ADAPTATION OF MCORTEX TO THE REGIS SIMULATION ENVIRONMENT.
U.S. NAVAL POSTGRADUATE SCHOOL, MONTEREY, CA.
W. R. ROME.
JUN 84.
UNCLASSIFIED.
THESIS

ADAPTATION OF MCORTEX TO
THE AEGIS SIMULATION ENVIRONMENT

by

Willis R. Rowe

June 1984

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution unlimited
**Adaptation of MCORTEX to the AEGIS Simulation Environment**

**WILLIS RALPH ROWE**

**Naval Postgraduate School**

Monterey, CA 93943

**June 1984**

**139**

**Unclassified**

**Approved for public release; distribution unlimited**

**Operating System; Microcomputers; Real-time processing; Multiprocessing; Distributed Computer Networks**

This thesis presents the adaptation of a multi-computer real-time executive, MCORTEX, to a target environment consisting of a set of INTEL 86/12A single board computers in a MULTIBUS back plane. CP/M-86 is brought under the control of MCORTEX, and mechanisms are implemented to provide access to the MCORTEX supervisor from Digital Research’s PL/I-86 language system. Initially CP/M-86 is operating the system of micro-computers in a multiuser mode. MCORTEX and user processes are loaded from...
CP/M-86 files. Use of all CP/M-86 functions is retained and MCORTEX can be used by PL/I-96 compiled applications programs to do multi-processing.
Approved for public release: Distribution unlimited

Adaptation of MCOPTEX to the AESIS Simulation Environment

by

Willis R. Rowe
Lieutenant, United States Navy
P.S., University of Kansas, 1977

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June, 1984

Author:

Willis R. Rowe

Approved by:

Thesis Advisor

Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences
ABSTRACT

This thesis presents the adaptation of a multi-computer real-time executive, MCCORTEX, to a target environment consisting of a set of INTEL 86/12A single board computers in a MULTIFUS back plane. CP/M-86 is brought under the control of MCCORTEX, and mechanisms are implemented to provide access to the MCCORTEX supervisor from Digital Research's PL/I-86 language system.

Initially CP/M-86 is operating the system of microcomputers in a multi-user mode. MCCORTEX and user processes are loaded from CP/M-86 files. Use of all CP/M-86 functions is retained and MCCORTEX can be used by PL/I-86 compiled applications programs to do multi-processing.
DISCLAIMER

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis will be listed below, following the firm holding the trademark:

1. INTEL Corporation, Santa Clara, California
   INTEL
   IS486/12A
   ISIS-II
   8086

2. Digital Research, Pacific Grove, California
   CP/M-86
   PL/I-86
   DDT-86

   68000
   LINK-86
   ASM-86
# Table of Contents

I. Introduction ........................................ 17
   A. General Discussion .................................. 12
   B. Background .......................................... 11
   C. Structure of the Thesis ............................ 13

II. Implementation Modification Issues ................. 15
   A. Design Considerations ............................. 15
   B. Shared Resources .................................. 15
   C. Process Integrity .................................. 16
   D. Interprocess Synchronization ...................... 16
   E. Deleted Functions .................................. 21

III. System Architecture ............................... 22
   A. System Hardware ................................... 22
   B. Operating Systems .................................. 25
   C. User Processes ..................................... 26

IV. MCoRTex LoadPE .................................... 32
   A. KORE.OPS / KORE.TRC ............................... 32
   B. KORE AS CMD FILE .................................. 33
   C. Operation of the MCoRTex Loader ................. 34

V. PL/I-66 Compatibility ................................ 36
   A. The Supervisor ..................................... 36
   B. PL/I-66 Parameter Passing Conventions .......... 38
   C. PL/M Preentrant Parameter Passing ............... 38
   D. Generating MCoRTex Processes using PL/I-66 .... 42

VI. Conclusions .......................................... 47
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ISIS-II to CP/M-86 Core Transfer Procedure</td>
<td>50</td>
</tr>
<tr>
<td>B</td>
<td>Mcortex Under DDT 57</td>
<td>85</td>
</tr>
<tr>
<td>C</td>
<td>Mcortex Loader Source Code</td>
<td>57</td>
</tr>
<tr>
<td>D</td>
<td>Gate Module Source Code</td>
<td>68</td>
</tr>
<tr>
<td>E</td>
<td>Demonstration Program Source Code</td>
<td>75</td>
</tr>
<tr>
<td>F</td>
<td>Link/6 Input Option Files</td>
<td>80</td>
</tr>
<tr>
<td>G</td>
<td>Level II Mcortex Source Code</td>
<td>83</td>
</tr>
<tr>
<td>H</td>
<td>Level I Mcortex Source Code</td>
<td>126</td>
</tr>
<tr>
<td>I</td>
<td>Scheduler &amp; Interrupt Handler Source Code</td>
<td>128</td>
</tr>
<tr>
<td>J</td>
<td>Global Database and Initial Process Code</td>
<td>133</td>
</tr>
<tr>
<td>K</td>
<td>List of Differences</td>
<td>137</td>
</tr>
<tr>
<td>L</td>
<td>Initial Distribution List</td>
<td>138</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1. IMPLEMENTATION ENVIRONMENT .................................. 23
2. MEMORY ALLOCATION .............................................. 27
3. PL/I-96 MODULES .................................................. 29
4. PL/I-96 PARAMETER PASSING ..................................... 38
5. PL/M PEERENTRANT PARAMETER PASSING ......................... 41
LIST OF TABLES

1. GLOBAL MEMORY...................................... 16
2. MAP FILE............................................ 44
I. INTRODUCTION

A. GENERAL DISCUSSION

This thesis presents the adaptation of a kernel, real-time micro-computer based multi-processor operating system, called MCCORTEX, to allow simultaneous user access to the CP/M operating system as well as to MCCORTEX. User program development using Digital Research's PL/I-86 language system is supported.

Improvement in micro-processor capabilities, and performance, combined with continued reductions in hardware cost portend the development of powerful, relatively inexpensive micro-processor systems. Continued success in VLSI technology applications in parallel with development of appropriate operating systems will produce systems superior in many respects to computers developed using current mainframe technology. Systems of processors allow for graceful degradation under fault conditions and for distribution of the system, enhancing survivability in hostile environments. Further, parallel processing allows increased throughput and response time, and in real time application can guarantee successful monitoring at high sample rates and densities, without conflict.

A successful multi-processor system must control sequencing of inter-independent processes and access to
limited resources. For efficiency it must provide the context switching necessary for multi-processing on individual processors. Additionally, conflicts arising from simultaneous multi-processor access to common memory must be minimized without degrading throughput. This should be accomplished at a reasonable cost and in a manner that allows as many processors as are necessary to achieve the desired degree of concurrency and robustness.

The purpose of this thesis is to advance the development of a real time multi-processor system within the overall goals of the AEGIS weapons system simulation project. These goals include the demonstration of the operating system on commercially available, inexpensive, general purpose microcomputers. The system should require minimum development of both hardware and software. To the maximum extent possible, custom developments should be completely general in nature. In pursuit of these goals, MOCORTEX is configured to execute in conjunction with a commercially available operating system, making the functions of both systems available to user programs. Additionally, mechanisms allowing user program development within the framework of a commercially available language system are provided.

B. BACKGROUND

The AEGIS weapons system relies on the four-processor AN/UYK-7 mainframe computer for real-time processing of large amounts of data concerning target detection and
discrete or can overlap on boundaries that are multiples of 16 bytes, depending on segment register values.

The iSBC96/12A provides serial I/O through an INTEL 8251A USART, parallel I/O through an INTEL 8255A PPI and a broad range of interrupt control through the INTEL 8259A PIC. MCORTEX operates using interrupt 4. The interrupt is generated via output to parallel port 5, as proposed by Perry [Ref. 5: pp. 65 to 69]. Both the hardware and software implementations are exactly as presented by Perry.

II. OPERATING SYSTEMS

A copy of MCORTEX resides in each processor's local memory and is a distributed part of the address space of each local process. Additionally, GLOBAL memory is accessible to MCORTEX to facilitate interprocess synchronization. A system interrupt under MCORTEX control, in conjunction with interrupt flags maintained in GLOBAL memory, provides communication initiation between real processors. Upon receiving an interrupt, each processor checks its flag in GLOBAL memory to determine if the interrupt is intended for a process in its local memory. If not, the process executing at the time of the interrupt continues. Otherwise a call is made to the MCORTEX scheduler and the highest priority ready process is given control of the CPU. For communication between processes in a common local memory, no interrupt is issued, a call to the scheduler is made directly.
Currently used wiring option excludes off-board access to local RAM. Differences between memory access times at the first two levels are negligible, but memory accesses involving MULTIBUS require a minimum 25% increase in access time.

The high performance, general purpose 8086 microprocessor base of the iSEC86/12A contains an Execution Unit (EU) and a Bus Interface Unit (BIU). EU functions are supported by instruction fetches and operand reads and writes conducted by the BIU. The BIU can stack instructions in an internal RAM to a level of six deep increasing EU efficiency and decreasing bus idle time.

The 8086 has eight 16-bit general purpose registers, four being byte addressable. The remaining four are primarily pointer registers, but can be used as accumulators. Additionally, the 8086 has four segment registers, an instruction pointer register and a flag register with nine status bits.

A segmented one megabyte address space is provided for by the 8086 microprocessor. This is accomplished by combining the 16 bits of each segment register left shifted four bits, with the 16 bits of an associated pointer register unshifted. The resulting 20 bits form a physical address. For any given segment register value 64k bytes of memory can be addressed through manipulation of the pointer register alone. The 64k byte memory spaces formed can be
FIG. 1 IMPLEMENTATION ENVIRONMENT
III. SYSTEM ARCHITECTURE

A. SYSTEM HARDWARE

This implementation of MCORTEX is based on the INTEL iSBC86/12A single board computer using a MULTIBUS backplane. Specific, detailed information pertaining to both these components is available in [Ref. 3] and [Ref. 1]. The MULTIBUS also connects two memory extensions into the system. A 32K extension is used as common memory for interprocess communication under MCORTEX and for CP/M multi-user system control. A 64K extension provides additional memory required to operate the PL/I-E6 compiler and other utilities not constrained to execute in the 64K of memory local to each processor. Additionally, a bubble memory system and a hard disk system are available on MULTIBUS. A second hard disk system is accessed through the parallel port of one SBC. Figure 1 is a representation of this configuration with two SBC's shown.

The iSBC86/12A provides a three level hierarchical bus structure. At the first level, the 8086 processor communicates through the on board bus with up to 4K of ROM, with serial and parallel I/O ports and with the dual-port bus. Control and access to local RAM is provided by the second level dual-port bus. The third bus level, the MULTIPUS interface, provides access to the MULTIBUS. The
pointer. (This information is stored in GLOBAL memory.)

Third, since each process was uniquely identified by its stack segment register, MCOPTEX functions did not need to identify the process they were responding to when using the scheduler. The scheduler simply stored state information at the base of whatever stack segment was active when the scheduler was called.

The assumption that each process used a different stack segment value is not completely general, and in fact was not true for procedures compiled and linked under the Digital Research PL/I-86 language system. This conflict forced changes in the context switching mechanisms of MCOPTEX. The entire stack pointer (SS and SP registers) is now stored in GLOBAL memory, and MCOPTEX functions making use of the scheduler must indicate (in the Processor Data Segment Table, PRS) which virtual processor they are servicing.

E. DELETED FUNCTIONS

Functions previously available under MCOPTEX include OUTSCHR, OUTSLINE, OUTSNUM, OUTSDNUM, INSSCHR, INSSNUM, and INSSDNUM. With CP/M-86 under the control of MCOPTEX, these utility functions are redundant and have been removed. However, a version of MCOPTEX with these functions incorporated has been retained for troubleshooting purposes. The monitor process incorporated by Klinefelter has also been removed in light of the availability of DDT86.
of portability. The context switching routines in MCORETEX, including the virtual processor scheduler and the interrupt handler, are the only portions of the MCORETEX core written in assembly language. Another decision motivated by the speed imperative assumed that each virtual processor owned a stack for storing state information. This decision was followed by another assuming that the stack segment pointer for each stack was different. This allowed a bootstrap like context recovery. A search through the virtual processor map identified the highest priority ready process. Virtual processor map information included the process stack segment value. This value was retrieved, and subsequently used to retrieve three additional pieces of processor state information. Offsets zero, two, and four from the stack segment were used to retrieve the process stack pointer value, the process stack base pointer value, and a return type indicator. Recovery of the stack state allowed recovery of the entire state of the virtual processor, and processing could continue.

This context switching method has many advantages. First, once the stack segment of a process has been stored in a known, retrievable location, it never needs updating. The base of the stack remains fixed, and access is controlled through the contents of the first few bytes at the base of the stack. Second, less space is required to store the stack segment than to store the entire stack.
always assigned to the highest priority ready virtual processor on each board regardless of which synchronization function invoked the scheduler. Before using ADVANCE or AWAIT, an event count must be created using CREATE$EVC. Consumers and producers then communicate using the agreed upon event count. The current value of an event count can be determined through a call on READ. The functions of CREATE$SEQ and TICKET are as discussed earlier, but with broader applications.

MCORTEX handles two types of context switching. The first type results when control of a CPU is relinquished through a MCORTEX function call. Under these conditions the calling process is not halted in the midst of some task, but at a place 'convenient' for the process. Some subset of the processors registers contains all required state information. MCORTEX assumes this subset includes the DS, IP, CS, SS, SP, and BP registers. Additionally, a "normal" return indicator is saved. The second type of context switching results from an interrupt. This switching assumes nothing, and saves the complete state of the process being interrupted as well as an 'interrupt' return indicator. This recognition of two switch types makes context switching faster for the more common 'normal' return.

Early implementers of MCORTEX considered the context switching overhead question in detail. Their solution gave greater importance to the issue of speed than to the issue
If the resource is not busy, the process will receive immediate access, otherwise the process gives up the CPU.

C. PROCESS INTEGRITY

The design of MCORTEX relies heavily on user cooperation for process integrity. The supervisor controls access to the MCORTEX functions, but even this is a software control and will not withstand malicious assault or catastrophic failure. MCORTEX is targeted at embedded systems applications where malicious assault is not expected. Protection from catastrophic failure requires hardware protection not presently in the system. The low cost of micro-computers however, allows for redundant back up systems which can limit the affects of catastrophic failure.

D. INTERPROCESS SYNCHRONIZATION

Process synchronization is accomplished under MCORTEX through the functions ADVANCE, AWAIT, and PREEMPT. These synchronizing primitives are supported with the functions CREATESEVC, CREATESEQ, READ, and TICKET. Consumer processes use AWAIT to ensure that data they require is ready. Producer processes use ADVANCE to inform consumers that new data has been computed. PREEMPT is used by one process to directly ready another process. This primitive is for activation of high priority system processes of a time critical nature. A call on a synchronizing primitive may, or may not result in relinquishing the CPU. The CPU is
lock, it is given sole access to the common bus for one instruction cycle. During this cycle, the process makes an exchange of the value in a register (contents 77H) with $GLOBAL\_LOCK$. The processor then examines the contents of the exchange register. If the register now contains zero, the processor is granted access; if not, the process repeats the procedure until a zero is obtained from $GLOBAL\_LOCK$.

Because access to $GLOBAL$ memory is controlled by $MCORTEX$, waits should be infrequent and short in duration. When relinquishing the software lock, the process merely sets $GLOBAL\_LOCK$ to zero.

Users have no access to $GLOBAL$ memory, however $MCORTEX$ provides for user control of shared resources through data held in $GLOBAL$ memory. Sequencers, located in the sequencer table section of $GLOBAL$ memory, are used to provide a turn taking mechanism. Each shared resource is assigned a corresponding sequencer. When processes require a resource, they request a turn through the supervisory function call $TICKET$, specifying the applicable sequencer. $TICKET$ returns a number indicating the callers turn at the required resource. This is similar to getting a turn number at a barber shop. $TICKET$ advances the sequencer value in global memory so that succeeding requests receive higher numbers. The process requesting the resource then makes another supervisory call, this time on $AWAIT$, providing both an identification of the resource and the process turn number.
logical organization of this shared resource (see the last four pages of Appendix H for actual locations.). Access to

TABLE 1: GLOBAL MEMORY

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>MNEMONIC</th>
<th>TYPE/INIT</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GLOBAL$LOCK</td>
<td>B 0</td>
<td>Number of real processors</td>
</tr>
<tr>
<td>1</td>
<td>V%PPS</td>
<td>B 0</td>
<td>Number of virtual processors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>one byte for each possible CPU,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MAX$CPU currently = 10)</td>
</tr>
<tr>
<td>2</td>
<td>NR$VPS(MAX$CPU)</td>
<td>F 0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>HDW$INT$FLAG(MAX$CPU)</td>
<td>B X</td>
<td>Hardware interrupt flag (one for each possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU, MAX$CPU currently = 10)</td>
</tr>
<tr>
<td>22</td>
<td>EVENTS</td>
<td>B 1</td>
<td>Number of events</td>
</tr>
<tr>
<td>23</td>
<td>EVC$TBL(100)</td>
<td>S</td>
<td>Event count table</td>
</tr>
<tr>
<td>24</td>
<td>VALU$E</td>
<td>W 0</td>
<td>Event count value</td>
</tr>
<tr>
<td>26</td>
<td>THREAD</td>
<td>B FF</td>
<td>Event count thread</td>
</tr>
<tr>
<td>423</td>
<td>CPU$INIT</td>
<td>E 0</td>
<td>Log in CPU number</td>
</tr>
<tr>
<td>424</td>
<td>SEQUENCE$S</td>
<td>B 0</td>
<td>Number of sequencers</td>
</tr>
<tr>
<td>425</td>
<td>SEQ$TBLF(100)</td>
<td>S</td>
<td>Sequencer table</td>
</tr>
<tr>
<td>426</td>
<td>SEQ$NAME</td>
<td>B X</td>
<td>Name of sequencer</td>
</tr>
<tr>
<td></td>
<td>SEQ$VALUE</td>
<td>W X</td>
<td>Value of sequencer</td>
</tr>
<tr>
<td></td>
<td>VPM(MAX$CPU * MAX$VPMS$CPU)</td>
<td>S</td>
<td>Virtual processor map</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(MAX$CPU currently = 10,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MAX$VPMS$CPU currently = 10)</td>
</tr>
<tr>
<td>725</td>
<td>VP$ID</td>
<td>P X</td>
<td>Virtual processor ident.</td>
</tr>
<tr>
<td>726</td>
<td>VP$STATE</td>
<td>B X</td>
<td>Virtual processor state</td>
</tr>
<tr>
<td>727</td>
<td>VP$PRIORITY</td>
<td>B X</td>
<td>Virtual processor priority</td>
</tr>
<tr>
<td>728</td>
<td>EVC$NAME$VALUE</td>
<td>W X</td>
<td>Count awaitei</td>
</tr>
<tr>
<td>730</td>
<td>SP$REG</td>
<td>W X</td>
<td>Stack pointer register</td>
</tr>
<tr>
<td>732</td>
<td>SS$REG</td>
<td>W X</td>
<td>Stack segment register</td>
</tr>
<tr>
<td>1725</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R - byte    W - word    S - structure    X - not initialized

GLOBAL memory is controlled through the combination of a hardware bus lock, and a software lock (GLOBAL$LOCK) located in GLOBAL memory. When a process sets the hardware bus...
II. IMPLEMENTATION MODIFICATION ISSUES

A. DESIGN CONSIDERATIONS

In a real-time system, multi-processing on a single processor decreases processor idle time. A multi-processor configuration extends the range of this economy and provides opportunities to exploit parallel and pipeline processing techniques that further enhance overall system goals. Careful consideration must be given to control of shared resources, process integrity, interprocess synchronization, methods of context switch initiation, and context switching overhead.

3. SHARED RESOURCES

The most important shared resource in a multi-processor environment is common memory. MCCORTEX relies on a hierarchical bus structure to limit the requirement for access to common memory. Each processor has local memory, addressable without access to a shared bus. A process executing in local memory makes demands on the common bus only to pass computed data to external processes, or when MCCORTEX functions are used. Related processes with high intercommunication rates should reside in the local memory of a single processor, thus avoiding high common bus usage.

To perform its functions, MCCORTEX sets up a section of common memory called GLOBAL memory. Table 1 shows the
Chapter III details the architecture of the MCORTEX environment, highlighting interactions between the hardware, CP/M-86 and MCORTEX.

Chapter IV presents the MCORTEX loader, discussing considerations given to alternative methods for invoking MCORTEX.

Chapter V explains the interface provided between PL/I-86 and the MCORTEX supervisor. Procedures necessary to successfully create MCORTEX virtual processors are discussed.

Chapter VI summarizes the current state of the system, points out problem areas, and makes suggestions for future research and testing.
and execution accomplished via download through a serial link to the target hardware.

The goals of this thesis are to:

1. Bring the powerful, highly portable functions of the CP/M-86 operating system under the control of MCORTEX. This will provide rapid expansion of user capabilities within the restrictions imposed by the non-reentrancy of CP/M-86 utilities. Using MCORTEX functions, control of access to CP/M-86 can be selectively applied depending on the contextual requirements of the application.

2. Sever the link with the development system, and provide a simple, convenient method of creating the MCORTEX environment. This should include user program and MCORTEX loading, transfer of control to MCORTEX, and mechanisms for return of control to CP/M-86.

3. Provide access mechanisms to the MCORTEX supervisor compatible with Digital Research's PL/I-86 language system, allowing user programs to be developed in a high level, portable language.

C. STRUCTURE OF THE THESIS

Chapter I discusses the overall direction of the AEGIS weapons system simulation project and the place this thesis holds in accomplishing project goals.

Chapter II addresses the issues which resulted in changes to MCORTEX as implemented by Klinefelter, and presents an overview of the MCORTEX functions.
acquisition. A project at the Naval Postgraduate School seeks to demonstrate that a system as complex as AEGIS can be controlled more economically, with improvements in graceful degradation characteristics, and without performance loss using a distributed system of microcomputers. The project requires identification and implementation of an applicable hardware configuration, development of a suitable operating system, duplication of significant real-time functions of the AEGIS weapon's system and incorporation of valid simulation processes for test and evaluation of the total system.

