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>CLR R12       ; Only 3 of 4 Bytes Used</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>CLR AOBUF     ; In Handshake Sequence</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>CLR AOBUF+2</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>LDB AOBUF+1, RECEND ; Ready String for 000D</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>; Tek Handshake</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>; Beginning of Loader Routine;</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>; Consult ZMON.DASSY and DUMP to Determine Actual Addresses</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>; Beginning of Romainable Routines, All Jumps Relative, Only</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>; RAM References Are Absolute for Duration of Loader Operation</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>; Note# of Bytes in String Must Be Even</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>53544420</td>
<td>TMMSG ASCII 'STD TEKHEX LOADER'</td>
</tr>
<tr>
<td>47</td>
<td>54454848</td>
<td>; Note# of Bytes in String Must Be Even</td>
</tr>
<tr>
<td>48</td>
<td>4558204C</td>
<td>4F144445</td>
</tr>
<tr>
<td>49</td>
<td>5220</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>00000820 4D08FE00 R</td>
<td>INTCOM CLR AOBUF ; Only 3 of 4 Bytes Used</td>
</tr>
<tr>
<td>51</td>
<td>00000824 4D08FE02 R</td>
<td>CLR AOBUF+2 ; In Handshake Sequence</td>
</tr>
<tr>
<td>52</td>
<td>00000828 4C05FE01 R</td>
<td>LDB AOBUF+1, RECEND ; Ready String for 000D</td>
</tr>
<tr>
<td>53</td>
<td>0000082E 4C05FE02 R</td>
<td>LDB AOBUF+2, PROMPT ; Tek Handshake</td>
</tr>
<tr>
<td>54</td>
<td>3E3E</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>00000834 8DC8</td>
<td>CLR R12 ; Flag: 0=New String, 1=Rep'it of Last String</td>
</tr>
<tr>
<td>56</td>
<td>00000836 0F56</td>
<td>CALR SETIO ; Set Up For Input Operations</td>
</tr>
<tr>
<td>57</td>
<td>00000838 8CAB</td>
<td>NEWSTR CLRBL R2L ; (R2)=0 For Zapping</td>
</tr>
<tr>
<td>58</td>
<td>0000083A 210AO050</td>
<td>LD R10, #80 ; Number of Bytes to Be Zapped</td>
</tr>
<tr>
<td>59</td>
<td>0000083E 2109FE04 R</td>
<td>LD R9, #100BUF</td>
</tr>
<tr>
<td>60</td>
<td>00000842 729A000</td>
<td>ZAPWRD LDB R9(R10), R2L ; Zero Out INBUF (I Hope...)</td>
</tr>
<tr>
<td>61</td>
<td>00000846 8ABD</td>
<td>DEC R10</td>
</tr>
<tr>
<td>62</td>
<td>00000848 EEFC</td>
<td>JR NZ, ZAPWRD</td>
</tr>
<tr>
<td>63</td>
<td>0000084A 2101FE05C R</td>
<td>OUTMSG LD R1, RKL</td>
</tr>
<tr>
<td>64</td>
<td>0000084E 7F00</td>
<td>SC #0 ; Output Prompt Via Monitor Routine</td>
</tr>
</tbody>
</table>

121
.asm
Z8001/Z8002
VOL.01-01 (8550)

