NASA TM-87649

NASA-TM-87649 19860009858

NASA Technical Memorandum 87649

IMPROVED MEMORY LOADING TECHNIQUES FOR THE TSRV DISPLAY SYSTEM

Wesley C. Easley
William A. Lynn
David G. McLuer

JANUARY 1986

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

```
ENTER:
     SELECT     RN/NASA-TM-86832
     TERM IN SELECT COMMAND NOT IN DICTIONARY
================================================================================
     SELECT     RN/NASA-TM-87652
     TERM IN SELECT COMMAND NOT IN DICTIONARY
================================================================================
 22      1       1 RN/NASA-TM-87649
           DISPLAY 22/6/1
      86N19329**   ISSUE 10   PAGE 1537   CATEGORY 9   RPT#: NASA-TM-87649 NAS
      1.15:87649   86/01/00  34 PAGES   UNCLASSIFIED DOCUMENT
UTTL: Improved memory loading techniques for the TSRV display system
AUTH: A/EASLEY, W. C.;  B/LYNN, W. A.;  C/MCLUER, D. G.
CORP: National Aeronautics and Space Administration.  Langley Research Center,
      Hampton, Va.    AVAIL.NTIS
SAP:  HC A03/MF A01
COI:  UNITED STATES
MAJS: /*CHANNELS (DATA TRANSMISSION)/*DATA STORAGE/*DISPLAY DEVICES/*FLIGHT
      CONTROL/*MEMORY (COMPUTERS)
MINS: / COMPUTER PROGRAMS/ MAGNETIC STORAGE/ MAGNETIC TAPES/ MICROPROCESSORS
ABA:  Author
```

TABLE OF CONTENTS

Summary

The efforts described in this report were performed to improve the memory loading methods for the display computer used in the TSRV experimental aircraft at Langley Research Center. Enhanced operation and improved reliability of the display system were achieved without costly changes to its internal electronics.

Display software to be loaded into memory consists of a BOOTSTRAP, a short control program, and larger applications programs. Previously, the BOOTSTRAP was loaded via paper tape and applications software was loaded by means of a cassette tape, methods which are outdated and unreliable. Paper and cassette tape generation and duplication are significant problems. Also, continuously increasing difficulty is encountered in maintaining transport mechanisms for these media.

The BOOTSTRAP data are in the form of ASCII characters punched along the tape. In this effort the same characters were written into successive addresses of an EPROM which was then installed in the display control unit. Then, rather than move the characters past the optical reader via a punched tape, a counter was implemented which stepped through the EPROM addresses in turn and presented identical data to latches in the display loader circuitry. Thus, the existing electronic latches could be used resulting in no design changes in the display loader circuitry.

Application software loading via cassette utilized a display input data bus for transferring an image from tape to memory. The improved technique developed during this effort consists of transferring the same memory image to a VT-180 disk and loading it via a VT-180 terminal installed in the TSRV as part of the recent upgrade. Software was written for the VT-180 to duplicate the function of the cassette tape loader. Thus, no changes were required in either loader circuitry or the display internal loader software (a component of the BOOTSTRAP). In addition, the same input data bus is used for transfer of the disk resident image into memory.

List of Figures

I. INTRODUCTION

The initial upgrade of the TSRV experimental system did not contain a new avionics display system. Instead, the old Advanced Electronic Display System (ADEDS) was retained with the intent to upgrade a color system later.

Several changes in the operation and support of the ADEDS were required with the new experimental system. Significant among these were the required interface to the new DATAC bus and a different method of memory loading due to elimination of cassette tape units. Both of these changes were accomplished by in-house design and development efforts. The DATAC interface utilizes a Display Interface Unit (DIU), and this report describes the new memory loading techniques.

ADEDS memory loading is a two step process, first a BOOTSTRAP, then the applications programs. BOOTSTRAP loading was previously accomplished using paper tape while applications software loading utilized the discontinued cassette tape unit. New methods developed use an EPROM for BOOTSTRAP loading and a VT-180 microprocessor terminal for loading the applications programs from disk.

The ADEDS system is a unique prototype developed using technology of the late 1960's for short term use and not much documentation was ever prepared. In addition, the system contains many unique parts which are no longer available. Therefore, modifications mostly require functional redesign which proves very difficult since most hardware and software functions must be determined by purely empirical means (i.e., electrical schematics and assembly software listings prepared years earlier with few narrative comments). Thus, it is important that all changes be kept to a minimum with any new external interface units requiring a design which meets the existent ADEDS interface. This philosophy was followed in developing the techniques described herein.

II. LIST OF ABBREVIATIONS

ADEDS	Advanced Electronic Display System
ACD	Analysis and Computation Division, LaRC
ASCII	American Standard for Computer Information Interchange
BDOS	Basic Disk Operating System, CP/M
CP/M	Control Program for Microcomputers
DATAC	Digital Autonomous Terminal Access Communication
DTU	Data Translation Unit
EOF	End of File
GS	Glide Slope
H	Hexadecimal Number
ILS	Instrument Landing System
INS	Inertial Navigation System
LOC	Localizer
LSB	Least Significant Bit
MLS	Microwave Landing System
MSB	Most Significant Bit
MTU	Magnetic Tape Unit
RAD ALT	Radio Altitude
RTN	RETURN
SPBP	Split Phase Bipolar (Data Modulation)
TSRV	Transport Systems Research Vehicle
UART	Universal Asynchronous Receiver Transmitter

III. ADEDS GENERAL DESCRIPTION

The Advanced Electronic Display System (ADEDS) in the Research Flight Deck of the TSRV aircraft is required to support all research flying. It is the means through which real time attitude and navigation information is presented to the research pilot. Reference 1 contains a description of the ADEDS and its interfacing in the TSRV upgraded system. Figure 1 outlines the general nature of the ADEDS interfaces.

ADEDS has no terminal interface, due to limitations in memory and computing time, and adding such would require almost total redesign of the memory addressing methods. A system control unit integral to ADEDS contains a paper tape transport and keypad. It serves as a crude terminal and is the prime means of operational troubleshooting and memory loading. The improved memory loading techniques described herein using the VT-180 terminal still interfaces with the integral system control unit as did the paper tape loader.

IV. ADEDS BOOTSTRAP LOADING

A. Introductory Discussion

1. General Description. The ADEDS memory is a magnetic core consisting of 8000 locations, each 24-bits. To make the ADEDS operational a short BOOTSTRAP must first be loaded into the lower memory locations. Logic residing in firmware in the ADEDS computer is used for this BOOTSTRAP loading. Principally, the BOOTSTRAP is a small operating system with very limited capability whose main functions are configuration of the main loader program (a software unit which allows the remainder of memory to be loaded with varied and larger applications programs) and activation of the keys on the system control unit panel. Paper tape has always been used for BOOTSTRAP loading, and although this method has been reasonably reliable, aging irreplaceable mechanical parts in the ADEDS tape transport mechanism are making it less so. Also paper tape creation and use are no longer routine. Therefore, to help insure usability of the ADEDS until it is replaced by a new color display system, it has been necessary to develop a backup method for BOOTSTRAP loading.

   Development of a BOOTSTRAP loading technique using a disk file was considered but not chosen due to lack of physical room and power for the RS-232 conversion circuitry required. Instead, it was decided to place the BOOTSTRAP data into an EPROM and configure its output to interface with the ADEDS loading circuitry existing beyond the tape output. Development of this method is described in subsequent sections of this report. The resulting technique has not been installed, as of the date of this report, due to extended downtime required; but is has been shown to function and will be available should the paper tape loader become unusable.

2. Tape Loading Technique. BOOTSTRAP data on the paper tape consists of a series of 7-bit characters, each with even parity, punched along the length of the tape. This is a byte serial, bit parallel

arrangement. In use the tape is pulled past an optical reader and when a character position is in front of the reader, light penetrates the punched holes and activates photodiodes whose outputs are in turn converted to TTL data for latching into registers. In this application a punched hole is a one and blank tape is a zero.

B. EPROM Replacement of Tape

The method chosen to replace the tape consists of writing an exact image of the characters on the tape into sequential locations of an EPROM. Then, instead of moving a physical tape, a counter is used to sequentially step through the EPROM addresses placing the data into the same ADEDS latches that received the TTL version of the tape data. In this way no circuit changes were required in the ADEDS loader, and the manual key that starts the tape transport now is used to start the address counter.

C. Creation of EPROM Tape Image

For creation of an EPROM image of the BOOTSTRAP tape, it was necessary to transfer the tape file into a VT-180 disk file which could, in turn, be written into the EPROM. At the time the task was performed the only reliable means of accomplishing this was to read the tape and dump the contents into a file. Unfortunately, no means was available to read the paper tape directly into the VT-180, but a PDP 11/55 with such a reader is part of the ground based TSRV support equipment. Thus, the tape could be read into a file on the 11/55 and then downloaded to the VT-180. Two significant problems existed in the resulting VT-180 file: (1) the 11/55 inserted newline characters (carriage return/line feed) at regular intervals in its file and (2) the parity bit of each tape character (which was actually punched on the tape) was eliminated by the transfers. Thus, the file required correcting before it was an exact tape image. To accomplish this, a VT-180 file utility program was written which performs four functions including stripping newline characters and adding proper parity bits to each file byte. The program was written in the "C" language to run under the CP/M operating system.

