Concept of a Programmable Maintenance Processor Applicable to Multiprocessing Systems

Richard D. Glover

February 1988
Concept of a Programmable Maintenance Processor Application to Multiprocessing Systems

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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>1</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>NOMENCLATURE</td>
<td>1</td>
</tr>
<tr>
<td>BACKGROUND</td>
<td>2</td>
</tr>
<tr>
<td>EVOLUTION OF XAIDS MAINT</td>
<td>3</td>
</tr>
<tr>
<td>HARDWARE CONFIGURATION</td>
<td>6</td>
</tr>
<tr>
<td>SOFTWARE DESCRIPTION</td>
<td>6</td>
</tr>
<tr>
<td>EXAMPLES OF TYPICAL OPERATIONS</td>
<td>9</td>
</tr>
<tr>
<td>CONCLUDING REMARKS</td>
<td>11</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>13</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>17</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td>48</td>
</tr>
<tr>
<td>APPENDIX D</td>
<td>53</td>
</tr>
<tr>
<td>APPENDIX E</td>
<td>57</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>62</td>
</tr>
<tr>
<td>TABLE 1</td>
<td>63</td>
</tr>
<tr>
<td>FIGURES</td>
<td>64</td>
</tr>
<tr>
<td>DISPLAYS</td>
<td>66</td>
</tr>
</tbody>
</table>

PRECEDING PAGE BLANK NOT FILMED
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
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 multiprocessor 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 multiprocessor 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 PCO-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 acknowledge signal, bit PC6 is used for the printer busy signal.

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 JSCMD, 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 <cnt1 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: <cnt1 B> to create an MBM backup file and <cnt1 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 PRINTERS$BUFFER is called whenever a character is written to the terminal so that the same character may be placed in CRT$LINE. The routine PRINTERS$SNAPSHOT checks whether a <cnt1 P> has been entered; if so, it copies CRT$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 <cnt1 P> is ignored. The routine PRINTERS$SERVICE is called from all wait loops and thus serves as the background task for managing the dump of PTR$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 (accessible 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> <O> 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> <O> <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 <O>. It was quickly aborted using the <esc> key and the TLCI prompt again appeared. Thereafter <1> was entered followed by <2>, <3>, <4>, and <cnt1 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> <D> <O> <O> <return>, thus creating a new line 1 in the job buffer. A new line 2 was created by entering <M> <R> <B> <O> <return> <P> <return>, which executed a dump of the desired block of PERPRO RAM. Finally append mode was terminated by entering <cnt1 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
Aeros Research Center
Dryden Flight Research Facility
Edwards, California, June 3, 1987

ORIGINAL PAGE IS OF POOR QUALITY.
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.ink, maint, maint.mp

linkB6 maint.obj, & Main program + terminal routines
ruts.obj, & Interrupt support routines
mbm.obj, & Magnetic bubble memory routines
lf.obj, & Line printer routines
?langs!plm86.lib & PLM library routines
to maint.ink print(maint.mp1)

locB6 maint.ink to maint
order(classes(data, stack, code)) &
inicode(0fffd0h) &
addresses(classes(data(400h), code(0fc000h))) &
objectcontrols(purse) &
print(maint.mp2) &
printcontrols(nopurse) &
symbolcolumns(3) &
bootstrap

delete maint.lo, maint.hi

ipps
i h B6
format maint(0fc000h, 0fffffff)
3
2
1
0 to maint.lo
1 to maint.hi
exit

; Invoke IPPS and perform following:
;  init hex B6
;  type 2732a
;  copy maint.lo to prom
;  copy maint.hi to prom
;  exit
INPUT FILES: MAINT.OBJ, RUPTS.OBJ, MBM.OBJ, LP.OBJ, :LANG:PLM86.LIB
OUTPUT FILE: MAINT.LNK
CONTROLS SPECIFIED IN INVOCATION COMMAND:
PRINT(MAINT.NP1)
DATE: 01/14/87 TIME: 10:23:13

LINK MAP OF MODULE MAINT.P86

<table>
<thead>
<tr>
<th>LENGTH</th>
<th>ADDRESS</th>
<th>ALIGN</th>
<th>SEGMENT</th>
<th>CLASS</th>
<th>OVERLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>3705H</td>
<td>--------</td>
<td>G</td>
<td>CODE</td>
<td>CODE</td>
<td></td>
</tr>
<tr>
<td>185CH</td>
<td>--------</td>
<td>W</td>
<td>DATA</td>
<td>DATA</td>
<td></td>
</tr>
<tr>
<td>0006H</td>
<td>--------</td>
<td>W</td>
<td>STACK</td>
<td>STACK</td>
<td></td>
</tr>
<tr>
<td>0000H</td>
<td>--------</td>
<td>W</td>
<td>MEMORY</td>
<td>MEMORY</td>
<td></td>
</tr>
<tr>
<td>0060H</td>
<td>0000H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0190H</td>
<td>00080H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFF0H</td>
<td>00210H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0400H</td>
<td>00000H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001H</td>
<td>C701EH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000H</td>
<td>--------</td>
<td>G</td>
<td>??SEG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

INPUT MODULES INCLUDED:
MAINT.OBJ(MAINT.P86)
RUPTS.OBJ(RUPTS.P86)
MBM.OBJ(MBM.P86)
LP.OBJ(LP.P86)
:LANG:PLM86.LIB(LO_DWORD_MULTIPLY)

ORIGINAL PAGE IS OFPOOR QUALITY
INPUT FILE: MAINT.LNK
OUTPUT FILE: MAINT

CONTROLS SPECIFIED IN INVOCATION COMMAND:
- TO MAINT ORDER (CLASSES (DATA/STACK/CODE)) INITCODE
- OFFFDH ADDRESSES (CLASSES (DATA (400H), CODE (0F000H))) OBJECTCONTROLS (PURGE)
- PRINT (MAINT, MP2) PRINTCONTROLS (NOPURGE) SYMBOLCOLUNMS (3) BOOTSTRAP

DATE: 01/14/87 TIME: 10:23:30

SYMBOL TABLE OF MODULE MAINT..PB6

<table>
<thead>
<tr>
<th>BASE</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>SYMBOL</th>
<th>BASE</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>SYMBOL</th>
<th>BASE</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0040H</td>
<td>0080H</td>
<td>PUB</td>
<td>BTODFLAG</td>
<td>0040H</td>
<td>0560H</td>
<td>PUB</td>
<td>EDITBUFFER</td>
<td>0040H</td>
<td>00A0H</td>
<td>PUB</td>
<td>FTCOUNTER</td>
</tr>
<tr>
<td>0040H</td>
<td>0880H</td>
<td>PUB</td>
<td>KEY</td>
<td>0040H</td>
<td>0C00H</td>
<td>PUB</td>
<td>NMI COUNTER</td>
<td>0040H</td>
<td>0C04H</td>
<td>PUB</td>
<td>UKNOWNCOUNTER</td>
</tr>
<tr>
<td>FC00H</td>
<td>16F0H</td>
<td>PUB</td>
<td>HEX1</td>
<td>FC00H</td>
<td>1707H</td>
<td>PUB</td>
<td>HEX2</td>
<td>FC00H</td>
<td>32F0H</td>
<td>PUB</td>
<td>INITIALIZE251</td>
</tr>
<tr>
<td>FC00H</td>
<td>303FH</td>
<td>PUB</td>
<td>INITIALIZEINTR</td>
<td>FC00H</td>
<td>3460H</td>
<td>PUB</td>
<td>INITIALIZEPRINT</td>
<td>FC00H</td>
<td>36F0H</td>
<td>PUB</td>
<td>LD...MUL</td>
</tr>
<tr>
<td>FC00H</td>
<td>3489H</td>
<td>PUB</td>
<td>PRINTERBUFFER</td>
<td>FC00H</td>
<td>35A3H</td>
<td>PUB</td>
<td>PRINTERSERVICE</td>
<td>FC00H</td>
<td>35F0H</td>
<td>PUB</td>
<td>PRINTERSNAPSHOT</td>
</tr>
<tr>
<td>FC00H</td>
<td>3371H</td>
<td>PUB</td>
<td>READ251</td>
<td>FC00H</td>
<td>1658H</td>
<td>PUB</td>
<td>WRITE</td>
<td>FC00H</td>
<td>33E0H</td>
<td>PUB</td>
<td>WRITE251</td>
</tr>
</tbody>
</table>

MEMORY MAP OF MODULE MAINT..PB6

MODULE START ADDRESS PARAGRAPH = FFFDH OFFSET = 0006H

SEGMENT MAP

<table>
<thead>
<tr>
<th>START</th>
<th>STOP</th>
<th>LENGTH</th>
<th>ALIGN</th>
<th>NAME</th>
<th>CLASS</th>
<th>OVERLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>003FFH</td>
<td>0400H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>DATA</td>
<td></td>
</tr>
<tr>
<td>0040H</td>
<td>01F5H</td>
<td>15CH</td>
<td>M</td>
<td>DATA</td>
<td>MEMORY</td>
<td></td>
</tr>
<tr>
<td>01F5H</td>
<td>01FEH</td>
<td>006H</td>
<td>M</td>
<td>STACK</td>
<td>STACK</td>
<td></td>
</tr>
<tr>
<td>0000H</td>
<td>0007FH</td>
<td>0080H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008H</td>
<td>0020H</td>
<td>0190H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0021H</td>
<td>00FFH</td>
<td>FDF0H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C701H</td>
<td>C701EH</td>
<td>0001H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC00H</td>
<td>FF70H</td>
<td>3705H</td>
<td>G</td>
<td>CODE</td>
<td>CODE</td>
<td></td>
</tr>
<tr>
<td>FF71H</td>
<td>FF710H</td>
<td>0000H</td>
<td>G</td>
<td>??SEG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFD0H</td>
<td>FFFF6H</td>
<td>0019H</td>
<td>A</td>
<td>??LOC86..INITCO</td>
<td>CODE</td>
<td></td>
</tr>
<tr>
<td>FFFF0H</td>
<td>FFFF4H</td>
<td>0005H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFF6H</td>
<td>FFFF6H</td>
<td>0000H</td>
<td>W</td>
<td>MEMORY</td>
<td>MEMORY</td>
<td></td>
</tr>
</tbody>
</table>

GROUP MAP

ADDRESS GROUP OR SEGMENT NAME
FC00H CGROUP CODE
0040H DGROUP DATA
APPENDIX B

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.
/* 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 */

MAIN..P86: DO ;

/* EXTERNAL ROUTINES DECLARATIONS */

INITIALIZE%INTERRUPTS: PROCEDURE EXTERNAL ; END
INITIALIZE%PRINTER: PROCEDURE EXTERNAL ; END
PRINTER%SNAPSHOT: PROCEDURE EXTERNAL ; END
PRINTER%SERVICE: PROCEDURE EXTERNAL ; END
PRINTER%BUFFER: PROCEDURE (CHAR) EXTERNAL ; DECLARE CHAR BYTE ; END
INITIALIZE%251: PROCEDURE EXTERNAL ; END
WRITE%251: PROCEDURE (FILE) EXTERNAL ; DECLARE FILE BYTE ; END
READ%251: PROCEDURE (NBLOCKS;START;PTR) BYTE EXTERNAL ;
DECLARE (NBLOCKS;START) WORD; PTR POINTER ; END

