The system in this thesis uses multiple Intel iSBC86/12A cards connected to the Intel Multibus. The system was written using the Intel language PLM86. The final result is a low level debug tool that provides the user with the following capabilities: examine/change memory, examine/change processor registers, load a user written program, set a breakpoint, and execute the loaded program all in a multi-microprocessor environment. The system is designed to use five iSBC86/12A cards currently two are implemented. The system interfaces to the Intel MDS 800 system via the SDM86 monitor hardware and firmware.
A MULTI-MICROPROCESSOR DEBUG EXECUTIVE

BY

DOUGLAS E. EDMONDS

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed August 21, 1986

Commencement June 1987
APPROVED:
Redacted for Privacy

Associate Professor of Electrical and Computer Engineering in charge of major

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date of Thesis is presented AUGUST 21, 1986

Typed by Douglas E. Edmonds for Douglas E. Edmonds
ACKNOWLEDGMENT

I would like to thank the Department of Electrical and Computer Engineering for the opportunity to study at Oregon State University. I am grateful to Professor Rathja for his patience and encouragement during the work on the hardware and software. Furthermore I am thankful for the support from my wife during the many hours I have spent working on this thesis and graduate courses. Finally, I thank God for His guidance through the rough times and the good times.

Douglas E. Edmonds
August 21, 1986
TRADEMARKS

THE FOLLOWING ARE TRADEMARKS OF THE INTEL CORPORATION AND ITS AFFILIATES.

iSBC, Multibus, Intel, UPI, Intellec, UPP, iSPC.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER ONE</th>
<th>INTRODUCTION</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER TWO</td>
<td>BACKGROUND</td>
<td>6</td>
</tr>
<tr>
<td>CHAPTER THREE</td>
<td>CRITICAL ISSUES</td>
<td>15</td>
</tr>
<tr>
<td>CHAPTER FOUR</td>
<td>THE MMDE SYSTEM</td>
<td>23</td>
</tr>
<tr>
<td>CHAPTER FIVE</td>
<td>SUMMARY AND CONCLUSION</td>
<td>35</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>SWITCH SETTINGS OF SDM CARD</td>
<td>41</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>SWITCH SETTINGS OF CPU CARD</td>
<td>43</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td>MEMORY MAP OF CPU CARD</td>
<td>45</td>
</tr>
<tr>
<td>APPENDIX D</td>
<td>MEMORY MAP OF SDM CARD</td>
<td>46</td>
</tr>
<tr>
<td>APPENDIX E</td>
<td>BUFMON PROCEDURES</td>
<td>47</td>
</tr>
<tr>
<td>APPENDIX F</td>
<td>DEBUG PROCEDURES</td>
<td>51</td>
</tr>
<tr>
<td>APPENDIX G</td>
<td>UEXT PROCEDURES</td>
<td>62</td>
</tr>
<tr>
<td>APPENDIX H</td>
<td>COMMON MEMORY DESCRIPTION</td>
<td>66</td>
</tr>
<tr>
<td>APPENDIX I</td>
<td>COMMAND SET/INSTRUCTIONS</td>
<td>69</td>
</tr>
<tr>
<td>APPENDIX J</td>
<td>SAMPLE USER CODE - LOCATE EXAMPLE</td>
<td>72</td>
</tr>
<tr>
<td>APPENDIX K</td>
<td>MMDE DISC DIRECTORY</td>
<td>75</td>
</tr>
<tr>
<td>APPENDIX L</td>
<td>MMDE LIBRARY SOURCE CODE</td>
<td>77</td>
</tr>
<tr>
<td>APPENDIX M</td>
<td>BUFMON SOURCE CODE</td>
<td>79</td>
</tr>
<tr>
<td>APPENDIX N</td>
<td>CPUID SOURCE CODE</td>
<td>89</td>
</tr>
<tr>
<td>APPENDIX O</td>
<td>DBGUGMON SOURCE CODE</td>
<td>92</td>
</tr>
<tr>
<td>APPENDIX P</td>
<td>UEXT SOURCE CODE</td>
<td>117</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURES</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. BUS STRUCTURES</td>
<td>4</td>
</tr>
<tr>
<td>2. COMMUNICATION STRUCTURES</td>
<td>4</td>
</tr>
<tr>
<td>3. BLOCK DIAGRAM OF SYSTEM</td>
<td>6</td>
</tr>
<tr>
<td>4. SYSTEM INTERCONNECTION</td>
<td>8</td>
</tr>
<tr>
<td>5. iSBC 86/12A BLOCK DIAGRAM</td>
<td>9</td>
</tr>
<tr>
<td>6. SYSTEM SOFTWARE</td>
<td>14</td>
</tr>
<tr>
<td>7. FORM OF DEADLOCK</td>
<td>16</td>
</tr>
<tr>
<td>8. SEMAPHORE SETTING</td>
<td>18</td>
</tr>
<tr>
<td>9. MUTUAL EXCLUSION CODE</td>
<td>19</td>
</tr>
<tr>
<td>10. MONITOR CONCEPT</td>
<td>20</td>
</tr>
<tr>
<td>11. MONITOR CALL</td>
<td>22</td>
</tr>
<tr>
<td>12. FLOW DIAGRAM OF CONSUMER</td>
<td>27</td>
</tr>
<tr>
<td>13. FLOW DIAGRAM OF PRODUCER</td>
<td>28</td>
</tr>
<tr>
<td>14. PARTIAL BLOCK DIAGRAM OF BUFMON</td>
<td>29</td>
</tr>
<tr>
<td>15. MMDE COMMANDS</td>
<td>32</td>
</tr>
<tr>
<td>16. EXAMINE COMMAND ENTRY</td>
<td>33</td>
</tr>
<tr>
<td>17. EXAMINE COMMAND REGISTER RESULTS</td>
<td>33</td>
</tr>
</tbody>
</table>
A MULTI-MICROPROCESSOR DEBUG EXECUTIVE

CHAPTER ONE

INTRODUCTION

This dissertation presents the investigation and implementation of a multiple-microprocessor system. The completed system provides a minimum set of tools to write and troubleshoot programs for a multi-microprocessor environment. Commands to change memory, change registers, set a breakpoint, and execute code are provided.

The need for multi-microprocessor systems is a natural outgrowth of the desire and drive to find greater performance in computing power. This performance is usually translated as computational speed or throughput. There are several metrics used that give an idea of the processor’s performance. MIPS (millions of instructions per second) or FLOPS (floating point operations per second) are common metrics used in comparing relative performance of a given processor system. Another way processor systems are ranked is to use a set of benchmark tests. These tests are usually software programs written to exploit the computational ability of the processor system. The metric used in these tests is the time it took the processor to execute the given benchmark test.

Only in the last ten years has the price of microprocessors decreased to where the building of a multi-microprocessor system was economical. Several systems have been built commercially to exploit this idea. The key point in these systems is to be able to break a task into its parallel processes and then keep all the processors in the system fully utilized in an efficient manner to achieve high performance.
The Flex/32 is a general purpose multicomputer from Flexible. The machine uses common communication hardware for access to shared memory. In addition, each processor has local memory. Within the message hardware is a lock system to provide exclusive access for processors sharing data and/or programs [1]. The system uses the National Semiconductor 32032 32-bit microprocessor and family peripherals.

The FX/8 series of computers from Alliant Computer Systems Corporation uses a Motorola 68012 microprocessor for handling interaction between processors running a common code sequence. The Motorola 68012 has virtual memory capability and addresses a two-gigabyte linear address range [2]. The core processor is a custom processor using a pipe-line data path. Each interactive processor has local memory and communicates with the system global memory [3].

Another parallel machine is the Connection Machine. The CM-1 prototype Connection machine uses 65,563 single bit processors. Each processor uses 4096 bits of memory for a total of 32 megabytes. What makes this machine interesting is its ability to configure an interprocessor communication network for the problem being solved [4,5].

Intel Corporation uses a HYPERCUBE connection scheme to boost the performance of its multi-microprocessor machine, the iPSC, (Intel Personal Supercomputer). The HYPERCUBE is composed of up to 64 nodes in the vector processor version (iPSC-VX). Each node contains two boards, a node processor and a vector processor board which communicates with the node processor via the iLBX backplane interface. Each node can communicate to its six nearest neighbors in a cube using six of the Ethernet channels. The seventh Ethernet channel is a global channel used to communicate to the cube manager. An eighth Ethernet channel is reserved for future use. The cube manager is an Intel System 310 and the node processors are Intel's 16 bit processor, the 80286, and the math coprocessors are Intel's 80287. The top-of-the-line cube costs about one-tenth the price of a comparable performance Cray system [6,7].
Another company, Sequent Computer Systems uses an array of National Semiconductor's 32-bit 32032 microprocessor's on a proprietary bus. The machine is capable of 2 to 7 MIPS depending on configurations. A two processor system provides the equivalent of a VAX 780 performance. The processors in the system communicate via a two-wire serial bus for setting semaphores and interlock mechanisms. By using a separate bus for resource sharing the traffic on the system bus is reduced [6].

In all of these systems there is a basic problem. It is difficult to write efficient code that fully utilizes all of the processors. The effort involved is much greater than that required for a uniprocessor system. Each multiprocessor system offered requires some additional software tools to reduce the burden on the programmer.

There are as many configurations and architectures as there are multi-microprocessor systems. The classification of these various systems is not an easy task or a clear cut one. One classification method involves the hardware architecture. These vary from a global bus structure, such as the Flex 32, to processors that act as processing elements with bus connections to near neighbors as in the case of the HYPERCUBE. There are many variations. (See Figure 1).
Another way to classify the systems is to use the communication process [9]. How is the information passed from one processor to the next processor? Is the information passed as a message or is it stored in a global shared memory? See Figure 2.

An excellent overview of these issues is given in an article by Gajski and Pier [10]. Briefly the issues are: hierarchial control,
partitioning, scheduling, synchronization and memory access. Hierarchial control deals with the way the system manages tasks. Partitioning deals with trying to exploit parallelism. Scheduling is the assigning of tasks to a specific processor for execution. Synchronization is the controlling of access to the shared information between competing processors. Memory access is the problem of interconnection of the memory to the multi-microprocessor system.

The MMDE (Multi-Microprocessor Debug Executive) system presented in this dissertation can be classified as a global bus system with symmetrical processors. Each processor is exactly the same and can perform the same tasks. The MMDE acts as a "floating master" system, since each processor has a copy of the MMDE system code. Each processor is responsible for its own tasks as they are assigned. Each processor may be at a different point of execution depending on the task it is executing. This distinguishes the system from a "master-slave" configuration where the master is the only processor that has a copy of the operating system.
CHAPTER TWO
BACKGROUND

The MMDE system was developed using Intel equipment and software. It can be divided into four areas. They are the Development station hardware, the Multi-microprocessor system, the Development station software and the MMDE software.

DEVELOPMENT STATION HARDWARE

The development station hardware consists of an MDS 800 Intellect system. This system uses an 8085 microprocessor. Attached to the MDS (Microprocessor Development System) is a dual 8" floppy disc drive. Also a Universal Prom Programmer is connected to program 2716 EPROMs. The MDS system uses a GT-110 terminal and a Teletype Model 40 impact printer. See Figure 3 for a block diagram of the system.

MULTI-MICROPROCESSOR SYSTEM

The equipment rack houses two main components; a power supply and a card cage. There is also a panel for connectors and several momentary push buttons. Only the Master Reset button is wired. The switch is
connected to the NMI (Non-Maskable Interrupt) on the auxiliary bus connector (P2) of the card cage.

Power Supply

The power supply is a GSC Power-Miser Model No. GOL4001 switching supply. The outputs are +5VDC @ 45A, +/- 12VDC @ 10A, -5VDC @ 4A and 24VDC @ 4A. The input is 120VAC. This is an open frame power supply with no forced cooling. The outputs are wired to the appropriate power supply pins on the backplane of the card cage. Each of the cards in the MMDE system use power from the backplane.

Card Cage

A MULTI-CAGE card cage from Electronics Solutions is used and has room for 21 card slots. Currently only four slots are used, and then only every other slot. This allows for better air cooling. The backplane of the card cage conforms to the IEEE 769 Standard System Backplane Bus standard, Multibus [11]. The Multibus is capable of a 10MHz bandwidth during 16-bit transfers and a 5Mhz bandwidth using 8-bit transfers. The following pieces of hardware are used in the card cage: SDM monitor board and firmware, the CPU cards and firmware, and a Memory extension card. See Figure 4 for a block diagram showing the interconnection.
<table>
<thead>
<tr>
<th>SLOT 1</th>
<th>SLOT 3</th>
<th>SLOT 5</th>
<th>SLOT 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDM CARD</td>
<td>BUS PORT</td>
<td>BUS PORT</td>
<td>MEMORY CARD</td>
</tr>
<tr>
<td>PARALLEL PORT (J3)</td>
<td>CPU CARD</td>
<td>CPU CARD</td>
<td>GLOBAL</td>
</tr>
<tr>
<td>RIBBON CABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SYSTEM INTERCONNECTION**

**FIGURE 4**

**SDM CARD**

The System Debug Monitor (SDM) board is an Intel single board computer (iSBC 86/12A), card customized with the Intel iSDM 86 System Debug Monitor hardware and firmware. Appendix A shows the switch settings and wire jumpers for the card. The system bus clock is 10MHz and is supplied by the SDM board to the backplane. The SDM monitor board connects to the INTELLECT system box via a 50 wide ribbon cable from the parallel port (J1) on the SDM card to the parallel port on the MDS system. At invocation of the MMDE, the SDM card is loaded with the software required for interfacing the MMDE to the MDS 800. The SDM card must be connected to the parallel port of the MDS 800 when using the MMDE system. During development the MDS parallel port is shared between the prom programmer and the SDM. See Figure 5 for a block diagram of the iSBC 86/12A card. The basic iSDM86 functions allow code development for a single board processor system. It provides the commands to load programs, set breakpoints, examine/change processor registers, and examine/change memory locations [12]. The SDM card is used in the MMDE system as an interface to the MDS system.

**CPU CARD**

Each of the CPU cards is also an iSBC 86/12A single board computer card which contains an Intel 8086 16-bit microprocessor running at 5MHz.
a serial communication interface, three programmable parallel I/O ports, two programmable timers, priority interrupt controller, Multibus interface logic, 32K bytes of RAM and up to 8K bytes of ROM [13]. The ranges of address settings for on and off board memory can be set using the switches on the CPU card. Each CPU card has its own copy of the MMDE software. The MMDE software is stored in EPROM and is identical for each processor. Appendix D shows the memory map of the 8612A cards. Appendix E shows the settings of the switches and wire jumpers. Figure 5 shows the block diagram of the iSBC 86/12A card.

The switch settings and wire jumpers on each board are the same except for the wire which indicates the unique identification of each processor. This is done using the on board timer (8253) and interrupt controller (8259). After power is applied, the first section of code initializes the timer and interrupt controller and reads the CPU identification, known as CPUID in the source code. Appendix B shows the required jumpers and switch settings. Appendix N shows the source code to implement the CPUID function.
The CPU board contains circuitry to allow an exclusive lock on the global bus. The exclusive lock permits an uninterruptable read/write bus cycle. This feature is important since it allows multiple microprocessors to control the use of shared resources. The CPU asserts the LOCK NOT signal which is used by the 8289 bus controller arbiter chip, to assert the bus busy signal, BUSY NOT [13].

MEMORY CARD

The extended memory card addresses start at 0000:8000 hex. This is where the common memory or global shared memory for each processor is located. All processors can reference this external memory using the same address. The external memory is 8K x 8 bytes of dynamic RAM. The MMDE system uses 126 bytes of this shared external memory. See appendix C and D for memory maps.

GLOBAL BUS

The backplane bus is a global bus, all the CPU cards, the SDM card and the Memory card communicate using this bus. The serial interrupt priority system is guaranteed to function properly for a maximum of 3 cards because of the timing constraints of the Multibus system. Currently the system uses a maximum of 3 cards. A parallel priority system can be added to expand the system to additional processors. The SDM card supplies the common bus clock to the global bus. The bus clock on the other cards is disabled.

DEVELOPMENT STATION SOFTWARE

The software used during development consisted of the following Intel supplied programs: CREDIT, PLM86, SDM86, LINK86, LOC86, LIB86 and UPM. Each of these software packages are briefly described in the following paragraphs.
CREDIT

Credit is the development station editor. It was used to write the source code of the MMDE software. It can be used as a screen oriented editor or a line editor [14].

PLM86

PLM86 is the language used to develop the software. It is a block structured language. The entire MMDE software package was written in PLM86. This particular version of the PLM compiler runs on an 8085 based development system [15,16]. PLM86 supports the hardware LOCK NOT signal of the 8086.

SDM86

SDM86 is the monitor program that allows the MDS system to communicate with the SDM board in the card cage. The monitor has a loader program for use in loading software to a target iAPX86,88 or 186-based system. The SDM monitor also has a CEI (command extension interface) which allows users to build their own commands [12]. This feature is used by the MMDE and the MDS system to communicate between the two systems. There are two parts to the SDM86 monitor system. One part is the software that runs on the MDS and the other part is the firmware that is resident on the SDM card in the form of EPROM.

LINK86, LOC86

LINK86 and LOC86 are utilities that allow linking and loading the various software modules together into one object code format. LOC86 allows absolute addresses to be assigned to the object code. This feature is used when locating the MMDE software for using the EPROM address range [17].
LIB86

LIB86 is a library program that allows libraries to be created. This program was used to create the MMDE library for use with the user written programs [17]. Appendix J shows an example of linking in the MMDE library.

OH86

OH86 is a program that converts an object code file into an ASCII hexadecimal file that is suitable for printing and/or copying. The hexadecimal file format is used by the MMDE system to download software to the CPU cards [17].

UPM86

UPM (Universal Prom Mapper) is a program that takes an object code file and sends it to the PROM programmer for programming an EPROM. The UPM also contains commands that allow splitting the object code into high and low bytes for use in an 8086 environment [18].

MMDE SYSTEM SOFTWARE

The MMDE system software is composed of four main components: DBUGMON, BUFMON, CPUID and UEXT. These segments will be explained briefly in this chapter. A detailed discussion will follow in Chapter 4.

DBUGMON

This is the main code segment in the MMDE system. This code along with BUFMON resides in EPROM on the CPU boards. DBUGMON is activated
when power is applied and is responsible for executing system commands. See appendix 0 for the source code of DEBUGMON.

BUFFMON

BUFFMON is the heart of the communication system for the MMDE system. It is responsible for passing information between processors. See appendix M for the source code of BUFFMON.

CPUID

This program is used to establish the unique identity of a calling processor. The program uses the interrupt controller in a poll mode to read the processor identity and returns the identity to the calling program. The identity of the calling processor is usually referenced as CPUID. This program routine is located in EPROM on the CPU boards. See appendix N for the source code of CPUID.

UEXT

UEXT is the program that resides in RAM on the SDM monitor board when the MMDE system is activated. This program is responsible for transferring requests between the MDS system and the MMDE system. This includes the request for the user to tell the MMDE system which commands to execute. It also interfaces with the MDS and the MMDE during the command to download user written programs. See Figure 6 for a diagram of the System Software. See appendix P for the source code of UEXT.
M D S 800 SYSTEM
SDM86 SOFTWARE

SDM CARD
SDM86 FIRMWARE
UEXT UEXT
MMDE SOFTWARE

GLOBAL BUS

CPU CARD
<---BUFMON, DBUGMON, CPUID Firmware --- EPROM

SYSTEM SOFTWARE

FIGURE 6
CHAPTER THREE
CRITICAL ISSUES

One of the main issues in a multiprocessor system is the communication between processors. The processors in the system are usually running independently of each other. Each has its own local processor clock. In a shared memory system, each processor can read or write to a shared memory location at any time. This can cause some interesting results.

Suppose processor A is writing data and processor B is reading and then clearing the data at the same shared memory locations. In a given sequence of events processor A writes the data, processor B reads the data, then before processor B can clear the data processor A writes new data. Now if processor B clears the data, it has cleared data that was never read, hence the data was lost. There is actually another problem here in that processor B could try reading the data before processor A is finished updating the data. Hence erroneous data could be read by processor B.

R.C. Holt describes several different situations where data which is being used by two processors can be lost or corrupted [19]. This type of problem is caused by race conditions in the processors. That is, one processor finishes execution before the other one. It is interesting to note that sometimes processor B might read and clear the data as intended. Correct operation depends on the processor's activities at the given time.

A second area of conflict in a multiprocessor system is deadlock. Deadlock can occur in several ways and could be given different names depending on the type of deadlock. A simple definition of deadlock is; a processor waits for an event that can never happen. Again a situation can arise where processor A calls processor B, now processor A is
waiting for processor B to respond, in order to respond to processor A. Processor B tries to call processor A. However, because processor A is busy waiting for processor B to respond, it cannot respond. So now processor B is busy waiting for processor A to respond which in turn is waiting for processor B to respond. Hence the system is deadlocked waiting for an event that can never happen. See Figure 5 for a diagram of this type of deadlock.

Another type of deadlock occurs when a given processor uses all of the system resources and does not return them for other processors to use. This is known as an unfriendly deadlock. A third type of deadlock can occur when there are X number of resources available and X+1 users trying to access the resources. The problem here is that the one user could always end up unable to get the resource it requires [19].

For the problem of reading and writing of shared data there must be some type of control during the reading/writing process. While processor A is writing the data, processor B cannot be allowed to read the data. What is needed is some form of mutual exclusion that prevents processor B from interfering with the shared data processor A is updating.

For a data update processor A needs some method of signaling other processors that the data is being updated. The signal would be set by one processor and checked by other processors before they attempt to use
the resource. If the signal indicates not busy, the checking processor would set the signal busy and then use the resource. A common name for this signal is a semaphore. The semaphore is a shared variable among all the processors. Each processor will require access to the semaphore to check its status. The semaphore has to be set and updated without interruption from the other processors.

In a multiprocessor system events are asynchronous, so software/hardware timing of events cannot guarantee a synchronous order of execution. To implement a semaphore requires the hardware to block any events from other processors that would interrupt the read/test/write cycle required in implementing the semaphore. Depending on the processors' architecture and instruction set this could be implemented in various forms. The implementation is also dictated by the system architecture.

These signals can be used to develop circuitry for an indivisible bus cycle. The sequence of events would be for a processor to read the semaphore, if it is clear then set the semaphore, if not continue checking (busy wait loop). During this reading/testing/writing of the semaphore other processors are prevented from interrupting.