01-DEC-83/08:48:48

65; HOPEFULLY WITH A SERIAL LINE DEDICATED TO THE Z8K-TEK INTERFACE
66; THERE WON'T BE ANY JUNK BEFORE THE PROMPT AND THE FIRST HEX RECORD.
67; UNTIL THAT SERIAL LINE IS ESTABLISHED, WE'LL SHARE THE ONE WITH
68; Z8K CONSOLE DEVICE AND PROVIDE FOR GETTING RID OF ANY BAD DATA
69; WE MAY HAVE TO READ. ONCE A SEPARATE SERIAL LINE IS AVAILABLE, WE CAN
70; DISCARD THE 'FIND START-OF-RECORD' ROUTINE
71; IDLE 8550 BEGINS TO TX AFTER THE PROMPT SENT BY OUTMSG
74;
75 00000850 2101FE54 R
76 00000854 7FO0
77 00000856 7608FE04 R
78 0000085A 0808FE.54 R
79 0000085E E605
80 00000860 OC812F2F
81 00000864 E606
82 00000866 A980
83 00000868 EEF8
84 0000086A 4DOBFEO0 R
85 00000870 EBE3
86 00000872 80C4
87 00000874 EEOI
88 00000876 80D8
89 00000878 DF84
90 0000087A 80A2
91 0000087C OF97
92 0000087E AO42
93 00000880 AO2A M
94 00000882 B2A9FCFC M
95 00000886 07020FOF M
96 00000888 A980
97 0000088A AO2A M
98 0000088C 80A2 M
99 0000088E AO42
100 00000890 B2A9FCFC M
101 00000892 NAK
102 00000894 A0.42
103 00000896 AO2A M
104 00000898 B2A9FCFC M
105 0000089A 07020FOF M
106 0000089C 80A2 M
107 0000089E AO42
108 000008A0 A980
109 000008A2 OF97
110 000008A4 AO46
111 000008A6 AE27
112 000008A8 80A2 M
113 000008AA B2A9FCFC M
114 000008AC 07020FOF M
115 000008AE AO42
116 000008B0 80A2 M
117 000008B2 B2A9FCFC M
118 000008B4 07020FOF M
119 000008B6 AO42
120 000008B8 A980
121 000008BA OF97
122 000008BB AO42
123 000008BD A980
124 000008BE OF97
125 000008C0 AO42
126 000008C2 A980
127 000008C4 OF97
128 000008C6 AO42
129 000008C8 B2A9FCFC M
130 000008CA 07020FOF M
131 000008CE AO42
132 000008CF A980
133 000008D0 OF97
0037

122
; ON TO THE BYTE COUNT
CALR ASHEX ; GET # OF BYTES IN MSG
LDB RH2,RH4 ; ADD IT TO CHKSUM
M
LDB RL2,RH2 ; TRANSPOSE HEX DIGITS
; MAKE HO THE LOD
SRAB RL2,#04
AND R2,#0FOFH ; MASK OFF HO BITS
ADD RH2,RL2 ; ADD NYBBLES W/O CARRY
LDB RL2,RH2 ; SAVE # OF DATA BYTES IN HEX FOR RAM LOAD
INC RH8 ; SET CHAR CNT FROM STRING
CALR CHKSUM ; TEST 1ST BYTE-CHKSUM
LDB @R6,RH4 ; IN A STRING
DECB RL7 ; JR NE,HXLOAD
; RECORD LOAD COMPLETE
INC R8 ; RECORD REPEAT OF MSG AND CLEAR INBUF
CALR CHKSUM ; PRODUCE AND COMPARE SECOND BYTE-CHKSUM
LDB RH2,RH4 ; SENT TO CHKSUM ACCUM
M
LDB RL2,RH2 ; TRANSPOSE HEX DIGITS
; MAKE HO THE LOD
SRAB RL2,#04
AND R2,#0FOFH ; MASK OFF HO BITS
ADD RH2,RL2 ; ADD NYBBLES W/O CARRY
LDB RL2,RH2 ; ANOTHER DIGIT TO BE SUMMED
LD8 @R6,RH4 ; STORE MACHINE CODE
LDB @R6,RH4 ; STORE BYTES!
INC R8 ; IN A STRING
CALR CHKSUM ; READY BAD TX MSG
LDB @R6,RH4 ; SEND IT TO TEK CONSOLE
CALR CHKSUM ; SEND IT OUT
RETURN TO Z8000 MONITOR
; TX MODE FOR SC#0
; NOT USED
ASHEX ASCII '//' ; ADDRESS OF ERROR MSG
; # OF CHARS IN STRING TO BE TX'd
ERROR LIMIT EXCEEDED, LOAD IS ABORTED' ; SELF-EXPLANATORY