DECLARE BACKUP%RESTORE%BUFFER LITERALLY '0B0000H' ;
DECLARE HDR (32) DWORD AT (BACKUP%RESTORE%BUFFER+000) ;
DECLARE DIR (100) STRUCTURE (INDEX WORD; LENGTH WORD)
   AT (BACKUP%RESTORE%BUFFER+128) ;
DECLARE INFO (05008) BYTE AT (BACKUP%RESTORE%BUFFER+528) ;

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

DECLARE PPIA LITERALLY '00CH' ; /* PROGRAMMABLE PERIPHERAL INTERFACE */
DECLARE PPIB LITERALLY '00CH' ;
DECLARE PPIE LITERALLY '00CH' ;
DECLARE PPIFLG LITERALLY '00CH' ;
DECLARE PIT0 LITERALLY '00DH' ; /* PROGRAMMABLE INTERVAL TIMER */
DECLARE PIT1 LITERALLY '00EH' ;
DECLARE PIT2 LITERALLY '002H' ;
DECLARE PIT3 LITERALLY '006H' ;
DECLARE USDATA LITERALLY '008H' ; /* USART */
DECLARE USFLAG LITERALLY '00AH' ;

/* I/O STATUS FLAGS */

DECLARE RXRDY LITERALLY 'SHR(INPUT(USFLAG)+1)' ;
DECLARE TXRDY LITERALLY 'INPUT(USFLAG)' ;
/* MISCELLANEOUS DECLARATIONS */

37  DECLARE NULL LITERALLY '000H';
38  DECLARE CNTLSB LITERALLY '002H';
39  DECLARE CNTLSC LITERALLY '003H';
40  DECLARE BEL LITERALLY '007H';
41  DECLARE BS LITERALLY '008H';
42  DECLARE LF LITERALLY '00AH';
43  DECLARE CR LITERALLY '00BH';
44  DECLARE CNTLSQ LITERALLY '011H';
45  DECLARE CNTLDR LITERALLY '012H';
46  DECLARE CNTLSR LITERALLY '013H';
47  DECLARE ESC LITERALLY '017H';
48  DECLARE SPACE LITERALLY '020H';
49  DECLARE CNTLM LITERALLY '01CH';
50  DECLARE CNTLSR LITERALLY '01DH';
51  DECLARE CNTLSS LITERALLY '01EH';
52  DECLARE CR LITERALLY '00AH';
53  DECLARE SPACE LITERALLY '020H';
54  DECLARE RUB LITERALLY '07FH';
55  DECLARE JOnIlkE LITERALLY 'JOBS)IODE=O';
56  DECLARE JOBSSETW LITERALLY 'JOB)I(ODE=1'
57  DECLARE CRMRSLF LITERALLY 'JOB)I(ODE=2'
58  DECLARE EXITSETUP LITERALLY '(JOBSSETUP AND KEY.CnTL%CC) '
59  DECLARE TRUE LITERALLY 'OFF';
60  DECLARE FALSE LITERALLY '0';
61  DECLARE FOREVER LITERALLY 'WHILE TRUE';

62  DECLARE NHC#PTR POINTER ;
63  DECLARE BYTE#DATA BASED NHC#PTR BYTE ;
64  DECLARE WORD#DATA BASED NHC#PTR WORD ;
65  DECLARE DWORD#DATA BASED NHC#PTR DWORD ;
66  DECLARE POINTER#DATA BASED NHC#PTR POINTER ;
67  DECLARE BYTENS#SAVE BYTE ;
68  DECLARE WORDNS#SAVE WORD ;
69  DECLARE DWORDNS#SAVE DWORD ;
70  DECLARE POINTERNS#SAVE POINTER ;
71  DECLARE PTR STRUCTURE (OFFSET WORD, SEG WORD) AT (#NHC#PTR) ;
72  DECLARE SEGMENT LITERALLY 'PTR.SEG' ;
73  DECLARE FROM#ADDR WORD AT (#NHC#PTR) ;
74  DECLARE INCREMENT WORD ;
75  DECLARE DATA#TYPE BYTE ;
76  DECLARE HEX#WORD DWORD ;
77  DECLARE HEX#BYTE BYTE AT (#HEX#WORD) ;
78  DECLARE FILLER#WORD DWORD ;
79  DECLARE FILLER#BYTE BYTE AT (#FILLER#WORD) ;
80  DECLARE TOADDR WORD ;
81  DECLARE KEY BYTE PUBLIC ;
82  DECLARE CHBUF (64) BYTE ;
83  DECLARE NCHAR BYTE ;
84  DECLARE CHD#SAVE BYTE ;
85  DECLARE USARTSINITSBYTE BYTE ;
86  DECLARE UART#INIT#BYTE BYTE ;
DECLARE OUTPUT*DISABLED BYTE ; /* SCREEN OUTPUT CONTROL */
DECLARE JOB.BUFFER*PTR POINTER ; /* POINTER TO CURRENT JOB BUFFER */
DECLARE JOB.BUFFER BASED JOB.BUFFER*PTR (1280) BYTE ; /* CURRENT JOB BUFFER */
DECLARE JOB.BUFFER*INDEX Word ; /* INDEX TO NEXT CHAR IN CURRENT JOB BUFFER */
DECLARE JOB.MODE BYTE ; /* 0=IDLE 1=SETUP 2=RUN 3=EDIT */
DECLARE FILE*DECADE BYTE ; /* 0 THRU 9 */
DECLARE FILE*NUMBER BYTE ; /* 0 THRU 99 */
DECLARE FILE&BUFFER (1280) BYTE ; /* USED BY FILE LOADER COMMANDS 0-9 */
DECLARE EDIT&FILE*NUMBER BYTE ; /* MATCHING FILE NUMBER (=255 IF NOT) */
DECLARE EDIT&BUFFER (1280) BYTE ; /* PUBLIC */
DECLARE ITEM*INDEX (128) Word ; /* EDIT BUFFER ITEM INDEXES */
DECLARE ITEM*ITEMS BYTE ; /* NUMBER OF ITEMS IN EDIT BUFFER */
DECLARE BLOCK (64) BYTE ; /* BUBBLE MEMORY TEMP STORAGE */
DECLARE ASCII (16) BYTE ; /* DATA ('0123456789ABCDEF') */

ERASE&EDIT&BUFFER: PROCEDURE ;
CALL SETB(0,&EDIT&BUFFER,1280) ;
EDIT&BUFFER(64) = OFFH ;
ITEM*INDEX(0) = 64 ;
EDIT&FILE*NUMBER = 255 ;
ITEMS = 0 ;
END ;

GENERATE&ITEM*INDEXES: PROCEDURE ; /* SET UP ITEM*INDEX ARRAY */
CALL SETB(64,&ITEM*INDEX(128)) ;
DO ITEMS = 0 TO 127 ;
IF EDIT&BUFFER((ITEM*INDEX(ITEMS)+1) = 0 THEN /* END OF BUFFER */
RETURN ;
ITEM*INDEX(ITEMS+1) = ITEM*INDEX(ITEMS) + 1 ;
END ;
END ;

LOAD&JOB: PROCEDURE ;
DECLARE FILE BYTE, I WORD ;
IF JOB*SETUP THEN /* ILLEGAL REQUEST */
DO ;
JOB*BUFFER*INDEX = JOB*BUFFER*INDEX - 2 ;
CALL ILLEGAL&CMD ;
RETURN ;
END ;
FILE = 10*FILE*DECADE + (KEY AND OFH) ;
IF FILE*NUMBER () FILE THEN
DO ;
FILE*NUMBER = FILE ;
IF NOT READ&251(20,20*FILE,FILE*BUFFER) THEN
DO ;
FILE*NUMBER = 255 ;
RETURN ;
END ;
END ;
CALL WRITE(25('Job file '/0)) ;
CALL CO(ASCII(FILE*DECADE)) ;
CALL CO(KEY) ;
IF FILE&BUFFER(65) () 0 THEN
DO ;
END.
140   3       CALL WRITE(CHR(10));
141   3       CALL WRITE(CHR(FILE+BUFFER));
142   3       JOB+BUFFER+PTR = FILE+BUFFER;
143   3       JOB+BUFFER+INDEX = 64;
144   3       JOBARUN;
145   3       END;
146   2       ELSE
147   2       CALL WRITE(CHR(13));
148   1       CO: PROCEDURE (CHAR); */ OUTPUT ASCII CHARACTER TO TERMINAL */
149   2       DECLARE CHAR BYTE;
150   2       IF OUTPUTDISABLED THEN RETURN;
152   2       DO WHILE NOT TXRDY;
153   3       CALL PRINTERSERVICE;
154   3       END;
155   2       OUTPUT(USDATA) = CHAR;
156   2       CALL PRINTERBUFFER(CHAR);
157   2       END;

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

21
CALL WRITE(B'(\RUB)\00') ;
ELSE
   CALL CO(KEY) ;
END ;

BLANK: PROCEDURE(N); /* SEND SPACE CHARACTERS TO TERMINAL */
DECLARE (N,I) BYTE ;
DO I = 1 TO N ;
   CALL CO(SPACE) ;
END ;
END ;

DEEP: PROCEDURE; /* SEND (BEL) CHARACTER TO TERMINAL */
CALL CO(BEL) ;
END ;

PURGEJOBENTRY: PROCEDURE ;
IF JOB$SETUP THEN
   DO WHILE JOB$BUFFER(JOB$BUFFERINDEX)<>OFFH ;
      JOB$BUFFERINDEX = JOB$BUFFERINDEX - 1 ;
   END ;
   JOB$BUFFER(JOB$BUFFERINDEX+1) = 0 ;
END ;

WRITE: PROCEDURE (STRINGPTR) PUBLIC ; /* WRITE MESSAGE STRING TO TERMINAL */
DECLARE STRINGPTR POINTER STRING_BASED STRINGPTR (65535) BYTE, I WORD ;
DO I = 0 TO 65535 ;
   IF STRING(I) = 0 THEN RETURN ;
   CALL CO(STRING(I)) ;
END ;
END ;

DELETE: PROCEDURE ;
IF KEY = 0 THEN RETURN FALSE ;
RETURN TRUE ;
END ;

TOITERATE: PROCEDURE
   CALL WRITE(C('To terminate) mess(ESC).'tO)) ;
HEX1: PROCEDURE ; /* DISPLAY LOWER NIBBLE OF BYTE AS 1 HEX CHAR */
CALL CO(OEH) ;
CALL CO(OAH) ;
END ;
DECLARE VAL BYTE 
CALL CO(ASCII(VI)+D)
END

HEX2: PROCEDURE (VAL) PUBLIC /* DISPLAY BYTE AS 2 HEX CHAR */
DECLARE VAL BYTE 
CALL HEX1(SHR(VAL,8))
CALL HEX1(VAL)
END

HEX4: PROCEDURE (VAL) */ DISPLAY WORD AS 4 HEX CHAR */
DECLARE VAL WORD 
CALL HEX2(LO(VAL))
CALL HEX2(HI(VAL))
END

HEX8: PROCEDURE (VAL) */ DISPLAY DWORD AS 8 HEX CHAR */
DECLARE VAL DWORD 
CALL HEA2(LO(VAL))
CALL HEA2(LO(VAL))
END

HEX4PTR: PROCEDURE (VAL) /* DISPLAY POINTER AS 8 HEX CHAR */
DECLARE VAL POINTER; DM DWORD AT (VAL)
CALL HEA4(LO(VAL))
CALL HEA4(HI(VAL))
CALL HEA4(LO(IND))
CALL HEA4(HI(IND))
END