One form of this instruction is known as a Test and Set (TAS) instruction in the Motorola 68020 processor [20]. The same type of instruction is known as an Interlock Operation (ILO NOT) in the National Semiconductor NS32032 32-bit microprocessor [21]. This instruction could be used in a loop until the semaphore is set (busy wait loop). See Figure 8 for a diagram of the TAS instruction and looping.

A similar instruction exists in the Intel 8086 microprocessor and is known as LOCKSET in the PLM86 language. The actual implementation of this instruction consists of the LOCK prefix instruction being added to the XCHG instruction. The XCHG instruction swaps two 8-bit bytes. When the LOCK prefix is added the instruction is executed in one indivisible
bus cycle. Normally the instruction would take two bus cycles that could be interrupted. At the hardware level the signal is known as LOCK NOT. The Bus Interface Unit (BIU) activates this signal when the processor executes the LOCK prefix instruction. This signal is only available in systems running in the 8086 maximum mode. This signal along with the bus arbiter chip (8289) implement the bus lock in the MULTIBUS system [22].

If such an instruction exists or can be implemented, then a system can be built that will allow reliable multiprocessor semaphore setting. By using the LOCKSET instruction, the semaphore can be properly and reliably used. Semaphores can then be used to protect shared data or other shared resources. Each time a processor wants to use or update shared data, it must first check the semaphore to determine if it is okay to change the data. By using a LOCKSET instruction, the semaphore can be set to busy. Then the shared data can be updated and the semaphore cleared. Several procedures or calls can be written that have functions such as "Mutual Exclusion Begin" and "Mutual Exclusion End". Then, when a shared variable needs to be updated the required mutual
exclusion function could be executed. Figure 9 shows the idea of using "Mutual Exclusion" calls.

```
Mutual Exclusion Begin
(Semaphore Setting)
    Critical Access Code
    Update Shared Variable
Mutual Exclusion End
(Semaphore Cleared)
```

MUTUAL EXCLUSION CODE

FIGURE 9

If each shared variable had a protecting semaphore the system should have a reliable method of sharing data. However as the number of shared variables grow so do the number of semaphores. One of the problems with too many semaphores, is that the global bus is busy with processors continually checking different semaphores, since the semaphores are in shared memory. Therefore, it is important to keep the number of semaphores to a minimum.

The section of code between the setting and clearing of the semaphore is known as the critical access code. The semaphore acts as a protection gate, only allowing one processor at a time to access the shared data or resource. This section of code should be executed in minimum time. By doing this a given processor will not tie up valuable resources for long periods of time. If other processors are waiting for this resource and cannot proceed with other tasks, then the power of the processor and the multiprocessor system is being wasted.

When there are several sections of code using semaphores to control access to shared variables, it may be worthwhile to include these into one module. This module would have some type of access control to the
critical section inside the module. This can reduce the number of semaphores required. The modules need to be self-sufficient (not requiring other resources) to help prevent deadlock. This module can be thought of as a monitor. The monitor protects shared variables and controls access to shared resources [19,23].

A monitor is basically a passive piece of code. It is only activated when a call is made to use one of the shared resources it is protecting. The monitor is responsible for scheduling the use of the shared resource, and notifying processors when the shared resource is available. The resources required by the monitor are unique to the monitor and can only be accessed by the monitor. Figure 10 presents the key concepts of a monitor.

<table>
<thead>
<tr>
<th>MONITOR FOR CONTROLLING ACCESS TO SHARED RESOURCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>* ONLY ACCESS TO SHARED RESOURCE VIA PUBLIC PROCEDURES</td>
</tr>
<tr>
<td>* USES SEMAPHORES -- MUTUAL EXCLUSION -- INDIVISIBLE INSTRUCTION</td>
</tr>
<tr>
<td>* SCHEDULING OF ACCESS TO SHARED RESOURCE</td>
</tr>
<tr>
<td>* LOCAL VARIABLES AND PROCEDURES FOR INTERNAL BOOKKEEPING</td>
</tr>
</tbody>
</table>

MONITOR CONCEPT

FIGURE 10

Reliable communications in a multiprocessor system can be established using the concepts of monitors and semaphores. Monitors and semaphores are used to protect the shared resources.

The need for a monitor arises when there are operations that could be executed at the same time. Suppose two processors want to read the same location at the same time. To avoid the deadlock of the faster
processor always reading and locking out the other processor there must be some type of scheduling method used.

A monitor also must schedule the resources in a multiprocessor system. Depending on the number and type of shared resources in a system, there may be more than one monitor. Shared resources can vary from a piece of hardware to the use of a section of software or a set of shared variables. There are many possible algorithms for the scheduling of resources.

Any algorithm needs to prevent the conditions which lead to deadlock. Each processor must get its turn to use the shared resources as requested. If a processor calls the monitor to use a shared resource and the resource is being used there must be a way to tell the processor to wait until the resource is available. It is still important to keep the critical access code fast.

The structure of the monitor requires some sort of waiting algorithm since it protects shared data. Only one process inside the monitor can be actively accessing shared data at any given time. Thus the basic structure of the monitor could be very similar to a section of critical code protected with semaphores, only now a scheduling algorithm has been added. See Figure 11.
So far the monitor discussed has only one shared resource to protect. It could have several shared resources to protect. Each would have a scheduling algorithm and mutual exclusion block.
CHAPTER FOUR

THE MMDE SYSTEM

The MMDE is implemented using the concept of monitors and semaphores. The heart of the system is the communication between processors. This was the first piece of software designed and tested. Since the time that it first worked, very few changes have been made.

The communication software is in a module known as BUFMON. This module is the shared buffer monitor. It is responsible for all of the communication operations between processors. This includes transferring information from a local buffer to the shared buffer and to a local buffer from the shared buffer, as well as an elementary scheduling algorithm for the waiting processors. The monitor is composed of two procedures which are accessible to the processors. All other procedures in the monitor are private to the monitor. This allows the use of the monitor in a controlled manner when calls are made to the monitor by a processor. BUFMON provides the MMDE system with a minimal system for interprocessor communication.

The monitor is accessible to any processor since its code is physically duplicated in each processor's memory. The monitor must respond to a call from its processor and transfer data into or out of shared memory. The monitor is a global shared resource. In turn the monitor acts as a manager of a shared resource, in this case the global memory located on the extended memory card. Only the monitor knows how to transfer data in and out of this memory area. In order for a program in a processor to use the shared memory it must make a call to one of the procedures in the monitor which are accessible to the processor.

There are two procedures a processor may call to use the shared memory resource. One copies data from the local memory of the calling
processor and produces a duplicate of the data in the global memory. As such it is known as PRODUCER. The second consumes the information in shared memory and transfers it to the calling processor’s local memory. It is known as CONSUMER. These two procedures support low level interprocessor communication. They do not support communication between tasks in a single processor.

Both of the procedures are similar. Depending on the procedure, the information required consists of processor identification, local buffer location, length of data, and the status pointer. The monitor then uses this information for scheduling and for updating local private variables. Also, each procedure does not return until it has finished its task. This ensures that calling processors can depend on reliable data transfer to or from global memory.

Internal to the buffer monitor there is a data structure into which the calling processor’s information is placed. The monitor adds a short header to the data which includes a description of the length of the message, and the identification of the sending processor and the receiving processor. In this way when another processor calls PRODUCER or CONSUMER the contents in shared memory can be interpreted properly.

When one of the procedures is called, the monitor is activated. Inside the monitor several other private procedures are then activated, one of which is the scheduling procedure. The complete interaction will first be explained by the way of an analogy.

The analogy unfolds in the local ice cream shop. Mr. A opens the door and walks into an ice cream shop to buy an ice cream cone. At this point there is a decision to make. If Mr. A is the only person in the ice cream shop, then he can go directly to the counter and ask for service. If there are other people in the shop then Mr. A must take a number from the ticket spool and wait for his turn. The ice cream shop
can only hold a certain number of people so the tickets are numbered accordingly. After the maximum number the numbers start over.

Suppose the first time Mr. A walks into the ice cream shop there is no one there, then Mr. A goes directly to the counter for his ice cream. However, Mr. A demands exclusive service and does not want to be interrupted so he turns over the sign that says "Please take a number". He does this before asking for service since he does not want any interruptions while being served. This protects his exclusive service, since the next customer could come in while he is being served. If a customer does come, then the customer will wait until their number is called.

Now Mr. A is at the counter demanding service. He informs the clerk he wants his order of ice cream. The clerk checks to see if Mr. A's ice cream is ready. The clerk then reports to Mr. A whether his ice cream is ready or not. If Mr. A's ice cream is ready then he takes the ice cream and prepares to leave. However just before leaving he lets the clerk know that he is done. On the way out of the ice cream shop he lets the next customer know that it is their turn. If there are no more customers Mr. A takes down the sign that says "Please take a number".

If Mr. A's ice cream is not ready, he could do several things. He could get mad and make a scene demanding that he will not go until he gets his ice cream. He could patiently wait at the end of the line and check to see if his ice cream is ready again after waiting for the other customers to go through the line. Since Mr. A is a kind person he chooses the later. After the other customers have checked on their ice cream then Mr. A checks again on his order.

The clerk in the ice cream shop is not always available, since she may have to refill ice cream containers. During the time she is gone, she hangs out a sign that lets her customers know that she is busy. Meanwhile customers can still come in and wait to be served. When she
is done refilling an ice cream container, she puts the sign away and goes back to serving the customers.

The customers coming to the shop correspond to the processors calling the procedure known as CONSUMER. Mr. A is going to consume the ice cream or in the case of the processor some data. Opening the door and checking the availability of being served is part of the scheduling algorithm. The scheduling algorithm is a very simple and straightforward modified FIFO (First In First Out). The first processor in gets service and is sent on if it has a message, otherwise the processor goes to the end of the line to wait until its turn comes up again, sort of a round-robin technique. This is done by the queue procedures which are local private procedures in BUFMON.

The filling of ice cream containers by the clerk corresponds to the processors calling the public procedure PRODUCER. The clerk also has the job of coordinating between the customer, emptying the container and filling the container. This coordination is handled by some internal private procedures. One of the procedures known as SIGNAL is responsible for all of the signs such as "Please take a number" and "Clerk Busy". These signs are implemented with the use of semaphores. The semaphores are implemented using the LOCKSET instruction. Another set of routines handle the scheduling algorithms. These include PRQUEUE, PRDEQUEUE, COQUEUE, CODEQUEUE.
Figures 12 and 13 show the flow diagrams of the two public procedures CONSUMER and PRODUCER. As far as a calling program is concerned, to transmit or receive a message, only PRODUCER or CONSUMER need to be called. None of the inner workings of the monitor need to be called or understood. In this way the monitor is very clean and
provides a well defined interface for communication. See Figure 14 for a block diagram of BUFMON.

A brief description of the private monitor procedures will be given. Detailed documentation of the procedures can be found in Appendix E.

**SIGNAL**

Signal is the procedure which handles the semaphores used by the queue routines. It also indicates when the shared buffer is busy. Signal is used to only set the semaphores. The semaphores are set using the indivisible PLM instruction LOCKSET. This instruction forces a special prefix to be added to the machine code instruction. At execution a bus lock signal is asserted. This forces a bus locking action which is supported by the circuitry of the CPU boards. The instruction checks a memory location for a zero, if it is a zero, then a one is written. If it is not a zero the next instruction is executed. In the BUFMON code LOCKSET is used in a DO WHILE loop which will exit
the loop only after a one is written. In this way the calling processor knows that it set the semaphore \([13]\). The using procedure clears the semaphore when it is done with the section of critical access code. Since the procedure "knows" it set the semaphore, it can clear the semaphore without using Signal. In this way the next processor waiting and checking the semaphore using LOCKSET can proceed.

**INITBUFMON**

This procedure is called once by the MMDE at program initialization to set the private local variables of BUFMON. The variables are used by BUFMON to keep track of the queues used in scheduling the calling processors.

**QUEUE ROUTINES**

PRQUEUE, PRDEQUEUE, COQUEUE, CODEQUEUE are local private procedures that handle the scheduling of the calling processors for PRODUCER and CONSUMER. Both CONSUMER and PRODUCER have separate queues to schedule the processors. Each procedure can be accessed concurrently. This allows the shared memory to be changed quickly.

**CMDMODE**

The monitor also controls access to the SDM card since it is a shared resource. This allows special control for loading user code and for the input and output of terminal messages.

BUFMON is the heart of the MMDE's communication system. There are two other pieces of software that are used in the MMDE system, DBUGMON and UEXT. DBUGMON is located with BUFMON in each of the EPROMS on the CPU boards. UEXT resides in local RAM on the SDM monitor board. These two code segments will be discussed next, along with an explanation of
the command flow and a brief command example. Details of the commands can be found in Appendix I. A sample user program with compiling and location instructions will be found in Appendix J.

**DEBUGMON**

DEBUGMON is the main program which coordinates the activities of the processors. Each processor contains the same code, so each processor can simultaneously run DEBUGMON. Going back to the analogy, each customer is the equivalent of a processor. When power is applied to the CPU cards the processor begins execution by jumping to the high end of memory to find the program start vector. This vector points to the main code sequence in DEBUGMON.

The first section of code initializes the various system tables in shared memory. It should be noted that processor zero is the only processor that is allowed to initialize these variables. The other processors wait until the basic initialization is done. After this each of the other processors identify themselves to the system. In this way the system is able to determine which processors are available.

After the initialization sequence the system is ready to accept commands from the user via the shared memory. Each processor calls CONSUMER to check for a message in shared memory. If a message is waiting for the processor then CONSUMER returns with the message copied to the local memory of the calling processor.

Once a command is received by the processor, it executes the command. DEBUGMON has many internal procedures to decode the command line and execute the commands. They are documented in Appendix F. Once each command has been executed the processor returns to check for its next command. During this sequence of events the other processors are also checking for their commands or messages. However to each processor, it appears as if the shared memory is exclusively theirs to
use. The coordination of processor requests for the use of the buffer is handled by BUFMON. This makes the use of the shared memory easy.

**UEXT**

UEXT resides in RAM on the SDM monitor board. At invocation of the MMDE system UEXT is loaded from the MMDE system disc into the SDM monitor board RAM by using the run command of the SDM86 monitor. Then a command "U" is executed by the user to start the MMDE system. The main purpose of UEXT is to act as an interface to the CRT terminal. UEXT uses CONSUMER and PRODUCER to transmit and receive messages from the CPU cards. Details of the startup procedures can be found in Appendix N.

The initialization sequence of UEXT involves clearing the CRT screen and displaying the MMDE system message. The next line on the display notifies the user which processors are active. Finally the MMDE system prompt is displayed. At this point the system is ready for a command from the user.

There are six basic commands that are supported by the MMDE system. The specifics of each command are described in Appendix I. The commands are shown in Figure 15.

L -- LOAD A USER WRITTEN PROGRAM
G -- EXECUTE A LOADED PROGRAM
X -- EXAMINE AND/OR CHANGE THE PROCESSOR'S REGISTERS
D -- DISPLAY MEMORY
S -- SUBSTITUTE MEMORY OR SET A BREAKPOINT
CNT E -- CONTROL E TO EXIT BACK TO THE SDM SYSTEM

**MMDE COMMANDS**

**FIGURE 15**
All of the commands can be entered in upper or lower case. Each command contains an identifier specifying which processor to use. For example, after a program has been executed and stops at a breakpoint the processor’s registers could be examined. The command would be entered as shown in Figure 16.

```
*>X0
<< Processor Identification
---------- Examine Register command
---------- MMDE System prompt
EXAMINE COMMAND ENTRY
FIGURE 16
```

The UEXT program then sends the command to Processor Zero using PRODUCER. Processor Zero then reads the command using CONSUMER. The command is then decoded and executed by Processor Zero. Processor Zero then uses PRODUCER to send the results of the command back to UEXT. UEXT using CONSUMER gets the information from the shared memory and displays the information to the screen. Figure 17 shows the display.

```
PROCESSOR 0 EXECUTING TASK
AX=0000 CS=0000 IP=0000 FL=0000
BX=0000 SS=0000 SP=0000 BP=0000
CX=0000 DS=0000 SI=0000
DX=0000 ES=0000 DI=0000

*>  
EXAMINE COMMAND REGISTER RESULTS
FIGURE 17
```

In addition to these commands a utility library has been developed that allows the user to exit from their program back into the MMDE.
With the MMDE laying the foundation, a multi-microprocessor operating system can now be developed. The MMDE allows code to be developed and then downloaded into RAM. This capability plus the debug tools allow an easy method for designing and developing code for a multi-microprocessor system.
CHAPTER FIVE

SUMMARY AND CONCLUSION

The MMDE system provides a low level tool for developing programs in a multi-microprocessor environment. This is done by providing a minimum set of utilities.

In particular these utilities are: examine/change memory, examine/change processor registers, load a program, set a breakpoint, and execute a program. In addition to these utilities an MMDE system library was developed that allows the user to exit from their program back to the MMDE system. The MMDE system requires the user to understand the Intel 8086 instruction set, but does not require an understanding of the inner workings of the MMDE system software.

The work done so far provides an important element necessary to develop an operating system. This is the communication software that allows interprocessor communication. BUFMON allows communication processes to operate in a very defined manner. BUFMON is a software monitor which defines a controlled access to shared memory resources. This software module was one of the most difficult to write. It is a very reliable piece of software.

The BUFMON code was written using the concepts of semaphores and monitors. Semaphores are required to control the use of the shared resources such as the shared memory. To implement the semaphores requires low level hardware support from the microprocessor. There must be at least one instruction that allows an indivisible read/test/write bus cycle. This is done in the MMDE by using the PLM86 instruction LOCKSET. The LOCKSET instruction is implemented by using the low level LOCK prefix instruction with the XCHG instruction. When the Intel 8086 LOCK NOT is activated the 8289 Bus Arbiter chip locks the bus until
completion of the instruction, which swaps two 8-bit bytes. This hardware support permits the use of semaphores in the MMDE system.

Using monitors and semaphores two public procedures CONSUMER and PRODUCER were implemented which support interprocessor communication. These procedures use semaphores internally to protect and grant access to the shared memory.

BUFMON allows several processors to be actively using its procedures at the same time. Processor A could be checking for a message using CONSUMER while Processor B is using PRODUCER to transmit a message to Processor A. Each processor "thinks" it has exclusive access to shared memory.

Even though BUFMON works very well it could be improved by building a message alert system. When a given processor has a message waiting an interrupt could be generated for that processor. The processor could then execute an interrupt routine that would service the alert. Later when the processor is ready it could respond to the message. This is like the idea of a call waiting light on the telephone system. This would require modification of the hardware as well as the system software.

Another improvement to BUFMON would be the addition of several shared buffers. This would allow several messages to be present in the buffer at the same time. An internal buffer scheduler would also need to be added to allocate the buffers as they are filled and emptied by CONSUMER and PRODUCER.

BUFMON could be modified to allow communication between processes within a single processor. This would be accommodated by changing the location of the shared buffer to on-board memory. This would keep the global bus free from the communication overhead of the intraprocessor
semaphore checking. Another modification would be to make the code reentrant so it could support a multi-tasking environment.

With the above mentioned changes it would be possible to have a local operating system on each processor, possibly supporting a terminal on its auxiliary port. Then when interprocessor communication was required, calls could be made to BUFMON.

The design of the system software allows up to five processors plus the SDM card. Currently only two processors plus the SDM card are operating. This restriction is primarily due to the serial interrupt timing of the Intel Multibus system. There is a parallel priority system that can be added with some hardware modifications [13]. The software of the system has been written with this in mind. Modification of the system to a parallel priority system should not require any changes to the MMDE system software.

One of the limiting factors of a global bus system is the bus. As more and more processors are added to the bus, the bus becomes loaded down with communication overhead as processors compete for shared resources. One possible solution is to dedicate the auxiliary ports of the CPU cards to the task of controlling the shared resources. This would alleviate the global bus from the competition for shared resources. The Sequent computer system uses a serial bus for shared resource communication [8].

The goal of the MMDE system was to provide the user a minimum set of tools to write, execute and debug programs in a multi-microprocessor environment. This has been accomplished.
BIBLIOGRAPHY


12. iSDM86 System Debug Monitor Reference Manual, Order No. 146165-001
   Intel Corporation, Literature Department, Intel Corporation, 3065
   Bowers Avenue, Santa Clara, California 95051.

13. iSBC 86/12A Single Board Computer Hardware Reference Manual, Order
    No. 9803074-02. Literature Department, Intel Corporation, 3065
    Bowers Avenue, Santa Clara, California 95051.

14. ISIS-II (CRT-Based Text Editor) User's Guide Order No. 980902-003
    Intel Corporation, Literature Department, Intel Corporation, 3065
    Bowers Avenue, Santa Clara, California 95051.

    Order No. 9800466-03 Rev C. Literature Department, Intel
    Corporation. 3065 Bowers Avenue, Santa Clara, California 95051.

16. PL/M Compiler Operating Instructions for 8086/8085 Based Development
    Systems. Order No. 9800478-04 Rev. D, Literature Department, Intel
    Corporation, 3065 Bowes Avenue, Santa Clara, California 95051.

17. 8086 Family Utilities User's Guide for 8086/8085 Based Development
    System, Order No. 9800639-04. Literature Department, Intel
    Corporation, 3065 Bowers Avenue, Santa Clara, California 95051.

    Literature Department, Intel Corporation, 3065 Bowers Avenue, Santa
    Clara, California 95051.

    Concurrent Programming with Operating Systems Applications,

    0-13-566878-6 (Motorola Edition).

21. Series 32000 Databook, Order No. LN/EE/1275, RRD-RRD21M114
    National Semiconductor Corporation, 2900 Semiconductor Drive,
    Santa Clara, California 95051.