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
; CALR TSTNUM ;ADJUST ASCII IF CHR IS A-F
; ANDB RL4,#OFH ;MASK OFF ZONE BITS
; SLAB RL4,#40 ;LSBITS BECOME MSBITS
; INC R8 ;NEXT DIGIT
; LDB RL4,#R8 ;GET IT
; CALR TSTNUM ;ADJUST ASCII IF CHR IS A-F
; ANDB RL4,#OFH ;PROCESS IT
; ORB RH4,RH4 ;FORM COMPLETE BYTE OF DATA
; RET ;GO HOME

; $0000096A 9E08

; 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 000F
; CHKSUM: CALR ASCHEX ;GET 1ST BYTE-CHECKSUM
; CPB R7,RH7 ;COMPARE CALCULATED AND GIVEN CHECKSUMS
; EXIT RET ;REQUEST ANOTHER TX OF THE STRING IF NEEDED

; CHKTMR: SCANS THE INPUT BUFFER FOR A BYTE COUNT OF ZERO. USES ASCHEX
; TRANSALTE 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

; $0000096E A407
; CHKTRM: LD RIO,R8 ;SAVE CURRENT POSITION IN STRING
; IN(R8),#5 ;AIM AT IST CHR OF BYTE COUNT
; CALR ASHEX ;FORM BYTE COUNT
; LD R8,RIO ;RECOVER ORIGINAL POINTER
; TESTB RH4 ;IS DATA STRING LENGTH ZERO?
; RET NE ;NO, GO BACK AND FINISH PROCESSING

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

; $0000097E 4D05FE00 R 0030
; SETIO: USED TO RESET FC8 FOR SC#O

; $00000984 2101FESC R
; SETIO: LD R1,#TSTNUM ;READY THE MSG
; SC #0 ;SIGNAL TRANSFER END TO HOST COMPUTER
; SC #3 ;LOAD COMPLETED, RETURN TO MONITOR

; $00000986 7F00
; TSTNUM: CORRECTS ASCII CHARACTERS FROM A TO F TO ALLOW
; FOR SIMPLE; MANIPULATION TO HEX FORM

; $00000988 7F13
; TSTNUM: CPB RL4,#39H ;IF 0-9, NO CORRECTION NEEDED
; JR LE, ISNUM ;ELSE ADO OFFSET OF 9 TO PRODUCE USEABLE LO NYBBLE
; ADDB RL4,#9 ;ELSE ADD OFFSET OF 9 TO PRODUCE USEABLE LO NYBBLE
; ISNUM RET ;BACK TO ASCHEX

; END OF LOADER AND SUBROUTINES
### Scalar Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>00000030</td>
<td>NAK 00000037</td>
</tr>
<tr>
<td>REDMK</td>
<td>0000002F</td>
<td>PROMPT 0000003E</td>
</tr>
</tbody>
</table>

### String Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERMSG</td>
<td>M</td>
</tr>
<tr>
<td>NYBSUM</td>
<td>M</td>
</tr>
</tbody>
</table>

### Section Information

- Section = $BMLLOAD, Inpage Relocatable, Size = 0000FE64

### assembler directives

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABRTLD</td>
<td>00000914</td>
</tr>
<tr>
<td>CHKSTR</td>
<td>00000972</td>
</tr>
<tr>
<td>GETSTR</td>
<td>00000850</td>
</tr>
<tr>
<td>INTCM</td>
<td>00000820</td>
</tr>
<tr>
<td>MSGBLK</td>
<td>0000081C</td>
</tr>
<tr>
<td>SEEK</td>
<td>0000085A</td>
</tr>
<tr>
<td>TKHIXIN</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>I0BLK</td>
<td>0000099E</td>
</tr>
<tr>
<td>NEXSTR</td>
<td>00000838</td>
</tr>
<tr>
<td>OL0STR</td>
<td>00000878</td>
</tr>
<tr>
<td>SETIO</td>
<td>0000098C</td>
</tr>
<tr>
<td>TKINAK</td>
<td>0000FE5C</td>
</tr>
<tr>
<td>WMOVE</td>
<td>00000998</td>
</tr>
</tbody>
</table>