The INTEL iSFC 86/12A, a single board micro-computer based on the 16 bit INTEL 8086 micro-processor, was selected as the system hardware base. Initial design of an operating system specific to the INTEL iSFC 86/12A was completed in 1982 and implementation was accomplished in three Naval Postgraduate School thesis in 1981 and 1982. The second thesis in this series written by Cox [Ref. 1] simplified the design of MCORTEX to more successfully address security and overhead issues in the real-time embedded applications targeted by the project. Cox also added a supervisory layer to the architecture, simplifying access and enhancing security. Klinefelter [Ref. 2] expanded and generalized Cox's work. All implementation to this point was done on the ISIS-II development system, with multi-processor test
Access to MCORTEX is through the supervisor at the outermost layer of the MCORTEX four level structure discussed by Klinefelter [Ref. 2 : pp. 44-46]. Due to incompatible parameter passing implementations in PL/M-86, and in PL/I-86, code allowing PL/I-86 access to the MCORTEX supervisor has been developed. This is discussed fully in Chapter V.

Also resident in each local memory, if required, is the CP/M-86 operating system. In this configuration the full range of CP/M utilities, [Ref. 6] and [Ref. 7], is available to the user. Additionally, development of user processes can make use of any of the broad scope of commercially available products compatible with CP/M-86. Figure 2 gives a representation of the locations of the system code. The diagram includes the location of DDT-46 as required for a debugging session. Also depicted are the locations of the MCORTEX / MXTACF loaders. During load, loader memory is not reserved, and care must be taken to ensure that a CMD module's code or data section does not overwrite it. It is permissible, however, to include this memory as part of a module stack or free space, since these structures are developed at module runtime when loader functions have been completed.

C. USEP PROCESSES

User processes may be located in areas indicated in Figure 2. Additionally, if CP/M-86 utilities are not
FIG. 2 MEMORY ALLOCATION
required, memory reserved for CP/M-86 may hold user processes.

Descriptions of processes in memory are provided to MCORTEX through CREATE$PROC. This MCORTEX function gives the process a unique identification number, priority, stack (SS and SP registers), next execution address (CS and IP registers), data segment (DS register), and extra segment (ES register). MCORTEX establishes the process initial context using this information to create a virtual processor. The virtual processor exists as a combination of data, both in GLOBAL memory, and in each process stack. When executing, the virtual processor becomes identical with the real processor state. Relinquishing the CPU forces the virtual processor again into GLOBAL memory and the process stack.

Special effort has been made to accommodate processes created under PL/I-86 and linked using LINK86. The internal architecture of such processes requires some consideration. LINK86 concatenates all PL/I-86 code segments into one segment. The same is done with data segments. Thus, PL/I-86 processes consist of a series of contiguous code segments followed by a series of contiguous data segments. Additionally, at run time PL/I-86 routines create a stack following the data area, and a free space following the stack. The resulting process configuration is shown in Figure 3.
CS REG.

USER PROCEDURE NUMBER 1
  
  USER PROCEDURE NUMBER n

PL/I-86 RUNTIME MODULE NUMBER 1
  
  PL/I-86 RUNTIME MODULE NUMBER m

DS, SS, ES REG.

USER DATA AREA NUMBER 1
  
  USER DATA AREA NUMBER n

PL/I-86 DATA AREA NUMBER 1
  
  PL/I-86 DATA AREA NUMBER m

GENERATED AT RUNTIME

RUNTIME STACK

GENERATED AT RUNTIME

FREESPACE

**FIG. 3 PL/I-86 MODULES**
Access to all data areas resulting from a single link, is referenced to a common data segment. Stack pointers are referenced to the stack segment register, and free space pointers to the extra segment register. Additionally, some PL/I-86 runtime routines assume the contents of all three segment registers (DS, SS, ES) are identical. This assumption disallows process stacks with unique stack segments, and was the motivation for modifications to MCCRTEX discussed in Chapter II. For the demonstration programs D1.CMD and D2.CMD (Appendix E) PL/I-86 generated a default stack of size 400H bytes. This area was subdivided to provide a 120H byte process stack and a 2E0H system stack in the case of D1.CMD, and two 120H byte process stacks and a 1C0H byte system stack in the case of D2.CMD. The documentation for PL/I-86 [Ref. E : p. 2.9] describes mechanisms incorporated in the PROCEDURE statement to specify the size of the runtime stack. If these mechanisms function as described, all process stacks can be contained within the area allocated to the runtime stack. Otherwise process stacks can be constructed following the free space. This area would be unprotected by normal CP/M CMD file memory management functions, and its use would require extra care.

The MCCRTEX CREATE$PROC parameters include the absolute location of process start, stack, and data. For this reason it is advantageous to locate processes absolutely when
LINK86 provides such an option [Ref. 9: p. 7.6]. However, the ABSOLUTE option is applicable to the entire CMT file created and cannot be used to distribute the file non-contiguously in memory. Also, experience has shown that the required code segment address must be placed in the data's ABSOLUTE declaration. Further, the code segment ABSOLUTE declaration must hold an address larger than the sum of the value placed in the data ABSOLUTE declaration and the size of the data segment. This value seems to have no effect on the location of the file but, too small a value will cause an error when the file is loaded. See Appendix F for examples of link option files that produce correct results.

MCCRTFX processes may be linked together as PL/I-86 procedures allowing sharing of PL/I-86 runtime routines or may be linked individually. Separate processes require more memory due to replication of PL/I-86 support routines, however, great care is required with shared routines as PL/I-86 runtime routines are not reentrant. Further, CP/M-86 subroutines are neither reentrant nor replicateable. I/O functions, therefore, must be viewed as shared resources and access to them strictly controlled.
IV. MCORTEX LOADER

A. KORE.OPS / KORE.TRC

During development the MCORTEX executive was assigned to the file KORE and was accessible through the INTELLEC MDS system. This file contained all the multi-processor operating system functions, the initial GLOBAL memory, the supervisor, the interrupt vector, and various low level functions not accessible to the user. To execute MCORTEX it was necessary to download KORE and user processes to the target system, disconnect the transfer cable, connect the target system terminals, and pass control to KORE on each processor. See [Ref. 2: Appendix A, E] for a complete description of the process. The KORE.OPS and KORE.TRC files loaded by the MCORTEX and MXTRACF loaders respectively, are derived from the original KORE file with changes as discussed in Chapter II. Additional changes were made to compact the KORE.OPS file, and to relocate the INITSMOD for simpler, more CP/M-66 compatible loading of user processes. Appendix A details the procedure used to produce KORE.OPS and KORE.TRC from KORE. Further discussion will use the terms KORE and MCORTEX to mean either KORE.OPS or KORE.TRC and MCORTEX or MXTRACE respectively. When this generalization does not hold, the differences will be noted.
Currently the MCORTEX environment can be established under the CP/M-86 operating system. Control is then passed to MCORTEX automatically, and user processes are created in the user initialization module. Control can be passed back to the CP/M-86 operating system if applicable.

B. KORE AS CMD FILE

Establishment of the MCORTEX environment through invocation of KORE as a command file is not feasible for several reasons. First, interpretation of CMD file headers assumes each CMD file to be continuously constructed. KORE is not. Second, KORE memory requirements include an interrupt vector. The CP/M-86 memory management system does not allow loading of command files into the interrupt vector space. Third, the data segment for the initialization module depends upon the amount of executable code generated by all processes linked with the module and is not static. The data segment register initial value must be passed to KORE after processes are loaded. Fourth, KORE includes GLOBAL memory, which should be loaded only once, while KORE must be loaded into each processors local memory. An additional consideration is the simplicity and flexibility gained when KORE and user processes are loaded via the same mechanism to produce the MCORTEX environment.
C. OPERATION OF THE MCCORTEX LOADER

MCCORTEX.CMD is an executable file under the CP/M-86 operating system. Invocation of MCCORTEX without KORE.OPS on the default drive results in an error message and return to CP/M-86. MXTRACE requires KORE.TRC. The loader announces that it is on line, and requests an entry to indicate whether or not GLOBAL memory should be loaded. Only the first processor activated should load GLOBAL memory. Subsequent loads of GLOBAL memory would destroy data needed by executing processors. If no initial load of GLOBAL memory is made the results are unpredictable.

KORE is immediately loaded with or without GLOBAL memory as directed. The load is accomplished using CP/M-86 functions, but does not use the CMD load utility. Instead, KORE is read in and positioned block at a time as required. The interrupt vector is not maintained as a part of the KORE files, but is generated within the loader itself with moves directly from loader data memory to the interrupt vector space.

KORE load is followed by a request for a process file name. The loader expects at least one file name to be entered, and results are unpredictable if one is not. User processes are loaded using the CP/M-86 CMD load utility, and user processes must be CMD files. The entire file name must be entered including the three letter extension. After loading the first and subsequent user files, the loader
requests another file name. To exit user process loading, a return with no preceding character should be entered. The last file entered must contain the initialization module, as the data segment register value of this file is determined and passed to KORE.

Completion of user process loading causes control to be passed to MCORTEX. MCORTEX initializations are performed, including creation of the IDLE and INIT processes (also MONITOR with MXTRACE), and the user initialization process is entered. Operation after this point is determined by the user processes. An ADVANCE on the initialization event count 'FF' by any process will halt all processors, returning them to CP/M-86 control. The demonstration programs in Appendix E end with a PREEMPT call to the INIT process. This is only to demonstrate the operation of PREEMPT and, in fact, due to multiple declarations of the INIT process causes only the first processor activated to return to CP/M-86 control.
V. PL/I-96 COMPATIBILITY

A. THE SUPERVISOR

KORE is written in PL/M-86, and requires calls made to the supervisor to meet PL/M-86 parameter passing conventions. Further, the supervisor requires four parameters with every call regardless of the function invoked. To meet parameter passing requirements, and to hide details of the supervisor implementation, a translation mechanism between user calls and the supervisor is required.

The first parameter expected by the supervisor is a byte value indicating the function required. Following the function code should be another byte, a word and a pointer. The formal parameters these actual parameters represent are different for different function calls, and in some cases the values passed are not used at all. The supervisor uses the function code to determine which parameters are applicable, and simply ignores the rest. It is inconvenient and unnecessary for the user to provide unneeded parameters or to remember which function codes belong to which functions.

Two files (see Appendix D) are provided to mitigate differences between simple user calls and supervisor requirements. The file GATEWAY.PLI should be %INCLUDE'd in all programs making calls on MCORTEX functions. It declares
the MCRTEX functions as ENTRY values with attribute lists matching the parameters expected by GATEMOD. Note that entry declarations reserve memory space for the parameters specified. Each user process must have separate memory set aside for these function calls to avoid concurrency problems in GATEMOD.

GATEMOD.OBJ (or GATETRC.OBJ) should be linked with all user processes. It provides the object code necessary to convert user calls to the format expected by the supervisor, including addition of function codes, and padding of calls with extraneous parameters. GATEMOD uses no variable data segment of its own, and simply makes moves from user data areas to the user stack. This ensures that, so long as the user data areas involved are unshared, GATEMOD is reentrant.

Note that all parameters in the GATEWAY declarations are BIT(6) or BIT(16). PL/M has two unsigned integer data types, BYTE and WORD, that are used extensively in MCRTEX. There are no corresponding data types in PL/I-96, and BIT(6) and BIT(16) are the closest available substitutes. In MCRTEX processes it is sometimes convenient to add two BIT(16) numbers. Unfortunately, mathematical computations or BIT(16) values are not supported in PL/I-96. This set of conditions necessitated the development of the function Add2BIT16 included in GATEWAY. As the name implies, this function adds two BIT(16) parameters as unsigned integers and returns the result as a BIT(16) value. If a carry is
orriuceri, it is ignored, and the result returned will, of course, be incorrect.

E. PL/I-86 PARAMETER PASSING CONVENTIONS

Parameters passed in a PL/I-86 procedure call are accessed via an array of pointers [Ref. 17; p. 16.1]. The location of the pointer array is provided to called routines through a pointer in the BX register. Using register indirection and indexing, pointers to actual parameters are loaded into system pointer registers. Parameter values can then be manipulated as required. Figure 4 is a diagrammatical representation of the parameter passing structure that might be established by PL/I-86 for a call on the MCCORTEX supervisor.

All BIT(16) values returned to user programs by the GATE2CD, either as a result of a call to ADE2BIT16 or as a result of calls to the MCCORTEX functions RTAD or TICRE, are returned in the BX register. This is the convention followed by AS/86 based PL/I-86.

C. PL/M REENTRANT PARAMETER PASSING

All MCCORTEX PL/M-86 routines are reentrant. The AS/86 routines lock out interrupts during execution so that reentrancy is not an issue. In particular the MCCORTEX supervisor is reentrant. This is the only KOE module accessible to user processes.
FIG. 4  PL/I-86 PARAMETER PASSING
DDT86 all are resident in the SBC memory and in the 32K shared memory board.

D. Using DDT86 commands, reposition the parts of KORE required so that the code can be saved into one file. Data necessary to determine the initial locations of the code is found in KORE.MP2. The DDT86 instructions used for the current KORE.OPS and KORE.TFC files follows:

*** KORE.OPS ***

MBBC:2,00F,480:0 *** Move, starting at address B80:0, 00F bytes of code (main part of KORE) to new start address 480:0.

M439:0,60,560:0 *** Move, starting at address 439:0, 60 bytes of code (initialization module) to new start address 560:0 (following main part as moved above).

ME794:?,68F,568:0 *** Move, starting at address E794:?, 68F bytes of code (GLOBAL memory) to new start address 568:0 (following initialization module).

WKORE.OPS,480:0,153F *** Write to the default disk a file called KORE.OPS starting at address 480:0 and containing 153F bytes.

*** KORE.TFC ***

M439:0,25,066:0 *** Move, starting at address 439:0, 25 bytes of code (initialization module) to new starting address 066:0 (following main KORE code).

MACP:0,1AFF,439:0 *** Move, starting at address 40:0, 1AFF bytes of code (main KORE - initialization module) to
D. When the interrupt light #2 lights on the front panel, press space bar on the console device.

E. Reset the boot switch by pushing the lower part of the switch.

F. ISIS-II will announce itself and give the '-' prompt.

IV. LOAD MGOREX

A. At MPS console, type "SBC861<CR>".

B. IF "*CONTROL*" appears, SBC was not able to set its baud rate. Press RESET on MULTIPUS frame and try again.

C. If 'Bad YMDS connection' appears, you will not be able to continue. Check connections. Make sure diskette is not write protected. Push RESET at frame. Try again.

D. SBC861 will announce itself and prompt with ".".

E. Type 'L KORE<cr>'. Wait for '. '. At this point the KORE module has been loaded into the SBC memory, and into the shared memory board.

V. SAVING KORE TO CP/M-86 FILE

A. Leaving the SBC861 process active on the MPS system, disconnect the PS232 J2 connector at the SBC, and connect the terminal prepared earlier.

B. At the newly connected terminal type 'GFD4:4<cr>'. The CRT will not echo this entry. Respond to the cues that follow as required until CP/M-86 is up.

C. Now enter DDT86. At this point KORE, CP/M-86, and
II. POWER ON PROCEDURES

A. Turn the power-on key to ON position at MULTIBUS frame.

B. Press RESET near power-on key.

C. If needed apply power to MICROPOLIS hard disk.

D. Apply power to REMEX disk system. After system settles, put START/STOP switch in START position. Following a lengthy time-out period, the READY light on the front of the REMEX disk system will illuminate, and the system is ready. Alternately, the RESET button on the MULTIBUS can be pressed three times, with a small time-out for the system to settle each time. Following the third button press, the READY on the front of the REMEX disk system will illuminate as before.

E. Insert the boot disk into drive B.

F. Apply power to the CRT.

G. Put the Bubble Device RUN/HALT switch to RUN.

H. Power up the MDS disk drive.

I. Power up the MDS terminal.

J. Turn power-on key to ON at MDS CPU.

III. BOOT UP MDS

A. Place diskette with executable modules and SEC61 in drive C.

E. Push upper part of boot switch in (It will remain in that position).

C. Press reset switch and then release it.
APPENDIX A

ISIS-II TO CP/M-86 TRANSFER

I. PRE-POWER-ON CHECKS

A. SBC configured for CP/M-86 cold boot is in MULTIBUS odd slot and no other clock master SBC is installed.

B. Bubble memory is in MULTIBUS.

C. REMEX controller is in MULTIBUS, and properly connected to REMEX drive.

D. If MICROPOLIS hard disk is to be used, ensure that it is connected to clock master SBC.

E. Ensure 32K shared memory module is installed.

F. Connect RS232 transfer cable between J2 on SBC, and 2400 baud CRT port of the MDS system. If this cable has a null modem switch on it, set it to 'null modem'. This transposes wires 2 and 3. The switch may alternately be marked 'computer to computer' and 'computer to terminal'. Set to "computer to computer".

G. Connect any CRT to the 9600 baud TTY port of the MDS system. Ensure CRT is set to 9600 baud.

H. A CRT will be connected to the SBC after the loading is completed, and should have an RS232 cable hooked to the serial port. The CRT connection should lead to a flat 25 wire ribbon and J2 connector so it can eventually be hooked to the SBC's serial port.
outside their assigned data segments. Finally, AEGIS system processes and real-time input simulation processes should be adapted to the MCORTEX environment, and performance measurements made.
As noted earlier, neither CP/M-86 nor PL/I-86 runtime routines are reentrant. Sharing any section of code from either system in a concurrent environment requires care and proper control of access to shared code. In many instances this can be accomplished through application of MCORTEX functions. When proper sequencing through PL/I-86 runtime routines cannot be guaranteed, processes using shared routines must be separated, and multiple links performed. This produces a copy of the runtime routines for each linked group of modules. Since processes not linked into the same CMD file do not share common data structures, communication between the modules becomes more complex. PL/I-86 uses sixteen bit pointers, and has no built-in mechanism to transfer data outside the data segment assigned to the linked module. This deficiency also adversely affects communication through common memory of processes on separate SBC's.

Future research with MCORTEX should investigate the problems discussed above. Testing of the system using more than two SBC's should be conducted. Investigation of the relationships between MCORTEX processes sharing sections of PL/I-86 and CP/M-86 code must be conducted, and the best means of controlling access to shared code determined. If possible, this should be accomplished in a high priority system process that is transparent to users. Some convenient means must be provided to give processes access to memory
VI. CONCLUSIONS

The principal goals of this thesis were met. MCORTEX has been integrated into a selected environment to provide multi-processing and multi-processor capabilities. Assets available under the CP/M-86 operating system have been made available to MCORTEX processes. Also, development of MCORTEX processes in the high-level language PL/I-86 has been provided for through reentrant gateway transformations between PL/I-86 calling structures and the structures expected by the MCORTEX supervisor. Programs have been written to demonstrate that each of the MCORTEX functions can be used from within a process written in PL/I-86. Two versions of the operating system kernel have been produced. One version, found in the KORE.TRC file, retains all diagnostic cues of the development version, primitive I/O functions, and the MONITOR. The second version, found in the KORE.CPS file, has these items removed.

No testing of the system, except to monitor the proper operation of the demonstration programs, has been accomplished. The demonstration programs have been run successfully using two slave SRC's and using the master SRC and one slave SRC. The loader program sometimes will not accept a file name without the drive prefix. No pattern to this behavior has been observed.
The MAP file also contains maps of the individual modules linked into the CMD file. These maps provide data about locations of code and data segments within the larger code and data segments summarized in the segments section. The beginning address of each module is given. This offset represents the IP value for that particular module.

With all parameter values determined, the initialization process must be recompiled, and all processes relinked. The resulting CMD file can be executed in the MCC/TEX environment.
can be determined by performing an executable load of the process CMD file under DTE6. Values displayed by DTE6 include the CS, and DS register values. As mentioned earlier, it is required that the DS, SS, and ES register values be equal for proper operation of some PL/I-96 runtime routines. Except under special carefully considered circumstances, programmers should ensure that this is the case. The remaining two parameters are pointer values obtainable from the link MAP file.

The first section of the MAP file gives a summary of all code and data segments included in the associated CMD file. Several data segments are listed in order of their occurrence in memory, from lowest offset to highest offset. The range of the last entry gives the last address offset occupied by any data segment. Higher address offsets still within the memory space of this CMD file are assigned to stack and free space structures by PL/I-96, with the system stack preceding free space. The SP value required by the CREATE PROC function can be obtained by adding the size of the stack required to the last offset occupied by data. If another MCORTEX process stack is required, its SP can be obtained by adding its size to the SP of the previous process. The system stack can be divided as necessary by continuing in this manner. The total number of bytes occupied by MCORTEX process stacks should not exceed the number of bytes provided by PL/I-96 for the system stack.

45
### TABLE 2:

Map for file: DITFC.CMD

<table>
<thead>
<tr>
<th>Length</th>
<th>Start</th>
<th>Stop</th>
<th>Align</th>
<th>Comb</th>
<th>Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>2CE3</td>
<td>(0000:0005-02E7)</td>
<td>BYTP PUB COTE</td>
<td>COTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2EC4</td>
<td>(0000:2108-06C3)</td>
<td>WOTT PUB DATA</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0213</td>
<td>(0000:0676-06F8)</td>
<td>WORD COM ?PBPSTK</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>(0000:36FA-0727)</td>
<td>WORD COM ?FPE</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0202</td>
<td>(0000:0728-0729)</td>
<td>WORD COM ?CNCL</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0079</td>
<td>(0000:0729-0732)</td>
<td>WORD COM ?FILAT</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0076</td>
<td>(0000:2733-073E)</td>
<td>WORD COM ?FMTS</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0219</td>
<td>(0000:073F-0756)</td>
<td>WORD COM ?EBUFF</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>(0000:0754-075A)</td>
<td>WORD COM ?CNCOD</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>(0000:075C-0760)</td>
<td>WORD COM SYSIN</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>002E</td>
<td>(0000:3722-07A9)</td>
<td>WORD COM SYSPRINT</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Groups

<table>
<thead>
<tr>
<th>Groups</th>
<th>Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODEUP</td>
<td>CODE</td>
</tr>
<tr>
<td>?CNOD</td>
<td>SYSIN</td>
</tr>
</tbody>
</table>

**map for module: INIT**

001E (0000:0005-0022) CODE
0021 (0000:0102-0120) DATA

**map for module: MCDemo**

0072 (0000:0023-0334) CODE
0039 (0000:0122-015A) DATA

**map for module: LOG.ON**

0127 (0000:0095-01FP) CODE
20C7 (0000:215C-2213) DATA

**map for module: GATM/T**

30D0 (0000:018C-228B) CODE
021F (0000:021C-021F) DATA
counts and sequencers, as well as processes. After all initializations are performed, an `AWAIT(\text{FE'84}, \text{'2021'84})` should be executed. This puts all initialization processes on a common reserved event count thread. An `ADVANCE(\text{FE'84})` by any process will return all processors to CP/M-86 control (assuming CP/M-86 is resident locally).

