NASA Technical Memorandum 100406

Concept of a Programmable Maintenance Processor Applicable to Multiprocessing Systems

Richard D. Glover

February 1988

National Aeronautics and
Space Administration

NASA Technical Memorandum 100406

Concept of a Programmable Maintenance Processor Application to Multiprocessing Systems

Richard D. Glover
Ames Research Center, Dryden Flight Research Facility, Edwards, California

NASA
National Aeronautics and
Space Administration
Ames Research Center
Dryden Flight Research Facility
Edwards, California 93523-5000

CONTENTS

SUMMARY

A programmable maintenance processor concept applicable to multiprocessing systems has been developed at the NASA Ames Research Center's Dryden Flight Research Facility. This stand-alone-processor concept is intended to provide support for system and application software testing as well as hardware diagnostics. An initial mechanization has been incorporated into the extended aircraft interrogation and display system (XAIDS) which is multiprocessing general-purpose ground support equipment. The XAIDS maintenance processor has independent terminal and printer interfaces and a dedicated magnetic bubble memory that stores system test sequences entered from the terminal. This report describes the hardware and software embodied in this processor and shows a typical application in the checkout of a new XAIDS.

INTRODUCTION

In 1979 the NASA Ames Research Center's Dryden Flight Research Facility (Ames-Dryden) undertook the development of microprocessor-based, general-purpose ground support equipment (GSE) called the aircraft interrogation and display system (AIDS, ref. 1). A desire for higher performance later led to the requirement (ref. 2) for an extended AIDS (XAIDS) employing multiple 16-bit processors. One XAIDS (ref. 3) entered service in 1984, and a second will enter service in 1987. During the definition of the XAIDS design, the requirement arose for an independent processor capable of supporting both software and hardware testing. This processor became known as the maintenance processor (MAINT) and is now one of four processors comprising the baseline complement of every XAIDS unit.

This report focuses on the evolution of MAINT as an essential member of the XAIDS processor suite. The configuration of the hardware and software for MAINT is described, and software listings are offered as appendixes. Examples of the MAINT displays are shown, and an overview of operator procedures is presented. The concept is believed to be useful to any multiprocessing system requiring independent maintenance support.

NOMENCLATURE

AIDS	aircraft interrogation and display system
Ames-Dryden	NASA Ames Research Center, Dryden Flight Research Facility
BTO	bus timeout
CKI	command key interpreter
CRT	cathode ray tube display
DMA	direct memory access

I/O	input/output
ICE	in-circuit emulator
LED	light-emitting diode
MAINT	maintenance processor
MBM	magnetic bubble memory
MDS	microcomputer development system
NMI	nonmaskable interrupt
PERPRO	peripheral processor
PCI	programmable communications interface
PIC	programmable interrupt controller
PIT	programmable interval timer
PPI	programmable peripheral interface
PROM	programmable read-only memory
RAM	random access memory
SBC	single-board computer
SBX	single-board expansion
SDB	system debugger
V/V	verification and validation
XAIDS	extended aircraft interrogation and display system

BACKGROUND

During the research and development of the original AIDS design, there was no convenient way to gain access to the AIDS internal system bus for troubleshooting or for AIDS software testing. In the early stages of AIDS development, an in-circuit emulator (ICE) was heavily used as a software debugger and hardware checkout tool. This required the external support of a microcomputer development system (MDS) which had an in-circuit emulator installed. This configuration was not convenient in the lab and was totally unsuitable in the hangar when the AIDS was in service.

Several alternate approaches were considered for the final AIDS configuration, but lack of free cardcage slots and limited memory space greatly reduced the number

of options available. The approach finally selected was a separate embedded monitor package stored in an alternate set of programmable read-only memory (PROM) chips. This monitor has access to all of the processor's on-board input-output (I/O) and memory, in addition to the entire system bus including I/O and memory domains. The desired PROM set (operating system or monitor) was selected by a front panel switch that automatically generated a system reset whenever a change of PROMs was made. Activating the monitor package, therefore, dumped the user's software and prevented any real-time software-related operations whatsoever. Nevertheless, this monitor did prove useful in analyzing hardware and for that purpose was certainly more convenient than an emulator.

When the 16-bit XAIDS development began, it soon became apparent that the number of support options had increased considerably. For the 8086 microprocessor, there are several monitor packages available as well as in-circuit emulators. All are designed to support software debugging but may be used to examine and modify external subsystems mapped to the system bus. The operating system used on the XAIDS offers both an on-line dynamic debugger and a static system debugger. Both of these are designed to examine operating system objects such as tasks, mailboxes, and messages. The dynamic debugger permits object examination while the system is running; the static debugger stops the system (on either operator command or error condition) and becomes an extension of the monitor firmware. The latest XAIDS operating system configuration employs a system debug monitor plus a system debugger. The dynamic debugger was not included since experience had shown that a debugger was only required during configuration (system generation), and the static SDB would be adequate for that purpose.

For most multiprocessing systems, a similar variety of tools is usually available to the system designer. In the case of XAIDS, however, the available software did not allow independent access to the XAIDS bus while I/O job software was executing under control of the operating system. This requirement resulted in a separate so-called maintenance processor being included in the baseline XAIDS design. Such stand-alone hardware can be an asset in any multiprocessing system where a diagnostic tool independent of the operating system is required.

EVOLUTION OF XAIDS MAINT

The basic XAIDS design involves a tightly mapped environment with numerous dual-port memories and I/O domain hardware elements, all mapped to a common system bus. Applications of MAINT include bus mapping checks during system build, functional testing of subsystems, and aid to software verification and validation (V/V) by permitting detailed examination of software mailboxes used for interprocessor communications. The main functional requirement for MAINT is to be able to police memory and I/O by giving the operator independent read and write capability in both domains. A secondary functional requirement is to independently monitor activity of the eight interrupt lines on the system bus. The early operational requirements included ability to function with any ASCII terminal regardless of baud rate, and the goal that operator keyboard protocols require the fewest possible keystrokes.

The early MAINT configuration involved a single board computer installed in the XAIDS cardcage hosting a simplified monitor resident in PROM, with the operator

interface provided via a separate terminal connected to the rear of the XAIDS console. The detailed requirements for the early MAINT configuration were

- independent 16-bit memory segment selection,
- 20-bit addressing in memory mapped domain,
- 16-bit addressing in I/O mapped domain,
- both 8- and 16-bit operations in both domains,
- both read and write operations in both domains,
- single keystroke repeat feature on read operations,
- byte sequential memory substitute operations,
- byte, word, and double-word memory fill operations,
- job buffer for test requiring multiple operations,
- job buffer editor allowing delete, append, and erase,
- eight bus interrupt counters displayed on demand, and
- nonmaskable bus timeout interrupt to prevent hangup.

Many of the features normally found in a monitor are missing: register manipulation, software loading, and execution control. These features are of benefit only within the processor executing the application software.

The MAINT installed in the XAIDS brassboard in the Ames-Dryden XAIDS laboratory was used extensively in support of XAIDS system and user software V/V. In addition, the brassboard MAINT has been quite useful in checking out new user boards, especially custom-engineered boards. The MAINT installed in the first production XAIDS received considerable use during system build and during several user software installation cycles. During two years of operation, the early MAINT configuration proved to be an invaluable tool, but several areas for possible improvement became evident.

The concept of a job buffer containing a sequence of assorted operations proved to be a valuable feature of MAINT, however, the original mechanization was inadequate in two ways. The first inadequacy was volatility because the buffer was mapped into random access memory (RAM). To make a hardware configuration change during testing of a new board, it is necessary to power down the system because boards cannot be either inserted or removed with power applied to the cardcage. Therefore, whatever job may have been created was lost and had to be entered again when the system was powered up. This lowered operator productivity with wasteful repetitive keyboard operations. A second inadequacy was the limitation to a single job buffer; frequently it was found desirable to have several jobs existing simultaneously. Combining these two aspects of the problem indicated a need for a separate nonvolatile memory of some sort, partitioned into multiple job files.

A shortcoming in the data readout format became evident when the MAINT was used in a software V/V support role requiring data areas in dual-port memories to be displayed. Most of the data in the XAIDS software suite is other than type BYTE; commonly used types are WORD, DWORD, POINTER, and ASCII. The display of these higher-order data types as fragmented byte arrays makes reading the data difficult, and this in turn lowers operator productivity. Thus a need arose to permit the operator to specify the desired data type for MAINT operations.

A variety of terminals has been employed ranging from 300-baud printing terminals to 19200-baud cathode ray tube (CRT) terminals. For some types of testing, the printing feature of some slower terminals was a requirement for record keeping purposes, however, the low baud rate reduced the speed at which tests could be performed. Because of this, a higher baud rate CRT-type terminal was normally employed, but on some occasions the need for hardcopy would unexpectedly arise. This experience underscored the need for a means to produce hardcopy selectively while using a CRT terminal.

In mid-1986 the decision was made to upgrade the MAINT processor in the brassboard and also in the XAIDS, then under development. The memory device selected was a magnetic bubble memory (MBM) that provides the proper capacity and speed for this application. The printer interface selected was the Centronix-type parallel standard. The supplemental requirements for the MAINT upgrade were

- MBM partitioned into 100 job files with a title for each,
- job files grouped as 10 decades of 10 jobs each,
- job directories show file titles by decades (0 thru 9),
- job editor can load, modify, or save any of 100 files,
- job execution within decade invoked by single keystroke,
- MBM backed up to an external RAM buffer,
- MBM restored from an external RAM buffer,
- memory writes to single address in BYTE or WORD format,
- memory fills in BYTE, WORD, DWORD, or ASCII format,
- memory reads in BYTE, WORD, DWORD, POINTER, or ASCII,
- I/O fill outputs a constant value to a range of ports,
- I/O operations may be in BYTE or WORD format,
- printer interface is Centronix-type parallel standard, and
- screen image snapshots buffered for printer output.

HARDWARE CONFIGURATION

The XAIDS MAINT is a single-board computer (SBC) employing an 8086-2, 16-bit microprocessor operating at an 8 MHz clock rate. It is configured with 8 Kbytes of static RAM, four 2732A PROMs (16 Kbytes), and a piggyback MBM module. As shown in figure 1, the 128-Kbyte MBM piggyback is installed at the single-board expansion (SBX) bus J4 connector position. For this application the MBM has its jumpers configured for the polled mode of operation so that no interrupts or direct memory access (DMA) operations are involved.

The serial interface for the terminal is controlled by the 8251A programmable communications interface (PCI) chip on the 86/05 board. The PCI, plus its interfacing driver and receiver chips, are configured for full duplex RS-232 in data-set mode. Any baud rate from 110 to 19,200 may be software-selected using the clock 2 output from the 8253 programmable interval timer (PIT) chip.

The printer interface is controlled by the 8255A programmable peripheral interface (PPI) chip on the 86/05 board. Port A is configured for output through an 8287 driver chip supplied on the board and controls the eight printer data lines. Port C bits PC0-PC3 are configured for output through a 7408 driver chip. Bit PC0 is used to control the printer strobe signal, while bit PC3 is used to control the light-emitting diode (LED) mounted on the 86/05 board. Port C bits PC4-PC7 are configured for input using one SBC-902 terminator chip containing four 1000-Ohm pullup resistors. Bit PC4 is to used for the printer select signal, bit PC6 is used for the printer acknowledge signal, and bit PC7 is used for the printer busy signal. Port B is not used but is configured with two SBC-902 terminators.

SOFTWARE DESCRIPTION

The software for MAINT is produced on the XAIDS brassboard system installed in the Ames-Dryden XAIDS laboratory. This software is written in PLM86 programming language and is broken into four separate modules: a main program, interrupt routines, line printer routines, and MBM routines. These modules are separately compiled using the COMPACT and OPTIMIZE(3) controls that minimize the size of the resulting code segments. The four object files produced by the compiler are then linked together and located using the utilities LINK86 and LOC86. The utility LOC86 is invoked using controls that cause it to generate the PROM bootstrap instruction, the register initialization code sequence, and the main program entry instruction. Thereafter PROM programming software is used first to create PROM files from the LOC86 output, and then to program the four 2732A PROMs on a universal programmer. Appendix A shows the submit file used to automate linking, locating, and PROM file generation; also included are the print files produced by LINK86 and LOC86.

Appendix B shows the compiler list file for the source module MAINT.P86 containing the main program and a number of supporting routines. The main program first performs initializations of RAM, interrupts, line printer, serial interface, edit buffer, and MBM. After showing the sign-on and command menu, the main program then enters a looping top-level command key interpreter (CKI). The CKI first gives the operator one of two prompts: an arrowhead if the job buffer is being

edited, otherwise, a period. The CKI then accepts any one of the following command keystrokes:

H	shows help page
0-9	executes MBM job file d0-d9 from current decade d
M	invokes memory servicing routine
I	invokes I/O servicing routine
J	invokes job servicing routine
R	displays interrupt tally counters
S	selects new memory segment
<space>	repeats last operation
<cntl C>	exits job buffer append mode
<cntl B>	invokes MBM backup routine
<cntl R>	invokes MBM restore routine
<return>	shows command menu
<esc>	reinitializes MBM and shows sign-on

Three special keystrokes are intercepted by the terminal I/O routines included in MAINT.P86. The <cntl P> keystroke sets a flag indicating that the operator wants to dump the current screen image to printer. During a memory or I/O read operation, the operator may halt the screen display using <cntl S> and restart it with <cntl Q>. In a special category is the <esc> key which serves two uses in addition to those shown previously. During a memory or I/O read operation, <esc> causes an abort and return to the CKI. Another use of <esc> is recognized by most command-servicing code sequences. Because CKI command servicing (usually) requests additional keystrokes from the operator to complete the setup of the command line, the <esc> key is interpreted at such times as an abort request.

A MAINT job file is a block of ASCII text that may exist either as one of 100 MBM files or in the job buffer used for editing. Job files always take the form shown in figure 2; they are 1,280 bytes in length and consist of a title plus a command sequence. The title may be up to 63 characters long and is always delimited (terminated) by an ASCII NUL character (a zero byte). The command sequence may be up to 1,215 characters in length and is broken into command lines, each delimited by a <return> character. The end of the sequence is delimited by a byte containing all ones. Within the MBM, the 100 job files are mapped, as shown in figure 3. The MBM is configured by software to a granularity of 64 bytes per block using error correction. A job file thus occupies 20 blocks, 100 files occupy 2,000 of the available 2,048 blocks, and the top 48 blocks are not used.

The routine titled J$CMD, one of the major command servicing routines in MAINT.P86, handles all operations related to job files. These operations are: selecting an MBM job file decade, displaying the ten job titles for the currently selected decade, executing an MBM file from the current decade, executing the contents of the job buffer, and handling several job buffer editing functions. The job buffer editor in turn has an internal command interpreter permitting the operator to perform two categories of operations. First of all, the editor performs two important MBM-related operations: loading any one of the 100 MBM files into the job buffer and saving the job buffer contents as a new (overwritten) MBM file. In addition, the editor allows the following job buffer modifications: erasing the title block and command sequence, inserting a new job title, deleting a designated command line from the command sequence, and appending command lines to the end of the command sequence. While in the append mode, command lines are executed individually as entered. The <cntl C> character forces an exit from the append mode.

The CKI recognizes two infrequently used special characters that do not appear in the menu or on the help page: <cntl B> to create an MBM backup file and <cntl R> to restore the contents of the MBM from an MBM backup file. Backup file operations are done by means of an external 64-Kbyte RAM buffer mapped at addresses 0B0000H through 0BFFFFH. This buffer shown in figure 4 is structured and copied to and from the desired hard disk file using utilities created at Ames-Dryden. The header is a 128-byte block filled with a bit pattern that uniquely identifies the file as an MBM backup file. The directory consists of 100 pairs of WORD arguments which indicate the position and length of each job file in the body portion of the backup file. When a backup file is to be created, the operator must first specify whether all decades or only selected decades are to be backed up. The desired decades are then scanned and each nonempty MBM file is copied to the backup file buffer in condensed form such that the unused portion of the job's command sequence block is truncated. Since normally far less than half of a job's command sequence block is ever used, the entire 128-Kbyte MBM is easily backed up in 64-Kbytes. When the MBM is restored from a backup file, only those jobs present (nonzero length) are written into the MBM.

Appendix C shows the compiler list file for the source module LP.P86 that contains all the servicing routines for the line printer. The routine INITIALIZE$PRINTER sets up the PPI control register, places the printer in the on-line state, resets the state variable PTRFLG, and initializes the buffer CRT$LINE that contains the screen image. The routine PRINTER$BUFFER is called whenever a character is written to the terminal so that the same character may be placed in CRT$LINE. The routine PRINTER$SNAPSHOT checks whether a <cntl P> has been entered; if so, it copies CRT$LINE to a second buffer called PTR$LINE and changes PTRFLG to indicate that PTR$LINE is ready. If the printer is off-line (PTRFLG=0) or if the previous dump is still in progress (PTRFLG>1), the <cntl P> is ignored. The routine PRINTER$SERVICE is called from all wait loops and thus serves as the background task for managing the dump of PTR$LINE to printer. This routine is controlled by PTRFLG that iterates through several states to properly position two snapshots on each 11-inch page.

Appendix D shows the compiler list file for the source module RUPTS.P86 that contains all the servicing routines for MAINT interrupts. The routine INITIALIZE$INTERRUPTS loads the 256 interrupt vectors, sets up various programmable interrupt controller (PIC) registers, unmasks the eight PIC interrupt inputs, clears the edge-triggered flip-flop, and enables interrupts. The eight bus inter-

rupts are serviced as interrupts 32 to 39 using the PIC while the bus timeout (BTO) interrupt is serviced as interrupt 2 using the nonmaskable interrupt (NMI) pin on the 8086. These nine routines each increment individual counters while the remaining 247 interrupts are not used and share a common counter. The BTO interrupt routine also sets two flags, one in MAINT called BTO$FLAG, and the other externally in the XAIDS peripheral processor (PERPRO). BTO$FLAG is used by various CKI servicing routines to terminate read or write operations whenever BTO occurs. The PERPRO flag causes a bell signal at the main operator's terminal and flashes a BTO warning LED on the console.

Appendix E shows the compiler list file for the source module MBM.P86 that contains all the servicing routines for the iSBX-251 MBM. The routine INITIALIZE$251 sends two commands to the MBM controller chip: command 9 first aborts any operations that may be in progress, and command 1 then performs a complete initialization of the controller. INITIALIZE$251 also writes a message to the terminal that informs the operator of the outcome of the initialization. The routine READ$251 is typed BYTE and performs all read operations from MBM to RAM, returning the success/fail status of the operation. It requires three arguments: a WORD specifying the number of blocks to be read, a WORD specifying the initial block, and a POINTER to the destination buffer. The routine WRITE$251 transfers the contents of the job editor buffer to the MBM; a single byte argument specifies the file number (0-99).

Four important supporting routines are used locally within MBM.P86: SETUP$251, COMMAND$251, EXECUTE$251, and COMPLETE$251. SETUP$251 merely loads up several registers in the MBM controller chip and is therefore not a typed routine. However it does require two WORD arguments specifying the number of blocks involved for the forthcoming operation and the first block to be transferred. The remaining three routines are typed BYTE and return the success/fail status of the requested operation; error messages are displayed on the terminal whenever an operation fails. COMMAND$251 requires a single BYTE argument specifying the desired command; EXECUTE$251 and COMPLETE$251 require no arguments.

Several routines in MBM.P86 have a timeout feature that aborts the routine if excessive time is used for a particular operation. This prevents, for example, a hangup if the number of bytes transferred to or from the MBM does not exactly match the expected value. Whenever such a timeout occurs, the routine writes an appropriate message to the terminal, the INITIALIZE$251 routine is called, and a fail status is returned where applicable.

EXAMPLES OF TYPICAL OPERATIONS

To use MAINT, a separate RS-232 terminal is required that is connected to the system interface panel inside the XAIDS rack (accessable from the left rear door). This terminal may employ any baud rate in the range 110 to 19,200. When the XAIDS is powered up, the MAINT firmware enters a lockon loop that senses one or two capital letter 'U' keystrokes to determine the baud rate. The program then displays a command menu that includes a 'Help' command, whereupon the operator may perform any desired memory or I/O operation. The examples given in this section were generated on an XAIDS unit under development in the XAIDS laboratory. The terminal used was a standard CRT with the baud rate set to 19,200. The displays cited were dumped

to a line printer and are presented in the exact order in which the operations were performed.

The system was powered up and an uppercase <U> was entered on the keyboard. Display 1 shows the CRT screen as it appeared following baud rate lockon. A similar display (without the baud rate message) will be generated by the TLCI whenever <esc> is entered. Display 2 shows the result of entering the <H> command at this point. This help page may be called up at any time; if called while the job editor is in append mode, the H command is purged from the job buffer.

Display 3 shows a sequence where the operator wishes to examine the contents of an existing MBM file known only by the title: "Scan PERPRO RAM". The first step is to determine the file number by selecting decades and examining job titles until the file is found. Job decade 0 was selected first by entering <J> <0> and file 00 was identified as being the one we wished to examine. Next the job editor was invoked by entering <J> <E>; calling up the editor always produces a listing of the job currently in the job buffer. Since in our case the job buffer was empty, the file number is ??, the title is blank, and no command lines are present. Next the desired file was loaded into the buffer by entering <L> <0> <return>. The file is seen to contain a two line command sequence consisting of the selection of segment C000 followed by a Memory Read Byte 4000 to 7FFF. Finally, the job editor was exited by entering <esc>.

