A REAL-TIME EXECUTIVE FOR MULTIPLE-COMPUTER CLUSTERS

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

D J BREWER

DEC 84
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A
THESIS

A REAL-TIME EXECUTIVE FOR MULTIPLE-COMPUTER CLUSTERS

by

David J. Brewer
December 1984

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited
This thesis extends the multi-computer real-time executive, MCORTEX, for a cluster of single board computers (INTEL iSBC 86/12 6601) on the MULTIBUS, to a multiple cluster system tied together by a Local Area Network (Ethernet). The E-MCORTEX system uses eventcounts and sequencers to synchronize processes resident in the network. Data communications between processes are presently limited to a single cluster with shared memory.
ABSTRACT (Continued)

However, future versions of E-MCORE will permit network-wide process synchronization and data communication.

Original supplied by JPL, included, appendix.
A Real-Time Executive for Multiple-Computer Clusters

by

David J. Brewer
Lieutenant, United States Navy
B.S., University of Idaho, 1978

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
from the
NAVAL POSTGRADUATE SCHOOL
December, 1984

Author:

David J. Brewer

Approved by:

Uno R. Kodres, Thesis Advisor

Mitchell L. Cotton, Second Reader

Bruce J. MacLennan, Chairman, Department of Computer Science

Kneale T. Marshall, Dean of Information and Policy Sciences
ABSTRACT

This thesis extends the multi-computer real-time executive, MCORTEX, for a cluster of single board computers (INTEL isBC 36/12) on the MULTIPUS, to a multiple cluster system tied together by a Local Area Network (Ethernet). The E-MCORTEX system uses eventcounts and sequencers to synchronize processes resident in the network. Data communications between processes are presently limited to a single cluster with shared memory. However, future versions of E-MCORTEX will permit network-wide process synchronization and data communication.
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
   MULTIBUS
   8086
   ISBEC 86/128
   ISIS-II
   PL/M-86
   PL/I-86
   DDT86

2. Digital Research, Pacific Grove, California
   CP/M-86
   LINK86
   ASM86

3. XEROX Corporation, Stamford, Connecticut
   Ethernet Local Area Network

4. InterLAN Corporation, Westford, Massachusetts
   N13010 Ethernet Communication Controller Board
# TABLE OF CONTENTS

I. INTRODUCTION ........................................... 11
   A. DISCUSSION ........................................... 11
   B. BACKGROUND .......................................... 14
   C. STRUCTURE OF THE THESIS .......................... 16

II. THE "EVENTCOUNT" MODEL ............................... 18
   A. A Model of Synchronization .......................... 12
   B. Model Variables ...................................... 19
   C. A Distributed Synchronization Model ............... 25

III. SYSTEM ARCHITECTURE ................................ 28
   A. HARDWARE REQUIREMENTS .............................. 28
      1. System Configuration .............................. 28
      2. The iSBC 86/12A Single Board Computer .......... 32
      3. The 8086 Microprocessor .......................... 33
      4. Ethernet ........................................... 35
      5. The NI3010 Ethernet Communication Controller Board ........................................... 38
   B. SOFTWARE SERVICES .................................. 42

IV. DETAILED SYSTEM DESIGN ............................... 51
   A. DESIGN ISSUES ....................................... 51
      1. Real-Time Processing .............................. 51
      2. Shared Resources ................................... 51
      3. Process Integrity ................................... 54
      4. Process Synchronization ............................ 55
      5. Interprocess Communication ........................ 58
LIST OF REFERENCES . . . . . . . . . . . . . . . . . . . . . . . 222
INITIAL DISTRIBUTION LIST . . . . . . . . . . . . . . . . . . . 224
LIST OF FIGURES

1. Producer-Consumer Process Synchronization .......... 22
2. Multiple-Producers/Single-Consumer Relationship ........... 26
3. Cluster Hardware Configuration ......................... 29
4. Real-Time Cluster STAR Architecture ..................... 31
5. Ethernet Packet Format ................................ 46
6. Cluster Memory Map ...................................... 48
7. Intercluster Input/Output Processing ...................... 60
8. Ethernet Request Packet Format ......................... 61
9. Eventcount Type Ethernet Packet ......................... 64
10. Ethernet Packet Routing ................................. 66
11. Demonstration Model ..................................... 112
LIST OF TABLES

1. GLOBAL MEMORY ............... 53
2. MAP FILE ................... 74
I. INTRODUCTION

A. DISCUSSION

1. General

The purpose of this thesis is to extend the existing version of MCORTEX to a distributed multi-computer real-time executive which transcends the boundaries of a length-limited parallel system's bus, the MULTIBUS. This extension is provided by a local area network (LAN) medium, the Ethernet, and the additional operating system primitives.

As the anti-air warfare (AAW) system of the 1980's for the U.S. Navy, the AEGIS Weapon System captured the attention of a project group at the Naval Postgraduate School (NPS). The project group was formed to look at the AN/SPY-1A phased array radar processing unit. This unit was selected due to the time critical nature of the processing requirements, i.e., the fast reaction to inbound hostile air contacts (missile and aircraft). The AEGIS Modeling Group has been working on the VLSI architecture and the MCORTEX real-time executive for several years.

The fundamental objective is to utilize commercially available LSI and VLSI components that can be implemented in a modular form within the AEGIS Weapons System. Subsequent low cost is a desirable effect, but the
must be mutually exclusive, but it is highly undesirable to place an a priori sequence constraint on several producers. Each producer obtains a ticket number from sequencer S for depositing its message in the buffer. Once a process obtains a ticket, it merely waits for the completion of all producers that obtained prior tickets. Each producer executes the code illustrated in Figure 2 and each consumer executes the same code shown in Figure 1.

The producers block in the following circumstances:

(1) Another producer has a lower ticket value and as yet has not deposited his message.

(2) The single consumer is unable to keep up with the messages deposited in the buffer.

C. A DISTRIBUTED SYNCHRONIZATION MODEL

1. Asynchronous Eventcounts

In distributed clusters without shared memory, a change to an eventcount (via advance) takes time to propagate down communication lines to other systems. Two major options exist: (1) implement all eventcounts so that a given eventcount exists only in the cluster where it is most frequently accessed. All other clusters which need the value must make remote accesses to the value. (2) distribute the eventcount values, so that each cluster maintains a local copy. The latter option is that selected in this extension of MCCRTX.
In synchronization problems that require exclusive use of a resource, the eventcount alone is inadequate. Two or more processes desiring to use a shared resource or write to a shared buffer location are natural examples. Another kind of an object, known as a "sequencer" can be used to provide the needed total ordering. A sequencer is considered a natural number generator, i.e., it returns the sequence 0,1,2,..., etc. Only one operation exists on a sequencer - ticket. When applied to a sequencer S, ticket(S) returns a non-negative integer value as its result. The ticket primitive is based on the idea of the first-come first-served principle used in everyday life. A ticket machine in a catalog sales store or shoe store is an example. The ticket machine issues successive integer values on the ticket, and the next customer to be served is based on the number on the ticket. The store clerk can determine the next person to be served by merely adding one to the previously served number. The customers are served in first-come first-served order. If a customer with the next ticket number has walked out of the store when his number is called, he loses his turn and must get another ticket. This service policy is usually implemented in both stores and in computer operations by a watchdog timer.

The use of sequencers implies mutual exclusion not present in eventcounts. In the case of multiple producers, in a producer-consumer relationship, all message deposits
(2) the producer does not store the \((i + N)\)th value into the buffer until the \(i\)-th value has been read by the consumer.

It is important to note that in the above producer-consumer example, each eventcount has only one writer. In the usual semaphore solution both processes would modify the same synchronization variable. For example, let \(P(S)\) represent the synchronizing primitive where processes wait for \(S\) (some resource) to become greater than zero and then subtract 1 from \(S\) before proceeding. Further, let \(V(S)\) represent the synchronizing primitive where the processes add 1 to \(S\) before proceeding. With this type of synchronization the consumption and production of a result or resource requires that all processes read or write \(S\). A reduction in write competition often occurs in eventcount solutions, resulting in simplified correctness proofs and simplifying the synchronization of physically distributed processes.

The power of eventcounts rest in their ability to achieve synchronization through a relative ordering of events, rather than by mutual exclusion. In the previous example, concurrency of execution is guaranteed if the producer starts out several steps ahead of the consumer and the speeds of production and consumption are equal. In that case there does not exist a time when the consumer or producer must wait for the other to complete an operation.
Figure 1  Producer-Consumer Process Synchronization
Frequently a process may not wish to continue executing unless an event, in a class in which it has interest, has occurred. A 'busy wait' could be implemented easily by looping around an execution of a read(E) primitive until a specified value of the eventcount, E, is reached. The implication of wasted CPU cycle time is evident and in many instances could be avoided. A process can voluntarily block itself with an await(E, v) primitive call. The calling process will remain suspended (i.e., not ready for execution) until the value of E is at least v. Processes written in PL/I - like pseudo-code, as illustrated in Figure 1, demonstrates the use of the advance and await primitives. The producer and consumer process must synchronize their use of a shared N-cell circul buffer. The circular buffer is implemented as an array in shared memory with indices from 0 to N-1. Two eventcounts MESSAGE_IN and MESSAGE_OUT are used to synchronize the producer and consumer. The producer generates a series of messages by calls on a function "receive_message" and stores the i-th iteration in message_buffer((i-1) mod N). The consumer reads these values out of the buffer in order and consumes them by calling a "xmit_message" subroutine and advancing eventcount MESSAGE_OUT.

The two eventcounts, MESSAGE_IN and MESSAGE_OUT, coordinate the use of the buffer so that:

(1) the consumer does not read the i-th message from the buffer until it has been stored by the producer, and
synchronization variable, known as a "sequencer," is needed.

An eventcount is primarily a count of the number of events of a particular class that have occurred in the past. It can be considered a non-negative integer variable whose value never decreases. This is reasonable, since events cannot 'unhappen.'

2. Model Primitives

To signal the occurrence of events, an advance primitive is used. Two primitives, *await* and *read*, are used to obtain values of eventcounts. A primitive operation *advance*(E) signals the occurrence of an event in the class associated with the eventcount E. This operation increases the integer value of E by 1. The value of the eventcount equals the number of advance operations performed on it. The initial value of an eventcount is zero.

A process can observe the value of an eventcount in one of two ways. The value may be read directly using the primitive *read*(E), or the process can block itself until the eventcount reaches a specific value v using the *await*(E,v) primitive. The value returned by *read*(E) counts all of the advance operations that precede the execution, and may or may not count those in progress during the read. The result of *read*(E) is, therefore, a lower bound on the current value of E after the *read*, and an upper bound on the value of E before the *read*.
normally unnecessary for a process to know the names or residences of other processes.

This model makes no assumptions about the environmental properties of systems and consequently is directly applicable to distributed systems. A distributed system is defined as a system which, due to the lack of a common memory, requires communication among processes to be via communication channels involving unpredictable time delays.

B. MODEL VARIABLES

1. Eventcounts and Sequencers

Unlike the semaphore model, the MCOKE model solves the synchronization problem in terms of timing constraints on occurrences of events, instead of mutual exclusion. Events are divided into event classes and events of a given class are represented by an associated synchronization variable of the type 'eventcount.' Primitive operations exist that permit processes to signal and observe occurrences of events.

The eventcount alone is inadequate in certain types of timing constraints problems. This type of synchronization problem has the characteristic that the order of different activities is not specified in advance. Instead the synchronization system dynamically defines a total order among them. To deal with this type of constraint a
II. THE EVENTCOUNT MODEL

A. A MODEL OF SYNCHRONIZATION

A computer system that manages resources used by concurrently operating, independent users requires a mechanism that allows processes to synchronize the use of shared resources.

The most common existing models of synchronization are based upon the principle of mutual exclusion and shared data to achieve synchronization. Semaphores [Ref. 7] and monitors [Ref. 8] are based on the concept of mutual exclusion. In this context mutual exclusion is a mechanism that forces the time ordering of execution of pieces of code, called critical sections.

The characteristics of the semaphore and monitor synchronization models have undesirable effects. These effects include complex proofs for program correctness and limitations on applicability to distributed systems.

The model upon which MCORTEX is based is an event oriented model of synchronization in which processes coordinate their activities by signalling and observing events via synchronization variables, known as "eventcounts" and "sequencers." These synchronization variables are interfaces for all interaction among processes. It is
Chapter II presents design concepts and criteria for the original MCORTEX model and the distribution model upon which the extension to MCORTEX is based.

Chapter III is a presentation of the system architecture, with primary emphasis on hardware components.

Chapter IV details the system design of MCORTEX, including the method by which user processes gain access to Ethernet services.