Detailed description of the "C" language is contained in references 2 and 3. A brief discussion is included here to aid in understanding the software described in this report. In "C" statements are terminated by a semicolon, and may be grouped into executable blocks by braces "{" and "}." Arrays are indicated by their name and index enclosed in square brackets (i.e., "list [5]"). Comments begin and end with the character pairs "/*" and "*/." Test conditions for "while" and "if" loops are enclosed in parentheses, with any non-zero value being considered true. If the test is an assignment statement, (a single "=") the assignment is made and then the value assigned is tested for non-zero. For example:

```
if(a=b){
/* group of statements */
}
```

first assigns the value of "b" to the variable "a," then executes the group of statements enclosed by the braces if "a" is non-zero. Relational operators

"==" (equality) and "!=" (inequality) perform the test but do not modify either operand. For example:

```
if(a==b){
*/group of statements */
}
```

leaves "a" and "b" unchanged.

The operator "++" causes a variable to be incremented after use. For example:

```
while ( a[i++] {
/* group of statements */
}
```

will use the index "i" to choose elements of the array "a[ ]" and increment the value of "i" after each use, thus stepping through the array and executing the group of statements until an element is found whose value is not zero.

The source listing and functional description of the file utility written for this effort follows. Line numbers have been added as an aid in description and are not part of the actual source file. (Line numbers are not used in "C.")

```
 1: #include "libc.h"
 2: main()
 3: {
 4: FILE *fopen(), *fo;
 5: char k, m, ct, sfile [20], dfile[20];
 6: FILE *fopen(), *fp
 7: int ne, no, nt, dc, sc, i, j, p, nb, nnl;
 8: i = 0;
 9: j = 0;
10:
11: printf("\n\n\t\t     MENU");
12: printf("\n  S  Strip 'new-line' chars from file.");
13: printf("\n  E    Create a file with EVEN parity.");
14: printf("\n  O    Create a file with ODD parity.");
15: printf("\n  C    Check parity of all characters in a file.");
16: printf("\n  X    Exit to CPM.");
17: printf("\n\nEnter your selection.  (Upper or lower case ok.)  ");
18:
19: p = getchar();getchar();
20:
21: if(p == 'X' || p == 'x') exit();
22: else{
23: if(p == 'C' || p == 'c'){
24:
25: /*Name the source file*/
26: printf("\n\n Enter the filename to undergo parity checking.\n");
27: while((sfile[i++] = get char ()) != \n
28:
29: /*Check for file opening problems*/
```

```
30: if((fp = fopen(sfile, "r"))  == NULL){
31:         printf("\n Can't open source file.");}
32: else{
33:         nb = 0; ne = 0; no = 0, nt = 0
34:         while((sc = getc(fp)) != EOF){
35:         ++nb;
36:         k = 0; m = 1;
37:         for(ct = 0; ct <= 7; ++ct){
38:         if(m & sc){k++;}
39:         m = m << 1;  } /*End of for*/
40:         if(k & 1)++no; else ++ne;
41:         k = 0; m =1; }      /*End of while */
42:         fclose(fp);
43:         printf("\nTotal # bytes in file %s is %d", sfile, nb);
44:         printf("\n # bytes found with EVEN parity:  %d", ne);
45:         printf("\n # bytes found with ODD parity:  %d\n\n", no);
46:         nt = no + ne;
47:         if(nt != nb){
48:         printf("\n\n ERROR; Total bytes not equal total of");
49:         printf("\n EVEN parity bytes + ODD parity bytes !!"); }
50:         }               /*End of else*/
51:         }          /*End of it */
52: else{
53:
54: /*Name the source file*/
55: printf("\n Enter the source filename.");
56: while((sfile[i++] = getchar()) != "\n";
57:
58: /*Name the destination file*/
59: printf("\n Enter the destination filename.");
60: while((dfile[j++] = getchar()) != "\n";
61:
62: /*Check for file opening problems*/
63: if((fp = fopen(sfile, "r")) == NULL){
64:         printf(=\n Can't open source file.");
65:         }
66:
67: else if((fo = fopen(dfile, "w")) == NULL){
68:         printf("\n Can't open source file.");
69:         }
70:
71: else{ if(p == '0' ||  p == 'o'){
72:         nb = 0;
73:         while((sc = getc(fp)) != EOF){
74:         ++nb
75:         dc = opar(sc);
76:         putc(dc, fo);} /*End of while */
77:         fclose(fp, fo);
78:         }
79:
80:         if(p == 'E' ||  p == 'e'){
81:         nb = 0;
82:         while ((sc = getc(fp)) != EOF {
83:         ++nb;
```

```
 84:          dc = epar(sc);
 85:          putc(dc, fo;}
 86:          fclose(fp, fo);
 87:          }
 88:
 89:          if(p == 'S' || p == 's'){
 90:          nb = 0;nnl = 0;
 91:          while((sc = agetc(fp)) != EOF){
 92:          ++nb;
 93:          if(sc == "\n")++nnl; else {aputc(sc, fo);}
 94:          }           /*End of while*/
 95:          fclose(fp, fo);
 96:          printf("\n # of newline chars found = %d, nnl);
 97:          }          /*End of if*/
 98:          }           /*End of else*/
 99: printf("\n\nThe source file was:  %s", sfile):
100: printf("\nTotal bytes in source file was %d \n ", nb);
101: }
102: }
103: }
104:
105: /*Even parity setting function*/
106: epar(s)
107: int s;
108: {
109: char m, n, k;
110: s = s & 0x7f;
111: m = 1;
112: k = 0;
113:
114: for(n = 0; n < 7; ++n){
115: if(m & s){
116: k++;}
117: m = m << 1;
118: }
119: if(k & 1)
120:          s = s | 0x80
121:          return(s);
122: }
123:
124: /* Odd parity setting function*/
125: opar(s)
126: int s;
127: {
128: char m, n, k;
129: s = s & 0x7f;
130: m = 1; k = 0
131:
132: for(n = 0; n < 7; ++n){
133: if(m & s) ++k;
134: m = m << 1;}
```

```
135:
136: if (k & 1)s = s; else{s = s   0x80}
137:         return(s);
138: }
```

D. Software Description

This program is a small utility which performs four menu selectable operations on a file: (1) strip newline characters, (2) create a file with even parity, (3) create a file with odd parity, and (4) check the parity of all characters in a file. The menu is produced upon program execution by lines 11 through 17.

After menu selection is made, the program (line 19) fetches the entered character, tests to determine what it is, and branches accordingly. If an X was entered, the program immediately exits to the CP/M operating system. Otherwise, it continues checking using lines 23, 71, 80, and 89 to determine what function is desired. The first operation needed herein is stripping newline characters from the file, the S option in the menu. After S is selected, the source and destination filenames will be requested by lines 55 and 59. (A new destination filename is requested since a new file is created with newline characters missing rather than overwriting the file operated on.) All further tests will be false until line 89 is reached. The true condition here will cause the program to execute lines 90 through 96. The while statement of line 91 will be true until the end of file (EOF) character is encountered, thus, resulting in a loop through the file. Upon reaching the EOF, the program will close both source and destination files and exit to CP/M (line 95). In addition to writing the new file to disk, the program indicates the number of newline characters found (line 96), verifies that the desired source file was used (line 99), and states the total number of bytes found in the source file (line 100).

The file resulting from the newline stripping operation was in turn operated upon with the same program, this time with E selected from the menu since even parity is required for a proper tape image. (It was determined by empirical examination that the ADEDS BOOTSTRAP tape contained even parity punched into each character.) Upon selection of E, the program will again request filenames in lines 55 and 59. Then the test in line 80 will be true resulting in execution of lines 81 through 86. Once again the while statement of line 91 will cause a loop through the entire file. Line 84 calls the function EPAR for each file character. This function is shown in lines 105 through 121 and contains logic necessary to count the ones in each character then configure the most significant bit as required for even parity. The parity correct byte is returned by line 121 to the calling program. As before when the EOF character is encountered in line 82, the files will be closed and execution will end. The destination file from this operation will be on the VT-180 disk and is an exact image of the original tape file. It can then be written directly into an EPROM.

The "C" program also will generate a file with all odd parity as well as check the parity of all bytes in a file. These two functions were not used in the present application but give the software a slightly increased capability.

E. Bootstrap Loader Hardware

Figure 2 contains a schematic of the circuitry designed to load the EPROM BOOTSTRAP data into the ADEDS. A counter composed of U 1, U 2, U 3, and U 4 steps through the EPROM addresses with the contents of each location being read into U 7, an inverting buffer. The output from U 7 goes into the same data latches in the ADEDS which receive the inverted tape data. An inverting buffer is necessary because the photodiode output of the optical tape reader, in addition to being converted to TTL, is inverted before being placed into ADEDS data latches. Thus, the EPROM data must be inverted after being read since it was not complemented before writing.

Buffer U 7 is a three state device and is used to perform another function required to make the EPROM data match that from the tape. Between each character punched in the tape is a section of blank tape which will yield all zeros at the reader output, or all ones when inverted and presented to the ADEDS latches. Thus, alternate bytes from the EPROM loader must have all bits set yielding FFH as the value. Rather than write FFH at alternate locations in the EPROM, the tri-state feature of U 7 is used. To accomplsh this, the least significant bit of the counter, pin 3 of U 1, is inverted by U 5 then fed into pins 1 and 19 (the enable controls) of U 7. When disabled U 7 will have all output bits high, thus, presenting FFH to the ADEDS data latches. Since the LSB of the counter toggles with every counter step, FFH will appear as data for alternate bytes. The next most significant bit of the counter, pin 2 of U 1 is the least significant bit of the EPROM address, pin 8 of U 6. This bit will toggle only on alternate step counter pulses and will occur coincident with the enabling of U 7. Thus, each time the EPROM address is incremented, U 7 will present its inverted data byte to the latches in the ADEDS loader circuitry.