Display 4 shows the result of partial execution of file 00, complete execution of files 01, 02, and 03, and finally attempted execution of empty file 04. Since the selected decade was already 0, file 00 was executed by simply entering <0>. It was quickly aborted using the <esc> key and the TLCI prompt again appeared. Thereafter <1> was entered followed by <2>, <3>, <4>, and <cntl P>. This example demonstrates the ease with which several jobs within a decade can be quickly performed in succession with a minimum of keystrokes.

Display 5 shows an example of the use of the job editor to create a temporary job in the job buffer. The case assumed a desire to display the first 16 bytes of PERPRO RAM which falls in the address range C4000 to C400F. First the job editor was entered and job 00 is discovered already stored there. The existence of the file number on the screen guarantees that it is indeed a duplicate of the contents of MBM file 00, either loaded from MBM and left undisturbed, or created by the editor and recently written to MBM file 00 (the former in our case). The job buffer was erased by entering <E> followed by <Y>, and the append mode was engaged by entering <A>. The segment C400 was selected by entering <S> <C> <4> <0> <0> <return>, thus creating a new line 1 in the job buffer. A new line 2 was created by entering <M> <R> <B> <0> <return> <F> <return>, which executed a dump of the desired block of PERPRO RAM. Finally append mode was terminated by entering <cntl C> and the editor was exited by entering <esc>. Temporary jobs such as this remain undisturbed as long as the editor is not used to alter the job buffer.

Display 6 shows the result of executing the contents of the job buffer five times: first by entering <J> <space> and then by entering <space> four times. Each time the buffer is executed, the job file number and title are first displayed, followed by the actual execution of the command lines. In our case the file number is missing and the title is blank. Select segment command lines in jobs are executed

but never displayed; the selected segment is always displayed during execution of memory-related operations.

Display 7 shows an example of a somewhat more lengthy job buffer and its execution. The job consists of eight lines involving an assortment of memory operations and was created similarly to the steps described for Display 5. The display snapshot shows the results of the following steps: (1) the job editor was entered to examine its contents, (2) the editor was exited using <esc>, and (3) the job buffer was executed using <J> <space>. The job demonstrates the use of the memory fill byte (MFB), memory read byte (MRB), memory substitute (MS), and memory read ASCII (MRA) commands.

Display 8 shows an MBM backup operation where only a few selected decades were backed up. The CKI command <cntl B> was entered first, then the subsequent queries were answered in the sequence <y> <n> <0> <2> <6> <return>. All nonempty files present in the three selected decades were copied to the external buffer, and last, the backup file directory was created. Such backup (as well as restore) operations can be done only when the XAIDS is not executing a user I/O job. This is because it is necessary to write the MBM backup buffer contents to a hard disk file for permanent storage using a utility program called "SAVE". A companion utility called "LOAD" is used to copy a MBM backup file into this same buffer so that the CKI command <cntl R> can restore whatever files are present back into the MBM.

Display 9 shows a few examples of I/O read operations. First <return> was pressed to display the command menu. This was followed by the selection of job decade 2 by entering <J> <2>. Last, jobs 25, 26, and 27 were executed by entering <5> <6> <7>. All three of these jobs are of the I/O read-byte (IRB) type. The 16-bit I/O address is displayed at the start of each line with the label "I/O" prefixed to clearly distinguish them as I/O readouts and not memory readouts.

CONCLUDING REMARKS

A stand-alone maintenance processor for a multiprocessing system was developed at Ames-Dryden and placed in service supporting system software V/V and hardware diagnostics. The latest mechanization provides an independent terminal interface, a line printer interface, and a magnetic bubble memory for nonvolatile storage of often-used job sequences. The software suite includes a large set of data display formatters, routines for snapshot printer dumps of terminal displays, file management routines for storage and retrieval of 100 job files, and routines for backup and restoration of the entire contents of bubble memory.

The major contribution of the maintenance processor concept was improved efficiency in testing of system hardware and software. Experience to date with the magnetic bubble memory indicates that it has adequate speed and is very reliable, making this type of nonvolatile memory ideally suited for this application. The ability to selectively hardcopy screen displays further speeded testing operations. Only off-the-shelf commercial hardware was used, resulting in a highly cost-effective subsystem.

The author believes that where suitable hardware and space is available, a highly useful stand-alone processor configured along these lines could be easily

designed or retrofitted into most multiprocessing systems. Although such a resource does remain idle most of the time, it is nevertheless invaluable when needed (usually on short notice) to access bus-mapped memory or hardware for whatever reason.

National Aeronautics and Space Administration
Ames Research Center
Dryden Flight Research Facility
Edwards, California, June 3, 1987

APPENDIX A

Files MAINT.CSD, MAINT.MP1, and MAINT.MP2

MAINT.CSD is a command sequence definition file (or simply, SUBMIT file) that is invoked by the operator using the RMX86 command line "SUBMIT MAINT." The SUBMIT utility adds the ".CSD" extension, fetches the MAINT.CSD file, and submits commands therefrom as if they were entered by the operator line by line. The final six lines of this file relate to the PROM programming phase which is handled separately.

MAINT.MP1 is the print file produced by the LINK86 utility as it links the four object files specified in the SUBMIT file to create the composite file called MAINT.LNK.

MAINT.MP2 is the print file produced by the LOC86 utility as it locates MAINT.LNK to create the absolute file MAINT.

No listing is produced by the iPPS software as it dissects the MAINT file into the two PROM files MAINT.LO and MAINT.HI.

```
; File MAINT.CSD   14 January 1987   R. D. Glover

; Submit file for creating PROM files for XAIDS Maintenance Processor

delete maint.lnk, maint, maint.mp*

link86 maint.obj,              &       Main program + terminal routines
       rupts.obj,              &       Interrupt support routines
       mbm.obj,                &       Magnetic bubble memory routines
       lp.obj,                 &       Line printer routines
       :lang:plm86.lib         &       PLM library routines
       to maint.lnk print(maint.mp1)

loc86 maint.lnk to maint                                        &
      order(classes(data,stack,code))                           &
      initcode(0fffd0h)                                         &
      addresses(classes(data(400h),code(0fc000h)))              &
      objectcontrols(purge)                                     &
      print(maint.mp2)                                          &
      printcontrols(nopurge)                                    &
      symbolcolumns(3)                                          &
      bootstrap

delete maint.lo, maint.hi

ipps
i h 86
format maint(0fc000h,0ffffh)
3
2
1
0 to maint.lo
1 to maint.hi

exit

; Invoke IPPS and perform followins :
;     init hex 86
;     type 2732a
;     copy maint.lo to prom
;     copy maint.hi to prom
;     exit
```

```
iRMX 86 8086 LINKER, V2.5

INPUT FILES: MAINT.OBJ, RUPTS.OBJ, MBM.OBJ, LP.OBJ, :LANG:PLM86.LIB
OUTPUT FILE: MAINT.LNK
CONTROLS SPECIFIED IN INVOCATION COMMAND:
 PRINT(MAINT.MP1)
DATE:  01/14/87  TIME:  10:23:13


LINK MAP OF MODULE MAINT_P86

LOGICAL SEGMENTS INCLUDED:
LENGTH ADDRESS ALIGN  SEGMENT         CLASS          OVERLAY
 3705H ------    G    CODE            CODE
 1B5CH ------    W    DATA            DATA
 0086H ------    W    STACK           STACK
 0000H ------    W    MEMORY          MEMORY
 00E0H B0000H    A    (ABSOLUTE)
 0170H B0080H    A    (ABSOLUTE)
 F2F0H B0210H    A    (ABSOLUTE)
 0400H 00000H    A    (ABSOLUTE)
 0001H C701EH    A    (ABSOLUTE)
 0000H ------    G    ??SEG

INPUT MODULES INCLUDED:
MAINT.OBJ(MAINT_P86)
RUPTS.OBJ(RUPTS_P86)
MBM.OBJ(MBM_P86)
LP.OBJ(LP_P86)
:LANG:PLM86.LIB(LQ_DWORD_MULTIPLY)
```

```
iRMX 86 8086 LOCATER, V2.5

INPUT FILE: MAINT.LNK
OUTPUT FILE: MAINT
CONTROLS SPECIFIED IN INVOCATION COMMAND:
   TO MAINT ORDER(CLASSES(DATA,STACK,CODE)) INITCODE(
  0FFFD0H) ADDRESSES(CLASSES(DATA(400H),CODE(0FC000H))) OBJECTCONTROLS(PURGE)
   PRINT(MAINT.MP2) PRINTCONTROLS(NOPURGE) SYMBOLCOLUMNS(3) BOOTSTRAP
DATE:  01/14/87  TIME:  10:23:30


SYMBOL TABLE OF MODULE MAINT_PB6

BASE    OFFSET TYPE SYMBOL              BASE    OFFSET TYPE SYMBOL              BASE    OFFSET TYPE SYMBOL

0040H   0C08H  PUB  BTOFLAG             0040H   0560H  PUB  EDITBUFFER          0040H   0BE0H  PUB  INTCOUNTER
0040H   0BC8H  PUB  KEY                 0040H   0C00H  PUB  NMICOUNTER          0040H   0C04H  PUB  UNKNOWNCOUNTER
FC00H   16F0H  PUB  HEX1                FC00H   1707H  PUB  HEX2                FC00H   32F8H  PUB  INITIALIZE251
FC00H   303FH  PUB  INITIALIZEINTERR     FC00H   3460H  PUB  INITIALIZEPRINTE     FC00H   36F0H  PUB  LO_DWORD_MUL
                    -UPTS                                    -R
FC00H   3489H  PUB  PRINTERBUFFER        FC00H   35A3H  PUB  PRINTERSERVICE       FC00H   353DH  PUB  PRINTERSNAPSHOT
FC00H   3371H  PUB  READ251              FC00H   1658H  PUB  WRITE               FC00H   33DEH  PUB  WRITE251


MEMORY MAP OF MODULE MAINT_PB6

MODULE START ADDRESS  PARAGRAPH = FFFDH  OFFSET = 0006H
SEGMENT MAP

START     STOP      LENGTH ALIGN NAME            CLASS          OVERLAY

00000H    003FFH     0400H   A   (ABSOLUTE)
00400H    01F5BH     1B5CH   W   DATA            DATA
01F5CH    01FE1H     0086H   W   STACK           STACK
B0000H    B007FH     0080H   A   (ABSOLUTE)
B0080H    B020FH     0190H   A   (ABSOLUTE)
B0210H    BFFFFH     FDF0H   A   (ABSOLUTE)
C701EH    C701EH     0001H   A   (ABSOLUTE)
FC000H    FF704H     3705H   G   CODE            CODE
FF710H    FF710H     0000H   G   ??SEG
FFFD0H    FFFE8H     0019H   A   ??LOC86_INITCO  CODE
                                 -DE
FFFF0H    FFFF4H     0005H   A   (ABSOLUTE)
FFFF6H    FFFF6H     0000H   W   MEMORY          MEMORY


GROUP MAP

ADDRESS  GROUP OR SEGMENT NAME
FC000H   CGROUP
          CODE
00400H   DGROUP
          DATA
```

APPENDIX B

File MAINT.P86

The following listing shows the PLM86 source language for the module containing the MAINT main program, a set of command servicing routines, plus a set of terminal I/O routines. These are bundled into a single module since they are tightly coupled through numerous common variable declarations. This module declares PUBLIC the following variables and procedures:

KEY	a BYTE variable which contains the most recent operator keystroke.
EDIT$BUFFER	a 1280 BYTE array which contains the job created by the job editor.
WRITE	a PROCEDURE with a POINTER argument that identifies an ASCII string (null delimited) to be written to the terminal.
HEX1	a PROCEDURE with a BYTE argument whose lower nibble is to be written to the terminal as one hexadecimal character.
HEX2	a PROCEDURE with a BYTE argument to be written to the terminal as two hexadecimal characters.

```
iRMX 86 PL/M-86 V2.7 COMPILATION OF MODULE MAINT_P86
OBJECT MODULE PLACED IN MAINT.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 MAINT.P86


            $COMPACT ROM NOINTVECTOR OPTIMIZE(3)
            $TITLE('NASA/ADFRF XAIDS MAINTENANCE PROCESSOR PROGRAM  14 JAN 1987')

            /*   NASA AMES DRYDEN FLIGHT RESEARCH FACILITY                  */

            /*   Extended Aircraft Interrogation & Display System  (XAIDS)  */
            /*   Maintenance Processor (MAINT) program including line printer */
            /*   and SBX251 Bubble Memory Multimodule support.              */

            /*   Written by Richard D. Glover, Research Engineering Division  */

  1         MAINT_P86: DO ;

            /* EXTERNAL ROUTINES DECLARATIONS */

  2   2     INITIALIZE$INTERRUPTS: PROCEDURE EXTERNAL ; END ;
  4   2     INITIALIZE$PRINTER:    PROCEDURE EXTERNAL ; END ;
  6   2     PRINTER$SNAPSHOT:      PROCEDURE EXTERNAL ; END ;
  8   2     PRINTER$SERVICE:       PROCEDURE EXTERNAL ; END ;
 10   2     PRINTER$BUFFER:        PROCEDURE (CHAR) EXTERNAL ; DECLARE CHAR BYTE ; END ;
 13   2     INITIALIZE$251: PROCEDURE EXTERNAL ; END ;
 15   2     WRITE$251: PROCEDURE (FILE) EXTERNAL ; DECLARE FILE BYTE ; END ;
 18   1     READ$251:  PROCEDURE (NBLOCKS,START,PTR) BYTE EXTERNAL ;
 19   2                DECLARE (NBLOCKS,START) WORD, PTR POINTER ; END ;

            /* EXTERNAL BACKUP/RESTORE BUFFER DECLARATIONS */

 21   1     DECLARE BACKUP$RESTORE$BUFFER LITERALLY '0B0000H' ;
 22   1     DECLARE HDR     (32) DWORD AT (BACKUP$RESTORE$BUFFER+000) ;
 23   1     DECLARE DIR    (100) STRUCTURE (INDEX WORD, LENGTH WORD)
                                  AT (BACKUP$RESTORE$BUFFER+128) ;
 24   1     DECLARE INFO (65008) BYTE  AT (BACKUP$RESTORE$BUFFER+528) ;

            /* SBC 86/05 BOARD I/O PORT MAPPING */

 25   1     DECLARE PPIA       LITERALLY '0C8H' ;   /* PROGRAMMABLE PERIPHERAL INTERFACE */
 26   1     DECLARE PPIB       LITERALLY '0CAH' ;
 27   1     DECLARE PPIC       LITERALLY '0CCH' ;
 28   1     DECLARE PPIFLG     LITERALLY '0CEH' ;
 29   1     DECLARE PIT0       LITERALLY '0D0H' ;   /* PROGRAMMABLE INTERVAL TIMER */
 30   1     DECLARE PIT1       LITERALLY '0D2H' ;
 31   1     DECLARE PIT2       LITERALLY '0D4H' ;
 32   1     DECLARE PITMOD     LITERALLY '0D6H' ;
 33   1     DECLARE USDATA     LITERALLY '0D8H' ;   /* USART  */
 34   1     DECLARE USFLAG     LITERALLY '0DAH' ;

            /* I/O STATUS FLAGS */

 35   1     DECLARE RXRDY                 LITERALLY 'SHR(INPUT(USFLAG),1)' ;
 36   1     DECLARE TXRDY                 LITERALLY '    INPUT(USFLAG)   ' ;
```

```
                /* MISCELLANEOUS DECLARATIONS */

 37  1      DECLARE NULL        LITERALLY '000H' ;
 38  1      DECLARE CNTL$B      LITERALLY '002H' ;
 39  1      DECLARE CNTL$C      LITERALLY '003H' ;
 40  1      DECLARE BEL         LITERALLY '007H' ;
 41  1      DECLARE BS          LITERALLY '008H' ;
 42  1      DECLARE LF          LITERALLY '00AH' ;
 43  1      DECLARE CR          LITERALLY '00DH' ;
 44  1      DECLARE CNTL$Q      LITERALLY '011H' ;
 45  1      DECLARE CNTL$R      LITERALLY '012H' ;
 46  1      DECLARE CNTL$S      LITERALLY '013H' ;
 47  1      DECLARE ESC         LITERALLY '01BH' ;
 48  1      DECLARE SPACE       LITERALLY '020H' ;
 49  1      DECLARE RUB         LITERALLY '07FH' ;
 50  1      DECLARE JOB$IDLE    LITERALLY 'JOB$MODE=0' ;
 51  1      DECLARE JOB$SETUP   LITERALLY 'JOB$MODE=1' ;
 52  1      DECLARE JOB$RUN     LITERALLY 'JOB$MODE=2' ;
 53  1      DECLARE CR$OR$LF    LITERALLY '(KEY=CR OR KEY=LF)' ;
 54  1      DECLARE EXIT$SETUP LITERALLY '(JOB$SETUP AND KEY=CNTL$C)' ;
 55  1      DECLARE TRUE        LITERALLY '0FFH' ;
 56  1      DECLARE FALSE       LITERALLY '0' ;
 57  1      DECLARE FOREVER     LITERALLY 'WHILE TRUE' ;

 58  1      DECLARE INT$COUNTER (8) DWORD EXTERNAL ; /* MULTIBUS INTERRUPT TALLY COUNTERS */
 59  1      DECLARE WDI$COUNTER     DWORD EXTERNAL ; /* DEADMAN TIMER INTERRUPT COUNTER */
 60  1      DECLARE UNKNOWN$COUNTER DWORD EXTERNAL ; /* REMAINING RUPT TYPES SHARE COUNTER */
 61  1      DECLARE BTO$FLAG         BYTE EXTERNAL ; /* BUS TIMEOUT FLAG */

 62  1      DECLARE MEM$PTR      POINTER ;
 63  1      DECLARE BYTE$DATA    BASED MEM$PTR BYTE ;
 64  1      DECLARE WORD$DATA    BASED MEM$PTR WORD ;
 65  1      DECLARE DWORD$DATA   BASED MEM$PTR DWORD ;
 66  1      DECLARE POINTER$DATA BASED MEM$PTR POINTER ;
 67  1      DECLARE BYTE$SAVE    BYTE ;
 68  1      DECLARE WORD$SAVE    WORD ;
 69  1      DECLARE DWORD$SAVE   DWORD ;
 70  1      DECLARE POINTER$SAVE POINTER ;
 71  1      DECLARE PTR          STRUCTURE (OFFSET WORD, SEG WORD) AT (@MEM$PTR) ;
 72  1      DECLARE SEGMENT      LITERALLY 'PTR.SEG' ;     /* MEMORY ADDRESS SEGMENT */
 73  1      DECLARE FROM$ADDR    WORD AT (@MEM$PTR)  ;     /* BLOCK START ADDR */
 74  1      DECLARE INCREMENT    WORD ;                    /* FROM$ADDR STEP SIZE */

 75  1      DECLARE DATA$TYPE    BYTE ;                    /* 'B', 'W', 'D', OR 'P' */
 76  1      DECLARE HEX$DWORD    DWORD ;                   /* GETHEX OUTPUT */
 77  1      DECLARE HEX$WORD     WORD AT (@HEX$DWORD) ;    /* LOWER 16 BITS OF HEX$DWORD */
 78  1      DECLARE HEX$BYTE     BYTE AT (@HEX$DWORD) ;    /* LOWERMOST 8 BITS OF HEX$DWORD */
 79  1      DECLARE FILLER$DWORD DWORD ;                   /* FILLER VALUE */
 80  1      DECLARE FILLER$WORD  WORD AT (@FILLER$DWORD) ; /* LOWER 16 BITS */
 81  1      DECLARE FILLER$BYTE  BYTE AT (@FILLER$DWORD) ; /* LOWERMOST 8 BITS */

 82  1      DECLARE TO$ADDR           WORD ; /* BLOCK STOP ADDR */
 83  1      DECLARE KEY         BYTE PUBLIC ; /* LAST OPERATOR KEYSTROKE */
 84  1      DECLARE CHBUF (64)        BYTE ; /* "KEYSIN" CHARACTER BUFFER */
 85  1      DECLARE NCHAR             BYTE ; /* # CHAR IN CHBUF */
 86  1      DECLARE CMD$SAVE          BYTE ; /* COMMAND SAVE */
 87  1      DECLARE USART$INIT$BYTE   BYTE ; /* USART MODE INSTRUCTION BYTE */
```

```
 88  1      DECLARE OUTPUT$DISABLED   BYTE ; /* SCREEN OUTPUT CONTROL */
 89  1      DECLARE JOB$BUFFER$PTR  POINTER ; /* POINTER TO CURRENT JOB BUFFER */
 90  1      DECLARE JOB$BUFFER BASED JOB$BUFFER$PTR (1280) BYTE ; /* CURRENT JOB BUFFER */
 91  1      DECLARE JOB$BUFFER$INDEX   WORD ; /* INDEX TO NEXT CHAR IN CURRENT JOB BUFFER */
 92  1      DECLARE JOB$MODE           BYTE ; /* 0=IDLE  1=SETUP  2=RUN  3=EDIT */
 93  1      DECLARE FILE$DECADE        BYTE ; /* 0 THRU 9 */
 94  1      DECLARE FILE$NUMBER        BYTE ; /* 0 THRU 99 */
 95  1      DECLARE FILE$BUFFER (1280) BYTE ; /* USED BY FILE LOADER COMMANDS 0-9 */
 96  1      DECLARE EDIT$FILE$NUMBER   BYTE ; /* MATCHING FILE NUMBER (=255 IF NOT) */
 97  1      DECLARE EDIT$BUFFER (1280) BYTE PUBLIC ; /* USED BY JE COMMAND */
 98  1      DECLARE ITEM$INDEX   (128) WORD ; /* EDIT BUFFER ITEM INDEXES */
 99  1      DECLARE NITEMS             BYTE ; /* NUMBER OF ITEMS IN EDIT BUFFER */
100  1      DECLARE BLOCK         (64) BYTE ; /* BUBBLE MEMORY TEMP STORAGE */
101  1      DECLARE ASCII (16)         BYTE      DATA ('0123456789ABCDEF') ;


102  1      ERASE$EDIT$BUFFER: PROCEDURE ;
103  2      CALL SETB(0,@EDIT$BUFFER,1280) ;
104  2      EDIT$BUFFER(64)  = 0FFH ;
105  2      ITEM$INDEX(0)    = 64   ;
106  2      EDIT$FILE$NUMBER = 255  ;
107  2      NITEMS           = 0    ;
108  2      END ;

109  1      GENERATE$ITEM$INDEXES: PROCEDURE;    /* SET UP ITEM$INDEX ARRAY */
110  2      CALL SETW(64,@ITEM$INDEX,128) ;
111  2          DO NITEMS = 0 TO 127 ;
112  3          IF EDIT$BUFFER(ITEM$INDEX(NITEMS)+1) = 0 THEN    /* END OF BUFFER */
113  3              RETURN ;
114  3          ITEM$INDEX(NITEMS+1) = ITEM$INDEX(NITEMS)
                    + FINDB(@EDIT$BUFFER(ITEM$INDEX(NITEMS)+1),0FFH,1280) + 1 ;
115  3          END ;
116  2      END ;

117  1      LOAD$JOB: PROCEDURE ;
118  2      DECLARE FILE BYTE, I WORD ;
119  2      IF JOB$SETUP THEN               /* ILLEGAL REQUEST */
120  2          DO ;
121  3          JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 2 ;
122  3          CALL ILLEGAL$CMD ;
123  3          RETURN ;
124  3          END ;
125  2      FILE = 10*FILE$DECADE + (KEY AND 0FH) ;
126  2      IF FILE$NUMBER <> FILE THEN
127  2          DO ;
128  3          FILE$NUMBER = FILE ;
129  3          IF NOT READ$251(20,20*FILE,@FILE$BUFFER) THEN
130  3              DO ;
131  4              FILE$NUMBER = 255 ;
132  4              RETURN ;
133  4              END ;
134  3          END ;
135  2      CALL WRITE(@('Job file ',0)) ;
136  2      CALL CO(ASCII(FILE$DECADE)) ;
137  2      CALL CO(KEY) ;
138  2      IF FILE$BUFFER(65) <> 0 THEN
139  2          DO ;
```

```
140   3          CALL WRITE(@(' = ',0)) ;
141   3          CALL WRITE(@FILE$BUFFER) ;
142   3          JOB$BUFFER$PTR = @FILE$BUFFER ;
143   3          JOB$BUFFER$INDEX = 64 ;
144   3          JOB$RUN ;
145   3          END ;
146   2      ELSE
               CALL WRITE(@(' is empty.',0)) ;
147   2      END ;

148   1      CO: PROCEDURE (CHAR) ;          /* OUTPUT ASCII CHARACTER TO TERMINAL */
149   2      DECLARE CHAR BYTE ;
150   2      IF OUTPUT$DISABLED THEN RETURN ;
152   2          DO WHILE NOT TXRDY ;
153   3          CALL PRINTER$SERVICE ;
154   3          END ;
155   2      OUTPUT(USDATA) = CHAR ;
156   2      CALL PRINTER$BUFFER(CHAR) ;
157   2      END ;

158   1      SHOW$KEY: PROCEDURE ;
159   2      IF KEY < SPACE THEN
160   2          DO CASE KEY ;
161   3          CALL WRITE(@('(NULL)',0)) ;
162   3          CALL WRITE(@('(SOH)',0)) ;
163   3          CALL WRITE(@('(STX)',0)) ;
164   3          CALL WRITE(@('(ETX)',0)) ;
165   3          CALL WRITE(@('(EOT)',0)) ;
166   3          CALL WRITE(@('(ENQ)',0)) ;
167   3          CALL WRITE(@('(ACK)',0)) ;
168   3          CALL WRITE(@('(BEL)',0)) ;
169   3          CALL WRITE(@('(BS)',0)) ;
170   3          CALL WRITE(@('(HT)',0)) ;
171   3          CALL WRITE(@('(LF)',0)) ;
172   3          CALL WRITE(@('(VT)',0)) ;
173   3          CALL WRITE(@('(FF)',0)) ;
174   3          CALL WRITE(@('(CR)',0)) ;
175   3          CALL WRITE(@('(SO)',0)) ;
176   3          CALL WRITE(@('(SI)',0)) ;
177   3          CALL WRITE(@('(DLE)',0)) ;
178   3          CALL WRITE(@('(DC1)',0)) ;
179   3          CALL WRITE(@('(DC2)',0)) ;
180   3          CALL WRITE(@('(DC3)',0)) ;
181   3          CALL WRITE(@('(DC4)',0)) ;
182   3          CALL WRITE(@('(NAK)',0)) ;
183   3          CALL WRITE(@('(SYN)',0)) ;
184   3          CALL WRITE(@('(ETB)',0)) ;
185   3          CALL WRITE(@('(CAN)',0)) ;
186   3          CALL WRITE(@('(EM)',0)) ;
187   3          CALL WRITE(@('(SUB)',0)) ;
188   3          CALL WRITE(@('(ESC)',0)) ;
189   3          CALL WRITE(@('(FS)',0)) ;
190   3          CALL WRITE(@('(GS)',0)) ;
191   3          CALL WRITE(@('(RS)',0)) ;
192   3          CALL WRITE(@('(US)',0)) ;
193   3          END ;
194   2      ELSE IF KEY = RUB THEN
```

```
195   2         CALL WRITE(@('(RUB)',0)) ;
196   2       ELSE
                CALL CO(KEY) ;
197   2       END ;

198   1       BLANK: PROCEDURE(N);            /* SEND SPACE CHARACTERS TO TERMINAL */
199   2       DECLARE (N,I) BYTE ;
200   2         DO I = 1 TO N ;
201   3         CALL CO(SPACE) ;
202   3         END ;
203   2       END ;

204   1       BEEP: PROCEDURE;                /* SEND (BEL) CHARACTER TO TERMINAL */
205   2       CALL CO(BEL) ;
206   2       END ;

207   1       PURGE$JOB$ENTRY: PROCEDURE ;
208   2       IF JOB$SETUP THEN
209   2         DO WHILE JOB$BUFFER(JOB$BUFFER$INDEX)<>0FFH ;
210   3         JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 1 ;
211   3         END ;
212   2       JOB$BUFFER(JOB$BUFFER$INDEX+1) = 0 ;
213   2       END ;

214   1       WRITE: PROCEDURE (STRING$PTR) PUBLIC ;   /* WRITE MESSAGE STRING TO TERMINAL */
215   2       DECLARE STRING$PTR POINTER, STRING BASED STRING$PTR (65535) BYTE, I WORD ;
216   2         DO I = 0 TO 65535 ;
217   3         IF STRING(I) = 0 THEN RETURN ;
219   3         CALL CO(STRING(I)) ;
220   3         END ;
221   2       END ;

222   1       ILLEGAL$CMD: PROCEDURE;         /* COMMAND KEY IN ERROR */
223   2       IF KEY <> CNTL$C THEN CALL BEEP ;
225   2       IF KEY > SPACE AND KEY < RUB THEN CALL CO(KEY) ;
227   2       ELSE CALL CO('@') ;
228   2       CALL WRITE(@('  illegal command',0)) ;
229   2       END ;

230   1       BTO: PROCEDURE BYTE;            /* HAS THERE BEEN A BUS TIME-OUT ? */
231   2       IF NOT BTO$FLAG THEN
232   2         RETURN FALSE ;
233   2       CALL WRITE(@(BEL,'  BTO',0)) ;
234   2       BTO$FLAG = FALSE ;
235   2       RETURN TRUE ;
236   2       END ;

237   1       CR$LF: PROCEDURE ;              /* SEND CR & LF TO TERMINAL */
238   2       CALL CO(0DH) ;
239   2       CALL CO(0AH) ;
240   2       END ;

241   1       TO$TERMINATE: PROCEDURE ;
242   2       CALL WRITE(@('To terminate, press (ESC).',0)) ;
243   2       END ;

244   1       HEX1: PROCEDURE (VAL) PUBLIC ; /* DISPLAY LOWER NIBBLE OF BYTE AS 1 HEX CHAR */
```

```
245  2     DECLARE VAL BYTE ;
246  2     CALL CO(ASCII(VAL AND 0FH)) ;
247  2     END ;

248  1     HEX2: PROCEDURE (VAL) PUBLIC ; /* DISPLAY BYTE AS 2 HEX CHARS */
249  2     DECLARE VAL BYTE ;
250  2     CALL HEX1(SHR(VAL,4)) ;
251  2     CALL HEX1(VAL) ;
252  2     END ;

253  1     HEX4: PROCEDURE (VAL) ;       /* DISPLAY WORD AS 4 HEX CHARACTERS */
254  2     DECLARE VAL WORD ;
255  2     CALL HEX2(LOW(SHR(VAL,8))) ;
256  2     CALL HEX2(LOW(VAL)) ;
257  2     END ;

258  1     HEX8: PROCEDURE (VAL) ;        /* DISPLAY DWORD AS 8 HEX CHARACTERS */
259  2     DECLARE VAL DWORD ;
260  2     CALL HEX4(LOW(SHR(VAL,16))) ;
261  2     CALL HEX4(LOW(VAL)) ;
262  2     END ;

263  1     HEX$PTR: PROCEDURE (VAL) ;     /* DISPLAY POINTER AS 8 HEX CHARACTERS */
264  2     DECLARE VAL POINTER, DW DWORD AT (@VAL) ;
265  2     CALL HEX4(LOW(SHR(DW,16))) ;
266  2     CALL CO(':') ;
267  2     CALL HEX4(LOW(DW)) ;
268  2     END ;

269  1     NEW$HR$LINE: PROCEDURE (PROMPT);  /* START NEW MEMORY READ LINE */
270  2     DECLARE PROMPT BYTE ;
271  2     CALL PRINTER$SNAPSHOT ;
272  2     CALL CR$LF ;
273  2     CALL HEX4(SEGMENT) ;
274  2     CALL CO(':') ;
275  2     CALL HEX4(FROM$ADDR) ;
276  2     CALL BLANK(1) ;
277  2     CALL CO(PROMPT) ;
278  2     CALL BLANK(2) ;
279  2     END ;

280  1     SPACER: PROCEDURE;                  /* PERFORM HEX DISPLAY LINE SPACING */
281  2     CALL BLANK(1);                      /* EACH BYTE GETS ONE BLANK */
282  2     IF ( FROM$ADDR AND 03H ) = 0 THEN   /* EVERY 4 BYTES ONE MORE */
283  2        CALL BLANK(1) ;
284  2     IF ( FROM$ADDR AND 07H ) = 0 THEN   /* EVERY 8 BYTES ONE MORE */
285  2        CALL BLANK(1) ;
286  2     END ;

287  1     NEW$IO$DISPLAY$LINE: PROCEDURE(PROMPT) ;  /* START NEW I/O DISPLAY LINE */
288  2     DECLARE PROMPT BYTE ;
289  2     CALL PRINTER$SNAPSHOT ;
290  2     CALL WRITE(@(CR,LF,'I/O  ',0)) ;
291  2     CALL HEX4(FROM$ADDR) ;
292  2     CALL BLANK(1) ;
293  2     CALL CO(PROMPT) ;
294  2     CALL BLANK(2) ;
```

```
295  2     END ;

296  1     OPERATOR$ABORT: PROCEDURE BYTE;          /* OPERATOR CONTROL KEY CHECK */
297  2     IF KEY=ESC THEN                          /* ESC */
298  2         GO TO GOT$ESC ;
299  2     ELSE IF KEY=CNTL$S THEN                  /* OPERATOR PAUSE */
300  2         DO FOREVER ;
301  3         CALL PRINTER$SERVICE ;
302  3         IF RXRDY THEN
303  3             DO ;
304  4             KEY = INPUT(USDATA) ;
305  4             CALL PRINTER$SNAPSHOT ;
306  4             IF KEY=ESC THEN
307  4                 GO TO GOT$ESC ;
308  4             IF KEY=CNTL$Q THEN           /* PROCEED */
309  4                 RETURN FALSE ;
310  4             END ;
311  3         END ;
312  2     ELSE
              RETURN FALSE ;
313  2     GOT$ESC:
          IF JOB$RUN THEN
314  2         DO ;
315  3         CALL WRITE(@(CR,LF,'Job aborted',0)) ;
316  3         JOB$IDLE ;
317  3         END ;
318  2     RETURN TRUE ;
319  2     END ;

320  1     KEY$CAPTURED: PROCEDURE BYTE ;           /* GRAB KEYIN IF AVAILABLE */
321  2     IF JOB$RUN THEN                          /* GET KEY FROM JOB BUFFER */
322  2         DO ;
323  3         IF RXRDY THEN                   /* CHECK FOR OPERATOR INTERVENTION */
324  3             DO ;
325  4             KEY = INPUT(USDATA) ;
326  4             IF OPERATOR$ABORT THEN
327  4                 RETURN TRUE ;
328  4             END ;
329  3         AGAIN:
              KEY = JOB$BUFFER(JOB$BUFFER$INDEX) ;
330  3         JOB$BUFFER$INDEX = JOB$BUFFER$INDEX + 1 ;
331  3         IF KEY = 0FFH THEN              /* END OF LINE - FETCH ANOTHER CHAR */
332  3             GO TO AGAIN ;
333  3         IF KEY=NULL THEN                /* END OF BUFFER */
334  3             DO ;
335  4             KEY = CR ;
336  4             JOB$IDLE ;
337  4             END ;
338  3         RETURN TRUE ;
339  3         END ;
340  2     ELSE                                     /* GET KEYBOARD INPUT IF WAITING */
              DO ;
341  3         IF NOT RXRDY THEN
342  3             RETURN FALSE ;
343  3         KEY = INPUT(USDATA) ;
344  3         IF KEY <> 10H THEN RETURN TRUE ;
346  3         CALL PRINTER$SNAPSHOT ;
```

```
347  3          RETURN FALSE ;
348  3          END ;
349  2      END ;

350  1      COMPLETE: PROCEDURE BYTE;              /* USED BY MR, MF, & IR COMMANDS */
351  2      DECLARE OLD$FROM$ADDR WORD ;
352  2      FROM$ADDR = (OLD$FROM$ADDR:=FROM$ADDR) + INCREMENT ;
353  2      IF FROM$ADDR < OLD$FROM$ADDR OR FROM$ADDR > TO$ADDR THEN
354  2          RETURN TRUE ;
355  2      IF NOT RXRDY THEN
356  2          RETURN FALSE ;
357  2      KEY = INPUT(USDATA) ;
358  2      RETURN OPERATOR$ABORT ;
359  2      END ;

360  1      CI: PROCEDURE ;                        /* WAIT FOR NEXT KEYIN */
361  2          DO WHILE NOT KEY$CAPTURED ;
362  3          CALL PRINTER$SERVICE ;
363  3          END ;
364  2      IF JOB$SETUP THEN                      /* PUT KEY IN BUFFER */
365  2          DO ;
366  3          IF KEY <> CNTL$C THEN
367  3              DO ;
368  4              JOB$BUFFER(JOB$BUFFER$INDEX) = KEY ;
369  4              JOB$BUFFER$INDEX = JOB$BUFFER$INDEX + 1 ;
370  4              END ;
371  3          END ;
372  2      END ;

373  1      YES$NO: PROCEDURE ;
374  2      CALL CI ;
375  2          DO WHILE KEY <> ESC ;
376  3          IF FINDB(@('YyNn'),KEY,4) < 4 THEN
377  3              DO ;
378  4              CALL CD(KEY) ;
379  4              CALL CAPITALIZE$KEY ;
380  4              RETURN ;
381  4              END ;
382  3          CALL BEEP ;
383  3          CALL CI ;
384  3          END ;
385  2      END ;

386  1      KEY$CASE: PROCEDURE (PTR) BYTE ;
387  2      DECLARE PTR POINTER, CHAR BASED PTR (*) BYTE, INDEX BYTE ;
388  2      DECLARE LENGTH LITERALLY 'CHAR(0)' ;
389  2          DO FOREVER ;
390  3          CALL CI ;
391  3          CALL CAPITALIZE$KEY ;
392  3          IF ( INDEX:=FINDB(@CHAR(1),KEY,LENGTH) ) < 0FFH THEN
393  3              RETURN INDEX ;
394  3          ELSE
                    CALL BEEP ;
395  3          END ;
396  2      END ;

397  1      GET$ANOTHER$KEY: PROCEDURE BYTE;     /* WAIT UP TO 1 SECOND FOR KEYIN */
```

```
398   2     DECLARE I BYTE ;
399   2        DO I = 1 TO 100 ;
400   3        IF KEY$CAPTURED THEN
401   3           RETURN TRUE ;
402   3        CALL TIME(100) ;        /* 10 MILLISECOND PAUSE */
403   3        END ;
404   2     RETURN FALSE ;             /* TIMEOUT */
405   2     END ;

           /*  "RUBOUT" DELETES MOST RECENT ADDITION TO CHBUF */
           /*  BEEP IS SOUNDED IF CHBUF ALREADY EMPTY OF CHARACTERS */

406   1     RUBOUT: PROCEDURE ;
407   2     IF NCHAR = 0 THEN
408   2        CALL BEEP ;
409   2     ELSE
              DO ;
410   3        NCHAR = NCHAR - 1 ;
411   3        CHBUF(NCHAR) = 0 ;
412   3        IF JOB$SETUP THEN
413   3           JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 1 ;
414   3        CALL WRITE(0,@(BS,SPACE,BS,0)) ;
415   3        END ;
416   2     END ;

417   1     ERASE: PROCEDURE;             /* EMPTY OUT CHBUF */
418   2        DO WHILE NCHAR > 0 ;
419   3        CALL RUBOUT ;
420   3        END ;
421   2     END ;

422   1     FLUSH: PROCEDURE ;            /* PURGE CHBUF OF FAULTY INPUT */
423   2     CALL BEEP ;
424   2     IF NCHAR = 0 THEN
425   2        RETURN ;
426   2     KEY = 0 ;
427   2        DO WHILE KEY<>ESC AND KEY<>RUB ;
428   3        CALL CI ;
429   3        IF JOB$SETUP THEN
430   3           JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 1 ;
431   3        END ;
432   2     CALL ERASE ;
433   2     END ;

434   1     CAPITALIZE$KEY: PROCEDURE ;
435   2     IF KEY >= 'a' AND KEY <= 'z' THEN
436   2        KEY = KEY - 20H ;
437   2     END ;

           /* "KEYSIN" ROUTINE PLACES OPERATOR KEYBOARD INPUT STRING IN CHBUF.
              63 = NUMBER OF CHARACTERS ALLOWED.  NCHAR = ACTUAL NUMBER RECEIVED.
              IF ESCAPE KEY PRESSED WITH NCHAR = 0 , A FALSE RETURN STATUS IS SET.
              OTHERWISE, ESC KEY PURGES PREVIOUS STRING AND ROUTINE STARTS OVER.
           */

438   1     KEYSIN: PROCEDURE BYTE;       /* GET OPERATOR ENTRY */
439   2     CALL SETB(0,@CHBUF,64) ;
```

```
440   2     NCHAR = 0 ;
441   2        DO WHILE NCHAR < 64 ;
442   3        CALL CI ;
443   3        IF EXIT$SETUP THEN
444   3           RETURN FALSE ;
445   3        IF KEY=ESC THEN
446   3           DO ;
447   4           IF NCHAR = 0 THEN
448   4              RETURN FALSE ;
449   4           IF JOB$SETUP THEN
450   4              JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 1 ;
451   4           CALL ERASE ;
452   4           END ;
453   3        ELSE IF CR$OR$LF THEN
454   3           RETURN TRUE ;
455   3        ELSE IF (KEY = RUB OR KEY = BS) THEN
456   3           DO ;
457   4           IF JOB$SETUP THEN
458   4              JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 1 ;
459   4           CALL RUBOUT ;
460   4           END ;
461   3        ELSE
                    DO ;
462   4           CALL CO(KEY);                    /* ECHO IT */
463   4           KEY = KEY AND 7FH ;              /* REMOVE PARITY BIT */
464   4           CMBUF(NCHAR) = KEY ;
465   4           NCHAR = NCHAR + 1 ;
466   4           END ;
467   3        END ;