22. iAPX86,86 User's Manual, August 1981, Literature Department, Intel
    Corporation, 3065 Bowers Avenue, Santa Clara, California 95051
## APPENDIX A

### SWITCH SETTINGS OF SDM CARD

<table>
<thead>
<tr>
<th>JUMPERS (1-4)</th>
<th>DESC.</th>
<th>JUMPERS (5-8)</th>
<th>DESC.</th>
<th>JUMPERS (9-12)</th>
<th>DESC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 --</td>
<td>51 --</td>
<td>101 --</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 --</td>
<td>52 --</td>
<td>102 --</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 --</td>
<td>53 --</td>
<td>103 -- 104</td>
<td>CONST CLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 --</td>
<td>54 -- 55</td>
<td>105 -- 106</td>
<td>BUS CLOCK</td>
<td>BUS CLOCK</td>
<td></td>
</tr>
<tr>
<td>5 -- 6</td>
<td>TIME OUT</td>
<td>57 -- 56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 -- 5</td>
<td>OPTION</td>
<td>58 -- 59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 -- 8,10</td>
<td></td>
<td>60 -- 61</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 -- 7,10</td>
<td></td>
<td>62 -- 63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 --</td>
<td></td>
<td>64 -- 65</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 -- 7,8</td>
<td></td>
<td>66 -- 67</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 --</td>
<td></td>
<td>68 -- 69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 -- 21</td>
<td>TEST GND</td>
<td>70 -- 71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 -- 27</td>
<td></td>
<td>72 -- 73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 -- 30</td>
<td></td>
<td>74 --</td>
<td>PIC IR7</td>
<td>124 --</td>
<td></td>
</tr>
<tr>
<td>15 -- 9,10</td>
<td></td>
<td>75 -- 76</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 -- 15</td>
<td></td>
<td>77 -- 78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 -- 18</td>
<td></td>
<td>79 -- 80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 -- 17,25,31</td>
<td></td>
<td>81 -- 82</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 --</td>
<td></td>
<td>83 -- 84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 -- 33</td>
<td></td>
<td>85 -- 86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 -- 12</td>
<td>TEST GND</td>
<td>87 -- 88</td>
<td>GROUND</td>
<td>137 --</td>
<td></td>
</tr>
<tr>
<td>22 --</td>
<td></td>
<td>89 -- 90</td>
<td>NMI TO GND</td>
<td>139 --</td>
<td></td>
</tr>
<tr>
<td>23 --</td>
<td></td>
<td>91 -- 92</td>
<td>8086 LOCAL</td>
<td>142 --</td>
<td></td>
</tr>
<tr>
<td>24 -- 35</td>
<td></td>
<td>93 -- 94</td>
<td>CLOCK</td>
<td>143 -- 144</td>
<td></td>
</tr>
<tr>
<td>25 -- 17,25,31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 -- 13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28 -- 29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29 -- 28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 -- 14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 -- 17,18,25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33 -- 20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34 --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35 -- 24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37 --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38 --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39 -- 40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40 -- 39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41 --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42 -- 43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43 -- 42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44 --</td>
<td>94 -- 96</td>
<td>2732 EPROM</td>
<td>144 -- 143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45 --</td>
<td>95 --</td>
<td>145 --</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46 --</td>
<td>96 -- 94</td>
<td>2732 EPROM</td>
<td>146 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47 --</td>
<td>97 -- 99</td>
<td>2732 EPROM</td>
<td>147 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48 --</td>
<td>98 --</td>
<td>148 --</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49 --</td>
<td>99 -- 97</td>
<td>2732 EPROM</td>
<td>149 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 --</td>
<td>100 --</td>
<td>150 --</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>151 -- 152 BUS PRI.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>152 -- 151 BUS PRI.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SWITCH S1**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-16 CLOSED</td>
<td>W1 OPENED</td>
<td>W9 A-C</td>
</tr>
<tr>
<td>2-15 CLOSED</td>
<td>W2 OPENED</td>
<td>W10 A-B</td>
</tr>
<tr>
<td>3-14 OPENED</td>
<td>W3 OPENED</td>
<td>W11 A-B</td>
</tr>
<tr>
<td>4-13 OPENED</td>
<td>W4 A-B</td>
<td></td>
</tr>
<tr>
<td>5-12 CLOSED</td>
<td>W5 A-B</td>
<td></td>
</tr>
<tr>
<td>6-11 CLOSED</td>
<td>W6 A-B</td>
<td></td>
</tr>
<tr>
<td>7-10 OPENED</td>
<td>W7 A-C</td>
<td></td>
</tr>
<tr>
<td>8-09 CLOSED</td>
<td>W8 A-D</td>
<td></td>
</tr>
</tbody>
</table>
# APPENDIX B

## SWITCH SETTINGS OF CPU CARD

<table>
<thead>
<tr>
<th>JUMPERS</th>
<th>DESC.</th>
<th>JUMPERS</th>
<th>DESC.</th>
<th>JUMPERS</th>
<th>DESC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 --</td>
<td></td>
<td>51 --</td>
<td></td>
<td>101 --</td>
<td></td>
</tr>
<tr>
<td>2 --</td>
<td></td>
<td>52 --</td>
<td></td>
<td>102 --</td>
<td></td>
</tr>
<tr>
<td>3 --</td>
<td></td>
<td>53 --</td>
<td></td>
<td>103 --</td>
<td>CONST CLK</td>
</tr>
<tr>
<td>4 --</td>
<td></td>
<td>54 -- 55</td>
<td></td>
<td>104 --</td>
<td>CONST CLK</td>
</tr>
<tr>
<td>5 --</td>
<td></td>
<td>55 -- 54</td>
<td></td>
<td>105 --</td>
<td>BUS CLOCK</td>
</tr>
<tr>
<td>6 --</td>
<td></td>
<td>56 --</td>
<td></td>
<td>106 --</td>
<td>BUS CLOCK</td>
</tr>
<tr>
<td>7 -- 8,10</td>
<td></td>
<td>57 -- 56</td>
<td></td>
<td>107 --</td>
<td></td>
</tr>
<tr>
<td>8 -- 7,10</td>
<td></td>
<td>58 --</td>
<td></td>
<td>108 --</td>
<td></td>
</tr>
<tr>
<td>9 --</td>
<td></td>
<td>59 -- 60</td>
<td></td>
<td>109 --</td>
<td></td>
</tr>
<tr>
<td>10 -- 7,8</td>
<td></td>
<td>60 -- 59</td>
<td></td>
<td>110 --</td>
<td></td>
</tr>
<tr>
<td>11 --</td>
<td></td>
<td>61 --</td>
<td></td>
<td>111 --</td>
<td></td>
</tr>
<tr>
<td>12 -- 21,25 NO NPX</td>
<td></td>
<td>62 --</td>
<td></td>
<td>112 --</td>
<td></td>
</tr>
<tr>
<td>13 -- 14</td>
<td></td>
<td>63 --</td>
<td></td>
<td>113 --</td>
<td></td>
</tr>
<tr>
<td>14 -- 13</td>
<td></td>
<td>64 --</td>
<td></td>
<td>114 --</td>
<td></td>
</tr>
<tr>
<td>15 -- 16</td>
<td></td>
<td>65 --</td>
<td></td>
<td>115 --</td>
<td></td>
</tr>
<tr>
<td>16 -- 15</td>
<td></td>
<td>66 --</td>
<td></td>
<td>116 --</td>
<td></td>
</tr>
<tr>
<td>17 -- 18</td>
<td></td>
<td>67 --</td>
<td></td>
<td>117 --</td>
<td></td>
</tr>
<tr>
<td>18 -- 17</td>
<td></td>
<td>68 -- 76</td>
<td></td>
<td>118 --</td>
<td></td>
</tr>
<tr>
<td>19 -- 20</td>
<td></td>
<td>69 --</td>
<td></td>
<td>119 --</td>
<td></td>
</tr>
<tr>
<td>20 -- 19</td>
<td></td>
<td>70 --</td>
<td></td>
<td>120 --</td>
<td></td>
</tr>
<tr>
<td>21 -- 12,25 NO NPX</td>
<td></td>
<td>71 --</td>
<td></td>
<td>121 --</td>
<td></td>
</tr>
<tr>
<td>22 --</td>
<td></td>
<td>72 --</td>
<td></td>
<td>122 --</td>
<td></td>
</tr>
<tr>
<td>23 --</td>
<td></td>
<td>73 --</td>
<td></td>
<td>123 --</td>
<td></td>
</tr>
<tr>
<td>24 -- 35</td>
<td></td>
<td>74 -- PIC IR7</td>
<td>124 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 -- 12,21 NO NPX</td>
<td></td>
<td>75 -- IR6</td>
<td>125 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 -- 27</td>
<td></td>
<td>76 -- 68</td>
<td>IR5</td>
<td>126 --</td>
<td></td>
</tr>
<tr>
<td>27 -- 26</td>
<td></td>
<td>77 -- IR4</td>
<td>127 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28 -- 29</td>
<td></td>
<td>78 -- IR3</td>
<td>128 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29 -- 28</td>
<td></td>
<td>79 -- 83</td>
<td>IR2</td>
<td>129 -- 130 SURRENDER</td>
<td></td>
</tr>
<tr>
<td>30 -- 31</td>
<td></td>
<td>80 -- SEE NOTE IR1</td>
<td>130 -- 129 BUS MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 -- 30</td>
<td></td>
<td>81 -- PIC IR0</td>
<td>131 --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 -- 33</td>
<td></td>
<td>82 --</td>
<td></td>
<td>132 --</td>
<td></td>
</tr>
<tr>
<td>33 -- 32</td>
<td></td>
<td>83 -- 79</td>
<td></td>
<td>133 --</td>
<td></td>
</tr>
<tr>
<td>34 --</td>
<td></td>
<td>84 --</td>
<td></td>
<td>134 --</td>
<td></td>
</tr>
<tr>
<td>35 -- 24</td>
<td></td>
<td>85 --</td>
<td></td>
<td>135 --</td>
<td></td>
</tr>
<tr>
<td>36 --</td>
<td></td>
<td>86 --</td>
<td></td>
<td>136 --</td>
<td></td>
</tr>
<tr>
<td>37 --</td>
<td></td>
<td>87 -- 89</td>
<td>GROUND</td>
<td>137 --</td>
<td></td>
</tr>
<tr>
<td>38 --</td>
<td></td>
<td>88 --</td>
<td></td>
<td>138 --</td>
<td></td>
</tr>
<tr>
<td>39 -- 40</td>
<td></td>
<td>89 -- 87</td>
<td>NMI TO GND</td>
<td>139 --</td>
<td></td>
</tr>
<tr>
<td>40 -- 39</td>
<td></td>
<td>90 --</td>
<td></td>
<td>140 --</td>
<td></td>
</tr>
<tr>
<td>41 --</td>
<td></td>
<td>91 -- 80</td>
<td>TIMER1/IR1</td>
<td>141 --</td>
<td></td>
</tr>
<tr>
<td>42 -- 43</td>
<td></td>
<td>92 -- 93</td>
<td>8086 LOCAL</td>
<td>142 --</td>
<td></td>
</tr>
<tr>
<td>43 -- 42</td>
<td></td>
<td>93 -- 92</td>
<td>CLOCK</td>
<td>143 -- 144 CBRQ TO</td>
<td></td>
</tr>
</tbody>
</table>
SWITCH S1

PADS

1-16 CLOSED W1 OPENED W9 A-C
2-15 OPENED W2 OPENED W10 A-B
3-14 CLOSED W3 OPENED W11 A-B
4-13 CLOSED W4 A-B
5-12 CLOSED W5 A-B
6-11 CLOSED W6 A-B
7-10 OPENED W7 A-C
8-09 CLOSED W8 A-D

NOTE:

FOLLOWING CHANGES FOR INTERRUPTS ON DIFFERENT CPU BOARDS

CPU 0 JUMPER BETWEEN 91 AND 81
CPU 1 JUMPER BETWEEN 91 AND 80
CPU 2 JUMPER BETWEEN 91 AND 79
## APPENDIX C

### MEMORY MAP OF CPU CARD

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>ABSOLUTE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F000:FFFF</td>
<td>FFFFF</td>
<td>BOOT STRAP ADDRESS</td>
</tr>
<tr>
<td>F000:FFE0</td>
<td>FFFFO</td>
<td>GOTO STARTT - DEBUG MONITOR</td>
</tr>
<tr>
<td>F000:FFE0</td>
<td>FFFEF</td>
<td>DEBUG MONITOR CODE</td>
</tr>
<tr>
<td>F000:E000</td>
<td>FE000</td>
<td>4 - 2716'S SOME CODE</td>
</tr>
<tr>
<td>F000:7FF0</td>
<td>F7FFF</td>
<td>SPACE LEFT</td>
</tr>
<tr>
<td>0000:807E</td>
<td>0807E</td>
<td>SHARED MEMORY SPACE</td>
</tr>
<tr>
<td>0000:807D</td>
<td>0807D</td>
<td>OFF BOARD RAM -- NOT USED</td>
</tr>
<tr>
<td>0000:8000</td>
<td>08000</td>
<td>COMMON MEMORY SPACE</td>
</tr>
<tr>
<td>0000:7FFF</td>
<td>07FFF</td>
<td>OFF BOARD RAM -- USED</td>
</tr>
<tr>
<td>0000:7F00</td>
<td>07F00</td>
<td>BY THE MMDE SYSTEM</td>
</tr>
<tr>
<td>0000:7EFF</td>
<td>07EFF</td>
<td>STACK AREA FOR UEXT CODE</td>
</tr>
<tr>
<td>0000:0800</td>
<td>00800</td>
<td>USER CODE AREA</td>
</tr>
<tr>
<td>0000:07FF</td>
<td>007FF</td>
<td>ON BOARD RAM</td>
</tr>
<tr>
<td>0000:0400</td>
<td>00400</td>
<td>INTEL RESERVED AREA</td>
</tr>
<tr>
<td>0000:03FF</td>
<td>003FF</td>
<td>INTERRUPT VECTOR SPACE</td>
</tr>
<tr>
<td>0000:0000</td>
<td>00000</td>
<td></td>
</tr>
</tbody>
</table>
## APPENDIX D

### MEMORY MAP OF SDM CARD

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>ABSOLUTE CS:IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>F000:FFFF FFFFFF</td>
<td>BOOT STRAP ADDRESS FOR SDM86 MONITOR</td>
</tr>
<tr>
<td>F000:FFFF FFFFO0</td>
<td>MONITOR EPROM -- SDM86 DEFAULT CONFIGURATION 4 - 2732'S</td>
</tr>
<tr>
<td>F000:FFE FFFEFE</td>
<td></td>
</tr>
<tr>
<td>F000:E000 FE000</td>
<td></td>
</tr>
<tr>
<td>F000:7FFF F7FFF</td>
<td></td>
</tr>
<tr>
<td>0000:807E 0807E</td>
<td>SHARED MEMORY SPACE</td>
</tr>
<tr>
<td>0000:807D 0807D</td>
<td>OFF BOARD RAM -- NOT USED</td>
</tr>
<tr>
<td>0000:8000 08000</td>
<td></td>
</tr>
<tr>
<td>0000:7FFF 07FFF</td>
<td></td>
</tr>
<tr>
<td>0000:7F00 07F00</td>
<td>UEXT PROGRAM STACK AREA</td>
</tr>
<tr>
<td>0000:7EFF 07EFF</td>
<td></td>
</tr>
<tr>
<td>0000:0800 00800</td>
<td>UEXT PROGRAM SPACE</td>
</tr>
<tr>
<td>0000:07FF 007FF</td>
<td>ON BOARD RAM</td>
</tr>
<tr>
<td>0000:0400 00400</td>
<td>SDM86 RAM DATA SPACE</td>
</tr>
<tr>
<td>0000:03FF 003FF</td>
<td></td>
</tr>
<tr>
<td>0000:0000 00000</td>
<td>INTERRUPT VECTOR SPACE</td>
</tr>
</tbody>
</table>
APPENDIX E

BUFMON PROCEDURES

Each of the following procedures are listed in the source code file BUFMON.SRC on the MMDE system disc.

<table>
<thead>
<tr>
<th>PROCEDURES</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD$MODE</td>
<td>USED TO SET THE OPERATING MODE OF UEXT</td>
</tr>
<tr>
<td>CODEQUEUE</td>
<td>DEQUEUES THE CONSUMER QUEUE AS IT IS USED</td>
</tr>
<tr>
<td>CONSUMER</td>
<td>USED TO EMPTY THE SHARED MESSAGE BUFFER</td>
</tr>
<tr>
<td>COQUEUE</td>
<td>QUEUES THE CONSUMER QUEUE AS IT IS REQUESTED</td>
</tr>
<tr>
<td>INIT$BUF$MON</td>
<td>INITIALIZES THE SEMAPHORES AND SYSTEM VARIABLES</td>
</tr>
<tr>
<td>PRDEQUEUE</td>
<td>DEQUEUES THE PRODUCER QUEUE AS IT IS USED</td>
</tr>
<tr>
<td>PRODUCER</td>
<td>USED TO FILL THE SHARED MESSAGE BUFFER</td>
</tr>
<tr>
<td>PRQUEUE</td>
<td>QUEUES THE PRODUCER QUEUE AS IT IS REQUESTED</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>USED TO SET THE SEMAPHORES</td>
</tr>
</tbody>
</table>

CMD$MODE

Changes the operation of the UEXT program on the SDM card. The change is allowed when the mode byte is in the not busy state and the calling processor is the current active processor.

Activation: Call CMD$MODE(CPU$ID,MODE,NUMMES,STATUS)

CPU$ID -- hardware identification of calling processor
MODE    -- 0 - output message to terminal
          1 - command edit
          2 - not busy state
          3 - load user code
NUMMES   -- number of messages to output to terminal
STATUS   -- error pointer  0 - no error  1 - error
CODEQUEUE

Updates the Consumer queue to allow the next user to use the Consumer routine. This procedure is called by Consumer just before it exits to the calling processor main program code. Semaphore3 is used to control updating of the queue.

Activation: Call CODEQUEUE

CONSUMER -- PUBLIC PROCEDURE

Moves a message from shared memory to the calling processor's local memory buffer.

Activation:

    Call Consumer(RCVR$CPU,SENDER$CPU,@MES$PTR,@NUM$CHAR,@STATUS)

RCVR$CPU -- cpu identification of calling processor
SENDER$CPU -- identification of processor that sent message
(not used)
@MES$PTR -- pointer to local buffer for storage of message
@NUMCHAR -- pointer to store number of characters in message
@STATUS -- pointer for error

COQUEUE

Updates Consumer queue and puts calling processor in sequence. Calling processor is forced to wait until its turn before using Consumer. Consumer calls this procedure before executing any message transfers. Semaphore3 is used to control the updating of the queue.

Activation: Call COQUEUE
INIT$BUF$MON

Sets the system semaphores and queue flags used by Consumer and
Producer. The following parameters are set:

SEMAPHORE, SEMAPHORE1, SEMAPHORE2, SEMAPHORE3, COQUEUE$BUSY
PR$QUEUE$BUSY, SIGNAL$BUSY, PRHEAD, PRTAIL, PRCOUNT, COHEAD,
COTAIL, COCOUNT = 0

MODE.CMD = 1

Activation: Call INIT$BUF$MON

PRDEQUEUE

Updates the Producer queue to allow the next user to use the
Producer routine. This procedure is called by Producer just before it
exits to the calling processor main program code. Semaphore2 is used to
control updating of the queue.

Activation: Call PRDEQUEUE

PRODUCER -- PUBLIC PROCEDURE

Produces a message in the shared buffer. The message is
transferred from the calling processor’s local memory to the shared
memory buffer. The procedure returns only after successfully producing
the message in the shared buffer.

Activation:
Call Producer(SOURCE$CPU, DEST$CPU, @MES$PTR, NUM$CHAR, @STATUS)

SOURCE$CPU -- cpu identification of calling processor
DEST$CPU -- cpu identification of processor where message
is being sent
MES$PTR -- pointer to address where local message is located
NUM$CHAR -- the ordinal number of characters/bytes in a message
STATUS -- pointer to where error message is stored

PRQUEUE

Updates Producer queue and puts calling processor in sequence. Calling processor is forced to wait until its turn before using Producer. Producer calls this procedure before executing any message transfers.

Activation: Call PRQUEUE

SIGNAL

Sets the semaphores as requested by the calling programs in the buffer monitor. The semaphores are used for mutual exclusion to system resources. This procedure uses the LOCK$SET instruction to accomplish the task.

Activation: Call Signal(Cpuid,Buffer$Status)

  Cpuid -- unique hardware identification
  Buffer$Status -- 0 - shared buffer ready to use -- Semaphore
                 1 - shared buffer busy -- Semaphore
                 2 - not used
                 3 - changing Producer queue -- Semaphore2
                 4 - changing Consumer queue -- Semaphore3

Semaphore Condition -- 1 lock
  0 unlock
APPENDIX F

DEBUG PROCEDURES

Each of the following procedures are listed in the source code file DEBUG.SRC on the MMDE system disc.