Control of the circuitry of figure 2 is accomplished by the clock pulses and the LOAD BOOTSTRAP gate shown. The clock is obtained from the ADEDS loader circuitry. Its frequency is 1 Khz and its use assures synchronization with the existent loader data handling logic. The LOAD BOOTSTRAP signal is activated manually via a key on the ADEDS control unit which starts the tape transport in the original hardware. Here this signal enables loader circuitry so that BOOTSTRAP data will be accepted and resets the counter to all zeros, thus, directing it to the first EPROM address. Simultaneously, the light emitting diode D1 lights because pin 6 of U 4, the counter most significant bit, (MSB) goes low, thus, forward biasing DI. This serves as indication that BOOTSTRAP loading is in progress. When the counter has finished stepping, pin 6 of U 4 will go high, thus, resetting the counter and removing the forward bias from D1 and turning it off. This is the indication that BOOTSTRAP loading is complete.

When the above described BOOTSTRAP loading functions have been accomplished, the ADEDS applications software loader is in place, the key switches on the control unit are activated, and the system is ready to accept memory image data representing applications software, loading of which is described in following sections of this report.

V. IMPROVED APPLICATIONS SOFTWARE LOADING METHODS

A. Introductory Discussion

1. Memory Loading Requirements. As previously stated, ADEDS memory is a magnetic core consisting of 8000 locations, each 24-bits. After BOOTSTRAP LOADING, the system is made fully operational by loading the applications software into the remainder of memory. This applications software which undergoes continual change depending upon the experiments to be supported is used to create the various attitude and horizontal display formats necessary for TSRV flight research.

2. Original ADEDS Memory Loading Method. Originally all memory loading (BOOTSTRAP and application software) was accomplished via paper tape using the paper tape reader integral to the ADEDS system control unit. The memory image file used to create the paper tape was generated by software modules created by the assembler and placed on the Cyber in ACD. Loading by this means, while reasonably reliable, was archaic and very slow.

3. Cassette Loading Method. A change made in 1979 allowed use of a cassette for loading ADEDS applications software into memory. The same Cyber image file generation method was used, but a new method for transfer of memory image data into ADEDS was necessary since the paper tape reader (an optical device) was not compatible with cassette tape. The method developed used Split Phase Bipolar (SPBP) bus 1, an existent ADEDS interface described in reference 1, to load cassette resident memory image data. Figure 1 illustrates the use of the SPBP buses and figure 3 shows the configuration of the 32-bit SPBP words. Use of SPBP bus 1 for memory loading required revision of the ADEDS software loader to accept memory image data from the new source. The operational method of the revised loader will be discussed in the following sections, but its technical details are not covered herein.

   Figure 4 illustrates the path of data flow used in the cassette loading technique. Data read from a tape by the Magnetic Tape Unit (MTU) is transmitted via the RS-232 serial bus to the Microwave Landing System (MLS) Data Translation Unit (DTU) which contains a microprocessor and associated software for formatting the memory image data into SPBP serial words properly packed to meet ADEDS loader requirements (described in subsequent sections). An SPBP transmitter resident in the DTU hardware is used for data output to the bus 1 SPBP receiver in the ADEDS computer unit. It is seen in figure 4 that the ADEDS SPBP bus 1 receiver is manually switched to receive data from the DTU transmitter for loading, then returned to the host computer transmitter for normal operation. (Reference 1)

   This method of ADEDS memory loading, while an improvement over the paper tape method, proved very cumbersome operationally due to the distribution in the TSRV of the required electronic equipment shown in figure 4. Also, reliable MTU operation and cassette production and duplication were constant problems.

B. Floppy Disk Loading Method

The TSRV upgraded system retained the ADEDS but replaced all cassette media with floppy disks used in VT-180 terminals. Hence, it was necessary to develop a method of loading the ADEDS applications software from the disk medium. Major requirements for the disk loader development (dictated by the necessity of not modifying incompletely documented hardware and software) were:

1. Use of the existent ADEDS loader software, thus, continuing utilization of the SPBP bus 1.

2. Leave the Cyber software which generates the memory image file unchanged.

3. Develop the capability to download the Cyber image memory file directly to the VT-180 disk.

    1. Memory File Capture. Figure 5 illustrates the technique used to transfer a memory image from the Cyber to the ADEDS computer on board the TSRV. The process can be roughly divided into two steps:

        (1) Downloading the file from the Cyber to the VT-180 and storing it on diskette. This is done remotely and replaces the cassette tape recording which was created in a similar manner.

        (2) Transferring this disk to an on-board VT-180 which reformats the file and sends it over an RS-232 link to the circuitry (described later) which converts the RS-232 data to the serial SPBP format (reference 1) for input to the ADEDS loader. This is done by specifically written VT-180 software (described below) which replaces the function previously performed by the on-board Magnetic Tape Unit (MTU) and Microwave Landing System Data Translator Unit (MLS DTU).

        Cyber files are downloaded in the following manner: MODEM7, a public domain communications program, is used in the VT-180 to retrieve and store the ADEDS file. In "text capture" mode the VT-180 is used as a standard computer terminal while also saving on disk all information being received from the host system, in this case the Cyber. The operator runs the MODEM7 program in text capture mode and logs on the Cyber in normal fashion. Copying the ADEDS file to the terminal will result in it's being stored on the VT-180 diskette. The logging-on dialogue is also saved; it provides a convenient tag of Cyber filename, date, and time. No special commands are given to the Cyber other than the normal copy-to-terminal commands and it is unaware that it's output is being saved. Also, it is not required that the Cyber control the starting of the device which receives the file as is the case for creating a paper tape or cassette.

    2. Memory Image File Configuration. Figure 6 shows a sample of the file as it exists on the Cyber and, hence, on the VT-180 disk after downloading. The data are shown in binary form with the corresponding ASCII character appearing at the right. The display loader resident

in ADEDS fills each 24-bit memory location by storing serially 12 bits in each of two consecutive words. However, the Cyber software takes ADEDS 12 bit binary memory image data, splits it into two groups of six bits and adds 64 to each group to make all characters printable alphabetics. Thus, bit 6 of each byte shown in figure 6 is set. This Cyber implementation was done in the past, probably to circumvent some problem with paper tape equipment handling data that could have been interpreted as commands. It was not changed in this application, and loader software was written in the VT-180 to reconstruct proper image words from the existent Cyber output.

3. Use of SPBP Bus in Memory Loading. Since the ADEDS software loader uses SPBP bus 1 for memory loading, it must be written to conform to normal SPBP operation. In normal operation the SPBP receiver in the ADEDS places input data into pre-allocated locations in 32 word blocks, each word being 32-bits with the least significant 8 bits constituting an address label (figure 3). When the 32 word block (buffer) is filled, an interrupt signal is generated by the SPBP receiver. The processor in the ADEDS then uses this interrupt as a command to begin reading the buffer. For memory loading the buffer is still used, and the data placed therein contains both address and memory image information extracted from the cyber file. The same interrupt signal is also used, but the processor now follows instructions found in the loader program and places the buffer data into memory locations rather than using the data for display production. Upon completion of loading control is transferred to the main display program (whose instructions were contained in the file just loaded) and the system is operational.

4. Algorithm for Loader Data Word Construction. Figure 6 and figure 7 will be used to describe the algorithm required for loader word configuration. Since exact documentation for the content of the Cyber file was not available, the file was inspected and its charactersitics determined empirically.

A 64 work SPBP buffer is used by the loader to load 30 ADEDS memory locations, thus, requiring two normal 32 word buffers with 15 locations being loaded from each half. The first word encountered (word 1 in figure 7) contains the word count in bits 6 through 11, and the high order address data in bits 0 through 5. In moving through the Cyber file (figure 6) when the "Device Control 1" (11H) is encountered, data usage for loading will begin. This character, which is marked in figure 6, was likely used originally to start a paper tape punch. Its use herein to begin data usage eliminated any need to change the Cyber function. The next character is seen to be 40H, or 01000000 binary, and will be used to derive the word count for the block that follows. As has been stated, only the lower 6 bits of the character are used; thus, the effective value will be 000000 binary, or 00H, yielding a word count of zero for the first block. The loader will fill no locations until a non-zero word count is found. Since the date is contained in the first block, it appears that the Cyber file was constructed this way to allow inclusion of the date in the tape for identification purposes without introducing bad data into the memory. For some reason no filename was included in this

identification, but such is easily added to the dowloaded disk file simply by using a text editor. Examination of figure 7 shows that 12 bits are required to fill each display loader word, thus, using two characters from the Cyber file (figure 6) per word. This means that 128 characters in figure 6 are used to fill the 64 word buffer used to construct the word configuration shown in figure 7. Concident with use of the 64th word will occur an SPBP interrupt which will signal the loader to start a new block. Counting 128 characters beyond the DC1 (11H) character in figure 6 yields a value of 5EH which will be interpreted as another word count character. Its value is 01011110 binary yielding 011110 when only the lowest order six bits are used. Thus, the effective value is 1EH or 30 decimal which will be placed in bits 6-11 of word 1 in figure 7, thus, setting the word count of the block to 30.