468   2     CALL WRITE(@(BEL,' overflow',0));          /* STRING TOO LONG */
469   2     RETURN FALSE ;
470   2     END ;

471   1     HEX$KEY$VALUE: PROCEDURE (CHAR) BYTE;    /* CONVERT ASCII HEX KEY TO VALUE */
472   2     DECLARE CHAR BYTE ;
473   2     RETURN LOW(FINDB(@ASCII,CHAR,16)) ;
474   2     END ;

475   1     LEGAL$DECADE: PROCEDURE (CHAR) BYTE;
476   2     DECLARE CHAR BYTE ;
477   2     IF FINDB(@ASCII,CHAR,10) < 10 THEN RETURN TRUE ;
479   2     RETURN FALSE ;
480   2     END ;

           /* "GETHEX" ROUTINE ACCEPTS FROM 1 TO 8 HEXADECIMAL KEYS FROM OPERATOR.
              THE DECODED VALUE IS WRITTEN TO THE LOCATION HEX$DWORD.  THE ARGUMENT
              SPECIFIES THE MAXIMUM NUMBER OF DIGITS ALLOWED (NCHAR).  NCHAR WILL LATER
              BE USED TO DETERMINE WHETHER A BYTE, WORD, OR DWORD VALUE IS USED.
              IF AN ILLEGAL DIGIT IS ENTERED, OR MORE THAN THE ALLOWED NUMBER OF DIGITS
              ARE ENTERED, A BEEP SOUNDS AND OPERATOR MUST REPEAT ENTRY.  AN ESCAPE KEY
              RETURNS A FALSE FLAG.  */

481   1     GET$HEX: PROCEDURE (MIN,MAX) BYTE ;
482   2     DECLARE (MIN,MAX,I) BYTE ;
483   2        DO FOREVER ;
484   3        IF NOT KEYSIN THEN RETURN FALSE ;
486   3           DO I = 0 TO 63 ;
```

```
487   4             IF CHBUF(I) >= 'a' AND CHBUF(I) <= 'z' THEN  /* MUST CAPITALIZE */
488   4                 CHBUF(I) = CHBUF(I) - 20H ;
489   4             END ;
490   3         IF NCHAR >= MIN AND NCHAR <= MAX THEN
491   3             DO ;
492   4             HEX$DWORD = 0 ;
493   4             IF NCHAR > 0 THEN
494   4                 DO I = 0 TO NCHAR-1 ;
495   5                 IF HEX$KEY$VALUE(CHBUF(I)) > 15 THEN
496   5                     GO TO BAD$KEY ;
497   5                 HEX$DWORD = 16*HEX$DWORD
                                + DOUBLE(DOUBLE(HEX$KEY$VALUE(CHBUF(I)))) ;
498   5                 END ;
499   4             RETURN TRUE ;
500   4             END ;
501   3         BAD$KEY:
                  CALL FLUSH ;
502   3         IF JOB$SETUP THEN        /* NEED TO PURGE (cr) ALSO */
503   3             JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 1 ;
504   3         END ;
505   2     END ;

506   1     LEGAL$MEMORY$ADDRESS: PROCEDURE (ADDR) BYTE ;
507   2     DECLARE ADDR WORD, ABS DWORD ;
508   2     ABS = SHL(DOUBLE(SEGMENT),4) + DOUBLE(ADDR) ;
509   2     IF ABS > 1FFFH AND ABS < 0FC000H THEN RETURN TRUE ;
511   2     CALL WRITE(@(CR,LF,BEL,'Restricted memory addresses : ',0)) ;
512   2     IF ABS < 2000H THEN
513   2         CALL WRITE(@('0000H thru 1FFFH',0)) ;
514   2     ELSE
              CALL WRITE(@('0FC000H thru 0FFFFFH',0)) ;
515   2     RETURN FALSE ;
516   2     END ;

517   1     MEMADR: PROCEDURE (ADDR$PTR) BYTE ;     /* GET MEMORY ADDRESS */
518   2     DECLARE ADDR$PTR POINTER, ADDR BASED ADDR$PTR WORD ;
519   2     CALL WRITE(@(' memory addr ',0)) ;
520   2     IF NOT GETHEX(1,4) THEN
521   2         RETURN FALSE ;
522   2     IF NOT LEGAL$MEMORY$ADDRESS(HEX$WORD) THEN
523   2         RETURN FALSE ;
524   2     ADDR = HEX$WORD ;
525   2     RETURN TRUE ;
526   2     END ;

527   1     FROM$TO: PROCEDURE BYTE;              /* GET OPERATOR "FROM" & "TO" ENTRIES */
528   2     CALL WRITE(@(' from',0)) ;
529   2     IF NOT MEMADR(@FROM$ADDR) THEN
530   2         RETURN FALSE ;
531   2     CALL WRITE(@(' to',0)) ;
532   2     IF NOT MEMADR(@TO$ADDR) THEN
533   2         RETURN FALSE ;
534   2     RETURN TRUE ;
535   2     END ;

536   1     LEGAL$IOPORT: PROCEDURE (PORT) BYTE ;
537   2     DECLARE PORT WORD ;
```

```
538  2     IF PORT < 80H OR PORT > 0DFH THEN RETURN TRUE ;
540  2     CALL WRITE(@(CR,LF,BEL,'Restricted I/O ports : 080H thru 0DFH',0)) ;
541  2     RETURN FALSE ;
542  2     END ;

543  1     LEGAL$IO$FROM$TO: PROCEDURE BYTE ;
544  2     IF FROM$ADDR < 80H AND TO$ADDR > 0DFH THEN
545  2        RETURN LEGAL$IOPORT(80H) ;
546  2     RETURN TRUE ;
547  2     END ;

548  1     IO$PORT: PROCEDURE (RESULT$PTR) BYTE ;              /* GET I/O ADDRESS */
549  2     DECLARE RESULT$PTR POINTER, RESULT BASED RESULT$PTR WORD ;
550  2     CALL WRITE(@(' port ',0)) ;
551  2     IF NOT GETHEX(1,4) THEN
552  2        RETURN FALSE ;
553  2     IF NOT LEGAL$IOPORT(HEX$WORD) THEN
554  2        RETURN FALSE ;
555  2     RESULT = HEX$WORD ;
556  2     RETURN TRUE ;
557  2     END ;

558  1     LOAD$TIMER$2: PROCEDURE (COUNT) ;
559  2     DECLARE COUNT WORD ;
560  2     OUTPUT(PITMOD) = 10110110B ;        /* CTR 2, 2 BYTES, MODE 3, BINARY */
561  2     OUTPUT(PIT2) = LOW(COUNT) ;
562  2     OUTPUT(PIT2) = HIGH(COUNT) ;
563  2     END ;

564  1     USART$INIT: PROCEDURE(MODE) ;
565  2     DECLARE MODE BYTE ;
566  2     USART$INIT$BYTE = MODE ;
567  2     CALL TIME(200) ;
568  2     OUTPUT(USFLAG) = 0 ;
569  2     CALL TIME(10) ;
570  2     OUTPUT(USFLAG) = 0 ;
571  2     CALL TIME(10) ;
572  2     OUTPUT(USFLAG) = 0 ;
573  2     CALL TIME(10) ;
574  2     OUTPUT(USFLAG) = 0 ;
575  2     CALL TIME(10) ;
576  2     OUTPUT(USFLAG) = 40H ;    /* RESET */
577  2     CALL TIME(10) ;
578  2     OUTPUT(USFLAG) = MODE ;   /* MODE INSTRUCTION */
579  2     CALL TIME(10) ;
580  2     OUTPUT(USFLAG) = 35H ;    /* /RTS, ER, RXE, /DTR, TXEN */
581  2     CALL TIME(10) ;
582  2     KEY = INPUT(USDATA) ;     /* THROWAWAY PENDING KEYIN, IF ANY */
583  2     END ;

584  1     MONLOK: PROCEDURE ;
585  2     DECLARE BAUD$RATE$PTR POINTER, I BYTE ;
586  2     BEGIN:
           OUTPUT(PPIFLG) = 8AH ;          /* SET UP PARALLEL PORT FOR LED */
587  2     OUTPUT(PITMOD) = 01110110B ;    /* CTR 1, 2 BYTES, MODE 3, BINARY */
588  2     OUTPUT(PIT1) = 0FFH ;           /* SET UP TIMER 1 FOR LED FLASH */
589  2     OUTPUT(PIT1) = 0FFH ;
```

```
      2     CALL LOAD$TIMER$2(4) ;          /* 19.2 K BAUD */
      2     CALL USART$INIT(01011110B);     /* 1 STOP BIT, 8 CHAR BITS, X16 */
      2         DO WHILE NOT KEY$CAPTURED ;
      3         OUTPUT(PITMOD) = 40H ;      /* CTR 1 LATCH */
      3         I = INPUT(PIT1) ;           /* THROW AWAY LSB */
                /* FLASH LED USING MOST SIGNIFICANT BIT OF TIMER 1 */
595   3         OUTPUT(PPIC) = ( SHR(INPUT(PIT1),4) AND 08H ) OR 01H ;
596   3         END ;
597   2     OUTPUT(PPIC) = 01H ;            /* TURN ON LED */
598   2     IF (KEY AND 7FH) = 'U' THEN
599   2         DO ;
600   3         BAUD$RATE$PTR = @('19200',0) ;
601   3         GO TO EUREKA ;
602   3         END ;
603   2     IF KEY = 66H THEN
604   2         DO ;
605   3         CALL LOAD$TIMER$2(8) ;
606   3         BAUD$RATE$PTR = @('9600',0) ;
607   3         GO TO EUREKA ;
608   3         END ;
609   2     IF KEY = 78H THEN
610   2         DO ;
611   3         CALL LOAD$TIMER$2(16) ;
612   3         BAUD$RATE$PTR = @('4800',0) ;
613   3         GO TO EUREKA ;
614   3         END ;
615   2     IF KEY = 80H THEN
616   2         DO ;
617   3         CALL LOAD$TIMER$2(32) ;
618   3         BAUD$RATE$PTR = @('2400',0) ;
619   3         GO TO EUREKA ;
620   3         END ;
621   2     CALL LOAD$TIMER$2(16) ;           /* 1200 BAUD */
622   2     CALL USART$INIT(01011111B) ;      /* 1 STOP BIT, 8 CHAR BITS, X64 */
623   2     IF NOT GET$ANOTHER$KEY THEN
624   2         GO TO BEGIN ;
625   2     IF (KEY AND 7FH) = 'U' THEN
626   2         DO ;
627   3         BAUD$RATE$PTR = @('1200',0) ;
628   3         GO TO EUREKA ;
629   3         END ;
630   2     IF KEY = 66H THEN
631   2         DO ;
632   3         CALL LOAD$TIMER$2(32) ;
633   3         BAUD$RATE$PTR = @('600',0) ;
634   3         GO TO EUREKA ;
635   3         END ;
636   2     IF KEY = 78H THEN
637   2         DO ;
638   3         CALL LOAD$TIMER$2(64) ;
639   3         BAUD$RATE$PTR = @('300',0) ;
640   3         GO TO EUREKA ;
641   3         END ;
642   2     IF KEY = 80H THEN
643   2         DO ;
644   3         CALL LOAD$TIMER$2(128) ;
645   3         BAUD$RATE$PTR = @('150',0) ;
```

```
646   3         GO TO EUREKA ;
647   3         END ;
648   2       CALL LOAD$TIMER$2(175) ;          /* 110 BAUD */
649   2       CALL USART$INIT(11011011B) ;      /* 2 STOP BITS, 7 CHAR BITS, X64 */
650   2       IF NOT GET$ANOTHER$KEY THEN
651   2         GO TO BEGIN ;
652   2       IF (KEY AND 7FH) = 'U' THEN
653   2         BAUD$RATE$PTR = @('110',0) ;
654   2       ELSE GO TO BEGIN ;

655   2       EUREKA:
              CALL USART$INIT(USART$INIT$BYTE AND 0FBH);  /* FINAL SET-UP (7 BIT CHAR) */
656   2       CALL WRITE(@('U',CR,LF,LF,'Baud rate = ',0)) ;
657   2       CALL WRITE(BAUD$RATE$PTR) ;       /* PRINT BAUD RATE ON SCREEN */
658   2       CALL CR$LF ;
659   2       END ;


660   1       M$CMD: PROCEDURE ;                /* MEMORY COMMAND */
661   2       DECLARE (SEG$SAVE,FROM$SAVE,TO$SAVE,TEMP) WORD, (TYPE$SAVE,I) BYTE ;
662   2       IF JOB$RUN THEN
663   2         OUTPUT$DISABLED = TRUE ;
664   2       PTR.SEG = SEGMENT ;
665   2       REPEAT:
              IF KEY = SPACE THEN
666   2         DO ;
667   3         SEGMENT = SEG$SAVE ;
668   3         FROM$ADDR = FROM$SAVE ;
669   3         TO$ADDR = TO$SAVE ;
670   3         DATA$TYPE = TYPE$SAVE ;
671   3         GO TO MRCMDO ;
672   3         END ;
673   2       CALL WRITE(@('Memory ',0)) ;
674   2         DO CASE KEY$CASE(@(6,' RFSW',ESC)) ;
675   3           DO ;
676   4           CALL WRITE(@('Read ',0)) ;
677   4           CMD$SAVE = 'M' ;
678   4           GO TO REPEAT ;
679   4           END ;
680   3         GO TO MRCMD ;
681   3         GO TO MFCMD ;
682   3         GO TO MSCMD ;
683   3         GO TO MWCMD ;
684   3         GO TO M$ABORT ;
685   3         END ;
686   2       M$ABORT:
              CALL PURGE$JOB$ENTRY ;
687   2       RETURN ;

688   2       MR$CMD:
              CALL WRITE(@('Read ',0)) ;
689   2         DO CASE KEY$CASE(@(6,'BWDPA',ESC)) ;
690   3         CALL WRITE(@('Bytes',0)) ;
691   3         CALL WRITE(@('Words',0)) ;
692   3         CALL WRITE(@('Dwords',0)) ;
693   3         CALL WRITE(@('Pointers',0)) ;
694   3         CALL WRITE(@('ASCII',0)) ;
```

```
 695   3          GO TO M$ABORT ;
 696   3          END ;
 697   2      DATA$TYPE = KEY ;
 698   2      IF DATA$TYPE = 'A' THEN
 699   2          DO ;
 700   3          CALL WRITE(@(' starting at',0)) ;
 701   3          IF NOT MEMADR(@FROM$ADDR) THEN
 702   3              GO TO M$ABORT ;
 703   3          END ;
 704   2      ELSE IF NOT FROM$TO THEN
 705   2          GO TO M$ABORT ;
 706   2      IF JOB$IDLE THEN
 707   2          DO ;
 708   3          SEG$SAVE = SEGMENT ;
 709   3          FROM$SAVE = FROM$ADDR ;
 710   3          TO$SAVE = TO$ADDR ;
 711   3          TYPE$SAVE = DATA$TYPE ;
 712   3          CMD$SAVE = 'M' ;
 713   3          END ;

 714   2      MR$CMDO:
               OUTPUT$DISABLED = FALSE ;
 715   2      CALL NEW$MR$LINE('=') ;

 716   2      MR$CMD1:
               IF DATA$TYPE = 'A' THEN            /* ASCII string dump */
 717   2          DO ;
 718   3          CALL WRITE(MEM$PTR) ;
 719   3          RETURN ;
 720   3          END ;
 721   2      I = 0 ;
 722   2          DO FOREVER ;
 723   3          IF DATA$TYPE = 'B' THEN
 724   3              DO ;
 725   4              BYTE$SAVE = BYTE$DATA;      /* GET MEMORY BYTE */
 726   4              IF BTO THEN RETURN ;
 728   4              CALL HEX2(BYTE$SAVE) ;      /* DISPLAY IT */
 729   4              INCREMENT = 1 ;
 730   4              END ;
 731   3          ELSE IF DATA$TYPE = 'W' THEN
 732   3              DO ;
 733   4              WORD$SAVE = WORD$DATA;      /* GET MEMORY WORD */
 734   4              IF BTO THEN RETURN ;
 736   4              CALL HEX4(WORD$SAVE) ;      /* DISPLAY IT */
 737   4              INCREMENT = 2 ;
 738   4              END ;
 739   3          IF DATA$TYPE = 'D' THEN
 740   3              DO ;
 741   4              DWORD$SAVE = DWORD$DATA;    /* GET MEMORY DWORD */
 742   4              IF BTO THEN RETURN ;
 744   4              CALL HEX8(DWORD$SAVE) ;     /* DISPLAY IT */
 745   4              INCREMENT = 4 ;
 746   4              END ;
 747   3          IF DATA$TYPE = 'P' THEN
 748   3              DO ;
 749   4              POINTER$SAVE = POINTER$DATA ; /* GET MEMORY POINTER */
 750   4              IF BTO THEN RETURN ;
```

```
752  4            CALL HEX$PTR(POINTER$SAVE) ;  /* DISPLAY IT */
753  4            INCREMENT = 4 ;
754  4            END ;
755  3        IF COMPLETE OR NOT LEGAL$MEMORY$ADDRESS(FROM$ADDR) THEN
756  3            RETURN ;
757  3        I = I + INCREMENT ;
758  3        IF ( I AND 0FH ) = 0 THEN        /* END OF LINE */
759  3            CALL NEW$MR$LINE('=') ;
760  3        ELSE
                 CALL SPACER ;
761  3        END ;

762  2    MF$CMD:            /* FILL BLOCK OF MEMORY */
         OUTPUT$DISABLED = FALSE ;
763  2    IF JOB$RUN THEN
764  2        CALL WRITE(@(CR,LF,'Memory ',0)) ;
765  2    CALL WRITE(@('Fill with ',0)) ;
766  2        DO CASE KEY$CASE(@(5,'BWDA',ESC)) ;
767  3        CALL WRITE(@('Byte ',0)) ;
768  3        CALL WRITE(@('Word ',0)) ;
769  3        CALL WRITE(@('Dword ',0)) ;
770  3        CALL WRITE(@('ASCII ',0)) ;
771  3        GO TO M$ABORT ;
772  3        END ;
773  2    DATA$TYPE = KEY ;
774  2    IF DATA$TYPE = 'A' THEN GO TO MFA$CMD ;
776  2    CALL WRITE(@('value ',0)) ;
777  2    IF NOT GETHEX(1,8) THEN
778  2        GO TO M$ABORT ;
779  2    FILLER$DWORD = HEX$DWORD ;
780  2    IF NOT FROM$TO THEN
781  2        GO TO M$ABORT ;
         /* FILL LOOP */
782  2        DO FOREVER ;
783  3        IF DATA$TYPE = 'B' THEN
784  3            DO ;
785  4            BYTE$DATA = FILLER$BYTE ;
786  4            INCREMENT = 1 ;
787  4            END ;
788  3        ELSE IF DATA$TYPE = 'W' THEN
789  3            DO ;
790  4            WORD$DATA = FILLER$WORD ;
791  4            INCREMENT = 2 ;
792  4            END ;
793  3        ELSE
                 DO ;
794  4            DWORD$DATA = FILLER$DWORD ;
795  4            INCREMENT = 4 ;
796  4            END ;
797  3        IF BTO OR COMPLETE OR NOT LEGAL$MEMORY$ADDRESS(FROM$ADDR) THEN
798  3            RETURN ;
799  3        END ;

800  2    MFA$CMD:               /* MEMORY FILL WITH ASCII STRING */
         IF JOB$RUN THEN
801  2        OUTPUT$DISABLED = TRUE ;
802  2    CALL WRITE(@('string starting at',0)) ;
```

```
803  2     IF NOT MEMADR(@FROM$ADDR) THEN
804  2        GO TO M$ABORT ;
805  2     CALL WRITE(@(CR,LF,
           'String is written to memory until (esc) encountered.',CR,LF,0)) ;
806  2     OUTPUT$DISABLED = FALSE ;
807  2     CALL HEX4(SEGMENT) ;
808  2     CALL CO(':') ;
809  2     CALL HEX4(FROM$ADDR) ;
810  2     CALL WRITE(@(' = ',0)) ;
           /* SUBSTITUTION LOOP */
811  2        DO FOREVER ;
812  3        CALL CI ;
813  3        IF KEY=NULL OR KEY=ESC OR FROM$ADDR=0FFFFH THEN
814  3           DO ;
815  4           BYTE$DATA = 0 ;            /* DELIMITER */
816  4           RETURN ;
817  4           END ;
818  3        BYTE$DATA = KEY ;
819  3        FROM$ADDR = FROM$ADDR + 1 ;
820  3        CALL SHOW$KEY ;
821  3        END ;

822  2     MS$CMD:                    /* MEMORY SUBSTITUTE ROUTINE */
           CALL WRITE(@('Substitute starting at',0)) ;
823  2     IF NOT MEMADR(@FROM$ADDR) THEN
824  2        GO TO M$ABORT ;
825  2     CALL WRITE(@(CR,LF,.
           'To alter location, enter 1 or 2 hex characters followed by (CR).'
           ,CR,LF,0)) ;
826  2     CALL WRITE(@('If no change desired, enter only (CR).  ',0)) ;
827  2     CALL TO$TERMINATE ;
828  2     OUTPUT$DISABLED = FALSE ;
           /* SUBSTITUTION LOOP */
829  2        DO FOREVER ;
830  3        CALL CR$LF ;
831  3           DO I = 1 TO 4 ;
832  4           CALL HEX4(SEGMENT) ;
833  4           CALL CO(':') ;
834  4           CALL HEX4(FROM$ADDR) ;
835  4           CALL CO('=') ;
836  4           BYTE$SAVE = BYTE$DATA;              /* GET MEMORY BYTE */
837  4           IF BTO THEN
838  4              RETURN ;
839  4           CALL HEX2(BYTE$SAVE) ;             /* DISPLAY IT */
840  4           CALL CO(')') ;                  /* PROMPT FOR SUBSTITUTION */
841  4           IF NOT GETHEX(0,2) THEN
842  4              RETURN ;                  /* ESC MEANS EXIT */
843  4           IF NCHAR > 0 THEN
844  4              BYTE$DATA = HEX$BYTE ;    /* WRITE NEW BYTE TO MEMORY */
845  4           FROM$ADDR = FROM$ADDR +1 ;
846  4           IF NOT LEGAL$MEMORY$ADDRESS(FROM$ADDR) THEN
847  4              RETURN ;
848  4           CALL BLANK(3) ;
849  4           END ;
850  3        END ;

851  2     MM$CMD:               /* WRITE INTO SPECIFIED MEMORY LOCATION */
```

```
                CALL WRITE(@('Write ',0)) ;
852   2             DO CASE KEY$CASE(@(3,'BW',ESC)) ;
853   3             CALL WRITE(@('Bytes',0)) ;
854   3             CALL WRITE(@('Words',0)) ;
855   3             GO TO M$ABORT ;
856   3             END ;
857   2         DATA$TYPE = KEY ;
858   2         CALL WRITE(@(' into',0)) ;
859   2         IF NOT MEMADR(@FROM$ADDR) THEN
860   2             GO TO M$ABORT ;
861   2         CALL CR$LF ;
862   2         CALL TO$TERMINATE ;
863   2         OUTPUT$DISABLED = FALSE ;
864   2         CALL MEM$MR$LINE(')') ;
865   2             DO FOREVER ;
866   3             IF NOT GETHEX(1,4) THEN RETURN ;
868   3             IF DATA$TYPE = 'W' THEN WORD$DATA = HEX$WORD ;
870   3             ELSE BYTE$DATA = HEX$BYTE ;
871   3             IF BTO THEN RETURN ;
873   3             CALL BLANK(1) ;
874   3             END ;

875   2         END ;



876   1         I$CMD: PROCEDURE;                   /* I/O COMMAND */
877   2         DECLARE (FROM$SAVE,TO$SAVE) WORD, (TYPE$SAVE,TEMP,I) BYTE ;
878   2         IF JOB$RUN THEN
879   2             OUTPUT$DISABLED = TRUE ;
880   2         REPEAT:
                IF KEY = SPACE THEN
881   2             DO ;
882   3             FROM$ADDR = FROM$SAVE ;
883   3             TO$ADDR = TO$SAVE ;
884   3             DATA$TYPE = TYPE$SAVE ;
885   3             GO TO IRCMDO ;
886   3             END ;
887   2         CALL WRITE(@('I/O ',0))  ;
888   2             DO CASE KEY$CASE(@(5,' RWF',ESC)) ;
889   3                 DO ;
890   4                 CALL WRITE(@('Read ',0)) ;
891   4                 CMD$SAVE = 'I' ;
892   4                 GO TO REPEAT ;
893   4                 END ;
894   3             GO TO IRCMD ;
895   3             GO TO IWCMD ;
896   3             GO TO IFCMD ;
897   3             GO TO I$ABORT ;
898   3             END ;
899   2         I$ABORT:
                CALL PURGE$JOB$ENTRY ;
900   2         RETURN ;

901   2         IR$CMD:                     /* DISPLAY GROUP OF I/O PORTS */
                CALL WRITE(@('Read ',0)) ;
902   2             DO CASE KEY$CASE(@(3,'BW',ESC)) ;
```

```
 903  3          CALL WRITE(@('Bytes',0)) ;
 904  3          CALL WRITE(@('Words',0)) ;
 905  3          GO TO I$ABORT ;
 906  3          END ;
 907  2      DATA$TYPE = KEY ;
 908  2      CALL WRITE(@(' from',0)) ;
 909  2      IF NOT IOPORT(@FROM$ADDR) THEN
 910  2          GO TO I$ABORT ;
 911  2      CALL WRITE(@(' to',0)) ;
 912  2      IF NOT IOPORT(@TO$ADDR) OR NOT LEGAL$IO$FROM$TO THEN
 913  2          GO TO I$ABORT ;
 914  2      IF JOB$IDLE THEN
 915  2          DO ;
 916  3          FROM$SAVE = FROM$ADDR ;
 917  3          TO$SAVE = TO$ADDR ;
 918  3          TYPE$SAVE = DATA$TYPE ;
 919  3          CMD$SAVE = 'I' ;
 920  3          END ;

 921  2      IR$CMD0:
              OUTPUT$DISABLED = FALSE ;
 922  2      CALL NEW$IO$DISPLAY$LINE('=') ;
 923  2      I = 0 ;
 924  2          DO FOREVER ;
 925  3          IF DATA$TYPE = 'B' THEN
 926  3              DO ;
 927  4              BYTE$SAVE = INPUT(FROM$ADDR) ;    /* GET I/O PORT BYTE VALUE */
 928  4              IF BTO THEN RETURN ;
 930  4              CALL HEX2(BYTE$SAVE) ;            /* DISPLAY IT */
 931  4              INCREMENT = 1 ;
 932  4              END ;
 933  3          ELSE IF DATA$TYPE = 'W' THEN
 934  3              DO ;
 935  4              WORD$SAVE = INWORD(FROM$ADDR) ;   /* GET I/O PORT WORD VALUE */
 936  4              IF BTO THEN RETURN ;
 938  4              CALL HEX4(WORD$SAVE) ;            /* DISPLAY IT */
 939  4              INCREMENT = 2 ;
 940  4              END ;
 941  3          IF COMPLETE OR NOT LEGAL$IOPORT(FROM$ADDR) THEN
 942  3              RETURN ;
 943  3          I = I + INCREMENT ;
 944  3          IF ( I AND 0FH ) = 0 THEN             /* END OF LINE */
 945  3              CALL NEW$IO$DISPLAY$LINE('=') ;
 946  3          ELSE
                  CALL SPACER ;
 947  3          END ;

 948  2      IW$CMD:                          /* I/O PORT WRITE ROUTINE */
              CALL WRITE(@('Write ',0)) ;
 949  2          DO CASE KEY$CASE(@(3,'BW',ESC)) ;
 950  3          CALL WRITE(@('Bytes',0)) ;
 951  3          CALL WRITE(@('Words',0)) ;
 952  3          GO TO I$ABORT ;
 953  3          END ;
 954  2      DATA$TYPE = KEY ;
 955  2      CALL WRITE(@(' to',0)) ;
 956  2      IF NOT IOPORT(@FROM$ADDR) THEN
```

```
 957  2          GO TO I$ABORT ;
 958  2       CALL CR$LF ;
 959  2       CALL TO$TERMINATE ;
 960  2       OUTPUT$DISABLED = FALSE ;
 961  2       CALL NEW$IO$DISPLAY$LINE(')') ;
 962  2          DO FOREVER ;
 963  3          IF NOT GETHEX(1,4) THEN RETURN ;
 965  3          IF DATA$TYPE = 'W' THEN OUTWORD(FROM$ADDR) = HEX$WORD ;
 967  3          ELSE OUTPUT(FROM$ADDR) = HEX$BYTE ;
 968  3          IF BTO THEN RETURN ;
 970  3          CALL BLANK(1) ;
 971  3          END ;

 972  2       IF$CMD:                        /* FILL BLOCK OF PORTS */
                OUTPUT$DISABLED = FALSE ;
 973  2       IF JOB$RUN THEN
 974  2          CALL WRITE(@(CR,LF,'I/O ',0)) ;
 975  2       CALL WRITE(@('Fill with ',0)) ;
 976  2          DO CASE KEY$CASE(@(3,'BW',ESC)) ;
 977  3          CALL WRITE(@('Byte ',0)) ;
 978  3          CALL WRITE(@('Word ',0)) ;
 979  3          GO TO I$ABORT ;
 980  3          END ;
 981  2       DATA$TYPE = KEY ;
 982  2       CALL WRITE(@('value ',0)) ;
 983  2       IF NOT GETHEX(1,4) THEN
 984  2          GO TO I$ABORT ;
 985  2       FILLER$DWORD = HEX$DWORD ;
 986  2       CALL WRITE(@(' from',0)) ;
 987  2       IF NOT IOPORT(@FROM$ADDR) THEN
 988  2          GO TO I$ABORT ;
 989  2       CALL WRITE(@(' to',0)) ;
 990  2       IF NOT IOPORT(@TO$ADDR) OR NOT LEGAL$IO$FROM$TO THEN
 991  2          GO TO I$ABORT ;
                /* FILL LOOP */
 992  2          DO FOREVER ;
 993  3          IF DATA$TYPE = 'B' THEN
 994  3             DO ;
 995  4             OUTPUT(FROM$ADDR) = FILLER$BYTE ;
 996  4             INCREMENT = 1 ;
 997  4             END ;
 998  3          ELSE
                   DO ;
 999  4             OUTWORD(FROM$ADDR) = FILLER$WORD ;
1000  4             INCREMENT = 2 ;
1001  4             END ;
1002  3          IF BTO OR COMPLETE THEN
1003  3             RETURN ;
1004  3          END ;

1005  2       END ;


1006  1       H$CMD: PROCEDURE;              /* HELP DISPLAY */