HEX4HR: LINE: PROCEDURE (PROMPT); /* START NEW MEMORY READ LINE */
DECLARE VAL BYTE 
CALL PRINTER$SMAPSHOT
CALL CR$LF
CALL HEA4(SEGMENT)
CALL CO('+$')
CALL HEA4(FROM$ADDR)
CALL BLANK(1)
CALL CO(PROMPT)
CALL BLANK(2)
END

SPACER: PROCEDURE;
CALL BLANK(1)
IF ( FROM$ADDR AND 03H ) = 0 THEN /* EVERY 4 BYTES ONE MORE */
CALL BLANK(1)
IF ( FROM$ADDR AND 07H ) = 0 THEN /* EVERY 8 BYTES ONE MORE */
CALL BLANK(1)
END

HEX10DISPLAY4LINE: PROCEDURE(PROMPT); /* START NEW I/O DISPLAY LINE */
DECLARE VAL BYTE 
CALL PRINTER$SMAPSHOT
CALL WRITE($CR$LF;$I/0 '$$01)
CALL HEA4(FROM$ADDR)
CALL BLANK(1)
CALL CO(PROMPT)
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 GOTSESC;
299 2 ELSE IF KEY=CNTL\# THEN  /* OPERATOR PAUSE */
300 2 DO FOREVER;
301 3 CALL PRINTERSERVICE;
302 3 IF RXRDY THEN
303 3 DO;
304 4 KEY = INPUT(USDATA);
305 4 CALL PRINTERSNAPSHOT;
306 4 IF KEY=CNTL\# THEN
307 4 GO TO GOTSESC;
308 4 IF KEY=CNTL\# THEN  /* PROCEED */
309 4 RETURN FALSE;
310 4 END;
311 3 END;
312 2 ELSE
313 2 GOTSESC:
314 2 IF JOB\RUN THEN
315 3 DO;
316 3 CALL WRITE(@CR@LF,'Job aborted',01);
317 3 JOB\IDLE;
318 3 END;
319 2 RETURN TRUE;
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:
330 3 KEY = JOB\BUFFER(JOB\BUFFER\INDEX);
331 3 JOB\BUFFER\INDEX = JOB\BUFFER\INDEX + 1;
332 3 IF KEY = OFFH THEN  /* END OF LINE - FETCH ANOTHER CHAR */
333 3 DO TO AGAIN;
334 3 IF KEY=NULL THEN  /* END OF BUFFER */
335 3 DO;
336 4 KEY = CR;
337 4 JOB\IDLE;
338 4 END;
339 3 RETURN TRUE;
340 2 ELSE  /* GET KEYBOARD INPUT IF WAITING */
341 3 IF NOT RXRDY THEN
342 3 RETURN FALSE;
343 3 KEY = INPUT(USDATA);
344 3 IF KEY() I0M THEN RETURN TRUE;
345 3 CALL PRINTERSNAPSHOT;
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#FROMADDR WORD ;
352 2    FROMADDR = (OLD#FROMADDR:=FROMADDR) * INCREMENT ;
353 2    IF FROMADDR < OLD#FROMADDR OR FROMADDR > TOADDR THEN
354 2    RETURN TRUE ;
355 2    IF NOT RRDY 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 JOBSSETUP THEN    /* PUT KEY IN BUFFER */
365 2    DO ;
366 3    IF KEY () CNTRL& 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    YESNO: PROCEDURE ;
374 2    CALL CI ;
375 2    DO WHILE KEY () ESC ;
376 3    IF FIND$(0('YnNo'),KEY,4) < 4 THEN
377 3    DO ;
378 4    CALL CI ;
379 4    CALL CAPITALIZE#KEY ;
380 4    RETURN ;
381 4    END ;
382 3    CALL DEEP ;
383 3    CALL CI ;
384 3    END ;
385 2    END ;
386 1    KEYSCASE: PROCEDURE (PTR) BYTE ;
387 2    DECLARE PTR POINTER; CHAR BASED PTR (0) BYTE; INDEX BYTE ;
388 2    DECLARE LENGTH LITERALLY 'CHAR(0)' ;
389 2    DO FOREVER ;
390 3    CALL CI ;
391 3    CALL CAPITALIZE#KEY ;
392 3    IF ( INDEX=FIND$(0(CHAR(1)),KEY,LENGTH ) ) = OFFH THEN
393 3    RETURN INDEX ;
394 3    ELSE
395 4    CALL DEEP ;
396 3    END ;
397 2    END ;
398 1    GET#ANOTHER#KEY: PROCEDURE BYTE;    /* WAIT UP TO 1 SECOND FOR KEYIN */
DECLARE I BYTE;

DO I = 1 TO 100;

IF KEY CAPTURED THEN

RETURN TRUE;

CALL TIME(100); /* 10 MicroSECOND PAUSE */

END;

RETURN FALSE; /* TIMEOUT */

END;

"RUBOUT" DELETES MOST RECENT ADDITION TO CHBUF */

BEEP IS SOUNDED IF CHBUF ALREADY EMPTY */

RUBOUT: PROCEDURE;

IF NCHAR = 0 THEN

CALL BEEP;

ELSE

DO;

NCHAR = NCHAR - 1;

CHBUF(NCHAR) = 0;

IF JOB SETUP THEN

JOB BUFFER INDEX = JOB BUFFER INDEX - 1;

CALL WRITE(#(5, SPACE, BS, 0));

END;

END;

ERASE: PROCEDURE; /* EMPTY OUT CHBUF */

DO WHILE NCHAR > 0;

CALL RUBOUT;

END;

END;

FLUSH: PROCEDURE; /* PURGE CHBUF OF FAULTY INPUT */

CALL BEEP;

IF NCHAR = 0 THEN

RETURN;

KEY = 0;

DO WHILE KEY < ESC AND KEY < RUB;

CALL CI;

IF JOB SETUP THEN

JOB BUFFER INDEX = JOB BUFFER INDEX - 1;

END;

END;

CALL ERASE;

END;

CAPITALIZE KEY: PROCEDURE;

IF KEY < = 'a' AND KEY < = 'z' THEN

KEY = KEY - 20H;

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.