The next character in the Cyber file must be used to construct the high order address data, bits 0-5 in word 1. This character (figure 6) is 40H, or 01000000 binary, and the lower 6 bits yield 00H. The next two characters form the low order address bits to fill word 2, bits 0 through 11. These characters are 40H and 7CH or, repectively, 01000000 and 01111100 binary. Taking the lowest 6 bits of each and combining the result yields 000000111100 binary, or 03CH. Combining this with the high order address value yields 003CH as the first address to be loaded. The next two characters of the Cyber file are used to fill the lowest 12 bits of word 3, the high order data bits to be placed at location 003CH. These characters contain 40H and 41H, or 01000000 and 01000001 binary. Again using the lower 6 bits of each and combining, the result is: 000000000001 binary, or 001H for the high order data. To complete formation of the data for location 003CH, the next two characters are used to fill word 4 in figure 7, the low order data bits. Once again examination of figure 6 yields 60H and 52H for these characters or, respectively, 01100000 and 01010010 binary. Combining the lowest 6 bits of both the result is 100000010010 binary, or 812H. Further combination of this data field with the high order data field derived above yields a value of 001812H being loaded at address 003CH in ADEDS memory.

From this point, the address value is incremented and two more data words identical to words 3 and 4 are formed, a process which continues until the 32 SPBP words are all used, thus, filling 15 ADEDS 24-bit memory locations. Address incrementation then continues through the second half of the 64 word buffer with all words formed being data words identical to words 3 and 4 of the first half. Since no address words are necessary in the second half, the last two words are not needed and are ignored by the loader. They are present with all zero data in the Cyber file. Their absence would result in blocks of characters not equal to 64. Although the Cyber software was probably not written with SPBP buffers being considered, the 64 word block implementation made the task of using the ADEDS SPBP data bus much easier. (As stated earlier, due to lack of complete documentation for the Cyber software implementation, empirical means were required in developing this loader to match it.) When 64 loader words of figure 7 have been filled, 128 characters in the Cyber file, figure 6, have been used and the following character is once again 5EH, the data from

which the word count of 30 decimal is derived. Thus, the cycle repeats.

The VT-180 software written for this effort performs the data manipulation described in the above paragraph. When either a "Device Control 4" character (14H) or an "End of Transmission" character (17H), both of which are found in the Cyber memory image file and marked in figure 6, is encountered the loader terminates the loading process and transfers control to the main ADEDS processor. Normal display symbology produced by the applications program just loaded then occurs.

It should be mentioned that all the loader words shown in figure 7 contain zeros in the upper 12 bits. The ADEDS loader was written in this manner, probably because bit 23 is parity and is controlled by hardware. This means that one loader word would be one bit short of the 24 bits needed to fill a 24-bit memory location. The VT-180 software, in addition to packing the loader words of figure 7, must place the required zeros in the upper 12 bits of each word.

5. VT-180 Software. The display loader program (DISPLOAD), like the file utility previously described, was written in the "C" language (references 2 and 3) to run under the CP/M operating system. Its specific purpose is to take a specially formatted file of characters previously downloaded from the Cyber computer system, translate them into binary data, and send this binary data to the RS-232-to-SPBP conversion circuitry (figure 5) through one of the VT-180's serial ports. The source program listing follows. Once again, the line numbers were added for documentation purposes and are not found in the original file.

```
 1: #include "b:libc.h";
 2: #define CTLC 3
 3: #define DC1 17
 4: #define DC4 23
 5: #define ETB 23
 6: main(){
 7: FILE *fopen(),*fp;
 8: char c,d;
 9: int null,pfn,ntemp,l,m;
10: static char filnam[20]; /* display program filename to be loaded */
11: static char spbp[4]; /* 4 bytes to be made into 32 bit spbp word */
12: static int br=0; /* nested loop break flag */
13: printf("filename ? (example a:dascas2.dsp) ");
14: /*
15: --- main loop ---
16: /*
17: while(1){
18:   /* accept filename ,read,process,send to DIU */
19:   pfn=0; /* init the filename pointer */
20:   while((filnam[pfn++]=getchar())!-"\n"; /*collect filename*/
21:   if((fp=fopen(filnam,"r"))==NULL){
22:     printf("can\'t open file\n);
23:     break: /* if file not found */
```

```
24:    }
25:    else {
26:      /* open the file, type out its contents until a DC1 is found */
27:      while((c=getc(fp))!-DC1)putc(c,stdout);
28:      m=0;
29:      br=0;
30:      /* read 64 pairs of ascii characters
31:        and make 64 spbp words, of which 2 are word count and address,
32:        60 are memory image for display computer and 2 are zeros. */
33:      while(br==0){
34:        printf("\nblock # %d\n",m);
35:        for(ntemp=0;ntemp<64;ntemp++){
36:          /* check for "punch off" or "end transmission block" */
37:          if(((c=getc(fp))==DC4)||((d=getc(fp))==DC4)) {br=1;break;}
38:          if((c==ETB)||(d==ETB)){br=1;break;}
39:          /* check spbp word count */
40:          if((ntemp==0)&&(c!=0x5e))
41:            printf("block count not 30 in block %d \n",m);
42:          /*(the CYBER pads with 0x40 to make it printable ascii)*/
43:          /* spbp label counts up to 31 twice in a block */
44:          spbp[0]=(ntemp & 0x1f);
45:          /* take low 6 bits of each ascii and make 12 bit word */
46:          spbp[1]=c;
47:          spbp[1]=(spbp[1]<<6)|(d & 0x3f);
48:          spbp[2]=(c>>2)& 0xf;
49:          spbp[3]=0;
50:          /* printf("ascii = %x %x  ",c,d); */
51:          /* printf("spbp = "); */
52:          /* for(i=0;i<4;i++)printf("%x ",spbp[i]); */
53:          /* printf("disp mem = %x%x:,(spbp[2]&0xf),spbp[1]; */
54:          /* printf("\n");*/
55:          /* send the reconstituted spbp word to the DIU */
56:          sndstr(spbp,4);
57:        }
58:        m=m+1;
59:        if(br==1)break;
60:      } /* end of file processing loop */
61:      printf("processed %d blocks\n",m);
62:      break;
63:    }
64: } /* end of file processing loop */
65: printf("terminated \n");
66: }    /* end of main() */
67: /*
68: */
69: sndstr(str,p) /* send p bytes out to the UC1: port */
70:      /* use bios to aviod tab expansion etc. */
71:    int p;
72:    char str[];{
73:    int iz;
74:    iz=0;
75:    touc1();
76:    while (iz<p) bios(4,str[iz++],0);
77:    tocon();
```

```
 78: }
 79: /*
 80: */
 81: toucl(){  */ switch console output to ucl: */
 82:   char iosave,null;
 83:   /* get the IOBTYE, bdos call 7 */
 84:   iosave=bdos(7,null);
 85:   /* set the lsb's to 11 and send console output to UCl: */
 86:   iosave=iosave| 0x3;
 87:   /* send the modified IOBYTE back to CP/M, bdos call 8 */
 88:   bdos(8,iosave);
 89:   /* now console output will go to the general purpose
 90:    port and thence to the DIU */
 91: }
 92: /*
 93: */
 94: /* switch the iobyte back so console output goes to the screen */
 95: tocon(){
 96:   char iosave,null;
 97:   iosave=bdos(7,null);
 98:   iosave=iosave & 0xfc;
 99:   iosave=iosave   0x01;
100:   bdos(8,iosave);
101: }
102:
```

6. Software Functional Description. Lines 1 through 12 declare data types and initializations. Line 13 prompts for the filename of the characters to be processed and line 20 concatenates the operator's keystrokes into an array to be used as the filename. Lines 17 through 64 form an endless loop. (The value 1 will always be true.) It will be iterated only once, however, being terminated from line 23 if the file specified does not exist or from line 62 at the end of processing. In line 21 the file is opened for read only and the pointer "fp" is set to point to the first character of the file. The file is then scanned and its contents (the log-on dialogue with the Cyber) displayed to the operator until the "Device Control 1" (11H) character is encountered in line 27, signaling the beginning of active data. Lines 33 through 60 form a loop which is iterated until either a "Device Control 4" (14H) or "End Transmission Block" (17H) character (both of which exist at the end of the ADEDS files) is found by lines 37 and 38. Sixty-four pairs of characters are read by the "for" loop which starts on line 35 and continues to line 57. It is seen that line 37 reads two characters from the file and assigns them to two variables "c" and "d" as well as testing their value. In lines 37 and 38, if a character is found, the "br" variable is set. This group of statements is nested within two "while" loops, and the "break" can force exit of only the first enclosing loop. The break flag "br" breaks the second loop in line 59. Line 40 tests the very first character of a 64 pair group for a value of 94 decimal (5EH), equivalent to a word count of 30 plus 64 added by the Cyber as mentioned above. If the word count is not 30, the operator is notified, but processing continues. This provides a check for lost characters in the Cyber download process. In operation, the first

group and last two groups of the ADEDS file are usually found to contain word counts of zero, but, nevertheless, contain the proper 64 pairs of characters.

The ADEDS loader expects to see the SPBP label increment from 0 to 31 twice during each block for a total of 64 loader words, each composed of a character pair. However, only 60 character pairs contain memory image data. Of the remaining four, two are used for word count and address data at the beginning and two are filled with zeros at the end. Lines 44 through 49 reassemble the two printable characters into 12 bit binary data, add a word count (SPBP label) and zeros (highest 12 bits) to fill the "spbp[]" array. The four bytes of this array at this point make up the 32 bit word which will be sent to the VT-180 serial output, then to the RS-232-to-SPBP conversion circuitry (described later). The output stage of this conversion circuitry is an SPBP transmitter which sends the data to the corresponding receiver in the ADEDS.