<table>
<thead>
<tr>
<th>PROCEDURES</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII$2$HEX</td>
<td>ASCII WORD TO HEXADECIMAL BYTE</td>
</tr>
<tr>
<td>ASCII$2$HEXWORD</td>
<td>FOUR ASCII CHARACTERS TO HEXADECIMAL WORD</td>
</tr>
<tr>
<td>ASCII$DECIMAL$2$HEX</td>
<td>ASCII DECIMAL SYMBOLS TO HEXADECIMAL</td>
</tr>
<tr>
<td>CHANGE$REG</td>
<td>CHANGE CONTENTS OF CPU REGISTERS</td>
</tr>
<tr>
<td>CHG$LOAD$MODE</td>
<td>CONTROLS DOWNLOADING OF CODE</td>
</tr>
<tr>
<td>CMD$MODE</td>
<td>CONTROLS MODE OF UEXT ON SDM BOARD</td>
</tr>
<tr>
<td>CMD$PROC</td>
<td>MESSAGE OF WHICH PROCESSOR IS ACTIVE</td>
</tr>
<tr>
<td>CONVERT$DATA</td>
<td>CALLS ASCII CONVERSION ROUTINES</td>
</tr>
<tr>
<td>CPU$ORDER</td>
<td>USED AT INITIALIZATION WHO IS HERE</td>
</tr>
<tr>
<td>DECODE</td>
<td>DECODE COMMAND LINE</td>
</tr>
<tr>
<td>ERROR</td>
<td>SYSTEM FAULT -- NEVER SHOULD OCCUR</td>
</tr>
<tr>
<td>GET$MORE</td>
<td>CHECKS FOR MORE CODE TO DOWNLOAD</td>
</tr>
<tr>
<td>GET$REG</td>
<td>GETS ASCII NAME OF CPU REGISTER</td>
</tr>
<tr>
<td>HEX$2$ASCII</td>
<td>CONVERTS HEXADECIMAL TO ASCII SYMBOLS</td>
</tr>
<tr>
<td>INDEX</td>
<td>DETERMINES REGISTER NAME</td>
</tr>
<tr>
<td>INIT$CPU$LIST</td>
<td>Initializes system variables</td>
</tr>
<tr>
<td>INIT$3</td>
<td>BREAKPOINT INTERRUPT ROUTINE</td>
</tr>
<tr>
<td>LINE</td>
<td>USED TO DISPLAY CPU REGISTERS</td>
</tr>
<tr>
<td>LOAD</td>
<td>DOWNLOAD CODE</td>
</tr>
<tr>
<td>MAKE$WORD</td>
<td>CONVERTS TWO BYTES INTO ONE WORD</td>
</tr>
<tr>
<td>NON$SPACE</td>
<td>HELPS TO DECODE COMMAND LINE</td>
</tr>
<tr>
<td>OUT$MEMORY</td>
<td>DISPLAYS MEMORY LOCATIONS</td>
</tr>
<tr>
<td>REG$ID</td>
<td>DECODE REGISTER NAME</td>
</tr>
<tr>
<td>RUN</td>
<td>EXECUTE DOWNLOADED CODE</td>
</tr>
</tbody>
</table>
SAVE$USER$REGISTERS
SET$MY$TURN
SET$NEXT$TURN
SHOW$MEMORY
SHOW$REGISTER
SPACE
SPURIOUS$INTERRUPT
SUBSTITUTE

SAVE USERS REGISTERS
CONTROLS ACCESS TO SDM CARD
CONTROLS ACCESS TO SDM CARD
USED TO DISPLAY MEMORY
DISPLAYS CPU REGISTERS
DECODES COMMAND LINE
HANDLES SPURIOUS INTERRUPTS
EXECUTES SUBSTITUTE COMMAND

ASCII$2$HEX

Converts a word containing two ASCII characters to one hexadecimal byte. Then it computes a running checksum on each conversion. The checksum is returned as a byte. If checksum equals zero before ASCII$2$HEX is called the returned value is the hexadecimal value.

Example:

ASCII symbols 5A
ASCII word 3541 -- 0011010101000001
Hexadecimal byte 5A -- 01011010

Activation: I = ASCII$2$HEX(ASCIIWORD)

ASCIIWORD -- 16 bits
15-8 First character
7-0 Second character
I -- byte for return value of checksum

ASCII$2$HEX$WORD

Converts four ASCII hexadecimal numeric characters into one hexadecimal word. The word is then returned.

Example:

ASCII symbols 9F3A 00111001 01000110 00110011 01000001
hexadecimal word 9F3A 1001111100111010

Activation: I = ASCII$2$HEX$WORD(BUF,STATUS)
BUF -- pointer to the first character to be converted
    each character is a byte in the buffer.
STATUS -- 0 error on conversion illegal hexadecimal digit
I
I -- word

ASCII$DECIMAL$2$HEX

Converts an ASCII number symbol to a hexadecimal number.

Example:   ASCII symbols  99
            ASCII hex     3939
            Hexadecimal  63

Activation: I = ASCII$DECIMAL$2$HEX(BUFFER,NUM$CHAR,STATUS)
            I -- word
            BUFFER -- address of ASCII symbols to convert
            NUMCHAR -- number of characters to convert maximum 4
            STATUS -- address of error message

CHANGE$REG

Changes the register indicated in the X command with the value in
the X command. The user registers being changed are stored in the
DEBUG$COPY$OF$USER$REGISTERS array.

Activation: Call CHANGE$REG(REG$ID)
            REG$ID -- identification of register to be changed

CHG$LOAD$MODE

Used during the load command to direct the UEXT program when to
download more code. The mode is only changed when UEXT is not busy.
Activation: Call CHG$LOAD$MODE(MODE)

MODE -- 0 - get more code
       1 - load done - no error
       2 - load done - error
       3 - wait for next mode (not busy)
       4 - buffer error
       5 - write error

CMD$MODE

Changes the operation of the UEXT program on the SDM card. The change is allowed when the mode byte is in the not busy state and the calling processor is the current active processor.

Activation: Call CMD$MODE(CPU$ID,MODE,NUMMES,STATUS)

CPU$ID -- hardware identification of calling processor
MODE -- 0 - output message to terminal
       1 - command edit
       2 - not busy state
       3 - load user code
NUMMES -- number of messages to output to terminal
STATUS -- error pointer 0 - no error 1 - error

CMD$PROC

CMD$PROC is used to output the processor executing task message to the terminal.

Activation: Call CMD$PROC
CONVERT$DATA

Used in procedure DECODE to call the appropriate conversion routine during the decoding of a command line. This is a local procedure to DECODE.

Activation: Call CONVERT$DATA(TYPE,POS,NUMBYTES,STATUS)

TYPE -- 0- ASCII address to hexadecimal
      1- ASCII numbers to hexadecimal
POS -- character position in command line
NUMBYTES -- number of bytes to convert
STATUS -- address to store error message

CPU$ORDER

Sets the array CPU$LIST.ORDER to a valid state which indicates which processors are alive and well. This is done by checking CPU$LIST.SYS$CPU for OFFH. If OFFH is not found then the CPUID is set in the CPU$LIST.SYS$CPU.

Activation: Call CPU$ORDER

DECODE

Decodes the address and number of memory locations in the D command.

Activation: Call DECODE(ERROR)

ERROR -- 0 - no error
        1 - error invalid input
ERROR

Outputs error message to SDM CPU when called.

Activation: Call ERROR

GET$MORE

Loads more code into the local buffer if needed. It is used in conjunction with CHG$LOAD$MODE. If no more code is needed the buffer position is advanced by one.

Activation Call GET$MORE

GET$REG

Used to view a user register by the X command.

Activation: Call GET$REG(REG$ID)

REG$ID - identification of register

HEX$2$ASCII

Converts one hexadecimal word into 4 ASCII characters or converts one byte into 2 ASCII characters.

Activation: Call HEX$2$ASCII(HEXWORD,@BUFFER,BYTEWORD)

HEXWORD -- the hexadecimal word or byte to convert
BUFFER -- pointer to memory location of conversion results
BYTEWORD -- 0 - convert byte
1 - convert hexadecimal word

Example:
INDEX

Returns contents of user register based on register name in X command.

Activation: \( I = \text{INDEX}(\text{REG$NAME}) \)

\( \text{REG$NAME} \) -- identification of user register
\( I \) -- word

INIT$CPU$LIST

Initializes system level global variables in the CPU$LIST structure array. This is done at the beginning of the MMDE system code in the DEBUG module.

Activation: Call INIT$CPU$LIST

INIT3

This is an interrupt routine that is executed during a breakpoint instruction. The routine calls other routines to save the users registers for later use.

Activation: Done when the breakpoint opcode is executed "CC"
LINE

Displays the lines of user registers to the terminal.

Activation: Call LINE(LINE$NUMBER)
  LINE$NUMBER -- 0 - displays AX,CS,IP,FL
  1 - displays BX,SS,SP,BP
  2 - displays CX,DS,SI
  3 - displays DX,ES,DI

LOAD

Loads a user program into the designated local processor’s memory. The user program must be converted to hex format using the OH86 program from Intel.

Activation: Call Load

MAKE$WORD

Combines two bytes into one word. The two bytes must be located in the LOC$BUF array. This routine is used by LOAD.

Activation: I = MAKE$WORD
  I -- word

NON$SPACE

Used to parse command line. Finds end of character substring. This procedure is used by procedure DECODE. It is a local procedure of procedure DECODE.

Activation: I = NON$SPACE(POS,STATUS)
I = byte
POS = byte position in command line
STATUS = address where to put error message

OUT$MEMORY

Displays the contents of memory as requested by the D command. This routine also generates the addresses to be displayed.

Activation: Call OUT$MEMORY

REG$ID

Converts the register name in a command to a number.

Activation: I = REG$ID
  I = byte

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>REGISTER</th>
<th>NUMBER</th>
<th>REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BP</td>
<td>7</td>
<td>DS</td>
</tr>
<tr>
<td>1</td>
<td>DI</td>
<td>8</td>
<td>ES</td>
</tr>
<tr>
<td>2</td>
<td>SI</td>
<td>9</td>
<td>IP</td>
</tr>
<tr>
<td>3</td>
<td>BX</td>
<td>10</td>
<td>CS</td>
</tr>
<tr>
<td>4</td>
<td>DX</td>
<td>11</td>
<td>FL</td>
</tr>
<tr>
<td>5</td>
<td>CX</td>
<td>12</td>
<td>SS</td>
</tr>
<tr>
<td>6</td>
<td>AX</td>
<td>13</td>
<td>SP</td>
</tr>
<tr>
<td></td>
<td>ERROR</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

RUN

Executes the user code stored in memory. Any changes to the users registers are loaded before execution. The routine executes machine level opcodes that are stored as data. This is necessary in order to pass control to the user code.
Activation: Call RUN

SAVE$USER$REGISTERs

The user's registers are saved following execution of the breakpoint instruction.

Activation: Call SAVE$USER$REGISTERs

SET$MY$TURN

Sets requesting processor's turn. The calling processor is forced to wait until the turn is granted by the current executing processor. The global shared variable TRN$LCK is used as the semaphore to control the processor's turn. After the turn is granted, then the shared global variable ACT$CPU is set to the calling processor's CPU$ID.

Activation: Call SET$MY$TURN

SET$NEXT$TURN

Releases the global shared variable TRN$LCK so that a waiting processor can execute its turn.

Activation: Call SET$MY$TURN

SHOW$MEMORY

Performs the memory display command.

Activation: Call SHOW$MEMORY
SHOW$REGISTER

Performs the display register command.

Activation: Call SHOW$REGISTER

SPACE

Used to parse command line. Finds start of character substring. This procedure is used by procedure DECODE. It is a local procedure of procedure DECODE.

Activation: I = SPACE(POS,STATUS)
I = byte
POS = byte position in command line
STATUS = address where to put error message

SPURIOUS$INTERRUPT

Handles any noise generated interrupts in the system. Resets system to STARTT.

SUBSTITUTE

Executes the S command. When breakpoint option is exercised the user code at the selected address is replaced with 'CC' hexadecimal. This is the code for a breakpoint interrupt in the Intel 8086 hardware.

Activation: Call SUBSTITUTE
## APPENDIX G

### UEXT PROCEDURES

Each of the following procedures are listed in the source code file UEXT.SRC on the MMDE system disc.

<table>
<thead>
<tr>
<th>PROCEDURES</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL$CMD</td>
<td>ACTIVATES RUN ALL COMMAND</td>
</tr>
<tr>
<td>*CLOSE</td>
<td>CLOSE ISIS FILE COMMAND</td>
</tr>
<tr>
<td>COMMAND$EDIT</td>
<td>COMMAND EDIT</td>
</tr>
<tr>
<td>COUNT</td>
<td>COUNTS NUMBER OF LOAD CYCLES DISPLAYS &quot;.&quot;</td>
</tr>
<tr>
<td>*EXIT</td>
<td>EXIT BACK TO SDM SYSTEM</td>
</tr>
<tr>
<td>LOAD</td>
<td>DOWNLOAD COMMUNICATION WITH DBUGMON</td>
</tr>
<tr>
<td>*MCI</td>
<td>WAITS FOR CHARACTER FORM TERMINAL</td>
</tr>
<tr>
<td>*MCO</td>
<td>OUTPUTS CHARACTER TO TERMINAL</td>
</tr>
<tr>
<td>*OPEN</td>
<td>OPEN ISIS FILE COMMAND</td>
</tr>
<tr>
<td>*OUTCRLF</td>
<td>USED TO OUTPUT CR,LF TO TERMINAL</td>
</tr>
<tr>
<td>PROMPT</td>
<td>OUTPUTS PROMPT TO TERMINAL</td>
</tr>
<tr>
<td>*READ</td>
<td>READ NEXT LINE IN ISIS FILE</td>
</tr>
<tr>
<td>SIGN$ON</td>
<td>OUTPUTS SIGN ON MESSAGE TO TERMINAL</td>
</tr>
<tr>
<td>SYS$ERROR</td>
<td>OUTPUTS MESSAGE DURING DOWN LOADING</td>
</tr>
<tr>
<td>TO$TERM</td>
<td>OUTPUTS MESSAGES TO TERMINAL</td>
</tr>
<tr>
<td>VALID$CPU</td>
<td>CHECKS IF CPU REQUESTED IS VAILD</td>
</tr>
<tr>
<td>WHOS$HERE</td>
<td>OUTPUTS VALID PROCESSOR ID TO TERMINAL</td>
</tr>
</tbody>
</table>

* CONTAINED IN SDM LIBRARY 'MUPIFL.EXT'
ALL$CMD

Used to send the same command to all processors.

Activation: Call ALL$CMD

COMMAND$EDIT

This is the command line editor used by UEXT to edit the command line as it is entered at the terminal. Once a command is entered and terminated with a carriage return (CR) it is sent to the appropriate processor using Producer.

COMMAND$EDIT checks for a proper processor using the Valid$cpu routine. Normal editing functions such as backspace are supported. The bell is given as an output for errors. This occurs when more than 80 characters are entered on a command line or when an illegal character is entered such as the escape (ESC) character.

Error Conditions:  * > 80 characters
                   * (ESC) sequence
                   * backspacing over prompt
                   * (CR) and invalid processor

Activation: Call COMMAND$EDIT

COUNT

Keeps track of number of disc accesses during the load command. Then outputs a '.' to the screen.

Activation: Call COUNT(COUNTT)

COUNTT -- number of times disc accessed
LOAD

Used to load a user written program into the designated processor’s memory. The file must be in hexadecimal format. This is done by using the Intel program OH86.

Activation: Call LOAD

PROMPT

The prompt character is output to the terminal when this procedure in called. The procedure outputs a (CR), (LF), the prompt and then the (BEL).

Activation: Call PROMPT

SIGN$ON

This procedure outputs the system SIGN$ON message to the terminal. The message is retrieved from shared memory using Consumer. The message was placed in shared memory by processor 0.

Activation: Call SIGN$ON

SYS$ERROR

Error messages are sent to the terminal using this routine. In normal operation this routine should never be called. It was used primarily as a tool during development.

Activation: CALL SYS$ERROR(ERR$NUM)

ERR$NUM -- 0 - *** I/O ERROR CHECK MODE STATUS ***
1 - *** ILLEGAL ERROR NUMBER ***
TO$TERM

Transfers information from local memory to the terminal.

Activation: Call TO$TERM

VALID$CPU

The Valid$cpu checks for a valid cpu identification number. This helps to prevent the system from hanging up when an invalid cpu is requested. The procedure returns false if the cpu is invalid, otherwise it returns true.

Activation: I = VALID$CPU(CPU$NUM)
I -- byte
CPU$NUM -- identification of cpu in command line

WHOS$HERE

At the beginning of the MMDE system after the system prompt a second line is shown. This line shows which processors are available for use. The procedure WHOS$HERE determines which processors are available and then displays them.

Activation: Call WHOS$HERE
APPENDIX H

COMMON MEMORY DESCRIPTION

Each of the following variables are listed in the source code file MMDE.MEM on the MMDE system disc.

GLOBAL VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE NAME</th>
<th>VALUE</th>
<th>BASE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARED$MEMORY</td>
<td>8000</td>
<td>16</td>
<td>STARTING LOCATION OF SHARED MEMORY</td>
</tr>
<tr>
<td>BUF$SIZE</td>
<td>80</td>
<td>10</td>
<td>SIZE OF COMMON MESSAGE BUFFER</td>
</tr>
<tr>
<td>MAX$NUM$CPUS</td>
<td>5</td>
<td>10</td>
<td>MAXIMUM NUMBER OF CPU'S IN MMDE SYSTEM</td>
</tr>
<tr>
<td>TOTAL$CPU</td>
<td>6</td>
<td>10</td>
<td>MAX$NUM$CPUS + 1</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>16</td>
<td>VALUE OF FALSE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FF</td>
<td>16</td>
<td>VALUE OF TRUE</td>
</tr>
<tr>
<td>DBUF</td>
<td>1</td>
<td>10</td>
<td>NUMBER OF BYTES IN DEBUG ARRAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(USED IN DEBUGGING CODE)</td>
</tr>
</tbody>
</table>

COMMON MEMORY VARIABLES

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>VARIABLE NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>SEMAPHORE</td>
<td>COMMON MESSAGE BUFFER READY/BUSY</td>
</tr>
<tr>
<td>8001</td>
<td>SEMAPHORE1</td>
<td>NOT USED</td>
</tr>
<tr>
<td>8002</td>
<td>SEMAPHORE2</td>
<td>PRODUCER QUEUE LOCK</td>
</tr>
<tr>
<td>8003</td>
<td>SEMAPHORE3</td>
<td>CONSUMER QUEUE LOCK</td>
</tr>
<tr>
<td>8004</td>
<td>COQUEUEBUSY</td>
<td>USED BY CONSUMER QUEUE -- QUEUE BUSY</td>
</tr>
<tr>
<td>8005</td>
<td>PRQUEUEBUSY</td>
<td>USED BY PRODUCER QUEUE -- QUEUE BUSY</td>
</tr>
<tr>
<td>8006</td>
<td>BUFFERFULL</td>
<td>CONSUMER/PRODUCER -- FLAT BUFFER FULL</td>
</tr>
<tr>
<td>8007</td>
<td>LOADMODE</td>
<td>LOAD COMMAND MODE USED BY UEXT/LOAD</td>
</tr>
<tr>
<td>8008</td>
<td>SIGNAL$BUSY</td>
<td>NOT USED</td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td>8009</td>
<td>PRHEAD LOCATION OF PRODUCER HEAD</td>
<td></td>
</tr>
<tr>
<td>800A</td>
<td>PRTAIL LOCATION OF PRODUCER TAIL</td>
<td></td>
</tr>
<tr>
<td>800B</td>
<td>PRCOUNT PRODUCER NEXT TURN FLAG</td>
<td></td>
</tr>
<tr>
<td>800C</td>
<td>COHEAD LOCATION OF CONSUMER HEAD</td>
<td></td>
</tr>
<tr>
<td>800D</td>
<td>COTAIL LOCATION OF CONSUMER TAIL</td>
<td></td>
</tr>
<tr>
<td>800E</td>
<td>COCOUNT CONSUMER NEXT TURN FLAG</td>
<td></td>
</tr>
<tr>
<td>800F</td>
<td>DBUG(0) USED TO DEBUG CODE</td>
<td></td>
</tr>
<tr>
<td>8010</td>
<td>CPULISTS.ORDER(0) OPERATING ORDER OF CPU'S</td>
<td></td>
</tr>
<tr>
<td>8011</td>
<td>CPULISTS.ORDER(1) CURRENTLY FOR REFERENCE</td>
<td></td>
</tr>
<tr>
<td>8012</td>
<td>CPULISTS.ORDER(2) AND USED TO DETERMINE WHICH PROCESSOR'S ARE AVAILABLE</td>
<td></td>
</tr>
<tr>
<td>8013</td>
<td>CPULISTS.ORDER(3)</td>
<td></td>
</tr>
<tr>
<td>8014</td>
<td>CPULISTS.ORDER(4)</td>
<td></td>
</tr>
<tr>
<td>8015</td>
<td>CPULISTS.TURN(0) NOT USED -- RESERVED</td>
<td></td>
</tr>
<tr>
<td>8016</td>
<td>CPULISTS.TURN(1) NOT USED</td>
<td></td>
</tr>
<tr>
<td>8017</td>
<td>CPULISTS.TURN(2) NOT USED FOR FUTURE</td>
<td></td>
</tr>
<tr>
<td>8018</td>
<td>CPULISTS.TURN(3) NOT USED</td>
<td></td>
</tr>
<tr>
<td>8019</td>
<td>CPULISTS.TURN(4) NOT USED USE</td>
<td></td>
</tr>
<tr>
<td>801A</td>
<td>CPULISTS.SYSCPU(0) NOT USED</td>
<td></td>
</tr>
<tr>
<td>801B</td>
<td>CPULISTS.SYSCPU(1) NOT USED</td>
<td></td>
</tr>
<tr>
<td>801C</td>
<td>CPULISTS.SYSCPU(2) NOT USED LEGAL CPU'S IN SYSTEM</td>
<td></td>
</tr>
<tr>
<td>801D</td>
<td>CPULISTS.SYSCPU(3) NOT USED</td>
<td></td>
</tr>
<tr>
<td>801E</td>
<td>CPULISTS.SYSCPU(4) NOT USED</td>
<td></td>
</tr>
<tr>
<td>801F</td>
<td>ORDER$INDEX CPU INITIALIZATION SEQUENCE</td>
<td></td>
</tr>
<tr>
<td>8020</td>
<td>GROUP$COMMAND GROUP COMMAND FLAG USED FOR GO COMMAND</td>
<td></td>
</tr>
<tr>
<td>8021</td>
<td>MODE.CMD UEXT/DBUGMON MODE FLAG</td>
<td></td>
</tr>
<tr>
<td>8022</td>
<td>MODE.MESSAGE USED FOR NUMBER OF TERMINAL MESSAGES</td>
<td></td>
</tr>
<tr>
<td>8023</td>
<td>TRNL$CK CONTROLS TURNS OF PROCESSORS</td>
<td></td>
</tr>
<tr>
<td>8024</td>
<td>ACT$CPU CURRENT PROCESSOR IN CONTROL</td>
<td></td>
</tr>
<tr>
<td>8025</td>
<td>SDM$CPU CPU IDENTITY OF SDM PROCESSOR</td>
<td></td>
</tr>
<tr>
<td>8026</td>
<td>SDM$START USED IN INITIALIZATION SEQUENCING</td>
<td></td>
</tr>
<tr>
<td>8027</td>
<td>CPU$START USED IN INITIALIZATION SEQUENCING</td>
<td></td>
</tr>
<tr>
<td>8028</td>
<td>EXITT USED BY EXIT COMMAND -- CLEAN EXIT</td>
<td></td>
</tr>
<tr>
<td>8029</td>
<td>GOS$ALL USED FOR GROUP COMMANDS -- GA COMMAND</td>
<td></td>
</tr>
<tr>
<td>802A</td>
<td>GLOBALS$CMD$CHK NOT USED</td>
<td></td>
</tr>
<tr>
<td>802B</td>
<td>SYS$BUF.SOURCE$CPU SOURCE CPU -- CPU SENDING MESSAGE</td>
<td></td>
</tr>
<tr>
<td>802C</td>
<td>SYS$BUF.DEST$CPU DESTINATION CPU -- CPU RCV MESSAGE</td>
<td></td>
</tr>
</tbody>
</table>
802D  SYS$BUF.NUMBCHAR  # OF CHARACTERS IN MESSAGE LOW BYTE
802E  SYS$BUF.NUMBCHAR  # OF CHARACTERS IN MESSAGE HIGH BYTE
802F  SYS$BUF.DATTA(0)  MESSAGE BUFFER -- 80 CHARACTERS

807E  SYS$BUF.DATTA(79)  END MESSAGE BUFFER AND SHARED MEMORY
APPENDIX I

COMMAND SET/INSTRUCTIONS

There are six commands used in the MMDE system. Each command is described below in alphabetical order. After the commands descriptions, instructions are given for invocation of the system.