KEYSIN: PROCEDURE BYTE; /* GET OPERATOR ENTRY */

CALL SETB(0, CHBUF, 64);
```plaintext
440 2 NCHAR = 0 ;
441 2 DO WHILE NCHAR < 64 ;
442 3 CALL CI ;
443 3 IF EXITSETUP THEN
444 3 RETURN FALSE ;
445 3 IF KEY = ESC THEN
446 4 DO ;
447 4 IF NCHAR = 0 THEN
448 4 RETURN FALSE ;
449 4 IF JOBSETUP THEN
450 4 JOBBUFFERINDEX = JOBBUFFERINDEX - 1 ;
451 4 CALL ERASE ;
452 4 END ;
453 3 ELSE IF CHORWLF THEN
454 3 RETURN TRUE ;
455 3 ELSE IF (KEY = RUB OR KEY = DS) THEN
456 3 DO ;
457 3 IF JOBSETUP THEN
458 4 JOBBUFFERINDEX = JOBBUFFERINDEX - 1 ;
459 4 CALL RUBOUT ;
460 4 END ;
461 3 ELSE
462 4 CALL CB(KEY) ;
463 4 KEY = KEY AND 7FH ;
464 4 CHUBF(NCHAR) = KEY ;
465 4 NCHAR = NCHAR + 1 ;
466 4 END ;
467 3 END ;
468 2 CALL WRITE(B(DM, ' overflow!',0)) ;
469 2 RETURN FALSE ;
470 2 END ;

471 1 HEX$KEY$VALUE: PROCEDURE (CHAR) BYTE ;
472 2 DECLARE CHAR BYTE ;
473 2 RETURN LOW(FINDB(ASC$|CHAR; 16)) ;
474 2 END ;

475 1 LEGAL$DEC: PROCEDURE (CHAR) BYTE ;
476 2 DECLARE CHAR BYTE ;
477 2 IF FINDB(ASC$|CHAR; 10) ( 10 THEN RETURN TRUE ;
478 2 RETURN FALSE ;
480 2 END ;

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

IF CHBUF(1) = 'a' AND CHBUF(1) = 'z' THEN /* MUST CAPITALIZE */
    CHBUF(1) = CHBUF(1) - 20H;
END;

IF NCHAR >= MIN AND NCHAR <= MAX THEN
    DO;
        HEX#WORD = 0;
        IF NCHAR = 0 THEN
            DO I = 0 TO NCHAR - 1;
                IF HEX#VALUE(CHBUF(I)) IS THEN
                    GO TO BADKEY;
                HEX#WORD = 16#HEX#WORD + DOUBLE(DOUBLE(HEX#VALUE(CHBUF(I))))
            END;
        RETURN TRUE;
    END;
    RETURN TRUE;
END;

BADKEY:
    CALL FLUSH;
    IF JOBSUP THEN /* NEED TO PURGE (cr) */
        JOB#BUFFER$INDEX = JOB#BUFFER$INDEX - 1;
    END;
    RETURN TRUE;
END;

LEGAL#MEMORY#ADDRESS: PROCEDURE (ADDR) BYTE;
DECLARE ADDR WORD, ADDR#WORD;
ABS = SML(DOUBLE(SEGMENT),4) + DOUBLE(ADDR);
IF ADDR1 IFFH AND ABS < OFFFFH THEN RETURN TRUE;
CALL WRITE(@('CR',LF)'Restricts on addresses : /O))
IF ABS < 2000H THEN
    CALL WRITE(@('0000H thru IFFFH',0))
ELSE
    CALL WRITE(@('0000H thru OFFFFH',0))
RETURN TRUE;
END;

MEM#READ: PROCEDURE (ADDRPTR) BYTE; /* GET MEMORY ADDRESS */
DECLARE ADDRPTR POINTER, ADDR#BASED ADDR#PTR#WORD;
CALL WRITE(@('Memory addr ',0))
IF NOT GETHEX(1,4) THEN
    RETURN FALSE;
IF NOT LEGAL#MEMORY#ADDRESS(HEX#WORD) THEN
    RETURN FALSE;
ADDR = HEX#WORD;
RETURN TRUE;
END;

FROM#TO: PROCEDURE BYTE; /* GET OPERATOR "FROM" & "TO" ENTRIES */
CALL WRITE(@('From',0))
IF NOT MEM#READ(0FROM#ADDR) THEN
    RETURN FALSE;
CALL WRITE(@('To',0))
IF NOT MEM#READ(0TO#ADDR) THEN
    RETURN FALSE;
RETURN TRUE;
END;

LEGAL#IOPORT: PROCEDURE (PORT) BYTE;
DECLARE PORT#WORD;
538 2 IF PORT ( BOH OR PORT ) ODFH THEN RETURN TRUE ;
540 2 CALL WRITE (8'(CR\LF)\BEL,'Restricted I/O ports: 000H thru 0DFH\0)) ;
541 2 RETURN FALSE ;
542 2 END ;
543 1 LEGAL I/O FROM TO: PROCEDURE BYTE ;
544 2 IF FROM ADDR ( BOH AND TO ADDR ) ODFH THEN
545 2 RETURN LEGAL I/O PORT ( BOH ) ;
546 2 RETURN TRUE ;
547 2 END ;
548 1 I/O PORT: PROCEDURE (RESULT PTR) BYTE ; /* GET I/O ADDRESS */
549 2 DECLARE RESULT PTR POINTER; RESULT BASED RESULT PTR WORD ;
550 2 CALL WRITE (8'(\port\0)) ;
551 2 IF NOT GETHEX (1,4) THEN
552 2 RETURN FALSE ;
553 2 IF NOT LEGAL I/O PORT (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 (PI T2 ) = 10110110B ; /* CTR 2; 2 BYTES; MODE 3; BINARY */
561 2 OUTPUT (PI T2 ) = LOW (COUNT) ;
562 2 OUTPUT (PI T2 ) = 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 (US FLAG) = 0 ;
569 2 CALL TIME (10) ;
570 2 OUTPUT (US FLAG) = 0 ;
571 2 CALL TIME (10) ;
572 2 OUTPUT (US FLAG) = 0 ;
573 2 CALL TIME (10) ;
574 2 OUTPUT (US FLAG) = 0 ;
575 2 CALL TIME (10) ;
576 2 OUTPUT (US FLAG) = 40H ; /* RESET */
577 2 CALL TIME (10) ;
578 2 OUTPUT (US FLAG) = MODE ; /* MODE INSTRUCTION */
579 2 CALL TIME (10) ;
580 2 OUTPUT (US FLAG) = 35H ; /* RTS, ER, RIE, DBR, TXEN */
581 2 CALL TIME (10) ;
582 2 KEY = INPUT (USDATA) ; /* THROWAWAY PENDING KEYIN, IF ANY */
583 2 END ;
584 1 MOMLOK: PROCEDURE ;
585 2 DECLARE BAUDRATE PTR POINT ; I BYTE ;
586 2 BEGIN ;
587 2 OUTPUT (PP IF LS) = 0AH ; /* SET UP PARALLEL PORT FOR LED */
588 2 OUTPUT (PI T2) = 01110110B ; /* CTR 1; 2 BYTES; MODE 3; BINARY */
589 2 OUTPUT (PI T1) = OFFH ; /* SET UP TIMER 1 FOR LED FLASH */
CALL LOADSTIMERS2(4) ; /* 19.2 K BAUD */
CALL USARTSINIT(01011110B); /* 1 STOP BIT, 8 CHAR BITS, X16 */
DO WHILE NOT KEYCAPTURED ;
OUTPUT(PITMOD) = 40H ; /* CTR 1 LATCH */
I = INPUT(PIT1) ; /* THROW AWAY LSB */
/* FLASH LED USING MOST SIGNIFICANT BIT OF TIMER 1 */
OUTPUT(PPIC) = ( SHR(INPUT(PIT1),4) AND 0BH ) OR 01H ;
END ;
OUTPUT(PPIC) = 01H ; /* TURN ON LED */
IF (KEY AND 7FH) = 'U' THEN
DO ;
BAUDRATESPTR = B('19200',0) ;
GO TO EUREKA ;
END ;
IF KEY = 66H THEN
DO ;
CALL LOADSTIMERS2(8) ;
BAUDRATESPTR = B('9600',0) ;
GO TO EUREKA ;
END ;
IF KEY = 78H THEN
DO ;
CALL LOADSTIMERS2(16) ;
BAUDRATESPTR = B('4800',0) ;
GO TO EUREKA ;
END ;
IF KEY = 80H THEN
DO ;
CALL LOADSTIMERS2(32) ;
BAUDRATESPTR = B('2400',0) ;
GO TO EUREKA ;
END ;
CALL LOADSTIMERS2(64) ; /* 1200 BAUD */
CALL USARTSINIT(01011110B); /* 1 STOP BIT, 8 CHAR BITS, X64 */
DO ;
IF (KEY AND 7FH) = 'U' THEN
DO ;
BAUDRATESPTR = B('1200',0) ;
GO TO EUREKA ;
END ;
IF KEY = 66H THEN
DO ;
CALL LOADSTIMERS2(32) ;
BAUDRATESPTR = B('600',0) ;
GO TO EUREKA ;
END ;
IF KEY = 78H THEN
DO ;
CALL LOADSTIMERS2(64) ;
BAUDRATESPTR = B('300',0) ;
GO TO EUREKA ;
END ;
IF KEY = 80H THEN
DO ;
CALL LOADSTIMERS2(128) ;
BAUDRATESPTR = B('150',0) ;
GO TO EUREKA;

CALL LOADTIMEMS #2(175); /* 110 BAUD */
CALL USARTINIT(1101011B); /* 2 STOP BITS, 7 CHAR BITS, X64 */
IF NOT GETANOTHERKEY THEN
    GO TO BEGIN;
ELSE GO TO BEGIN;

EUREKA:
    CALL USARTINIT(USER2INITBYTE AND OFH); /* FINAL SET-UP (7 BIT CHAR) */
    CALL WRITE(@('U',CR,LF,LF,'Baud rate = ',0));
    CALL WRITE(BAUDRATEPTR) /* PRINT BAUD RATE ON SCREEN */
    CALL CR,LF;
    END;

M4CMD: PROCEDURE; /* MEMORY COMMAND */
DECLARE SEG#SAVE, FROM#SAVE, TO#SAVE, TEMPWORD, (TYPE#SAVE), I BYTE I
IF JOBRUN THEN
    OUTPUTDISABLED = TRUE;
PTR.SEG = SEGMENT;
REPEAT:
    IF KEY = SPACE THEN
        DO;
        SEGMENT = SEG#SAVE;
        FROMADDR = FROM#SAVE;
        TOADDR = TO#SAVE;
        DATASTYPE = TYPE#SAVE;
        GO TO M4CMDO;
        END;
        CALL WRITE(@('Memories', 0));
        DO CASE KEYCASE(@6, 'RFSH', ESC));
        CALL WRITE(@('Read', 0));
        CMD#SAVE = 'M';
        GO TO REPEAT;
        END;
        GO TO M4CMD;
        GO TO M4CMD;
        GO TO M4CMD;
        GO TO M4CMD;
        END J
        M4ABORT:
        CALL PURGE@JOB@ENTRY;
        RETURN;

M4CMD:
    CALL WRITE(@('Read', 0));
    DO CASE KEYCASE(@6, 'BWP', ESC));
    CALL WRITE(@('Bytes', 0));
    CALL WRITE(@('Words', 0));
    CALL WRITE(@('Dwords', 0));
    CALL WRITE(@('Pointers', 0));
    CALL WRITE(@('ASCII', 0));
GO TO MABORT ;
END ;
IF DATASTYPE = 'A' THEN
DO ;
CALL WRITE('(', starting at',0));
IF NOT MEMADR(0FROMADDR) THEN
GO TO MABORT ;
END ;
ELSE IF NOT FROMTO THEN
GO TO MABORT ;
IF JOBABLE THEN
DO ;
SEGSAVE = SEGMENT ;
FROMSAVE = FROMADDR ;
TOSAVE = TOSADDR ;
TYPESAVE = DATASTYPE ;
CMDSAVE = 'M' ;
END ;
MRACMD1:
OUTPUTDISABLED = FALSE ;
CALL NEWSMSLINE('="');
MRACMD1:
IF DATASTYPE = 'A' THEN /* ASCII strings dump */
DO ;
CALL WRITE(NEMAPTR) ;
RETURN ;
END ;
I = 0 ;
DO FOREVER ;
IF DATASTYPE = 'B' THEN
DO ;
BYTESSAVE = BYTEDATA ; /* GET MEMORY BYTE */
IF BTO THEN RETURN ;
CALL HEX2(BYTESSAVE) ; /* DISPLAY IT */
INCREMENT = 1 ;
END ;
ELSE IF DATASTYPE = 'M' THEN
DO ;
WORDSSAVE = WORDDATA ; /* GET MEMORY WORD */
IF BTO THEN RETURN ;
CALL HEX4(WORDSSAVE) ; /* DISPLAY IT */
INCREMENT = 2 ;
END ;
IF DATASTYPE = 'D' THEN
DO ;
DWORDSSAVE = DWORDDATA ; /* GET MEMORY DWORD */
IF BTO THEN RETURN ;
CALL HEX8(DWORDSSAVE) ; /* DISPLAY IT */
INCREMENT = 4 ;
END ;