### Summary

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

I:

/*
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 Ox01 /* UART transmitter ready flag bit */
16: #define RX_RDY Ox02 /* receiver ready bit */
17: #define COMM_DATA Ox58 /* UART data register port number */
18: #define COMM_STAT Ox59 /* status register port number */
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:
41: FILE *infile,*fopen();
42: int numchar,outptr,argc,errcount = 0;
43: char iolinebuffer[80],reply,tx_stat,rx_stat;
44:
45: /**************************************************************************/
46:
47: main(argc,argv)
48: char *argv[];
49:
50: { 
51: }
52: /*
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 */

    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 IQRs 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 IQRs for BDOS/BIOS disk I/O routines */
        #ENDASM

    EI  /* bring back IQRs 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 */

    /*****************************************************************************/
get_line()
/*****************************************************************************/

/*****************************************************************************/
get_line()
/*****************************************************************************/

/*****************************************************************************/
get_line()
/*****************************************************************************/

/*****************************************************************************/
get_line()
/*****************************************************************************/
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:.hstack()
117:DOCTYPE  */
118: get_reply()
119: /* receives reply token from the BusMon unit after tx_line is performed */
120: {
121: while (((rx_stat = in(COMM_STAT)) && RX_RDY) != RX_RDY) {}
122: /* idle until UART receiver is ready */
123:  reply = in(COMM_DATA);  /* get ACK/NAK token */
124:  if (reply != ACK) {
125:    if (++errcount > 5) load_error(); /* if too many errors, exit */
126:  }
127: }
128: /****************************************************************************
129: retrans_record()  
130: */
131: retrans_record()
132: /* retrans by another name, done for improved legibility */
133: /* since numchar is not destroyed by tx_line, this offers a very convenient */
134: /* way to retransmit the same line of characters */
135: {
136: tx_line();
137: }
138: /****************************************************************************
139: load_error()*/
140: load_error()
141: /* only if five successive load errors are reported by the BusMon */
142: {
143: /* EI */ /* restore IRQs for standard I/O functions */
144: EI
145: printf("error limit exceeded, load operation aborted\n");
146: fclose(infile); /* close the disk file */
147: exit(88); /* return to CP/M with error code 88 */
148: }
APPENDIX C

LISTING 3

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

ORG 4000H
LD R9,#7A3AH ;SET UP UART FOR 2400 BAUD,
OUTB O006H,RHO ;EVEN PARITY, 1 STOP BIT
OUTB O006H,RL0 ;7 DATA BITS ON 6510
LD R0,#27H ;'B' SERIAL PORT TO DUMP
OUTB O007H,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
YOU PROM YOU ARE PROGRAMMING

INIT 
LD R9,R10
;USE R9 AS WORKSPACE, SAVE R10 FOR NEXT LOAD
ADD R9,R12
;SET EVEN/ODD ADDRESSES TO BE DUMPED
MOVE LD8 RL3,#89
;GET DATUM
LD8 RL3, @R9
LDB RH3,RL3
;IS DIGIT DECIMAL OR HEX??
CPB RH3,#9
;IF DECIMAL, NO OFFSET NEEDED
ADD8 RH3,#7
;IF HEX, ADD 7 TO PUSH ASCII CODE TO ALPHA
LDB RL4,RL3
CALR PUTCRL
CPB RL3,#9
;SAME AS ABOVE
ADD8 RL3,#7
;SAME OFFSET
LDB RL3,#30H
SAME ZONE BITS
LDB RL4,RL3
PUT LETTER IN THE MAILBOX
CALR PUTCRL
;HERE COMES THE POSTMAN
INC R9,#2
;MOVE TO NEXT BYTE OF THE PROGRAM
CP R9,R11
;AT THE END OF THE PROGRAM?
JR R9, move
;IF NOT, GET ANOTHER BYTE!!
JR INIT
;BREAKPOINT SET TO STALL HERE, THEN
GO TO INIT FOR NEXT PROM
OUTB O004H,RL4
;SEND IT OUT
RET
END GO
;THAT'S ALL FOLKS!!