During program development, lines 50 through 54 displayed data in its various forms as it went through these processes. Output is done by passing the "spbp[ ]" array to the "sndstr" function from line 56. Lines 58 through 61 keep a running total of blocks processed and terminate the program when the break flag ("br") is set. DISPLOAD terminates and returns to the CP/M operating system from line 66.

The "sndstr" function, line 69, is presented with an array and a number of bytes to be sent out through the VT-180's console device. It uses "tocon" and "toucl" to manipulate CP/M's IOBYTE, (reference 5) directing console output through the "UCl" serial port, which is connected to the RS-232-to-SPBP conversion circuitry, rather than the "CON" port (the keyboard and screen.) Because the ADEDS data is binary, BIOS (Basic Input Output System) call (reference 5) must be used to keep data from being interpreted as screen formatting commands by the BDOS (Basic Disk Operating System) (reference 5).
Lines 71 through 74 declare data types and initializations; output occurs in the "while" loop of line 76.

Functions "toucl" and "tocon" perform mirror image bit manipulation on the two least significant bits of the IOBYTE, lines 85 through 88 and 98 through 100.

7. Loader Hardware. The software described packs words conforming to ADEDS loader requirements. These words are then sent one byte at a time to the proper VT-180 serial port and then by an RS-232 link to the conversion circuitry shown in figure 5. This circuitry comprises the hardware designed and constructed for the loader task. A description of RS-232 is contained in reference 4 and is not covered in detail herein.

The basic function of the conversion is to accept serial RS-232 input, separate this data into bytes, arrange four consecutive bytes into one 32-bit SPBP word, and transfer this word to the input of an SPBP transmitter. The resulting circuitry shown in figure 8 was placed in the Display Interface Unit (DIU) which is described in reference 1.

This is advantageous because the bus 1 SPBP transmitter existent in the DIU can be used. A schematic of the conversion circuitry is shown in figure 8 and a description is contained in the following paragraphs.

Serial data bits are received by U41 and sent into U39, a Universal Asynchronous Receiver Transmitter (UART). The data bits are placed in a parallel register in the UART, and when eight bits (1 byte) have arrived, the data bits are on pins 5 through 12 of U 39 and are available to the 8-bit latches, U 33, U 32, U 29, and U 33. All four of these latches must be filled using four consecutive bytes to form one 32-bit word for the SPBP transmitter. The first byte must go into U 33, the second in U 32, third in U 29, and the fourth in U 31.

Routing of the data bytes into proper latches is accomplished by the circuit elements shown at the right in figure 8. Principal elements are the counter, U 34, and the decoder, U 35. The counter is configured with two output bits, pins 2 and 3, which will yield four states, the number of bytes needed to fill the four latches. The decoder has four outputs, pin 4, 5, 6, and 7, which are connected through inverters to pin 11, the ENABLE gate of each latch (54LS373). The following paragraph traces a byte of data through the chain.

Serial data bits entering the UART (U 39) are placed sequentially into a parallel register represented by pins 5 through 12. When 8 bits (1 byte) have entered, an output pulse occurs at pin 19, the DATA READY ilne. This pulse enters U 37, a monostable mulitvibrator (one shot) on pin 2 resulting in an output pulse at pin 4 which in turn enables the decoder, U 35. The decoder in response activates one of its outputs on pins 4, 5, 6, and 7 with the particular output activated being controlled by the state of the input lines at pins 2 and 3. In the case of the first data byte, pins 2 and 3 will both be low (logic zero) causing the output on pin 4 to be enabled, thus, latching this byte (byte 1) into U 33, the least significant byte of the SPBP word. Returning to U 37, it is seen that another output exists at pin 12. This output is controlled by an input on pin 10 which is connected to pin 4, the decoder enabling signal just described. In effect this means that pin 12 is controlled indirectly by the input to pin 2 with a delay equal to the pulse width at pin 4. This delayed signal serves two functons: (1) stepping the counter, pin 5 of U 34 and (2) resetting the DATA RECEIVED RESET input (pin 18) of U 39. These pulses render the circuitry ready to receive the next byte of data and place it into U 32; thus, they must be delayed until after the previous byte is latched.

This process continues until byte four is latched at which time one 32-bit SPBP word is contained in the latches and ready for transmission to the ADEDS loader, a task accomplished by sending the byte 4 latch pulse to the SPBP transmitter as a start pulse. Once started the transmitter will serially send all data bits in the latches to the loader, then stop and wait for the next start pulse. This process will continue until the VT-180 software has sent the entire image memory file to the loader circuitry. Then the functions end and the ADEDS computer jumps to its main processing routines.

To activate the loader hardware in the DIU, a means must be provided to switch the SPBP bus 1 data source from normal operation (host computer) to the loader. This is accomplished by SW 1 of figure 8, which alternatively activates the output control of the loader latches and another identical set of DIU latches (not shown) which are enabled during normal operation (reference 1). When SW 1 is open, the loader latches have a high state on pin 1 thus disabling them. At the same time, the inverter U 30 will supply a low state to the normal DIU bus 1 latches. When SW 1 is closed, a low state will exist on pin 1 of each of the loader latches, thus enabling them, and the same inverter, U 30, will supply a high state to the normal bus 1 latches, thus, disabling them. It is also seen from figure 8 that closing SW 1 triggers the monostable multivibrator U 36 causing a pulse from pin 13 which resets both the counter (U 34) and the MASTER RESET of the UART (U 39). This initializes all the loader circuitry in preparation for receiving data. Switch SW 1 corresponds to the LOAD/NORMAL switch shown in figure 4 for the cassette loader, but in this instance it is more advantageous since the actual SPBP line does not require breaking.

The hardware and software described above has been implemented and is fully operational. Its use has resulted in significant improvements in ADEDS operation.

VII. CONCLUDING REMARKS

New techniques with increased reliability and user convenience have been developed for loading the core memory of the ADEDS, the TSRV monochrome display system which was retained as part of the initial upgrade.

The BOOTSTRAP loading function can now be accomplished using an EPROM created as an exact image of the original paper tape BOOTSTRAP file. Loading applications software into memory is done from a VT-180 disk onto which the memory file is written from the Cyber. Very significant improvements in operational reliability and creation of backup files has resulted.

These improvements were accomplished entirely by in-house effort. For the most part empirical means were required to determine ADEDS hardware and software functions since complete documentation was never prepared for this prototype system.

The changes described herein and the new interface to the DATAC data communication bus (reference 1) are the last intended modifications to the ADEDS and should improve its usability until it is replaced by a color system to complete the TSRV upgrade.

VII. REFERENCES

1. Herzog, H. K.: Commercial Airplane Data Bus Requirements and DATAC. SAE-A-2K High Speed Data Bus Subcommittee, Dayton, Ohio, WPAFB/ASD, May 22, 1981.