IF DATASTYPE = 'P' THEN
DO ;
POINTERSSAVE = POINTERDATA ; /* GET MEMORY POINTER */
IF BTO THEN RETURN ;
CALL HEXPTR(POINTER$SAVE) ; /* DISPLAY IT */
INCREMENT = 4 ;
END ;
IF COMPLETE OR NOT LEGAL MEMORY ADDRESS (FROM ADDR) THEN
RETURN ;
I = I + INCREMENT ;
IF (I AND 0FH) = 0 THEN ; /* END OF LINE */
CALL MEM$PRINT('=' ) ;
ELSE
CALL SPACER ;
END ;

NFACMD: ; /* FILL BLOCK OF MEMORY */
OUTPUT$DISABLED = FALSE ;
IF JOB$RUN THEN
CALL WRITE($('CR$LF$Memory ',0)) ;
CALL WRITE($('Fill with ',0)) ;
DO CASE KEY$CASE($('5$BOX$ESC') ) ;
CALL WRITE($('Byte ',0)) ;
CALL WRITE($('Word ',0)) ;
CALL WRITE($('Dword ',0)) ;
CALL WRITE($('ASCII ',0)) ;
GO TO MABORT ;
END ;
DATA$TYPE = KEY ;
IF DATA$TYPE = 'A' THEN GO TO NFACMD ;
CALL WRITE($('value ',0)) ;
IF NOT SETHEX(1,0) THEN
GO TO MABORT ;
FILL$WORD = HEX$WORD ;
IF NOT FROM$ADT THEN
GO TO MABORT ;
/* FILL LOOP */
DO FOREVER ;
IF DATA$TYPE = 'B' THEN
DO ;
BYTE$DATA = FILL$BYTE ;
INCREMENT = 1 ;
END ;
ELSE IF DATA$TYPE = 'W' THEN
DO ;
WORD$DATA = FILL$WORD ;
INCREMENT = 2 ;
END ;
ELSE
DO ;
DWORD$DATA = FILL$DWORD ;
INCREMENT = 4 ;
END ;
IF BTO OR COMPLETE OR NOT LEGAL MEMORY ADDRESS (FROM ADDR) THEN
RETURN ;
END ;
NFACMD: ; /* MEMORY FILL WITH ASCII STRING */
IF JOB$RUN THEN
OUTPUT$DISABLED = TRUE ;
CALL WRITE($('string starting at ',0)) ;
003 2 IF NOT MEMADR(@FROMADDR) THEN
004 2 GO TO MMABORT;
005 2 CALL WRITE(@CR+LF,
006 2 'String is written to memory until (esc) encountered',@CR+LF,0));
007 2 OUTPUT*DISABLED = FALSE;
008 2 CALL HEX4(SEGMENT);
009 2 CALL CO("=");
010 2 CALL HEX4(FROMADDR);
011 2 CALL WRITE(@CR+LF,0));
012 2 /* SUBSTITUTION LOOP */
013 2 DO FOREVER;
014 2 CALL CI;
015 2 IF KEY=NULL OR KEY=ESC OR FROMADDR=0FFFFH THEN
016 2 DO;
017 2 BYTEDATA = 0; /* DELIMITER */
018 2 RETURN;
019 2 END;
020 2 BYTEDATA = KEY;
021 2 FROMADDR = FROMADDR + 1;
022 2 CALL SHOW#KEY;
023 2 END;
024 2 /* MEMORY SUBSTITUTE ROUTINE */
025 2 CALL WRITE(@CR+LF); /* To alter location, enter 1 or 2 hex characters followed by (CR).'
026 2 CALL WRITE(@CR+LF,0));
027 2 CALL TOSTATEMINATE;
028 2 OUTPUT*DISABLED = FALSE;
029 2 /* SUBSTITUTION LOOP */
030 2 DO FOREVER;
031 2 CALL CR+LF;
032 2 DO I = 1 TO 4;
033 2 CALL HEX4(SEGMENT);
034 2 CALL CO("=");
035 2 CALL HEX4(FROMADDR);
036 2 CALL CO("=");
037 2 BYTESAVE = BYTESDATA; /* GET MEMORY BYTE */
038 2 IF BTO THEN
039 2 RETURN;
040 2 CALL HEX2(BYESAVE); /* DISPLAY IT */
041 2 CALL CO("="); /* PROMPT FOR SUBSTITUTION */
042 2 IF NOT GETHEX(0,2) THEN
043 2 RETURN; /* ESC MEANS EXIT */
044 2 IF NCHAR > 0 THEN
045 2 BYTEDATA = HEX$BYTE; /* WRITE NEW BYTE TO MEMORY */
046 2 FROMADDR = FROMADDR + 1;
047 2 IF NOT LEGAL$MEMORY$ADDRESS(FROMADDR) THEN
048 2 RETURN;
049 2 CALL BLANK(3);
050 2 END;
051 2 /* WRITE INTO SPECIFIED MEMORY LOCATION */
052 2
CALL WRITE(@('Write ','0)) ;
852 2
DO CASE KEY$CASE(0('3','BH','ESC')) ;
853 3
CALL WRITE(@('Bytes','0)) ;
854 3
CALL WRITE(@('Words','0)) ;
855 3
GO TO #ABORT ;
856 3
END ;
857 2
DATA$TYPE = KEY ;
858 2
CALL WRITE(@('inta','0)) ;
859 2
IF NOT NOTAB($FROM$ADDR) THEN
860 2
GO TO #ABORT ;
861 2
CALL CR LF ;
862 2
CALL TOSTERMINATE ;
863 2
OUTPUT$DISABLED = FALSE ;
864 2
CALL MEM$M$HLINE('@') ;
865 2
DO FOREVER ;
866 3
IF NOT GET$HEX(1,4) THEN RETURN ;
868 3
IF DATA$TYPE = 'W' THEN WORD$DATA = HEX$WORD ;
870 3
ELSE BYTE$DATA = HEX$BYTE ;
871 3
IF DTO THEN RETURN ;
873 3
CALL BLANK(1) ;
874 3
END ;
875 2
END ;
876 1
IACHD: PROCEDURE; /* I/O COMMAND */
877 2
DECLARE (FROM$SAVE,TO$SAVE) WORD, (TYPE$SAVE,TEMP,1) BYTE ;
878 2
IF JON$RUN THEN
879 2
OUTPUT$DISABLED = TRUE ;
880 2
REPEAT:
881 2
IF KEY = SPACE THEN
882 3
DO ;
883 3
FROM$ADDR = FROM$SAVE ;
884 3
TO$ADDR = TO$SAVE ;
885 3
DATA$TYPE = TYPE$SAVE ;
886 3
GO TO IACHD ;
887 3
END ;
888 2
CALL WRITE(@('I/O ','0)) ;
889 2
DO CASE KEY$CASE(0('5',' RWF','ESC')) ;
890 3
DO ;
891 4
CALL WRITE(@('Read ','0)) ;
892 4
CH$SAVE = 'I' ;
893 4
GO TO REPEAT ;
894 4
END ;
895 3
GO TO IACHD ;
896 3
GO TO JON$CHAR ;
897 3
GO TO #ABORT ;
898 3
END ;
899 2
#ABORT:
CALL PURGE$JOBENTRY ;
900 2
RETURN ;
901 2
IACHD: /* DISPLAY GROUP OF I/O PORTS */
902 2
CALL WRITE(@('Read ','0)) ;
903 2
DO CASE KEY$CASE(0('3','BH','ESC')) ;


903  CALL WRITE(('Bytes',0)) ;
904  CALL WRITE(('Words',0)) ;
905  GO TO IABORT ;
906  END ;
907  DATA$TYPE = KEY ;
908  CALL WRITE(('From',0)) ;
909  IF NOT IOPORT(0FROMADDR) THEN
910  GO TO IABORT ;
911  CALL WRITE(('to',0)) ;
912  IF NOT IOPORT(0TOADDR) OR NOT LEGAlSIOP FROM TO THEN
913  GO TO IABORT ;
914  IF JOBSIBLE THEN
915  DO ;
916  FROM$SAVE = FROM$ADDR ;
917  TO$SAVE = TO$ADDR ;
918  TYPE$SAVE = DATA$TYPE ;
919  CMD$SAVE = 'I' ;
920  END ;
921  IMP$CMDO:
922  OUTPUT$DISABLED = FALSE ;
923  CALL NEW$IDODISPLAY$LINE('=',) ;
924  I = 0 ;
925  DO FOREVER ;
926  IF DATA$TYPE = 'B' THEN
927  BYTE$SAVE = INPUT(0FROM$ADDR) ; /* GET I/O PORT BYTE VALUE */
928  IF BYTE$SAVE THEN RETURN ;
929  CALL HEX2(BYTE$SAVE) ; /* DISPLAY IT */
930  INCREMENT = 1 ;
931  END ;
932  ELSE IF DATA$TYPE = 'W' THEN
933  DO ;
934  WORD$SAVE = INPUT(0FROM$ADDR) ; /* GET I/O PORT WORD VALUE */
935  IF WORD$SAVE THEN RETURN ;
936  CALL HEX4(WORD$SAVE) ; /* DISPLAY IT */
937  INCREMENT = 2 ;
938  END ;
939  IF COMPLETE OR NOT LEGAlSIOPORT(0FROM$ADDR) THEN
940  RETURN ;
941  I = I + INCREMENT ;
942  IF (~I AND (OFH)) = 0 THEN /* END OF LINE */
943  CALL NEW$IDODISPLAY$LINE('=',) ;
944  ELSE
945  CALL SPACER ;
946  END ;
947  END ;
948  IN$CMDO: /* I/O PORT WRITE ROUTINE */
949  CALL WRITE(('Write ',0)) ;
950  DO CASE KEY$CASE(3,3'M',3ESC)) ;
951  CALL WRITE(('Bytes',0)) ;
952  CALL WRITE(('Words',0)) ;
953  GO TO IABORT ;
954  DATA$TYPE = KEY ;
955  CALL WRITE(('to',0)) ;
956  IF NOT IOPORT(0FROM_ADDR) THEN
GO TO IAABORT;

CALL CRLF;

CALL TO TERMINATE;

IF NOT GETHEX(1,4) THEN RETURN;

IF DATA TYPE = 'W' THEN OUTWORD(FROM ADDR) = HEX#WORD;

ELSE IF TO THEN RETURN;

CALL BLANK(1);

END

IF CMBD;

/* FILL BLOCK OF PORTS */

OUTPUT#DISABLED = FALSE;

IF JOB#RUN THEN

CALL WRITE(0(CR;LF;/I/O 'I':0))

CALL WRITE(0("Fill with ',0))

DO CASE KEY#CASE(G3/*BN*/ESC))

CALL WRITE(0("Byte '/0))

CALL WRITE(0("Word '/0))

GO TO IAABORT;

END

DATA TYPE = KEY;

CALL WRITE(0("Value '/0))

IF NOT GETHEX(1,4) THEN

GO TO IAABORT;

FILL#WORD = HEX#WORD;

CALL WRITE(0("From'/0))

IF NOT IMPORT(FROM ADDR) THEN

GO TO IAABORT;

CALL WRITE(0("To'/0))

IF NOT IMPORT(ATO ADDR) OR NOT LEGAL#ID#FROM#TO THEN

GO TO IAABORT;

/* FILL LOOP */

DO FOREVER;

IF DATA TYPE = 'D' THEN

DO;

OUTPUT(FROM ADDR) = FILL#BYTE;

INCREMENT = 1;

END;

ELSE

DO;

INDEXIWORD(FROM ADDR) = FILL#WORD;

INCREMENT = 2;

END;

IF RTO OR COMPLETE THEN

RETURN;

END;

END

HACMD: PROCEDURE; /* HELP DISPLAY */

IF JOB#SETUP THEN /* REMOVE "H" CMD FROM JOB BUFFER */

JOB#BUFFER#INDEX = JOB#BUFFER#INDEX - 2;

CALL WRITE(0;
PL/M-86 COMPILER NASA/ABRF XAIDS MAINTENANCE PROCESSOR PROGRAM 14 JAN 1987

HEX: /CR/LF;
'(u) = Job load & run. u = 0 thru 9, selects file = 10^d + u.'/CR/LF;
'(d) = Job decade. d = 0 thru 9, selects decade.'/CR/LF;
'JE = Job edit. Views/modify/load/stores job buffer.'/CR/LF;
'JD = Job directors. Shows titles for selected job decade.'/CR/LF;
'ISP = 1/0 read rerun. Repeat current "IR" setup.'/CR/LF;
'MSP = Memory read rerun. Repeat current "MR" setup.'/CR/LF;
'(SP) = Job rerun. Repeat current job buffer.'/CR/LF;
'(SP) = Space bar. Repeats last rerun (ISp, MSp, or JSp).'/CR/LF;
'7 = B or N B (stale) or N(word),'/CR/LF;
'110 = 1/0 read. Sets up byte or word I/O port display.'/CR/LF;
'110 = I/O write. Repetitive byte or word outputs to selected port.'/CR/LF;
'110 = 1/0 fill. Writes byte or word to a block of ports.'/CR/LF;
'MMW = Memory write. Repetitive byte or word memory writes.'CR/LF;
'R = D, H, D, P, or ASCII (data), N(word), D(word), P(input), or ASCII dump.'CR/LF;