1007  2       IF JOB$SETUP THEN              /* REMOVE "H" CMD FROM JOB BUFFER */
1008  2          JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 2 ;
1009  2       CALL WRITE(@(
```

```
                'Help:',CR,LF,
                '(u)   = Job load & run.     u = 0 thru 9, selects file = 10*d + u.',CR,LF,
                'J(d)  = Job decade.         d = 0 thru 9, selects decade.',CR,LF,
                'JE    = Job edit.           Views/modifies/loads/stores job buffer.',CR,LF,
                'JD    = Job directory.      Shows titles for selected job decade.',CR,LF,
                'I(sp) = I/O read rerun.     Repeat current "IR" setup.',CR,LF,
                'M(sp) = Memory read rerun.  Repeat current "MR" setup.',CR,LF,
                'J(sp) = Job rerun.          Repeat current job buffer.',CR,LF,
                '(sp)  = Space bar.          Repeats last rerun ( I(sp), M(sp), or J(sp) )',CR,LF,
                '#     = B or W              B(yte) or W(ord).',CR,LF,
                'IR#   = I/O read.           Sets up byte or word I/O port group display.',CR,LF,
                'IW#   = I/O write.          Repetitive byte or word outputs to selected port.',CR,LF,
                'IF#   = I/O fill.           Writes byte or word to a block of ports.',CR,LF,
                'MW#   = Memory write.       Repetitive byte or word memory writes.',CR,LF,
                '*     = B, W, D, P, or A    B(yte), W(ord), D(word), P(ointer), or A(SCII).',CR,LF,
                'MR*   = Memory read.        Sets up byte, word, dword, pointer, or ASCII dump.',CR,LF,
                '@     = B, W, D, or A       B(yte), W(ord), D(word), or A(SCII).',CR,LF,
                'MF@   = Memory fill.        Byte, word, dword, or ASCII string fill.',CR,LF,
                'MS    = Memory substitute.  Byte scan/modify.',CR,LF,
                'R     = Rupts.              Displays multibus interrupt counters.',CR,LF,
                'S     = Segment.            Reloads memory base register.',
                0)) ;
1010  2       END ;


1011  1       J$CMD: PROCEDURE;             /* JOB SETUP/RUN COMMAND */
1012  2       DECLARE I BYTE, INDEX WORD ;
1013  2       IF EXIT$SETUP THEN
1014  2           DO ;
1015  3           CALL PURGE$JOB$ENTRY ;
1016  3           CALL WRITE(@('Job ',0)) ;
1017  3           GO TO JE$CMD ;
1018  3           END ;
1019  2       IF JOB$SETUP THEN             /* REMOVE "J" COMMAND FROM JOB BUFFER */
1020  2           DO ;
1021  3           JOB$BUFFER$INDEX = JOB$BUFFER$INDEX - 2 ;
1022  3           CALL ILLEGAL$CMD ;
1023  3           RETURN ;
1024  3           END ;
1025  2       IF KEY = SPACE THEN
1026  2           CALL WRITE(@('Job ',0)) ;
1027  2       REPEAT:
                IF KEY = SPACE THEN
1028  2           DO ;
1029  3           CALL WRITE(@('file ',0)) ;
1030  3           IF EDIT$FILE$NUMBER < 100 THEN
1031  3               DO ;
1032  4               CALL CO(ASCII(EDIT$FILE$NUMBER/10)) ;
1033  4               CALL CO(ASCII(EDIT$FILE$NUMBER MOD 10)) ;
1034  4               END ;
1035  3           ELSE
                    CALL WRITE(@('??',0)) ;
1036  3           CALL WRITE(@(' = ',0)) ;
1037  3           CALL WRITE(@EDIT$BUFFER) ;
1038  3           CMD$SAVE = 'J' ;
1039  3           JOB$BUFFER$PTR = @EDIT$BUFFER ;
1040  3           JOB$BUFFER$INDEX = 64 ;
```

```
1041  3          JOB$RUN ;
1042  3          RETURN ;
1043  3          END ;
1044  2       CALL WRITE(@('Job ',0))  ;
1045  2       IF KEY = 'E' THEN
1046  2          GO TO JE$CMD ;
1047  2       CALL CI ;                  /* EXPECT (sp), D, E, (ESC), OR 0-9 */
1048  2       CALL CAPITALIZE$KEY ;
1049  2       IF KEY = SPACE THEN
1050  2          GO TO REPEAT ;
1051  2       IF KEY=ESC THEN
1052  2          RETURN ;
1053  2       JE$CMD:
                IF EXIT$SETUP OR (KEY = 'E') THEN        /* JOB BUFFER EDITOR */
1054  2          DO ;
1055  3          JOB$IDLE ;
1056  3          CALL WRITE(@('Editor',0)) ;
1057  3          SHOW$ITEMS:
                   CALL WRITE(@(CR,LF,'File  = ',0)) ;
1058  3          IF EDIT$FILE$NUMBER < 100 THEN
1059  3             DO ;
1060  4             CALL CO(ASCII(EDIT$FILE$NUMBER/10)) ;
1061  4             CALL CO(ASCII(EDIT$FILE$NUMBER MOD 10))
1062  4             END ;
1063  3          ELSE
                      CALL WRITE(@('??',0)) ;
1064  3          CALL WRITE(@(CR,LF,'Title = ',0)) ;
1065  3          CALL WRITE(@EDIT$BUFFER) ;
1066  3          CALL GENERATE$ITEM$INDEXES ;
1067  3          IF NITEMS = 0 THEN
1068  3             CALL WRITE(@(CR,LF,'   01 =',0)) ;
1069  3          ELSE
                      DO ;
1070  4                DO I = 1 TO NITEMS ;
1071  5                CALL WRITE(@(CR,LF,'   ',0)) ;
1072  5                CALL HEX2(I) ;
1073  5                CALL WRITE(@(' = ',0)) ;
1074  5                INDEX = ITEM$INDEX(I-1) + 1 ;
1075  5                KEY = EDIT$BUFFER(INDEX) ;
1076  5                   DO WHILE KEY <> 0FFH ;
1077  6                   CALL SHOW$KEY ;
1078  6                   CALL BLANK(1) ;
1079  6                   INDEX = INDEX + 1 ;
1080  6                   KEY = EDIT$BUFFER(INDEX) ;
1081  6                   END ;
1082  5                END ;
1083  4             END ;
1084  3          SHOW$MENU:
                   CALL WRITE(@(CR,LF,
                   'Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit ',0)) ;
1085  3             DO FOREVER ;
1086  4             CALL CI ;
1087  4             CALL CAPITALIZE$KEY ;
1088  4             IF KEY=ESC THEN
1089  4                DO ;
1090  5                JOB$BUFFER$INDEX = 64 ;
1091  5                RETURN ;
```

```
1092  5              END ;
1093  4           IF KEY = 'A' THEN
1094  4              DO ;
1095  5              CALL WRITE(@(CR,LF,'Appending to buffer - use (cntl C) to terminate.',0)) ;
1096  5              JOB$BUFFER$PTR = @EDIT$BUFFER ;
1097  5              JOB$BUFFER$INDEX = ITEM$INDEX(NITEMS) ;
1098  5              JOB$SETUP ;
1099  5              EDIT$FILE$NUMBER = 255 ;
1100  5              RETURN ;
1101  5              END ;
1102  4           ELSE IF KEY = 'D' THEN
1103  4              DO ;
1104  5              CALL WRITE(@(CR,LF,'Delete item number (in hex) ) ',0)) ;
1105  5              IF NOT GETHEX(1,2) THEN
1106  5                 GO TO SHOW$MENU ;
1107  5              IF HEX$BYTE = 0 OR HEX$BYTE ) NITEMS THEN
1108  5                 DO ;
1109  6                 CALL WRITE(@(BEL,' out of range',0)) ;
1110  6                 GO TO SHOW$MENU ;
1111  6                 END ;
1112  5              CALL MOVB(@EDIT$BUFFER(ITEM$INDEX(HEX$BYTE)) ,
                            @EDIT$BUFFER(ITEM$INDEX(HEX$BYTE-1)) ,
                            1280-ITEM$INDEX(HEX$BYTE) ) ;
1113  5              EDIT$FILE$NUMBER = 255 ;
1114  5              GO TO SHOW$ITEMS ;
1115  5              END ;
1116  4           ELSE IF KEY = 'E' THEN
1117  4              DO ;
1118  5              CALL WRITE(@(CR,LF,'Erase entire buffer ? ) ',0)) ;
1119  5              CALL YES$NO ;
1120  5              IF KEY = 'Y' THEN
1121  5                 DO ;
1122  6                 CALL ERASE$EDIT$BUFFER ;
1123  6                 GO TO SHOW$ITEMS ;
1124  6                 END ;
1125  5              EDIT$FILE$NUMBER = 255 ;
1126  5              GO TO SHOW$MENU ;
1127  5              END ;
1128  4           ELSE IF KEY = 'L' THEN
1129  4              DO ;
1130  5              CALL WRITE(@(CR,LF,'Load job number (0-99 decimal) ) ',0)) ;
1131  5              IF GETHEX(1,2) THEN
1132  5                 DO ;
1133  6                 BYTE$SAVE = 10 * SHR(HEX$BYTE,4) ;    /* CONVERT TO DECIMAL */
1134  6                 HEX$BYTE = BYTE$SAVE + (HEX$BYTE AND 0FH) ;
1135  6                 IF HEX$BYTE < 100 THEN
1136  6                    DO ;
1137  7                    EDIT$FILE$NUMBER = HEX$BYTE ;
1138  7                    IF NOT READ$251(20,20*EDIT$FILE$NUMBER,@EDIT$BUFFER) THEN
1139  7                       CALL ERASE$EDIT$BUFFER ;
1140  7                    END ;
1141  6                 ELSE
                             CALL WRITE(@(BEL,' out of range',0)) ;
1142  6                 END ;
1143  5              GO TO SHOW$ITEMS ;
1144  5              END ;
1145  4           ELSE IF KEY = 'S' THEN
```

```
1146  4              DO ;
1147  5              CALL WRITE(@(CR,LF,'Save as job number (0-99 decimal) ) ',0)) ;
1148  5              IF GETHEX(1,2) THEN
1149  5                  DO ;
1150  6                  BYTE$SAVE = 10 * SHR(HEX$BYTE,4) ;  /* CONVERT TO DECIMAL */
1151  6                  HEX$BYTE = BYTE$SAVE + (HEX$BYTE AND 0FH) ;
1152  6                  IF HEX$BYTE < 100 THEN
1153  6                      DO ;
1154  7                      EDIT$FILE$NUMBER = HEX$BYTE ;
1155  7                      CALL WRITE$251(EDIT$FILE$NUMBER) ;
1156  7                      END ;
1157  6                  ELSE
                               CALL WRITE(@(BEL,' out of range',0)) ;
1158  6                  END ;
1159  5              GO TO SHOW$MENU ;
1160  5              END ;
1161  4          ELSE IF KEY = 'T' THEN
1162  4              DO ;
1163  5              CALL WRITE(@(CR,LF,'Enter title (63 char max)',CR,LF,') ',0)) ;
1164  5              IF KEYSIN THEN
1165  5                  DO ;
1166  6                  CALL MOVB(@CHBUF,@EDIT$BUFFER,64) ;
1167  6                  EDIT$FILE$NUMBER = 255 ;
1168  6                  END ;
1169  5              GO TO SHOW$ITEMS ;
1170  5              END ;
1171  4          ELSE CALL BEEP ;
1172  4          END ;
1173  3      END ;
1174  2  ELSE IF KEY >= '0' AND KEY <= '9' THEN        /* JOB DECADE SELECT */
1175  2      DO ;
1176  3      CALL WRITE(@('decade ',0)) ;
1177  3      CALL CO(KEY) ;
1178  3      CALL WRITE(@('u selected',0)) ;
1179  3      FILE$DECADE = HEX$KEY$VALUE(KEY) ;
1180  3      CALL WRITE(@(CR,LF,'Job ',0))    ;
1181  3      KEY = 'D' ;
1182  3      GO TO JE$CMD ;          /* SHOW DIRECTORY NEXT */
1183  3      END ;
1184  2  ELSE IF KEY = 'D' THEN      /* JOB DIRECTORY */
1185  2      DO ;
1186  3      CALL WRITE(@('Directory for decade ',0)) ;
1187  3      CALL CO(ASCII(FILE$DECADE)) ;
1188  3          DO I = 0 TO 9 ;
1189  4          CALL WRITE(@(CR,LF,'  ',0)) ;
1190  4          CALL CO(ASCII(FILE$DECADE)) ;
1191  4          CALL CO(ASCII(I)) ;
1192  4          CALL WRITE(@(' : ',0)) ;
1193  4          IF READ$251(1,200*FILE$DECADE+20*I,@BLOCK) THEN
1194  4              CALL WRITE(@BLOCK) ;
1195  4          ELSE
                       RETURN ;
1196  4          END ;
1197  3      END ;
1198  2  ELSE CALL ILLEGAL$CMD ;
1199  2  END ;
```

```
1200  1     R$CMD: PROCEDURE ;          /* DISPLAY MULTIBUS INTERRUPT COUNTERS */
1201  2     DECLARE COUNTER$SNAP DWORD, I BYTE ;
1202  2     IF NOT JOB$RUN THEN
1203  2        CALL WRITE(@('Rupt activity on Multibus',0)) ;
1204  2     CALL CR$LF ;
1205  2        DO I = 0 TO 7 ;
1206  3        CALL WRITE(@('INT',0)) ;
1207  3        CALL HEX1(I) ;
1208  3        CALL CO(':') ;
1209  3        DISABLE ;
1210  3        COUNTER$SNAP = INT$COUNTER(I);
1211  3        ENABLE ;
1212  3        CALL HEX4(HIGH(COUNTER$SNAP)) ;
1213  3        CALL HEX4(LOW(COUNTER$SNAP)) ;
1214  3        IF I = 3 THEN
1215  3           CALL CR$LF ;
1216  3        ELSE
                    CALL BLANK(5) ;
1217  3        END ;

1218  2     END ;

1219  1     BACKUP: PROCEDURE ;
1220  2     DECLARE (I,J,INDEX) WORD ;
1221  2     CALL WRITE(@('Backup current files ? ) ',0)) ;
1222  2     CALL YES$NO ;
1223  2     IF KEY <> 'Y' THEN RETURN ;
1225  2     CALL WRITE(@('  All decades ? ) ',0)) ;
1226  2     CALL YES$NO ;
1227  2     IF KEY = 'Y' THEN
1228  2        DO ;
1229  3        CALL MOVB(@ASCII,@CMBUF,10) ;
1230  3        NCHAR = 10 ;
1231  3        END ;
1232  2     ELSE IF KEY = 'N' THEN
1233  2        DO ;
1234  3        CALL WRITE(@('  Which decades ? ) ',0)) ;
1235  3        IF NOT KEYSIN OR NCHAR = 0 THEN RETURN ;
1237  3        END ;
1238  2     ELSE RETURN ;
1239  2     CALL SETW(0,BACKUP$RESTORE$BUFFER,32768) ;
1240  2     INDEX = 0 ;
1241  2        DO I = 0 TO NCHAR-1 ;
1242  3        KEY = CMBUF(I) ;
1243  3        IF LEGAL$DECADE(KEY) THEN
1244  3           DO J = 0 TO 9 ;
1245  4           FILE$NUMBER = 10*HEX$KEY$VALUE(KEY) + J ;
1246  4           IF NOT READ$251(20,20*DOUBLE(FILE$NUMBER),@FILE$BUFFER) THEN RETURN ;
1248  4           IF FILE$BUFFER(65) <> 0 THEN
1249  4              DO ;
1250  5              CALL WRITE(@(CR,LF,'  Job file ',0)) ;
1251  5              CALL CO(ASCII(FILE$NUMBER / 10)) ;
1252  5              CALL CO(ASCII(FILE$NUMBER MOD 10)) ;
1253  5              CALL WRITE(@(' = ',0)) ;
1254  5              CALL WRITE(@FILE$BUFFER) ;
1255  5              DIR(FILE$NUMBER).INDEX = INDEX ;
```

```
1256  5              DIR(FILE$NUMBER).LENGTH = 65+FINDB(@FILE$BUFFER(64),0,1280-64) ;
1257  5              CALL MOVB(@FILE$BUFFER,@INFO(INDEX),DIR(FILE$NUMBER).LENGTH) ;
1258  5              INDEX = INDEX + DIR(FILE$NUMBER).LENGTH ;
1259  5              END ;
1260  4           END ;
1261  3        END ;
               /* FINAL STEP : FILL HEADER BLOCK WITH ID */
1262  2        DO I = 0 TO 31 ;
1263  3        HDR(I) = 'X251' ;
1264  3        END ;
1265  2     END ;

1266  1     RESTORE: PROCEDURE ;
1267  2     DECLARE I BYTE ;
1268  2     CALL WRITE(@('Restore bubble files ? ) ',0)) ;
1269  2     CALL YES$NO ;
1270  2     IF KEY <> 'Y' THEN RETURN ;
1272  2        DO I = 0 TO 31 ;
1273  3        IF HDR(I) <> 'X251' THEN
1274  3           DO ;
1275  4           CALL WRITE(@(CR,LF,BEL,
                  'External backup/restore buffer does not contain restore file.',0)) ;
1276  4           RETURN ;
1277  4           END ;
1278  3        END ;
1279  2     CALL WRITE(@(CR,LF,
               'Active files will be overwritten.  Are you sure ? ) ',0)) ;
1280  2     CALL YES$NO ;
1281  2     IF KEY = 'Y' THEN
1282  2        DO I = 0 TO 99 ;
1283  3        CALL SETW(0,@EDIT$BUFFER,640) ;
1284  3        FILE$BUFFER(64) = 0FFH ;
1285  3        IF DIR(I).LENGTH > 0 THEN
1286  3           DO ;
1287  4           CALL WRITE(@(CR,LF,'Job file ',0)) ;
1288  4           CALL CO(ASCII(I / 10)) ;
1289  4           CALL CO(ASCII(I MOD 10)) ;
1290  4           CALL MOVB(@INFO(DIR(I).INDEX),@EDIT$BUFFER,DIR(I).LENGTH) ;
1291  4           CALL WRITE(@(' = ',0)) ;
1292  4           CALL WRITE(@EDIT$BUFFER) ;
1293  4           EDIT$FILE$NUMBER = I ;
1294  4           CALL WRITE$251(I) ;
1295  4           END ;
1296  3        END ;
1297  2     END ;

1298  1     SHOW$MENU: PROCEDURE ;
1299  2     CALL WRITE(@(CR,LF,
               'NASA/ADFRF XAIDS Maintenance Processor   14 January 1987   R. Glover',CR,LF,
               'Menu: Help, 0-9, J0-J9, JE, JB, I(sp), M(sp), J(sp), (sp), IRB, IRW, IFB, IFW,', CR, LF,
               '      IWB, IWW, MWB, MWW, MRB, MRW, MRD, MRP, MRA, MFB, MFW, MFD, MS, R, S',0)) ;
1300  2     END ;


1301  1     RESET:                    /* MAIN PROGRAM */

            DISABLE ;                 /* DISABLE INTERRUPTS */
```

```
1302  1     CALL SETW(0,0,4094) ;        /* CLEAR RAM (86/05 RAM 8K BYTES LONG) */
1303  1     CALL INITIALIZE$INTERRUPTS ;
1304  1     CALL INITIALIZE$PRINTER ;
1305  1     CALL MONLOK ;                /* PERFORM BAUD RATE LOCK-ON */
1306  1     FILE$NUMBER = 255 ;
1307  1     CALL ERASE$EDIT$BUFFER ;
1308  1     CMD$SAVE  = 'M' ;            /* SET UP REPEAT FOR MRB */
1309  1     DATA$TYPE = 'B' ;

1310  1     RESTART:
            CALL INITIALIZE$251 ;        /* INITIALIZE SBX251 BUBBLE MEMORY MULTIMODULE */

1311  1     CALL WRITE(@(CR,LF,
            LF,
            '         XX     XX         AA         IIIIII   DDDDDDD     SSSSSSS  ',CR,LF,
            '          XX   XX         AAAA          II     DD    DD   SS     SS ',CR,LF,
            '           XX XX         AA  AA         II     DD     DD  SS        ',CR,LF,
            '            XXX         AA    AA        II     DD     DD   SSSSSSS  ',CR,LF,
            '           XX XX       AAAAAAAAAA       II     DD     DD         SS ',CR,LF,
            '          XX   XX     AA        AA      II     DD    DD   SS     SS ',CR,LF,
            '         XX     XX   AA          AA   IIIIII   DDDDDDD     SSSSSSS  ',CR,LF,
            '                                                                    ',CR,LF,
            '        MMM     MMM         AA         IIIIII   NN      NN   TTTTTTTT',CR,LF,
            '        MM MM MM MM        AAAA          II     NNN     NN      TT   ',CR,LF,
            '        MM  MMM  MM       AA  AA         II     NN NN   NN      TT   ',CR,LF,
            '        MM   M   MM      AA    AA        II     NN  NN  NN      TT   ',CR,LF,
            '        MM       MM     AAAAAAAAAA       II     NN   NN NN      TT   ',CR,LF,
            '        MM       MM    AA        AA      II     NN    NNN      TT   ',CR,LF,
            '        MM       MM   AA          AA   IIIIII   NN     NN      TT   ',CR,LF,
            0)) ;

1312  1     REPEAT:                      /* DISPLAY MENU AGAIN */
            CALL SHOW$MENU ;

1313  1     NEXT:                        /* PROCESS NEXT COMMAND */
            OUTPUT$DISABLED = FALSE ;
1314  1     INCREMENT = 1 ;
1315  1     IF JOB$RUN THEN
1316  1        DO ;
1317  2        IF JOB$BUFFER(JOB$BUFFER$INDEX) = 0FFH THEN
1318  2           JOB$BUFFER$INDEX = JOB$BUFFER$INDEX + 1 ;
1319  2        IF JOB$BUFFER(JOB$BUFFER$INDEX) = 0 THEN
1320  2           JOB$IDLE ;
1321  2        END ;
1322  1     IF NOT JOB$RUN THEN
1323  1        CALL CR$LF ;
1324  1     IF JOB$IDLE THEN
1325  1        CALL CO('.');             /* NORMAL PROMPT CHAR */
1326  1     ELSE IF JOB$SETUP THEN
1327  1        DO ;
1328  2        CALL CO(')') ;
1329  2        JOB$BUFFER(JOB$BUFFER$INDEX) = 0FFH ;
1330  2        JOB$BUFFER$INDEX = JOB$BUFFER$INDEX + 1 ;
1331  2        END ;

1332  1     TRY$AGAIN:
```

```
                DO CASE KEY$CASE(@(22,'0123456789',ESC,SPACE,'MHIJRS',CNTL$C,CNTL$B,CNTL$R,CR));
1333   2        CASE0:
                    CALL LOAD$JOB ;
1334   2        CASE1:
                    CALL LOAD$JOB ;
1335   2        CASE2:
                    CALL LOAD$JOB ;
1336   2        CASE3:
                    CALL LOAD$JOB ;
1337   2        CASE4:
                    CALL LOAD$JOB ;
1338   2        CASE5:
                    CALL LOAD$JOB ;
1339   2        CASE6:
                    CALL LOAD$JOB ;
1340   2        CASE7:
                    CALL LOAD$JOB ;
1341   2        CASE8:
                    CALL LOAD$JOB ;
1342   2        CASE9:
                    CALL LOAD$JOB ;
1343   2        CASE10:                          /* ESC */
                    DO ;
1344   3           CALL PURGE$JOB$ENTRY ;
1345   3           CALL CR$LF ;
1346   3           CALL CR$LF ;
1347   3           GO TO RESTART ;
1348   3           END ;
1349   2        CASE11:                          /* BLANK */
                    DO ;
1350   3           IF JOB$SETUP THEN
1351   3              DO ;
1352   4              CALL PURGE$JOB$ENTRY ;
1353   4              CALL WRITE(@(BEL,' (space) - illegal reread in job mode',0)) ;
1354   4              END ;
1355   3           ELSE IF CMD$SAVE='M' THEN
1356   3              CALL M$CMD ;
1357   3           ELSE IF CMD$SAVE='I' THEN
1358   3              CALL I$CMD ;
1359   3           ELSE IF CMD$SAVE='J' THEN
1360   3              CALL J$CMD ;
1361   3           END ;
1362   2        CASE12:                          /* MEMORY COMMAND */
                    CALL M$CMD ;
1363   2        CASE13:                          /* HELP COMMAND */
                    CALL H$CMD ;
1364   2        CASE14:                          /* I/O COMMAND */
                    CALL I$CMD ;
1365   2        CASE15:                          /* JOB COMMAND */
                    CALL J$CMD ;
1366   2        CASE16:                          /* RUPT COUNTER DISPLAY */
                    CALL R$CMD ;
1367   2        CASE17:                          /* CHANGE MEMORY SEGMENT */
                    DO ;
1368   3           IF JOB$RUN THEN
1369   3              OUTPUT$DISABLED = TRUE ;
1370   3           CALL WRITE(@('Current memory segment register : ',0)) ;
```

```
1371  3            CALL HEX4(SEGMENT) ;
1372  3            CALL WRITE(@(CR,LF,'Enter desired segment value (0 - FFFF) ) ',0)) ;
1373  3            IF GETHEX(1,4) THEN
1374  3               DO ;
1375  4               SEGMENT = HEX$WORD ;
1376  4               CALL WRITE(@(CR,LF,'Updated memory segment register : ',0)) ;
1377  4               CALL HEX4(SEGMENT) ;
1378  4               END ;
1379  3            END ;
1380  2         CASE18:                          /* CNTL C */
                     DO ;
1381  3            IF JOB$SETUP THEN
1382  3               CALL J$CMD ;
1383  3            ELSE
                        GO TO ILLEGAL ;
1384  3            END ;
1385  2         CASE19:                          /* CNTL B */
                     DO ;
1386  3            IF JOB$IDLE THEN             /* OK TO DO BACKUP */
1387  3               CALL BACKUP ;
1388  3            ELSE
                        GO TO ILLEGAL ;
1389  3            END ;
1390  2         CASE20:                          /* CNTL R */
                     DO ;
1391  3            IF JOB$IDLE THEN             /* OK TO DO RESTORE */
1392  3               CALL RESTORE ;
1393  3            ELSE
                        GO TO ILLEGAL ;
1394  3            END ;
1395  2         CASE21:                          /* CR */
                     GO TO REPEAT ;
1396  2         END ;

1397  1      GO TO NEXT ;

1398  1      ILLEGAL:
               CALL BEEP ;
1399  1      CALL PURGE$JOB$ENTRY ;
1400  1      GO TO TRY$AGAIN ;

1401  1      END ;



MODULE INFORMATION:

     CODE AREA SIZE     = 1C7BH   7291D
     CONSTANT AREA SIZE = 11F8H   4600D
     VARIABLE AREA SIZE = 0BE0H   3040D
     MAXIMUM STACK SIZE = 003AH     58D
     1633 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:
```

```
    195KB MEMORY AVAILABLE
    26KB MEMORY USED   (13%)
    0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

APPENDIX C

File LP.P86

The following listing shows the PLM86 source language for the module containing the line printer initialization and servicing routines. This module declares PUBLIC the following procedures:

INITIALIZE$PRINTER	a PROCEDURE that initializes the 8255 parallel peripheral I/O chip, commands the printer on line, and erases the screen image buffer.
PRINTER$BUFFER	a PROCEDURE with a BYTE argument containing an ASCII character which is to be copied into the screen image buffer.
PRINTER$SNAPSHOT	a PROCEDURE that synchronously copies the screen image buffer into the printer buffer by lines properly ordered for printer dump.
PRINTER$SERVICE	a PROCEDURE called frequently that serves as a background task to supervise the printer state.