Chapter V is a thorough presentation of the development of user processes and the modifications to the MCORTEX loader.

Chapter VI is a summary of the current state of the system, with particular emphasis on future enhancements and scheduled modifications.
the control of MCORTEX. He also developed access mechanisms to the MCORTEX supervisor compatible with Digital Research's PL/I-66 language system. User programs could then be developed in a high level, portable language. The kernels of MCORTEX, system processes, and user processes could then be loaded into single board processors from the CP/M-86 environment. Just as importantly, access to the disk sharing capabilities of the multi-user CP/M-86 system, via MCORTEX processes, was made possible. Rowe's efforts were a culmination of the planned synergism of the individual research projects.

C. STRUCTURE OF THE THESIS

The goals of this thesis are to:

1. Extend the existing MCORTEX real-time executive for a single cluster of single board computers with shared memory to a real-time executive for a multiple cluster system without shared memory.

2. Extend the existing MCORTEX without introducing substantial changes either to the MCORTEX executive or its primitives.

3. Use the Ethernet interface between clusters to communicate systems data.

Chapter I discusses the overall intent of the AEGIS Weapons System Simulation Project and the emphasis area this thesis covers in accomplishing project goals.
of an operating system tailored to real-time image processing. His design used the MULTICS concept of segmentation and per process stacks and Reed and Kanodia's [Ref. 2] eventcount synchronization methods. Rawantzikos [Ref. 3] began the initial implementation of Wasson's efforts. At this point, MCORTEX used the concept of a "two level traffic controller" to effect processor multiplexing among eligible processes.

Cox [Ref. 4] simplified the design of MCORTEX. He reduced the traffic controller to one level of abstraction, favoring reduced MCORTEX execution overhead over the security of the two level traffic controller. Cox's other contribution was the addition of a "gatekeeper" module to the entry to the operating system, so the user's access to system calls was simplified. Klinefelter [Ref. 5] generalized Cox's work and developed a technique to dynamically interact with the operating system during its execution.

During the early stages of development of MCORTEX concurrent research efforts, within the AEGIS Modeling Group, were producing a multi-user CP/M-86 based disk sharing environment. It was envisioned this system would be used to develop software in support of the SPY-1A processing emulation.

Howe [Ref. 6] brought the powerful, highly portable functions of the multi-user CP/M-86 operating system under
The distinction must be made between user processes and system processes. MCORTEX is the executive which provides primitives to allow processes to synchronize and communicate asynchronously. The only system process invoked by MCORTEX is the device-dependent Ethernet Communication Controller Board (ECC3) handler and packet interpreter. This system process is resident within one SBC at each cluster. As a consumer of Ethernet Request Packets (EPP), produced by each kernel, this virtual processor does not compete against other processes for a time quantum. It is through the EPP's that user processes make known their need to transmit information over Ethernet. It is transparent to the user processes, however, that an EPP is generated; MCORTEX takes care of this detail. The ECCB handler and packet interpreter is scheduled under MCORTEX and never surrenders the CPU. When it does not have any Ethernet Request Packets to consume, it idles in a "Busy Wait" loop. It is anticipated that its wait will be minimal. User processes are those which are independent of cluster hardware, generally cyclic in nature, and provide a function in support of the Aegis Weapon System Simulation and Modeling effort.

B. BACKGROUND

The initial design of MCORTEX was completed in 1980. The implementation for the ISPC 66/12 single board processors was completed in three Naval Postgraduate School theses in 1981 and 1982. Wasson [Ref. 1] defined the detailed design
Derformarce is realized by expanding MCORTEX to allow multiple kernels to schedule processes that synchronize and communicate via an intercluster bus (Ethernet). The benefits of interconnecting processing nodes to facilitate information exchange and resource sharing has long been recognized. Those recognized benefits are being applied in the development of extended MCORTEX. The collection of available clusters and the high speed interconnect is collectively known as RTC* (Real-time Cluster Star). As will be seen, the Kleene closure connotes the true power and extensibility of MCORTEX.

The locality of processing modules in a real-time environment is tantamount to speed and efficiency. By effectively co-locating real-time sensors and related processing modules, real-time data acquisition and processing is assured. The use of the Ethernet medium allows the extension of needed process synchronization and interprocess communications to processing nodes which cannot be located physically close enough for shared memory.

As a fully distributed real-time executive, MCORTEX consists of single board resident kernels which support multiprocessing. Process synchronization between virtual processes in the same cluster or in different clusters is provided, entirely transparent to user processes, through integrated cluster hardware and kernel primitives.
proposed replacement of the current four-tay AN/UYK-7 computers, of the AN/SPY-1A phased array suite, is not solely cost-based. Reliability and functional redundancy in the event of failure are extremely important criteria. Mean time to repair (MTTR) is a crucial issue for deployed units (ship or aircraft), due primarily to the unavoidable disrupted Sea Lanes of Communication (SLOC). An onboard technician could discard a failed component and replace a low cost LSI device, such as a microprocessor, from an onboard supply bin.

The project team has produced (up to and including this thesis) a highly modular hardware base, integrated with an equally modular and highly extensible software base. The use of Ethernet as the highest level bus has introduced another commercial-grade product into an existing system of commercial-grade products. As an established standard in the marketplace, the low cost, availability, and support of Ethernet is virtually guaranteed for years to come.

2. Specific

We define a cluster as a group of single board computers (SBC), controlled by multiple kernels of MCORTEX, sharing a common backplane. The integration of the kernels with a general purpose commercially available operating system (CF/M-86) collectively provides multiprogramming capability, multiprocessing capability, and standard disk operating system (DOS) functions. Increased cost-
producer: procedure;

do while (FOREVER);
  t = ticket(S);
  /* producers synchronize */
  call await(MESSAGE_IN, t);
  /* at this point in execution it's this
     processes' turn, but now must
     synchronize with the consumer */
  j = read(MESSAGE_IN);
  k = read(MESSAGE_OUT);
  if (j - k >= N) then
    call await(MESSAGE_OUT, k + 1);
    /* if buffer is full then block */
    message_buffer(t MOD N)=receive_message();
  end; /* do while */
end; /* procedure */

Figure 2  Multiple-Producers/Single-Consumer Relationship
We define a local eventcount to be an eventcount which is not accessed outside the cluster. We define a remote eventcount as one which is accessed in at least two clusters. A producer at a cluster simply advances an eventcount. If the eventcount is a remote eventcount, the operating system generates the necessary commands which advance a local copy as well as the remote copies of this eventcount. The distributivity of the eventcount is entirely transparent to the producer and consumer. Only the operating system knows in which cluster the producers and consumers reside during their lifetime.

By transmitting the eventcount value, the robustness of the system is assured. Even if the transmission message is lost or not properly received, the very next advance and a subsequent successful transmission will bring the remote copy up to its correct value. The non-decreasing nature of the eventcount value accounts for this robustness.

The design modifications to MCORTEX to allow the eventcount values to be distributed are fully presented and discussed in Chapter 4.
III. SYSTEM ARCHITECTURE

A. HARDWARE REQUIREMENTS

1. System Configuration

A cluster of Real-Time Cluster Star (RTC®), as shown in Figure 3, is based on the INTEL iSBC 86/12A single board computer (SBC) with MULTIBUS serving as the intracluster bus. Figure 4 illustrates two clusters connected by the Ethernet LAN medium, which serves as the intercluster bus.

Although only four SBC's are shown at each cluster, the limitation is entirely dependent on the number of bus masters. A bus master can drive the command and address lines: it can control the bus. Since multiple bus masters exist in this configuration, some means must be available in hardware to arbitrate their simultaneous requests to use the MULTIBUS. A customized random priority bus resolver, designed specifically for this system, serves a maximum of eight bus masters. A bus slave, such as a RAM board, cannot control the bus and does not require arbitration circuitry.

Two shared memory boards also share the MULTIBUS. A 32K RAM extension board is used as shared memory for process synchronization and control under MCORTEX and for CP/M-86 multi-user system control. A 64K RAM extension board provides additional shared memory required for user
FIGURE 3 Cluster Hardware Configuration
process data communications. Two hard disk systems are available for application process use within a cluster. The EEMEX hard disk system has a disk controller card which is placed in an odd slot (required for a bus master) in the MULTIBUS backplane.

The InterLAN NI3010 Ethernet Communications Controller is a MULTIBUS-based single board processor which along with a transceiver provides the cluster with a complete connection to an Ethernet medium. This is the hardware extension to the cluster which allows MCORTEX to be distributed over the Ethernet.

Although only two clusters are shown in Figure 4, the Ethernet specification [Ref. 9] allows for a maximum of 1024 nodes. However, the limiting factor in MCORTEX is the number of clusters that can be addressed with the current packet routing algorithm. As will be discussed in Chapter 4, the upper bound is 16 clusters which is more than adequate considering the current availability of only two NI3010 boards and three NI3210 boards (enhanced version of the NI3010) in the ARGIS Simulation Laboratory at the U.S. Naval Postgraduate School, Monterey, California.

An experimental system that contains both analogous and dissimilar components to that of RTC* is CM* [Ref. 10]. The most important comparison is between CM*'s Kmap and RTC*'s NI3010 and Driver. The Kmap must effectively route a
FIGURE 4 Real-Time Cluster STAR Architecture
shared address between clusters, whereas the NI3210 Driver routes an entire datagram of information. The intercluster response time of Kmap is on the order of 36 microseconds, while the Ethernet is on the order of milliseconds. Therefore, the use of Ethernet is appropriate where relatively long messages with not very demanding response times are used. The Kmap has a relatively low transfer rate with fast response times. Additionally, the cost of the "NI3210 and the Driver development, the flexibility, and its extensibility is far superior to the Kmap. The NI3210 is expected to further increase the speed and efficiency of intercluster communications.

2. The iSBC 86/12A Single Board Computer

The iSBC 86/12A board includes a 16-bit CPU, 64K bytes of dynamic RAM, a serial communications interface, three 8-bit programmable parallel I/O ports, programmable timers, priority interrupt control, MULTIBUS interface control logic, and bus expansion drivers for interface with other MULTIBUS interface-compatible expansion boards. The iSBC 86/12A board has an internal bus for all onboard memory and I/O operations and accesses MULTIBUS for all external memory and I/O operations. Therefore, local (onboard) operations do not disturb the MULTIBUS interface available for parallel processing when several bus masters (e.g., DMA devices and other SBC's) are operating concurrently.
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 32K 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 MULTIPUS. The presently used wiring option prohibits off-board access to local RAM, so that the local RAM is protected from external contamination.

3. The 8086 Microprocessor

The 8086 microprocessor, the heart of the single board computer, performs the system processing functions and generates the address and control signals to access memory and I/O devices.

This high-performance, general-purpose microprocessor base of the iSBC86/12A 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. A 16-bit arithmetic/logic unit (ALU) in the EU maintains the CPU status and control flags, and manipulates the general registers and instruction operands. All registers and data
paths in the FU are 16 bits wide for fast internal transfers.

The 8086 has eight 16 bit general purpose registers. Four byte addressable registers, known as the data registers, can be used without constraint in most arithmetic and logic operations. 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.

The 8086 can address up to one megabyte of memory, 'viewed' as a group of segments, as defined by the application. A segment is a logical unit of memory that may be up to 64K bytes long. The segment registers point to the four currently addressable segments. Programs obtain access to code and data in other segments by changing the segment registers to point to the desired segments.

It is convenient to think of every memory location as having two kinds of addresses, physical and logical. A physical address is a 20-bit value that uniquely identifies each byte location in the megabyte address space. Physical addresses range from 0H through FFFFFFH. Programs, however, deal with logical instead of physical addresses. A logical address consists of a base value and an offset value. Whenever the FU accesses memory - to fetch an instruction or to obtain or store a variable - it generates
a physical address from the logical one. This is accomplished by shifting the base value left four bits and adding the offset. The resultant 20 bit value is then used to access memory.

4. Ethernet

Ethernet is a local area network (LAN) optimized for the high-speed exchange of data between information processing equipment within a moderate-sized geographic area. It is the result of a collaborative effort by Digital Equipment Corporation, Xerox Corporation, and Intel Corporation. The Ethernet specification [Ref. 9] provides precise, detailed design information for a baseband local area network and, for brevity's sake, only general aspects pertaining to the RTC* implementation will be discussed here.

Ethernet implements the lowest two layers of the 7-layer OSI/ISO model [Ref. 11 pp. 46-53]. The Data Link layer defines the format and addressing of packets that are broadcast over the "Ether", detects transmission errors, controls access of the network by nodes, and allocates channel capacity. These functions are, in fact, implemented in the VI3010 Ethernet to MULTIPUS communications controller board. The functions carried out by this layer for sending and receiving transmissions are as follows.

a. Data Encapsulation/Decapsulation

Defining the format of message packets - the different fields of information within the packets.
Constructing packets from data supplied by the nodes through the higher layers; disassembling network messages and supplying data to the higher layer protocols of the node.