MCORTEX processes are written as parameterless PL/I-ME procedures. Execution of `CREATE PROC` functions in the initialization module establishes a virtual processor for each process, and sets all process states to ready. The `AWAIT` call at the end of initializations forces a scheduling to take place. The highest priority virtual processor will be granted access to the real processor. Further scheduling is controlled by user processes using MCORTEX functions.

Parameters required by the `CREATE PROC` function include values unknown to the programmer until after all processes have been compiled and linked. This requires that dummy values be provided for the first compilation and linking. Links should be performed with the `MAP` command option selected, as this provides information relevant to user process definition. A partial MAP printout for the demonstration process is shown in Table 2.

`CREATE PROC` has eight parameters. The first two are process identification and process priority. These are arbitrary \text{F'TI(8)} values assigned by the programmer. Four other parameters, the CS, DS, SS, and ES register values,
value, and some runtime routines expect this relationship to be maintained. To overcome the consequences of these opposing positions, the gate modules push the ES register onto the stack on entry, and pop it before return to the calling routine. From the standpoint of user processes, the ES register value is unchanged during MCORTEX calls.

D. GENERATING MCORTEX PROCESSES USING PL/I-86

Procedures written in PL/I-86 become MCORTEX processes via execution of CREATE PROC functions. MCORTEX processes, though written, compiled, and linked as PL/I-86 procedures, are distinct processes. Each requires the state of the processor to be prepared by the MCORTEX executive prior to every entry into the process. This is accomplished transparently when making MCORTEX function calls.

Procedures in a MCORTEX process can be accessed from within the process normally, however, a MCORTEX process must be entered through a MCORTEX function call, and never through a PL/I-86 procedure call. Also MCORTEX processes can be linked into a single CMD module or can be developed as separate CMD modules. In the first case processes may share common PL/I-86 runtime routines as well as CP/M-86 utilities. In the second case PL/I-86 runtime routines are not shared, but CP/M-86 utilities, if used, are still shared.

MCORTEX currently expects an initialization module to be located starting at 0439FF. This module is the first user process executed, and can be used to create user event
Fig. 5  PL/M Reentrant Parameter Passing
PL/M-66 reentrant processes expect parameters to be passed on the stack in the order they appear in the procedure declaration. Byte values require two bytes on the stack even though only one byte contains usable information. Parameters are followed immediately on the stack by the call generated return address. The called process stores the callers DS and BP registers on the stack, and establishes its own DS and BP values. Access to parameters is via an index referenced to the called process BP value. Figure 5 is a diagrammatical representation of how a stack is structured following a call to GATEKEEPER.

GATEMOD and GATFTRC both act as translators of user calls into formats required by the MCORTEX and MXTRACE supervisors respectively. The only difference in the two gate modules is the address of GATEKEEPER in their associated KOREs. Using the EX register link to retrieve data, they build the stack structure expected by the supervisor module, supplying function codes and padding when required. They then make a call on GATEKEEPERx. If the call is to READ or TICKET, space is reserved on the stack for the returned value. This value is PCP'd into the EX register before exiting to the calling process.

The gate modules provide one additional service. KORE functions do not guarantee the integrity of the ES register. PL/I-66 in OPTIONS (MAIN) initializations, however, establishes the ES, SS, and DS registers to be of equal
new starting address 439:0.

**Move, starting at address 439:0, 68F bytes of code (GLOBAL memory) to new starting address 439:1E00 (following initialization module).**

**Write to the default disk a file called KORE.TRC starting at address 439:0 and containing 21C0 bytes.**

**NOTE:** The main KORE module, the initialization module, and GLOBAL memory are located to separate parts of the SEC by the MCORTEX loader. The system used requires that these modules be saved into the file in 128 byte blocks. Further, any change in the number of 128 byte blocks occupied by each must be reflected in the MCORTEX loader code.
APPENDIX B

MCORTEX UNDER DDT86

When troubleshooting MCORTEX processes using DDT86, it is important to realize that DDT86 break points are implemented as 8086 commands written at the locations in memory selected as break points. If 'DDT86 MCORTEX' is executed, the MCORTEX system will be loaded under the control of DDT86. If an attempt is made to execute the loader code to a break point inside a user module which is still to be loaded, DDT86 installs the break point command as directed, but this command will be overwritten when the user code is loaded. The code will execute through the intended break point, and the desired result will not be achieved.

To enable break points within user processes, execute 'DDT86 MCORTEX' as before. Now set a break point inside the MCORTEX loader code, but after KORE and the user processes have been loaded. The loader will now input KORE and user modules as directed, and DDT86 will break inside the loader. At this point further break points within KORE and user code can be successfully set, and will not be overwritten.

Trying to use DDT86 on PL/I-66 code can be very confusing as the 8086 code produced is not familiar. Use the MAP function of the LINK86 linker to give yourself
landmark addresses as you traverse the code. The MAP file gives you beginning addresses for each of your procedures and each of the runtime modules provided by PL/I-86. Similar information is found in the MF2 files for KORR code.

When tracing code, use a hierarchical search. Use 20 instructions with break points, or individual trace instructions to execute small sections of code at a time. Break points should be set just past the next call to be executed. When a failure occurs, you will have bracketed the possible code causing the error. If the error is within the call, simply trace into the call one trace step, list the code and proceed in the hierarchical manner used before. Note that you must be mindful of jump instructions in the execution path. You may have to trace several bytes of code to ensure that the execution path includes the break address. This procedure will get you to the errant code with the least amount of tracing.
APPENDIX C

MCOFTEX LOADER

This file when assembled produces the MCOFTEX loader. The loader when invoked from CP/M-86, gives an indication that it is on line, and then asks if GLOBAL memory is to be loaded. The first CPU entering the MCOFTEX environment should load GLOBAL memory, all others should not. The last process loaded on each SBC must contain the initialization routine containing all create process functions. This file contains code that is conditionally assembled to create MIXTRACE. The value of MCOFTEX in the code controls which module is produced, and the name of the file produced must be changed by the user.
This program loads the MCORTEX operating system from disk into the current CP/M environment. The system memory space is reserved using CP/M memory management functions. Since INITIALPROC must be overwritten by the user INITIALPROC, the memory it occupies is not reserved. The portions loaded into the interrupt area and into shared memory (i.e., GLOBALMODULE) are in areas not managed by CP/M and are thus protected from user overwrite when using PLI CMD files. Conditional assemblies allow assembly of either MCORTEX or MXTRACE depending on the value assigned to MCORTEX at the beginning of the code. Nine such conditional assembly statements are included.

**ADDRESS CONSTANTS**

```assembly
DSEG ORG 0000H

* MCORTEX / MXTRACE SELECTION ****************************************/

;*** ADDRESS CONSTANTS ***************************************************

FCB EQU 005CH ;*** FILE CONTROL
FCB_NAME EQU 0065H ;*** BLOCK
FCB_EXTENT EQU 0066H
FCB_CR EQU 007CH

INT_ADD_CS EQU 0011H ;*** INTERRUPT CODE
INTRPT_OFFSET EQU 0033H ;*** SEGMENT AND
IF MCORTEX INTRPT_CS EQU 0C6BH ;*** VECTORS
ELSE INTRPT_CS EQU 0C31H ;### 1 #### <----
ENDIF

;*** PURE NUMBER CONSTANTS *************************************************

EIGHTF_K EQU 0080H
IF MCORTEX EQU 001CH
ELSE NUM_KORE_BLOCKS EQU 0C35H ;### 2 #### <----
ENDIF

ASCII_0 EQU '0'
ASCII_9 EQU '9'
```

58
ASCII_A ECU 'A'
ASCII_Z ECU 'Z'
COLON ECU ':'
SPACE ECU ' '.
PERIOD ECU '.'
CR EQU $00DH
LF EQU $00AH

;*** CONTROL TRANSFER CONSTANTS *****************************

IF MCCRTEX
  KORE_SP EQU $08CH
KORE_SS_VAL EQU $0C7EH
KORE_DS_VAL EQU $0C69H
ELSE
  KORE_SP EQU $0F8F ;### 3 ### <-----
KORE_SS_VAL EQU $0C30H ;### 4 ### <-----
KORE_DS_VAL EQU $0C00H ;### 5 ### <-----
ENDIF

;*** CP/ Memory FUNCTION CONSTANTS *****************************

CP/%_DOS_CALL EQU 224
SYSTEM_FSFST EQU $007CH
CONSLF_OUTPUT EQU $0022H
READ EQU $008AH
PRINT_STRING EQU $0029H
OPEN_FILE EQU $000FH
READ_SEQUENTIAL EQU $014EH
SET_DMA_OFFSPT EQU $01AH
SET_DMA_BASE EQU $033CH
ALLOC_MEM_ABS EQU $032EH
FREE_ALL_MEM EQU $033AH
PROGRAM_LOADED EQU $033BH
NOT_FOUND EQU $02FFH

;*** MESSAGES ***********************************************

IN_STRING DB 15
FB 16

NO_FILE_MSG DB 'KORE NOT ON DEFAULT DRIVES'
NO_IN_FILE_MSG DB 'INPUT FILE NOT ON DESIGNATED DRIVES'
NO_MEMORY_MSG DB 'UNABLE TO ALLOCATE MEMORY SPACE FOR'
       'MCCRTEX'
FILE_FORM_ERR_MSG DB 'INCORRECT FILE FORMAT - TRY AGAIN'
START_MSG DB 'MCCRTEX SYSTEM LOADER *** ON LINES'
P_NAME_MSG DB CRLF,LF,'ENTER PROCESSOR FILE NAME:',CRLF
       DE '$'
GLOBAL_Q_MSG DB CR,LF,LF,"LOAD GLOBAL MEMORY?",CR,LF
GLOBAL2_MSG DB "Y" TO LOAD, "RETURN" TO SKIP",CR,LF,"

;*** MCORTEX RELOCATION VARIABLES ****************************

;*** CAUTION *** CAUTION *** CAUTION *** CAUTION *** CAUTION ****************************
;*** The following five lines of code should not be ***
;*** separated as this program assumes they will be ***
;*** found in the order shown. The code is used for ***
;*** memory allocation and as a pointer to KORE. ***
;*** CAUTION *** CAUTION *** CAUTION *** CAUTION ****************************

KORE2_START DW $C30H ;*** CAUTION
IF MCORTEX
KORE2_BASE DW $B0H ;*** CAUTION
ELSE
KORE2_BASE DW $AC0H ;### 6 ### <------
ENDIF
IF MCORTEX
EQU DWORD PTR KORE2_START ;*** CAUTION
ELSE
ENDIF
IF MCORTEX
KORE2_LENGTH DW $C30H ;*** CAUTION
ELSE
KORE2_LENGTH DW $1C3H ;### 7 ### <------
ENDIF
KORE2_EXT DB 0 ;*** CAUTION
IF MCORTEX
KORE_NAME DB 'KORE OPS'
ELSEF
KORE_NAME DB 'KORE TEC' ;### 8 ### <---
ENDIF

KORE2_BASE DW $E794H ;*** GLOBAL MEMORY
INTERRUPTVECTOR DW INTRPT_OFFSET,INTRPT_CS
INTVECTOR_ADD DW INT_ADD_CS
INIT_OFFSET DW $0000H ;*** INITIALIZATION
INIT_BASE DW $439H ;*** ROUTINE PARAMETERS
IF MCORTEX
INIT_DS_SEG DW $C98H ;*** FOR DYNAMIC ASSIGNMENT
ELSEF
INIT_DS_SEG DW $C58H ;### 9 ### <------
ENDIF
INIT_DS_OFFSET DW $C68H ;*** WHEN USER INITIALIZATION
IF INIT_IP_OFFSET DW $074H ;*** IS INDICATED
ENDIF

;*** CONTROL TRANSFER VARIABLES ****************************

KORE_SS DW KORE_SS_VAL
KORE_DS DW KORE_DS_VAL
;*** START CODE SEGMENT *******************************************************/

;**CORTEX_LOADPR CSEG

CALL CLR_SCREEN ;*** SCREEN CONTROL & LOC ON
CALL MCORTX_LOAD ;*** MESSAGES
CALL CLR_SCREEN ;***

CLD ;*** INITIALIZATION
PUS AX ;***

;*** GET LOAD GLOBAL INDICATOR ***********************************************/

CALL IN_GLOBAL ;*** ASK IF GLOBAL TO BE LOADED
MOV DX,OFFSET IN_STRING ;*** GET BUFFER LOCATION
MOV CL,READ ;*** CP/M PARAMETER
INT CP/M_BDOS_CALL ;*** GET INDICATE

;*** GENERATE KORE FILE CONTROL BLOCK ****************************************

GEN_KORF_FCB:

MOV BX,10 ;*** MOVE 11 CHARACTERS
MOV SI,OFFSET KORE_NAME ;*** POINT TO KORE NAME
MOV DI,FCF_NAME ;*** POINT TO FCF NAME
MOV KORE: -

MOV AL,[SI+BX] ;*** GET CHARACTER
MOV [DI+BX],AL ;*** STOPE CHARACTER
DFF BX
JGE MOV KORE:

;*** OPEN KORE.OPS FILE ON DEFAULT DISK ****************************************

OPEN_KORE:

MOV CL,OPEN_FILE ;*** CP/M PARAMETER
MOV DX,FCB ;*** CP/M PARAMETER
INT CP/M_BDOS_CALL ;*** OPEN FILE
CMP AL,NOT_FOUND ;*** FILE FOUND?
JNE PROCESS_KORE ;*** FILE FOUND! CONTINUE
JMP NO_FILE ;*** GO INDICATE ERROR
PROCESS_KORE:

MOV DI,0
MOV FCF_CK,[DI].DI ;*** START WITH REC ZERO

;*** RESERVE MEMORY ***********************************************************/

MOV CL,FREE_ALL_MEM ;*** CP/M PARAMETER
INT CP/M_BDOS_CALL ;*** FREE ALL MEMORY
MOV CL,ALLOC_MEM_ABS ;*** CP/M PARAMETER
MOV DX,OFFSET KORE1_BASE ;*** CP/M PARAMETER
INT CP/M_BDOS_CALL ;*** FREE ALLOCATE MEMORY
CMP AL,NOT_FOUND ;*** MEMORY AVAILABLE?
JNE LOAD_MCORTPX ;*** MEMORY AVAILABLE! CONTINUE
LOAD_MCORTEX:

MOV DI,0
MOV BP,NUM_KORE_BLOCKS

MOVE_KORE_LOOP:

MOV DX,PCR
MOV CL,READ_SEQUENTIAL
INT CPM_READ_SUCENTIAL
MOV ES,EPK1_BASE
MOV CX,EIGHTH_K

FEP MOVSE
DEC BP
JNZ MOVE_KORE_LOOP

;*** LOAD INITIALIZATION MODULE ***************************************************************************/

MOV DI,INIT_OFFSET
MOV DX,PCR
MOV CL,READ_SEQUENTIAL
INT CPM_READ_SUCENTIAL
MOV ES,INIT_BASE
MOV CX,EIGHTH_K

REP MOVSB

;*** LOAD GLOBAL MEMORY ***************************************************************************/

CMP IN_STRING+1,0H
JZ INSTALL_INTERRUPT
MOV DI,0

MOVE_GLOBAL_LOOP:

MOV CX,PCR
MOV CL,READ_SEQUENTIAL
INT CPM_READ_SUCENTIAL
TEST AL,AL
JNZ INSTALL_INTERRUPT
MOV ES,KORE2_BASE

REP MOVSB
JMP MOVE_GLOBAL_LOOP

;*** INITIALIZE INTERRUPT VECTOR ***************************************************************************/

INSTALL_INTERRUPT:

MOV ES,INT_VECTOR_ADD
MOV DI,0
MOV DIR,OFFSET INTERRUPT_VECTOR

;*** GO INDICATE ERROR ***************************************************************************/

JMP NO_MEMORY_ALLOC
MOV CX,2
PUSH MOV AX,AX

;*** READ IN A FILE NAME ************************************

READ A NAME:
CALL PROCESSOR NAME
MOV DX,OFF SET IN_STRING
MOV CL,READ
INT CMP_EDCS_CALL

;*** SET FCB DRIVE DESIGNATION *******************************

CMP IN_STRING+1,0
JF EXIT_ROUTINE_F
POP AX

MOV DI,0

CMP IN_STRING+3,Colon
JF SET_DRIVE
POP AX
MOV FCB[DI],DI
MOV SI,2
JS FORM_FCB

SET DRIVE:
MOV AL,IN_STRING+2
AND AL,5FH
SUP AL,40H
MOV FCB[DI].AL
MOV SI,2
JS FORM_FCB

FORM_FCB:
MOV EX,2AH
MOV AL,SPACE
FILL SPACES:
MOV FCB_NAME[EX].AL
DEC BX
JGE FILL SPACES

MOV FCB_CR[DI].DI
MOV FCB_EXTENT[DI].DI

NAME LOOP:
MOV AL,IN_STRING[SI]
**OPEN THE PROCESSOR FILE**

**OPEN_PROCESSOR:**

```assembly
MOV DX, FCB  ;*** CP/M PARAMETER
MOV CL, OPEN_FILE  ;*** CP/M PARAMETER
INT CPM_BOS_CALL  ;*** OPEN THE FILE
CMP AL, NOT_FOUND  ;*** WAS FILE ON DISK
JMP LOAD_PROCESSOR  ;*** IF YES, GO LOAD THE FILE
JMP NC_INPUT_FILE  ;*** IF NO, SIGNAL ERROR
```

**LOAD_PROCESSOR:**

```assembly
MOV DX, FCB  ;*** CP/M PARAMETER
MOV CL, PROGRAM_LOAD  ;*** CP/M PARAMETER
INT CPM_BOS_CALL  ;*** LOAD THE FILE
PUSH AX  ;*** SAVE DATA SEGMENT
JMP READ_A_NAME  ;*** GET NEXT PROCESSOR
```

**SET UP THE INITIALIZATION STACK**

```assembly
```
;*** CAUTION *** CAUTION *** CAUTION *** CAUTION *** CAUTION -------------------------------/

EXIT_ROUTINE:
PO AX
MOV ES,INIT_DS_SEG ;*** RECOVER DATA SEGMENT
MOV BX,INIT_DS_OFFSET ;*** POINT TO INIT STACK
MOV ES:[3X],AX ;*** INSTALL NEW INIT DS
MC DX,0 ;*** SET NEW IP VALUE
MOV EX,INIT_IP_OFFSET ;*** POINT TO IP ON STACK
MOV ES:[3X],DX ;*** INSTALL NEW INIT IP
MOV CL,SET_DMA_BASE ;*** CP/M PARAMETER
MOV DX,AX ;*** SET DMA BASE
INT CPM_PDOS_CALL ;*** SET DMA OFFSET
INT CP_PDOS_CALL

;*** TRANSFER CONTROL TO MCCORTEX *******************************

MOV SP,KORE_SP ;*** KORE STACK POINTER
MOV BP,SP ;*** KORE STACK BASE
MOV SS,KORE_SS ;*** KOLE STACK SEGMENT
MOV AX,DS ;*** GET DATA SEGMENT
MOV ES,AX ;*** POINT DS TO DS
MOV DS,KORE_DS ;*** KORE DATA SEGMENT
JMPF ES,KORE ;*** JUMP TO MCCORTEX

;*** VALID CHARACTER FOR FILE NAME CHECK ********************

VALID_INPUT:
CMP AL,ASCII_0 ;*** IS THE CHARACTER A NUMBER
JB KORE_VALID ;***
CMP AL,ASCII_9 ;***
JBE IS_VALID ;***
AND AL,5FH ;*** CONVERT CHARACTER TO UPPER CASE
CMP AL,ASCII_A ;*** IS THE CHARACTER A LETTER
JB NOT_VALID ;***
CMP AL,ASCII_Z ;***
JBE IS_VALID ;***
NOT_VALID:
MOV AX,0 ;*** INDICATE BAD CHARACTER
IS_VALID:
RET ;*** CHARACTER OK

;*** ABORT MESSAGES ******************************************

NO_FILE:
CALL CLR_SCREEN
MOV DX,OFFSET NO_FILE_MSG ;*** PTR TO MSG
JMP MSG_OUTPUT ;*** PUT MSG

NO_MEMORY_ALLOC:
CALL CLR_SCREEN
MOV DX,OFFSET QC_MEMORY_MSG ;*** PTR TO MSG

MSG_OUTPUT:
MOV CL,PRINT_STRING ;*** CP/M PARAMETER
INT CP/M EDOS CALL ;*** SEND CHAR TO CONSOLE
CALL CLR_SCREEN
MOV CL,SYSTEM_RESET ;*** CP/M PARAMETER
MOV DL,? ;*** RELASE MEMORY
INT CP/M_EDOS_CALL ;*** EXIT TC CP/M

;*** SCREEN CONTROL *******************************************/

CLR_SCREEN:
MOV CL,CONSOLE_OUTPUT ;*** ISSUE CARRIAGE RETURN
MOV DL,CR ;***
INT CP/M_EDOS_CALL ;***
MOV DI,CHR _ ;*** ISSUE 12 LINE FEEDS
LINE_FEED:
MOV DL,LF ;***
MOV CL,CONSOLE_OUTPUT ;***
INT CP/M_EDOS_CALL ;***
DEC DI ;***
JNF LINE_FEED ;***
RET

SEND_MSG:
MOV CL,PRINT_STRING ;*** CP/M PARAMETER
INT CP/M_EDOS_CALL ;*** PRINT A STRING TO CONSOLE
RET

;*** NON ABORT MESSAGES *******************************************/

MCORTEX_LOAD:
MOV DX,OFFSET START_MSG
CALL SEND_MSG
RET

PROCESS03_NAME:
MOV DX,OFFSET P_NAME_MSG
CALL SEND_MSG
RET

GLOBAL:
MOV DX,OFFSET GLOBAL_Q_MSG
CALL SEND_MSG
RET

INPUT_ERROR:
CALL CLR_SCREEN
MOV DX,OFFSET FILE_FORM_ERR_MSG
JMP EXIT_ERR
NO_INPUT_FILE:
CALL CLR_SCREEN
MOV DX,OFFSET NO_IN_FILE_MSG
EXIT_ERR:
CALL SEND_MSG
CALL CLR_SCREEN
JMP READ_A_NAME

END
**CORTEX input option file**

**CORTEX = TEX/TRC [code[ab[547]],data[ab[15C]]]**

**D1 input option file**

D1 = DINIT [code[ab[547]], data[ab[434],m[0],ad[52]], map[all]], McEMC, LOGON, \*

**D2 input option file**

D2 = DINIT [code[ab[547]], data[ab[434],m[0],ad[52]], map[all]], McEMC, LOGON, DELAYER, \*