2. Kernighan, Brian W.; Ritchie, Dennis M.: The C Programming Language. Prentice-Hall, Inc., 1978.

3. Purdum, Jack: C Programming Guide. Que Corporation, 1983.

4. Seyer, Martin D.: RS-232 Made Easy. Prentice-Hall, Inc., 1984.

5. Cortesi, David E.: Inside CP/M, A Guide for Users and Programmers. Holt, Rinehart, and Winston, 1982.

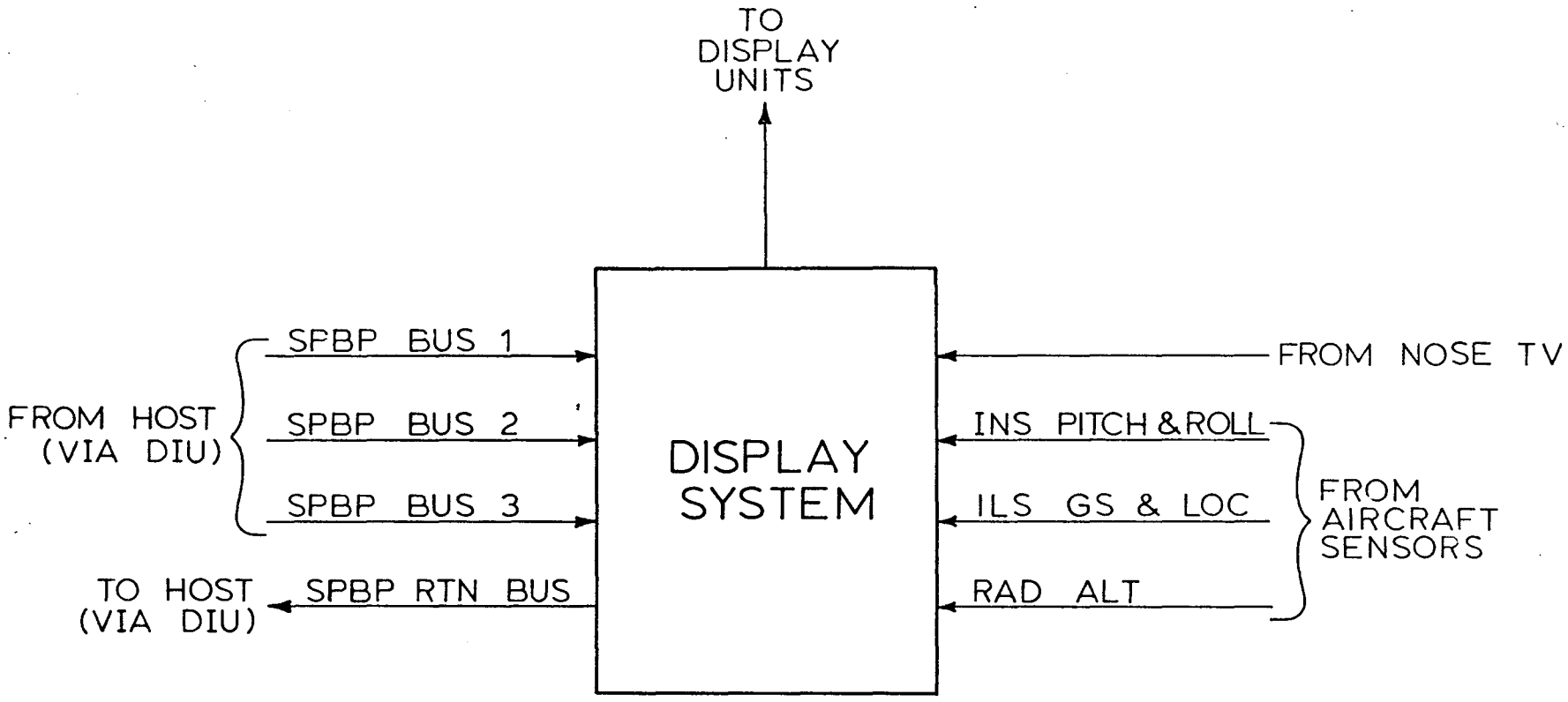


FIGURE 1. DISPLAY SYSTEM INTERFACE SIGNALS

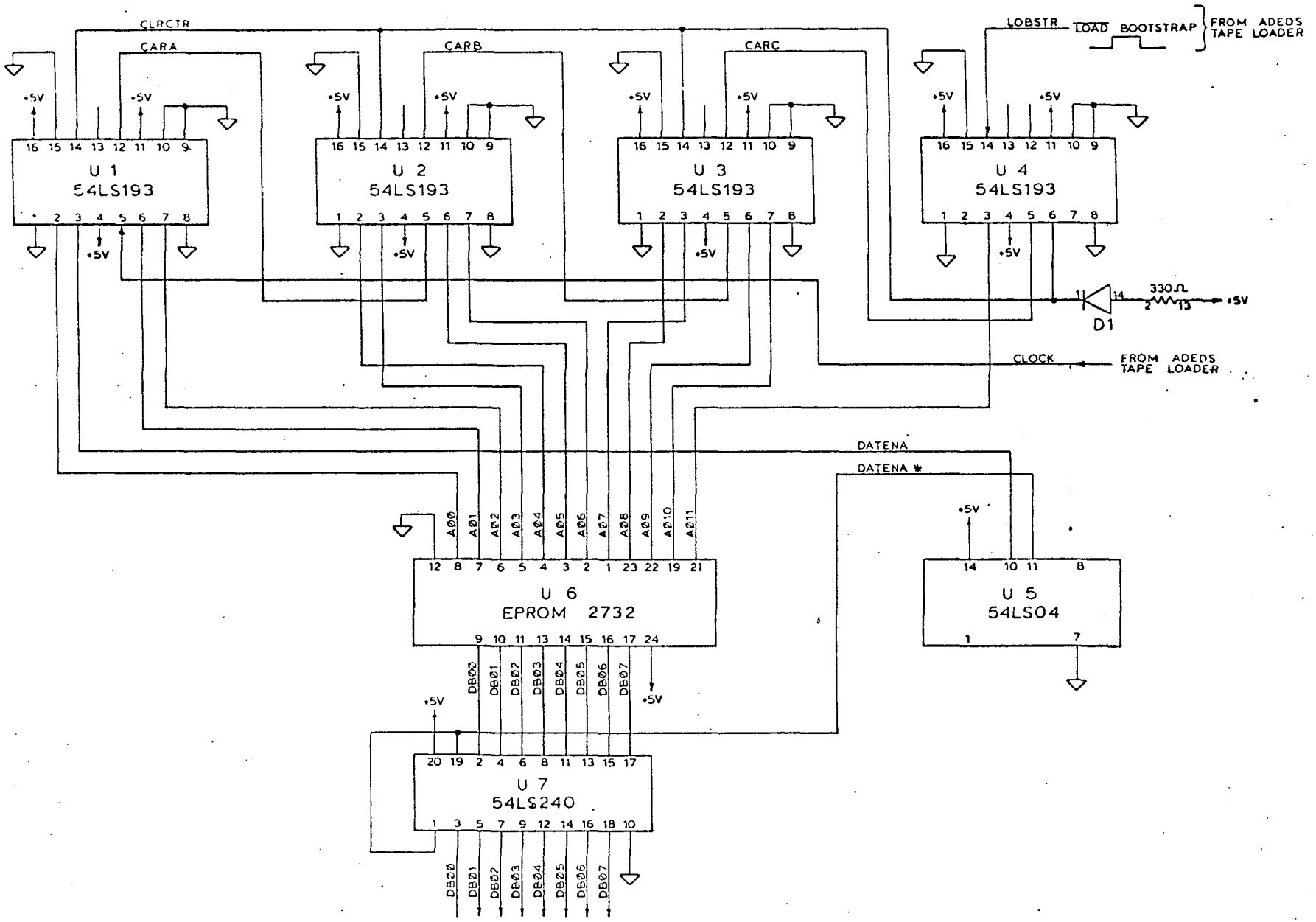
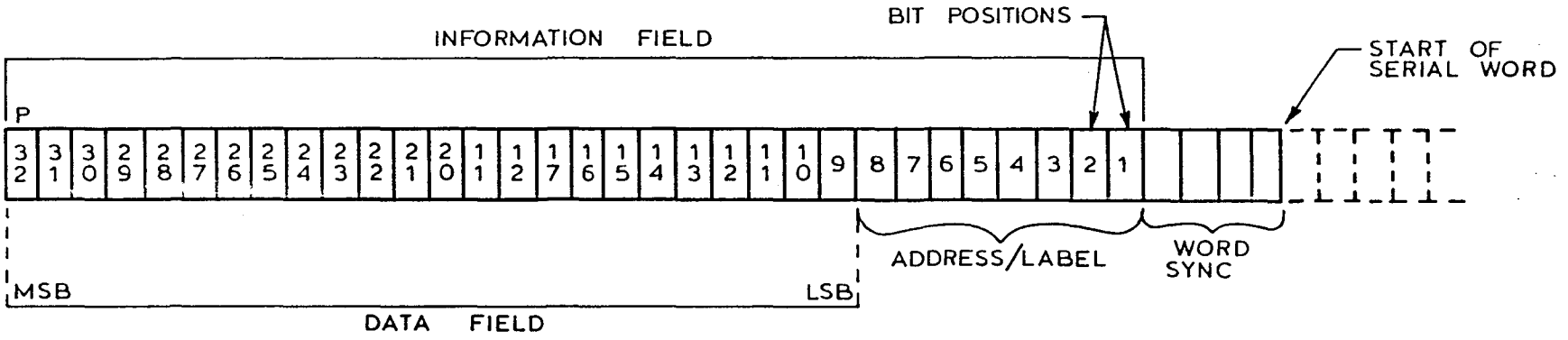