Addressing - handling of source and destination addresses.

Error detection - physical channel transmission errors.

b. Link Management

Channel allocation - the length of time of channel use is determined by the packet size.

Channel access - access to the channel is controlled by a contention-avoidance-and-resolution technique, called CSMA/CD, part of which is carried out in each of the two layers. The Data Link level responds to the channel or carrier sensing of the Physical layer. This means that the sender defers sending in the case of traffic, sends in the absence of traffic, and backs off and resends the message a random time interval later in the case of collisions.

The construction and processing of the packets that are transmitted on the Ethernet, is part of the data encapsulation function of the Data Link layer. The Ethernet packet is made up of five fields, as shown in Figure 5 (all bytes are eight bits in length). The smallest total size of a packet transmitted over Ethernet is 64 bytes, and the maximum size of a packet is 1,518 bytes (these figures do not include the eight-byte preamble). Details of the fields are included in [Ref. 9], so the only field discussed in detail will be the destination address. Knowledge of this field will simplify the discussion of the packet routing algorithm presented in Chapter 4.
A packet can be sent to one, several, or all nodes simultaneously, through unique broadcasting and addressing capabilities. The address of the node (or nodes) that the packet is intended for is placed in this field, which is six bytes in length. A node address can be one of two types:

**Physical address** – the unique address of a single node on any Ethernet.

**Multicast address** – a multidestination address of one or more given nodes on a given Ethernet, of which there are two kinds:

- **multicast group address** – virtually any number of node groups can be assigned a group address so they are all able to receive the same packet in a single transmission by a sending node. This is a key feature in the packet routing algorithm to be discussed in Chapter 4.

- **broadcast address** – a single multicast address by which a packet can be sent to the set of all nodes on a given Ethernet.

The first bit in the Destination Address field is set to indicate a physical or multicast address. The remaining 47 bits specify the address itself. If a packet is to be broadcast to all nodes, the 47 bits are all set to "1." The 47 remaining bits allow for \(2^{47}\) (over 147 trillion) possible addresses.

The **Physical Layer** of Ethernet provides a ten-million-bit-per-second channel over a coaxial cable medium. It specifies all the essential physical characteristics of Ethernet, including bit encoding, timing, voltage levels, and two compatibility interfaces.
The main functions of this layer are:

**Data encoding/decoding:**

- Generation and removal of 64 preamble bits before each packet is transmitted for synchronization and timing of messages.
- Bit encoding and decoding - between the binary encoded form of the Data Link level and the phase encoded form required for transmission on the coaxial cable.

  **Manchester phase encoding** is specified for all data transmitted on the Ethernet at a data transmission rate of ten million bits per second (10 Mbps).

**Channel Access**

- Transmission and reception of encoded data.
- Carrier sense - monitoring the channel for traffic and signaling the Data Link layer if traffic is detected.
- Collision detect - signaling the Data Link layer, during transmission, when a collision is detected.

Two important compatibility interfaces, the transceiver cable interface and the coaxial cable interface, are also specified in the Physical layer. Detailed information regarding these interfaces is contained in [Ref. 3].

5. **NI3010 Ethernet Communication Controller Board**

In the following discussion of the NI3010's operation, reference to a "host" is synonymous with a single board computer in a cluster which contains the device driver for the NI3010 board. Details concerning this driver's system role are contained in Chapter 4.

The NI3010 ECCEP is a MULTIBUS-compatible component that implements layers one and two of the ISO/OSI 7-layer model. Although programmable as a polled or interrupt-driven
DMA device, it is used entirely as an interrupt-driven component in this implementation. The NI3010 serves as a bus master when controlling the DMA operations between the NI3310 buffers and the host's memory, and as a slave to the commands of the host.

The host controls the NI3010 by writing to onboard registers which are MULTIRUS addressable I/O ports. Depending on the state of execution, the host may direct the NI3010:

1. To perform a load command
2. In preparation for a DMA operation - load a memory address and a byte count, or
3. To enable an interrupt register, to inform the host when a directed operation is complete.

The host programs the NI3010 by writing a command to the command register, whose I/O address is currently set at B0H (base register). The command function codes are contained in Table 3-1 of [Ref. 12]. After issuing a command, the host must check for a value in the Command Status Register. The details of this read operation are covered in [Ref. 12], but briefly: Any value other than zero or one in the Command Status Register, following execution, represents a board failure. If at any time during MCUSTEX execution a diagnostic appears that indicates an NI3010 board failure, the FTC* system operator can run a diagnostic program that fully exercises the board. The code and
TABLE 1 - GLOBAL MEMORY

<table>
<thead>
<tr>
<th>Offset</th>
<th>Mnemonic</th>
<th>Type</th>
<th>Init</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LOCAL$CLUSTER$ADDR</td>
<td>W</td>
<td>X</td>
<td>Address of this cluster</td>
</tr>
<tr>
<td></td>
<td>EVENTBL(128)</td>
<td>S</td>
<td></td>
<td>Event count table</td>
</tr>
<tr>
<td>2</td>
<td>VVCSNAMP</td>
<td>P</td>
<td>P</td>
<td>Event count name</td>
</tr>
<tr>
<td>3</td>
<td>VALUE</td>
<td>W</td>
<td>0</td>
<td>Event count value</td>
</tr>
<tr>
<td>5</td>
<td>REMOTE ADDR</td>
<td>W</td>
<td>FF</td>
<td>Remote addr of remote copy</td>
</tr>
<tr>
<td>7</td>
<td>THREAD</td>
<td>P</td>
<td>FF</td>
<td>Event count thread</td>
</tr>
<tr>
<td></td>
<td>VBM(MAX$CPU * MAX$VPMS$CPU)</td>
<td>S</td>
<td></td>
<td>Virtual processor mat</td>
</tr>
<tr>
<td></td>
<td>(MAX$CPU = 10, MAX$VPMS$CPU = 1W)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>602</td>
<td>VP$ID</td>
<td>P</td>
<td>Y</td>
<td>Virtual processor ID</td>
</tr>
<tr>
<td>603</td>
<td>VP$STATE</td>
<td>R</td>
<td>X</td>
<td>Virtual processor state</td>
</tr>
<tr>
<td>604</td>
<td>VP$PPICITY</td>
<td>S</td>
<td>X</td>
<td>Virtual processor pri.</td>
</tr>
<tr>
<td>605</td>
<td>PVCSAWS$VALUE</td>
<td>W</td>
<td>Y</td>
<td>Count awaited</td>
</tr>
<tr>
<td>606</td>
<td>SP$REG</td>
<td>W</td>
<td>X</td>
<td>Stack pointer register</td>
</tr>
<tr>
<td>607</td>
<td>SS$REG</td>
<td>W</td>
<td>Y</td>
<td>Stack segment register</td>
</tr>
<tr>
<td>1602</td>
<td>GLOBSLCSK</td>
<td>P</td>
<td>0</td>
<td># of real processors</td>
</tr>
<tr>
<td>1603</td>
<td>NPS$PSS</td>
<td>B</td>
<td>0</td>
<td># of virtual processors</td>
</tr>
<tr>
<td></td>
<td>(one byte for each possible CPU, MAX$CPU currently = 10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1604</td>
<td>NPS$VPS(MAX$CPU)</td>
<td>P</td>
<td>0</td>
<td># of virtual processors</td>
</tr>
<tr>
<td></td>
<td>(one byte for each possible CPU, MAX$CPU currently = 10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1614</td>
<td>HD$INT$FLAG(MAX$CPU)</td>
<td>R</td>
<td>X</td>
<td>H/W interrupt flag (one for each possible CPU, MAX$CPU currently = 10)</td>
</tr>
<tr>
<td>1624</td>
<td>EVENTS</td>
<td>R</td>
<td>1</td>
<td>Number of events</td>
</tr>
<tr>
<td>1625</td>
<td>CPU$INIT</td>
<td>R</td>
<td>0</td>
<td>Log in CPU number</td>
</tr>
<tr>
<td>1626</td>
<td>SEQUENCERS</td>
<td>R</td>
<td>0</td>
<td>Number of sequencers</td>
</tr>
<tr>
<td></td>
<td>SEQ$STAB$(100)</td>
<td>S</td>
<td></td>
<td>Sequencer table</td>
</tr>
<tr>
<td>1627</td>
<td>SEQ$VAMP</td>
<td>R</td>
<td>X</td>
<td>Name of sequencer</td>
</tr>
<tr>
<td>1628</td>
<td>SEQ$VALUE</td>
<td>W</td>
<td>X</td>
<td>Value of sequencer</td>
</tr>
</tbody>
</table>

* - byte    W - word    S - structure    X - not initialized
range when MCORTEX primitives (access to common memory) are used or data computed by a producer must be placed in shared memory for consumption by another process within that cluster. MCORTEX performs its functions by setting up a section of common memory called GLOBAL memory. Table 1 shows how this shared resource is logically organized (Appendix H contains the actual memory locations).

Access to GLOBAL memory is resolved through the combination of a hardware bus lock (LOCK prefix preceding a machine level instruction), and a software lock (GLOBAL$LOCK) located in GLOBAL memory. MCORTEX primitives that access GLOBAL memory set the hardware bus lock through the PL/M-86 function LOCKS$ET [Ref. 16]. The real processor executing the kernel, that is executing LOCKS$ET, is given sole access to the MULTIBUS for the duration of a single instruction. A LOCK prefix preceding an XCHG instruction causes a value in a register (contents ?7?) to be exchanged with GLOBAL$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 kernel repeats the procedure until a zero is obtained from GLOBAL$LOCK. The XCHG instruction requires two bus cycles to swap 32-bit values, thus without the LOCK prefix it is possible for another processor to obtain the bus between cycles and gain access to the partially-updated GLOBAL$LOCK semaphore. When relinquishing the software lock, the kernel
IV. DETAILED SYSTEM DESIGN

A. DESIGN ISSUES

1. Real-Time Processing

Real-Time processes are of a time-critical nature, and as such are always resident in memory. The time required to swap a real-time process out of memory, to make room for another, would consume the very same resource being allocated - the CPU. The early designers of CORTEX considered this issue carefully and the result is an operating system that minimizes context switching overhead. CORTEX processes reside permanently in memory once loaded; and only CPU registers, critical to a context switch, are modified. Just as important are issues such as:

(1) allocation of shared resources, (2) process integrity, (3) process synchronization, and (4) interprocess communication.

2. Shared Resources

Within a cluster (Figure 3) are three critical shared resources: the N13010 ESCB (i.e., Ethernet), common memory, and shared memory itself. The hierarchical bus structure limits the access of each real processor to common memory and shared memory, and the bus arbiter grants access in a random manner. Each processor executes processes in its own local RAM and only makes memory accesses outside that
concerning the multi-user CP/M-86 system RICS is described in [Ref. 15] and will not be reiterated here.
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, FS) are identical.

The MCORTEX CRFTFPROC parameters include the absolute location of process start, stack, and data. For this reason it is advantageous to locate processes absolutely when linking. LINK86 provides such an option [Ref. 13: p. 7.6], however, the ABSOLUTE option is applicable to the entire CMD file created and cannot be used to distribute the file non-contiguously in memory.

Rowe [Ref. 6] experienced some difficulty using LINK86 as described in [Ref. 13]. His observation was entirely correct, but it was easily corrected by generating a new CP/M-86 operating system using Version 1.1 CCP and BTOS (integrated with a modified BIOS). Version 1.0 contained an error that caused the 128 byte header, preceding CMD files, to be parsed incorrectly at file load time. Details concerning this header are contained in [Ref. 14]. The BIOS was modified due to the removal of the bubble memory board from the multi-user CP/M-86 system. This process of generating a new CP/M-86 operating system is described in adequate detail in [Ref. 14]. The details
### LOCAL RAM

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04390</td>
<td>CP/M-86 OS</td>
</tr>
<tr>
<td>08200</td>
<td>USER AREA</td>
</tr>
<tr>
<td>08700</td>
<td>LOADER</td>
</tr>
<tr>
<td>0C800</td>
<td>MCORTEX</td>
</tr>
<tr>
<td>0FFFF</td>
<td>DDT86</td>
</tr>
</tbody>
</table>

### COMMON MEMORY

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0000</td>
<td>CP/M MULTI-USER AREA</td>
</tr>
<tr>
<td>E5300</td>
<td>MCORTEX GLOBAL DATA</td>
</tr>
<tr>
<td>E7FFF</td>
<td></td>
</tr>
</tbody>
</table>

### SHARED MEMORY

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>USER SHARED DATA</td>
</tr>
<tr>
<td>10078</td>
<td>TRANSMIT DATA BLOCK</td>
</tr>
<tr>
<td>10666</td>
<td>RECEIVE DATA BLOCK</td>
</tr>
<tr>
<td>10C58</td>
<td></td>
</tr>
</tbody>
</table>

**ERB** - Ethernet Request Block  
**ERP** - Ethernet Request Packet

**FIGURF 6** Cluster Memory Map
required, memory reserved for CP/M-86 may hold user processes.

Descriptions of processes in memory are provided to MCORTEX through the CREATE$PROC primitive. 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, which is a software abstraction of a real 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 status into GLOBAL memory and the process stack into local memory.