**MTRACE input option file**

**MTRACE = TEX/TRC [code[ab[547]], data[ab[15C]]]**

**DTRACE input option file**

DTRACE = DINIT [code[ab[547]], data[ab[434],m[0],ad[52]], map[all]], McEMC, LOGON, \*
APPENDIX F

LINK66 INPUT OPTION FILES

This group of files allows linkage of specified object code modules using the LINK66 input abbreviation. As an example, after compilation of DINIT.PLI, MCODEM.PLI, and LOC0N.PLI, and assembly of GATEMOD, the demonstration program D1 is created invoking "LINK66 D1[1]". For further information on input option files, see [Ref. 13].
CALL advance (log_in);
END

*******************************************************************************************************
*******************************************************************************************************
****
**** DELAYE.PLI code
****
**** This code provides a time delay to demonstration
**** programs D1 and D2, under the control of D2.
****
*******************************************************************************************************

delay

PROCEDURE;

YINCLUDE 'gateway.pli';

DECLARE
max_count FIXED STATIC INITIAL ('10000'),
iterations FIXED STATIC INITIAL ('10'),
'k','i','j') FIXED,

start BIT(16) STATIC INITIAL ('00000000'),
um_processors BIT(16) STATIC INITIAL ('00000000'),
delay BIT(8) STATIC INITIAL ('00000000'),
sync BIT(8) STATIC INITIAL ('00000000'),

DO k = 1 to max_count;
   DO i = 1 to iterations;
      DO j = 1 to max_count;
         END
   END
END; /* DO */
CALL advance (delay);
start = add2bit16 (start, num_processors);
PUT EDIT ('sync await is ', start) (skip, A(17), E4(4));
CALL await (sync, start);
END; /* DO */
EXIT delay;
This code allows the operator to start all real processors executing in MCDEMO at the same time regardless of the order that they came on line. This is a demonstration only and is not required under MCOREX.

```
log_on:
PROCEDURE;

#include 'gateway.pli';

DECLARE
no_signal CHAR VARYING,
num_stc_less_1 BIT(16) STATIC INITIAL ('$001B4'),
one BIT(16) STATIC INITIAL ('$001B4'),
turn BIT(16) STATIC INITIAL ('$003B4'),
log_in BIT(8) STATIC INITIAL ('$1B4');

DECLARE
msg1 CHARACTER(39) STATIC INITIAL
  ("MCOREX Demonstration Program "ON LINE".
msg2 CHARACTER(30) STATIC INITIAL
  ("Press M". "TURN to Continue".
msg3 CHARACTER(14) STATIC INITIAL
  ("Turn Value is ");

put EDIT (msg1) (skip(12), '21'), A(39));
put EDIT (""") (skip(13), A(7));
call create_evc (log_in);
call create_seo (log_in);
turn = ticket (log_in);
put EDIT (msg2, turn) (A(14), 'B4(4));
if turn = num_stc_less_1 then
  do:
    put EDIT (msg2) (skip, x('25'), A(30));
    ?l LIST (no_signal);
    ind; /* do */
  else
    do:
      turn = add2bit16 (turn, one);
      put EDIT ("ENTRY: await(log_in, turn) = await(", log_in,
      ", ", turn, ")", 'B4(4), A(1));
      call await (log_in, turn);
```

79
CALL await ("fe", "01", "P4");
/* CALL await (EVC, COUNT); */
END;
END;

******************************************************************************
******************************************************************************

***
*** This code is the main controlling code for the demonstration programs D1 and D2. It is compiled separately and linked using the D1 and D2 input option files.
***
***
******************************************************************************
******************************************************************************

**modemo:**

PROCEDURE;

%INCLUDE 'gateway.pli';

DECLARE
   log_on ENTRY;

DECLARE
   delay_value BIT(16) STATIC INITIAL ("000", "P4"),
   one BIT(16) STATIC INITIAL ("000", "P4"),
   enough BIT(16) STATIC INITIAL ("000", "P4"),
   delay BIT(8) STATIC INITIAL ("02", "P4"),
   sync BIT(8) STATIC INITIAL ("03", "P4"),
   exit BIT(8) STATIC INITIAL ("ff", "P4");

DECLARE
   ms2 CHARACTER(21) STATIC INITIAL
       ("Delay Event Count is ");

CALL log_on;
CALL create_evc (delay);
CALL create_evc (sync);
WHILE (delay_value < enough);
   PUT EDIT (ms2, delay_value) (SKIP(5), A(21), P4:4));
   CALL advance (sync);
   CALL await (delay, delay_value);
   delay_value = read (delay);
   delay_value = add2bit16 (delay_value, one);
END; /* DO WHILE */
CALL preempt (exit);
END modemo;
**DINIT.PLI code**

*** This code creates the D1 process for execution under MCORTEX. Using the MCORTEX loader, the last process to be loaded must contain the initialization process.

```pli
init:
PROCEDURE OPTIONS('MAIN) RETURNS();
%INCLUDE 'gateway.pli';
BEGIN;
  CALL create_proc ('01'B4, 'fd'B4, '929'B4, '0713'B4, '023'B4, '439'B4, '070a'B4, '072a'B4);
  /*CALL create_proc (PROCESS_ID, PROCESS_PRIORITY, */
  /* SP SS IP */
  /* CS DS ES */
  CALL await ('fe'B4, '01'B4);
  /*CALL await ( EVC, COUNT); */
END;
```

**DINIT2.PLI code**

*** This code creates the D2 process and the delayer process for execution under MCORTEX. Using the MCORTEX loader, the last process loaded must contain the initialization process.

```pli
init:
PROCEDURE OPTIONS('MAIN) RETURNS();
%INCLUDE 'gateway.pli';
BEGIN;
  CALL create_proc ('01'B4, 'ed'B4, '929'B4, '0713'B4, '029'B4, '439'B4, '0713'B4, '0713'B4);
  /*CALL create_proc (PROCESS_ID, PROCESS_PRIORITY, */
  /* SP SS IP */
  /* CS DS ES */
  CALL create_proc ('02'B4, 'fd'B4, '0949'B4, '0713'B4, '01c2'B4, '439'B4, '0713'B4, '0713'B4);
```
APPENDIX E

DEMONSTRATION PROGRAM

The files presented here are a series of procedures that can be compiled separately and linked in accordance with LINKRG input option files in APPENDIX F. The results will be demonstration processes D1 and D2, or D1TRC and D2TRC depending on the option files selected.
CALLF GATEKEEPER
POP ES
RPT

;*** ADD2BIT16 *** ADD2BIT16 *** ADD2BIT16 *** ADD2BIT16 ***

ADD2BIT16:
MOV SI,[BX] ;SI <-- PTR TO BIT(16)#1
MOV BX,2[BX] ;BX <-- PTR TO BIT(16)#2
MOV BX,[BX] ;BX <-- BIT(16)#2
ADD BX,[SI] ;BX <-- BIT(16)#1 + BIT(16)#2
RET

END
PUSH SS ;PTR_SEG <-- EVENT COUNT SEGMENT
PUSH CX ;PTR_OFFSET <-- EVENT COUNT POINTER
CALL GET_EVT_CNT
POP PX
POP ES
RET

;*** CREATE_PROC *** CREATE_PROC *** CREATE_PROC **********/

CREATE_PROC:
PUSH ES
MOV SI,14[BX] ;SI <-- PTR TO PROCESS ES
PUSH WORD PTR [SI] ;STACK PROCESS ES
MOV SI,12[BX] ;SI <-- PTR TO PROCESS DS
PUSH WORD PTR [SI] ;STACK PROCESS DS
MOV SI,1[BX] ;SI <-- PTR TO PROCESS CS
PUSH WORD PTR [SI] ;STACK PROCESS CS
MOV SI,8[BX] ;SI <-- PTR TO PROCESS IP
PUSH WORD PTR [SI] ;STACK PROCESS IP
MOV SI,6[BX] ;SI <-- PTR TO PROCESS SS
PUSH WORD PTR [SI] ;STACK PROCESS SS
MOV SI,4[BX] ;SI <-- PTR TO PROCESS SP
PUSH WORD PTR [SI] ;STACK PROCESS SP
MOV SI,2[BX] ;SI <-- PTR TO PROCESS PRIORITY
MOV AH,[SI] ;GET PROCESS PRIORITY
MOV SI,[BX] ;SI <-- PTR TO PROCESS ID
MOV AL,[SI] ;GET PROCESS ID
PUSH AX ;STACK PROCESS PRIORITY AND ID
MOV CX,SP
MUL CX,CREATE_PROC_IND
PUSH AX ;N <-- CREATE PROCESS IND
PUSH AX ;BYTE <-- UNUSED WORD
PUSH AX ;WORDS <-- UNUSED WORD
PUSH SS ;PROC_PTR SEGMENT <-- STACK SEG
PUSH CX ;PROC_PTR OFFSET <-- DATA POINTER
CALL GET_EVT_CNT
ADD SP,14 ;REMOVE STACKED DATA
POP ES
RET

;*** PREEMPT *** PREEMPT *** PREEMPT *** PREEMPT *** PREEMPT **********/

PREEMPT:
PUSH ES
MOV BX,[BX] ;BX <-- PTR TO NAME OF PROCESS
MOV AL,PREEMPT_IND
PUSH AX ;N <-- PREEMPT INDICATEER
MOV 42,[BX]
PUSH AX ;BYTE <-- PREEMPT PROCESS NAME
PUSH AX ;WORDS <-- UNUSED WORD
PUSH AX ;PTR_SEG <-- UNUSED WORD
PUSH AX ;PTR_OFFSET <-- UNUSED WORD

72
PUSH AX
CALLF GATEKEEPER
POP ES
RET

*** CREATE_SEQ *** CREATE_SEQ *** CREATE_SEQ ***************/

CREATE_SEQ:
PUSH ES
MOV PX,[BX]
MOV AL,CREATE_SEQ_IND
PUSH AX
MOV AL,[BX]
PUSH AX
MOV AL,CREATE_SEQ_IND
PUSH AX
CALLF GATEKEEPER
POP ES
RET

*** TICKET *** TICKET *** TICKET *** TICKET *** TICKET *** TICKET ****/

TICKET:
PUSH ES
PUSH ES
MOV CX,SP
MOV BX,[BX]
MOV AL,TICKET_IND
PUSH AX
MOV AL,[BX]
PUSH AX
MOV SS
PUSH CX
CALLF GATEKEEPER
POP BX
POP ES
RET

*** READ *** READ *** READ *** READ *** READ *** READ *** READ ****/

READ:
PUSH ES
PUSH ES
MOV CX,SP
MOV BX,[BX]
MOV AL,READ_IND
PUSH AX
MOV AL,[BX]
PUSH AX
RET

;PTR_OFFSET <--UNUSED WORD

;BX <-- PTR TO NAME OF SEQ

;IN <-- CREATE_SEQ INDICATOR

;BYTES <-- NAME OF SEQ

;WORDS <-- UNUSED WORD

;PTR_SEG <-- UNUSED WORD

;PTR_OFFSET <--UNUSED WORD

;CPL.LF

;POP

;STI

;Cv-"T

;ICYET

;TICKET

;PUSH

;FS

;TICKET

;NUM Fl

;DUMMY STORAGE

;POINTER TO TICKET NUMBER

;BX <-- PTR TO TICKET NAME

;IN <-- TICKET INDICATOR

;BYTES <-- TICKET NAME

;WORDS <-- UNUSED WORD

;PTR_SEG <-- TICKET NUMBER SPG

;PTR_OFFSET <-- TICKET NUMBER POINTER

;RETIEVF TICKET NUMBER

;EVENT COUNT DUMMY STORAGE

;POINTER TO EVENT COUNT

;BX <-- PTR TO EVENT NAME

;IN <-- READ INDICATOR

;BYTES <-- EVENT NAME

;BYTES <-- UNUSED WORD
GATEKEEPER_IP DW 0062H ;#### 1 #### <=----------------
GATEKEEPER_CS DW 0B4AH ;#### 2 #### <=----------------

ENDIF
GATEKEEPER: EQU DWORD PTR GATEKEEPER_IP

CSEG

;*** AWAIT *** AWAIT *** AWAIT *** AWAIT *** AWAIT ***********/

AWAIT:
PUSH ES
MOV SI,2[BX] ;SI <-- PTR TO COUNT AWAITED
MOV BX,[BX] ;BX <-- PTR TO NAME OF EVENT
MOV AL,AWAIT_IND
PUSH AX ;IN <-- AWAIT INDICATOR
MOV AL,[BX]
PUSH AX ;BYT <-- NAME OF EVENT
MOV AX,[SI] ;AX <-- COUNT AWAITED
PUSH AX ;WORDS <-- COUNT AWAITED
PUSH AX ;PTR_SEG <-- UNUSED WORD
PUSH AX ;PTR_OFFSET <-- UNUSED WORD
CALLF GATEKEEPER
POP ES
RET

;*** ADVANCE *** ADVANCE *** ADVANCE *** ADVANCE ***********/

ADVANCE:
PUSH FS
MOV BX,[BX] ;BX <-- PTR TO NAME OF EVENT
MOV AL,ADVANCE_IND
PUSH AX ;IN <-- ADVANCE INDICATOR
MOV AL,[BX]
PUSH AX ;BYTE <-- NAME OF EVENT
PUSH AX ;WORDS <-- UNUSED WORD
PUSH AX ;PTR_SEG <-- UNUSED WORD
PUSH AX ;PTR_OFFSET <-- UNUSED WORD
CALLF GATEKEEPER
POP FS
RET

;*** CREATE_EVC *** CREATE_EVC *** CREATE_EVC *************/

CREATE_EVC:
PUSH FS
MOV BX,[BX] ;BX <-- PTR TO NAME OF EVENT
MOV AL,CREATE_EVC_IND
PUSH AX ;IN <-- CREATE_EVC INDICATOR
MOV AL,[BX]
PUSH AX ;BYTE <-- NAME OF EVENT
PUSH AX ;WORDS <-- UNUSED WORD
PUSH AX ;PTR_SEG <-- UNUSED WORD
This module is given to the user in object form to link with his initial and process modules. Any changes to user services available from the OS must be reflected here. In this way the user need not be concerned with actual GATEKEEPER services codes. Two lines of code are contained in conditional assembly statements and control the output to be GATEMOD or GATETRC depending on the value of GATEMOD at the code start.

This module reconciles parameter passing anomalies between MCO:TEX (written in PL/M) and user programs (written in PL/I).

All calls are made to the GATEKEEPER in LEVEL2 of the OS. The address of the GATEKEEPER must be given below.

The ADD2BIT16 function does not make calls to MCO:TEX. Its purpose is to allow the addition of two unsigned 16 bit numbers from PL/I programs.

***********************************************************************************************************************/
GATETRC / GATEMOD File GATEM/T.a9E Row 12 Feb F4 */

;***********************************************************************************************************************/
GA TMO / GATETRC File GATEM/T.a9E Row 12 Feb F4 */

;***********************************************************************************************************************/
TSEG

GATEMOD EQU 2 ;*** SET TO ZERO FOR GATETRC ;*** SET TO ONE FOR GATEMOD
PUBLIC ADVANCE ;*** THESE DECLARATIONS MAKE THE
PUBLIC AWAIT ;*** GATEKEEPER FUNCTIONS VISIBLE
PUBLIC CREATE_FVC ;*** TO EXTERNAL PROCESSES
PUBLIC CREATE_PROC
PUBLIC CREATE_SEQ
PUBLIC PREEMPT
PUBLIC READ
PUBLIC TICKET
PUBLIC ADD2BIT16

AWAIT_IND EQU 0 ;*** THESE ARE THE IDENTIFICATION
ADVANCE_IND EQU 1 ;*** CODES RECOGNIZED BY THE
CREATE_FVC_IND EQU 2 ;*** GATEKEEPER IN LEVEL II OF
CREATE_SEQ_IND EQU 3 ;*** MCORTEX
TICKET_IND EQU 4
READ_IND EQU 5
CREATE_PROC_IND EQU 6
PREEMPT_IND EQU 7

IF GATEMOD
GATEKEEPER_IP DW 002AH
GATEKEEPER_CS DW 0BBH
ELSE

72
This section of code is given as a PLI file to be \*\* INCLUDE'ed with MCORTEX user programs. ENTRY \*\*
** declarations are made for all available MCORTEX \*\*
** functions and for ADD2BIT16, a utility function \*\*
** allowing unsigned addition of 16 bit numbers. \*\*

DECLARE

advance ENTRY (BIT (8)),
/* advance (event_count_id) */

await ENTRY (PIT (8), BIT (16)),
/* await (event_count_id, awaited_value) */

cREATE entry (BIT (8)),
/* create_evc (event_count_id) */

create_proc ENTRY (BIT (8), BIT (8),
BIT (16), BIT (16), BIT (16), BIT (16),
/* create_proc (processor_id, processor_priority, */
/* stack_pointer_highest, stack_segment, ip */
/* code_segment, data_segment, extra_segment) */

cREATE SEQ ENTRY (BIT (8)),
/* CREATE SEQ (sequence_id) */

preempt FNTPY (BIT (8)),
/* preempt (processor_id) */

read ENTRY (BIT (8)) RETURNS (BIT (16)),
/* read (event_count_id) */
/* RETURNS current_event_count */

ticket ENTRY (BIT (8)) RETURNS (BIT (16)),
/* ticket (sequence_id) */
/* RETURNS unique_ticket_value */

ADD2BIT16 ENTRY (BIT(16), BIT(16)) RETURNS (BIT (16));
/* ADD2BIT16 (a_16bit_, another_16bit_) */
/* RETURNS a_16bit_ + another_16bit_ */
APPENDIX D

GATE MODULE CODE

Two files are contained here. The first is PL/I code, GATEWAY, which must be %INCLUDE'd with every user process requiring access to MCORTFX. The second is Assembler code which provides an interface between the GATEWAY and the MCORTFX supervisor. The object code obtained from assembly of this file must be linked with all user processes to provide "gateway" access to MCORTFX functions. Two lines of code are conditionally assembled to produce either GATEMCD or GATEFACE. The conditional variable is called GATEMCD.
D2TRC input option file

D2TRC = DINIT2 [code[ab[547]], data[ab[439], m[0], ad[82]], map[all]], MCTEMO, LOGON, DELAYER, TATETAC
APPENDIX G

LEVEL II -- MCOPTEX SOURCE CODE

All the LEVEL II source code written in PL/M is contained in the file LEVEL2.SRC. It is compiled with the L3FE attribute. LEVEL II is one of the relocateable code modules in file: KORE.LNK. It is part of the executable code module in file: KORE. KOR-E is the development system version of the file KORE.OPS loaded by MCOPTEX.CMD under the CP/M-86 operating system. Two memory maps (KORE.OPS and KORE.TRC) located in Appendix H give information on this module. The maps come from file: KORE.MPZ after compiling, linking and locating the applicable files. KORE(OPS) is produced with the code unaltered. KORE(TRC) is obtained by removing and adding appropriate comment marks from the indicated code before processing.
/* FILE: LEVEL2.SRC
   VERSION: PROW 6-22-84

PROCEDURES DEFINED: GATEKEEPER CREATEFVC
                   READ AVOID
                   ADVANCE PREEMPT
                   TICKET CREATEPROC
                   OUT$CHAR OUT$LINE
                   OUT$NUM OUT$NUM
                   SEND$CHAR OUT$HEX
                   REC$CHAR IN$CHAR
                   IN$NUM IN$NUM
                   IN$HEX

REMARKS: !!! CAUTION !!! !!! CAUTION !!! !!! CAUTION !!!!!!
IF NEW USER SERVICES ARE ADDED TO THIS MODULE OR CHANGES ARE MADE TO EXISTING ONES, MAKE SURE THE LOCATOR MAP (FILE: KORP.MAP) IS CHECKED TO SEE IF THE LOCATION OF 'GATEKEEPER' HAS NOT CHANGED. THE ABSOLUTE ADDRESS OF THIS PROCEDURE MUST BE SUPPLIED TO THE GATES MODULE IN FILE: GATE.SRC. IF IT HAS CHANGED THE NEW ADDRESS SHOULD BE UPDATED IN FILE: GATE.SRC AND RECOMPILED. ALL USER PROCESSES WILL HAVE TO BE RELINKED WITH FILE: GATE.OBJ AND FLOATED.

LITERAL DECLARATIONS GIVEN AT THE BEGINNING OF SEVERAL MODULES ARE LOCAL TO THE ENTIRE MODULE. HOWEVER, SOME ARE LISTED THE SAME IN MORE THAN ONE MODULE. THE VALUE AND THEREFORE THE MEANING OF THE LITERAL IS COMMUNICATED ACROSS MODULE BOUNDARIES. "NOTFOUND" USED IN LOCATEFVC AND CREATEFVC IS AN EXAMPLE. TO CHANGE IT IN ONE MODULE AND NOT THE OTHER WOULD KILL THE CREATION OF ANY NEW EVENTCOUNTS BY THE OS.