'MR = Memory read. Sets up byte, word, dword, pointer, or ASCII dump.'/CR/LF;
'110 = D, H, D, or A B (stale), N(word), D(word), or ASCII dump.'CR/LF;
'MFB = Memory fill. D, word, dword, or ASCII string fill.'CR/LF;
'MS = Memory substitute. Byte scan/modify.'CR/LF;
'R = Rupts. Displays multitask interrupt counters.'CR/LF;
'S = Segment. Reloads memory base register.'CR/LF;
0));

1010 2 END;

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

38
1041 3  JOB#RUN;
1042 3  RETURN;
1043 3  END;
1044 2  CALL WRITE(@(Job 's,0));
1045 2  IF KEY = 'E' THEN
1046 2  GO TO JEOCHMD;
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  JEOCHMD;
1054 2  IF EXIT#SETUP OR (KEY = 'E') THEN /* JOB BUFFER EDITOR */
1055 3  DO;
1056 3  JOB#IDLE;
1057 3  CALL WRITE(@(Editor 's,0));
1058 3  SHOW#ITEMS;
1059 3  CALL WRITE(@(CR,LF,File = 's,0));
1060 3  IF EDIT#FILE#NUMBER < 1 THEN
1061 4  CALL CI(ASCII(Edit#FILE#NUMBER/10));
1062 4  CALL CI(ASCII(Edit#FILE#NUMBER MOD 10));
1063 4  END;
1064 3  ELSE
1065 3  CALL WRITE(@(??,0));
1066 3  CALL WRITE(@(CR,LF,Title = 's,0));
1067 3  CALL GENERATE#ITEM#INDEXES;
1068 3  IF NITEMS = 0 THEN
1069 3  CALL WRITE(@(CR,LF, '01 =',0));
1070 3  ELSE
1071 4  DO I = 1 TO NITEMS;
1072 5  CALL WRITE(@(CR,LF, 's,0));
1073 5  CALL HEX2(I);
1074 5  CALL WRITE(@( 's,0));
1075 5  INDEX = ITEM#INDEX(I-1) + 1;
1076 5  KEY = EDIT#BUFFER(INDEX);
1077 6  DO WHILE KEY <> STRING;
1078 6  CALL SHOW#KEY;
1079 6  CALL BLANK(1);
1080 6  INDEX = INDEX + 1;
1081 6  KEY = EDIT#BUFFER(INDEX);
1082 6  END;
1083 4  END;
1084 3  SHOW#MENU:
1085 3  CALL WRITE(@(CR,LF, 'Edit menu: A(ppend) D(elete) E(merge) L(oad) S(ave) T(itle) (esc)=Exit 's,0));
1086 4  CALL CI;
1087 4  CALL CAPITALIZE#KEY;
1088 4  IF KEY=ESC THEN
1089 4  DO I
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 <crtl 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$MENU ;
1111 6      END ;
1112 5      CALL MOV$(EDIT$BUFFER(ITEM$INDEX(HEX$BYTE)),
1113 5      EDIT$BUFFER(ITEM$INDEX(HEX$BYTE-1)),
1114 5      1280-ITEM$INDEX(HEX$BYTE) ) ;
1115 5      EDIT$FILE$NUMBER = 255 ;
1116 5      GO TO SHOW$ITEMS ;
1117 5      END ;
1118 4      ELSE IF KEY = 'E' THEN
1119 4      DO;
1120 5      CALL WRITE@(CR,LF,'Erase entire buffer ? ',0)) ;
1121 5      CALL YES$NO ;
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 OFH) ;
1135 6      IF HEX$BYTE < 100 THEN
1136 6      DO;
1137 7      EDIT$FILE$NUMBER = HEX$BYTE ;
1138 7      IF NOT READ$251(CR,20*EDIT$FILE$NUMBER,EDIT$BUFFER) THEN
1139 7      CALL ERASE@EDIT$BUFFER ;
1140 7      END ;
1141 6      ELSE
1142 6      CALL WRITE@(BEL,' out of range',0)) ;
1143 5      GO TO SHOW$ITEMS ;
1144 5      END ;
1145 4      ELSE IF KEY = 'S' THEN
DO
CALL WRITE(B(CR;LF,'Save as job number (0-99 decimal) ',0))
IF BETH:6(1,0) THEN
BYTESSAVE = 10 * SHR(HEXBYTE;4) ; /* CONVERT TO DECIMAL */
HEXBYTE = BYTESAVE + (HEXBYTE AND OFH) ;
IF HEXBYTE < 100 THEN
DO;
EDITFILENUMBER = HEXBYTE ;
CALL WRITE$25I(EDITFILENUMBER) ;
END;
ELSE
CALL WRITE(B(DEL,' out of range',0))
END;
GO TO SHOWMENU
END;
ELSE IF KEY = '7' THEN
DO
CALL WRITE(B(CR;LF,'Enter title (63 char max)',CR;LF,' ',0))
IF KEYSIN THEN
DO;
EDITFILE$251(EDITFILENUMBER)
END;
GO TO SHOWITEMS
END;
ELSE CALL DEEP
END;
ELSE IF KEY = '0' AND KEY = '9' THEN /* JOB DECADE SELECT */
DO;
CALL WRITE(B('decade ',0))
CALL CO(KEY)
CALL WRITE(B('u selected',0))
FILEDECADE = HEXBYTEVALUE(KEY)
CALL WRITE(B(CR;LF,'Job ',0))
KEY = 'y'
GO TO JENCHO ; /* SHOW DIRECTORY NEXT */
END;
ELSE IF KEY = '0' THEN /* JOB DIRECTORY */
DO;
CALL WRITE(B('Directories for decade ',0))
CALL CO(ASCII(FILEDECADE))
DO I = 0 TO 9 ;
CALL WRITE(B(CR;LF, ' ',0))
CALL CO(ASCII(FILEDECADE))
CALL CO(ASCII(I))
CALL WRITE(B(' ',0))
IF RED$25I(1,200+FILEDECADE+201+@BLOCK) THEN
CALL WRITE(@BLOCK)
ELSE
RETURN
END;
END;
ELSE CALL ILLEGALCHO
END;
END;
END;
END;
END;
END;
RCHD: PROCEDURE ;  /* DISPLAY MULTIBUS INTERRUPT COUNTERS */
DECLARE COUNTER#SNAP WORD, I BYTE ;
IF NOT JOB#RUN THEN
CALL WRITE(0('Run activity on Multibus',0)) ;
CALL CRALF ;
DO I = 0 TO 7 ;
CALL WRITE(0('INT',0)) ;
CALL HEX(I) ;
CALL CO(':' ) ;
DISABLE ;
COUNTER#SNAP = INT$COUNTER(I) ;
ENABLE ;
CALL HEX(HIG(HCUTER#SNAP)) ;
CALL HEX(LOW(COUNTER#SNAP)) ;
IF I = 3 THEN
CALL CRALF ;
ELSE
CALL BLANK(5) ;
END ;
END ;
BACKUP: PROCEDURE ;
DECLARE (I,J,INDEX) WORD ;
CALL WRITE(0('Backup current files ? ',0)) ;
CALL YESNO ;
IF KEY = 'Y' THEN RETURN ;
CALL WRITE(0(' All decades ? ',0)) ;
CALL YESNO ;
IF KEY = 'Y' THEN
DO ;
CALL MOVE(ASCII#CHBUF;10) ;
NCHAR = 10 ;
END ;
ELSE IF KEY = 'N' THEN
DO ;
CALL WRITE(0(' Which decades ? ',0)) ;
IF NOT KEYSIN OR NCHAR = 0 THEN RETURN ;
END ;
ELSE RETURN ;
CALL SETN(0,BACKUP$RESTORE#BUFFER;32768) ;
INDEX = 0 ;
DO I = 0 TO NCHAR-1 ;
KEY = CHBUF(I) ;
IF LEGAL#DEC(ASC) THEN
DO J = 0 TO 9 ;
FILE#NUMBER = 10#HEX#VALUE(KEY) + J ;
IF NOT READ#251(20#DOUBLE(FILE#NUMBER),@FILE$BUFFER) THEN RETURN ;
END ;
IF FILE#BUFFER(65) <> 0 THEN
DO ;
CALL WRITE(0(CR#LF,' Job file ',0)) ;
CALL CO(ASCII(FILE#NUMBER / 10)) ;
CALL CO(ASCII(FILE#NUMBER MOD 10)) ;
CALL WRITE(0(' = ',0)) ;
CALL WRITE(@FILE#BUFFER) ;
DIR(FILE#NUMBER).INDEX = INDEX ;
DIR(FILENUMBER).LENGTH = 64 + FINDDBB(FILEBUFFER(64), 0, 1280-64);
CALL MOVBUFILEBUFFER(WINFO(INDEX), DIR(FILENUMBER).LENGTH);
INDEX = INDEX + DIR(FILENUMBER).LENGTH;
END;
END;

ORIGINAL PAGE IS OF POOR QUALITY
1302 1 CALL SETM(0,0,4094);      /* CLEAR RAM (64K 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 FILENUMBER = 255;
1307 1 CALL ERASEEDIT*BUFFER;
1308 1 CMDSAVE = 'M';          /* SET UP REPEAT FOR MRB */
1309 1 DATA*TYPE = 'B';
1310 1 RESTART:               /* INITIALIZE S8X251 BUBBLE MEMORY MULTIMODULE */
1311 1 CALL WRITE*(CR/\LF, 
1312               LF;
1313               XX  XX  AAA  IIIIII  DDDDDD  SSSSSS  'CR/\LF, 
1314               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1315               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1316               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1317               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1318               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1319               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1320               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1321               XX  XX  AAA  II  DD  DD  SS  SS  'CR/\LF, 
1322   )
1323 1 REPEAT:               /* DISPLAY MENU AGAIN */
1324               CALL SHOW*MENU;
1325 1 NEXT:                 /* PROCESS NEXT COMMAND */
1326 1 OUTPUT*DISABLED = FALSE;
1327 1 INCREMENT = 1;
1328 1 IF JOB*RUN THEN
1329 2 DO;
1330 3 IF JOB*BUFFER(JOB*BUFFER*INDEX) = OFFH THEN
1331 4 JOB*BUFFER*INDEX = JOB*BUFFER*INDEX + 1;
1332 5 IF JOB*BUFFER(JOB*BUFFER*INDEX) = 0 THEN
1333 6 JOB*IDLE;
1334 7 END;
1335 8 IF NOT JOB*RUN THEN
1336 9 CALL CR/\LF;
1337 10 CALL CRO LF;
1338 11 IF JOB*IDLE THEN;
1339 12 CALL CO('.');        /* NORMAL PROMPT CHAR */
1340 13 ELSE IF JOB*SETUP THEN
1341 14 DO;
1342 15 CALL CO('"');
1343 16 JOB*BUFFER(JOB*BUFFER*INDEX) = OFFH;
1344 17 JOB*BUFFER*INDEX = JOB*BUFFER*INDEX + 1;
1345 18 END;
1346 19 TRY*AGAIN:
DO CASE KEYCASE ("(2,"'0123456789';'ESC';'SPACE';'NHJRS';'CRTLAC';'CRTL0B';'CRTLOR;CR));

CASE0:
  CALL LOADJOB;
CASE1:
  CALL LOADJOB;
CASE2:
  CALL LOADJOB;
CASE3:
  CALL LOADJOB;
CASE4:
  CALL LOADJOB;
CASE5:
  CALL LOADJOB;
CASE6:
  CALL LOADJOB;
CASE7:
  CALL LOADJOB;
CASE8:
  CALL LOADJOB;
CASE9:
  CALL LOADJOB;
CASE10:
  /* ESC */
    DO;
      CALL PURGEJOBENTRY;
      CALL CRALF;
      CALL CRALF;
      GO TO RESTART;
      END;
CASE11:
  /* BLANK */
    DO;
      IF JOBSETUP THEN
        DO;
          CALL PURGEJOBENTRY;
          CALL WRITE("(BEL; ' <space> - illegal repeat in job mode:0)";
          END;
        ELSE IF CMDSAVE='N' THEN
          CALL MCHD;