As described by Rowe [Ref. 6: p. 28], special effort has been made to accommodate processes created under PL/I-86 and linked using LINK86. LINK86 concatenates all PL/I-86 code segments into one segment and data segments into one segment. 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 configuration is shown in Figure 3 of [Ref. 6].
FIGURE 5  Ethernet Packet Format
process is completed, the state of the system is not disturbed.

Access to MCORTEX is through the supervisor at the outermost layer of the MCORTEX four level structure discussed by Klinefelter [Ref. 5: pp. 44-46].

Also resident in each local memory, if required, is the CP/M-86 operating system. In this configuration the full range of CP/M-86 utilities, [Ref. 13] and [Ref. 14], 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 5 gives a representation of the locations of the system code. The diagram includes the location of DDT-8C as required for a debugging session. A developer of user processes should anticipate needing this powerful debugging tool; the space should remain reserved. Also depicted are the locations of the MCORTEX/TRACE 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.

2. **User Processes**

User processes may be located in areas indicated in Figure 6. Additionally, if CP/M-86 utilities are not
a call was made to the MCORETEX scheduler and the highest priority ready process was given control of the CPU. For communication between processes in local memory, no interrupt was issued, a call to the scheduler was made directly.

The use of the interrupt was inconsistent with the philosophy of switching processes only at "safe" points in their execution. These "safe" points were required because of non-reentrant PL/I-86 user process code. An interrupt must not occur during a call to a PL/I procedure that is shared among multiplexed processes. Therefore, the original design had a design error which needed correction.

Also, the use of a preemptive interrupt to signal a possible change to all real processors in a cluster was somewhat counterproductive. To cause all real processors to be disrupted in their execution, just because as few as one virtual processor was made ready, is unjustifiable. However, this preemptive interrupt structure has been maintained in MCORETEX in the event a high priority process must be scheduled. A primitive known as PREEMPT, provides this capability. The PREEMPT primitive is the mechanism to schedule time urgent processing which is vital in real-time systems. PREEMPT, however, must be used carefully and sparingly. Processes that are time critical must only use reentrant code, so that when a return from the time critical
eventcount or calls the await primitive "risks" surrendering the CPU to a higher priority ready to run process. A call to the advance primitive always results in a call to the scheduler. If the calling process is still the highest priority "ready to run" process, it will continue in its execution, otherwise another virtual processor will be scheduled to run and the original process will be blocked ("ready" if an advance operation, "waiting" if an await operation).

In the event there are no user processes in the ready state, the kernel's idle process will run. This process blocks itself every 4 milliseconds and calls the kernel scheduler. If any offboard operation caused an onboard process to be readied, as the only process "ready to run", it will be scheduled. The idle process is always "ready to run", of course, but it has the lowest possible priority.

This implementation of MCORTEX is a major change in the philosophy of previous versions, whereby a system interrupt under MCORTEX control, in conjunction with interrupt flags maintained in GLOBAL memory, provided communication initiation between real processors. Upon receiving an interrupt, each processor checked its flag in GLOBAL memory to determine if the interrupt was intended for a process in its local memory. If not, the process executing at the time of the interrupt continued. Otherwise
(7) The NI3010 moves the received frame from its receive queue to host memory. The NI3217 precedes the packet with a frame status byte, a null byte, and two bytes containing the frame’s byte length. After transferring each data byte, the DMA controller increments the address in the bus address registers and decrements the byte count in the byte count registers. The NI3010 generates a receive-DMA-done interrupt when it finishes transferring the frame or when the byte count reaches zero.

(8) The host responds to the RDD interrupt by issuing an interrupt code of zero, disabling the interrupt from the NI3010 board.

The determination of the order in which commands are given is entirely dependent on the application. The 16K byte receive buffer allows the host to read this buffer (via RDD interrupt operation) at its own convenience. This buffers the MULTIBUS from the unpredictable arrival times of intercluster traffic, consequently reducing the time-critical service requirements on the receiving cluster. In contrast is the 2K byte, single packet, transmit buffer. The host system should strive to favor outbound packets to reduce the processing delay by any processors in the cluster.

E. SOFTWARE SERVICES

1. Operating Systems

A copy of a kernel of MCORTEX resides in each processor’s local memory and is a part of the address space of each local process. Additionally, GLOBAL memory is accessible to MCORTEX to facilitate interprocess synchronization. Processes are scheduled for execution by a kernel of MCORTEX on each SEC. Any process that advances an
(5) The NI3010 moves the transmit packet from host memory to its transmit buffer (only one packet at a time may be resident in this buffer). After accepting each data byte, the DMA controller increments the address in the bus address registers and decrements the byte count in the byte count registers. When the byte count reaches zero and its transmit register is empty, the NI3010 interrupts the host processor. This is a transmit-DMA-done (TDD) interrupt. The transmit data is now stored in the transmit buffer.

(6) To transmit this data or the Ethernet, the host issues a Load transmit Data and Send command (29H). The NI3010 carries out the command, reflecting its status in the register. The host must read the status register.

The following describes what happens when a receive packet goes from the NI3010's receive queue (16K byte capacity) to MULTIBUS memory:

(1) The host issues an interrupt code of 4. This enables a receive-block-available (RBA) interrupt from the NI3010.

(2) The host gets a receive-block-available interrupt. The host now knows that the NI3010's receive queue has a frame awaiting transfer to MULTIBUS memory.

(3) The host writes an interrupt code of zero to the NI3010's interrupt enable register. Writing this register clears the NI3010's interrupt line.

Note: Just as in the transmit process, this step ensures that the DMA controller does not start a DMA transfer as soon as the byte count register contains a non-zero value.

(4) The host writes the 24-bit MULTIBUS memory address into the NI3010's bus address registers.

(5) The host writes the byte count of its MULTIBUS buffer into the NI3010's byte count registers.

(6) The host initiates a DMA transfer. It does this by issuing an interrupt code of 7. This also enables a receive-DMA-done interrupt (RDD) from the NI3010.
invocation procedures for this diagnostic routine is
contained in Appendix L.

Of particular importance is the requirement to read
the Command Status Register at the beginning of any code
that controls the NI3010. This is necessary because of the
power-up diagnostic that runs at system start-up or due to a
MULTIBUS reset. This automatic testing feature places a
value in the Status Register that must be read to clear the
register before any commands can be issued to the NI3010.

The NI3010 transmit process consists of obtaining
data packets from shared data memory, via a DMA operation,
forming them into Ethernet frames, and successfully
delivering them to the intercluster bus (the "Ether").

The following describes what happens when a
transmit packet goes from MULTIBUS memory to the NI3010:

1. The host writes an interrupt code of zero to the
   interrupt enable register on the NI3010. Writing this
   register clears the NI3010's interrupt line
   currently set for interrupt 5).

   **Note:** This step ensures that the DMA controller
does not start a DMA transfer as soon as the byte
count registers contain a non-zero value.

2. The host writes a 24-bit MULTIBUS memory address into
   the NI3010’s bus address registers.

3. The host writes the packet’s byte count into the
   NI3010’s byte count registers.

4. The host initiates a DMA transfer by writing to the
   interrupt enable register an interrupt code of 6.
   The NI3010 will now interrupt the host processor when
   it completes the DMA transfer.
merely sets GLOBAL$LOCK to zero. The "granularity of locking" by the kernels, is all of GLOBAL memory, i.e., no two kernels have access to GLOBAL memory simultaneously.

Users have no access to GLOBAL memory, however. MCORETEX 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 caller’s turn at the required resource. TICKET advances the sequencer value in GLOBAL memory so that succeeding requests receive higher numbers. Given the situation where a "busy wait" is not to be employed, a process requesting the resource then makes another supervisory call, this time on WAIT, providing both an identification of the resource and the process turn number. If the resource is not busy, the process will receive immediate access, otherwise the process gives up the CPU.

3. Process Integrity

The design of MCORETEX relies heavily on user cooperation for process integrity. The supervisor controls access to the MCORETEX functions, but even this is a software control and a process that intentionally or inadvertently
destroys GLOBAL data would be disastrous. Although local RAM of a processor is inaccessible from MULTIBUS, thus protected from a 'runaway' process, common memory and shared memory are not. Protection from this type of failure requires hardware protection not presently in the system. The low cost of microcomputers however, allows for redundant back up systems which can limit the effects of such failure due to a processor hardware fault.

4. Process Synchronization

Process synchronization is accomplished under MCORTEX through the functions ADVANCE, AWAIT, and PREEMPT. These synchronizing primitives are supported with the functions CREAT$EVC, CREAT$FIFO, READ, DEFINE$CLUSTER, DISTRIBUTION$MAP, and TICKET. Consumer processes use AWAIT to ensure that data they require is ready. Producer processes use ADVANCE to inform consumers that a new iteration of 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 highly time critical nature. A call on a synchronizing primitive may, or may not result in relinquishing the CPU. The CPU is always assigned to the highest priority ready virtual processor on each board regardless of which synchronization function invoked the scheduler (except for PREEMPT, of course).
Before using ADVANCE or AWAIT, an eventcount must be created using CREATESEVC. Consumers and producers then communicate using the agreed upon eventcount. The current value of an eventcount can be determined through a call on READ. The functions of CREATE$SEQ and TICKET are as discussed earlier, but with broader applications.

The only entity presently distributed by MCOTEX over Ethernet is eventcounts. However, this feature alone allows distributed processes to synchronize. The manner in which processes synchronize is no different than that already discussed. The fundamental issue then becomes the means by which an eventcount of interest can be made available to a producing or consuming process.

Eventcounts may be used in any number of combinations. Producing and consuming processes may be resident in the same cluster, different clusters, or mixed (i.e., a producer and one consumer in the same cluster, with another consumer of the same data type in another cluster). Processes are not aware, however, as to their own distribution - they continue to advance eventcounts and await values just as they always did. This transparency is provided through the primitives DEFINE$CLUSTER and DISTRIBUTION$MAP.

DEFINE$CLUSTER is a procedure that assigns a 16-bit address (the last two bytes of the destination field of an Ethernet packet) to a cluster, and DISTRIBUTION$MAP causes
the "remote_addr" field of an eventcount name (see Table 1) to be assigned a value. It is necessary to statically manage the distribution of eventcounts, just as it is necessary to statically manage blocks of shared memory for user processes. It is a decision that must be made by personnel responsible for the development of AEGIS software that will run on FTC* under MCORTEX.

A user process does not need to know the address of the cluster in which it resides, nor is it required to know the cluster addresses of processes that it synchronizes with. Therefore, DEFINE$CLUSTER and DISTRIBUTIONS$MAP are not primitives called by a user process, but by a system process that calls these primitives in its initialization module. As mentioned before, eventcounts must be created prior to their use. The convention of MCORTEX is that user processes do not create or define them (as a constant) in any way. The same system process that calls DEFINE$CLUSTER and DISTRIBUTIONS$MAP, also creates all user and system eventcounts and sequencers. Thus, symbolic names only are used by user processes at run-time and the system initialization module at creation time, providing a level of security. It will be seen later how this security is even further enhanced. The manner in which user and system processes are created is covered in complete detail in Chapter V.
5. **Interprocess Communication**

MCOTFX, at this stage of development, does not provide any means by which data (produced) can be transmitted between clusters. Within the same cluster, however, shared data is stored for consumption in the 64K byte FAM shared memory board. Any buffering of data by user processes must be done explicitly. There is no dynamic allocation of this resource.

With Ethernet serving as the intercluster bus, with eventual data transfer planned, due consideration must be given to the distribution of user processes within RTC*. Processes with a high interprocess communication rate should be located as close together as possible. When this is not feasible, a fairly high efficiency penalty will have to be paid. The Ethernet is clearly the highest level bus in RTC* and memory located at a remote cluster must be viewed as the highest level memory in the memory hierarchy of RTC*. As such, a nonlocal memory access should be avoided as much as possible, but it will never be entirely avoidable. Clearly average memory access times will drop as the rate of local memory references increase. In a distributed system such as RTC*, the nonlocal "hits" on memory should be kept to a minimum. To reiterate, if high volume communicating processes can possibly reside in the same cluster, then they should be so located.
B. ETHERNET ACCESS

1. Cluster Input/Output

MCO:TEX must provide a means to transmit copies of values of eventcounts to a remote cluster. This operation must be entirely transparent to user processes, since they have no knowledge of their distributivity.