*/

/* LOCAL DECLARATIONS */

L2$MODULE: DO;

/* LOCAL DECLARATIONS */

84
DECLARE
MAX$CPU LITERALLY '10',
MAX$VP$CPU LITERALLY '10',
MAX$CPU$MAX$VP$CPU LITERALLY '100',
FALSE LITERALLY '0',
HEAD LITERALLY '1',
RUNNING LITERALLY '3',
WAITING LITERALLY '7',
TRUE LITERALLY '119',
NOT$FOUND LITERALLY '255',
PORT$CA LITERALLY '00CA',
RESET LITERALLY '0',
INT$RETURN LITERALLY '7F',

/* PROGRAM DATA SEGMENT TABLE */
/* DECLARED PUBLIC IN MODULE 'L1$MODULE' */
/* IN FILE 'LEVEL1' */

DECLARE P3S STRUCTURE
(CPU$NUMBER BYTE,
VP$START BYTE,
VPS$END BYTE,
VPS$CPU BYTE,
LAST$RUN BYTE,
COUNTER WORD)
EXTERNAL;

/* GLOBAL DATA BASE DECLARATIONS */
/* DECLARED PUBLIC IN FILE 'GLOBAL.SRC' */
/* IN MODULE 'GLOBAL$MODULE' */

DECLARE VM$( MAX$CPU$MAX$VP$CPU ) STRUCTURE
(VPS$ID BYTE,
STATE BYTE,
VP$PRIORITY BYTE,
EVCS$THRO'A BYTE,
EVCS$AW$VALUE WORD,
SP$REG WORD,
SS$REG WORD)
EXTERNAL;

DECLARE EVENTS BYTE EXTERNAL;

DECLARE EVCS$TR (100) STRUCTURE
( EVCS$NAME BYTE,
VALUE WORD,
THRO'A BYTE)
EXTERNAL;

DECLARE
SEQUTNCERS
BYTE
EXTERNAL;

DECLARE SEQ$TABLE (100) STRUCTURE
(SEQ$NAME
BYTE;
SEQ$VALUE
WORD)
EXTERNAL;

DECLARE
NR$VPS (MAX$CPU ) BYTE EXTERNAL,
NR$RPS BYTE EXTERNAL,
HDW$INT$FLAG (MAX$CPU ) BYTE EXTERNAL,
GLOBA$LOCK BYTE EXTERNAL;

/*0166***************************************************************************/
/* DECLARATION OF EXTERNAL PROCEDURE REFERENCES */
/* DECLARED PUBLIC IN FILE "LEVEL1.SRC" */
/* IN MODULE "LEVEL1;MODULE" */

VPSCHEDULER: PROCEDURE EXTERNAL; END;
IN FILE "SCHED.ASM" */

F?V$VP : PROCEDURE BYTE EXTERNAL; END;

LOCATE$SVC : PROCEDURE (EVENT$NAME) BYTE EXTERNAL;
DECLARE EVENT$NAME BYTE;
END;

LOCATE$SEQ : PROCEDURE (SEQ$NAME) BYTE EXTERNAL;
DECLARE SEQ$NAME BYTE;
END;

/*0178***************************************************************************/
/* DIAGNOSTIC MESSAGES (WILL EVENTUALLY BE REMOVED) */

/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******
/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******
/* DECLARE
/* MSG16(*) BYTE INITIAL("ENTERING PREEMPT",13,10,",",).
/* MSG17(*) BYTE INITIAL("ISSUING INTERUPT!",13,10,",",).
/* MSG18(*) BYTE INITIAL("ENTERING AWAIT",13,10,",",).
/* MSG19(*) BYTE INITIAL("ENTERING ADVANCE",13,10,",",).
/* MSG21(*) BYTE INITIAL("ENTERING CREATE$SVC FOR",).
/* MSG23(*) BYTE INITIAL("ENTERING HEADER FOR SVC",).
/* MSG24(*) BYTE INITIAL("ENTERING TICKET",13,10,",",).
/* MSG25(*) BYTE INITIAL("ENTERING CREATE$SEQ",).
/* MSG27(*) BYTE INITIAL("ENTERING CREATE$PROC",10,13,",",).
/* MSG29(*) BYTE INITIAL("ENTERING GATE$KEEPER ",).

/* DECLARE
/* CS LITERALLY "OADY";
/* LP LITERALLY "PAH";

/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******
THI$ PROCEDURE IS THE ENTRY INTO THE OPERATING SYSTEM DOMAI$ FROM THE USER DOMAIN. THIS IS THE ACCESS POINT TO THE UTILITY/SERVICE ROUTINES AVAILABLE TO THE USER. THIS PROCEDURE IS CALLED BY THE GATE MODULE WHICH IS LINKED WITH THE USER PROGRAM. IT IS THE GATE MODULE WHICH PROVIDES TRANSLATION FROM THE USER DESIRED FUNCTION TO THE FORMAT REQUIRED FOR THE GATEKEEPER. THE GATEKEEPER CALLS THE DESIGNED UTILITY/SERVICE PROCEDURE IN LEVEL2 OF THE OPERATING SYSTEM AGAIN PERFORMING THE NECESSARY TRANSLATION FOR A PROPER CALL. THE TRANSLATIONS ARE INVISIBLE TO THE USER. THE GATEKEEPER ADDRESS IS PROVIDED TO THE GATE MODULE TO BE USED FOR THE INDIRECT CALL.

THE PARAMETER LIST IS PROVIDED FOR CONVENIENCE AND REPRESENTS NO FIXED MEANING, EXCEPT FOR 'N'.

PARAMETER LIST PROVIDED BY GATE

N FUNCTION CODE PROVIDED BY GATE

BYTE VARIABLE FOR TRANSLATION

WORD

PTV POINTER VARIABLE FOR TRANSLATION

THE I-O SERVICES ARE NOT ACKNOWLEDGED FOR TWO REASONS:

1. THEY ARE CALLED SO OFTEN THAT DIAGNOSTIC OUTPUT WOULD BE TOO CLUTTERED.

2. THEY THEMSELVES PRODUCE I-O EFFECTS THAT ACKNOWLEDGE THEY ARE BEING CALLED.

IF N < 9 THEN DO:

CALL OUT$LINE(OMSG27);

CALL OUT$NUM(N);

CALL OUT$CHAR(CR);

CALL OUT$CHAR(LF);

END;

DECLARE (N, BYT) BYTF,

WORDS WORD,

PTV POINTER;

DECLARE

GATEKEEPER: PROCEDURE(N, BYT, WORDS, PTR) REENTRANT PUBLIC:

87
DC CASE N;
   CALL $WIT(BYT,WORDS); /* 6 */
   CALL ADVANCE(BYT); /* 1 */
   CALL $ATE$FVC(BYT); /* 2 */
   CALL $ATE$SEQ(BYT); /* 3 */
   CALL TICKET(BYT, PTR); /* 4 */
   CALL READV(BYT, PTR); /* 5 */
   CALL $ATE$PROC(PTR); /* 6 */
   CALL $PTEMPT(BYT ); /* 7 */
   CALL CREATET$FVCPROCEDURE ROWE 6-22-84
   CALL OUT$CHAR(BYT); /* 8 */
   CALL OUT$LINE(PTR); /* 9 */
   CALL OUT$NUM(BYT); /* 10 */
   CALL OUT$NUM(WORDS); /* 11 */
   CALL IN$CHAR(PTR); /* 12 */
   CALL IN$NUM(PTR); /* 13 */
   CALL IN$NUM(PTR); /* 14 */
END; /* CASE */
RETURN; /* GAT$KEEPER */
END;

DECLARE NAME BYTE;

IF /* THE EVENTCOUNT DOES NOT ALREADY EXIST */
LOCATE$FVC(NAME) = NOT$FOUND THEN DO:
/* CREATE THE EVENTCOUNT ENTRY BY ADDING THE */
/* NEW EVENTCOUNT TO THE END OF THE FVC$TABLE */
EVC$TBL(EVENTS).EVC$NAME = NAME;
EVC$TBL(EVENTS).VALUE = $;
EVC$TBL(EVENTS).THREAD = 255;

98
INCREMENT THE SIZE OF THE EVCTABLE */
EVENTS = EVENTS + 1;

END; /* CREATE THE EVENTCOUNTER */
/* RELEASE THE GLOBAL LOCK */
GLOBAL$LOCK = 0;
RETURN;

END; /* CREATE$EVC PROCEDURE */

READ: PROCEDURE ( EVC$NAME, RETS$PTR ) REENTRANT PUBLIC;

DECLARE
EVC$NAME BYTE,
EVCTBL$INDEX BYTE,
RETS$PTR POINTER,
EVC$VALUES$RET BASED RETS$PTR WORD;

/* SET THE GLOBAL LOCK */
DO WHILE LOCKSET(GLOBAL$LOCK,119); END;

/**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
/**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
/**** CALL OUT$LINE(3MSG23); ****
/**** CALL OUT$NUM(EVC$NAME); ****
/**** CALL OUT$CHR(CR); ****
/**** CALL OUT$CHR(LF); ****
/**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
/**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/

/* OBTAIN INDEX */
EVCTBL$INDEX = LOCATE$EVC( EVC$NAME );

/* OBTAIN VALUE */
EVC$VALUES$RET = EVCTBL( EVCTBL$INDEX ).VALUE;

/* UNLOCK GLOBAL LOCK */
GLOBAL$LOCK = 0;
RETURN;
END; /* READ PROCEDURE */
/* Awaiting Procedure */

/*Interprocess synchronization primitive. Suspends */
/* Execution of running process until the eventcount has */
/* Reached the specified threshold value, "awaited$value." */
/* Used by the operating system for the management of */
/* System resources. */

*/
/*==============================================*/

Await: Procedure(EVCSID, Awaited$Value) Reentrant Public;

Declare

Awaited$Value Word,
(EVCSID, Need$Sched, Running$VP, EVCTBL$INDEX) Byte;

/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */
/* CALL OUTLINE(MSG12); */
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */

/* Lock global lock */
Do While Lock$Set(GLOBAL$LOCK, 119); End;
Need$Sched = TRUE;

/* Determine the running virtual processor */
Running$VP = RFS$VP;

/* Get EVCTBL$INDEX */
EVCTBL$INDEX = Locate$EVCTBL(EVCSID);
/* Determine if current value is less than the */
/* Awaited$Value */
If EVCTBL(EVCTBL$INDEX).Value < Awaited$Value Then Do;
/* Block process */
VPM(Running$VP).EVCSThread = EVCTBL(EVCTBL$INDEX).Thread;
VPM(Running$VP).EVCSAwa$Value = Awaited$Value;
EVCTBL(EVCTBL$INDEX).Thread = Running$VP;
DISABLE;
PDS.$Last$Run = Running$VP;
VPM(Running$VP).State = Waiting;
End; /* Block process */

Else /* Do not block process */
Need$Sched = FALSE;

/* Schedule the virtual processor */
If Need$Sched = TRUE Then
Call VPScheduler; /* No return */

/* Unlock global lock */
GLOBAL$LOCK = 0;
Return;

90
 /* AWAIT PROCEDURE */

/* ADVANCE PROCEDURE */

/* INTER PROCESS SYNCHRONIZATION PRIMITIVE. INDICATES */
/* SPECIFIED EVENT HAS OCCURRED BY ADVANCING/INCREMENTING */
/* THE ASSOCIATED EVENT ACCOUNT. EVENT IS BROADCAST TO ALL */
/* VIRTUAL PROCESSORS AWAITING THAT EVENT. */

/* CALLS MADE TO: OUTSLINE */

/* VPSCHEDULER (NO RETURN) */

/* ADVANCE PROCEDURE(EVC$ID) REENTRANT PUBLIC; */

DECLARE
(EVC$ID, NEED$SCHED, NEED$INTR, EVCTL$INDEX) BYTE,
(SAVE, RUNNING$VP, I, CPU) BYTE;

/** MYTRACE */
/** MXTRACF */
/** IXTRACE */
/** MXTRACE */
/** MXTRACF */
/** MXTRAC2 */
/** MXTRACE */
/**~cMXTRACF */
/** MXTRACF */
/** MXTRACE */
/** MXTRACE */
/** MXTRACE */
/** MXTRACE */
/** MXTRACE */

/* LOCK THE GLOBAL LOCK */
DO WHILE LOCKSET(GLOBAL$LOCK,119); END;

RUNNING$VP = RET$VP;
EVCTL$INDEX = LOCATE$EVC(EVC$ID);
EVC$TBL(EVCTL$INDEX).VALUE=EVC$TBL(EVCTL$INDEX).VALUE + 1;
NEED$SCHED = FALSE;
NEED$INTR = FALSE;
SAVE = 255;
I = EVC$TBL(EVCTL$INDEX).THREAD;
DO WHILE I <> 255;
  IF VPM(I).EVC$AW$VALUE <= EVC$TBL(EVCTL$INDEX).VALUE
  THEN DO; /* AWAKE THE PROCESS */
    VPM(I).STATE = READY;
    VPM(I).EVC$AW$VALUE = 0;
    CPU = I / MAX$VPS$CPU ;
  ELSE IF SAVE = 255 THEN DO; /* THIS FIRST ONE IN LIST */
    EVC$TBL(EVCTL$INDEX).THREAD=VPM(I).EVC$THREAD;
    VPM(I).EVC$THREAD = 255;
    I = EVC$TBL(EVCTL$INDEX).THREAD;
  END; /* IF FIRST */
  ELSE DO; /* THEN THIS NOT FIRST IN LIST */
    VPM( SAVE ).EVC$THREAD = VPM(I).EVC$THREAD;
    VPM(I).EVC$THREAD = 255;
    I = VPM( SAVE ).EVC$THREAD;

91
END; /* IF NOT FIRST */
IF CPU <> PRDS.CPU$NUMBER THEN DO:
HDWSINTSFLAG( CPU ) = TRUE;
NEED$INTR = TRUE;
END;
ELSE NEED$SCHED = TRUE;
END; /* IF AWAKEN */
ELSE DO; /* DO NOT AWAKEN THIS PROCESS */
SAVE = I;
I = VPM( I ).EVCS$THREAD;
END; /* IF NOT AWAKEN */
ENDIF; /* DC WHILE */
IF NEED$SINT = TRUE THEN DO; /* HARDWARE INT */
/*** MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE */
/** CALL OUT$LINE( @MSG17 );
/*** MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE MTRACE /*
DISABLE;
OUTPUT(PORT$CA) = 80H;
CALL TIME(1);
OUTPUT(PORT$CA) = RESET;
ENABLE;
END; /* NEED$INTR */
IF NEED$SCHED = TRUE THEN DO:
DISABLE;
PRDS.LAST$RUN = RUNNING$VP;
VPM(RUNNING$VP).STATE = READY;
CALL VPSCHEDULER; /* NO RETURN */
ENDIF; /* IF NEED$SCHED */
/* UNLOCK THE GLOBAL LOCK */
GLOBALSLOCK = 0;
RETURN;
END; /* ADVANCE PROCEDURE */

/* 581***************************************************************************/
/* PREEMPT PROCEDURE ROWE 6-22-84 */
*******************************************************************************/
/* THIS PROCEDURE AWAKENS A HIGH PRIORITY PROCESS LEAVING */
/* THE CURRENT RUNNING PROCESS IN THE READY STATE AND */
/* CALLS FOR A RESCHEDULING. THE HIGH PRIORITY PROCESS */
/* SHOULD BLOCK ITSELF WHEN FINISHED. */
/* IF THE VSID IS 'FE' OR THE MONITOR PROCESS, IT WILL */
/* MAKE IT READY WHEREVER IT IS IN THE VPM. THE FOLLOW- */
/*ING CODE DOES NOT TAKE ADVANTAGE OF THE FACT THAT */
/* CURRENTLY IT IS THE THIRD ENTRY IN THE VPM FOR EACH */
/* REAL PROCESS. */
*******************************************************************************/
/* CALLS MADE TO: OUTLINE, VPSCHEDULER */
*******************************************************************************/

PREEMPT: PROCEDURE( VSID ) REENTRANT PUBLIC;
DECLARE (VP$ID, SFARCH$ST, SFARCH$END, CPU, INDEX) BYTE;

/*** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
/*** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
/*** CALL OUTSLINE(OMSG16);
/*** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
/*** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****

IF VP$ID <> 0FEH THEN DO; /* NORMAL PREEMPT */
/* SEARCH VPM FOR INDEX FOR ID */
SEARCH$ST = 3;
DO CPU = 0 TO (NRSRPS - 1);
SEARCH$END = SEARCH$ST + NRSVPS(CPU) - 1;
DO INDEX = SEARCH$ST TO SEARCH$END;
   IF VPM(INDEX).VP$ID = VP$ID THEN GO TO FOUND;
END; /* DO INDEX */
SEARCH$ST = SEARCH$ST + MAX$VPS$CPU;
END; /* DO CPU */
/* CASE IF NOT FOUND IS NOT ACCOUNTED FOR CURRENTLY */
FOUND:
/* LOCK THE GLOBAL LOCK */
DO WHILE LOCK$SET(@GLOBAL$LOCK.119); END;
/* SET PREEMPTED VPM TO READY */
VPM(INDEX).STATE = READY;
/* NEED HARDWARE INTERRUPT OR RE-SEED */
IF (CPU = PRDS$CPU$NUMBER) THEN DO;
   INDEX = RFT$VP; /* DETERMINE RUNNING PROCESS */
   CALL VPSCHEDULER; /* NO RETURN */
END;
ELSE DO; /* CAUSE HARDWARE INTERRUPT */
/* CALL OUTSLINE(OMSG17); */
/* CALL OUTSLINE(OMSG17); */
END; /* NORMAL PREEMPT */
ELSE DO; /* PREEMPT THE MONITOR */
/* SEARCH VPM FOR ALL ID'S OF 0FH */
SEARCH$ST = 0;
DO WHILE LOCK$SET(@GLOBAL$LOCK.119); END;
DO CPU = 0 TO (NRSRPS - 1);
SEARCH$END = SEARCH$ST + NRSVPS(CPU) - 1;
/* SET ALL INT$FLAGS EXCEPT THIS CPU'S */
IF PRDS$CPU$NUMBER <> CPU THEN

93
HDW$INT$FLAG( CPU ) = TRUE;

DO ( INDEX = SEARCH$ST TO SEARCH$END;
   IF VPM( INDEX ).VPSID = VPSID THEN
      VPM( INDEX ).STATE = READY;
   END; /* DO */
   SEARCH$ST = SEARCH$ST + MAX$VPS$CPU;
END; /* ALL MONITOR PROCESS SET TO READY */
/* INTERRUPT THE OTHER CPU'S AND RESCHEDULE THIS ONE */

/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ****/
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ****/
/* CALL OUT$LINE( @MSG17 ); */
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ****/
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ****/
DISABLE;
OUTPUT( PORT$CA ) = 00H;
CALL TIME( 1 );
OUTPUT( PORT$CA ) = RESET;
ENABLE;
INDEX = PET$VP;
DISABLE;
PADS.LAST$RUN = INDEX;
VPM( INDEX ).STATE = READY;
CALL VPSCHEDULER; /* NO RETURN */
END; /* ELSE */
/* UNLOCK GLOBAL MEMORY */
GLOBAL$LOCK = 0;
RETURN;
END; /* PREAMPT PROCEDURE */

/* CREATE$SEQ PROCEDURE ROWE 6-22-84 */
/* -------------------------------------------*/
/* CREATE CF INTER PROCESS SEQUENCER PRIMITIVES FOR USE */
/* PROGRAMS, CREATE A SPECIFIED SEQUENCER AND INITIALIZE */
/* ZERES IT TO 0, BY ADDING THE SEQUENCER TO THE END OF THE */
/* SEQUENCER TAIL. */
/* */
/* CALLS MADE TO: OUT$LINE OUT$CHAR */
/* CALL$HEX */
/* -------------------------------------------*/

CREATE$SEQ: PROCEDURE( NAME ) REENTRANT PUBLIC;

DECLARE NAME BYTE;

/* ASSERT GLOBAL LOCK */
DO, WHILE LOCKSET( @GLOBAL$LOCK, 113 ); END;

/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ****/
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ****/
/* CALL OUT$LINE(GMSG25); */
/* CALL W$HEX(NAME); */
/* CALL OUT$CHAR(CR); */
/* CALL OUT$CHAR(IF); */
/**** MRETRACF ***** MRETRACF ***** MRETRACF ***** MRETRACF ***/
/**** MRETRACF ***** MRETRACF ***** MRETRACF ***** MRETRACF ***/

IF /* THE SEQUENCER DOES NOT ALREADY EXIST, IE */
LOCATE$SEQ(NAME) = NOT$FOUND THEN DO;
/* CREATE THE SEQUENCER ENTRY BY ADDING THE */
/* NEW SEQUENCER TO THE END OF THE SEQ$TABLE */
SEQ$TABLE(SEQUENCERS).SEQ$NAME = NAME;
SEQ$TABLE(SEQUENCERS).SEQ$VALUE = 0;
/* INCREMENT NUMBER OF SEQUENCERS */
SEQUENCERS = SEQUENCERS + 1;
END; /* CREATE THE SEQUENCER */
/*PLEASE THE GLOBAL LOCK */
GLOBAL$LOCK = 0;
RETURN;
END; /* CREATE$SEQ PROCEDURE */

/*769***************************************************************************************************/
/* TICKET PROCEDURE ROWE 6-22-94 */
*---------------------------------------------------------------------*/
/* IFP - VIRTUAL PROCESSOR SEQUENCER PRIMITIVE FOR USE */
/* PROCAM, SIMILAR TO "TAKE A NUMBER AND WAIT." RETURNS */
/* PRESENT VALUE OF SPECIFIED SEQUENCER AND INCREMENTS THE */
/* SEQUENCER. A POINTER IS PASSED TO PROVIDE A BASE TO A */
/* VARIABLE IN THE CALLING Routine FOR PASSING THE RETURN */
/* VALUE BACK TO THE CALLING ROUTINE. */
*---------------------------------------------------------------------*/
/* CALLS MADE TO: OUT$LINE */
*/p*************************************************************/

TICKET: PROCEDURE(SEQ$NAME, RETS$PTR) REENTRANT PUBLIC

DECLARE
SEQ$NAME BYTE,
SEQ$TABLE INDEX BYTE,
RETS$PTR POINTER,
SEQ$VALUE$RET BASED RETS$PTR WORD;

/* ASSERT GLOBAL LOCK */
DO WHILE LOCKSET(GLOBAL$LOCK,119);
END;

/**** MRETRACF ***** MRETRACF ***** MRETRACF ***** MRETRACF ***/
/**** MRETRACF ***** MRETRACF ***** MRETRACF ***** MRETRACF ***/
/**** CALL OUT$LINE(GMSG24); */
/**** MRETRACF ***** MRETRACF ***** MRETRACF ***** MRETRACF ***/
/**** MRETRACF ***** MRETRACF ***** MRETRACF ***** MRETRACF ***/
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A
/* OBTAIN SEQ$NAME INDEX */
SEQ$TABLE$INDEX = LOCATESEQ( SEQ$NAME );
/* OBTAIN SEQUENCER VALUE */
SEQ$VALUE$PTR = SEQ$TABLE( SEQ$TABLE$INDEX ).SEQ$VALUE;
/* INCREMENT SEQUENCER */
SEQ$TABLE( SEQ$TABLE$INDEX ).SEQ$VALUE =
SEQ$TABLE( SEQ$TABLE$INDEX ).SEQ$VALUE + 1 ;

/* UNLOCK THE GLOBAL LOCK */
GLOBAL$LOCK = 0;
RETURN;
END; /* TICKET PROCEDURE */

/*-------------------------------------------------------------
/* THIS PROCEDURE CREATES A PROCESS FOR THE USEL AS */
/* SPECIFIED BY THE INPUT PARAMETERS CONTAINED IN A */
/* STRUCTURE IN THE GATE MODULE. THE PARAMETER PASSED */
/* IS A POINTER WHICH POINTS TO THIS STRUCTURE. */
/* INFO CONTAINED IN THIS STRUCTURE IS: PROCESS ID, */
/* PROCESS PRIORITY, THE DESIRED PROC STACK LOCATION, */
/* AND THE PROCESS CODEF STARTING LOCATION WHICH IS */
/* IS TWO ELEMENT: THE IP REGISTER (OFFSET) AND THE */
/* CS REGISTER (CODE SEGMENT). */
/*-------------------------------------------------------------
/* CALLS MAOF TO: OUTLINE */
/*-------------------------------------------------------------