DISPLAY

FORMAT:  DO -- DISPLAYS ONE MEMORY LOCATION AT SET ADDRESS
          DO 100 -- DISPLAYS 100 MEMORY LOCATIONS AT SET ADDRESS
          DO 100 XXXX:XXXX
          OR
          DISPLAYS 100 MEMORY LOCATIONS STARTING
          DO XXXX:XXXX 100 AT ADDRESS XXXX:XXXX

EXAMPLE: DO 123 1234:1234 DISPLAY 123 MEMORY LOCATIONS STARTING
AT ADDRESS 1234:1234

EXAMINE

FORMAT:  XO -- DISPLAYS CONTENTS OF CPUO'S REGISTERS
          XOAX -- DISPLAYS CONTENTS OF CPUO'S AX REGISTER
          XOAX=0034 -- SETS CPUO'S AX REGISTER TO 0034H

EXAMPLE: X1CX=0012 SETS CPU1'S CX REGISTER TO 0012H
          FORMAT MUST BE AS SHOWN!

EXIT

FORMAT:  CONTROL E -- EXITS MMDE SYSTEM BACK TO SDMB6 SYSTEM
          -- U Returns back to MMDE system.
GO

FORMAT:  GO         -- STARTS CPUO AT CURRENT CS:IP  
           GA         -- STARTS ALL PROCESSORS (NO CARRIAGE RETURN)

LOAD

FORMAT:  LO :F1:FILES.HEX     -- LOADS HEX CONVERTED OBJECT FILE TO CPUO

NOTE:    MUST BE HEX CONVERTED OBJECT FILE USING OH86.

SUBSTITUTE/BREAKPOINT:

FORMAT:  SO 43 1234:1234  SUBSTITUTES 43H AT ADDRESS 1234:1234
          OR  IN CPUO'S ADDRESS SPACE
          SO 1234:1234 43

          SO ** 1234:1234 SETS BREAKPOINT AT ADDRESS 1234:1234
          OR  IN CPUO'S ADDRESS SPACE
          SO 1234:1234 **

          SO  CLEAR BREAKPOINT AT CPUO

NOTE:    MAXIMUM NUMBER OF BREAKPOINTS IS ONE. ALSO
          BREAKPOINT IS AUTOMATICALLY CLEARED AFTER EACH
          GO COMMAND.

******************************************************************************
SOFTWARE DOES NOT PROTECT FROM OPERATOR
ERRORS AND IS NOT VERY FORGIVING. IT IS
EASILY SENT OUT TO LUNCH.
******************************************************************************
INSTRUCTIONS

1. PUT SDM86 SYSTEM IN DRIVE 0 OF INTELLECT SYSTEM.
2. PUT MMDE SYSTEM IN DRIVE 1 OF INTELLECT SYSTEM.
3. BOOT SYSTEM PER INTEL DOCUMENTATION.
4. TYPE SDM86 AT THE TERMINAL.
5. TYPE R :F1:MMDE AT THE SDM86 PROMPT.
6. TYPE U AT SDM86 PROMPT.
7. THE MMDE SYSTEM PROMPT SHOULD NOW APPEAR.
APPENDIX J

SAMPLE USER CODE -- LOCATE EXAMPLE

The following is an example of locating user written code in the MMDE environment. Note the addition of the extra 20 bytes to the program stack. This allows the MMDE system to save and change user registers.

EXAMPLE: See file TESTLD.CSD on system disc.
PL/M-86 COMPILER  TEST CODE FOR LOADING AND RUNNING

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE TESTLD
OBJECT MODULE PLACED IN :Fl:TESTLD.OBJ
COMPILER INVOKED BY: PLM86 :Fl:TESTLD.SRC

$dat(e(AUG-08-86)
$LARGE XREF CODE
$TITLE ('TEST CODE FOR LOADING AND RUNNING')

1 TESTLD: DO;
2 1 = EXIT: PROCEDURE EXTERNAL;
3 2 = END EXIT;

4 1 declare a(6) byte;
5 1 declare i byte;
6 1 do i = 0 to 5:

0004 FA CLI
0005 2E8E160000 MOV SS,CS:@STACK$FRAME
000A BC0600 MOV SP,@STACK$OFFSET
000D 8BEC MOV BP,SP
000F 2E8E1E0200 MOV DS,CS:@DATA$FRAME
0014 FB STI
0015 C606060000 MOV I,0H
001A 8A060600 MOV AL,1
001E 80F805 CMP AL,5H
0021 7603 JBE $+5H
0023 E91400 JMP @2

7 2 a(i) = i+1;

8 2 end:

0026 FEC0 INC AL
0028 8A1E0600 MOV BL,1
002C B700 MOV BH,0H
002E 884700 MOV A[BX],AL

9 1 CALL EXIT:

10 1 END TESTLD;

003F FB STI
0040 F4 HLT
PL/M-86 COMPILER  TEST CODE FOR LOADING AND RUNNING

CROSS-REFERENCE LISTING

<table>
<thead>
<tr>
<th>DEFN</th>
<th>ADDR</th>
<th>SIZE</th>
<th>NAME, ATTRIBUTES, AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0000H</td>
<td>6</td>
<td>A. . . . . . . .  BYTE ARRAY(6)</td>
</tr>
<tr>
<td>2</td>
<td>0000H</td>
<td></td>
<td>EXIT . . . . . .  PROCEDURE EXTERNAL(0) STACK=0000H</td>
</tr>
<tr>
<td>5</td>
<td>0006H</td>
<td>1</td>
<td>I. . . . . . . .  BYTE 6</td>
</tr>
<tr>
<td>1</td>
<td>0004H</td>
<td>61</td>
<td>TESTLD . . . . .  PROCEDURE STACK=0006H</td>
</tr>
</tbody>
</table>

MODULE INFORMATION:

- CODE AREA SIZE = 0041H 65D
- CONSTANT AREA SIZE = 0000H 0D
- VARIABLE AREA SIZE = 0007H 7D
- MAXIMUM STACK SIZE = 0006H 6D
- 16 LINES READ
- 0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
APPENDIX K

MMDE DISC DIRECTORY

There are two discs required to run the MMDE system. They are the MMDE System disc and the SDM86 disc. The required files are listed below and a brief explanation of the file.

First Disc: MMDE System

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFDMON.SRC</td>
<td>BUFMON SOURCE CODE FOR BUFFER MONITOR</td>
</tr>
<tr>
<td>CPUID.SRC</td>
<td>CPU IDENTITY SOURCE CODE -- GET CPUID</td>
</tr>
<tr>
<td>DEBUG.SRC</td>
<td>DEBUG MONITOR SOURCE CODE</td>
</tr>
<tr>
<td>MMDE.SRC</td>
<td>SDM86 ENTRY CODE TO MMDE SYSTEM SOURCE CODE</td>
</tr>
<tr>
<td>MMIFLB.EXT</td>
<td>DECLARATIONS FOR MMDE SYSTEM LIBRARY</td>
</tr>
<tr>
<td>MMEXIT.SRC</td>
<td>SOURCE CODE OF EXIT COMMAND IN SYSTEM LIBRARY</td>
</tr>
<tr>
<td>MMDE.LIB</td>
<td>MMDE SYSTEM LIBRARY</td>
</tr>
<tr>
<td>MMDE.MEM</td>
<td>MMDE SYSTEM COMMON MEMORY DECLARATIONS</td>
</tr>
<tr>
<td>UEXT.SRC</td>
<td>UEXT SOURCE CODE FOR SDM BOARD</td>
</tr>
<tr>
<td>TESTLD.SRC</td>
<td>TEST PROGRAM FOR LOADING AND LOCATING</td>
</tr>
<tr>
<td>TESTLD.HEX</td>
<td>HEX CONVERTED FILE OF TESTLD.OBJ</td>
</tr>
<tr>
<td>DEBUG.ABS</td>
<td>ABSOLUTE OBJECT CODE FOR EPROMS</td>
</tr>
<tr>
<td>MMDE</td>
<td>ABSOLUTE CODE FOR INVOKING MMDE SYSTEM</td>
</tr>
<tr>
<td>BURN.CSD</td>
<td>COMMAND FILE TO BURN EPROMS</td>
</tr>
<tr>
<td>DEBUG.CSD</td>
<td>COMMAND FILE TO BUILD DEBUG.ABS</td>
</tr>
<tr>
<td>MMDE.CSD</td>
<td>COMMAND FILE TO BUILD MMDE ABSOLUTE CODE</td>
</tr>
<tr>
<td>TESTLD.CSD</td>
<td>COMMAND FILE TO BUILD TESTLD.HEX</td>
</tr>
</tbody>
</table>
Second Disc: SDM SYSTEM

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE</td>
<td>ISIS ATTRIBE COMMAND</td>
</tr>
<tr>
<td>DELETE</td>
<td>ISIS DELETE COMMAND</td>
</tr>
<tr>
<td>COPY</td>
<td>ISIS COPY COMMAND</td>
</tr>
<tr>
<td>DIR</td>
<td>ISIS DIRECTORY COMMAND</td>
</tr>
<tr>
<td>RENAME</td>
<td>ISIS RENAME COMMAND</td>
</tr>
<tr>
<td>IDISK</td>
<td>ISIS INITIALIZE DISC COMMAND</td>
</tr>
<tr>
<td>SUBMIT</td>
<td>ISIS SUBMIT COMMAND</td>
</tr>
<tr>
<td>VERS</td>
<td>ISIS VERSION COMMAND</td>
</tr>
<tr>
<td>PLM86.XXX</td>
<td>PLM86 COMPILER</td>
</tr>
<tr>
<td>CREDIT.XXX</td>
<td>ISIS EDITOR</td>
</tr>
<tr>
<td>LINK86.XXX</td>
<td>LINKER</td>
</tr>
<tr>
<td>LOC86.XXX</td>
<td>LOCATER</td>
</tr>
<tr>
<td>LIB86.XXX</td>
<td>LIBRARY BUILDER</td>
</tr>
<tr>
<td>OH86</td>
<td>CONVERTS OBJECT CODE TO ASCII HEXADECIMAL</td>
</tr>
<tr>
<td>INIT</td>
<td>TELETYPETE PRINTER DIRVER</td>
</tr>
<tr>
<td>SDM86.XXX</td>
<td>SDM86 SYSTEM</td>
</tr>
<tr>
<td>UPM.XXX</td>
<td>PROM PROGRAMMER PROGRAM</td>
</tr>
<tr>
<td>IXREF</td>
<td>INTERMODULE CROSS REFERENCE</td>
</tr>
<tr>
<td>SDMIOX.XXX</td>
<td>SDM LIBRARIES</td>
</tr>
<tr>
<td>MUPIFL.XXX</td>
<td>SDM86-MDS INTERFACE LIBRARY</td>
</tr>
</tbody>
</table>

.XXX - REPRESENTS DIFFERENT EXTENSIONS TO BASIC FILE NAME.
ALL THE ABOVE FILES ARE DOCUMENTED IN THE INTEL LITERATURE.
APPENDIX L

MMDE LIBRARY SOURCE CODE
MODULE INFORMATION:

CODE AREA SIZE = 001CH 28D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 0001H 1D
MAXIMUM STACK SIZE = 0008H 8D
16 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 Compilation
APPENDIX M

BUFMON SOURCE CODE
PL/M-86 COMPILER

MMDE BUFFER MONITOR  AUG-08-86  PAGE 1

SHARED MEMORY

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE BUFFERMONITOR
OBJECT MODULE PLACED IN :F1:BUFMON.OBJ
COMPILER INVOKED BY:  PLM86 :F1:BUFMON.SRC

$DATE(AUG-08-86)
$LARGE ROM
$SPACETIME WIDTH(60)
$TITLE('MMDE BUFFER MONITOR')
$SUBTITLE('SHARED MEMORY')

BUFFER$MONITOR: DO;
$INCLUDE(:F1:MMDE.MEM)
/* SHARED MEMORY DEFINITIONS */
/* JULY 12, 1986 */

DECLARE SHARED$MEMORY LITERALLY '8000H',
BUFSIZE LITERALLY '80D',
MAX$NUM$CPUS LITERALLY '5D',
TOTAL$CPUS LITERALLY '6D',
FALSE LITERALLY '00H',
TRUE LITERALLY '0FFH',
DBUF LITERALLY '01D',

*/ SHARED BUFFER VARIABLES */
SEMAPHORE BYTE AT (SHARED$MEMORY),
SEMAPHORE1 BYTE AT (SHARED$MEMORY + 1),
SEMAPHORE2 BYTE AT (SHARED$MEMORY + 2),
SEMAPHORE3 BYTE AT (SHARED$MEMORY + 3),
COQUEUESBUSY BYTE AT (SHARED$MEMORY + 4),
PRQUEUE$BUSY BYTE AT (SHARED$MEMORY + 5),
BUFFERSFULL BYTE AT (SHARED$MEMORY + 6),
LOADSMODE BYTE AT (SHARED$MEMORY + 7),
SIGNALSBUSY BYTE AT (SHARED$MEMORY + 8),
(PRHEAD, PRTAIL, PRCOUNT) BYTE AT (SHARED$MEMORY + 9),
(COHEAD, COTAIL, COCOUNT) BYTE AT (SHARED$MEMORY + 12),
DBG(DBG) BYTE AT (SHARED$MEMORY + TOTAL$CPUS + 9)
*/ VARIABLES USED FOR PROCESSOR ORDERING */
CPUSLISTS STRUCTURE (ORDER(MAX$NUM$CPUS) BYTE,
TURN (MAX$NUM$CPUS) BYTE,
SYSSCPU (MAX$NUM$CPUS) BYTE)
AT (SHARED$MEMORY + TOTAL$CPUS + DBUF + 9),
(ORDER$INDEX, GROUP$COMMAND) BYTE AT
(SHARED$MEMORY + TOTAL$CPUS + DBUF + MAX$NUM$CPUS + MAX$NUM$CPUS + 9),
MODE STRUCTURE (CMD BYTE, MESSAGE BYTE) AT
(SHARED$MEMORY + TOTAL$CPUS + DBUF + MAX$NUM$CPUS
SHARED MEMORY

+ MAX$NUM$CPUS + MAX$NUM$CPUS + 11),

/* SEMAPHORE DEFINITIONS */

= (TRN$LCK, ACT$CPU, SDM$CPU, SDM$START, CPU$START,
  EXIT$T, GO$ALL, GLOBAL$CMD$CHK) BYTE

= AT (13 + SHARED$MEMORY + TOTAL$CPUS + DBUF +
  MAX$NUM$CPUS + MAX$NUM$CPUS + MAX$NUM$CPUS),

= /* SYSTEM BUFFER STRUCTURE */

= SYSS$BUF STRUCTURE(SOURCE$CPU BYTE, DEST$CPU BYTE,
  NUMBS$CHAR WORD, DATTA (BUF$SIZE) BYTE) AT

= (SHARED$MEMORY + DBUF + TOTAL$CPUS +
  MAX$NUM$CPUS + MAX$NUM$CPUS + MAX$NUM$CPUS + 21);
$SUBTITLE('BUFFER DESCRIPTION')

/* THIS MONITOR PROTECTS ACCESS TO THE SHARED BUFFER. TWO PROCEDURES ARE PROVIDED TO CHANGE DATA IN THE BUFFER. THEY DO ALL THE REQUIRED BOOKKEEPING FOR MUTUAL EXCLUSIVE ACCESS. INITIALIZATION OF THE MONITOR IS DONE BY THE MAIN PROGRAM MODULE. 

PROCEDURES IN THE MONITOR
NAME    SCOPE    FUNCTION
producer public WRITE BUFFER
consumer public READ BUFFER
init$buf$mon public INITIALIZE MONITOR
signal   local    MUTUAL EXCLUSION */

DECLARE
BUFFERSBUSY LITERALLY '1D',
BUFFERSREADY LITERALLY '0D',
NOT$BUSY     LITERALLY '03H',
MYPRTURN     BYTE,
MYCOTURN     BYTE,
RCVR$CPU1    BYTE;
$SUBTITLE('initSbuf$mon/signal PROCEDURES')

initSbuf$mon: PROCEDURE PUBLIC;
/* INITIALIZE MONITOR SEMAPHORES TO '0H' */

SEMAPHORE = BUFFER$READY;
SEMAPHORE1 = BUFFER$READY;
SEMAPHORE2 = BUFFER$READY;
SEMAPHORE3 = BUFFER$READY;
COQUEUE$BUSY = BUFFER$READY;
PRQUEUE$BUSY = BUFFER$READY;
SIGNAL$BUSY = BUFFER$READY;

/* SET BUFFER STATUS TO EMPTY */
BUFFER$FULL = FALSE;

/* INITIALIZE QUEUES TO STARTING VALUES */
PRHEAD, PRTAIL, PRCOUNT = BUFFER$READY;
COHEAD, COTAIL, COCOUNT = BUFFER$READY;
MODE.CMD = 01H;

END initSbuf$mon;

signal: PROCEDURE (MPU$ID, BUFFER$STATUS);
DECLARE (MPU$ID, BUFFER$STATUS, LOCKING$CPU) BYTE,
          (I, STEP) WORD;

DO CASE BUFFER$STATUS;
  /* BUFFER$READY */
  IF LOCKING$CPU = MPU$ID THEN
    SEMAPHORE = 0;
  END;

  /* BUFFER$BUSY */
  DO;
    DO WHILE LOCKSET (@SEMAPHORE, 1);
    END;
    LOCKING$CPU = MPU$ID;
  END;

  /* MODE$BUSY */
  DO;
    DO WHILE LOCKSET (@SEMAPHORE1, 1);
    END;
    LOCKING$CPU = MPU$ID;
  END;

  /* PR QUEUE LOCK */
  DO WHILE LOCKSET (@SEMAPHORE2, 1);
  END;

  /* CO QUEUE LOCK */
  DO WHILE LOCKSET (@SEMAPHORE3, 1);
34 4 END;
35 3 END;
36 2 SIGNAL$BUSY = 0;
37 2 END signal;
$SUBTITLE('PRQUEUE/PRDEQUEUE ROUTINES')

PRQUEUE:

PROCEDURE;

DECLARE PRQUEUE$LOCK LITERALLY '03H';

CALL SIGNAL (0,PRQUEUE$LOCK);

IF PRCOUNT = 0 AND PRQUEUE$BUSY = FALSE

    DO;

    PRQUEUE$BUSY = TRUE;

    SEMAPHORE2 = 0;

    RETURN;

    END;

ELSE PRCOUNT = PRCOUNT + 1;

PRTAIL = (PRTAIL + 1) MOD TOTAL$CPUS;

MYPRTURN = PRTAIL;

SEMAPHORE2 = 0;

DO WHILE MYPRTURN <> PRHEAD;

END;

END prqueue;

PRDEQUEUE:

PROCEDURE;

DECLARE PRQUEUE$LOCK LITERALLY '03H';

CALL SIGNAL (0,PRQUEUE$LOCK);

IF PRCOUNT = 0 THEN

    DO;

    PRQUEUE$BUSY = FALSE;

    SEMAPHORE2 = 0;

    RETURN;

    END;

PRHEAD = (PRHEAD + 1) MOD TOTAL$CPU;

SEMAPHORE2 = 0;

END prdequeue;
$SUBTITLE ('COQUEUE/CODEQUEUE ROUTINES')

coqueue: PROCEDURE;
DECLARE COQUEUE$LOCK LITERALLY '04H';

CALL SIGNAL (0,COQUEUE$LOCK);
IF COCOUNT = 0 AND COQUEUE$BUSY = FALSE THEN
   DO;
      COQUEUE$BUSY = TRUE;
      SEMAPHORE3 = 0;
      RETURN;
   END;
ELSE COCOUNT = COCOUNT + 1;
COTAIL = (COTAIL + 1) MOD TOTAL$CPUS;
MYCOTURN = COTAIL;
SEMAPHORE3 = 0;
DO WHILE MYCOTURN <> COHEAD;
END;
END coqueue;

codequeue: PROCEDURE;
DECLARE COQUEUE$LOCK LITERALLY '04H';

CALL SIGNAL (0,COQUEUE$LOCK);
IF COCOUNT = 0 THEN
   DO;
      COQUEUE$BUSY = FALSE;
      SEMAPHORE3 = 0;
      RETURN;
   END;
ELSE COCOUNT = COCOUNT - 1;
COHEAD = (COHEAD + 1) MOD TOTAL$CPUS;
SEMAPHORE3 = 0;
END codequeue;
$SUBTITLE('producer PROCEDURE')

95 1  producer: PROCEDURE (SOURCE$CPU, DEST$CPU, MESSPTR, NUM$CHAR, STATUS) PUBLIC;
96 2   - TR, NUM$CHAR, STATUS) PUBLIC;
97 2   - EST$CPU) BYTE;
98 2   DECLARE (MESSPTR, STATUS) POINTER,
99 2   NUM$CHAR WORD, (SOURCE$CPU, D
100 2   DECLARE ERR BASED STATUS BYTE,
101 2   MESSAGE BYTE:
102 2
103 2   ERR = FALSE;
104 2   MESSAGE = TRUE;
105 2   DO WHILE MESSAGE:
106 3   CALL PRQUEUE;
107 3   IF BUFFERS$FULL = FALSE THEN
108 4     DO;
109 4       CALL signal(SOURCE$CPU, BUFFERS$BUSY
110 4       SY$BUF.DEST$CPU = DEST$CPU;
111 4       SY$BUF.SOURCE$CPU = SOURCE$CPU;
112 4       SY$BUF.NUM$CHAR = NUM$CHAR;
113 4       CALL MOVE(MESSPTR, SY$BUF.DATTA(0
114 4 - ), NUM$CHAR);
115 4       MESSAGE = FALSE;
116 4       BUFFERS$FULL = TRUE;
117 4       CALL signal(SOURCE$CPU, BUFFERS$READ
118 4 - Y);
119 4    END;
120 4    CALL PRDEQUEUE;
121 4    END;
122 4 END producer;
\texttt{CONSUMER PROCEDURE}

\texttt{
\$SUBTITLE ('consumer PROCEDURE')

116 1 consumer: \texttt{PROCEDURE(RCVR$CPU,SENDER$CPU,MES$PT-R,NUM$CHAR,STATUS) PUBLIC;}
117 2 ) \texttt{DECLARE (MES$PTR,STATUS,NUM$CHAR}}
118 2 \texttt{POINTER, (RCVR$CPU,SENDER$CPU) BY}
119 2 \texttt{TE, ERR BASED STATUS}
120 2 \texttt{BYTE, NUMBSCHAR BASED NUM$CHA}
121 2 \texttt{R WORD, NOSMESSAGE BYTE;}
122 2 ERR = FALSE;
123 2 NOSMESSAGE = TRUE;
124 2 DO WHILE NOSMESSAGE:
125 3 \texttt{CALL COQUEUE;}
126 3 \texttt{IF BUFFER$FULL = TRUE THEN}
127 4 \texttt{CALL signal(RCVR$CPU,BUFFER$BUSY);}\texttt{DO;}
128 5 \texttt{IF SYSSBUF.DEST$CPU = RCVR$CPU THEN}
129 5 \texttt{BUFFERSFULL = FALSE;}
130 5 \texttt{NOSMESSAGE = FALSE;}
131 5 \texttt{END;}
132 4 \texttt{CALL signal(RCVR$CPU,BUFFER$READY);}\texttt{END;}
133 4 \texttt{END;}
134 3 \texttt{CALL codequeue;}
135 3 \texttt{END;}
136 2 \texttt{END consumer;}
137 1 \texttt{END buffer$monitor;}

\texttt{MODULE INFORMATION:}

\begin{verbatim}
CODE AREA SIZE = 0445H 1093D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 000AH 10D
MAXIMUM STACK SIZE = 001EH 30D
273 LINES READ
0 PROGRAM ERROR(S)
\end{verbatim}

END OF PL/M-86 COMPILATION
APPENDIX N

CPUID SOURCE CODE
This procedure uses the 8252-5 programmable timer and the 8259A programmable interrupt controller to return the CPU$ID. The timer generates an interrupt signal in the 8259A controller. Each CPU generates a different interrupt.