FIGURE 2 . ADEDS EPROM BOOTSTRAP LOADER CIRCUITRY



NOTE:
P = PARITY

FIGURE 3. SPBP WORD STRUCTURE

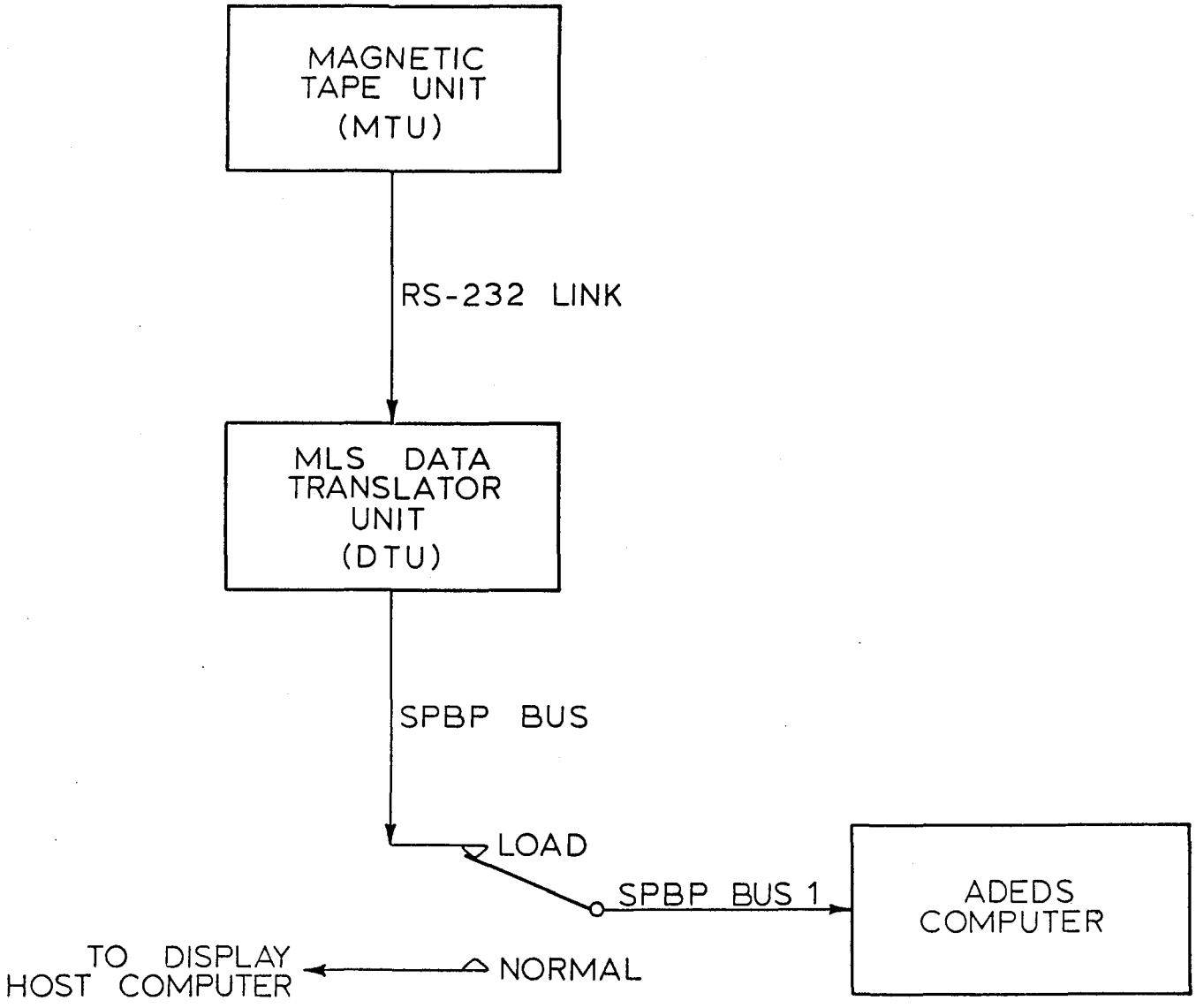


FIGURE 4. ADEDS CASSETTE LOADING METHOD

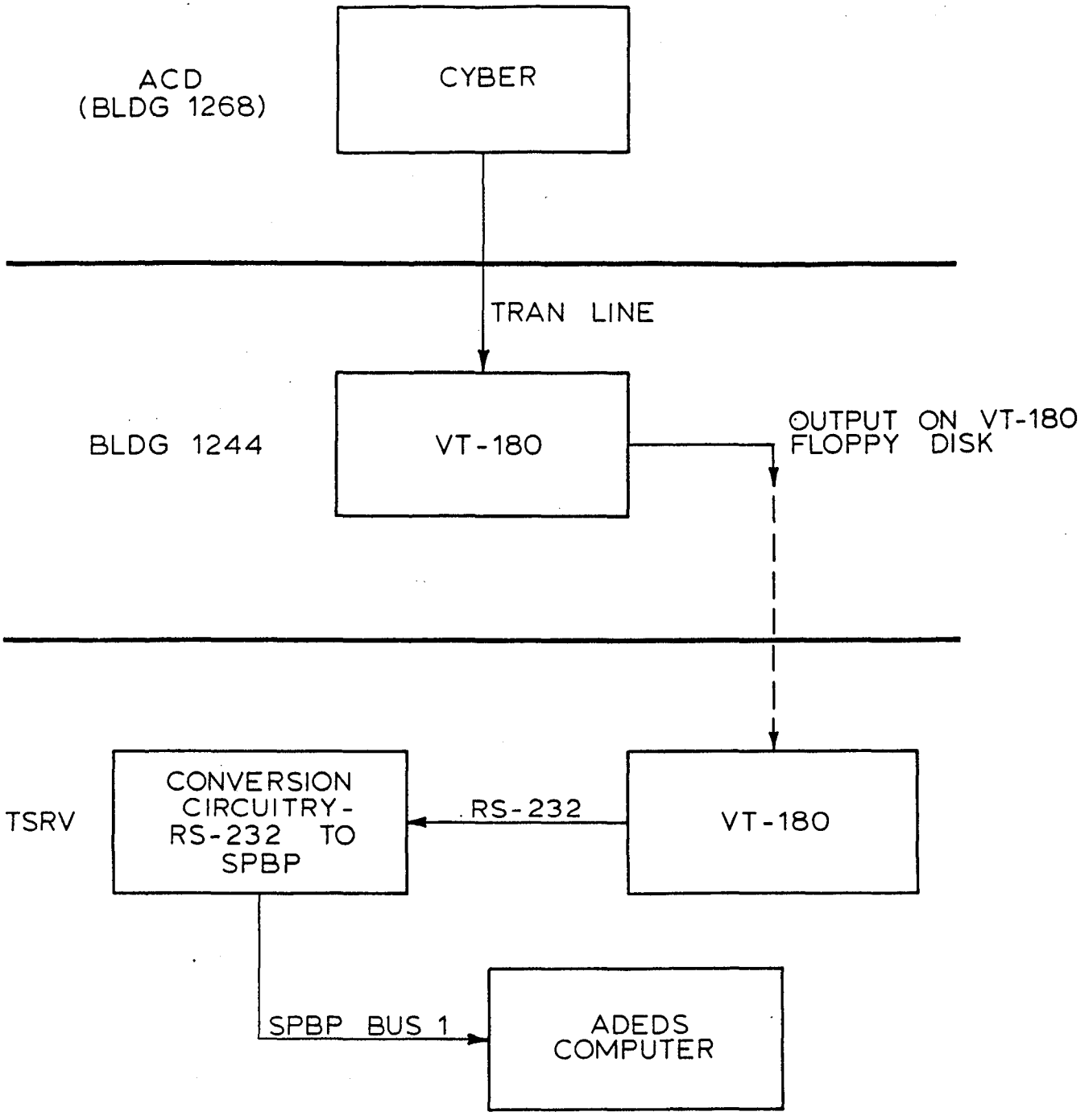


FIGURE 5. ADEDS MEMORY FILE TRANSFER FOR DISK LOADER

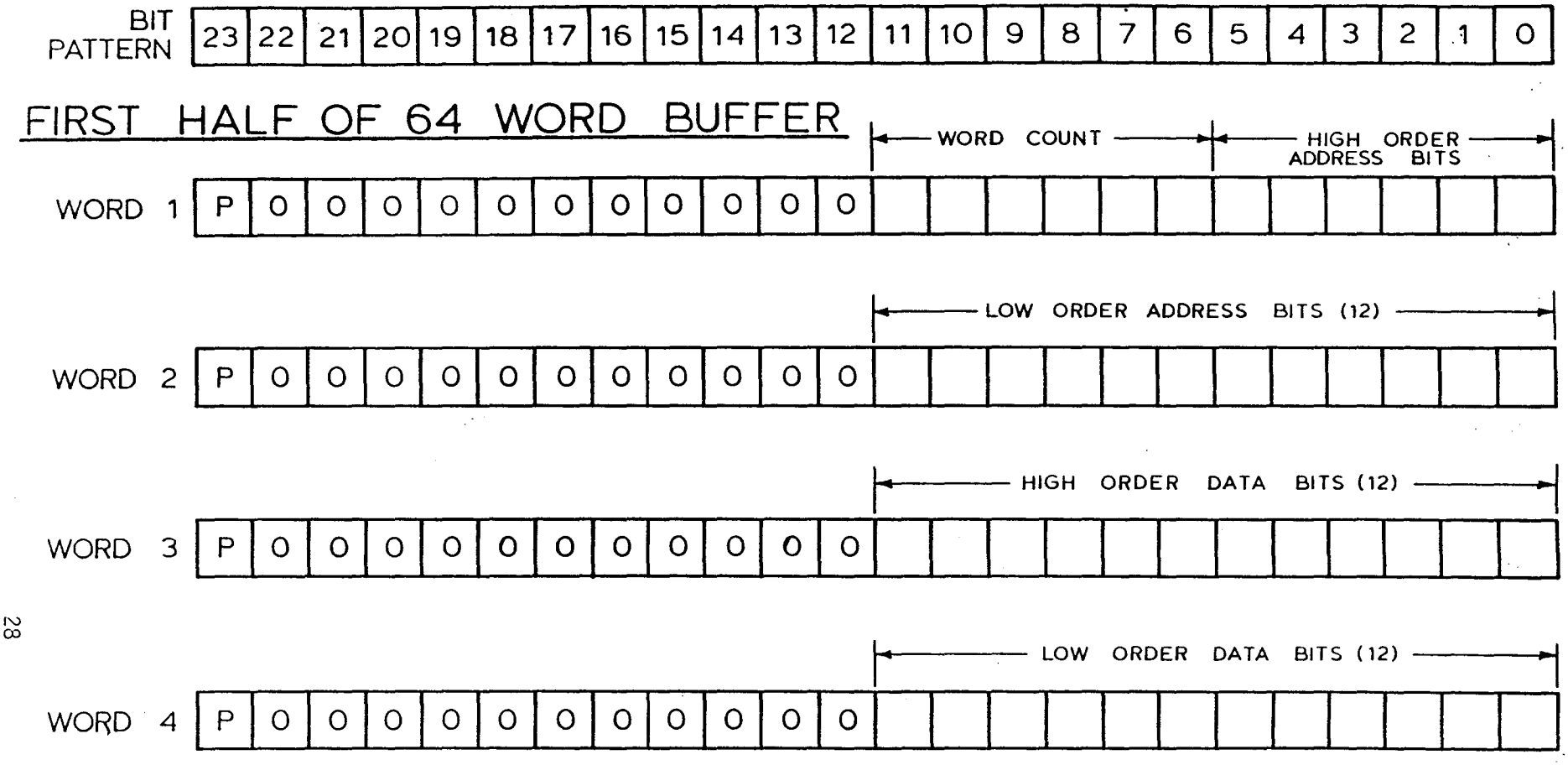
```
0D 0A 2F 72 65 77 69 6E-64 2C 2A 0D 0A 20 20 20
33 20 46 49 4C 45 28 53-29 20 50 52 4F 43 45 53
53 45 44 2E 0D 0A 2F 63-6F 70 79 2C 64 61 73 63
61 73 0D 0A 0D 0A 20 20-44 53 50 4C 59 20 43 41
53 53 45 54 54 45 20 54-41 50 45 0D 0A 11 40 40
40 40 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 38 34 2F 30 32 2F-31 30 20 0D 0A 20 20 20     84/02/10 ..
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20

61 7A 40 41 5E 50 40 41-5F 50 40 40 40 40 5E 40   az@A^P@A_P@@@@^@
40 7C 40 41 60 52 40 40-72 7E 40 40 7E 7E 40 41   @|@A`R@@r~@@~~@A
77 50 40 41 7A 40 40 41-40 40 40 41 40 60 40 41   wP@Az@@A@@@A@`@A
41 60 40 41 40 60 40 41-41 60 40 41 42 60 40 41   A`@A@`@AA`@AB`@A
40 40 40 41 40 60 40 41-41 60 40 41 40 40 40 41   @@@A@`@AA`@A@@@A
40 60 40 41 41 60 40 40-40 40 40 40 40 40 40 40   @`@AA`@@@@@@@@@@
5C 40 40 40 5C 40 40 40-40 40 40 40 40 40 40 41   \@@@\@@@@@@@@@@A
45 74 40 40 40 40 40 40-40 43 40 40 40 64 40 47   Et@@@@@@@C@@@d@G

7F 7F 40 40 40 40 40 40-40 40 40 40 40 40 5E 40   ..@@@@@@@@@@@@^@
41 5A 40 40 40 40 40 40-40 40 40 40 40 40 40 40   AZ@@@@@@@@@@@@@@@
40 40 40 40 40 40 40 40-40 40 40 40 4E 4E 40 40   @@@@@@@@@@@@NN@@
40 40 40 40 40 40 40 40-40 40 40 40 40 40 40 40   @@@@@@@@@@@@@@@@
50 40 40 40 54 40 40 40-40 40 40 40 40 40 40 40   P@@@T@@@@@@@@@@@
40 40 40 40 40 40 40 40-40 40 40 41 40 40 40 40   @@@@@@@@@@@A@@@@
6B 78 40 40 40 40 40 48-40 46 40 40 40 40 40 40   kx@@@@@H@F@@@@@@
40 40 40 40 40 40 40 40-40 40 40 40 40 40 40 40   @@@@@@@@@@@@@@@@

40 40 40 40 40 40 40 40-40 40 40 40 40 40 17 14   @@@@@@@@@@@@@@..
0D 0A 20 20 20 14 0D 0A-13 0D 0A 20 43 41 53 53
45 54 54 45 20 52 45 43-4F 52 44 49 4E 47 20 43
4F 4D 50 4C 45 54 45 0D-0A 1C 20 20 20 20 20 1C
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 15 20 45
4F 49 20 45 4E 43 4F 55-4E 54 45 52 45 44 2E 0D
0A 2F 62 79 65 0D 0A 0D-0A 34 31 34 31 30 33 4E
```

FIGURE 6. SAMPLE OF CYBER MEMORY FILE



SECOND HALF OF 64 WORD BUFFER

WORDS 1-30 ARE DATA WORDS IDENTICAL TO WORDS 3 AND 4 IN FIRST HALF OF BUFFER