Figure 7 illustrates an abstraction of the flow of data and control signals necessary to achieve a transmission over Ethernet. It embodies the principles of a flowchart, as well as an abstraction of processing modules and control signals. Refer to Figure 7 for the following discussion.

The user processes resident in either SBC 1 or 2 advance an eventcount through the ADVANCE primitive operation. The ADVANCE primitive makes a determination as to the locality of the eventcount and calls the internal routine SYSTEM$10 only if the eventcount is distributed, i.e., a remote copy is needed at another cluster. The SYSTEM$10 routine makes a determination as to the eventcount communication path (currently the only option is Ethernet). Since the path is Ethernet, the SYSTEM$10 routine writes an Ethernet Request Packet (ERP) to a circular buffer in shared memory, known as the Ethernet Request Block (ERP).

As a shared resource among MCO:TEX kernels, an ERP slot in the ERP must be arbitrated for. The TICKET mechanism is employed in SYSTEM$10, and the circular buffer (ERP) contains ERP's that must be processed. The SYSTEM$10 routine
FIGURE 7 Intercluster Input/Output Processing
increments a system reserved eventcount (ERE$WRITE) to notify the NI3010 Device Driver and Packet Processor that an ERP has been written. This "advancing" of ERE$WRITE also allows any other kernel executing the SYSTEM$IO routine to continue if it was attempting a simultaneous write to the ERP. The NI3010 Device Driver and Packet Processor (hereafter referred to as the Driver) is a consumer of ERP's and also processes Ethernet packets received from other clusters. As a consumer of ERP's it is a system process of a cyclic nature that is scheduled in the same manner as user processes. However, this routine is dedicated to high density I/O operations, and as such is never blocked. In the highly unlikely situation where there are no ERP's to consume or packets to receive and process, the Driver idles in a 'busy wait.'

Currently the only type of ERP to be processed is an 'eventcount type', whose format is shown in Figure 8. The NI3010 Driver decodes the ERP and based on the information

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventcount Type</td>
<td>Eventcount Name</td>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8 Ethernet Request Packet Format

it sets up a transmit-data-block in shared memory. In fact, this block is the Ethernet packet, less the 64 bit preamble.
and 4-byte Frame Check Sequence (FCS). The Driver then initiates a Transmit-DMA-Done (TDD) operation to transfer the block to the transmit queue of the NI3010. The Driver follows up the TDD interrupt with a Load and Send command (29H) to the NI3010 directing it to transmit the packet over Ethernet.

Inbound packets are processed by the Driver through the Receive-Block-Available (RBA) and receive-TMA-Done (RDD) operation sequence described in Chapter 3. The Driver favors outbound packets, to avoid the possibility of a bottleneck due to a "clogging up" of the FRP. When it does set up for an RBA interrupt, it will continue to the conclusion of processing the packet received. Following the DMA of the packet to the receive-data-block area in shared memory, the Driver decodes the data fields of the packet (Figure 9) and calls the appropriate MCORTEX synchronization primitives. The Driver continues to operate in this manner, determining via an eventcount value (incremented by SYSTEMS10) whether or not an FRP exists in the FRP that needs to be processed and in the absence of one receives or inbound packet for processing.

The truly asynchronous nature of the Ethernet service should be apparent. Once SYSTEMS10 deposits an FRP, it returns immediately to the user process. The user process is not held up in its execution due to a transparent request for system input/output. The Ethernet Request Packet is the
embodiment of the request, and in different forms is passed between various clusters of RTQ*. It contains all the information needed to perform the operation independently of the requesting process.

C. PACKET ROUTING ALGORITHM

Thus far, all illustrations and discussions of RTQ* pertained to only two clusters, but this should not be construed as a limitation. Given that more than two clusters can exist in RTQ*, some methodology must exist to route packets to as few as one and to as many as needed (up to the maximum clusters that exist).

The established convention is that no cluster will send a packet to itself. If an eventcount is advanced that requires a local update and one remote update (to one cluster) then only the local copy will be updated and only the cluster that is to receive the eventcount value will receive a packet. This clearly reduces needless packet processing at a cluster that has no interest in that eventcount, i.e., there are no producers or consumers interested in its value. Therefore an algorithm had to be developed that selectively eliminated packets from being transmitted to an inappropriate cluster.

The N13010 has a packet addressing mode known as GROUP addressing, whereby multicast addresses can be loaded into a multicast address table onboard the N13010. Provided this
table is loaded prior to \texttt{NI3010} use, any packet received that has bit 1 of the destination address field set to one (i.e., the first byte is odd) is interpreted as a multicast packet and a lookup is done in the table. If a match of the destination address is found in the table, the packet is loaded in the \texttt{NI3010}'s receive queue. If the Driver (Figure 7) enabled an \texttt{RBA} interrupt, the \texttt{NI3010} will issue an interrupt signifying that a packet has been received for this cluster. The Driver will then process the packet accordingly (format shown in Figure 9).

The Driver programs the \texttt{NI3010} to accept \texttt{GROUP} addresses in its multicast table, depending on the distribution of event counts in PTC*. The Driver (Appendix K) has a module

\begin{tabular}{|c|c|c|}
\hline
\textbf{DATA FIELD} & & \\
\hline
\textbf{Byte 1} & \textbf{Byte 2} & \textbf{Byte 3} & \textbf{Byte 4} \\
\hline
Packet Type & Type Name & Value & \\
\hline
(FVC) & (EVC ID) & \\
\hline
\end{tabular}

* - Packet is decoded based on byte 1.

\textbf{Figure 9} Event count Type Ethernet Packet

that reads the local cluster address and group addresses from a file called "address.dat". The local cluster address is used to set up the physical address of the \texttt{NI3010} see
[Ref. 12] for details). Any packets on Ethernet that has one of the group addresses or the physical address in the destination field is received and processed.

For packets to be transmitted over Ethernet, only the last two bytes of the destination field is programmable. This minimizes the amount of data that must be maintained and manipulated for packet addressing. The 'remote:ad:in' field in the EVENTCOUNT TABLE in GLOBAL memory contains the two bytes.

Figure 10 contains an example of a logical connection of clusters (they are all physically connected by Ethernet) dependent on the distributivity of the eventcounts. The lines, with numbers adjacent to them, represent a connectivity relationship of classes of data whose producers and consumers synchronize on certain eventcount values. The vertical dotted lines represent a partitioning of process types and group addresses, shown below the clusters. The number in the cluster block is the physical address of each cluster. It can be seen that a producer of Type 1 data, a consumer of Type 2 data, and a consumer of Type 3 data are all present in cluster 8. A logical connectivity exists between all clusters as a result of the Type 1 data. Type 1 consumers exist at clusters 1, 2, and 4). An advance by producer P1 must cause a packet to be sent to clusters 1, 2, and 4.
<table>
<thead>
<tr>
<th>Process Types</th>
<th>Process Types</th>
<th>Process Types</th>
<th>Process Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
</tr>
<tr>
<td>C2</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td>C3</td>
<td>C3</td>
<td>C4</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group ADDR</th>
<th>Group ADDR</th>
<th>Group ADDR</th>
<th>Group ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0008</td>
<td>0004</td>
<td>0002</td>
<td>0001</td>
</tr>
<tr>
<td>000C</td>
<td>0007</td>
<td>0007</td>
<td>0007</td>
</tr>
<tr>
<td></td>
<td>0006</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P1 - Producer of Type 1 Data
C1 - Consumer of Type 1 Data

Figure 10  Ethernet Packet Routing
Consider the logical connectivity of a certain eventcount to be represented by a binary one at each cluster it connects. Therefore, for the Type 1 eventcount the 4 cluster connectivity would result in 1111 base-2 or 1F base-16. By performing an exclusive-or operation on AF4 with the producer’s own physical address (88 base-16, in this case) a result of 007H would be formed. Since consumers at clusters 1, 2, and 4 are interested in Type 1 data, the NI3013 Driver must program 007H into the multicast table. In reality the address (03-00-00-02-02-00-00) (6 bytes in length, first byte being odd) would appear as an entry in the table.

Continuing with this example, consider the Type 4 connectivity. The binary connectivity is 1111 and by performing an exclusive-or with the value 001H (address of cluster 1, where the producer is present) results in 207H. The NI3013 at clusters 2 and 4 must have (03-00-00-02-02-00) in the multicast table. All other values shown in Figure 18 are derived in an analogous manner.

The "remotesaidir" field of an eventcount contains the binary connectivity discussed above. The ADVANC3 procedure of MCOTEX takes a test to see if the remotesaidir field is equivalent to the "localsaidir" (as defined by the DFINFSCCLUSTER primitive). If they are the same then SYSTEM$I0 is not called and intracluster processing continues. If they are not equivalent, then an exclusive-or
VI. CONCLUSIONS

The principal goals of this thesis were achieved. The modifications to the previous version of MCOI\textTeX, to allow the distribution of processes over a high speed interclusterous, were developed and appropriately tested. Eventcount values are currently the only entities that are transferred in packet form over Ethernet. However, the framework to easily extend the distributivity of other entities is established.

From the viewpoint of user processes, access to Ethernet is paired in an entirely transparent manner. This access is truly asynchronous in the sense that a return to the requesting process occurs when an Ethernet Request Packet is written to shared memory, not when actual output of the information occurs. Provided the W13012 Driver and Packet Processor keeps up with the I/O rate, a bottleneck will not result. The dedication of the Driver to its own real processor assures this.

The Driver software needed to distribute MCOI\textTeX over Ethernet is device-dependent, however MCOI\textTeX only interfaces with this routine through the convenient abstraction of an Ethernet Request Packet. Any changes in the Driver will not cause an undesirable ripple effect of
creation of the IDLE and INIT processes (also "UNIT0" with \[XTRACE\]), and the user initialization process is then entered. Operation after this point is determined by the user processes.
KORE.TRC and MCOPTEX or MXTRACF respectively. When this
generalization does not hold, the differences will be noted.

2. Operation of the MCOPTEX Loader

MCOPTEX.CMD is an executable file under the CP/M-86
operating system. Invoking MCOPTEX without KOPE.OPS on the
default drive results in an error message and an abrupt
return to CP/M-86. MXTRACF requires KOPE.TRC. The loader
announces that it is on line, and provides a prompt to query
the interactive user whether or not GLOBAL memory should be
loaded. Only the first processor activated should load
GLOBAL memory. Subsequent loads of GLOBAL memory will
destroy data needed by executing processors. If no initial
load of GLOBAL memory is made the results are undefined.

KOPE 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,
KOPE is read in and positioned block at a time as required.
KOPE load is followed by a request for a process file name.
The loader expects one file name to be entered, and results
are unpredictable if a "filename.filetype" does not precede
a keyboard <RETURN>. 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 or filetype (.CMD). After loading
the user file, the loader passes control to MCOPTEX.
MCOPTEX initializations are performed within KOPE, including
F. MCORTEX LOADER

1. The Loader

Prior to Rowe's [Ref. 6] work the MCORTEX executive was assigned to the file KORE and was accessible only through utilities in the INTELTEC MDS system. This file contained all the multiprocessor 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. 5: Appendix A, B] for a complete description of the process.

The KORE.OPS and KORF.TPC files, now loadable under CP/M-86 through the MCORTEX and MXTRACE loaders, are derived from KOPE. KORE.OPS provides no system diagnostics, whereas KORE.TRC provides CRT output to indicate the entry into MCORTEX primitives. It is expected that during the software development phases, KORE.TRC will be used to facilitate debugging. In some circumstances this may not be feasible due to the reduced speed of execution as a result of the I/O overhead.

Appendix A details the procedure used to produce KORE.OPS and KORE.TPC from KOPE. Further discussion will use the terms KORE and MCORTEX to mean either KOPE.OPS or
GATEMOD and GATETRC both act as translators of user calls into formats required by the MCO$TEX and MXT-ACF supervisors respectively. The only difference in the two gate modules is the address of GATES$KEEPER in their associated KOPFs. As assembly language routines called by PL/I-86 MCO$TEX processes, GATEMOD or GATETRC use the established parameter passing conventions (PL/I-86 to ASY86) to build the stack structure expected by the supervisor module (PL/I-86 format), supplying function codes and padding when required. A call is then made to GATES$KEEPER. If the call is to READ or TICKET, space is reserved on the stack for the returned value. This value is popped into the FX register (PL/I-86 convention) before exiting to the calling process.