**INITIALIZE WORDS FOR 8259A**

- **ICW1** -- EDGE TRIGGERED, SINGLE PIC, NEED ICW4
- **ICW2** -- FAKE INTERRUPT VECTOR
- **ICW3** -- NOT USED
- **ICW4** -- 8086 MODE, BUFFERED MASTER, MANUAL EOI, NESTED MODE

- **OCW1** -- INTERRUPT MASK, ALWAYS ENABLED
- **OCW2** -- NON-SPECIFIC EOI
- **OCW3** -- POLL COMMAND

```plaintext
DECLARE
 ICW1 LITERALLY '013H', OCW1 LITE
 RALLY '000H', ICW2 LITERALLY '000H', OCW2 LITE
 RALLY '020H', ICW4 LITERALLY '00DH', OCW3 LITE
 RALLY '000H', PIC$PORT1 LITERALLY '00C0H', COUNT LITE
 RALLY '0FFH', PIC$PORT2 LITERALLY '00C2H', COUNT$TIMER1 LITE
 RALLY '00D2H', MODE$TIMER0 LITERALLY '030H', BITS$MASK LITE
 RALLY '07FH', MODE$TIMER1 LITERALLY '070H', CPU$ID BYTE:
 /* INITIALIZE 8259A PIC */
 OUTPUT(PIC$PORT1) = ICW1;
```
OUTPUT(PIC$PORT2) = ICW2;
OUTPUT(PIC$PORT 2) = ICW4;

OUTPUT(PIC$PORT2) = OCW1;
OUTPUT(PIC$PORT 1) = OCW2;

/* INITIALIZE 8253-5 TIMER -- TIMER 0 SET LOW
   BUT NOT USED
   SET INTERRUPT
   MODE WORD -- TIMER = 1, TWO BYTES OF COUNT, MO
   DE = 0
   BINARY 16 BIT COUNTER*/

OUTPUT(CONTROL$PORT) = MODE0TIMERO;
OUTPUT(CONTROL$PORT) = MODE0TIMER1;
OUTPUT(COUNT$TIMER1) = COUNT;
OUTPUT(COUNT$TIMER1) = COUNT;

/* GO READ INTERRUPT NUMBER */
DO WHILE 1;
OUTPUT(PIC$PORT1) = OCW3;
IF((CPU$ID:=INPUT(PIC$PORT1)) AND 80H) = 8
   THEN RETURN (CPU$ID AND BIT$MASK);
OUTPUT(PIC$PORT1) = OCW2;
END;

END cpuid$set;

MODULE INFORMATION:

CODE AREA SIZE  = 0056H  86D
CONSTANT AREA SIZE = 0000H  0D
VARIABLE AREA SIZE = 0001H  1D
MAXIMUM STACK SIZE = 0002H  2D
61 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
APPENDIX 0

DBUGMON SOURCE CODE
EXTERNAL PROCEDURE DECLARATIONS

PLACED IN :F1:DEBUG.OBJ
COMPILER INVOKED BY: PLM86 :F1:DEBUG.SRC

$DATE(AUG-10-86) PAGEWIDTH(60)
$TITLE('MMDE MONITOR')
$SUBTITLE('EXTERNAL PROCEDURE DECLARATIONS')
$LARGE ROM

MMDE$MAIN: DO;
$INCLUDE(:F1:MMDE.MEM)

/* SHARED MEMORY DEFINITIONS */

/* JULY 12, 1986 */

DECLARE SHARED$MEMORY LITERALLY '8000H',
BUF$SIZE LITERALLY '80D',
MAX$NUM$CPUS LITERALLY '5D',
TOTAL$CPUS LITERALLY '6D',
FALSE LITERALLY '00H',
TRUE LITERALLY 'OFFH',
DBUF LITERALLY '01D',

/* SHARED BUFFER VARIABLES */

SEMAPHORE BYTE AT (SHARED$MEMORY),
SEMAPHORE1 BYTE AT (SHARED$MEMORY + 1),
SEMAPHORE2 BYTE AT (SHARED$MEMORY + 2),
SEMAPHORE3 BYTE AT (SHARED$MEMORY + 3),
COQUEUE$BUSY BYTE AT (SHARED$MEMORY + 4),
PRQUEUE$BUSY BYTE AT (SHARED$MEMORY + 5),
BUFFERSFULL BYTE AT (SHARED$MEMORY + 6),
LOADS$MODE BYTE AT (SHARED$MEMORY + 7),
SIGNAL$BUSY BYTE AT (SHARED$MEMORY + 8),

(PRHEAD,PRTAIL,PRCOUNT) BYTE AT (SHARED$MEMORY + 9),
(COHEAD,COTAIL,COCOUNT) BYTE AT (SHARED$MEMORY + 12),

DBUF(DBUF) BYTE AT (SHARED$MEMORY+TOTAL$CPUS+9)

/* VARIABLES USED FOR PROCESSOR ORDERING */

CPUS$LISTS STRUCTURE (ORDER(MAX$NUM$CPUS) BYTE,
TURN (MAX$NUM$CPUS) BYTE,
SYS$CPU (MAX$NUM$CPUS) BYTE)
AT (SHARED$MEMORY + TOTAL$CPUS + DBUF + 9),

(ORDER$INDEX,GROUP$COMMAND) BYTE AT
(SHARED$MEMORY+TOTAL$CPUS+DBUF + MAX$NUM$CPUS + MAX$NUM$CPUS + 9),

MODE STRUCTURE (CMD BYTE, MESSAGE BYTE) AT
(SHARED$MEMORY + TOTAL$CPUS + DBUF + MAX$NUM$CPUS + MAX$NUM$CPUS + 11),
EXTERNAL PROCEDURE DECLARATIONS

/* SEMAPHORE DEFINITIONS */

( TRN$LCK, ACT$CPU, SDM$CPU, SDM$START, CPU$START,
  EXITT, GO$ALL, GLOBAL$CMD$CHK) BYTE
AT (13 + SHARED$MEMORY + TOTAL$CPUS + DBUF +
MAXNUM$CPUS + MAXNUM$CPUS + MAX$NUM$CPUS),

/* SYSTEM BUFFER STRUCTURE */

SYSS$BUF STRUCTURE (SOURCE$CPU BYTE, DEST$CPU BYTE,
NUMBS$CHAR WORD, DATTA (BUF$SIZE) BYTE) AT
( SHARED$MEMORY + DBUF + TOTAL$CPUS +
MAX$NUM$CPUS + MAXNUM$CPUS + MAX$NUM$CPUS + 21):

producer: PROCEDURE (SOURCE$CPU, DEST$CPU, MES$PTR,
TR, NUMBS$CHAR, STATUS) EXTERNAL;
DECLARE (MES$PTR, STATUS) POINTER,
NUMBS$CHAR WORD, (SOURCE$CPU, DEST$CPU)
BYTE;
END producer;

consumer: PROCEDURE (RCVR$CPU, SENDER$CPU, MES$PTR,
TR, NUMBS$CHAR, STATUS) EXTERNAL;
DECLARE (MES$PTR, NUMBS$CHAR, STATUS) POINTER,
(RCVR$CPU, SENDER$CPU) BYTE;
END consumer;

init$buf$mon: PROCEDURE EXTERNAL;
END init$buf$mon;

set$cpu$id: PROCEDURE BYTE EXTERNAL;
END set$cpu$id;
VARIABLE DECLARATIONS

DECLARE CPUID BYTE, LOCSBUF(BUFSIZE) BYTE, 
(NUMSCHAR, BUFSPOS, CMDREP, VALUE) WORD, 
(I, J, ERR, COMMAND, SUBCOMMAND, STATUS, CHKSUM, 
NEWVALUE, BREAKPOINTFLAG, RUNN, RESTART) BYTE, 

ASCII$COMMANDARY (*) BYTE DATA ('lgxdsea'), 
SIGNON (*) BYTE DATA 
(' Multi-Microprocessor Debug Executive 
- Version 1.0 August 10, 1986 '), 
CMD$CPU (*) BYTE DATA 
('PROCESSOR EXECUTING TASK'), 
TASK$DONE (*) BYTE DATA 
('PROCESSOR COMPLETED TASK'), 

MEM$PTR POINTER, 
MEM STRUCTURE (OFFSET WORD, SEGMENT WORD) 
AT (MEM$PTR), BYTE$LOC BASED MEM$PTR(1) BYTE, 
CHAR BASED MEM$PTR (1) BYTE, OUTCPU (80) BYTE;

DECLARE 

ABSENT LITERALLY 'OFFH', 
ASCII LITERALLY '61H', 
BELL LITERALLY '07H', 
CR LITERALLY '0DH', 
COLON LITERALLY '3AH', 
SPC LITERALLY '20H', 
PERIOD LITERALLY '2EH', 
GROUP LITERALLY '47H', 
HEX$F LITERALLY '5FH', 
LF LITERALLY '0AH', 
LOWER$CASE LITERALLY '20H', 
NOSMESSAGE LITERALLY '00H', 
START$POSITION LITERALLY '0D', 
TURN$DONE LITERALLY '0D', 
TURNN LITERALLY '1D', 
UPPER$CASE LITERALLY '5FH', 
ZERO LITERALLY '0000H';

/* GO AND BREAKPOINT COMMAND DECLARATIONS */
DECLARE (DBGSTACKBASE, DBGSTACK$PTR) WORD, 
XFER$STACK$PTR POINTER, 
XFER$STACK BASED XFER$STACK$PTR (16) WORD, 
XFER$STACKSADR STRUCTURE (OFF WORD, SEG WORD) 
AT (@XFER$STACK$PTR), COPY$REGISTERS(16) WORD;

DECLARE 

/* USED BY CALL GOUSERCODEPTR FOR IP (00) 
- */ 
/* USED BY CALL GOUSERCODEPTR FOR CS (01) 
- */
BP LITERALLY 'COPY$REGISTERS (02)', 
DI LITERALLY 'COPY$REGISTERS (03)', 
SI LITERALLY 'COPY$REGISTERS (04)', 
BX LITERALLY 'COPY$REGISTERS (05)'.
VARIABLE DECLARATIONS

DX LITERALLY 'COPY$REGISTERS (06)',
CX LITERALLY 'COPY$REGISTERS (07)',
AX LITERALLY 'COPY$REGISTERS (08)',
DS LITERALLY 'COPY$REGISTERS (09)',
ES LITERALLY 'COPY$REGISTERS (10)',
IP LITERALLY 'COPY$REGISTERS (11)',
CS LITERALLY 'COPY$REGISTERS (12)',
FLAG LITERALLY 'COPY$REGISTERS (13)',
SS LITERALLY 'COPY$REGISTERS (14)',
SP LITERALLY 'COPY$REGISTERS (15)',

REG$ARY (*) WORD DATA (' BPDISIBXDXCXAX
- DSESIPCSFLSSSP');
$SUBTITLE('init$cpu listar set$command list')

17 1  init$cpu list: PROCEDURE;
18 2  DO I = 0 TO MAX$NUM$CPUS - 1;
19 3  CPU$LISTS$TURN(I) = TURN$DONE;
20 3  CPU$LISTS$ORDER(I) = ABSENT;
21 3  CPU$LISTS$SYS$CPU(I) = MAX$NUM$CPUS;
22 3  END;
   /* INITIALIZE MEMORY POINTERS */
23 2  MEM$SEGMENT = 0000H;
24 2  MEM$OFFSET = 0000H;
25 2  END init$cpu list;

26 1  cpu$order: PROCEDURE;
27 2  DECLARE (I,J) BYTE;
28 2  ORDERS$INDEX,J = 0;
29 2  DO I = 0 TO MAX$NUM$CPUS - 1;
30 3  IF CPU$LISTS$SYS$CPU(I) <> MAX$NUM$CPUS
31 4  THEN DO;
32 4  CPU$LISTS$ORDER(J) = CPU$LISTS$SYS$CPU(I);
33 4  J = J + 1;
34 4  END;
35 3  END;
36 2  END cpu$order;
$SUBTITLE('UEXT MONITOR CONTROL ROUTINES')

procedure(cmd$mode: PROCEDURE(MPUID, CMD, NUMMES, STATUS));
DECLARE (MPUSID, CMD, NUMMES) BYTE,
STATUS POINTER, NOT$BUSY LITERALLY '02H',
ERR BASED STATUS BYTE;

ERR = fALSE;
IF MPUSID = ACT$CPU THEN
DO;
LOOP: IF MODE.CMD = NOT$BUSY THEN
DO:
MODE.MESSAGE = NUM$MES;
IF CMD >= 0 AND CMD <= 3 THEN
DO CASE CMD:
MOD$CMD = 0; /*SDM TO TERMINAL*/
MOD$CMD = 1; /*SDM EDIT MODE */
MOD$CMD = 3; /*SDM LOAD MODE */
END;
ELSE ERR = tRUE;
END;
ELSE GOTO LOOP;
END;
ELSE ERR = tRUE;
END cmd$mode;

set$my$turn: PROCEDURE;
DO WHILE LOCKSET(@TRN$LCK, 1);
END;
ACT$CPU = CPU$ID;
END set$my$turn;

set$next$turn: PROCEDURE;
TRN$LCK = fALSE;
END set$next$turn;

SPURIOUS$INTERRUPT: PROCEDURE INTERRUPT 32;
DECLARE ICW1 LITERALLY '013H',
ICW2 LITERALLY '020H',
ICW4 LITERALLY '00DH',
OCW1 LITERALLY 'OFFH',
PICS$PORT1 LITERALLY '00C0H',
PICS$PORT2 LITERALLY '00C2H';
DISABLE;
OUTPUT(PICS$PORT1) = ICW1;
OUTPUT(PICS$PORT2) = ICW2;
OUTPUT(PICS$PORT2) = ICW4;
OUTPUT(PICS$PORT1) = OCW1; /* MASK INTERRUPTS */
ENABLE;
END SPURIOUS$INTERRUPT;
ERROR

$SUBTITLE('error')

75 1 error: PROCEDURE;
76 2 OUT$CPU(0) = BELL;
77 2 OUT$CPU(1) = CR;
78 2 OUT$CPU(2) = LF;
79 2 OUT$CPU(3) = 'E';
80 2 OUT$CPU(4) = 'R';
81 2 OUT$CPU(5) = 'R';
82 2 OUT$CPU(6) = 'O';
83 2 OUT$CPU(7) = 'R';
84 2 OUT$CPU(8) = CR;
85 2 OUT$CPU(9) = LF;
86 2 CALL cmd$mode(CPU$ID,0,1,@STATUS);
87 2 CALL producer (CPU$ID,SDM$CPU,@OUT$CPU,10,@STATUS);
88 2 CALL cmd$mode(CPU$ID,2,0,@STATUS);
89 2 END error;
90 1 CHG$LOAD$MODE: PROCEDURE (MODE);
91 2 DECLARE MODE BYTE;
92 2 DO WHILE LOAD$MODE <> 3;
93 3 END;
94 2 LOAD$MODE = MODE;
95 2 END CHG$LOAD$MODE;
96 1 GETMORE: PROCEDURE;
97 2 IF BUF$POS + 1 >= NUMCHAR THEN
98 2 DO;
99 3 CALL CHG$LOAD$MODE (0);
100 3 CALL consumer(CPU$ID,SDM$CPU,@LOCBUF,@NUMCHAR,A@STATUS);
101 3 BUF$POS = 0;
102 3 END;
103 2 ELSE
104 2 BUF$POS = BUF$POS + 1;
105 2 END GETMORE;
ASCII -- HEX CONVERSIONS

$SUBTITLE ('ASCII -- HEX CONVERSIONS')

105 1 ASCII$2$HEX$WORD: PROCEDURE(BUF,STATUS) WORD;
106 2 DECLARE (BUF,STATUS) POINTER,
107 2 CHAR BASED BUF (1) BYTE,
108 2 ERROR BASED STATUS BYTE,
109 2 HEXWORD WORD, I BYTE;
110 2 ERROR = fALSE;
111 2 DO I = 0 TO 3;
112 3 IF CHAR(I) >= 30H AND CHAR(I) <= 39H THEN
113 4 CHAR(I) = CHAR(I) AND OFH;
114 3 ELSE IF (CHAR(I) OR 20H) >= 61H AND (CHAR(I)
115 3 OR 20H) <= 66H THEN
116 4 CHAR(I) = (CHAR(I) AND 0FH) + 09H;
117 3 ELSE ERROR = tRUE;
118 3 HEXWORD = SHL(HEXWORD,4);
119 3 HEXWORD = HEXWORD OR CHAR(I);
120 3 END;
121 2 RETURN HEXWORD;
122 2 END ASCII$2$HEX$WORD;

123 1 HEX$2$ASCII: PROCEDURE(HEXWORD,BUF,BYTE$WORD);
124 2 DECLARE BUF POINTER,HEXWORD WORD,
125 2 (BYTESWORD,LIMIT,I) BYTE,
126 2 ASCIISARY (*) BYTE DATA ('0123456789ABCDEF'),
127 2 CHAR BASED BUF (1) BYTE;
128 2 LIMIT = (2*BYTE$WORD) + 1;
129 2 DO I = 0 TO LIMIT;
130 3 CHAR(LIMIT - I) = LOW(SHR(HEX$WORD,4*I)) AND
131 3 LO$NYBBLE;
132 3 CHAR(LIMIT - I) = ASCIISARY (CHAR(LIMIT - I)
133 3 );
134 2 END;
135 2 END HEX$2$ASCII;

136 1 ASCII$DECIMAL$2$HEX: PROCEDURE (BUFFER,NUMCHAR
137 2 ,STATUS) WORD;
138 2 DECLARE BUFFER POINTER,STATUS POINTER,
139 2 CHAR BASED BUFFER (1) BYTE,
140 2 ERROR BASED STATUS BYTE,
141 2 (HEXWORD,TMP) WORD,(NUMCHAR,I) BYTE;
142 2 ERROR = fALSE;
143 2 HEXWORD = 0;
144 2 IF NUMCHAR = 0 THEN RETURN HEXWORD;
145 2 DO I = 0 TO NUMCHAR - 1;
146 3 TMP = CHAR(NUMCHAR - 1 -I);
147 3 TMP = TMP AND 00FFH;
148 3 IF TMP < 30H OR TMP > 39H THEN
149 4 DO:
150 5 ERROR = tRUE;
151 4 RETURN HEXWORD;
152 4 END;
153 3 TMP = TMP AND 000FH;
DO CASE I;
    TMP = TMP * 10;
    TMP = TMP * 100;
    TMP = TMP * 1000;
END;
HEXWORD = HEXWORD + TMP;
END;
RETURN HEXWORD;
END ASCII$DECIMAL$2$HEX;

ASCII$2$HEX: PROCEDURE (TWO$ASCII) BYTE;
DECLARE TWO$ASCII WORD,
    (T$HEX1,T$HEX2,TMP$CHK$SUM) BYTE;
IF LOW(TWO$ASCII) <= '9' THEN
    T$HEX1 = LOW(TWO$ASCII) - 30H;
ELSE
    T$HEX1 = LOW(TWO$ASCII) - 37H;
IF HIGH(TWO$ASCII) <= '9' THEN
    T$HEX2 = HIGH(TWO$ASCII) - 30H;
ELSE
    T$HEX2 = HIGH(TWO$ASCII) - 37H;
TMP$CHK$SUM = SHL(T$HEX2,4) OR (T$HEX1 AND OFH);
CHK$SUM = CHK$SUM + TMP$CHK$SUM;
RETURN TMP$CHK$SUM;
END ASCII$2$HEX;

MAKEWORD: PROCEDURE WORD;
DECLARE FULWORD WORD;
FULWORD = DOUBLE (LOC$BUF(BUF$POS));
CALL GETMORE;
FULWORD = SHL(FULWORD,8) OR LOC$BUF(BUF$POS);
CALL GETMORE;
RETURN FULWORD;
END MAKEWORD;