WORDS 31 AND 32: ZERO

FIGURE 7 - DISPLAY LOADER WORD CONFIGURATION

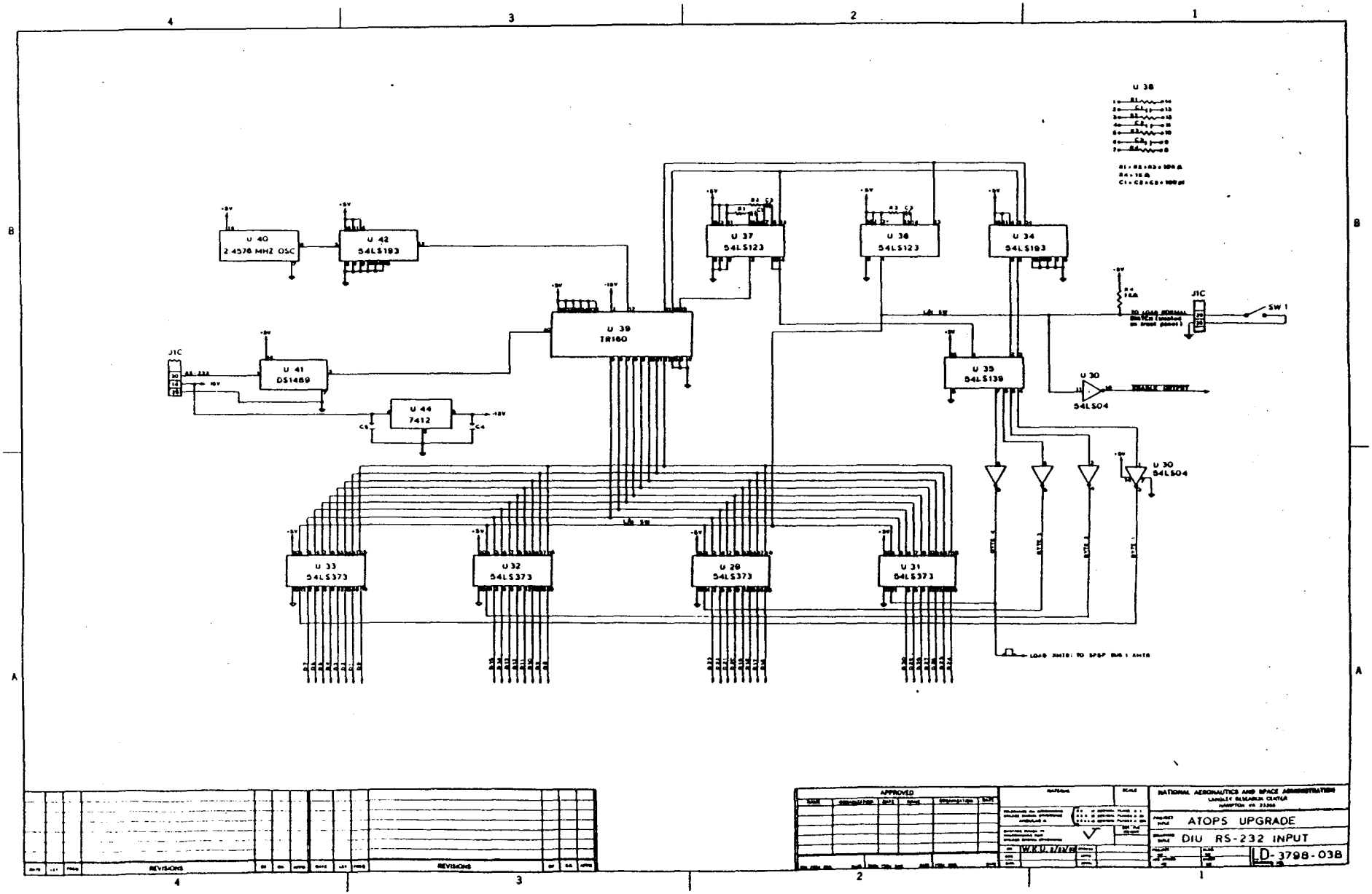


FIGURE 8. LOAD CUITRY

Standard Bibliographic Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-87649		

4. Title and Subtitle	5. Report Date
Improved Memory Loading Techniques for the TSRV Display System	January 1986
	6. Performing Organization Code: 505-66-41-22
7. Author(s): Wesley C. Easley, William A. Lynn, David G. McLuer	8. Performing Organization Report No.
9. Performing Organization Name and Address: NASA Langley Research Center, Hampton, VA 23665-5225	10. Work Unit No.
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address: National Aeronautics and Space Administration, Washington, DC 20546	13. Type of Report and Period Covered: Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

William A. Lynn and David G. McLuer, PRC-Kentron, Hampton, Virginia.

16. Abstract

A recent upgrade of the TSRV research flight system at NASA Langley Research Center retained the original monochrome display system. However, the display memory loading equipment was replaced requiring design and development of new methods of performing this task.

This paper describes the new techniques developed to load memory in the display system. An outdated paper tape method for loading the BOOTSTRAP control program was replaced by EPROM storage of the characters contained on the tape. Rather than move a tape past an optical reader, a counter was implemented which steps sequentially through EPROM addresses and presents the same data to the loader circuitry. A cumbersome cassette tape method for loading the applications software was replaced with a floppy disk method using a microprocessor terminal installed as part of the upgrade. The cassette memory image was transferred to disk and a specific software loader was written for the terminal which duplicates the function of the cassette loader.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
flight operational improvements flight display systems airborne display computer experimental flight displays	Unclassified - Unlimited Subject Category 09

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	35	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA Langley Form 63 (June 1985)