KOPE functions do not guarantee the integrity of the ES register. PL/I-86 in OPTIONS (MAIN) initializations, however, establishes the FS, SS, and DS registers to be of equal value, and some runtime routines expect this relationship to be maintained. The gate modules push the FS register onto the stack on entry, and pop it before return to the calling routine, thus preserving its precall value. Entirely transparent to user processes, the FS register value is preserved throughout MCO$TEX calls.
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-86 for the system stack.

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 MCORTEX environment.

2. Gate Module

GATEMOD.OBJ (or GATETRC.OBJ) must 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.
The `CREATE_PROC` procedure has eight actual parameters. The first two are process identification and process priority. These are `BIT(8)` values assigned by the software developer, with due consideration given to the module's function. Four other parameters, the CS, DS, SS, and ES register values can be determined by performing an executable load of the process CMD file under DDT86. Values displayed by DDT86 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-86 runtime routines. Except under carefully considered circumstances, this should be 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-86, 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 `MCOPTFX` process stack is required, its SP can be
# Table 2 - Map File

Map for file: C2USERS,CMD

Segments

<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>272D</td>
<td>'0007:0005-0731)</td>
<td>PUB</td>
<td>CODE</td>
<td></td>
<td></td>
<td>CODE</td>
</tr>
<tr>
<td>002F</td>
<td>'0007:0120-0141)</td>
<td>PUB</td>
<td>DATA</td>
<td></td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>0021</td>
<td>'0007:06E-062E)</td>
<td>WORD</td>
<td>COM</td>
<td>CONSP</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>0018</td>
<td>'0007:0630-0642)</td>
<td>WORD</td>
<td>COM</td>
<td>FPBSTK</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>002E</td>
<td>'0007:0644-0671)</td>
<td>WORD</td>
<td>COM</td>
<td>FPB</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>002E</td>
<td>'0007:0672-0673)</td>
<td>WORD</td>
<td>COM</td>
<td>CACOL</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>0022</td>
<td>'0007:0674-0676)</td>
<td>WORD</td>
<td>COM</td>
<td>FILAT</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>0022</td>
<td>'0007:0677-0678)</td>
<td>WORD</td>
<td>COM</td>
<td>PMTS</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>002B</td>
<td>'0007:0686-0687)</td>
<td>WORD</td>
<td>COM</td>
<td>FPBUFF</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>002A</td>
<td>'0007:0692-0694)</td>
<td>WORD</td>
<td>COM</td>
<td>CONSP</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>0025</td>
<td>'0007:06DC-06DA)</td>
<td>WORD</td>
<td>COM</td>
<td>SYSIN</td>
<td></td>
<td>DATA</td>
</tr>
<tr>
<td>0029</td>
<td>'0007:06FC-06F3)</td>
<td>WORD</td>
<td>COM</td>
<td>SYSPRINT</td>
<td></td>
<td>DATA</td>
</tr>
</tbody>
</table>

Groups

<table>
<thead>
<tr>
<th>CGROUP</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGROUP</td>
<td>DATA</td>
</tr>
</tbody>
</table>

map for module: C2_USERS_INIT

C224  (C007:0075-0028)  CODE
C237  (C023:0107-0136)  DATA

map for module: MSLORDER

C225  (C007:0029-005D)  CODE
C22B  (C007:003E-0072)  DATA

map for module: TRKPPRT

C22B  (C007:00DF-0128)  CODE
C012  (C007:0174-0185)  DATA

map for module: GATEM/T

C213  (C023:0103-020P)  CODE
C004  (C004:0166-0180)  DATA
user process executed, and is used to create user processes
only. A system process written in PL/I-86 can use its
initialization module to create event counts, sequencers, as
well as creating itself. After all initializations are
performed, an AWAIT('FF'B4,'0001'B4) should be executed.
This puts all initialization processes on a common reserved
event count threat. An ADVANCE('FF'B4) by any process will
return all processors to CP/M-86 control (providing CP/M-86
is resident locally).

MCOTEX processes are written as parameterless
PL/I-86 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 dynamically dependent on the use of MCOTEX synchronizing
primitives by user processes.

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 are performed with the MAP command option
selected, since this provides information required to define
user processes. A partial MAP print out for a demonstration
process (full discussion in Appendix E) is shown in Table 2.
themselves at 'convenient' points in their execution (with the \texttt{AWAIT} primitive), this lack of reentrancy is not a problem. In previous versions of \textsc{MCCITEX}, with the preemptive interrupt 4 to signify that a process has been readied by an offboard operation, the interrupt could easily "catch" two multiplexed processes using the same non-reentrant runtime routine or utility. The change in scheduling philosophy, as discussed in the \textsc{SOFTWARE SERVICES} section of Chapter 3, reduces this "window of vulnerability." If a process is scheduled, via a \texttt{PREAMPT} operation (which still uses interrupt 4), behind a process that was blocked and using the same runtime routine or utility, the originally scheduled process's execution state could be catastrophically altered. This type of situation can be avoided through a careful distribution of user processes. That is, don't allow a process that may be readied via a \texttt{PREAMPT} operation to be multiplexed with a process that might possibly use the same utilities or PL/I runtime routines. If this cannot be avoided, the only remaining alternative is to write the shared code as reentrant procedures. It is anticipated that future research, Inc. language compilers and CP/M-86 operating system functions will address and resolve this lack of reentrancy. For now, it remains a problem.

\textsc{MCCITEX} currently expects an initialization module to be located starting at \texttt{043900h}. This module is the first
allow user processes to access shared memory outside the 64K byte range), without resorting to assembly language code to effect data moves. The ABSOLUTE feature of LINK-26 [Ref. 18] provides such an alternative. The DS register can be assigned a value (by using DATA [ABS[v]], where v represents the value) sufficiently high to allow an offset to be added to it at runtime, forming a physical address in the range 1000H - 1FFFFH (first segment shared memory). This accomplishes the desired effect. It is precisely this technique that is used in the NIO10 Driver. The Driver was linked with a value of 0800H in the LINK option file, and when added to an offset of 8000H allowed access to a based array structure called PnP (Ethernet Request Block). Note that 0800:8000 is the same as 1000:0, but the first logical address permits local data to reside in local memory and shared data in the first segment. User processes can use this same technique for interprocess communication.

MCORTEX processes that are multiplexed (multiprogrammed) on one real processor must be linked into a single CMP module. Multiprogrammed processes may share common PL/I-86 runtime routines as well as CP/M-86 utilities. However, this sharing of runtime routines and utilities presents a problem. Careful examination of the machine code of the runtime procedures and utilities revealed the fact that they are not reentrant routines. Under normal circumstances, since processes only block
One such mechanism is the file GATEWAY.PLI, as referred to by Rowe, and now known as the SYSPFF.PLI (for System Definitions) file. This file must be included in all programs (using the PL/I INCLUDE directive) making calls on MCOREFX functions. The change in filename was introduced as a result of this file's multifunction role. In addition to declaring the MCOREFX functions as ENTRY values with attribute lists, the file also contains the symbolic names of events, sequencers, and pointers for shared data structures. This adds a level of security not present in previous versions of MCOREFX. The misspelling of a symbolic name will be caught by the compiler as the use of an undeclared variable.

An example of the use of pointers to access a data structure in shared memory is provided by the N13012 Device Driver and Packet Processor. This routine performs an UNSPFC function (described in [Ref. 17 p. 72]) call to absolutely locate the Ethernet Request Block structure so that it can consume Ethernet Request Packets generated by KORF's SYSTMSIC routine. The value appearing on the right-hand side of the UNSPFC assignment statement is a symbolic name defined in the SYSPFF.PLI file. Proper static management of shared memory, with symbolic assignments, assures the integrity of user data.

Due to the limitation of pointer variables to sixteen bits in PL/I-90, some method had to be devised to
V. PROCESS DEVELOPMENT AND THE MCOTEX LOADER

A. PROCESS DEVELOPMENT

1. PL/I-86 User Processes

Rowe [Ref. 6] is responsible for the integration of MCOSEX into the CP/M-86 environment. Although his discussion of PL/I-86 user process development is more than adequate, enough changes have been made to warrant another discussion.

Procedures written in PL/I-86 become MCOTEX processes through execution of CREATE_PROC functions. MCOSEX 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 MCOTEX executive prior to every entry into the process. This is accomplished transparently when making MCOSEX function calls. User-defined or built-in PL/I-86 procedures in a MCOTEX process can be accessed from within the process normally, however, a MCOTEX process must be entered through a MCOSEX function call.

KOF is the name assigned to the kernel of MCOSEX and is written in PL/M-86, and it is necessary for calls to the supervisor to meet PL/M-86 parameter passing conventions. Rowe [Ref. 6] provided mechanisms to resolve differences between simple user calls and supervisor calls.
operation is performed on the remote$addr field (remote$addr XOR locals$cluster$addr) and the resultant two byte value and appropriate eventcount information is written to an ERP. The NI3210 Driver then queues the ERP and forms the appropriate packet format (Figure 9), initiates the DMA operation to the NI3210, and issues the Load and Send operation.
changes in the operating system code. This integration of
hardware and software is easily modified and extensible.

The creation of eventcount and sequencers in the
initialization module of a carefully tested system process
provide a level of security not present before. This
security is further enhanced by expanding the role of the
SYSDEF.PLI file that is included in each MCORTEX process. By
convention the user processes cannot alter the constant
definitions present in SYSDEF. The user processes are not
hostile anyway, but it will clearly not be to their
advantage to alter this file. The assigning of pointers for
shared structures further elevates the level of security.

The NI3010 Device Driver and Packet Processor is a basic
MCORTEX system process that is highly modular, virtually
self-documenting, and extensible in nature. By modifying
this code and the supporting code in MCORTEX, the
distribution of other entities can be achieved. The
distribution of sequencers is a nontrivial matter and
careful consideration must be given to the speed at which a
ticket value is returned to the requesting cluster. Ethernet
packets will unavoidably be queued up in NI3010 input
buffers, and the speed in which they would be processed by
the current Driver is fixed. A sequencer-type packet (not
recognized by the current driver) would be processed
immediately by the Driver, i.e., a value would be returned
from the GLOBAL data of the cluster responsible for the
shared resource, and an Ethernet packet would be sent out immediately.

The distribution of user shared data could similarly be achieved, with the buffering of data in the shared memory of each cluster. The synchronization on successive iterations of data would be realized in the same manner as previously discussed.

The issue of packet security is a crucial one. The inherently reliable Ethernet is adequate in most instances, but a one bit error in \((10^{**8})\) to \((10^{**11})\) bits could be catastrophic enough when it occurs, so that an "acknowledging Ethernet" may have to be developed. Enough adequate testing has not been conducted in the AEGIS Simulation Laboratory to draw any conclusions in this area.

The lack of reentrancy in runtime code and CP/M-86 utilities is an issue that needs to be more actively addressed. A 'LARGE' PL/I-86 compiler is under development by Digital Research, Inc. that should resolve the reentrancy problem and the limited range (64K bytes) of pointer variables. This product should be available in January 1985.

In addition to solving the aforementioned problems, the "LARGE" compiler will also sever the umbilical cord between the ISIS-II and CP/M-86 operating systems. MCO\textup{TEX} development can then continue in PL/I-86 instead of PL/M-86. MCO\textup{TEX} will then evolve rapidly and consistently with increasingly more complex user processes.
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. REMEX controller is in MULTIBUS, and properly connected to REMEX drive.

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

D. Ensure 32K shared memory module is installed.

E. 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 be marked "computer to computer" and "computer to terminal". Set to "computer to computer".

F. Connect any CRT to the 9600 baud TTY port of the MIS system. Ensure CRT is set to 9600 baud.

G. A CRT will be connected to the SEC 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 SEC's serial port.
II. POWER ON PROCEDURES

A. Turn the power-on key to ON position at MULTIBUS frame.
B. Press RESFT near power-on key.
C. If needed apply power to MICROPOLIS hard disk.
D. Apply power to RFMFX 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 RFMFX disk system will illuminate, and the system is ready.
F. Insert the boot disk into drive B.
F. Apply power to the CRT.
G. Power up the MDS disk drive.
H. Power up the MDS terminal.
I. Turn power-on key to ON at MDS CPU.

III. BOOT UP MDS

A. Place diskette with executable modules and SPIE61 in drive 0.
B. Push upper part of boot switch in (it will remain in that position).
C. Press reset switch and then release it.
D. When the interrupt light #2 lights on the front panel, press space bar on the console device.
F. Reset the boot switch by pushing the lower part of the switch.
F. ISIS-II will announce itself and give the '-' prompt.