CREATE$PROC: PROCEDURE( PROC$PTR ) REENTRANT PUBLIC:

DECLARE
PROC$PTR: POINTER,
PROC$TABLE BASED PROC$PTR STRUCTURE
(PROC$ID BYTE,
PROC$PRI BYTE,
PROC$SP WORD,
PROC$SS WORD,
PROC$IP WORD,
PROC$CS WORD,
PROC$ST WORD,
PROC$ES WORD);

DECLARE
(PS1, PS2) WORD,
TEMP BYTE;

DECLARE PROC$STACK$PTR POINTER AT(@PS1),
PROC$STACK BASED PROC$STACK$PTR STRUCTURE
.LENGTH(OFFH) BYTE,
FET$TYPE = WORD,
BP = WORD,
DI = WORD,
SI = WORD,
DS = WORD,
DX = WORD,
CX = WORD,
AX = WORD,
BX = WORD,
ES = WORD,
IP = WORD,
CS = WORD,
FL = WORD;

/** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */
/** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */
CALL OUTSLINE(@MSG26);
/** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */
/** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE */
/* TO SET UP PROC$STACK$PT */
PS1 = PROC$TABLE.PROC$SP - 118H;
PS2 = PROC$TABLE.PROC$SS;

PROC$STACK.FET$TYPE = INT$RETURN;
PROC$STACK.BP = PROC$TABLE.PROC$SP;
PROC$STACK.DI = 0;
PROC$STACK.SI = 0;
PROC$STACK.DS = PROC$TABLE.PROC$DS;
PROC$STACK.DX = 0;
PROC$STACK.CX = 0;
PROC$STACK.AX = 0;
PROC$STACK.BX = 0;
PROC$STACK.ES = PROC$TABLE.PROC$ES;
PROC$STACK.IP = PROC$TABLE.PROC$IP;
PROC$STACK.CS = PROC$TABLE.PROC$CS;
PROC$STACK.FL = 200H; /*SET IF FLAG (ENABLE INTR)*/

/* SET GLOBAL LOCK */
DC WHILE LOCKSET(QGLOBAL$LOCK,119); END;

IF PRDs.VP$PER$CPU < MAX$VP$CPU THEN DO;
TEMP = PRDs.VP$PER$CPU + PRDs.VP$START;
VPM( TEMP ).VP$ID = PROC$TABLE.PROC$ID;
VPM( TEMP ).STATE = 01; /* READY */
VPM( TEMP ).VP$PRIORITY = PROC$TABLE.PROC$PRI;
VPM( TEMP ).EVCTH$THAD = 255;
VPM( TEMP ).EVCSAW$VALUE = 2;
VPM( TEMP ).SS$REG = PROC$TABLE.PROC$SP - 1AH;
VPM( TEMP ).SS$REG = PROC$TABLE.PROC$SS;
PRDs.VP$PER$CPU = PRDs.VP$PER$CPU + 1;
PRDs.VP$END = PRDs.VP$END + 1;
NR5PS(PDDS.CPUSNUMBER) =
NR5PS(PDDS.CPUSNUMBER) + 1;
END; /* DO */

/* ELIFS THERE GLOBAL LOCK */
GLOBALLOCK = 0;
RETURN;
END; /* CREATE$PROCESS */

/******************** PROCEDEURED RCE 6-22-84 */
/* GETS A CHAR FROM THE SERIAL PORT. CHAR IS !!NCT!!! */
/* ECHOED. THAT IS RESPONSIBILITY OF USER IN THIS CASE. */
/* INPUT TO SERIAL PORT VIA SBC861 DOWN LOAD PROGRAM MAY */
/* NOT BE ACCEPTED. */
/* POINTER IS PROVIDED BY USER SO HE CAN BE RETURNED THE */
/* CHARACTER. */
/*---------------------------------------------*/
/* CALLS MADE TO: RCVSCHAR */
/**************************************************************************/

/*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ****/
/*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ****/
/*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ****/
/* IN$CHAR: PROCEDURE ( RETS$PTR ) REENTRANT PUBLIC */
/* DEFINE */
/* RETS$PTR POINTER, */
/* INCHR BASED RETS$PTR BYTE; */
/* DISABLE; */
/* INCHR = RCVS$CHAR; */
/* ENABLE; */
/* RETURN; */
/* FND; /* IN$CHAR */

/*-------------------------------*/
/* MTRACE **** MTRACE **** MTRACE **** MTRACE *****/
/* MTRACE **** MTRACE **** MTRACE **** MTRACE *****/
/*-------------------------------*/
/* IN$NUM PROCEDURE RCE 6-22-84 */
/*-------------------------------*/
/* GETS TWO ASCII CHAR FROM THE SERIAL PORT, EXAMINES */
/* THEM TO SEE IF THEY ARE IN THE SET OF HEX AND POINTS */
/* A BYTE VALUE. EACH VALID HEX DIGIT IS ECHOED TO THE */
/* CRT. IMPROPER CHAR ARE IGNORED. NO ALLOWANCE ARE */
/* MADE FOR WRONG DIGITS. GET IT RIGHT THE FIRST TIME. */
/* IF YOU ARE INDIRECTLY ACCESSING THE SERIAL PORT VIA */
/* THE SBC861 DOWN LOAD PROGRAM FROM THE MDS SYSTEM */
/* INPUT MAY NOT BE ACCEPTED. A POINTER IS PASSED BY THE*/
/* USER SO THAT HE RETURNED THE CHARACTER. */

99
/*---------------------------------------------*/
/* CALLS MADE TO: IN$HEX */
/* *********************************************************/

/*** MXTACE ***** MXTACE ***** MXTACE *****/
/*** MXTACE ***** MXTACE ***** MXTACE *****/
/*** IN$NUM: PROCEDURE ( RET$PTR ) ENTERANT PUBLIC; */
/* DEFINE */
/* RET$PTR POINTFR, */
/* NUM BASED RET$PTR BYTE; */

/** */
/* DISABLE; */
/* NUM = IN$HEX; */
/* ENABLE; */
/* RETURN; */
/* END; /* IN$NUM */
/*** MXTACE ***** MXTACF ***** MXTACE *****/
/*** MXTACE ***** MXTACE ***** MXTACE *****/

/* 1034*********************************************************************/
/* OUTSCAR PROCEDURE ROWE 6-22-84 */
/*----------------------------------------------------------------------*/
/* SENDS A BYTE TO THE SERIAL PORT */
/*----------------------------------------------------------------------*/
/* CALL MADE TO: SEND$CHAR */
/*----------------------------------------------------------------------*/

/*** MXTACE ***** MXTACE ***** MXTACE *****/
/*** MXTACE ***** MXTACE ***** MXTACE *****/
/* OUT$CHAR: PROCEDURE( CHAR ) ENTERANT PUBLIC; */
/* DECLARE CHAR BYTE; */
/* DISABLE; */
/* CALL SEND$CHAR( CHAR ); */
/* ENABLE; */
/* RETURN; */
/* END; */
/*** MXTACE ***** MXTACE ***** MXTACE *****/
/*** MXTACE ***** MXTACE ***** MXTACE *****/

/* 1072*********************************************************************/
/* OUT$LINE PROCEDURE ROWE 6-22-84 */
/*----------------------------------------------------------------------*/
/* USING A POINTER TO A BUFFER IT WILL OUTPUT AN ENTIRE */
/* LINE THRU THE SERIAL PORT UNTIL AN 'N' IS ENCOUNTERED */
/* OR 90 CHARACTERS IS REACHED--WHICH EVER IS FIRST. CR'S*/
/* AND LF'S CAN BE INCLUDED. */
/*----------------------------------------------------------------------*/
/ * CALLS MADE TO: SEND$CHAR * / 
/ *******************************************************************************/

/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */
/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */
/ ** OUT$LINE: PROCEDURE( LINESPTR ) REENTRANT PUBLIC; */

/ * DECLARE */
/ * LINESPTR POINTER, */
/ * LINE BASED LINESPTR (30) BYTE, */
/ * II BYTE; */

/ * DISABLE; */
/ * DO II = 0 TO 79; */
/ * IF LINE( II ) = 'X' THEN GO TO DONE; */
/ * CALL SEND$CHAR( LINE( II ) ); */
/ * END; */
/ * DONE: ENABLE; */
/ * RETURN; */
/ ** END; */
/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */
/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */

/ **1104***************************************************************************** */
/ ** OUT$NUM: PROCEDURE ROWE 6-22-84 */
/ ******************************************************************************* */
/ ** OUTPUTS 8 BYTE VALUE NUMBER THRU THE SERIAL PORT */
/ ******************************************************************************* */
/ ** CALLS MADE TO: OUT$HEX */
/ ******************************************************************************* */

/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */
/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */
/ ** OUT$NUM: PROCEDURE( NUM ) REENTRANT PUBLIC; */

/ * DECLARE NUM BYTE; */

/ * DISABLE; */
/ * CALL OUT$HEX( NUM ); */
/ * ENABLE; */
/ * RETURN; */
/ ** END; */
/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */
/ *** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** */

/ **1142***************************************************************************** */
/ ** IN$ENUM: PROCEDURE ROWE 6-22-34 */
/ ******************************************************************************* */
/ ** GETS FOUR ASCII FROM SERIAL PORT TO FORM WORD VALUE. */
/* 100 */
OUTSDNUM: PROCEDURE ( IN$NUM ) REENTRANT ;

DFCLAFE

OUT$PTR  POINTER,
IN$NUM BASED OUT$PTR WORD,
(H, L) WORD;

DISABLE;

H = IN$HEX;

F = SHL ( H, 8 );

L = IN$HEX;

DNUM = ( H OR L );

ENABLE;

RETURN;

END;

OUTSDNUM: PROCEDURE ( OUT$DEC ) REENTRANT PUBLIC;

DFCLAFE

DNUM WORD,
SEND BYTE;

DISABLE;

SEND = HIGH ( DNUM );

CALL OUT$HFX ( SEND );

SEND = LOW ( DNUM );

CALL OUT$HFX ( SEND );

ENABLE;

RETURN;

END;
/*-bottom level procedure that obtains a char from the serial port. parity bit is removed. char is not echoed.*/
/* calls made to: none */

כיושתôleו, שורת 126:
* declare byte;
* check port status bit 2 for receive-ready signal; d
* do while (input(0da) and 02h) < 0; end;
* chp = (input(0d8) and 07ff);
* return cur;
* end;
** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ***** mtrace ***** mtrace ***/**** mtrace ***** mtrace ***** mtrace ****
** INCHR = 13H THEN
   TO WHILE (INCHR <> 11H);
   IF ((INPUT(0DH) AND 02H) <> 2) THEN
      INCHR = (INPUT(0DH) AND 07H);
   END;
   DO WHILE (INPUT(0DH) AND 01H) = 7; END;
   OUTPUT(0DEH) = CHAR;
   RETURN;
   END;
   ** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
   ** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/

   1263******************************************************************************
   ** IN$HEX  PROCEDURE  ROWE 6-22-94  */
   ** -----------------------------------------------------------
   ** GETS 2 HEX CHAR FROM THE SERIAL PORT AND IgNORES ANY- */
   ** THING ELSE. EACH VALID HEX DIGIT IS EchoED TO THE */
   ** SERIAL PORT. A BYTE VALUE IS FORMED FROM THE TWO HEX */
   ** CHAR. */
   ** -----------------------------------------------------------
   ** CALLS MADE TO:  RECV$CHAR */
   ** IN$HEX:  PROCEDURE BYTE REENTRANT PUBLIC:
   **
   ** DECLARE */
   ** ASCII(*) BYTE DATA ('0123456789ABCDEF'), */
   ** ASCIIH(*) BYTE DATA ('0123456789',61H,62H,63H,64H,65H, */
   ** 66H), */
   **
   ** (INCHR, HEXNUM, H, L) BYTE, */
   ** FOUND BYTE, */
   ** STOP BYTE;
   **
   ** /* GET HIGH PART OF BYTE */
   ** FOUND = 0;
   ** DO WHILE NOT FOUND;
   ** /* IF INVALID CHAR IS INPUT, COME BACK HERE */
   ** INCHR = RECV$CHAR;
   ** H = 0;
   ** STOP = 0;
   ** /* COMPARE CHAR TO HEX CHAR SET */
   ** DO WHILE NOT STOP;
   ** IF (INCHR=ASCII(F)) OR (INCHR = ASCIIH(F)) THEN DO:
     ** STOP = OFFFH;
     ** FOUND = 2FFH;
     ** CALL SEND$CHAR( INCHR ); /* TO Echo IT */
     ** END;
   ** END DO;
```c
/*
  H = H + 1;
  IF H = 10H THEN STOP = 0FFH;
  END; /* ELSE */
  END; /* DO WHILE */
  H = SHL( H, 4 );
  END; /* DO WHILE */
  FOUND = 0;
  /* GET LOW PART OF BYTE */
  DO WHILE NOT FOUND;
  /* AGAIN DO UNTIL VALID HEX CHAR IS INPUT */
  INCHR = RECVCCHAR;
  L = 0H;
  STOP = 0;
  DO WHILE NOT STOP;
  IF (INCHR=ASCII(L)) OR (INCHR=ASCIIH(L)) THEN DO:
    STOP = 0FFH;
    FOUND = 0FFH;
    CALL SENDSCHAR(INCHR);
    END;
  ELSE DO:
    L = L + 1;
    IF L = 10H THEN STOP = 0FFH;
  END; /* ELSE */
  END; /* DO WHILE */
  RETURN (H OR L);
  END; /* INSHEX */
*/

*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE *****
*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE *****

/* OUT$FX PROCEDURE ROWE 6-22-84 */
**---------------**
* TRANSLATES BYTE VALUES TO ASCII CHARACTERS AND OUTPUTS*
* THEN THRU THE SERIAL PORT*
**---------------**
* CALLS MADE TO: SENDSCHAR*
**---------------**

*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE *****
*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE *****
* OUT$FX: PROCEDURE(B) REENTRANT PUBLIC:
*/

DECLARE P BYTE;
DECLARE ASCII(*) BYTE DATA ("0123456799ABCDEF");
CALL SENDSCHAR(ASCII(SHR(P,4) AND $FFH));
CALL SENDSCHAR(ASCII(P AND $FFH));
RETURN;
*/
** END;
*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE *****
*** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE *****
```
END; /* L25MODULE */

/*************************************************************/
  
/*************************************************************/

/*************************************************************/
APPENDIX E

LEVEL I -- MCO\textsc{TEX} SOURCE CODE

All the LEVEL I source code written in PL/M is contained in the file LEVEL1.SRC. It is compiled with the \texttt{LARGE} attribute. Two other LEVEL I functions, SCHEDULER and INTERRUPT HANDLER, were written in ASMS6 and are listed in their own modules. LEVEL I is one of the relocatable code modules in file: KORE.LNK. It is part of the executable code module in file: KORE. KORE is the development system version of the file KORE.OPS loaded by MCO\textsc{TEX}.CMD under the CP/M-86 operating system. This module contains utility procedures used only by the operating system. Two memory maps of KORE (.OPS and .TRC) are located at the end of this appendix. The maps come from file: KORE.MP2 after compiling, linking and locating the applicable files. KORE(OPS) is produced with the code unaltered. KORE(TRC) is obtained by removing and adding appropriate comment marks from the indicated code before processing.
```c
/* LOCAL DECLARATIONS */
DECLARE
    MAX$CPU LITERALLY '12',
    MAX$VP$CPU LITERALLY '10',
    MAX$CPU$VP$CPU LITERALLY '100',
    FALSE LITERALLY '0',
    READY LITERALLY '1',
    RUNNING LITERALLY '3',
    WAITING LITERALLY '5',
    TRUE LITERALLY '119',
    NOT$FOUND LITERALLY '255',
    PORT$C LITERALLY '0C0F',
    PORT$C2 LITERALLY '0C2F',
    PORT$CE LITERALLY '0CEH',
    PORT$CA LITERALLY '0C4H',
    RESET LITERALLY '7F',
    INT$RETURN LITERALLY '7F',

/**** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX ****/
    IDLE$STACK$SEG LITERALLY '0C80',/****
    IDLE$STACK$SAP LITERALLY '0C80',/****
    INIT$STACK$SEG LITERALLY '0C80',/****
    INIT$STACK$SAP LITERALLY '0C80',/****
/**** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX ****/
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/
```
/ * PM(INDFX).STATE = WAITING: 
/ * CALL VPSCHEDULER: /* NO RETURN */ 
/ * END; /* IF */ 
/ * GO TO LOOP; 
/ * END; /* MONITOR PROCESS */ 
/ **** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** 
/ **** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** 

/ * STARTING POINT OF THE OPERATING SYSTEM */ 
/ * "---------------------------------------------------" */ 
/ * ROUTINE INITIALIZES THE OS AND IS NOT REPEATED */ 
/ * "---------------------------------------------------" */ 

/ * TO INITIALIZE THE PPD$ TABLE FOR THIS CPU */ 
DECLARE CPU$PPD$ POINTER DATA(@PRDS.CPU$NUMBER), ZZ BYTES: 
DISABLE; 
/ **** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** 
/ **** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** 
/ CALL CUTF$LINE(2MSG12); 
/ **** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** 
/ **** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** 

/ * INITIALIZE P P I AND P I C */ 
OUTPUT(POT$CE) = 00H; /* PPI - MICROPOLIS + MCCORTEX */ 
OUTPUT(POT$CP) = 13H; /* PIC - ICW1 - EDGE TRIGGERED */ 
OUTPUT(POT$SC2) = 42H; /* PIC - ICW2 -VECTOR TABLE ADDRESS */ 
OUTPUT(POT$SC3) = 0FH; /* PIC - ICW4 -MCSHE MODE. AUTO FCI */ 
OUTPUT(POT$SC2) = 0AH; /* PIC - MASK ALLOWING INT. < & E */ 

/* ESTABLISH UNIQUE SEQUENTIAL NUMBER FOR THIS CPU */ 
/* SET GLOBAL$LOCK */ 
do while lock$SET(@GLOBAL$LOCK, 119); end; 
PRDS.CPU$NUMBER = CPU$INIT; 
CPU$INIT = CPU$INIT + 1; 
/* PTLF: SF GLOBAL LOCK */ 
GLOBAL$LOCK = 0; 

/* SET UP INITIAL START AND END FOR PRCC TABLE */ 
PRDS.VPS$START = 0; 
do ZZ = 1 TO PRDS.CPU$NUMBER; 
PRDS.VPS$START = PRDS.VPS$START + MAX$VPS$CPU; 
end; 
/ **** MCCORTEX ***** MCCORTEX ***** MCCORTEX ***** MCCORTEX ***** 
/ **** MCCORTEX ***** MCCORTEX ***** MCCORTEX ***** MCCORTEX ***** 

121
/*
/* CURRENT CONTENTS SHOULD NOT BE AVAILABLE */
*/
CALL OUTSNUM( CONTENTS );
LOOP2 = TRUE;
DC WHILE LOOP2 = TRUE;
   DO WHILE ( INCHR<>' ',') AND (INCHR<>' ' )
      AND ( INCHR<>'.' )
      CALL IN$CHAR( @INCHR );
   END;
/* SKIP THIS AND GO TO NEXT FOR SUB */
IF ( INCHR = ' ' ) THEN LOOP2 = FALSE;
IF ( INCHR = ',' ) THEN DO;
   CALL OUT$CHAR( CR );
   CALL OUT$CHAR( LF );
   ADDR.OFFSET = ADDR.OFFSET + 1;
   PTR = PTR3;
   CALL OUTSNUM( ADDR.BASF );
   CALL OUT$CHAR( ':' );
   CALL OUTSNUM( ADDR.OFFSET );
   CALL OUT$CHAR( '-' );
   CALL OUTSNUM( CONTENTS );
   END; /* IF SKIP FOR NEXT SUB */
IF ( INCHR = ',' ) THEN DC;
CALL OUT$CHAR( ':' );
CALL IN$NUM( CONTENTS );
DO WHILE ( INCHR<>' ' ) AND ( INCHR<>'.' )
      CALL IN$CHAR( @INCHR );
END;
IF ( INCHR = CR ) THEN LOOP2 = FALSE;
IF ( INCHR = ',' ) THEN DC;
ADDR.OFFSET = ADDR.OFFSET + 1;
PTR = PTR3;
CALL OUT$CHAR( CR );
CALL OUT$CHAR( LF );
CALL OUTSNUM( ADDR.BASF );
CALL OUT$CHAR( ':' );
CALL OUTSNUM( ADDR.OFFSET );
CALL OUT$CHAR( '-' );
CALL OUTSNUM( CONTENTS );
END; /* IF GO TO NEXT ADDR */
END; /* IF CHANGE CONTENTS */
INCHR = 'X'; /* E. INITIALIZE CMD */
END; /* LOOP, CONTINUOUS SUB CMD */
END; /* SUBSTITUTE COMMAND SECTION */
*/
IF ( INCHR='E' ) OR ( INCHR='EH' ) THEN DO:
/* FIND OUT WHICH WPS IS RUNNING 'MR' */
INDEX = EXT$VP;
/* NOW LOCK MYSELF */
DISABLE;
PFS.LAST$UN = INDEX;
*/
/* IF (INCHR = '64H) OR (INCHR = '65H) THEN CALL OUT$CHAR(INCHR); */
/* VALID$CMD = @PPH; */
/* IF VALID$CMD = OFFH THEN CALL OUT$CHAR(INCHR); */
/* END; /* DO WHILE */
/* IF (INCHR = 'D') OR (INCHR = '64H) THEN DO; */
/* DISPLAY COMMAND SECTION */
/* CALL INS$NUM(ADDR,BASE); */
/* CALL OUT$CHAR(':''); */
/* CALL INS$NUM(ADDR,OFFSET); */
/* PTR2 = ADDR; */
/* PTR = PTR3; */
/* CONTENTS SHOULD NOW BE SET */
/* DO WHILE (INCHR<>CR) AND (INCHR<>23H); */
/* CALL INS$CHAR(INCHR); */
/* END; /* DO WHILE */
/* IF INCHR = CF THEN DO; */
/* CALL OUT$CHAR('-'); */
/* CALL OUT$NUM(CONTENTS); */
/* CALL OUT$CHAR(CR); */
/* CALL OUT$CHAR(LF); */
/* END; /* IF NORMAL I ADDF DISPLAY */
/* IF INCHR = 23H THEN DC; */
/* COUNT = 0; */
/* CALL OUT$CHAR('#'); */
/* CALL INS$NUM(QANTITY); */
/* DO WHILE QANTITY > 0; */
/* CALL OUT$CHAR(CF); */
/* CALL OUT$CHAR(LF); */
/* CALL OUT$NUM(ADDR,BASE); */
/* CALL OUT$CHAR(':''); */
/* CALL OUT$NUM(ADDR,OFFSET); */
/* LINECOMPLETE = FALSE; */
/* DO WHILE LINECOMPLETE = FALSE; */
/* CALL OUT$CHAR(''); */
/* CALL OUT$NUM(CONTENTS); */
/* ADDR.OFFSET = ADDR.OFFSET + 1; */
/* PTR = PTR3; */
/* QANTITY = QANTITY - 1; */
/* IF ((ADDR.OFFSET ANT 220FFH)=2) OR */
/* (QANTITY = 0) THEN LINECOMPLETE=TRUE; */
/* END; /* DO WHILE LINE NOT COMPLETE */
/* END; /* DC WHILE QANTITY */
/* END; /* IF MULTI ADDF DISPLAY */
/* END; /* DISPLAY COMMAND SECTION */
/* IF (INCHR='S') OR (INCHR='73H) THEN DC; */
/* SUBSTITUTE COMMAND SECTION */
/* CALL INS$NUM(ADDR,BASE); */
/* CALL OUT$CHAR(':''); */
/* CALL INS$NUM(ADDR,OFFSET); */
/* CALL OUT$CHAR('-'); */
/* PTR2 = ADDR; */

119
CALL TIME( 250 );

PEND;

***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****
***** MXTRACE ***** MXTRACE ***** MXTRACE *****
/* CALL OUT$LINE( MSG11 );
***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****
***** MXTRACE ***** MXTRACE ***** MXTRACE *****

PDS.COUNTF = PDS.COUNTER + 1;
GO TO LOOP;
PEND; /* IDLE$PROC */

**7E9**
// MONITOR PROCESS
// ** THE MONITOR PROCESS IS INITIALIZED BY THE CS LIKE */
// INIT AND IDLE. IT HAS THE RESERVED ID OF APPF AND A */
// PRIORITY OF 26. IT IS ALWAYS BLOCKED OR WAITING UNTIL*/
// IT IS PREEMPTED BY IF$ USER.