```
iRMX 86 PL/M-86 V2.7 COMPILATION OF MODULE LP_P86
OBJECT MODULE PLACED IN LP.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 LP.P86


          $COMPACT ROM OPTIMIZE(3)
          $TITLE('NASA/ADFRF XAIDS MAINT PRINTER ROUTINES  14 JAN 1987')

          /*   NASA AMES DRYDEN FLIGHT RESEARCH FACILITY   R GLOVER   */

   1      LP_P86: DO ;

          /* EXTERNAL ROUTINES */

   2  2   WRITE: PROCEDURE (PTR) EXTERNAL; DECLARE PTR POINTER; END;
   5  2   HEX1:  PROCEDURE (VAL) EXTERNAL; DECLARE VAL BYTE; END;
   8  2   HEX2:  PROCEDURE (VAL) EXTERNAL; DECLARE VAL BYTE; END;

          /* SBC 86/05 BOARD PROGRAMMABLE PERIPHERAL INTERFACE I/O PORT MAPPING */

  11  1   DECLARE PPIA      LITERALLY '0C8H' ;   /* PA0-PA7 8287 (INVERTING)     */
  12  1   DECLARE PPIB      LITERALLY '0CAH' ;   /* PB0-PB7 902  (TERMINATOR)    */
  13  1   DECLARE PPIC      LITERALLY '0CCH' ;   /* PC0-PC3 7408 (NON-INVERTING) */
                                                 /* PC4-PC7 902  (TERMINATOR)    */
  14  1   DECLARE PPIFLG    LITERALLY '0CEH' ;

          /* I/O STATUS FLAGS */

  15  1   DECLARE PTR$BUSY   LITERALLY 'SHR(INPUT(PPIC),7)' ;
  16  1   DECLARE PTR$SELECT LITERALLY 'SHR(INPUT(PPIC),4)' ;

          /* MISCELLANEOUS DECLARATIONS */

  17  1   DECLARE TRUE      LITERALLY '0FFH' ;
  18  1   DECLARE FALSE     LITERALLY '0' ;
  19  1   DECLARE FOREVER   LITERALLY 'WHILE TRUE' ;
  20  1   DECLARE KEY BYTE EXTERNAL ;
  21  1   DECLARE CRT$LINE (24) STRUCTURE (CHAR (80) BYTE) ;
  22  1   DECLARE PTR$LINE (24) STRUCTURE (CHAR (80) BYTE) ;
  23  1   DECLARE (ROW,COL,FF$FLG) BYTE ;
  24  1   DECLARE PTRFLG BYTE ;
             /*  PTRFLG POSSIBLE STATES :              */
             /*     0   RESET, DESELECTED, OR OFF      */
             /*     1   IDLE AND SELECTED (ON LINE)    */
             /*     2   PERFORMING PRINTER FORM FEED   */
             /*     3   PERFORMING PRINTER LINE FEEDS  */
             /*     4   PERFORMING PRINTER INDENT      */
             /*     5   PERFORMING PRINTER LINE DUMP   */
             /*     6   PERFORMING PRINTER CR & LF     */

  25  1   STROBE: PROCEDURE (CHAR) ;
  26  2   DECLARE CHAR BYTE ;
  27  2   OUTPUT(PPIA) = NOT CHAR ;   /* INVERTING DRIVERS ON PORT A */
  28  2   DISABLE ;
  29  2   OUTPUT(PPIFLG) = 00H ;      /* RESET PORT C BIT 0  (.NOT. STROBE)  */
  30  2   OUTPUT(PPIFLG) = 01H ;      /* SET PORT C BIT 0  (REMOVE STROBE)  */
```

```
 31  2     ENABLE ;
 32  2     END ;

 33  1     INITIALIZE$PRINTER: PROCEDURE PUBLIC ;
 34  2     OUTPUT(PPIFLG) = 8AH ;      /* A=OUT(0)  B=IN(0)  C(7-4)=IN  C(3-0)=OUT  */
 35  2     CALL STROBE(11H) ;          /*  "ON LINE" CODE */
 36  2     CALL SETB(' ',@CRT$LINE,1920) ;
 37  2     ROW = 1 ;
 38  2     COL = 1 ;
 39  2     PTRFLG = 0 ;
 40  2     END ;

 41  1     PRINTER$BUFFER: PROCEDURE (CHAR) PUBLIC ;
 42  2     DECLARE CHAR BYTE ;
 43  2     CHAR = CHAR AND 7FH ;
 44  2     IF CHAR >= ' ' AND CHAR < 7FH THEN
 45  2         DO ;
 46  3         CRT$LINE(ROW-1).CHAR(COL-1) = CHAR ;
 47  3         COL = COL + 1 ;
 48  3         IF COL > 80 THEN                    /* NEED AUTO LF & CR */
 49  3             DO ;
 50  4             COL = 1 ;
 51  4             ROW = ROW + 1 ;
 52  4             IF ROW > 24 THEN ROW = 1 ;
 54  4             CALL SETB(' ',@CRT$LINE(ROW-1),80) ;
 55  4             END ;
 56  3         END ;
 57  2     ELSE IF CHAR = 08H THEN     /* BS */
 58  2         DO ;
 59  3         IF COL > 1 THEN COL = COL - 1 ;
 61  3         END ;
 62  2     ELSE IF CHAR = 0AH THEN     /* LF */
 63  2         DO ;
 64  3         ROW = ROW + 1 ;
 65  3         IF ROW > 24 THEN ROW = 1 ;
 67  3         CALL SETB(' ',@CRT$LINE(ROW-1),80) ;
 68  3         END ;
 69  2     ELSE IF CHAR = 0BH THEN     /* VT */
 70  2         DO ;
 71  3         IF ROW > 1 THEN ROW = ROW - 1 ;
 73  3         END ;
 74  2     ELSE IF CHAR = 0DH THEN     /* CR */
 75  2         DO ;
 76  3         COL = 1 ;
 77  3         END ;
 78  2     END ;

 79  1     PRINTER$SNAPSHOT: PROCEDURE PUBLIC ;
 80  2     DECLARE (I,J) BYTE ;
 81  2     IF KEY = 10H THEN                   /* CNTL P */
 82  2         DO ;
 83  3         IF PTRFLG = 1 THEN              /* READY FOR DUMP */
 84  3             DO ;
 85  4             J = ROW + 1 ;               /* J IS FIRST LINE OF SCREEN IMAGE */
 86  4                 DO I = 0 TO 23 ;
 87  5                 IF J = 25 THEN J = 1 ;
 89  5                 CALL MOVB(@CRT$LINE(J-1),@PTR$LINE(I),80) ;
```