DECODE: PROCEDURE(GOOF);
/* COMMAND LINE CONVERSION FOR D,S COMMANDS */
DECLARE (FIRSTDATA,ENDFIRSTDATA,
    SECONDDATA,ENDECONDDATA) BYTE,
    (POS,STATUS) BYTE,GOOF POINTER,
    ERROR BASED GOOF BYTE;

CONVERT$DATA: PROCEDURE(TYPE,POS,NUMBYTES,STATUS);
DECLARE (TYPE,POS,NUMBYTES,GOOF) BYTE,
    STATUS POINTER,ERROR BASED STATUS BYTE;
ERROR = FALSE;
DO CASE TYPE:
    /* ASCII ADD. TO HEX ADD. */
/* ASCII -- HEX CONVERSIONS */

MEM.SEGMENT = ASCII2HEXWORD(@LOCBUF(POS),@GOOF);
MEM.OFFSET = ASCII2HEXWORD(@LOCBUF(POS) + 5),@GOOF);

CMDREP = AsciiDecimal2Hex(@LOCBUF(POS), NUMBYTES,@GOOF);

IF GOOF = true THEN ERROR = true;
END CONVERT$DATA;

SPACE: PROCEDURE (POS,STATUS) BYTE;
DECLARE POS BYTE,STATUS POINTER,
DONE BASED STATUS BYTE;
DONE = false;
DO WHILE LOC$BUF(POS) <> SPC;
IF POS = NUMCHAR -1 THEN
DO;
DONE = true;
RETURN POS + 1;
END;
POS = POS + 1;
END;
RETURN POS;
END SPACE;

NONSPACE: PROCEDURE (POS,STATUS) BYTE;
DECLARE POS BYTE,STATUS POINTER,
DONE BASED STATUS BYTE;
DONE = false;
DO WHILE LOC$BUF(POS) = SPC;
IF POS = NUMCHAR -1 THEN
DO;
DONE = true;
RETURN POS + 1;
END;
POS = POS + 1;
END;
RETURN POS;
END NONSPACE;

/* MAIN CODE FOR DECODE */

POS = SPACE(0,@STATUS); /* FIND FIRST SPACE */
IF STATUS = true THEN
DO: /* SHORT COMMAND */
CMDSREP = 1;
NEWVALUE = ZERO;
RETURN;
END;
FIRSTDATA = NONSPACE(POS,@STATUS); /* FIND FIRST CHARACTER */
IF STATUS = true THEN
DO:
ASCII -- HEX CONVERSIONS

```plaintext
223  3  CMD$REP = 1;
224  3  NEWVALUE = ZERO;
225  3  RETURN;
226  3  END;
227  2  ENDFIRSTDATA = SPACE(FIRSTDATA,@STATUS);

/* CHECK FIRST DATA FOR TYPE AND CONVERT */

228  2  IF LOC$BUF(FIRSTDATA + 4 ) = COLON AND
229  2  FIRSTDATA + 4 < NUMCHAR THEN
230  3  CALL CONVERTDATA (0,FIRSTDATA,4,@ERROR);
231  3  CMD$REP = 1;
232  3  END;
233  2  ELSE DO;
234  3  VALUE = LOCBUF(FIRSTDATA);
235  3  VALUE = SHL(VALUE,8) OR LOCBUF(FIRSTDATA + 1);
236  3  IF VALUE <> '**' THEN
237  3  CALL CONVERTDATA (1,FIRSTDATA,ENDFIRSTDATA-FIRSTDATA,
238  3  A,@ERROR);
239  3  END;
240  2  IF STATUS = tRUE THEN
241  3  NEWVALUE = 3;
242  3  RETURN;
243  3  END;

244  2  SECONDDATA = NONSPACE (ENDFIRSTDATA,@STATUS);
245  2  IF STATUS = tRUE THEN RETURN;
246  2  ENDESECONDDATA = SPACE(SECONDDATA,@STATUS);

/* CHECK SECOND DATA FOR TYPE AND CONVERT */

248  2  IF LOC$BUF (SECONDDATA + 4 ) = COLON AND
249  2  SECONDDATA + 4 < NUMCHAR THEN
250  3  CALL CONVERTDATA (0,SECONDDATA,4,@ERROR);
251  3  NEWVALUE = 2;
252  3  END;
253  2  ELSE DO;
254  3  VALUE = LOCBUF(SECONDDATA);
255  3  VALUE = SHL(VALUE,8) OR LOCBUF(SECONDDATA + 1);
256  3  IF VALUE <> '**' THEN
257  3  CALL CONVERTDATA (1,SECONDDATA,ENDESECONDDATA-SECOND
258  3  DATA,@ERROR);
259  3  NEWVALUE = 2;
260  3  END;
261  1  CMD$PROC: PROCEDURE;
262  2  CALL MOVB (@CMDCPU,@OUTCPU,LENGTH(CMDCPU));
```
ASCII -- HEX CONVERSIONS

263  2  OUT$CPU (10) = CPUID OR 3OH;
264  2  OUT$CPU (1 + LAST(CMDCPU)) = CR;
265  2  OUT$CPU (2 + LAST(CMDCPU)) = LF;
266  2  CALL SET$MY$TURN;
267  2  CALL producer(CPU$ID,SDM$CPU,OUT$CPU,LENGTH(CM - DCPU)+2,@STATUS);
268  2  CALL CMD$MODE (CPU$ID,0,1,@STATUS);
269  2  END CMD$PROC;
$SUBTITLE ('SUBSTITUTE')

SUBSTITUTE: PROCEDURE;

DECLARE (STATUS, USERCODE, MODE) BYTE,
BPPTR POINTER, BRK$PTR STRUCTURE
- (OFF WORD, SEG WORD) AT (@BPPTR), BPLOC BASED B
- PPTR BYTE,
BPCODE LITERALLY 'OCCH';

STATUS = FALSE;
VALUE = ZERO;
MODE = 3;
CALL DECODE (@STATUS);
IF VALUE = '**' THEN MODE = 1;
ELSE MODE = NEWVALUE;
IF MODE >= 0 AND MODE <= 2 THEN
DO CASE MODE:
  DO; /* CLEAR BREAKPOINT */
  IF BREAKPOINTFLAG = TRUE THEN
    DO:
      BPLOC = USERCODE;
      BREAKPOINTFLAG = FALSE;
    END;
  END;
  DO; /* CLEAR OLD BREAKPOINT, SET NEW BREAKPOINT */
  IF BREAKPOINTFLAG = TRUE THEN
    DO;
      BPLOC = USERCODE;
      BREAKPOINTFLAG = FALSE;
    END;
  END;
  BREAKPOINTFLAG = TRUE;
  BRK$PTR.SEG = MEM.SEGMENT;
  BRK$PTR.OFF = MEM.OFFSET;
  USERCODE = BPLOC;
  BPLOC = BPCODE;
END;

DO; /* SUBSTITUTE USER CODE */
CHKSUM = ZERO;
BYTE$LOC(0) = ASCIIS2SHEX(VALUE);
END;
ELSE
CALL ERROR;
END SUBSTITUTE;
RUN

$SUBTITLE ('RUN')

RUN: PROCEDURE PUBLIC;

/* GO$USER$CODE$PTR CODE ENTRY POINT
MACHINE LANGUAGE ROUTINE TO POP USER
REGISTERS FROM XFER STACK AND START
EXECUTING USER CODE BY EXECUTING AN
INTERRUPT RETURN INSTRUCTION */

DECLARE GO$USER$CODE (*) BYTE DATA
(05DH, /* DUMMY POP*/
  05DH, /* DUMMY POP*/
  05DH, /* POP BP */
  05FH, /* POP DI */
  05EH, /* POP SI */
  05BH, /* POP BX */
  05AH, /* POP DX */
  059H, /* POP CX */
  058H, /* POP AX */
  01FH, /* POP DS */
  007H, /* POP ES */
  0CFH    /* IRET AUTOMATICALLY
POPS IP, CS, FLAGS */),
GO$USER$CODE$PTR POINTER DATA (@GO$USER$CODE);

/* THIS PROCEDURE LOADS THE XFER STACK
BEFORE TRANSFERRING TO THE USER CODE*/

CALL CMD$PROC;
DO I = 2 TO 15;
  XFER$STACK (I) = COPY$REGISTERS (I);
END;

STACKBASE = XFER$STACK$ADR.SEG;
STACKPTR = XFER$STACK$ADR.OFF + 4;
CALL SET$NEXT$TURN;
CALL GO$USER$CODE$PTR;
END RUN;

SAVE$USER$REGISTERS: PROCEDURE;
DECLARE I BYTE;
BP=XFER$STACK(0);
XFER$STACK$ADR.OFF = XFER$STACK$ADR.OFF - 2;
DO I = 1 TO 13;
  COPY$REGISTERS (I) = XFER$STACK (I);
END;
XFER$STACK$ADR.OFF = XFER$STACK$ADR.OFF + 28;
SS = XFER$STACK$ADR.SEG;
SP = XFER$STACK$ADR.OFF;
END SAVE$USER$REGISTERS;
INT$3: PROCEDURE INTERRUPT 3;

XFER$STACK$ADR.OFF = STACK$PTR;
XFER$STACK$ADR.SEG = STACKBASE;
STACKBASE = DBUGSTACKBASE;
STACK$PTR = DBUGSTACK$PTR;
CALL SAVE$USER$REGISTERS;
IP = IP - 1;
NEWVALUE = ZERO;
CALL SET$MY$TURN;
CALL SUBSTITUTE;

CALL MOVB(@TASKDONE,@OUTCPU,LENGTH(TASKDONE));
OUTCPU (10) = CPUID OR 30H;
OUTCPU (1+LAST(TASK$DONE)) = CR;
OUTCPU (2+LAST(TASK$DONE)) = LF;
CALL producer (CPUID,SDMCPU,@OUTCPU,LENGTH(CMD)
                   - CPU) + 2,@STATUS);
CALL CMD$MODE (CPUID,0,1,@STATUS);
CALL CMD$MODE (CPUID,1,0,@STATUS);
CALL SET$NEXT$TURN;
GOTO CMD;
END INT$3;}
LOAD: PROCEDURE;
DECLARE ERROR (*) BYTE DATA
('ERROR - CHECK FILE NAME'),
LOD (*) BYTE DATA ('LOADING '),
DONE (*) BYTE DATA
('*** THIS IS THE END OF THE FILE ***'),
(NOT$DONE,TMP1,TMP2,CS1,CS2,IP1,IP2,TMPBYTE,
RECSLEN,NEXT$TWO1,NEXT$TWO2,RECTYPE,CHKSUM1)
BYTE;

CALL CMD$PROC;
XFER$STACKSPTR = @COPY$REGISTERS;
CALL CMD$MODE (CPUID,1,0,@STATUS);
CALL consumer(CPUID,SDM$CPU,@LOCBUF,@NUMCHAR,@
STATUS);
IF LOCBUF(0) = 'e' THEN
CALL PRODUCER(CPUID,SDM$CPU,0ERROR,LENGTH(ERR
OR),@STATUS);
ELSE IF LOCBUF(0) = 'r' THEN
DO;
CALL PRODUCER(CPUID,SDM$CPU,LOD,LENGTH(LO
D),@STATUS);
BUFSPOS,NUMCHAR = ZERO;
NOT$DONE = tRUE;
CALL GET$MORE;
DO WHILE NOT$DONE;
IF LOC$BUF(BUF$POS) = COLON THEN
DO;
CHKSSUM = ZERO;
CALL GET$MORE;
RECSLEN = ASCII$2$HEX(MAKEWORD);
NEXT$TWO1 = ASCII$2$HEX(MAKEWORD);
NEXT$TWO2 = ASCII$2$HEX(MAKEWORD);
RECTYPE = ASCII$2$HEX(MAKEWORD);
DO CASE RECTYPE:
DO;
/** CASE 0 -- DATA RECORD */
MEM.OFFSET = NEXT$TWO1;
MEM.OFFSET = SHL(MEM.OFFSET,8) 0
R NEXT$TWO2;
DO I = 0 TO RECSLEN - 1;
TMPBYTE,BYTE$LOC(I) = ASCII$2$HEX(MAKEWORD);
IF TMPBYTE <> BYTELOC(I) THEN
DO;
CALL CHG$LOAD$MODE (5);
GOTO QUIT;
END;
END;
/** CASE 1 -- END OF FILE */
NOT$DONE = FALSE;
CALL CHG$LOAD$MODE (1);
GOTO QUIT;
END;

- /* CASE 2 -- EXTENDED ADDRESS */

TMP1 = ASCII$2$HEX(MAKEWORD);
TMP2 = ASCII$2$HEX(MAKEWORD);
MEM.SEGMENT = TMP1;
MEM.SEGMENT = SHL(MEM.SEGMENT,8)

- OR TMP2;

END;

- /* CASE 3 -- START ADDRESS */

CS1 = ASCII$2$HEX(MAKEWORD);
CS2 = ASCII$2$HEX(MAKEWORD);
CS = CS1;
CS = SHL(CS,8) OR CS2;
IP1 = ASCII$2$HEX(MAKEWORD);
IP2 = ASCII$2$HEX(MAKEWORD);
IP = IP1;
IP = SHL(IP,8) OR IP2;

END;

CHKSUM1 = ASCII$2$HEX(MAKEWORD);
IF CHKSUM MOD 256 <> ZERO THEN
DO;
CALL CHG$LOAD$MODE (2);
GO TO QUIT;
END;
ELSE
DO;
CALL GET$MORE;
CALL GET$MORE;
END;
ELSE
DO;
CALL CHG$LOAD$MODE (4);
NOT$DONE = fALSE;
END;
END:
QUIT: CALL CMD$MODE(CPUID,1,0,@STATUS);
CALL SET$NEXT$TURN;
END LOAD:
$SUBTITLE ('REGISTER VIEW/CHNGE')

427 1 GET$REG: PROCEDURE (REG$ID);
428 2 DECLARE REG$ID BYTE;

429 2 IF REG$ID = ABSENT THEN RETURN;
430 2 LOC$BUF(0) = HIGH(REGSARY(REG$ID));
431 2 LOC$BUF(1) = LOW(REGSARY(REG$ID));
432 2 LOC$BUF(2) = '=';
433 2 CALL HEX$2$ASCII(COPY$REGISTERS(REG$ID),@LOC$BUF(3),1);
434 2 LOC$BUF(7) = CR;
435 2 LOC$BUF(8) = LF;
436 2 END GET$REG;

437 1 CHANGE$REG: PROCEDURE(REG$ID);
438 2 DECLARE (ERROR,REG$ID) BYTE,TMP WORD;

439 2 IF REG$ID = ABSENT THEN RETURN;
440 2 TMP = ASCII$2$HEX$WORD(@LOC$BUF(5),@ERROR);
441 2 IF ERROR <> 1 THEN
442 3 DO;
443 4 COPY$REGISTERS (REG$ID) = TMP;
444 4 CALL GET$REG(REG$ID);
445 3 END;
446 2 END CHANGE$REG;

447 1 INDEX: PROCEDURE(REG$NAME) WORD;
448 2 DECLARE REG$NAME WORD, I BYTE;

449 2 DO I = 1 TO LAST(REGSARY);
450 3 IF REG$NAME = REGSARY(I) THEN
451 4 RETURN COPY$REGISTERS(I);
452 3 END;
453 2 END INDEX;
SET LINE COMMAND

**$SUBTITLE ('SET LINE COMMAND')**

1. **LINE**: PROCEDURE(LINES$NUMBER);
2. DECLARE (LINES$NUMBER,I,J) BYTE,
   LINE$PTR POINTER,
   CHAR BASED LINE$PTR (1) WORD,
   LINE0 (*) WORD DATA ('AXCSIPFL'),
   LINE1 (*) WORD DATA ('BXSSSPBP'),
   LINE2 (*) WORD DATA ('CXDSSI'),
   LINE3 (*) WORD DATA ('DXESDI');

DO CASE LINES$NUMBER:

1. **LINE$PTR** = @LINE0;
2. J=27;
3. END;
4. **LINE$PTR** = @LINE1;
5. J=27;
6. END;
7. **LINE$PTR** = @LINE2;
8. J=18;
9. END;
10. **LINE$PTR** = @LINE3;
11. J=18;
12. END;
DO I = 0 TO J BY 9;
1. LOC$BUF(I) = HIGH(CHAR(I/9));
2. LOC$BUF(I+1) = LOW(CHAR(I/9));
3. CALL HEX$2$ASCII (INDEX(CHAR(I/9)),@LOC$BUF(I+3),1);
4. LOC$BUF(1+7) = "\";
5. LOC$BUF(1+8) = '\';
6. LOC$BUF(3+7) = CR;
7. LOC$BUF(3+8) = LF;
8. CALL PRODUCER (CPUID,SDMCPU,@LOCBUF,J+9,@STATUS);
9. END CASE;

**REG$ID**: PROCEDURE BYTE,
1. DECLARE (I,REG) BYTE,
2. TMP WORD;
3. TMP = LOC$BUF(2) AND UPPERCASE;
4. TMP = SHL(TMP,8) OR (LOC$BUF(3) AND UPPERCASE);
5. DO I = 2 TO LAST(REGSARY);
6. IF TMP = REGSARY(I) THEN RETURN I;
7. ELSE REG = ABSENT;
8. END;
9. RETURN REG;
10. END REG$ID;
$SUBTITLE('DISPLAY COMMAND')

499 1  show$register: PROCEDURE;
500 2  DECLARE REG BYTE;
501 2  CALL CMD$PROC;
502 2  IF NUM$CHAR = 2 THEN
503 2  DO;
504 3  CALL CMD$MODE(CPUID,0,4,@STATUS);
505 3  CALL LINE(0);
506 3  CALL LINE(1);
507 3  CALL LINE(2);
508 3  CALL LINE(3);
509 3  CALL CMD$MODE(CPUID,1,0,@STATUS);
510 3  CALL SET$NEXT$TURN;
511 3  RETURN;
512 3  END;
513 2  REG = REG$ID;
514 2  IF REG = ABSENT THEN
515 2  DO;
516 3  CALL ERROR;
517 3  RETURN;
518 3  END;
519 2  ELSE IF LOC$BUF(4) = '=' THEN
520 2  CALL CHANGE$REG(REG);
521 2  ELSE IF NUM$CHAR = 4 THEN
522 2  CALL GET$REG(REG);
523 2  CALL CMD$MODE(CPUID,0,1,@STATUS);
524 2  CALL PRODUCER(CPUID,SDM$CPU,@LOC$BUF,9,@STATUS -
525 2  CALL CMD$MODE(CPUID,1,0,@STATUS);
526 2  CALL SET$NEXT$TURN;
527 2  END show$register;
528 1  OUTMEMORY: PROCEDURE;
529 2  DECLARE (NUMLINES,PARCHR) WORD,
530 2  (J,K,I,FNISH) BYTE,DISPLAY$PTR POINTER,
531 2  CHAR BASED DISPLAY$PTR (1) BYTE,
532 2  OUT STRUCTURE (LWAD WORD,UPAD WORD)
533 2  AT (@DISPLAY$PTR),
534 2  S STRUCTURE (BUF1 (5) BYTE,BUF2 (4) BYTE,
535 2  BUF3 (50) BYTE,BUF4 (20) BYTE);
536 2  OUT.UPAD = MEM.SEGMENT;
537 2  OUT.LWAD = MEM.OFFSET;
538 2  IF CMD$REP <= 16 THEN
539 3  DO;
540 4  FINISH = CMD$REP;
541 4  NUMLINES = 1;
542 3  END;
543 2  ELSE
544 3  DO;
545 4  IF (OUT.LWAD AND 000FH) =0 THEN FINISH =0;
546 4  ELSE FINISH = 16 - (OUT.LWAD AND 000FH);
547 3  CMD$REP = CMD$REP - FINISH;
548 3  PARCHR = CMD$REP MOD 16;
549 3  CMD$REP = CMD$REP - PARCHR;
NUMLINES = CMDSREP/16;

IF FINISH <> 0 THEN NUMLINES = NUMLINES + 1;
IF PARCHR <> 0 THEN NUMLINES = NUMLINES + 1;
END;

CALL CMD$MODE (CPUID, 0, NUMLINES, @STATUS);
IF FINISH = 0 THEN FINISH = 16;
DO K = 1 TO NUMLINES;
CALL HEX$2$ASCII (OUT.UPAD, @S.BUF1(0), 1);
S.BUF1(4) = COLON;
CALL HEX$2$ASCII (OUT.LWAD, @S.BUF2(0), 1);
DO I = 0 TO FINISH - 1;
CALL HEX$2$ASCII (CHAR(I), @S.BUF3((I*3)+1), 0);
S.BUF3((I*3)+1) = SPC;
IF CHAR(I) >= 21H AND CHAR(I) <= 7EH THEN
S.BUF4(I+1) = CHAR(I);
ELSE S.BUF4(I+1) = PERIOD;
END;
S.BUF3(48) = SPC;
S.BUF3(49) = SPC;
CALL PRODUCER(CPUID, SDMCPU, $S, 63+J, @STATUS);
OUT.LWAD = OUT.LWAD + FINISH;
IF K+1 = NUMLINES AND PARCHR <> 0 THEN
FINISH = PARCHR;
ELSE FINISH = 16;
END;
CALL CMD$MODE (CPUID, 1, 0, @STATUS);
CALL SET$NEXT$TURN;
END OUTMEMORY;

SHOW$MEMORY: PROCEDURE;
DECLARE STATUS BYTE;

CALL CMD$PROC;
STATUS = FALSE;
CALL DECODE(@STATUS);
IF STATUS = TRUE THEN
DO:
CALL ERROR;
RETURN;
END;
CALL OUTMEMORY;
END show$memory;
INITIALIZATION

$SUBTITLE('INITIALIZATION')