/* CALLS MADE TO: OUT$LINE OUT$CHAR */
/* OUT$NUM IN$NUM */
/* IN$NUM */

/*****************************/

***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****
***** MXTRACE ***** MXTRACE ***** MXTRACE *****
/* MONITOR$PROC: PROCEDURE REENTRANT PUBLIC; */
/* DECLARE */
/* PTR POINTER, */
/* PTR2 POINTER, */
/* PTR3 BASED PTR2 POINTER, */
/* ADDR STRUCTURE (OFFSET WORD, EASE WORD), */
/* CONTENTS BASED PTR BYTE; */
/* DECLARE */
/* (LINECOMPLETE, LCOP2) BYTE, */
/* (QUANTITY, COUNT) BYTE, */
/* (IN$CHR, INDEX, VALID$CMD) BYTE; */
/* LOC$P: VALID$CMD = $; */
/* CALL OUT$CHAR( 'C' ); */
/* CALL OUT$CHAR( 'L' ); */
/* CALL OUT$CHAR( '.' ); */
/* DO WHILE NOT VALID$CMD; */
/* CALL INS$CHAR( 'IN$CHR$' ); */
/* IF (IN$CHR = 'D') OR (IN$CHR = 'S') OR (IN$CHR = 'E') THEN */
/* VALID$CMD = APPF; */

118
END; */ WHILE */
 IF (MATCH = TRUE) THEN DO;
 /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
 /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
 /** CALL CUTLINE(@MSG23); */
 /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
 RETURN NOTFOUND;
 END; */ IF */
 ELSE DO;
 /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
 /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
 /** CALL CUTLINE(@MSG24); */
 /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
 RETURN NOTFOUND;
 END; */ ELSE */
END; /* LOCATESEC PROCEDURE */

**698***************************************************/
/** SYSTEM PROCESSES */
**

******************************************************************************
*/ IDLE PROCESS RC&L E-22-24 */
******************************************************************************
*/ THIS PROCESS IS SCHEDULED IF ALL OTHER PROCESSES IN */
*/ THE RPM ARE BLOCKED. THE STARTING ADDRESS IS PROVIDED TO */
*/ THE IDLESTACK AND PLACED IN PRDS.IDLESP. */
*/ COUNTER IS INCREMENTED ABOUT EVERY SECOND. THE COUNT */
*/ IS MAINTAINED IN THE PRDS TABLE AND IN A ROUGH MEASURE */
*/ OF SYSTEM PERFORMANCE BY GIVING AN INDICATION OF THE */
*/ AMOUNT OF TIME SPENT IN THE IDLE PROCESS. */
******************************************************************************
*/ CALLS "ADC TO: PLMBR PROCEDURE TIME" */
/* OUTFINE */
******************************************************************************
IDLEPROC: PROCEDURE REENTRANT PUBLIC;

DECLARE I BYTE;

******************************************************************************
*/ MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** */
*/ MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** */
/* CALL CUTLINE(@MSG10); */
******************************************************************************
*/ MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
/* MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/

/* DELAYS ONE (1) SECOND */
LOOP: DO I = 1 TO 4?;
LOCATESSEQ: PROCEDURE(SEQSNAMF) BYTE REENTRANT PUBLIC;

DECLARE SEQSNAMF BYTE;
DECLARE (MATCH, SEQETLSINDEX) BYTE;
/* MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** */
/* MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** */
/* CALL OUTSLINE(@MSG23); */
/* MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** */
/* MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** MTRACE ***** */
MATCH = FALSE;
SEQETLSINDEX = 0;
DO WHILE (MATCH = FALSE) AND (SEQETLSINDEX < SEQUENCES);
   IF SEQSNAMF = SEQUENCE(SEQETLSINDEX).SEQSNAMF THEN MATCH = TRUE;
ELSE
   SEQETLSINDEX = SEQETLSINDEX + 1;
END; /* LOCATESSEQ */
END; /* DO LOOP SEARCH OF VPM */

/* SET SELECTED VIRTUAL PROCESSOR */
VM(N).STATE = RUNNING;
SELECTED$DBR = VM(N).$SSREG;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/
/* CALL OUT$LINE(OMSG7A);*/
/* CALL OUT$CHAR(N);*/
/* CALL OUT$CHAR(CR);*/
/* CALL OUT$CHAR(LF);*/
/* CALL OUT$LINE(OMSG7B);*/
/* CALL OUT$CHAR(SSELECTED$DBR);*/
/* CALL OUT$CHAR(CR);*/
/* CALL OUT$CHAR(LF);*/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/

RETURN SELECTED$DBR;

END; /* GETWORK PROCEDURE */

LOCATES$EVK: PROCEDURE(EVENT$NAME) BYTE REENTRANT PUBLIC;

DECLARE EVENT$NAME BYTE;
DECLARE (MATCH, EVK$INDEX) BYTE;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/
/* CALL OUT$LINE(OMSG20);*/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE *****/

MATCH = FALSE;
EVK$INDEX = 0;
/* SEARCH DOWN THE EVENTCOUNT TABLE TO LOCATE THE */
/* DESIRED EVENTCOUNT BY MATCHING THE NAMES */
/* DO WHILE (MATCH = FALSE) AND (EVK$INDEX < EVENTS);*/
/* DO WHILE HAVE NOT FOUND THE EVENTCOUNT AND HAVE NOT */
/* REACHED END OF THE TABLE */
IF EVENT$NAME = EVK$INDEX(EVK$INDEX).EVK$NAME THEN

115
GET$SP: PROCEDURE WORD REENTRANT PUBLIC;

DECLARE N BYTE;
N = RET$VP; /* GET CURRENT RUNNING VIRTUAL PROCESSOR */

RETURN VPM(N).SP$REG; /* RETURN NEW VP STACK POINTER */

END;

GETWORK: PROCEDURE WORD REENTRANT PUBLIC;

DECLARE (PRI,N,I) BYTE;
DECLARE SELECTED$DER WORD;
DECLARE DISPLAY BYTE;

**** MXTSPACE **** MXTSPACE **** MXTTRACE **** MXTTRACE **** MXTTRACE **** MXTTRACE ****
**** MXTSPACE **** MXTSPACE **** MXTSPACE **** MXTSPACE **** MXTSPACE **** MXTSPACE ****
/* CALL OUT$LINE(9MSG?); */
**** MXTTRACE **** MXTTRACE **** MXTTRACE **** MXTTRACE **** MXTTRACE **** MXTTRACE ****
**** MXTTRACE **** MXTTRACE **** MXTTRACE **** MXTTRACE **** MXTTRACE **** MXTTRACE ****

PRI = 255;
DO /* SEARCH VPM FOR ELIGIBLE VIRTUAL PROCESSOR TO RUN */
I = PRDS.VP$START TO PRDS.VP$END;
IF /* THIS VP'S PRIORITY IS HIGHER THAN PRI */
((VPM(I).VP$PRIORITY <= PRI) AND
(VPM(I).STATE = READY)) THEN DO;
/* SELECT THIS VIRTUAL PROCESSOR */
PRI = VPM(I).VP$PRIORITY;
N = I;
END; /* IF */
RDYTHISVP: PROCEDURE REENTRANT PUBLIC:

**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
**** MXTRACE **** MXTRACE **** MXTRACE ****
** CALL OUT$LINE(OMSO):
**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
**** MXTRACE **** MXTRACE **** MXTRACE ****
PRDS.LAST$RUN = RRDIVP; /* SAVE THIS PROCESSOR INDEX */

**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
**** MXTRACE **** MXTRACE **** MXTRACE ****
** CALL OUT$LINE(OMSOA):
** CALL OUT$HEX(PRDS.LAST$RUN);
** CALL OUT$CHAR(CR);
** CALL OUT$CHAR(LF);
**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
**** MXTRACE **** MXTRACE **** MXTRACE ****
VPM(PRDS.LAST$RUN).STATE = READY;
RETURN;
END; /* RDYTHISVP PROCEDURE */

SAVECONTEXT: PROCEDURE (STACK$PTR, STACK$SEG) REENTRANT PUBLIC:

DECLARE (STACK$PTR, STACK$SEG) WORD;

IF PRDS.LAST$RUN <> 255 THEN DO; /* IF ENTRY IS NOT */
*/ FROM KORE START */
VPM(PRDS.LAST$RUN).SP$REG = STACK$PTR; /* SAVE STACK */
VPM(PRDS.LAST$RUN).SS$REG = STACK$SEG; /* STATE */
END;
END;

113
**AX**

**EX**

**ES**

**START**

**FL**

**END**

**INITIAL**

**0354**

**RETSVP**

**OUT$HEL**

**OUT$CHAR**

**RETSVP: PROCEDURE** BYTE REENTRANT PUBLIC;

**DECLARE RUNNING$VP$INDEX BYTE:**

**SEARCH THE VP MAP FOR RUNNING PROCESS INDEX */

**Returning process. It's index in VPM is returned. */

**CALLS MADE TO: OUT$HEL OUT$CHAR */

**RETURNS PROCEDURE ROWE 6-22-84 */

**USED BY THE SCHEDULER TO FIND OUT WHAT IS THE CURRENT */

**USING PROCESS:**

**DO **

**FOUND:**

**RETURN RUNNING$VP$INDEX;**

**END; /* RETSVVP PROCEDURE */

112
DECLARE IDLE$STACK STRUCTURE
   (LENGTH(030H)) WORD,
   RET$TYPE WORD,
   BP WORD,
   DI WORD,
   SI WORD,
   DS WORD,
   DX WORD,
   CX WORD,
   AX WORD,
   EX WORD,
   ES WORD,
   START POINTER, /* IP, CS */
   PL WORD) AT(IDLE$STACK$ABS)

INITIAL(
   ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   INT$RETURN,7AH,0,0,0,0,0,0,0,0@IDLE$PROC,200H );

DECLARE INIT$STACK STRUCTURE
   (LENGTH(030H)) WORD,
   RET$TYPE WORD,
   BP WORD,
   DI WORD,
   SI WORD,
   DS WORD,
   DX WORD,
   CX WORD,
   AX WORD,
   EX WORD,
   ES WORD,
   START POINTER, /* IP, CS */
   PL WORD) AT(INIT$STACK$ABS)

INITIAL(
   ,7AH,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   INT$RETURN,7AH,0,0,0,0,0,0,0,0@INITIAL$PROC,200H );

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/
/* DECLARE MONITOR STACK STRUCTURE
/* (LENGTH(030H)) WORD,
//** RET$TYPE WORD,
//** BP WORD,
//** DI WORD,
//** SI WORD,
//** DS WORD,
//** DX WORD,
//** CX WORD.
/* MSG4A(*) BYTE INITIAL (' SET VP TO READY: VP = \x' ),
/* MSG7(*) BYTE INITIAL ('ENTERING GATEWAY',13,10, \x'),
/* MSG7A(*) BYTE INITIAL (' SET VP TO RUNNING: VP = \x' ),
/* MSG7B(*) BYTE INITIAL (' SELECTED$DB = \x' ),
/* MSG10(*) BYTE INITIAL ('ENTERING IDLE$VP ',13,10, \x'),
/* MSG11(*) BYTE INITIAL ('UPDATE IDLE COUNT ',13,13, \x'),
/* MSG12(*) BYTE INITIAL ('ENTERING KERNEL$INIT',10,13, \x'),
/* MSG20(*) BYTE INITIAL ('ENTERING LOCATE$EV',10,13, \x'),
/* MSG22(*) BYTE INITIAL ('ENTERING LOCATE$SEQ',10,13, \x'),
/* MSG23(*) BYTE INITIAL (' FOUND',10,13, \x'),
/* MSG24(*) BYTE INITIAL (' NOT FOUND',10,13, \x');

/* DECLARE
/* CR LITERALLY 'ODH',
/* LF LITERALLY 'PAH';
/* OUTF$CHAR: PROCEDURE( CHAR ) EXTERNAL;
/* DECLARE CHAR BYTE;
/* END;
/* OUTF$LINE: PROCEDURE( LINESPTR ) EXTERNAL;
/* DECLARE LINESPTR POINTER;
/* END;
/* OUTF$NUM: PROCEDURE( NUM ) EXTERNAL;
/* DECLARE NUM BYTE;
/* END;
/* OUTF$DNUM: PROCEDURE( DNUM ) EXTERNAL;
/* DECLARE DNUM WORD;
/* END;
/* OUTF$HEX: PROCEDURE(B) EXTERNAL;
/* DECLARE B BYTE;
/* END;
/* INS$CHAR: PROCEDURE( RET$PTR ) EXTERNAL;
/* DECLARE RET$PTR POINTER;
/* END;
/* INS$DNUM: PROCEDURE( RET$PTR) EXTERNAL;
/* DECLARE RET$PTR POINTER;
/* END;
/* INS$NUM: PROCEDURE( RET$PTR) EXTERNAL;
/* DECLARE RET$PTR POINTER;
/* END;

**** MXT$ACE **** MXT$ACE ***** MXT$ACE **** MXT$ACE ****
**** MXT$ACE **** MXT$ACE ***** MXT$ACE ****

/0273************************************************************
/ * STACK DATA & INITIALIZATION FOR SYSTEM PROCESSES */
HDGSINT$FLAG (MAX$CPU) BYTE EXTERNAL,
HDGSVPS (MAX$CPU) BYTE EXTERNAL,
HRSRPS BYTE EXTERNAL,
GLOBAL$LOCK BYTE EXTERNAL.

DECLARE
EVENTS BYTE EXTERNAL,
EVCS$TBL (100) STRUCTURE
(EVCS$NAME BYTE, VALUE WORD, THREAD BYTE) EXTERNAL;

DECLARE
SEQUENCERS BYTE EXTERNAL,
SEQ$TABLE (100) STRUCTURE
(SEQ$NAME BYTE, SEQ$VALUE WORD) EXTERNAL;

/****0159************************************************************************/
/* Declaration of external procedure references */
/* The file and module where they are defined are */
/* listed. */

INITIAL$PROC: PROCEDURE EXTERNAL; END;
/** IN FILE: INITKE.SFC */
/** IN MODULE: INIT$MCD */

AWAIT: PROCEDURE (EVCS$ID, AWAIT$D$VALUE) EXTERNAL;
DECLARE EVCS$ID BYTE, AWAIT$D$VALUE WORD;
END;

VPSCHEDULE$: PROCEDURE EXTERNAL; END;
/** IN FILE: SCHED.ASM */

DECLARE INTVEC LABEL EXTERNAL;
/** IN FILE: SCHED.ASM */

DECLARE INTR$VECTOR POINTER AT ($1100) INITIAL (@INTVEC);
/** IN FILE: SCHED.ASM */

/****0192************************************************************************/
/* These diagnostic messages may eventually be removed. */
/* The utility procedures, however, are also used by the */
/* monitor process. They should not be removed. */

/**** MXTRACE ***** MXTRACF ***** MXTRACF ***** MXTRACF ***** MXTRACF *****
/**** MXTRACE ***** MXTRACF ***** MXTRACF ***** MXTRACF *****
/** DECLARE
/** MSG1(*) BYTE INITIAL ('ENTERING RETSVP ', 13, 10, 'x'),
/** MSG1A(*) BYTE INITIAL ('RUNNING$VP$INDEX = %'),
/** MSG4(*) BYTE INITIAL ('ENTERING EDYTHISVP', 13, 10, 'x').
 DECLARE CPU$NUMBER BYTE,
      VP$START BYTE,
      VP$END BYTE,
      VP$PRIORITY BYTE,
      VP$SP$SR$CPU BYTE,
      LAST$RUN BYTE,
      COUNTER WORD) PUBLIC INITIAL(0,0,0,0,0,0); 

 DECLARE VPM( MAX$CPU$NUMBER$MAX$VP$CPU ) STRUCTURE
 (VP$ID BYTE,
  STATE BYTE,
  VP$PRIORITY BYTE,
  EV$CSV$THREAD BYTE,
  EV$CSV$SW$VALUE WORD,
  SP$REG WORD,
  SS$REG WORD) EXTERNAL;

 DECLARE CPUSINIT BYTE EXTERNAL,
PRDS.VPSSEND = PRDS.VP$START + 1;
PRDS.VP$ & CPU = 2;
//******** MCORTEX ******* MCORTEX ******* MCORTEX ******* MCORTEX *******/
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/
// PRDS.VPSSEND = PRDS.VP$START + 2;
// PRDS.VP$CPU = 3;
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/

/* INITIALIZE THE VP MAP FOR IDLE AND INIT PROC */
/* AND MCOITC PROCESS */
VPM(PRDS.VP$START).VPSID = 255;
VPM(PRDS.VP$START).STATE = 1;
VPM(PRDS.VP$START).VPS PRIORITY = 0;
VPM(PRDS.VP$START).EVC$THRESH = 255;
VPM(PRDS.VP$START).EVC$AW$VALUE = 0;
VPM(PRDS.VP$START).SP$REG = 60;
VPM(PRDS.VP$START).SS$REG = INIT$STACK$SEG;
VPM(PRDS.VP$START+1).VPSID = 255;
VPM(PRDS.VP$START+1).STATE = 1;
VPM(PRDS.VP$START+1).VPS PRIORITY = 255;
VPM(PRDS.VP$START+1).EVC$THRESH = 255;
VPM(PRDS.VP$START+1).EVC$AW$VALUE = 0;
VPM(PRDS.VP$START+1).SP$REG = 60;
VPM(PRDS.VP$START+1).SS$REG = IDLE$STACK$SEG;

//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/
// VPM(PRDS.VP$START+2).VPSID = 0;
// VPM(PRDS.VP$START+2).STATE = 0;
// VPM(PRDS.VP$START+2).VPS PRIORITY = 0;
// VPM(PRDS.VP$START+2).EVC$THRESH = 255;
// VPM(PRDS.VP$START+2).EVC$AW$VALUE = 0;
// VPM(PRDS.VP$START+2).SP$REG = 60;
// VPM(PRDS.VP$START+2).SS$REG = M chó$STACK$SEG;
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/

PRP$ = PRP$ + 1;
//******** MCORTEX ******* MCORTEX ******* MCORTEX ******* MCORTEX *******/
//******** MCORTEX ******* MCORTEX ******* MCORTEX ******* MCORTEX *******/
// PRPS($)CPU$NUMBER) = 2;
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/
// PRPS($)CPU$NUMBER) = 3;
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/
//******** MTRACE ******* MTRACE ******* MTRACE ******* MTRACE *******/

HD$INT$FLAG (PRPS($)CPU$NUMBER) = 0;
ENABLE;

122
PDCS.LASTRUN = 255; /* INDICATE START ENTRY TO SCHEDULER */
CALL VPSCHEDULER; /* = - NO RETURN */

IND; /* LI$MODULE */

***************

***************

***************
/** McorTey **** McorTeX **** McorTEx **** McorTEx **** McorTEx ****/  

.F1: L0C096 L0R7.1NK ADDRRE5E5 (SEGMENTS &  
STACK (2C790H), &  
INITMOD_CODE (24862H), &  
GLOBAL MODULE DATA (7F7942H)) &  
SEGMENT (STACK (27EH)) &  
RESERVED TO 0 BAFFER)  
WARNING SE: SEGMENT IN RESERVED SPACE  
SEGMENT: (NO NAME)  
WARNING SE: SEGMENT IN RESERVED SPACE  
SEGMENT: INITMOD_CODE  