```
 90  5              J = J + 1 ;
 91  5              END ;
 92  4          PTRFLG = 2 ;
 93  4          END ;
 94  3      KEY = 0 ;
 95  3      END ;
 96  2  END ;

 97  1  PRINTER$SERVICE: PROCEDURE PUBLIC ;
 98  2  DECLARE (INDEX,LINE) BYTE ;
 99  2  IF NOT PTR$SELECT THEN PTRFLG = 0 ;
101  2  IF NOT PTR$BUSY THEN
102  2      DO CASE PTRFLG MOD 7 ;
103  3      RESET:
                 DO ;
104  4          FF$FLG = TRUE ;
105  4          IF PTR$SELECT THEN PTRFLG = 1
107  4          END ;
108  3      IDLE:
                 DO ;
109  4          RETURN ;
110  4          END ;
111  3      FORM$FEED:
                 DO ;
112  4          IF FF$FLG THEN                  /* NEED FF EVERY OTHER PASS */
113  4              DO ;
114  5              CALL STROBE(0CH) ;
115  5              FF$FLG = FALSE ;
116  5              END ;
117  4          ELSE
                     FF$FLG = TRUE ;
118  4          INDEX = 0 ;
119  4          PTRFLG = 3 ;                    /* LINE$FEEDS */
120  4          END ;
121  3      LINE$FEEDS:
                 DO ;
122  4          IF INDEX < 6 THEN CALL STROBE(0AH) ;
124  4          ELSE CALL STROBE(0DH) ;
125  4          INDEX = INDEX + 1 ;
126  4          IF INDEX > 6 THEN
127  4              DO ;
128  5              LINE = 0 ;
129  5              INDEX = 0 ;
130  5              PTRFLG = 4 ;                /* INDENT */
131  5              END ;
132  4          END ;
133  3      INDENT:
                 DO ;
134  4          CALL STROBE(' ') ;
135  4          INDEX = INDEX + 1 ;
136  4          IF INDEX > 16 THEN
137  4              DO ;
138  5              INDEX = 0 ;
139  5              PTRFLG = 5 ;                /* LINE$DUMP */
140  5              END ;
141  4          END ;
142  3      LINE$DUMP:
```

```
                    DO ;
 143   4                CALL STROBE(PTR$LINE(LINE).CHAR(INDEX)) ;
 144   4                INDEX = INDEX + 1 ;
 145   4                IF INDEX > 79 THEN                /* END OF LINE */
 146   4                    DO ;
 147   5                    INDEX = 0 ;
 148   5                    PTRFLG = 6 ;                  /* CR$LF */
 149   5                    END ;
 150   4                END ;
 151   3            CR$LF:
                        DO CASE INDEX MOD 3 ;
 152   4                CR:
                            DO ;
 153   5                        CALL STROBE(0DH) ;
 154   5                        INDEX = 1 ;
 155   5                        END ;
 156   4                LF:
                            DO ;
 157   5                        CALL STROBE(0AH) ;
 158   5                        INDEX = 2 ;
 159   5                        END ;
 160   4                NEW$LINE:
                            DO ;
 161   5                        LINE = LINE + 1 ;
 162   5                        IF LINE > 23 THEN             /* IDLE */
 163   5                            PTRFLG = 1 ;
 164   5                        ELSE                          /* INDENT */
                                DO ;
 165   6                            INDEX = 0 ;
 166   6                            PTRFLG = 4 ;
 167   6                            END ;
 168   5                        END ;
 169   4                END ;
 170   3            END ;
 171   2        END ;

 172   1        END ;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 029FH    671D
    CONSTANT AREA SIZE = 0000H      0D
    VARIABLE AREA SIZE = 0F08H   3848D
    MAXIMUM STACK SIZE = 0008H      8D
    202 LINES READ
    0 PROGRAM WARNINGS
    0 PROGRAM ERRORS
```

DICTIONARY SUMMARY:

```
    195KB MEMORY AVAILABLE
    6KB MEMORY USED   (3%)
    0KB DISK SPACE USED
```

END OF PL/M-86 COMPILATION

APPENDIX D

File RUPTS.P86

The following listing shows the PLM86 source language for the module containing the interrupt initialization and servicing routines. This module declares PUBLIC the following variables and procedure:

INT$COUNTER	an array of eight DWORD counters which tally the interrupts from the Multibus.
NMI$COUNTER	a DWORD counter which tallies the number of bus timeout interrupts.
UNKNOWN$COUNTER	a DWORD counter which tallies extraneous interrupts.
BTO$FLAG	a BYTE state variable which signals the occurrence of a nonmaskable bus timeout interrupt.
INITIALIZE$INTERRUPTS	a PROCEDURE which initializes the 8259 interrupt controller chip, sets up 256 interrupt vectors, and unmasks and enables all interrupts.

```
iRMX 86 PL/M-86 V2.7 COMPILATION OF MODULE RUPTS_P86
OBJECT MODULE PLACED IN RUPTS.OBJ
COMPILER INVOKED BY:  :LANG:plm86 RUPTS.P86 .


                $COMPACT ROM NOINTVECTOR OPTIMIZE(3)
                $TITLE('NASA/ADFRF XAIDS MAINT INTERRUPT ROUTINES  14 Jan 1987')

                /*   NASA AMES DRYDEN FLIGHT RESEARCH FACILITY   R GLOVER   */

  1             RUPTS_P86: DO ;

                /* SBC 86/05 BOARD I/O PORT MAPPING */

  2   1         DECLARE PICFLG     LITERALLY '0C0H' ;   /* PROGRAMMABLE INTERRUPT CONTROLLER */
  3   1         DECLARE PICMSK     LITERALLY '0C2H' ;

                /* MISCELLANEOUS DECLARATIONS */

  4   1         DECLARE VECTOR (256) POINTER AT (000000H) ;
  5   1         DECLARE MPBTO          BYTE AT (0C701EH) ; /* PERPRO MAINT BTO FLAG */

  6   1         DECLARE INT$COUNTER (8) DWORD PUBLIC ; /* MULTIBUS INTERRUPT TALLY COUNTERS */
  7   1         DECLARE NMI$COUNTER     DWORD PUBLIC ; /* DEADMAN TIMER INTERRUPT COUNTER */
  8   1         DECLARE UNKNOWN$COUNTER DWORD PUBLIC ; /* REMAINING RUPT TYPES SHARE COUNTER */
  9   1         DECLARE BTO$FLAG         BYTE PUBLIC ; /* BUS TIMEOUT FLAG */


                /*  FOLLOWING ARE INTERRUPT TALLY ROUTINES */

 10   1         UNKNOWN$COUNTER$INCREMENT: PROCEDURE INTERRUPT 0 ;
 11   2         UNKNOWN$COUNTER = UNKNOWN$COUNTER + 1 ;
 12   2         END ;

 13   1         NMI$COUNTER$INCREMENT: PROCEDURE INTERRUPT 2 ;
 14   2         NMI$COUNTER = NMI$COUNTER + 1 ;
 15   2         BTO$FLAG,MPBTO = 0FFH ;      /* SET BTO FLAG LOCALLY & IN PERPRO */
 16   2         OUTWORD(PICFLG) = 40H ;      /* CLEAR FLIP-FLOP (OCW2 NO-OP) */
 17   2         END ;

 18   1         EOI: PROCEDURE;              /* END OF INTERRUPT */
 19   2         OUTPUT(PICFLG) = 20H ;
 20   2         END ;

 21   1         INT0$COUNTER$INCREMENT: PROCEDURE INTERRUPT 32 ;
 22   2         INT$COUNTER(0) = INT$COUNTER(0) + 1 ;
 23   2         CALL EOI ;
 24   2         END ;

 25   1         INT1$COUNTER$INCREMENT: PROCEDURE INTERRUPT 33 ;
 26   2         INT$COUNTER(1) = INT$COUNTER(1) + 1 ;
 27   2         CALL EOI ;
 28   2         END ;

 29   1         INT2$COUNTER$INCREMENT: PROCEDURE INTERRUPT 34 ;
 30   2         INT$COUNTER(2) = INT$COUNTER(2) + 1 ;
```

```
31  2     CALL EOI ;
32  2     END ;

33  1     INT3$COUNTER$INCREMENT: PROCEDURE INTERRUPT 35 ;
34  2     INT$COUNTER(3) = INT$COUNTER(3) + 1 ;
35  2     CALL EOI ;
36  2     END ;

37  1     INT4$COUNTER$INCREMENT: PROCEDURE INTERRUPT 36 ;
38  2     INT$COUNTER(4) = INT$COUNTER(4) + 1 ;
39  2     CALL EOI ;
40  2     END ;

41  1     INT5$COUNTER$INCREMENT: PROCEDURE INTERRUPT 37 ;
42  2     INT$COUNTER(5) = INT$COUNTER(5) + 1 ;
43  2     CALL EOI ;
44  2     END ;

45  1     INT6$COUNTER$INCREMENT: PROCEDURE INTERRUPT 38 ;
46  2     INT$COUNTER(6) = INT$COUNTER(6) + 1 ;
47  2     CALL EOI ;
48  2     END ;

49  1     INT7$COUNTER$INCREMENT: PROCEDURE INTERRUPT 39 ;
50  2     INT$COUNTER(7) = INT$COUNTER(7) + 1 ;
51  2     CALL EOI ;
52  2     END ;


53  1     INITIALIZE$INTERRUPTS: PROCEDURE PUBLIC ;
54  2     DECLARE I BYTE ;
55  2     DISABLE ;                          /* DISABLE INTERRUPTS */

           /*  LOADING RUPT VECTORS FOLLOWS */

56  2        DO I = 0 TO 255 ;
57  3        VECTOR(I) = INTERRUPT$PTR(UNKNOWN$COUNTER$INCREMENT) ;
58  3        END ;

59  2     VECTOR(02) = INTERRUPT$PTR(NMI$COUNTER$INCREMENT) ;
60  2     VECTOR(32) = INTERRUPT$PTR(INT0$COUNTER$INCREMENT) ;
61  2     VECTOR(33) = INTERRUPT$PTR(INT1$COUNTER$INCREMENT) ;
62  2     VECTOR(34) = INTERRUPT$PTR(INT2$COUNTER$INCREMENT) ;
63  2     VECTOR(35) = INTERRUPT$PTR(INT3$COUNTER$INCREMENT) ;
64  2     VECTOR(36) = INTERRUPT$PTR(INT4$COUNTER$INCREMENT) ;
65  2     VECTOR(37) = INTERRUPT$PTR(INT5$COUNTER$INCREMENT) ;
66  2     VECTOR(38) = INTERRUPT$PTR(INT6$COUNTER$INCREMENT) ;
67  2     VECTOR(39) = INTERRUPT$PTR(INT7$COUNTER$INCREMENT) ;

68  2     OUTPUT(PICFLG) = 00010011B ;    /* EDGE TRIGGER, SINGLE PIC */
69  2     OUTPUT(PICMSK) = 32 ;           /* VECTORED TO TYPES 32 THRU 39 */
70  2     OUTPUT(PICMSK) = 00001101B ;    /* NORMAL NESTING, BUFFERED MASTER, NORM EOI */
71  2     OUTPUT(PICMSK) = 00000000B ;    /* UNMASK ALL RUPTS */
72  2     OUTWORD(PICFLG) = 40H ;         /* CLEAR FLIP-FLOP (OCW2 NO-OP) */
73  2     ENABLE ;                        /* ENABLE RUPTS */
74  2     END ;
```

```
  75  1     END ;



MODULE INFORMATION:

     CODE AREA SIZE     = 0280H    640D
     CONSTANT AREA SIZE = 0008H      8D
     VARIABLE AREA SIZE = 002AH     42D
     MAXIMUM STACK SIZE = 0022H     34D
     110 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     195KB MEMORY AVAILABLE
     4KB MEMORY USED   (2%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

APPENDIX E

File MBM.P86

The following listing shows the PLM86 source language for the module containing the SBX251 magnetic bubble memory (MBM) initialization and servicing routines. This module declares PUBLIC the following procedures:

INITIALIZE$251	a PROCEDURE that initializes the SBX251 MBM multimodule
READ$251	a PROCEDURE typed BYTE that returns the success/fail status of SBX251 read operations. Three arguments are required: WORD that specifies the number of 64 byte blocks to be read, a WORD that specifies the initial block number, and a POINTER that identifies the destination buffer
WRITE$251	a PROCEDURE that copies the contents of EDIT$BUFFER to the SBX251 MBM. A BYTE argument is required that specifies the number of the file to receive the 20 blocks

```
iRMX 86 PL/M-86 V2.7 COMPILATION OF MODULE MBM_P86
OBJECT MODULE PLACED IN MBM.OBJ
COMPILER INVOKED BY:  :LANG:plm86 MBM.P86


            $COMPACT ROM OPTIMIZE(3)
            $TITLE('NASA/ADFRF XAIDS MAINT MBM ROUTINES  14 Jan 1987')

            /*   NASA AMES DRYDEN FLIGHT RESEARCH FACILITY   R GLOVER   */

            /*   SBX251 Magnetic Bubble Memory Multimodule Routines   */

  1         MBM_P86: DO ;

            /* EXTERNAL ROUTINES */

  2   2     WRITE: PROCEDURE (PTR) EXTERNAL; DECLARE PTR POINTER; END;
  5   2     HEX1:  PROCEDURE (VAL) EXTERNAL; DECLARE VAL BYTE; END;
  8   2     HEX2:  PROCEDURE (VAL) EXTERNAL; DECLARE VAL BYTE; END;

            /* SBC 86/05 BOARD I/O PORT MAPPING */

 11   1     DECLARE MBMDAT     LITERALLY '080H' ;   /* MAGNETIC BUBBLE MEMORY */
 12   1     DECLARE MBMCMD     LITERALLY '082H' ;

            /* I/O STATUS FLAGS */

 13   1     DECLARE MBM$BUSY                LITERALLY 'SHR(INPUT(MBMCMD),7)' ;
 14   1     DECLARE MBM$OP$COMPLETE         LITERALLY 'SHR(INPUT(MBMCMD),6)' ;
 15   1     DECLARE MBM$OP$FAIL             LITERALLY 'SHR(INPUT(MBMCMD),5)' ;
 16   1     DECLARE MBM$TIMING$ERROR        LITERALLY 'SHR(INPUT(MBMCMD),4)' ;
 17   1     DECLARE MBM$CORRECTABLE$ERROR   LITERALLY 'SHR(INPUT(MBMCMD),3)' ;
 18   1     DECLARE MBM$UNCORRECTABLE$ERROR LITERALLY 'SHR(INPUT(MBMCMD),2)' ;
 19   1     DECLARE MBM$PARITY$ERROR        LITERALLY 'SHR(INPUT(MBMCMD),1)' ;
 20   1     DECLARE MBM$FIFO$READY          LITERALLY '    INPUT(MBMCMD)   ' ;

            /* MISCELLANEOUS DECLARATIONS */

 21   1     DECLARE TRUE  LITERALLY '0FFH' ;
 22   1     DECLARE FALSE LITERALLY '0' ;
 23   1     DECLARE BLOCK         (64) BYTE ; /* BUBBLE MEMORY TEMP STORAGE */
 24   1     DECLARE LAST$251$COMMAND   BYTE ; /* BUBBLE MEMORY LAST COMMAND */
 25   1     DECLARE BYTE$COUNTER       WORD ; /* BUBBLE MEMORY DATA TRANSFER COUNTER */
 26   1     DECLARE TIMEOUT$COUNTER    WORD ; /* BUBBLE MEMORY DATA TRANSFER DEADMAN */
 27   1     DECLARE EDIT$BUFFER (1280) BYTE EXTERNAL ; /* USED BY JE COMMAND */

 28   1     MBM$ERROR: PROCEDURE (MSG$PTR) ;
 29   2     DECLARE MSG$PTR POINTER ;
 30   2     CALL WRITE(@(' MBM cmd ',0)) ;
 31   2     CALL HEX1(LAST$251$COMMAND) ;
 32   2     CALL WRITE(@(' error : ',07H,0)) ;
 33   2     CALL WRITE(MSG$PTR) ;
 34   2     END ;

 35   1     COMMAND$251: PROCEDURE (CMD) BYTE ;
 36   2     DECLARE CMD BYTE ;
```

```
37  2     LAST$251$COMMAND = CMD ;
38  2     OUTPUT(MBMCMD) = 00010000B OR ( CMD AND 00001111B ) ;
39  2     TIMEOUT$COUNTER = 0 ;
40  2        DO WHILE NOT MBM$BUSY ;
41  3        TIMEOUT$COUNTER = TIMEOUT$COUNTER - 1 ;
42  3        IF TIMEOUT$COUNTER = 0 THEN
43  3           DO ;
44  4           CALL MBM$ERROR(@('command accept timeout ',0)) ;
45  4           RETURN FALSE ;
46  4           END ;
47  3        END ;
48  2     RETURN TRUE ;
49  2     END ;

50  1     EXECUTE$251: PROCEDURE BYTE ;
51  2     TIMEOUT$COUNTER = 0 ;
52  2        DO WHILE MBM$BUSY ;
53  3        TIMEOUT$COUNTER = TIMEOUT$COUNTER - 1 ;
54  3        IF TIMEOUT$COUNTER = 0 THEN
55  3           DO ;
56  4           CALL MBM$ERROR(@('command execute timeout ',0)) ;
57  4           RETURN FALSE ;
58  4           END ;
59  3        END ;
60  2     RETURN TRUE ;
61  2     END ;

62  1     COMPLETE$251: PROCEDURE BYTE ;
63  2     TIMEOUT$COUNTER = 0 ;
64  2        DO WHILE NOT MBM$OP$COMPLETE ;
65  3        IF MBM$OP$FAIL THEN
66  3           DO ;
67  4           CALL MBM$ERROR(@('op fail status ',0)) ;
68  4           CALL HEX2(INPUT(MBMCMD)) ;
69  4           RETURN FALSE ;
70  4           END ;
71  3        TIMEOUT$COUNTER = TIMEOUT$COUNTER - 1 ;
72  3        IF TIMEOUT$COUNTER = 0 THEN
73  3           DO ;
74  4           CALL MBM$ERROR(@('op complete timeout ',0)) ;
75  4           RETURN FALSE ;
76  4           END ;
77  3        END ;
78  2     RETURN TRUE ;
79  2     END ;

80  1     SETUP$251: PROCEDURE(NBLOCKS,START) ;
81  2     DECLARE (NBLOCKS,START) WORD ;
82  2     OUTPUT(MBMCMD) = 00011101B ;          /* RESET FIFO */
83  2     CALL TIME(16) ;
84  2     OUTPUT(MBMCMD) = 00011111B ;          /* SOFTWARE RESET */
85  2     CALL TIME(16) ;
86  2     OUTPUT(MBMCMD) = 00001011B ;          /* START WITH BLOCK LENGTH REG */
87  2     CALL TIME(16) ;
88  2     OUTPUT(MBMDAT) = LOW(NBLOCKS) ;
89  2     OUTPUT(MBMDAT) = 00010000B OR ( HIGH(NBLOCKS) AND 00000111B ) ;
90  2     OUTPUT(MBMDAT) = 00100000B ;          /* ENABLE RCD ONLY */
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
 91  2      OUTPUT(MBMDAT) = LOW(START) ;
 92  2      OUTPUT(MBMDAT) = HIGH(START) AND 00000111B ;
 93  2      BYTE$COUNTER   = 64 * NBLOCKS ;
 94  2      END ;

 95  1      INITIALIZE$251: PROCEDURE PUBLIC ;
 96  2      CALL WRITE(@(13,10,'Bubble memory initialization ....',0)) ;
 97  2      CALL TIME(50*16) ;
 98  2      IF NOT COMMAND$251(9) THEN RETURN ;     /* ABORT COMMAND */
100  2      IF NOT EXECUTE$251    THEN RETURN ;
102  2      IF NOT COMPLETE$251   THEN RETURN ;
104  2      CALL TIME(100*16) ;
105  2      CALL SETUP$251(0,0) ;
106  2      IF NOT COMMAND$251(1) THEN RETURN ;     /* INITIALIZE COMMAND */
108  2      IF NOT EXECUTE$251    THEN RETURN ;
110  2      IF NOT COMPLETE$251   THEN RETURN ;
112  2      CALL WRITE(@(' complete.',0)) ;
113  2      END ;

114  1      READ$251: PROCEDURE (NBLOCKS,START,PTR) BYTE PUBLIC ;
115  2      DECLARE (NBLOCKS,START,INDEX) WORD, PTR POINTER, BUFFER BASED PTR (*) BYTE ;
116  2      TIMEOUT$COUNTER = 0 ;
117  2      INDEX = 0 ;
118  2      CALL SETUP$251(NBLOCKS,START) ;
119  2      IF NOT COMMAND$251(2) THEN RETURN FALSE ;
121  2          DO WHILE BYTE$COUNTER <> 0 ;
122  3          IF MBM$FIFO$READY THEN
123  3              DO ;
124  4              BUFFER(INDEX) = INPUT(MBMDAT) ;
125  4              INDEX = INDEX + 1 ;
126  4              BYTE$COUNTER = BYTE$COUNTER - 1 ;
127  4              END ;
128  3          ELSE
                    DO ;
129  4              TIMEOUT$COUNTER = TIMEOUT$COUNTER - 1 ;
130  4              IF TIMEOUT$COUNTER = 0 THEN
131  4                  DO ;
132  5                  CALL MBM$ERROR(@('read FIFO timeout ',0)) ;
133  5                  GO TO ABORT ;
134  5                  END ;
135  4              END ;
136  3          END ;
137  2      IF NOT EXECUTE$251  THEN GO TO ABORT ;
139  2      IF NOT COMPLETE$251 THEN GO TO ABORT ;
141  2      RETURN TRUE ;
142  2      ABORT:
             CALL INITIALIZE$251 ;
143  2      RETURN FALSE ;
144  2      END ;

145  1      WRITE$251: PROCEDURE (FILE) PUBLIC ;
146  2      DECLARE FILE BYTE, INDEX WORD ;
147  2      TIMEOUT$COUNTER = 0 ;
148  2      INDEX = 0 ;
149  2      CALL SETUP$251(20,20*FILE) ;
150  2      IF NOT COMMAND$251(3) THEN RETURN ;
152  2          DO WHILE BYTE$COUNTER <> 0 ;
```

```
153   3          IF MBM$FIFO$READY THEN
154   3             DO ;
155   4             OUTPUT(MBMDAT) = EDIT$BUFFER(INDEX) ;
156   4             INDEX = INDEX + 1 ;
157   4             BYTE$COUNTER = BYTE$COUNTER - 1 ;
158   4             END ;
159   3          ELSE
                     DO ;
160   4             TIMEOUT$COUNTER = TIMEOUT$COUNTER - 1 ;
161   4             IF TIMEOUT$COUNTER = 0 THEN
162   4                DO ;
163   5                CALL MBM$ERROR(@('write FIFO timeout ',0)) ;
164   5                GO TO ABORT ;
165   5                END ;
166   4             END ;
167   3          END ;
168   2       IF NOT EXECUTE$251  THEN GO TO ABORT ;
170   2       IF NOT COMPLETE$251 THEN GO TO ABORT ;
172   2       RETURN ;
173   2       ABORT:
              CALL INITIALIZE$251 ;
174   2       END ;


175   1       END ;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
MODULE INFORMATION:

     CODE AREA SIZE     = 028AH    650D
     CONSTANT AREA SIZE = 00C1H    193D
     VARIABLE AREA SIZE = 0049H     73D
     MAXIMUM STACK SIZE = 0022H     34D
     191 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     195KB MEMORY AVAILABLE
     6KB MEMORY USED   (3%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

REFERENCES

1. Glover, Richard D.: Aircraft Interrogation and Display System: A Ground Support Equipment for Digital Flight Systems. NASA TM-81370, 1982.

2. Glover, Richard D.: Application Experience With the NASA Aircraft Interrogation and Display System: A Ground Support Equipment for Digital Flight Systems. Proceedings of IEEE/AIAA Fifth Digital Avionics Systems Conference, p. 17.3.1 to 17.3.10, Seattle, Washington, Oct. 31-Nov. 3, 1983.

3. Glover, Richard D.: Design and Initial Application of the Extended Aircraft Interrogation and Display System: Multiprocessing Ground Support Equipment for Digital Flight Systems. NASA TM-86740, 1987.

TABLE 1. — TOP-LEVEL COMMAND KEYSTROKES

Keystroke	Function invoked
0-9	Executes MBM job file d0-d9 from current decade d
(esc)	Reinitializes MBM and shows signon
(space)	Repeats last J(space), M(space), or I(space) rerun
M	Invokes memory servicing routine
H	Shows help page
I	Invokes I/O servicing routine
J	Invokes job servicing routine
R	Displays the interrupt tally counters
S	Selects new memory segment
(cntl C)	Exits job buffer append mode
(cntl B)	Invokes MBM backup routine
(cntl R)	Invokes MBM restore routine
(return)	Shows command menu

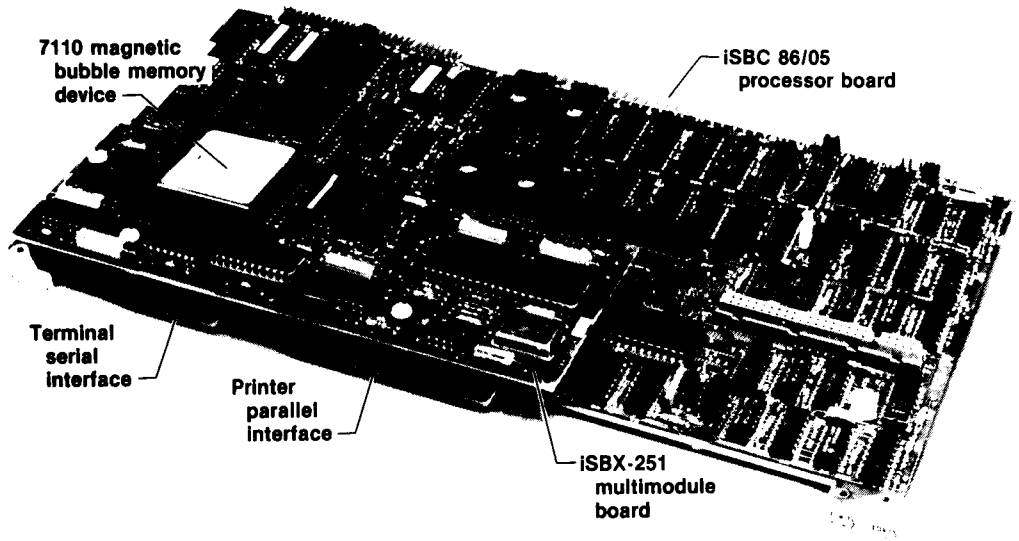

Figure 1. XAIDS maintenance processor.

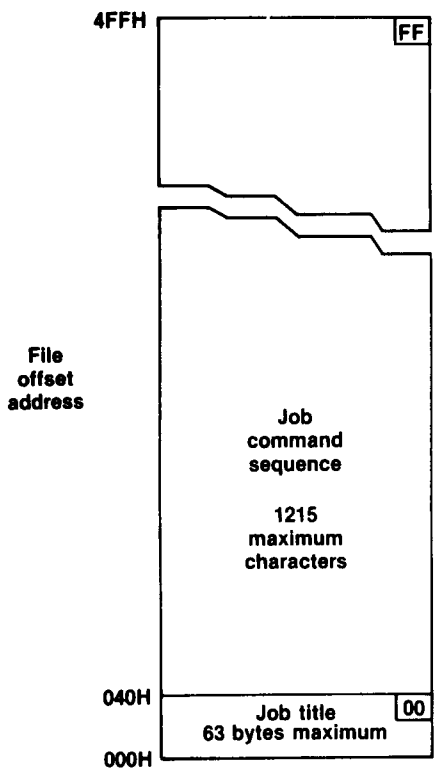

Figure 2. Job file mapping.

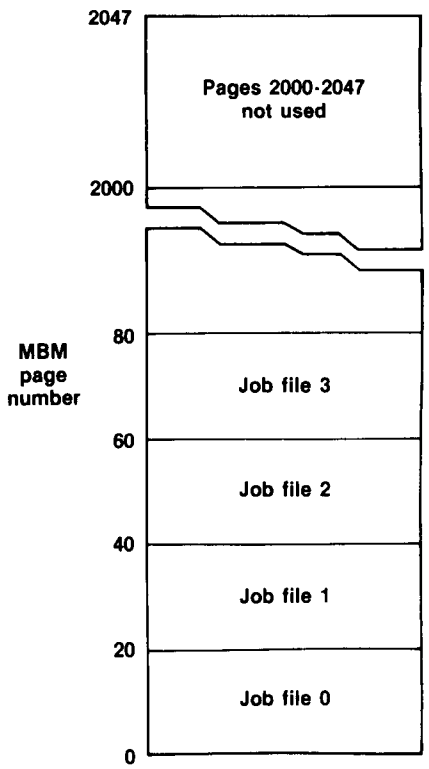

Figure 3. MBM file structure.

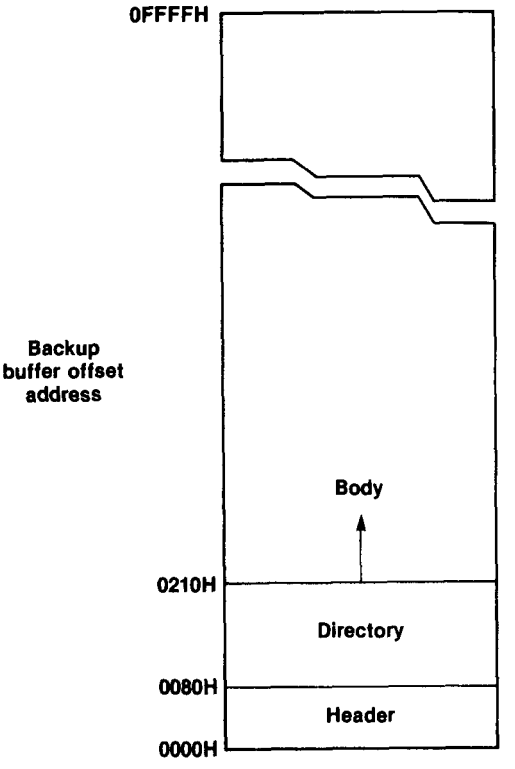

Figure 4. Backup file structure.

```
Baud rate = 19200

Bubble memory initialization .... complete.

          XX     XX         AA         IIIIII   DDDDDDD      SSSSSSS
           XX   XX         AAAA          II     DD    DD    SS     SS
            XX XX         AA  AA        II     DD     DD   SS
             XXX         AA    AA       II     DD     DD    SSSSSSS
            XX XX       AAAAAAAAAA      II     DD     DD          SS
           XX   XX     AA        AA     II     DD    DD    SS     SS
          XX     XX   AA          AA  IIIIII   DDDDDDD      SSSSSSS

         MMM     MMM         AA         IIIIII   NN      NN   TTTTTTTT
         MM MM MM MM        AAAA          II     NNN     NN      TT
         MM  MMM  MM       AA  AA         II     NN NN   NN      TT
         MM   M   MM      AA    AA        II     NN  NN  NN      TT
         MM       MM     AAAAAAAAAA       II     NN   NN NN      TT
         MM       MM    AA        AA      II     NN    NNN      TT
         MM       MM   AA          AA  IIIIII   NN     NN      TT

NASA/ADFRF XAIBS Maintenance Processor   14 January 1987   R. Glover
Menu: Help, 0-9, J0-J9, JE, JD, I(sp), M(sp), J(sp), (sp), IRB, IRW, IFB, IFW,
      INB, INW, MWB, MWW, MRB, MRW, MRD, MRP, MRA, MFB, MFW, MFD, MS, R, S
```

Display 1. Initial signon.

```
Menu: Help, 0-9, J0-J9, JE, JD, I(sp), M(sp), J(sp), (sp), IRB, IRW, IFB, IFW,
      INB, INW, MWB, MWW, MRB, MRW, MRD, MRP, MRA, MFB, MFW, MFD, MS, R, S
.Help:
(u)    = Job load & run.      u = 0 thru 9, selects file = 10*d + u.
J(d)   = Job decade.          d = 0 thru 9, selects decade.
JE     = Job edit.            Views/modifies/loads/stores job buffer.
JD     = Job directory.       Shows titles for selected job decade.
I(sp)  = I/O read rerun.      Repeat current "IR" setup.
M(sp)  = Memory read rerun.   Repeat current "MR" setup.
J(sp)  = Job rerun.           Repeat current job buffer.
(sp)   = Space bar.           Repeats last rerun ( I(sp), M(sp), or J(sp) )
@      = B or W               B(yte) or W(ord).
IR@    = I/O read.            Sets up byte or word I/O port group display.
IW@    = I/O write.           Repetitive byte or word outputs to selected port.
IF@    = I/O fill.            Writes byte or word to a block of ports.
MW@    = Memory write.        Repetitive byte or word memory writes.
#      = B, W, D, P, or A     B(yte), W(ord), D(word), P(ointer), or A(SCII).
MR#    = Memory read.         Sets up byte, word, dword, pointer, or ASCII dump.
@      = B, W, D, or A        B(yte), W(ord), D(word), or A(SCII).
MF@    = Memory fill.         Byte, word, dword, or ASCII string fill.
MS     = Memory substitute.   Byte scan/modify.
R      = Rupts.               Displays multibus interrupt counters.
S      = Segment.             Reloads memory base register.
```

Display 2. Help page.

```
.Job decade 0u selected
Job Directory for decade 0
  00 : Scan PERPRO RAM
  01 : Display PERPRO rupts : 0, 1, 2, clock, 4, xmt, rcv, 7
  02 : Display PERPRO stack
  03 : Display PERPRO program version
  04 :
  05 : Send LP message "Line printer interface operational."
  06 : Place PERPRO in RESET mode.
  07 : Place PERPRO in RMX86 I/O mode.
  08 : Place PERPRO in XAIDS I/O mode.
  09 : Display TCU regs : sec, min, hour, day, month, year, status
.Job Editor
File  = ??
Title =
   01 =
Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
Load Job number (0-99 decimal) ) 0
File  = 00
Title = Scan PERPRO RAM
   01 = S C 0 0 0 (CR)
   02 = M R B 4 0 0 0 (CR) 7 F F F (CR)
Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
```

Display 3. Job file examination.

```
.Job file 00 = Scan PERPRO RAM
C000:4000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4010 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4020 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4030 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4040 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4050 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4060 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4070 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4080 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:4090 =  00 00 00 00  00 00 00 00   00 00 00 00  00
Job aborted
.Job file 01 = Display PERPRO rupts : 0, 1, 2, clock, 4, xmt, rcv, 7
C000:7310 =  00 00 00 00  00 00 00 00   00 00 00 00  5F 02 15 00
C000:7320 =  00 00 00 00  04 FA 05 00   85 37 00 00  00 00 00 00
.Job file 02 = Display PERPRO stack
C000:7FC0 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
C000:7FD0 =  00 00 00 00  00 00 00 00   00 00 00 00  5F 04 4B 40
C000:7FE0 =  5F 04 20 74  00 20 B4 0C   2F 0E 0D 05  2F 0E 37 04
C000:7FF0 =  5F 04 37 04  37 04 1C 70   A5 00 84 00  29 0A 42 0D
.Job file 03 = Display PERPRO program version
C000:7280 =  PERPRO-II V1.6   27 FEB 1985   R GLOVER
.Job file 04 is empty.
```

Display 4. Job file executions.

```
.Job Editor
File  = 00
Title = Scan PERPRO RAM
   01 = S C 0 0 0 (CR)
   02 = M R B 4 0 0 0 (CR) 7 F F F (CR)
Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
Erase entire buffer ? ) Y
File  = ??
Title =
   01 =
Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
Appending to buffer - use (cntl C) to terminate.
)Current memory segment register : C400
Enter desired segment value (0 - FFFF) ) C400
Updated memory segment register : C400
)Memory Read Bytes from memory addr 0 to memory addr F
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
)Job Editor
File  = ??
Title =
   01 = S C 4 0 0 (CR)
   02 = M R B 0 (CR) F (CR)
Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
```

Display 5. Editing job buffer.

```
Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
Appending to buffer - use (cntl C) to terminate.
)Current memory segment register : C400
Enter desired segment value (0 - FFFF) ) C400
Updated memory segment register : C400
)Memory Read Bytes from memory addr 0 to memory addr F
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
)Job Editor
File  = ??
Title =
   01 = S C 4 0 0 (CR)
   02 = M R B 0 (CR) F (CR)
Edit menu : A(ppend) D(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
.Job file ?? =
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
.Job file ?? =
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
.Job file ?? =
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
.Job file ?? =
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
.Job file ?? =
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
```

Display 6. Executing job buffer.

```
.Job Editor
File  = ??
Title = Demo of MFB, MRB, MS, and MRA
   01 = S C 4 0 0 (CR)
   02 = M F B 0 (CR) 0 (CR) F (CR)
   03 = M R B 0 (CR) F (CR)
   04 = M F B A A (CR) 4 (CR) B (CR)
   05 = M R B 0 (CR) F (CR)
   06 = M S 4 (CR) 4 7 (CR) 7 2 (CR) 6 5 (CR) 6 5 (CR) 7 4 (CR) 6 9 (CR) 6 E (CR
) 6 7 (CR) 7 3 (CR) (ESC)
   07 = M R B 0 (CR) F (CR)
   08 = M R A 4 (CR)
Edit menu : A(ppend) B(elete) E(rase) L(oad) S(ave) T(itle) (esc)=Exit
.Job file ?? = Demo of MFB, MRB, MS, and MRA
Memory Fill with Byte value 0 from memory addr 0 to memory addr F
C400:0000 =  00 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00
Memory Fill with Byte value AA from memory addr 4 to memory addr B
C400:0000 =  00 00 00 00  AA AA AA AA   AA AA AA AA  00 00 00 00
C400:0004=AA)47   C400:0005=AA)72   C400:0006=AA)65   C400:0007=AA)65
C400:0008=AA)74   C400:0009=AA)69   C400:000A=AA)6E   C400:000B=AA)67
C400:000C=00)73   C400:000D=00)
C400:0000 =  00 00 00 00  47 72 65 65   74 69 6E 67  73 00 00 00
C400:0004 =  Greetings
```

Display 7. Miscellaneous memory operations.

```
C400:0004=AA)47   C400:0005=AA)72   C400:0006=AA)65   C400:0007=AA)65
C400:0008=AA)74   C400:0009=AA)69   C400:000A=AA)6E   C400:000B=AA)67
C400:000C=00)73   C400:000D=00)
C400:0000 =  00 00 00 00  47 72 65 65   74 69 6E 67  73 00 00 00
C400:0004 =  Greetings
.Backup current files ? ) y  All decades ? ) n  Which decades ? ) 0 2 6
  Job file 00 = Scan PERPRO RAM
  Job file 01 = Display PERPRO rupts : 0, 1, 2, clock, 4, xmt, rcv, 7
  Job file 02 = Display PERPRO stack
  Job file 03 = Display PERPRO program version
  Job file 05 = Send LP message "Line printer interface operational."
  Job file 06 = Place PERPRO in RESET mode.
  Job file 07 = Place PERPRO in RMX86 I/O mode.
  Job file 08 = Place PERPRO in XAIDS I/O mode.
  Job file 09 = Display TOD regs : sec, min, hour, day, month, year, status
  Job file 20 = Scan CENPRO RAM
  Job file 21 = Scan Megalink LAN RAM
  Job file 22 = Scan IEEE 488 controller RAM
  Job file 25 = Scan IEEE 488 controller I/O ports
  Job file 26 = Read MSS status I/O port
  Job file 27 = Scan Xylogics tape controller I/O ports
  Job file 60 = Display CCU #1 I/O ports : status, USART rcvr data.
  Job file 61 = Display CCU #2 I/O ports : status, USART rcvr data.
```

Display 8. MBM backup operation.

```
NASA/ADFRF XAIDS Maintenance Processor   14 January 1987   R. Glover
Menu: Help, 0-9, J0-J9, JE, JB, I(sp), M(sp), J(sp), (sp), IRB, IRW, IFB, IFW,
      IWB, IWW, MWB, MWW, MRB, MRW, MRD, MRP, MRA, MFB, MFW, MFD, MS, R, S
.Job decade 2u selected
Job Directors for decade 2
  20 : Scan CENPRO RAM
  21 : Scan Megalink LAN RAM
  22 : Scan IEEE 488 controller RAM
  23 :
  24 :
  25 : Scan IEEE 488 controller I/O ports
  26 : Read MSS status I/O port
  27 : Scan Xylogics tape controller I/O ports
  28 :
  29 :
.Job file 25 = Scan IEEE 488 controller I/O ports
I/O  4880 =  FF 9A 9C FF  FF 43 12 FF   F0 FF FF FF  FF 00 FF FF
I/O  4890 =  00 00 00 00  FE FE 00 00   00 00 7F 7F  FE FE FE FE
.Job file 26 = Read MSS status I/O port
I/O  0101 =  07
.Job file 27 = Scan Xylogics tape controller I/O ports
I/O  4720 =  00 00 00 00  00 00
```

Display 9. I/O read examples.

1. Report No. NASA TM-100406	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Concept of a Programmable Maintenance Processor Applicable to Multiprocessing Systems		5. Report Date February 1988 6. Performing Organization Code
7. Author(s) Richard D. Glover		8. Performing Organization Report No. H-1425
9. Performing Organization Name and Address NASA Ames Research Center Dryden Flight Research Facility P.O. Box 273 Edwards, CA 93523-5000		10. Work Unit No. RTOP 533-02-51 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		

16. Abstract

A programmable maintenance processor concept applicable to multiprocessing systems has been developed at the NASA Ames Research Center's Dryden Flight Research Facility. This stand-alone-processor concept is intended to provide support for system and application software testing as well as hardware diagnostics. An initial mechanization has been incorporated into the extended aircraft interrogation and display system (XAIDS) which is multiprocessing general-purpose ground support equipment. The XAIDS maintenance processor has independent terminal and printer interfaces and a dedicated magnetic bubble memory that stores system test sequences entered from the terminal. This report describes the hardware and software embodied in this processor and shows a typical application in the check-out of a new XAIDS.

17. Key Words (Suggested by Author(s)) Computer debugging Computer maintenance Multiprocessing systems	18. Distribution Statement Unclassified — Unlimited Subject category 62		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 71	22. Price* A04

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