        ELSE IF CMDSAVE='I' THEN
          CALL ICHD;
        ELSE IF CMDSAVE='J' THEN
          CALL JCHD;
        END;
      END;
CASE12:
  /* MEMORY COMMAND */
    CALL MCHD;
CASE13:
  /* HELP COMMAND */
    CALL HCHD;
CASE14:
  /* I/O COMMAND */
    CALL ICHD;
CASE15:
  /* JOB COMMAND */
    CALL JCHD;
CASE16:
  /* RSTART DISPLAY */
    CALL RCHD;
CASE17:
  /* CHANGE MEMORY SEGMENT */
    DO;
      IF JOBRUN THEN
        OUTPUTDISABLED = TRUE;
        CALL WRITE("("Current memory segment register : ";0));
      END;

CALL HEX4(SEGMENT)
CALL WRITE($OCR,LF,'Enter desired segment value (0 - FFFF) ','0'))
IF GETHEX(1,4) THEN
DO;
SEGMENT = HEX4WORD;
CALL WRITE($OCR,LF,'Updated memory segment register ','0'))
CALL HEX4(SEGMENT);
END;
CASE18: /\ CNTL C \/
DO;
IF JOB*SETUP THEN
CALL JACMD;
ELSE
GO TO ILLEGAL;
END;
CASE19: /\ CNTL B \/
DO;
IF JOB*IDLE THEN /\ OK TO DO BACKUP \/
CALL BACKUP;
ELSE
GO TO ILLEGAL;
END;
CASE20: /\ CNTL R \/
DO;
IF JOB*IDLE THEN /\ OK TO DO RESTORE \/
CALL RESTORE;
ELSE
GO TO ILLEGAL;
END;
CASE21: /\ CR \/
GO TO REPEAT;
END;

GO TO NEXT;

ILLEGAL:
CALL JACMD;
CALL PURGE*JOB*ENTRY;
GO TO TRYAGAIN;

END;

MODULE INFORMATION:

CODE AREA SIZE = 1C7BH 7291D
CONSTMNT AREA SIZE = 11FBH 4600D
VARIABLE AREA SIZE = 0BEOH 3040D
MAXIMUM STACK SIZE = 003AH 5BD
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
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.
LP/M-86 Compiler NASA/ADRF XAIDS Maint Printer Routines 14 Jan 1987

IRIX 86 PL/M-86 V2.7 Compilation of Module LP_P86
Object Module Placed In LPOBJ
Compiler Invoked By: 'LANG=PLM86 LP_P86

/* COMPACT RUN OPTIMIZE(3)
TITLE('NASA/ADRF 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 6A05 Board Programmable Peripheral Interface I/O Port Mapping */

11 1 DECLARE PPIA LITERALLY '0C8H'; /* PAO-PA7 8287 (INVERTING) */
12 1 DECLARE PPIB LITERALLY '0BAH'; /* PB0-PB7 902 (TERMINATOR) */
13 1 DECLARE PPI_C LITERALLY '0CH'; /* PCO-PC3 7408 (NON-INVERTING) */
14 1 DECLARE PPIFLG LITERALLY '0C7H'; /* PCA-PC7 902 (TERMINATOR) */

/* I/O STATUS FLAGS */

15 1 DECLARE PTRSBUSY LITERALLY 'SHR(INPUT(PPIA),7)';
16 1 DECLARE PTRSSELECT LITERALLY 'SHR(INPUT(PPIA),4)';

/* MISCELLANEOUS DECLARATIONS */

17 1 DECLARE TRUE LITERALLY 'OFFH';
18 1 DECLARE FALSE LITERALLY '01H';
19 1 DECLARE FOREVER LITERALLY 'WHILE TRUE';
20 1 DECLARE KEY BYTE EXTERNAL;
21 1 DECLARE CRTSLINE (24) STRUCTURE (CHAR (80) BYTE);
22 1 DECLARE PTRFLG (24) STRUCTURE (CHAR (80) BYTE);
23 1 DECLARE RemmCol,FF$FLG BYTE;
24 1 DECLARE PTRFLG BYTE;

/* PTRFLG possible states: */
/* 0 RESET, DESELECTED, OR OFF */
/* 1 INLR 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; /* SET 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 INITIALIZEPRINTER: PROCEEDURE PUBLIC;
34 2 OUTPUT(PTRFLG) = 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(7FH,0CRTLINE(1920));
37 2 ROW = 1;
38 2 COL = 1;
39 2 PTRFLG = 0;
40 2 END;

41 1 PRINTERSNAPSHOT: 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 CRTLINE(ROW-1),CHAR(COL-1) = CHAR;
47 3 COL = COL + 1;
48 3 IF COL = 0 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;
53 4 CALL SETB(7FH,0CRTLINE(ROW-1),80);
54 4 END;
55 4 END;
56 3 END;
57 2 ELSE IF CHAR = 0BH THEN    /* BS */
58 2 DO;
59 3 IF COL = 1 THEN COL = COL - 1;
60 3 END;
61 2 ELSE IF CHAR = 0AH THEN    /* LF */
62 2 DO;
63 3 ROW = ROW + 1;
64 3 IF ROW = 24 THEN ROW = 1;
65 3 CALL SETB(0FH,0CRTLINE(ROW-1),80);
66 3 END;
67 2 ELSE IF CHAR = 0DH THEN    /* VT */
68 2 DO;
69 3 IF ROW = 1 THEN ROW = ROW - 1;
70 3 END;
71 2 ELSE IF CHAR = 09H THEN    /* CR */
72 2 DO;
73 3 COL = 1;
74 3 END;
75 2 END;

79 1 PRINTERSNAPSHOT: 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;
86 4 DO I = 0 TO 23;
87 5 IF J = 25 THEN J = 1;
88 5 CALL MOVB(0CRTLINE(J-1),&PTRFLINE(I));
89 5
PL/N-66 Compiler
NASA/ARDF XADS MAINT PRINTER Routines 14 Jan 1987
01/14/87 09:49:35 Page 3

ORIGINAL PAGE IS OF POOR QUALITY.

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:
104 4    FF$FLG = TRUE;
105 4    IF PTR$SELECT THEN PTRFLG = 1;
107 4    END;
108 3    IDLE:
109 4    RETURN;
110 4    END;
111 3    FORM FEED:
112 4    DO;
113 4    IF FF$FLG THEN /* NEED FF EVERY OTHER PASS */
114 5    DO;
115 5    CALL STROBE(OCH);
116 5    FF$FLG = FALSE;
117 4    ELSE
118 4    FF$FLG = TRUE;
119 4    INDEX = 0;
120 4    PTRFLG = 3; /* LINE FEEDS */
121 3    LINE FEEDS:
122 4    DO;
123 4    IF INDEX < 6 THEN CALL STROBE(OAH);
124 4    ELSE CALL STROBE(OBH);
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:
134 4    DO;
135 4    CALL STROBE(' ');
136 4    INDEX = INDEX + 1;
137 4    IF INDEX > 16 THEN
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(PTRFLGLINE(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#FLF:
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 /# IN#LE #/
163 5
PTRFLG = 1 ;
164 5
ELSE /# INDENT #/
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 08
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 (3X)
0KB DISK SPACE USED

END OF PL/M-86 COMPILATION

52
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTSCOUNTER</td>
<td>an array of eight DWORD counters which tally the interrupts from the Multibus.</td>
</tr>
<tr>
<td>NMISCOUNTER</td>
<td>a DWORD counter which tallies the number of bus timeout interrupts.</td>
</tr>
<tr>
<td>UNKNOWNSCOUNTER</td>
<td>a DWORD counter which tallies extraneous interrupts.</td>
</tr>
<tr>
<td>BTOSFLAG</td>
<td>a BYTE state variable which signals the occurrence of a nonmaskable bus timeout interrupt.</td>
</tr>
<tr>
<td>INITIALIZE$INTERRUPTS</td>
<td>a PROCEDURE which initializes the 8259 interrupt controller chip, sets up 256 interrupt vectors, and unmasks and enables all interrupts.</td>
</tr>
</tbody>
</table>
$COMPACT RON NOINTVECTOR OPTIMIZE(3)
TITLE('NASA/ADRF/ XAIDS MAINT INTERRUPT ROUTINES 14 Jan 1987')

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

RPTPS.P86: DO 1

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

2 1 DECLARE PICFLS LITERALLY 'OC0H' ; /* PROGRAMMABLE INTERRUPT CONTROLLER */
3 1 DECLARE PICSK  LITERALLY 'OC2H' ;

/* MISCELLANEOUS DECLARATIONS */

4 1 DECLARE VECTOR (256) POINTER AT (000000H) ;
5 1 DECLARE MP8TO BYTE AT (0C701EH) ; /* PERPRO MAINT DTO FLAG */
6 1 DECLARE INTACOUNTER (0) WORD PUBLIC ; /* MULTIBUS INTERRUPT TALLY COUNTERS */
7 1 DECLARE HW1COUNTER WORD PUBLIC ; /* DEADMAN TIMER INTERRUPT COUNTER */
8 1 DECLARE UNKN0WNCOUNTER WORD PUBLIC ; /* REMAINING Rupt TYPES SHARE COUNTER */
9 1 DECLARE DTOZFLAG BYTE PUBLIC ; /* BUS TIMEOUT FLAG */

/* FOLLOWING ARE INTERRUPT TALLY ROUTINES */

10 1 UNKN0WNCOUNTERINCR = PROCEDURE INTERRUPT 0 ;
11 2 UNKN0WNCOUNTER = UNKN0WNCOUNTER + 1 ;
12 2 END ;
13 1 HW1COUNTERINCR = PROCEDURE INTERRUPT 2 ;
14 2 HW1COUNTER = HW1COUNTER + 1 ;
15 2 DTOZFLAG/MP8TO = OFFH ; /* SET DTO FLAG LOCALLY & IN PERPRO */
16 2 OUTWORD(PICFLS) = 40H ; /* CLEAR FLIP-FLOP (OC02 NO-OP) */
17 2 END ;
18 1 EDI: PROCEDURE ; /* END OF INTERRUPT */
19 2 OUTPUT(PICFLS) = 20H ;
20 2 END ;
21 1 INTACOUNTERINCR = PROCEDURE INTERRUPT 32 ;
22 2 INTACOUNTER(0) = INTACOUNTER(0) + 1 ;
23 2 CALL EDI ;
24 2 END ;
25 1 INTACOUNTERINCR = PROCEDURE INTERRUPT 33 ;
26 2 INTACOUNTER(1) = INTACOUNTER(1) + 1 ;
27 2 CALL EDI ;
28 2 END ;
29 1 INTACOUNTERINCR = PROCEDURE INTERRUPT 34 ;
30 2 INTACOUNTER(2) = INTACOUNTER(2) + 1 ;
31 2 CALL EOI;
32 2 END;

33 1 INT4COUNTER$INCREMENT: PROCEDURE INTERRUPT 35;
34 2 INT4COUNTER(3) = INT4COUNTER(3) + 1;
35 2 CALL EOI;
36 2 END;

37 1 INT4COUNTER$INCREMENT: PROCEDURE INTERRUPT 36;
38 2 INT4COUNTER(4) = INT4COUNTER(4) + 1;
39 2 CALL EOI;
40 2 END;

41 1 INT4COUNTER$INCREMENT: PROCEDURE INTERRUPT 37;
42 2 INT4COUNTER(5) = INT4COUNTER(5) + 1;
43 2 CALL EOI;
44 2 END;

45 1 INT4COUNTER$INCREMENT: PROCEDURE INTERRUPT 38;
46 2 INT4COUNTER(6) = INT4COUNTER(6) + 1;