SYMBOL TABLE OF MODULE L1 MODULE  
READ FROM FILE KONE.1NK  
WRITTEN TO FILE: F2: KORSE  

<table>
<thead>
<tr>
<th>BASE</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>SYMBOL</th>
<th>BASE</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>006AH</td>
<td>007CH</td>
<td>PUB</td>
<td>PRDS</td>
<td>03B0H</td>
<td>0380H</td>
<td>PUB</td>
<td>IDLFPROC</td>
</tr>
<tr>
<td>0B6FH</td>
<td>0342F</td>
<td>PUB</td>
<td>LOCATESEQ</td>
<td>0B6FH</td>
<td>02B4F</td>
<td>PUB</td>
<td>LOCATEVEC</td>
</tr>
<tr>
<td>0B6FH</td>
<td>02B4F</td>
<td>PUB</td>
<td>GETWORK</td>
<td>0B6FH</td>
<td>01E3F</td>
<td>PUB</td>
<td>GETSP</td>
</tr>
<tr>
<td>0B6FH</td>
<td>01E3F</td>
<td>PUB</td>
<td>SAVECONTEXT</td>
<td>0B6FH</td>
<td>01F5F</td>
<td>PUB</td>
<td>ADDRHSVFP</td>
</tr>
<tr>
<td>0B6FH</td>
<td>01F5F</td>
<td>PUB</td>
<td>LTVP</td>
<td>0B6FH</td>
<td>068FF</td>
<td>PUB</td>
<td>CREATEPROC</td>
</tr>
<tr>
<td>0B6FH</td>
<td>068FF</td>
<td>PUB</td>
<td>TICKFT</td>
<td>0B6FH</td>
<td>06C7F</td>
<td>PUB</td>
<td>CREATESEQ</td>
</tr>
<tr>
<td>0B6FH</td>
<td>06C7F</td>
<td>PUB</td>
<td>PREEMPT</td>
<td>0B6FH</td>
<td>022BF</td>
<td>PUB</td>
<td>ADVANCE</td>
</tr>
<tr>
<td>0B6FH</td>
<td>022BF</td>
<td>PUB</td>
<td>AWAIT</td>
<td>0B6FH</td>
<td>0127F</td>
<td>PUB</td>
<td>READ</td>
</tr>
<tr>
<td>0B6FH</td>
<td>0127F</td>
<td>PUB</td>
<td>CREATEVEC</td>
<td>0B6FH</td>
<td>002EF</td>
<td>PUB</td>
<td>GATEKEEPER</td>
</tr>
<tr>
<td>0C6FH</td>
<td>0000H</td>
<td>PUB</td>
<td>VPSCHEDULE</td>
<td>0C6FH</td>
<td>0033H</td>
<td>PUB</td>
<td>INITVEC</td>
</tr>
<tr>
<td>043AH</td>
<td>0002H</td>
<td>PUB</td>
<td>INITIALPROC</td>
<td>0794H</td>
<td>0192H</td>
<td>PUB</td>
<td>VPM</td>
</tr>
<tr>
<td>0794H</td>
<td>0533H</td>
<td>PUB</td>
<td>SEOTABLE</td>
<td>0794H</td>
<td>052CH</td>
<td>PUB</td>
<td>SEQUENCERS</td>
</tr>
<tr>
<td>1794H</td>
<td>0561H</td>
<td>PUB</td>
<td>CRUNIT</td>
<td>0794H</td>
<td>0002H</td>
<td>PUB</td>
<td>EVCTEL</td>
</tr>
<tr>
<td>0794H</td>
<td>0561H</td>
<td>PUB</td>
<td>EVENTS</td>
<td>0794H</td>
<td>0566H</td>
<td>PUB</td>
<td>HDWINTFLAG</td>
</tr>
<tr>
<td>0794H</td>
<td>0566H</td>
<td>PUB</td>
<td>NAVPS</td>
<td>0794H</td>
<td>057EH</td>
<td>PUB</td>
<td>NAVPS</td>
</tr>
<tr>
<td>0794H</td>
<td>057EH</td>
<td>PUB</td>
<td>GLOBALLOCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

124
MEMORY MAP OF MODULE LIMODULE
READ FROM FILE KORE.LNK
WRITTEN TO FILE KORE

MODULE START ADDRESS PARAGRAPH = 0F820 OFFSET = 0F83H

SEGMENT MAP

<table>
<thead>
<tr>
<th>START</th>
<th>STOP</th>
<th>LENGTH</th>
<th>ALIGN</th>
<th>NAME</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0013H</td>
<td>0013H</td>
<td>0004H</td>
<td>A</td>
<td>INITMOD_CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>0439F</td>
<td>0439F</td>
<td>001AH</td>
<td>W</td>
<td>LIMODULE_CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>0683H</td>
<td>0683H</td>
<td>0354H</td>
<td>W</td>
<td>L2MODULE_CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>07E7H</td>
<td>07E7H</td>
<td>0007H</td>
<td>W</td>
<td>GLOBALMODULE_C</td>
<td>CODE</td>
</tr>
<tr>
<td>O6A0H</td>
<td>06A7H</td>
<td>0022H</td>
<td>A</td>
<td>LIMODULE_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>06A6H</td>
<td>06A6H</td>
<td>0077H</td>
<td>W</td>
<td>L2MODULE_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>06A1H</td>
<td>06A1H</td>
<td>0007H</td>
<td>W</td>
<td>INITMOD_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>06E0H</td>
<td>06E2H</td>
<td>0001H</td>
<td>W</td>
<td>??SEG</td>
<td>CODE</td>
</tr>
<tr>
<td>0CB7H</td>
<td>0C74H</td>
<td>0097H</td>
<td>G</td>
<td>SCHEDULER</td>
<td>CODE</td>
</tr>
<tr>
<td>0C76H</td>
<td>0C74H</td>
<td>0075H</td>
<td>W</td>
<td>STACK</td>
<td>STACK</td>
</tr>
<tr>
<td>0CB3H</td>
<td>0C74H</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>STACK</td>
</tr>
<tr>
<td>0C74H</td>
<td>0C74H</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>STACK</td>
</tr>
<tr>
<td>F7942H</td>
<td>F7FFH</td>
<td>063DF</td>
<td>W</td>
<td>GLOBALMODULE_D</td>
<td>DATA</td>
</tr>
<tr>
<td>F8002H</td>
<td>F8002H</td>
<td>0007H</td>
<td>W</td>
<td>MEMORY</td>
<td>MEMORY</td>
</tr>
</tbody>
</table>
**MYTRACEx** MYTRACEx MYTRACEx MYTRACEx MYTRACEx MYTRACEx

ISIS-II COS-66 LOCATER, V1.1 INVOKEd BY:  
:1:LOG36 KOFF.IEN ADDRFSSES(SFMMENTS( &  
STACK(OC4EFH),S  
INITMOD CODE 04390H), &  
GLOBALMODULE DAT(PE942H))&  
SEG SIZE(STACK,7EF))&  
RESERVE(8H TO 2ABFPH)  
WARNING 56: SEGMENT IN RESERVED SPACE  
SEGMENT: (NO NAME)  
WARNING 56: SEGMENT IN RESERVED SPACE  
SEGMENT: INITMOD_CODE

SYMBOL TABLE OF MODULE LIMODULE  
FILE FROM FILE KOFF.IEN  
WRITTEN TO FILE: P0:KORE  

<table>
<thead>
<tr>
<th>BASE</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C2EH</td>
<td>0074H</td>
<td>PUB</td>
<td>PUDS</td>
</tr>
<tr>
<td>0A7EH</td>
<td>046CH</td>
<td>PUB</td>
<td>IDLE.PROC</td>
</tr>
<tr>
<td>0A2EH</td>
<td>735EH</td>
<td>PUB</td>
<td>LOCATEFNC</td>
</tr>
<tr>
<td>0AC2H</td>
<td>0254H</td>
<td>PUB</td>
<td>GETSP</td>
</tr>
<tr>
<td>0AC2H</td>
<td>015EH</td>
<td>PUB</td>
<td>CREATESF</td>
</tr>
<tr>
<td>0B44H</td>
<td>0C0EH</td>
<td>PUB</td>
<td>OUTEX</td>
</tr>
<tr>
<td>0B44H</td>
<td>0A91H</td>
<td>PUB</td>
<td>SENDCHAR</td>
</tr>
<tr>
<td>0B44H</td>
<td>0C62H</td>
<td>PUB</td>
<td>OUTNUM</td>
</tr>
<tr>
<td>0B44H</td>
<td>0A11H</td>
<td>PUB</td>
<td>OUTNUM</td>
</tr>
<tr>
<td>0B44H</td>
<td>09A1H</td>
<td>PUB</td>
<td>OUTCHAR</td>
</tr>
<tr>
<td>0B44H</td>
<td>0B74H</td>
<td>PUB</td>
<td>INCHAR</td>
</tr>
<tr>
<td>0B44H</td>
<td>0796H</td>
<td>PUB</td>
<td>TICKET</td>
</tr>
<tr>
<td>0B44H</td>
<td>0479H</td>
<td>PUB</td>
<td>PSEEMPT</td>
</tr>
<tr>
<td>0B44H</td>
<td>0241H</td>
<td>PUB</td>
<td>WAIT</td>
</tr>
<tr>
<td>0B44H</td>
<td>0142H</td>
<td>PUB</td>
<td>CREATFNC</td>
</tr>
<tr>
<td>0C31H</td>
<td>0022H</td>
<td>PUB</td>
<td>VSCHEDULE</td>
</tr>
<tr>
<td>0C31H</td>
<td>0302H</td>
<td>PUB</td>
<td>INITIALPROC</td>
</tr>
<tr>
<td>0F94H</td>
<td>05C3H</td>
<td>PUB</td>
<td>SEQUEABLE</td>
</tr>
<tr>
<td>0F94H</td>
<td>05B1H</td>
<td>PUB</td>
<td>CPUINIT</td>
</tr>
<tr>
<td>0F94H</td>
<td>0594H</td>
<td>PUB</td>
<td>EVENTS</td>
</tr>
<tr>
<td>0F94H</td>
<td>057CH</td>
<td>PUB</td>
<td>NVPES</td>
</tr>
</tbody>
</table>

126
Memory Map of Module: LOADlNS

Start Address Paragraph = DACOH Offset = 0030H

<table>
<thead>
<tr>
<th>Start</th>
<th>Stop</th>
<th>Length</th>
<th>Align</th>
<th>Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011FH</td>
<td>0013H</td>
<td>0002H</td>
<td>A</td>
<td>(Absolute)</td>
<td>CODE</td>
</tr>
<tr>
<td>04390H</td>
<td>043A4H</td>
<td>0025H</td>
<td>W</td>
<td>INITMOD_CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>0AC00H</td>
<td>0AC4EH</td>
<td>0047H</td>
<td>W</td>
<td>LIMODULE_CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>0C04EH</td>
<td>000DH</td>
<td>0C32H</td>
<td>W</td>
<td>L2MODULE_CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>0C2D4H</td>
<td>0C2FCH</td>
<td>0074H</td>
<td>W</td>
<td>GLOBALMODULE_C</td>
<td>CODE</td>
</tr>
<tr>
<td>0C2F2H</td>
<td>0C31FH</td>
<td>0013H</td>
<td>W</td>
<td>INITMOD_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>0C31FH</td>
<td>0C35FH</td>
<td>0022H</td>
<td>G</td>
<td>??SEC</td>
<td></td>
</tr>
<tr>
<td>0C35FH</td>
<td>0C3B6H</td>
<td>0087H</td>
<td>G</td>
<td>SCHEDULER</td>
<td></td>
</tr>
<tr>
<td>0C45FH</td>
<td>0C474H</td>
<td>0075H</td>
<td>W</td>
<td>STACK</td>
<td>STACK</td>
</tr>
<tr>
<td>0C57FH</td>
<td>0C5B4H</td>
<td>007AH</td>
<td>A</td>
<td>(Absolute)</td>
<td></td>
</tr>
<tr>
<td>0C5B4H</td>
<td>0C5F9H</td>
<td>007AH</td>
<td>A</td>
<td>(Absolute)</td>
<td></td>
</tr>
<tr>
<td>0C622H</td>
<td>0C679H</td>
<td>007AH</td>
<td>A</td>
<td>(Absolute)</td>
<td></td>
</tr>
<tr>
<td>07842H</td>
<td>07FFEH</td>
<td>06BDH</td>
<td>W</td>
<td>GLOBALMODULE_D</td>
<td>DATA</td>
</tr>
<tr>
<td>0820F8H</td>
<td>08275E</td>
<td>007AH</td>
<td>W</td>
<td>MEMORY</td>
<td>MEMORY</td>
</tr>
</tbody>
</table>

127
APPENDIX I

SCHEDULER & INTERRUPT HANDLER SOURCE CODE

The ASM code in file: SCHED.ASM is part of LEVEL I. No special attributes are required for the assembler. This module is linked into file: KOR7.LNK, and its memory map is included in the map for KORE. KOR7 is the development system version of the file KORE.OPS loaded by KCOPTEX.COM under the CP/M-86 operating system.
; ** SCHEDULER ASM FILE *********
; * THE FOLLOWING ARE THE EXTERNAL PLM6 PROCEDURES CALLED * 
; * BY THIS MODULE. * 

EXTERN SAVECONTEXT: FAR
EXTERN GETSP: FAR
EXTERN GETWORK: FAR
EXTERN FYTHISVP: FAR
EXTERN PADSP: BYTE
EXTERN HDWINTFLAG: BYTE
EXTERN GLOBALLOCK: BYTE

SCHEDULER SEGMENT

PUBLIC VPSCHEDULER
PUBLIC INTVEC

VPSCHEDULER PROC FAR

ASSUME CS:SCHEDULER
ASSUME DS:NOTHING
ASSUME SS:NOTHING
ASSUME ES:NOTHING

; ENTRY POINT FOR A CALL TO SCHEDULER.

CLI
PUSH DS
MOV CX, 2H

;SWAP VIRTUAL PROCESSORS. THIS IS DONE BY SAVING THE
;STACK BASE POINTER AND THE RETURN TYPE FLAG ON THE
;STACK, AND BY SAVING THE STACK SEGMENT AND STACK
;POINTER IN THE VIRTUAL PROCESSOR MAP.

INTJOIN: PUSH BP
PUSH CX

MOV AX, SP
PUSH AX
PUSH SS
CALL SAVECONTEXT

CALL GETWORK
PUSH AX
CALL GETSP
POP SS
MOV SP, AX

;SWAP VIRTUAL PROCESSOR CONTEXT COMPLETE AT THIS POINT

129
; NOW OPERATING IN NEWLY SELECTED PROCESS STACK

POP CX                ; GET IRET INVALID FLAG
POP BP                ; INSTALL NEW STACK BASE

; CHECK FOR RETURN TYPE, NORMAL OR INTERRUPT

CMP CX, 77H
JZ INTRET

NORM_RET: POP DS
; UNLOCK GLOBAL$LOCK
MOV AX, SEG GLOBALLOCK
MOV ES, AX
MOV ES:GLOBALLOCK, ?

STI
IRET

; SCHEDULAR ENDP

; ***********************************************

; ***********************************************

; INTERRUPT_HANDLER PROC NEAR

; 

ASSUME CS: SCHEDULER
ASSUME DS: NOTHING
ASSUME SS: NOTHING
ASSUME FS: NOTHING

INTVEC: CLI
PUSH ES          ; SAVE NEEDED REGS TO TEST INTERRUPT FLAG
PUSH BX
PUSH AX
PUSH CX
CALL HARDWARE_INT_FLAG
MCV AL, 2
XCHG AL, ES:HTWINTFLAG[BX]
CMP AL, 77H       ; IS INT FLAG ON?
JZ PUSH_REST_REGS ; IF 'YES' SAVE REST REGS
PUSF CX           ; IF 'NOT' RESUME PREVIOUS
POP AX
POP BX
POP ES
STI
IRET
PUSH REST_REGS: PUSH DX ; FLAG WAS ON SO NEED
PUSH DS ; RESCHEDULE
PUSH SI
PUSH DI
MOV AX,SEG GLOBALLOCK
MOV ES, AX
CX: MOV AL,119 ; LOCK GLOBAL LOCK
LOCK XCHG ES:GLOBALLOCK,AL
TEST AL,AL
JNZ CX
CALL HYTHISVP
MOV CX,77F ; JUMP TO SCHEDULER
JMP INTJOIN
INTRET: POP DI ; RETURN FOR
PCP SI ; PROCESS WHICH
POP DS ; HAD PREVIOUSLY
POP DX ; BEEN INTERRUPTED
PCP CX ; UNLOCK GLOBAL LOCK
MOV AX,SEG GLOBALLOCK
MOV ES, AX
MOV ES:GLOBALLOCK,0
PCP AX
PCP BX
POP FS
STI
IRET
INTERRUPT_HANDLER ENDP

;********************************************************************

;********************************************************************
;*. HARDWARE INTERRUPT FLAG
;*. HAWDWARE_INT_FLAG PROC NEAR
ASSUME CS:SCHEDULER
ASSUME DS:NOTHING
ASSUME SS:NOTHING
ASSUME ES:NOTHING
HW_FLAG: MOV AX,SEG PRDS
MOV FS, AX
MOV ES, AX

131
MOV CL, ES:[HDS[+X]] ;GET CPU #
MOV CH, 0 ; RETURN IN PX
MOV BX, CX
MOV AX, SEG HDWINTFLAG ;SET UP HDWSINTFLAG
MOV ES, AX ; SEGMENT
RET ; RETURN IN ES .ESG

HARDWARE_INT_FLAG ENDP

SCHEDULE: ENDS

END
APPENDIX J

GLOBAL DATA BASE AND INITIAL PROCESS CODE

Two files are presented here: GLOBAL.SRC and INITK.SRC. They are both separately compiled with the LARGE attribute. They are linked into the file: KORE.LNK. They are represented in the memory map for KORE located at the end of Appendix E. INITK will be overwritten by the users initialization process.
FILE: GLOBAL.SEC
VERSION: REV 6-22-64
PROCEDURES
DEFINED: NONE

REMARKS: THIS MODULE CONTAINS DECLARATIONS FOR ALL THE
GLOBAL DATA THAT RESIDES IN SHARED COMMON
MEMORY. IT'S LOCATED THERE BY THE LOCATE COM-
MAND AND BY SPECIFYING THAT THE
GLOBAL MODULE DATA SEGMENT BE LOCATED AT SOME
ABSOLUTE ADDRESS.

GLOBAL MODULE: DO;

THE FOLLOWING THREE LITERAL DECLARATIONS ARE ALSO
GIVEN IN THE LEVEL1 & LEVEL2 MODULES OF THE OPERATING
SYSTEM. A CHANGE HERE WOULD HAVE TO BE REFLECTED IN
THOSE MODULES ALSO.

DECLARE
MAX$CPU LITERALLY '10';
MAX$VPS$CPU LITERALLY '10';
MAX$CPU$SS$MAX$VPS$CPU LITERALLY '100';

DECLARE
GLOBAL LOCK BYTE PUBLIC INITIAL(0);

THIS SHOULD REFLECT THE MAX$CPU ABOVE '*'
DECLARE
RES$EPS BYTE PUBLIC INITIAL(0),
RES$VPS(MAX$CPU) BYTE PUBLIC
INITIAL $0,$2,$3,$4,$5,$6,$7,$8;

DECLARE HP$INT$FLAG(MAX$CPU) BYTE PUBLIC;

DECLARE EVTS$BYTES BYTE PUBLIC INITIAL(1);

DECLARE EVC$STAR(100) STRUCTURE
(EVCS$NAME BYTE,
VALUE WORD,
THREAD BYTE) PUBLIC
INITIAL(0FEH,0,2E5);

/* EVC "FE" IS RESERVED FOR THE OS SYS */
DECLARE CPU$INIT BYTE PUBLIC INITIAL(2);

DECLARE SEQUENCERS BYTE PUBLIC INITIAL(2);

DECLARE STQ$TABLE(100) STRUCTURE
( STQ$NAME BYTE,
  STQ$VALUE WORD ) PUBLIC;

DECLARE VPM( MAX$CPUSS$MAX$VPSS$CPU ) STRUCTURE
( VP$ID BYTE,
  VP$STATE BYTE,
  VP$PRIORITY BYTE,
  EVCS$READ BYTE,
  EVCS$VALUE WORD,
  SP$=SG *W,C, *W,SIG WORD ) PUBLIC;

END: /* MODULE */

/*****************************************************************************/
/* INIT$MOD: PUBLIC PROCEDURE

DECLARE I BYTE;

/* AFTER INITIALIZATION THIS PROCESS BLOCKS ITSELF TO ALLOW THE NEWLY CREATED PROCESSES TO BE SCHEDULED.
/* THIS AREA SHOULD BE WRITTEN OVER BY USER INIT PROCEDURE.

CALL CUT$LINE(@MSG13);
CALL AVOID(@OFIH, 1);

END; /* INITIAL$PROC */
END; /* INIT$MOD */
LIST OF REFERENCES


<table>
<thead>
<tr>
<th>No. of copies</th>
<th>INITIAL DISTRIBUTION LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 2</td>
<td>Defense Technical Information Center</td>
</tr>
<tr>
<td></td>
<td>Cameron Station</td>
</tr>
<tr>
<td></td>
<td>Alexandria, Virginia 22341</td>
</tr>
<tr>
<td>2. 2</td>
<td>Library, Code 0142</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>3. 1</td>
<td>Department Chairman, Code 52</td>
</tr>
<tr>
<td></td>
<td>Department of Computer Science</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>4. 1</td>
<td>Dr. M. L. Cotton, Code 620c</td>
</tr>
<tr>
<td></td>
<td>Department of Electrical Engineering</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>5. 3</td>
<td>Dr. Uno R. Kodres, Code 52Kr</td>
</tr>
<tr>
<td></td>
<td>Department of Computer Science</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>6. 1</td>
<td>Commanding Officer VXF-6</td>
</tr>
<tr>
<td></td>
<td>ATTN: LT Willis R. Rowe</td>
</tr>
<tr>
<td></td>
<td>Point Mugu, California 93042</td>
</tr>
<tr>
<td>7. 1</td>
<td>Daniel Green, Code 20F</td>
</tr>
<tr>
<td></td>
<td>Naval Surface Weapons Center</td>
</tr>
<tr>
<td></td>
<td>Dahlgren, Virginia 22449</td>
</tr>
<tr>
<td>8. 1</td>
<td>CAPT J. Donegan, USN</td>
</tr>
<tr>
<td></td>
<td>PMS 400P5</td>
</tr>
<tr>
<td></td>
<td>Naval Sea Systems Command</td>
</tr>
<tr>
<td></td>
<td>Washington, D.C. 20362</td>
</tr>
<tr>
<td>9. 1</td>
<td>RCA AFGIS Data Repository</td>
</tr>
<tr>
<td></td>
<td>RCA Corporation</td>
</tr>
<tr>
<td></td>
<td>Government Systems Division</td>
</tr>
<tr>
<td></td>
<td>Mail Stop 127-327</td>
</tr>
<tr>
<td></td>
<td>Moorestown, New Jersey 08057</td>
</tr>
</tbody>
</table>
10. Library (code E33-05)
   Naval Surface Warfare Center
   Dahlgren, Virginia 22449

11. Dr. M. J. Gralia
    Applied Physics Laboratory
    Johns Hopkins Road
    Laurel, Maryland 20707

12. Dana Small
    Code B242, NOSC
    San Diego, California 92152
END

FILMED

4-85

DTIC