IV. LOAD KORE
   A. At MDS console, type 'SRC861<CR>'.
   B. IF "*CONTROL*" appears, SRC was not able to set its band rate. Press RESET on MULTIPUS frame and try again.
   C. If 'Pad EMDS 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. SRC861 will announce itself and prompt with '.'.
   E. Type 'T KORE<CR>'. Wait for '. '. At this point the KORE module has been loaded into the SRC memory, and into the common memory board.

V. SAVING KORE TO CP/M-86 FILE
   A. Leaving the SRC861 process active on the MDS system, disconnect the RS232 J2 connector at the SRC, and connect the terminal prepared earlier.
   B. At the newly connected terminal type "3FPOD4:4<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 DDT96. At this point KOE, CP/M-86, and DDT96 all are resident in the SRC memory and in the 32K shared memory board.
   D. Using DDT96 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 KORF.MP2. The DD06 instructions used for the current KORF.OPS and KORE.TRC files follows:

*** KORF.OPS ***

MB70:0,1932.480:0 *** Move, starting at address 273:0, 1000 bytes of code (main part of KORF) to new start address 480:0.

M439:0,60,560:0 *** Move, starting at address 459:0, 800 bytes of code (initialization module) to new start address 560:0 (following main part as moved above).

ME530:0,800.568:0 *** Move, starting at address E530:0, 800 bytes of code (GLOBAL memory) to new start address 568:0 (following initialization module).

WKORF.OPS.480:0,19900 *** Write to the default disk a file called KORF.OPS starting at address 480:0 and containing 19900 bytes.

*** KORE.TRC ***

M439:0,1000.480:0 *** Move, starting at address 1000:0, 1000 bytes of code (main part of KORF) to new start address 480:0.

M439:0,80,640:0 *** Move, starting at address 108:0, 80 bytes of code (initialization module) to new start address 640:0 (following main part of KORF).

ME530:0,800.648:0 *** Move, starting at address E530:0, 800 bytes of code (GLOBAL memory) to new start address 648:0 (following main KORE & initialization module).
Write to the default disk a file called KOBE.TRC starting at address 4EB0:0 and containing 2480 bytes.

NOTE: The main KOBE 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
DEBUTING TECHNIQUES

DDT86 [Ref. 13] is the primary debugging tool used in software product development in the AEGIS Simulation Laboratory. This debugger allows the user to test and debug programs interactively in a CP/M-86 environment. Far from being a high level debugging tool, DDT86 nevertheless provides the user with the ability to interactively enter assembly language statements, display the contents of memory, trace program execution, and utilize other commands to provide software development assistance.

The use of DDT86 in the development of the N13010 Driver and Packet Processor was invaluable. Ethernet Request Packets could be interactively written to shared memory and the response of the Driver was easily monitored from the same terminal. Breakpoints can be set in processes and the execution of a single board computer will continue until the breakpoint is reached. A process can block and when scheduled next, by a kernel of MCORTEX, the CPU will break at the setpoint.

A particularly valuable feature, that unfortunately is unavailable in DDT86, is that of a watchpoint. A watchpoint is defined here as a location that a debugger would monitor and inform the user when an executing program has made an attempt to execute an instruction at that location. This
feature can be emulated under EDTE6 by using the "A" command (enter assembly language statements) to enter an INT 3 (interrupt 3) command. What the user does not get, however, is a history of the instructions that got the CPU to this execution point. In a single step trace this is not a problem, but execution at near real-time is. In highly modular software, such as MCCRTFX, the single step trace through levels of procedure calls can be an extremely laborious task.

In situations where the state of the CPU does not appear consistent with the executing software, and the reliability of the hardware is questionable, there are few acceptable alternatives to using a digital logic analyzer. The Paratronics 532 is the logic analyzer used extensively in the AEGIS Simulation Laboratory.
APPENDIX C

MCORTEX LOADER

This file is assembled using the RASM86 assembler [Ref. 18]. After linking, when invoked as a transient command from the CCP level of CP/M-86, this file will interactively allow the loading of a CMD file containing a MCORTEX process or multiplexed MCORTEX processes. Only the first real processor entering the MCORTEX environment is to specify that GLOBAL data is to be loaded. Conditional assembly features pervade this code to allow either MCORTEX or MXTTRACE (the diagnostic version) to be loaded. The conditional switch is called "MCORTEX", which is set equal to one (or TRUE) when the MCORTEX version of the loader is to be assembled. The use of the MCORTEX or MXTTRACE LINK86 input option files (APPENDIX F) determine which transient command is generated.
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 area 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.

DSEG
ORG 0000H

;*** MCORTEX / MXTRACE SELECTION

MCORTEX EQU 0 ;*** SET TO ZERO FOR
;*** MXTRACE

;*** ADDRESS TABLE

FCB FCB_NAME EQU 005CH ;*** FILE CONTROL
FCB FCB_EXTENT EQU 005DH ;*** BLOCK
FCB FCB_CP EQU 0068H

INT_ADD_CS INT_INTERRUPT OFFSET EQU 0011H ;*** INTERRUPT CODE
IF MCORTEX
INTRPT_CS EQU 0C4BH ;*** VECTOR
ELSE
INTRPT_CS EQU 001FH ;### 1 ### <---
ENDIF

;*** PURE NUMBER TABLE

FIGHTH_K IF MCORTEX FIGHK_EQU 0080H
NUM_KORE_BLOCKS ELSE
NUM_KORE_BLOCKS EQU 0038H ;### 2 ### <---
ENDIF
NUM_GLOBAL_BLOCKS EQU 0010H
ASCII_EQU '0'
92
**ASCII**

<table>
<thead>
<tr>
<th>ASCII</th>
<th>ECU</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>0</td>
</tr>
<tr>
<td>#</td>
<td>1</td>
</tr>
<tr>
<td>:</td>
<td>2</td>
</tr>
<tr>
<td>SPACE</td>
<td>3</td>
</tr>
<tr>
<td>PERIOD</td>
<td>4</td>
</tr>
<tr>
<td>CR</td>
<td>5</td>
</tr>
<tr>
<td>LF</td>
<td>6</td>
</tr>
</tbody>
</table>

```plaintext
*** CONTROL TRANSFEE CONSTANTS ***********************
```

```plaintext
IF Mcortex
  KORE SP          EQU 0075H
  KORE SS VAL      EQU 0C65H
  KORE DG VAL      EQU 0C49H
ELSE
  KORE SP          EQU 0075H ;### 3 ### <------
  KORE SS VAL      EQU 0C5EH ;### 4 ### <------
  KORE DG VAL      EQU 0C2CH ;### 5 ### <------
ENDIF
```

```plaintext
*** CP/M FUNCTION CONSTANTS ***********************
```

```plaintext
CPM_BLKCaller   EQU 0075H
SYSTEM_RESET    EQU 0009H
CONSOLE_OUTPUT  EQU 0002H
READ            EQU 000AH
PRINT_STRING    EQU 0009H
OPEN_FILE       EQU 0009H
READSEQUENTIAL  EQU 0014H
SET_DMA_OFFSET  EQU 001AH
SET_DMA_PASS    EQU 0033H
ALLOC_MEM ABS   EQU 0038H
FREE_ALL MEM    EQU 003AH
PROGRAM LOAD    EQU 003BH
NOT_FOUND       EQU 00FFH
```

```plaintext
*** MESSAGES ***********************
```

```plaintext
IN_STRING      DB 15
               RE 16
```

```plaintext
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'
                DB 'MCORTEX'
FILE_FORM_ERR_MSG DB 'INCORRECT FILE FORMAT - TRY AGAIN'
START_MSG DB 'MCORTEX SYSTEM LOADEDEST *** ON LINE 93'
```
GLOBAL Q_MSG DB 'Y' TO LOAD, 'N' NOT TO LOAD

*** MCOREX FLOCATION 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

KORE_START
IF MCOREX
KOFL1_BASE
ELSE
KOFL1_BASE
ENDIF
KOFL2
EQU DW ORD PTR KORE_START

IF MCOREX
KOFL1_LENGTH
ELSE
KOFL1_LENGTH
ENDIF
KOFL1_EXT
DB 0

IF MCOREX
KOFF_NAME
ELSE
KOFF_NAME
ENDIF

KOFF2_BASE
DW 003CH

INTERRUPT VECTORS
INT_VCTOR_ADD
TW INTPT_OFFSET, INTPT_CS
DW INT_ADD_CS

INIT OFFSET
DW 003CH

INIT_BASE
DW 0439H

IF MCOREX
INIT_DS_SEG
ELSE
INIT_DS_SEG
ENDIF

INIT_DS_OFFSET
DW 0069H

INIT_IP_OFFSET
DW 0074H

*** CONTROL TRANSFER VARIABLES

KORE_SS
DW KORE_SS_VAL

94
KOPE DS

: *** START CODE SEGMENT: ---------------------------------------------------------------

MOCRTEX_LOAD CSSEG

CALL CLS_SCREH; ; *** SCRAPH CONTROL & LOG ON
CALL MOCRTEX_LOAD ; ; *** MESSAGES
CALL CLS_SCREH; ; ***

C LD
PUSH AX; ; *** INITIALIZATION

; *** 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 CPM_EDOS CALL; ; *** GET INDICATE:

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

GEN_KORE_FCB:
MOV BX, 10; ; *** MOVE 11 CHARACTERS
MOV SI, OFFSET KORE_NAME; ; *** POINT TO FILE NAME
MOV DI, FCB_NAME; ; *** POINT TO FCB NAME
MOV KORE:
MOV AL, [SI + BX]; ; *** GET CHARACTER
MOV [DI + BX], AL; ; *** STORE CHARACTER
DEC BX
JGF 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 CPM_EDOS CALL; ; *** OPEN FILE
CMP AL, NOT_FOUND; ; *** FILE FOUND?
JNF PROCESS_KORE; ; *** FILE FOUND! CONTINUE
JMP NO_FILE;
PROCESS_KORE:
MOV DI, 0;
MOV FCB_DIR [DI], DI; ; *** START WITH REC ZERO

; *** RESERVE MEMORY: ********************************************************************

MOV CL, FREE ALL_MEM; ; *** CP/M PARAMETER
INT CPM_EDOS_CALL; ; *** FREE ALL MEMORY
MOV CL, ALLOC_MEM_ACK; ; *** CP/M PARAMETER
MOV DX, OFFSET KORE_FCB; ; *** CP/M PARAMETER
INT CPM_EDOS_CALL; ; *** ALLOCATE MEMORY
CMP AL, NOT FOUND ;*** MEMORY AVAILABLE
JNE LOAD_MCORTEX ;*** MEMORY AVAILABLE! CONTINUE
JMP NO_MEMORY_ALLOC ;*** GO INDICATE ERROR

*** LOAD_MCORTEX CODE *********************************************/

LOAD_MCORTEX:
  MOV DI,0 ;*** SET DEST. OFFSET
  MOV BP,NUM_KORE_BLOCKS ;*** SET BLOCK COUNTER
  MOV KORE_LOOP:
  MOV DX,PCB ;*** CP/M PARAMETER
  MOV CL,READSEQUENTIAL ;*** CP/M PARAMETER
  INT CP/M_EDOS_CALL ;*** READ IN 128 BYTES
  MOV ES,KORE BASE ;*** SET DESTINATION SEGMENT
  MOV CX,FIGHTH_K ;*** SET BYTE COUNT
  MOV SI,CX
  REP MOVSB ;*** MOVE 128 BYTES
  DEC BP
  JNZ MOVE_KORE_LOOP ;*** DFC BLOCKS TO MOVF

*** LOAD_INITIALIZATION_MODULE *************************************

MOV DI,INIT_OFFSET ;*** SET DEST. OFFSET
MOV DX,PCB ;*** CP/M PARAMETER
MOV CL,READSEQUENTIAL ;*** CP/M PARAMETER
INT CP/M_EDOS_CALL ;*** READ IN 128 BYTES
MOV ES,INIT_BASE ;*** SET DESTINATION SEGMENT
MOV CX,FIGHTH_K ;*** SET BYTE COUNT
MOV SI,CX
REP MOVSB ;*** MOVE 128 BYTES

*** LOAD_GLOBAL_MEMORY *********************************************/

CMP IN STRING+1,PH ;*** SHOULD GLOBAL BE LOADED?
JZ INSTALL_INTERRUPT ;*** IF NOT, SKIP LOAD
MOV DI,0 ;*** SET DEST. OFFSET
MOVF_GLOBAL_LOOP:
  MOV DX,PCB ;*** CP/M PARAMETER
  MOV CL,READSEQUENTIAL ;*** CP/M PARAMETER
  INT CP/M_EDOS_CALL ;*** READ 128 BYTES
  TFST AL,AL
  JNZ INSTALL_INTERRUPT ;*** NO MORE DATA?
  MOV ES,KOR+9ASF ;*** SET DEST. SEGMENT
  MOV CX,FIGHTH_K ;*** SET BYTE COUNT
  MOV SI,CX
  REP MOVSB ;*** MOVE 128 BYTES
  JMP MOVE_GLOBAL_LOOP ;*** IF NOT DONE, DO AGAIN