47 2 CALL EOI;
48 2 END;

49 1 INT4COUNTER$INCREMENT: PROCEDURE INTERRUPT 39;
50 2 INT4COUNTER(7) = INT4COUNTER(7) + 1;
51 2 CALL EOI;
52 2 END;

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

 /**< LOADING RUPIT 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$PTR(WMI$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(PICFILB) = 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(PICFILB) = 40H /**< Clear Flip-Flop (Oct2 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 34B
- 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
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
$COMPACT ROM OPTIMIZE(3)
$TITLE('NASA/ADFRF XAIDS MAINT MDM ROUTINES 14 Jan 1987')

// NASA Ames Dryden Flight Research Facility R Glover #/
// SBX251 Magnetic Bubble Memories Multimodule Routines #/

1 MDM_PB6: 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 MAPPINGS */

11 1 DECLARE MEMDAT LITERALLY '000H' ; /* MAGNETIC BUBBLE MEMORY */
12 1 DECLARE MEMCMD LITERALLY '002H' ;

/* I/O STATUS FLAGS */

13 1 DECLARE MEM|BUSY LITERALLY 'SHR(INPUT(MEMCMID),7)' ;
14 1 DECLARE MEM|COMPLETE LITERALLY 'SHR(INPUT(MEMCMID),6)' ;
15 1 DECLARE MEM|FAIL LITERALLY 'SHR(INPUT(MEMCMID),5)' ;
16 1 DECLARE MEM|TMOING|ERROR LITERALLY 'SHR(INPUT(MEMCMID),4)' ;
17 1 DECLARE MEM|CORRECTABLE|ERROR LITERALLY 'SHR(INPUT(MEMCMID),3)' ;
18 1 DECLARE MEM|UNCORRECTABLE|ERROR LITERALLY 'SHR(INPUT(MEMCMID),2)' ;
19 1 DECLARE MEM|PARITY|ERROR LITERALLY 'SHR(INPUT(MEMCMID),1)' ;
20 1 DECLARE MEM|FIFO|READY LITERALLY 'INPUT(MEMCMID)' ;

/* MISCELLANEOUS DECLARATIONS */

21 1 DECLARE TRUE LITERALLY 'OFFH' ;
22 1 DECLARE FALSE LITERALLY '0' ;
23 1 DECLARE BLOCK (64) BYTE ; /* BUBBLE MEMORY TEMP STORAGE */
24 1 DECLARE LASTt|25|COMMAND BYTE ; /* BUBBLE MEMORY LAST COMMAND */
25 1 DECLARE BYTE|COUNT|ER WORD ; /* BUBBLE MEMORY DATA TRANSFER COUNTER */
26 1 DECLARE TIME|COUNT|ER WORD ; /* BUBBLE MEMORY DATA TRANSFER DEADLINE */
27 1 DECLARE EDIT|BUFFER (1280) BYTE EXTERNAL ; /* USED BY JE COMMAND */

28 1 MDM|ERROR: PROCEDURE (MSG|PTR) ;
29 2 DECLARE MSG|PTR|POINTER ;
30 2 CALL WRITE(0, MDM cmd ;/0) ;
31 2 CALL HEX1(LASTt|25|COMMAND) ;
32 2 CALL WRITE(0, error ;/107H,0) } ;
33 2 CALL WRITE(MSG|PTR) ;
34 2 END ;

35 1 COMMAND|25|1: PROCEDURE (CMD) BYTE ;
36 2 DECLARE CMD BYTE ;
37 2 LAST251: COMMAND = CMD ;
38 2 OUTPUT(NMBCHD) = 00010000B OR ( CMD AND 00001111B ) ;
39 2 TIMEOUT4COUNTER = 0 ;
40 2 DO WHILE NOT NMB0BUSY ;
41 3 TIMEOUT4COUNTER = TIMEOUT4COUNTER - 1 ;
42 3 IF TIMEOUT4COUNTER = 0 THEN
43 3 DO ;
44 4 CALL NMBMERROR(0('command accept timeout '/0)) ;
45 4 RETURN FALSE ;
46 4 END ;
47 3 END ;
48 2 RETURN TRUE ;
49 2 END ;

50 1 EXECUTE251: PROCEDURE BYTE ;
51 2 TIMEOUT4COUNTER = 0 ;
52 2 DO WHILE NOT NMB0BUSY ;
53 3 TIMEOUT4COUNTER = TIMEOUT4COUNTER - 1 ;
54 3 IF TIMEOUT4COUNTER = 0 THEN
55 3 DO ;
56 4 CALL NMBMERROR(0('command execute timeout '/0)) ;
57 4 RETURN FALSE ;
58 4 END ;
59 3 END ;
60 2 RETURN TRUE ;
61 2 END ;

62 1 COMPLETE251: PROCEDURE BYTE ;
63 2 TIMEOUT4COUNTER = 0 ;
64 2 DO WHILE NOT NMB0COMPLETE ;
65 3 IF NMB0FAIL THEN
66 3 DO ;
67 4 CALL NMBMERROR(0('or fail status '/0)) ;
68 4 CALL HEX2INPUT(NMBCHD)) ;
69 4 RETURN FALSE ;
70 4 END ;
71 3 TIMEOUT4COUNTER = TIMEOUT4COUNTER - 1 ;
72 3 IF TIMEOUT4COUNTER = 0 THEN
73 3 DO ;
74 4 CALL NMBMERROR(0('or complete timeout '/0)) ;
75 4 RETURN FALSE ;
76 4 END ;
77 3 END ;
78 2 RETURN TRUE ;
79 2 END ;

80 1 SETUP251: PROCEDURE(NBLOCS)/START) ;
81 2 DECLARE (NBLOCS)/START) WORD ;
82 2 OUTPUT(NMBCHD) = 00011101B ; /* RESET FIFO */
83 2 CALL TIME(16) ;
84 2 OUTPUT(NMBCHD) = 00011111B ; /* SOFTWARE RESET */
85 2 CALL TIME(16) ;
86 2 OUTPUT(NMBCHD) = 00010111B ; /* START WITH BLOCK LENGTH REG */
87 2 CALL TIME(16) ;
88 2 OUTPUT(NMMDAT) = LOW(NBLOCS) ;
89 2 OUTPUT(NMMDAT) = 00010000B OR ( HIGH(NBLOCS) AND 00001111B ) ;
90 2 OUTPUT(NMMDAT) = 00100008 ; /* ENABLE RCD ONLY */

ORIGINAL PAGE IS OF POOR QUALITY
RUBS251: PROCEDURE (NBLOCKS,ISTRATIONPTR) BYTE PUBLIC
DECLARE (NBLOCKS,ISTRATION,INDEX) WORD; PTR POINTER, BUFFER BASED PTR (\$) BYTE
TIMEOUTCOUNTER = 0;
INDEX = 0;
CALL SETUP251(NBLOCKS,ISTRATION);
IF NOT COMMAND251(2) THEN RETURN FALSE;
DO WHILE BYTE\$COUNTER (> 0)
IF MBM4FIFO\$READY THEN
DO;
BUFFER(INDEX) = INPUT(NMBDAT);
INDEX = INDEX + 1;
BYTE\$COUNTER = BYTE\$COUNTER - 1;
END;
ELSE
DO;
TIMEOUTCOUNTER = TIMEOUTCOUNTER - 1;
IF TIMEOUTCOUNTER = 0 THEN
DO;
CALL MBM4ERROR('@('read FIFO timeout \',0))';
GO TO ABORT;
END;
END;
END;
END;
IF NOT EXECUTE251 THEN GO TO ABORT;
IF NOT COMPLETE251 THEN GO TO ABORT;
RETURN TRUE;
ABORT:
CALL INITIALIZE251;
RETURN FALSE;
END;

WRITE251: PROCEDURE (FILE) PUBLIC
DECLARE FILE BYTE, INDEX WORD
TIMEOUTCOUNTER = 0;
INDEX = 0;
CALL SETUP251(20,20+FILE);
IF NOT COMMAND251(3) THEN RETURN;
DO WHILE BYTE\$COUNTER (> 0)
153 3 IF MBM$FIFO$READY THEN
154 3 DO ;
155 4 OUTPUT(MBM$DAT) = EDIT$BUFFER(INDEX) ;
156 4 INDEX = INDEX + 1 ;
157 4 BYTE$COUNTER = BYTE$COUNTER - 1 ;
158 4 END ;
159 3 ELSE
160 4 TIMEOUT$COUNTER = TIMEOUT$COUNTER - 1 ;
161 4 IF TIMEOUT$COUNTER = 0 THEN
162 4 DO ;
163 5 CALL MBM$ERROR(0I('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 ;
169 2 IF NOT COMPLETE$251 THEN GO TO ABORT ;
170 2 RETURN ;
171 2 ABORT;
172 2 CALL INITIALIZE$251 ;
173 2 END ;
174 2 END ;
175 1 END ;

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%)
OKB DISK SPACE USED

END OF PL/M-86 COMPILATION
REFERENCES


### TABLE 1. - TOP-LEVEL COMMAND KEYSTROKES

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

<table>
<thead>
<tr>
<th>XX</th>
<th>XX</th>
<th>AA</th>
<th>IIIII</th>
<th>DDDDDDD</th>
<th>SSSSSSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX</td>
<td>XX</td>
<td>AAA</td>
<td>II</td>
<td>DD</td>
<td>DD</td>
</tr>
<tr>
<td>XX</td>
<td>XX</td>
<td>AA</td>
<td>AA</td>
<td>II</td>
<td>DD</td>
</tr>
<tr>
<td>XX</td>
<td>XX</td>
<td>AAA</td>
<td>AAA</td>
<td>II</td>
<td>DD</td>
</tr>
<tr>
<td>XX</td>
<td>XX</td>
<td>AA</td>
<td>AA</td>
<td>II</td>
<td>DD</td>
</tr>
</tbody>
</table>

NMM NMM AA IIIIIII NNN NNN TTTT
NMM NMM NMM AAA II NNN NNN TT
NMM NMM NMM AA AA II NNN NNN TT
NMM NMM NMM AA AA II NNN NNN TT
NMM NMM NMM AAA AA AA II NNN NNN TT
NMM NMM NMM AAA AA AA II NNN NNN TT

NMM NMM AA AA IIIIIII DDDDDDD SSSSSSS

Nasa/af/e/rf XaibS maintenance processor 14 January 1987 R. Glover
Menu: Help; 0-9, JD, JE, Jd, I(sp), M(sp), J(sp), (sp), IRB, IRH, IFB, IFW, IMB, IMN, MNB, MND, MRB, MRP, MRA, MFB, MFN, MFD, MFS, R, S

Display 1. Initial signon.

Menu: Help; 0-9, JD, JE, Jd, I(sp), M(sp), J(sp), (sp), IRB, IRH, IFB, IFW, IMB, IMN, MNB, MND, MRB, MRP, MRA, MFB, MFN, MFD, MFS, R, S

Help:
(u) = Job load & run. u = 0 thru 9, selects file = 10dd + u.
(Jd) = Job decade. d = 0 thru 9, selects decade.
(JB) = Job directory. Shows titles for selected job decade.
(I(sp)) = 1/0 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)).
0 = B or N Display M(word).
I0 = 1/0 read. Sets up byte or word I/O port group display.
I0 = 1/0 write. Repetitive byte or word outputs to selected port.
I0 = 1/0 fill. Writes byte or word to a block of ports.
M0 = Memory write. Repetitive byte or word memory writes.
0 = B, W, D, P, or A Display M(word), D(word), Plainter, or ASCII.
H0 = Memory read. Sets up byte, word, dword, pointer, or ASCII dumb.
0 = B, W, D, or A Display M(word), D(word), or ASCII.
H0 = Memory fill. Dete, word, dword, or ASCII string fill.
S0 = Memory substitute. Date scan/modify.
S0 = Rupts. Displays multibus interrupt counters.
S = Sesent. Reloads memory base register.

Display 2. Help page.

Display 4. Job file executions.
Display 5. Editing job buffer.

Display 7. Miscellaneous memory operations.

Display 8. MBM backup operation.
Display 9. I/O read examples.
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.