STARTT = DISABLE;
RESTART = FALSE;
CPU$ID = set$cpu$Id;
IF CPU$ID = ZERO THEN
  DO;
    EXITT = FALSE;
    SDM$START = FALSE;
    CPU$START = TRUE;
    TRNLCK = FALSE;
    CALL init$buf$mon;
    CALL init$cpu$list;
    CPU$LISTS.SYS$CPU(CPU$ID) = CPU$ID;
    CPU$START = FALSE;
    DO WHILE NOT CPU$START;
    END;
    CALL CPU$ORDER;
    SDM$CPU = MAX$NUM$CPUS;
    CALL producer(CPU$ID,SDM$CPU,@SIGNON,LNGTH(SIGNON),@STATUS);
    CPU$START = FALSE;
  END;
ELSE
  DO;
    CPU$START = TRUE;
    DO WHILE CPU$START;
    END;
    CPU$LISTS.SYS$CPU(CPU$ID) = CPU$ID;
    CPU$START = TRUE;
    DO WHILE CPU$START;
    END;
  END;
END;
/* END OF INITIALIZATION */
$SUBTITLE('COMMAND PROCESSING LOOP')

/*
START COMMAND PROCESSING LOOP */

625 1 EXITT = FALSE;
626 1 SDM$START = TRUE;
627 1 CALL SET$INTERRUPT (32, SPURIOUS$INTERRUPT);
628 1 CALL SET$INTERRUPT (3, INT$3);
629 1 BREAKPOINTFLAG = FALSE;
630 1 ENABLE;
631 1 DEBUGSTACKBASE = STACKBASE;
632 1 DEBUGSTACKPTR = STACKPTR;
633 1 CMD: DO WHILE TRUE;
634 2 CALL consumer(CPU$ID, SDM$CPU, @LOC$BUF, @NUMCH
- AR, @STATUS);
635 2 IF STATUS = FALSE THEN
636 2 DO I = 0 TO LAST(ASCII$COMMAND$ARY);
637 3 IF (LOC$BUF(0) OR LOWER$CASE) = ASCII$COMMAND$ARY(I) THEN DO;
639 4 COMMAND = I;
640 4 GOTO JUMP$TABLE;
641 4 END;
642 3 ELSE COMMAND = 99;
643 3 END;
644 2 ELSE

/ * COMMAND JUMP TABLE */

645 2 JUMP$TABLE:
- IF COMMAND >= 0 AND COMMAND <= LAST(ASCII$COMMAND$ARY) THEN
646 2 DO:
647 3 DO CASE COMMAND:
648 4 CALL LOAD; /* L COMMAND */
649 4 CALL RUN; /* G COMMAND */
650 4 CALL SHOW$REGISTER; /* X COMMAND */
651 4 CALL SHOW$MEMORY; /* D COMMAND */
652 4 DO;
653 5 CALL CMD$PROC;
654 5 CALL SUBSTITUTE; /* S COMMAND */
655 5 CALL CMD$MODE(CPU$ID, 1, 0, @STATUS);
656 5 CALL SET$NEXT$TURN;
657 5 END:
658 4 RESTART = TRUE; /* E COMMAND */
659 4 RUNN = TRUE; /* GA COMMAND */
660 4 END;
661 3 IF RESTART = TRUE THEN
662 3 DO:
663 4 DO WHILE gOSALL = TRUE;
664 5 END;
665 4 RESTART = FALSE;
666 4 GOTO STARTT;
667 4 END;
668 3 IF RUNN = TRUE THEN
669 3 DO:
670 4 DO WHILE gOSALL = TRUE;
END;
RUNN = false;
END;
END;
ELSE
DO;
CALL SET$MY$TURN;
CALL ERROR;
CALL CMD$MODE(CPUID,1,0,0STATUS);
CALL SET$NEXT$TURN;
GOTO CMD;
END;
END;
END MMDE$main;

MODULE INFORMATION:

CODE AREA SIZE = 16A2H 5794D
CONSTANT AREA SIZE = 0000H 0D
VARIABLE AREA SIZE = 0178H 376D
MAXIMUM STACK SIZE = 0046H 70D
994 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
APPENDIX P

UEXT SOURCE CODE
ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE MMDEEXT
OBJECT MODULE PLACED IN :F1:UEXT.OBJ
COMPILER INVOKED BY: PLM86 :F1:UEXT.SRC

$DATE(AUG-08-86) PAGEWIDTH(60)
$LARGE

1

DECLARE SHARED$MEMORY LITERALLY '8000H',
BUFSIZE LITERALLY '80D',
MAXNUMCPUS LITERALLY '5D',
TOTALCPUS LITERALLY '6D',
FALSE LITERALLY '00H',
TRUE LITERALLY '0FFH',
DBUF LITERALLY '01D',

/* SHARED MEMORY DEFINITIONS */
/* JULY 12, 1986 */

SEMAPHORE BYTE AT (SHARED$MEMORY),
SEMAPHORE1 BYTE AT(SHARED$MEMORY + 1),
SEMAPHORE2 BYTE AT (SHARED$MEMORY + 2),
SEMAPHORE3 BYTE AT (SHARED$MEMORY + 3),
COQUEUE$BUSY BYTE AT (SHARED$MEMORY + 4),
PRQUEUE$BUSY BYTE AT (SHARED$MEMORY + 5),
BUFFER$FULL BYTE AT (SHARED$MEMORY + 6),
LOAD$MODE BYTE AT (SHARED$MEMORY + 7),
SIGNAL$BUSY BYTE AT (SHARED$MEMORY + 8),
(PRHEAD, PRTAIL, PRCOUNT) BYTE AT(SHARED$MEMORY + 9),
(COHEAD, COTAIL, COCOUNT) BYTE AT(SHARED$MEMORY + 12),
DBG(DBG) BYTE AT(SHARED$MEMORY+TOTALCPUS+9),

/* VARIABLES USED FOR PROCESSOR ORDERING */

CPU$LISTS STRUCTURE (ORDER(MAXNUMCPUS) BYTE,
TURN (MAXNUMCPUS) BYTE,
SYSSCPU (MAXNUMCPUS) BYTE)
AT (SHARED$MEMORY + TOTALCPUS + DBUF + 9),

(MODE$INDEX, GROUP$COMMAND) BYTE AT
(SHARED$MEMORY+TOTALCPUS+DBUF+MAXNUMCPUS
+ MAXNUMCPUS + MAXNUMCPUS + 9),

MODE STRUCTURE (CMD BYTE, MESSAGE BYTE) AT
(SHARED$MEMORY + TOTALCPUS + DBUF + MAXNUMCPUS
+ MAXNUMCPUS + MAXNUMCPUS + 11),
/* SEMAPHORE DEFINITIONS */

(TRNLCK, ACT$CPU, SDM$CPU, SDMS$START, CPU$START,
EXITT, GOSALL, GLOBALS$CMD$CHK) BYTE
AT (13 + SHAR$MEMORY + TOTAL$CPUS + DBUF +
MAX$NUM$CPU + MAX$NUM$CPUS + MAX$NUM$CPUS),

/* SYSTEM BUFFER STRUCTURE */

SYS$BUF STRUCTURE(SOURCE$CPU BYTE, DEST$CPU BYTE,
NUMSCHAR WORD, DATTA (BUFSIZE) BYTE) AT
(SHAR$MEMORY + DBUF + TOTAL$CPUS +
MAX$NUM$CPU + MAX$NUM$CPUS + MAX$NUM$CPUS + 21):

DECLARE CPUID BYTE PUBLIC,
STARTT LABEL PUBLIC,
(AFTN, NUMCHAR) WORD,
(OLDSTACKPTR, OLDDSTACKBASE) WORD,
STATUS, ACTUAL, i, NUMS$MES, CPUNUM) BYTE,
CHRS$COUNT BYTE.
LOC$BUF(80) BYTE, NAME(15) BYTE,
NO$MESSAGE LITERALLY '00H',
NOT$BUSY LITERALLY '02H';

EXIT: PROCEDURE EXTERNAL;
END EXIT:

OPEN: PROCEDURE (AFTN, FILE, ACCESS, ECHOAFTN, STA-
TUS) EXTERNAL;
DECLARE (AFTN, FILE, STATUS) POINTER,
(ACCESS, ECHOAFTN) WORD;
END OPEN:

CLOSE: PROCEDURE (AFTN, STATUS) EXTERNAL;
DECLARE AFTN WORD, STATUS POINTER;
END CLOSE:

READ: PROCEDURE (AFTN, BUFFER, COUNT, ACTUAL, STA-
TUS) EXTERNAL;
DECLARE (AFTN, COUNT) WORD,
(BUFFER, ACTUAL, STATUS) POINTER;
END READ:

consumer: PROCEDURE (RCVR$CPU, SENDER$CPU, MESS$PT-
R, NUMSCHAR, STATUS) EXTERNAL;
DECLARE (MESSPTR, STATUS, NUMSCHAR) PO-
INTER,
(RCVR$CPU, SENDER$CPU) BYTE;
END consumer:

producer: PROCEDURE (SOURCE$CPU, DEST$CPU, MESS$PT-
R, NUMSCHAR, STATUS) EXTERNAL;
DECLARE (MESSPTR, STATUS) POINTER,
NUMSCHAR WORD, (SOURCE$CPU, DEST$CPU)
- BYTE:
86 2

END producer;
sys$error: PROCEDURE (ERR$NUM);
DECLARE (ERR$NUM,I) BYTE,
       BUF(40) BYTE,
       MES0 (*) BYTE DATA ('ERROR MODE STATUS.'),
       MES1 (*) BYTE DATA ('ILLEGAL ERROR #.'),
       MES2 (*) BYTE DATA ('CONSUMER - NO MESSAGE.'),
       MES3 (*) BYTE DATA ('PRODUCER - NO MESSAGE.'),
       MES4 (*) BYTE DATA ('PRODUCER - OKAY.'),
       MES5 (*) BYTE DATA ('LOAD MODE ERROR.'),
       MES6 (*) BYTE DATA ('CHECK SUM ERROR.'),
       MES7 (*) BYTE DATA ('LOAD OKAY.'),
       MES8 (*) BYTE DATA ('BUFFER POINTER ERROR.'),
       MES9 (*) BYTE DATA ('WRITE ERROR - ROM.');
IF ERR$NUM >= 0 AND ERR$NUM <= 9 THEN
   DO CASE ERR$NUM;
      CALL MOVB (@MES0,@BUF,LENGTH(MES0));
      CALL MOVB (@MES1,@BUF,LENGTH(MES1));
      CALL MOVB (@MES2,@BUF,LENGTH(MES2));
      CALL MOVB (@MES3,@BUF,LENGTH(MES3));
      CALL MOVB (@MES4,@BUF,LENGTH(MES4));
      CALL MOVB (@MES5,@BUF,LENGTH(MES5));
      CALL MOVB (@MES6,@BUF,LENGTH(MES6));
      CALL MOVB (@MES7,@BUF,LENGTH(MES7));
      CALL MOVB (@MES8,@BUF,LENGTH(MES8));
      CALL MOVB (@MES9,@BUF,LENGTH(MES9));
   END;
ELSE
   CALL MOVB (@MES1,@BUF,LENGTH(MES1));
   CALL OUT$CRLF;
   CALL MCO ('*');CALL MCO('*');CALL MCO('*');CALL MCO('*');
   I = 0;
   DO WHILE BUF(I) <> '.
      CALL MCO(BUF(I));
      I = I + 1;
   END;
   CALL MCO ('*');CALL MCO('*');CALL MCO('*');CALL MCO('*');
   CALL OUT$CRLF;
END SYS$ERROR;
SIGN$ON: PROCEDURE;
DECLARE CLEAR LITERALLY 'OCH',
       (NUM$CHAR,I) WORD, STATUS BYTE;
CALL MCO (CLEAR): CALL MCO ('*');
/* GET SIGNON MESSAGE */
EXITT = FALSE;
CALL CONSUMER (SDM$CPU,0,@LOC$BUF,0@NUM$CHAR,@S
       STATUS);
   /* OUTPUT SIGNON MESSAGE TO SCREEN */
IF STATUS = TRUE THEN
   CALL sys$error(1);
ELSE
   DO I = 0 TO NUMSCHAR-1;
   CALL MCO (LOC$BUF(I));
   CALL MCO('*');CALL MCO('*');
   END SIGNSON;

   VALID$CPU: PROCEDURE(CPUNUM) BYTE;
   DECLARE (CPUNUM,I) BYTE;
   IF CPUNUM >= MAXNUMCPUS THEN RETURN FALSE;
   ELSE DO I = 0 TO LAST(CPUSLISTS.SYS$CPU);
      IF CPUNUM = CPUSLISTS.SYS$CPU(I) THEN RETURN TRUE;
   END;
   RETURN FALSE;
   END VALID$CPU;

   WHOS$HERE: PROCEDURE;
   DECLARE I BYTE;
   DO I = 0 TO MAXNUMCPUS-1;
      IF VALID$CPU (I) = TRUE THEN DO;
         CALL MCO (I OR 30H);
         CALL MCO (2OH);
      END;
   END;
   CALL OUT$CRLF;
   END WHOS$HERE;

   PROMPT: PROCEDURE (PROMPTSCHAR);
   DECLARE PROMPTSCHAR BYTE,
       BELL LITERALLY '07H';
   CALL OUT$CRLF; CALL MCO(BELL);
   CALL MCO(PROMPTSCHAR);CALL MCO('>');
   END prompt;

   COUNT: PROCEDURE(COUNTT);
   DECLARE (I,COUNTT) BYTE;
   IF COUNTT = 0 THEN I = 0;
   I = I + 1;
   IF I = 80 THEN CALL MCO('.');
LOAD

DECLARE (CPUID, NOT$DONE) BYTE,
READ$ONLY LITERALLY '01H',
NOECHO LITERALLY '00H',
(Status1, ACTUAL, I, COUNTT) WORD,
DONE (*) BYTE DATA

('*** THIS IS THE END OF THE FILE ***');

COUNTT = 0;
LOAD$MODE = 3;
STATUS1 = FALSE;
CALL OPEN (@AFTN, @NAME(0), READONLY, NOECHO, STAT
US1);

IF STATUS1 <> FALSE THEN

DO;

LOC$BUF (0) = 'e';
CALL PRODUCER (SDM$CPU, ACT$CPU, @LOCBUF, 1, @ST
ATUS);
CALL CONSUMER (SDM$CPU, ACT$CPU, @LOCBUF, @NUM
CHAR, @STATUS);
DO I = 0 TO NUMCHAR - 1;
CALL MCO (LOCBUF(I));
END;

NOT$DONE = TRUE;
DO WHILE NOT$DONE;
COUNTT = 1;
IF LOAD$MODE >= 0 AND LOAD$MODE <= 4 THEN

DO:

/* CASE 0 - MORE CODE TO LOAD */
CALL READ (AFTN, @LOCBUF, 80, @ACTUAL,
$BUF, ACTUAL, @STATUS);

/* CASE 1 - LOAD DONE NO ERRORS */
CALL SY$ERROR(7);

/* CASE 2 - LOAD DONE ERRORS */
CALL SY$ERROR(6);
210 6  NOT$DONE = FALSE;
211 6  END;
212 5  DO;
213 6  /* CASE 3 -- WAIT FOR NEXT MODE */
214 6  END;
215 5  DO;
216 6  /* CASE 4 -- BUFFER ERROR */
217 6  CALL SYS$ERROR (8);
218 6  NOT$DONE = FALSE;
219 5  END;
220 6  /* CASE 5 -- WRITE ERROR */
221 6  CALL SYS$ERROR (9);
222 6  NOT$DONE = FALSE;
223 5  END;
224 4  ELSE CALL SYS$ERROR (5);
225 4  END;
226 3  END;
227 2  CALL CLOSE (AFTN,@STATUS1);
228 2  END LOAD;
COMMAND EDITOR

PROCEDURE COMMAND_EDIT (PROMPT_CHAR);

DECLARE (CMD$NOT$VALID, STATUS, CHAR, PROMPT$CHAR) BYTE,
BS LITERALLY '08H', DEL LITERALLY '7FH',
CR LITERALLY '0DH', SP LITERALLY '020H',
BEL LITERALLY '0111',
LF LITERALLY '0AH', ESC LITERALLY '01BH', CNTR LITERALLY '12H',
CNTE LITERALLY '05H', ASCIIA LITERALLY '061H';

START:
CHAR$COUNT = 0;
CMD$NOT$VALID = tRUE;
DO WHILE
CMD$NOT$VALID AND CHAR$COUNT <= 80;
IF mODE.CMD = 0 THEN do; /*Call mco('1');*/
return; end;
CHAR = MCI;
IF CHAR = CNTE AND CHAR$COUNT = 0 THEN
DO;
CALL MCO('E');
NAME(0) = 'E';
gO$ALL = tRUE;
CALL ALL$CMD;
geXITT = tRUE;
RETURN;
END;
ELSE IF CHAR or 20H = ASCII AND CHAR$COUNT = 1 THEN
DO;
CALL MCO (CHAR);
NAME(0) = ASCII;
gO$ALL = tRUE;
CALL ALL$CMD;
geO$ALL = FALSE;
RETURN;
END;
ELSE IF CHAR$COUNT = 1 AND CHAR <= 39H AND CHAR >= 30H THEN
DO;
IF VALID$CPU (CHAR AND OFH) = tRUE THEN
DO;
CALL MCO (CHAR);
NAME(0) = ASCII;
gO$ALL = tRUE;
CALL ALL$CMD;
geO$ALL = FALSE;
RETURN;
END;
ELSE IF CHAR$COUNT = 1 AND CHAR <= 39H AND CHAR >= 30H THEN
DO;
IF VALID$CPU (CHAR AND OFH) = tRUE THEN
DO;
CALL MCO (CHAR);
NAME(0) = ASCII;
LOC$BUF(1) = CHAR;
CHAR$COUNT = CHAR$COUNT + 1;
END;
ELSE
   DO;
   275  CALL MCO(BEL);
   CALL MCO(BS):CALL MCO(SP):CALL MCO(BS);
   END;
   ELSE IF CHAR = BS OR CHAR = DEL AND CHAR$COUNT > 0 THEN
      DO;
      280  CALL MCO(BL):CALL MCO(SP):CALL MCO(BS);
      IF CHAR$COUNT > 0 THEN
         CHAR$COUNT = CHAR$COUNT - 1;
      END;
      ELSE IF CHAR = CR AND CHAR$COUNT = 0 THEN
         CALL PROMPT(PROMPT$CHAR);
      ELSE IF CHAR = BS OR CHAR = DEL AND CHAR$COUNT > 1 THEN
         CMD$NOT$VALID = FALSE;
      ELSE
         CALL MCO(BL);
         CALL MCO(BS);CALL MCO(SP);CALL MCO(BS);
         END;
         ELSE IF CHAR = ESC THEN
            CALL MCO(BL);
         ELSE IF CHAR = CR AND CHAR$COUNT > 1 THEN
            CMD$NOT$VALID = FALSE;
         ELSE
            CALL MCO(CHAR);
            LOC$BUF(CHAR$COUNT) = CHAR;
            CHAR$COUNT = CHAR$COUNT + 1;
            END;
            END;
            IF CHAR$COUNT >= 81 THEN
               DO;
               CALL MCO(BL);
               CALL PROMPT(PROMPT$CHAR);
               GOTO START;
               END;
               ELSE
                  IF (LOC$BUF(0) OR 20H) = '1' THEN
                     DO;
                     LOCSBUF(CHAR$COUNT) = SP;
                     CHAR$COUNT = CHAR$COUNT + 1;
                     CALL MOVB($LOC$BUF(3), $NAME(0), 15);
                     END;
                     CALL PRODUCER(SDM$CPU, LOCSBUF(1) AND OFH, @
                     LOCSBUF, CHAR$COUNT, @STATUS);
                     END;
                     ELSE END command$edit;
$SUBTITLE ('TO TERMINAL PROCEDURE')

TO$TERM: PROCEDURE (NUM$MESSAGES);
    DECLARE (J,NUM$MESSAGES) BYTE,
        (I,NUM$CHAR) WORD;
    IF NUM$MESSAGES = 0 THEN RETURN;
    ELSE
        DO J = 1 TO NUM$MESSAGES;
        CALL consumer(sdm$cpu,0,@locbuf,@NUM$CHAR,@
            status);
        DO I = 0 to NUM$CHAR-1;
        CALL MCO(LOC$BUF(I));
        END;
    END;
    END TO$TERM;
$SUBTITLE('MMDE.EXT PROGRAM LOOP')

UEXT: PROCEDURE PUBLIC;

DECLARE NEWSTACKPTR LITERALLY '00FFH',
    NEWSTACKBASE LITERALLY '07FOH',
    UEXTMODE BYTE;

    /* SAVE STACK POINTER */
OLDSTACKBASE = STACKBASE;
OLDSTACKPTR = STACKPTR;

    /* SET NEW STACK POINTER */
STACKBASE = NEWSTACKBASE;
STACKPTR = NEWSTACKPTR;

    /* SET CPUID FOR USE IN BUFMON */
CPUID = sDMCPU;

    /* EXECUTE TERMINAL COMMANDS */
DO WHILE NOT sDM$START;
END;
gOSALL = 0;
CALL SIGN$ON;
CALL WHO$HERE;
DO WHILE NOT eXITT;
IF mode.CMD >= 0 AND mode.CMD <= 3 THEN
DO CASE mode.CMD;
    DO;
        NUM$MES = mode.MESSAGE;
        mode.CMD = NOT$BUSY;
        CALL TO$TERM(NUM$MES);
    END;
    DO;
        mode.CMD = NOT$BUSY;
        CALL PROMPT('*');
        CALL COMMAND$EDIT(' * ');
        CALL OUT$CRLF;
    END;
    DO;
DO;
    END;
    DO;
    END;
    END;
ELSE
DO;
    CALL SYS$ERROR (0);
    mode.CMD = 2;
    END;
    END;
STACKBASE = OLDSTACKBASE;
STACKPTR = OLDSTACKPTR;
CALL EXIT;
372  2  END UEXT;
$SUBTITLE ('MAIN PROGRAM START')

/* SAVE CURRENT STACK POINTER */

373 1 STARTT: ;
374 1 RESTART: ;
375 1 CALL UEXT;
376 1 END MMDE$EXT;

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE AREA SIZE</td>
<td>0AE5H</td>
</tr>
<tr>
<td>CONSTANT AREA SIZE</td>
<td>0000H</td>
</tr>
<tr>
<td>VARIABLE AREA SIZE</td>
<td>00B3H</td>
</tr>
<tr>
<td>MAXIMUM STACK SIZE</td>
<td>0026H</td>
</tr>
<tr>
<td>614 LINES READ</td>
<td></td>
</tr>
<tr>
<td>0 PROGRAM ERROR(S)</td>
<td></td>
</tr>
</tbody>
</table>

END OF PL/M-86 COMPILATION