*** INITIALIZE_INTERRUPT_VECTOR *************************************/

INSTALL_INTERRUPT:
  MOV ES,INTVECTOR_ADD ;*** SET DESTINATION SEGMENT

96
MOV DI,0                  ;*** SET DEST. OFFSET
MOV SI.OFFSET INTERRUPTVECTOR;*** SRC. OFFSET
MOV CX,2                   ;*** 2 WORDS TO MOVE
REP MOVSW AX,AX            ;*** MOV TWO WORDS

;*** READ IN & FILE NAME *********************************************/

READ_A_NAME:
CALL PROCESSOR NAME        ;*** MSG TO INPUT A FILE NAME
MOV DX,OFFSET IN_STRING    ;*** DX <-- BUFFER LOCATION
MOV CL,READ                ;*** CPM PARAMETER
INT CPMBIOS CALL

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

MOV DI,0                   ;*** SET DESTINATION INDEX TO ZERO

CMP IN_STRING+3,COLON;*** IS DRIVE DESIGNATED?
JE SET_DRIVE               ;*** IF YES, PUT DRIVE IN FCB
MOV FCB[DI],DI            ;*** SET DEFAULT DRIVE
MOV SI,2                   ;*** 3RD POSIT IN_STRING, IS 1ST LETTER
JMP FORM_FCB

SET_DRIVE:
MOV AL,IN_STRING+2;*** GET DRIVE LETTER
AND AL,5FH                 ;*** CONVERT TO UPPER CASE
SUB AL,40H                 ;*** CONVERT TO A BINARY NUMBER
MOV FCB[DI],AL            ;*** SET DRIVE
AND AL,0FH                 ;*** LIMIT LINF DRIVE TO A THROUGH O
TEST AL,AL
JNZ INPUT_EXPORB
MOV SI,4                   ;*** 5TH POSIT IN_STRING IS 1ST LETTER

;*** INITIALIZE FILE CONTROL BLOCK ***********************************/

FORM_FCB:
MOV BX,0AH                 ;*** FILL FCB NAME WITH SPACES
MOV AL,SPACEF

FILL_SPACES:
MOV FCB_NAME[BY],AL        ;***
DEC BX                     ;***
JGE FILL_SPACES

MOV FCB_CP[DI],DI          ;*** NEW FILE CURRENT RECORD IS ZERO
MOV FCB_EXTENT[DI],DI      ;*** NEW FILE CURRENT EXTENT IS ZERO

;*** INSTALL FILE CONTROL BLOCK NAME *******************************/

NAME_LOOP:
MOV AL,IN_STRING[SI]       ;*** GET A CHARACTER
CMP AL,PERIOD             ;*** START TYPE ?
JNE FCB_CONT_1            ;*** IF NO, CONTINUE
MOV DI,8
JMP FCB_CONT_2
FCB_CONT_1:
CALL VALID_INPUT
TEST AX, AX
JE INPUT_ERROR_E
MOV FCB_NAME[DI], AL
MOV AX, SI
CMP IN_STRING +1, AL
JB OPEN_PROCESSOR
INC DI
FCB_CONT_2:
INC SI
JMP NAME_LOOP
EXIT_ROUTINE_B:
JMP EXIT_ROUTINE
INPUT_ERROR_B:
JMP INPUT_ERROR
EXIT_ROUTINE:

;*** OPEN THE PROCESSOR FILE ****************************

OPEN_PROCESSOR:
MOV DX, FCP
MOV CL, OPEN_FILE
INT CPM_BDOS_CALL
CMP AL, NOTFOUND
JNF LOAD_PROCESSOR
JMP NO_INPUT_FILE
LOAD_PROCESSOR:
MOV DX, FCP
MOV CL, PROGRAM_LOAD
INT CPM_BDOS_CALL
;*** DATA SEGMENT IN AX

;*** SET UP THE INITIALIZATION STACK ****************************

;*** CAUTION *** CAUTION *** CAUTION *** CAUTION *******
;*** This code is highly dependent upon input of PL/I  ***
;*** CMD file with CS header first and data header  ***
;*** second. This is the normal situation and should  ***
;*** cause no difficulty. Also this code is highly  ***
;*** dependent upon the location of the initialization  ***
;*** module stack and the location of the DS and IP  ***
;*** values within that stack. Changes in stack  ***
;*** location or organization should be reflected here.  ***
;*** CAUTION *** CAUTION *** CAUTION *** CAUTION *******

EXIT_ROUTINE:
MOV FS,INIT DS SEG
MOV FS,INIT DS OFFSET
MOV ES:[R],AX
MOV DX,0
MOV AX,INIT IP OFFSET
MOV ES:[R],AX
MOV CL,SET_DMA_BASE
MOV AX,AX
INT CPM BDOS CALL
MOV CL,SET_DMA_OFFSET
MOV DX,EIGHTH_K
INT CPM BDOS CALL

;*** TRANSFER CONTROL TO MCORTEX *****************************/

MOV SP, KORE SP
MOV BP, SP
MOV SS, KORE SS
MOV AX, DS
MOV ES, AX
MOV DS, KORE DS
JMPF ES:KORE

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

VALID INPUT:
CMP AL, SLASH
JE IS VALID
CMP AL, ASCII_0
JB NOT VALID
CMP AL, ASCII_9
JBE IS VALID
AND AL, 5FH
CMP AL, ASCII_A
JB NOT VALID
CMP AL, ASCII_Z
JBE IS VALID
NOT VALID:
MOV AX, 0
IS VALID:
RET

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

NO FILE:
CALL CLR SCREEN
MOV DX, OFFSET NO_FILE_MSG
JMP MSG_OUTPUT

NO MEMORY_ALLOC:
CALL CLR SCREEN
MOV DX, OFFSET NO_MEMORY_MSG

99
MSG_OUTPUT:
MOV CL,PRINT_STRING ;*** CP/M PARAMETER
INT CPM_BDOS_CALL ;*** SEND CHAP TO CONSOLE
CALL CLR_SCREEN
MOV CL,SYSTEM_RESET ;*** CP/M PARAMETER
MOV DL,0 ;*** RELEASE MEMORY
INT CPM_BDOS_CALL ;*** EXIT TO CP/M

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

CLR_SCREEN:
MOV CL,CONSOLE_OUTPUT ;*** ISSUE CARRIAGE RETURN
MOV DL,CR ;***
INT CPM_BDOS_CALL ;***
MOV DL,0 ;***
LINE_FEED:
MOV DL,LF ;***
MOV CL,CONSOLE_OUTPUT ;***
INT CPM_BDOS_CALL ;***
DEC DI ;***
JNE LINE_FEED ;***
RET

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

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

MCORTEX_LOAD:
MOV DX,OFFSET START_MSG
CALL SEND_MSG
RET

PROCESSORS NAME:
MOV DX,OFFSET P_NAME_MSG
CALL SFND_MSG
RET

IN 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
MO7 DX,OFFSET NO_IN.FILP.MSG
EXIT_ERR:
CALL SFND_MSG
CALL CLR_SCREEN
JMP RPAD_A_NAME

END
APPENDIX D

GATE MODULE SOURCE CODE

SYSDFF.PLI and GATEM/T.A86 files are contained in this appendix. PL/I-86 entry variables in SYSDFF.PLI provide a "gateway" to the MCORTFX (kernel) supervisor via GATEMOD or GATETRC. Also contained in SYSDFF.PLI are constant (or symbolic) definitions that are used by the demonstration processes contained in Appendix F. Note that system reserved constants, used by MCORTFX kernels and the V301K Driver and Packet Processor are also contained in this file.

GATEM/T.A86 is assembled, and as a relocatable object file, is linked with MCORTFX processes to set up the PL/I-86 to PL/M-86 parameter passing interface.

A conditional assembly switch "GATEMOD" allows for assembly of a GATEMOD or GATETRC version.
This section of code is given as a PLI file to be \(*\) included with MCORTEX user programs. ENTFY \(*\) declarations are made for all available MCORTEX \(*\) functions. \(*\) 

**DECLARE**

```
advance FNTRY (BIT (8)),
/* advance (event_count_id) */

await FNTRY (BIT (8), BIT (16)),
/* await (event_count_id, awaited_value) */

create_evc FNTRY (BIT (8)),
/* create_evc (event_count_id) */

create_proc FNTRY (BIT (8), BIT (8),
    BIT (16), BIT (16), BIT (16),
    BIT (16), BIT (16), BIT (16)),
/* create_proc (processor_id, processor_priority, */
/*    stack_pointer_highest, stack_seg, ip */
/*    code_seg, data_seg, extra_seg) */

create_seq FNTRY (BIT (8)),
/* create_seq (sequence_id) */

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

read FNTRY (BIT (8)) RTURNS (BIT (16)),
/* read (event_count_id) */
/* :RTURNS current_event_count */

ticket FNTRY (BIT (8)) RTURNS (BIT (16)),
/* ticket (sequence_id) */
/* :RTURNS unique_ticket_value */

define_cluster FNTRY (bit (16)),
/* define_cluster (local_cluster_address) */

distribution_map FNTRY (bit (8), bit (8), bit (16)),
/* distribution_map (distribution_type, id, */
/*    cluster_addr) */
```
add2bit16 ENTRY(BIT(16),BIT(16)) : ERETURN5 (BIT(16));
    /* add2bit16 ( a_16bit_, another_16bit_ ) */
    /* ERETURN5 a_16bit_ + another_16bit_ */

replace

*/--------------------------------------------------------------*

    *** EVC$IN's ***

    (1) USFR
    /*
    T-ACK_IN by '01'b4,
    TRACK_OUT by '02'b4,
    MISSILE ORDER_IN by '03'b4,
    MISSILE ORDER_OUT by '04'b4,
    */

    /* (2) SYSTEM */
    EBF_READ by '0c'b4,
    EBF_WRITE by '0d'b4,

*/--------------------------------------------------------------*

    *** SEQUENCER NAMES ***

    (1) USFR
    /*
    */

    /* (2) SYSTEM */
    EBF_WRITE_REQUEST by 'ff'b4,

*/--------------------------------------------------------------*

    *** SHARED VARIABLE POINTERS ***

    (1) USFR
    /*
    */

    /* (2) SYSTEM */
    block_ptr_value by '8000'b4,
    xmit_ptr_value by '8078'b4,
    rcv_ptr_value by '8666'b4,
    END_RESERVE by 'FFFF'b4;

;*********************************************************************
/* GATEMOR / CATETRC File GATEM/T.a86 BREWER 1 SEP 94 */
104
This module is given to the user in obj 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 GATMOD or GATETRC depending on the value of GATMOD at the code start.

This module reconciles parameter passing anomalies between MCOREX (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 ADD2PIT16 function does not make calls to MCOREX. It's purpose is to allow the addition of two unsigned 16 bit numbers from PL/I programs.

DSEG

GATEMOD ECU 0 ;*** SET TO ZERO FOR GATETRC
                    ;*** SET TO ONE FOR GATMOD
PUBLIC ADVANCE ;*** THESE DECLARATIONS MAKE THE
PUBLIC AWAIT ;*** GATEKEEPER FUNCTIONS VISIBLE
PUBLIC CREATEPROC
PUBLIC CPATE_EVC ;*** TO EXTERNAL PROCESSES
PUBLIC CREATESEQ
PUBLIC CP4TF_SEQ
PUBLIC PREEMPT
PUBLIC READ
PUBLIC TICKFT
PUBLIC DEFINF CLUSTFR
PUBLIC DISTRIBUTION_MAP
PUBLIC ADD2PIT16

AWAIT_IND ECU 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 ;*** MCOREX
TICKFT_IND EQU 4
READ_IND EQU 5
CREATE_PIOC_IND EQU 6
PREEMPT_IND EQU 7
DEFINE_CLUSTER_IND EQU 8
DISTRIBUTION_MAP_IND EQU 9

IF GATMOD
GATEKEEPER_IP DW 0036H

105
GATFKEFPE: CS DW 0B4CH
ELSE
GATFKEFPE: IP DW 036RH
GATFKEPEP: CS T 0B4CH
FNDTP
GATFKEPFF PER FQO DwORt PTP GATFKEPEPE: IP
CSEG
*** AWAIT *** AWAIT *** AWAIT *** AWAIT *** AWAIT *** AWAIT **********/

AWAIT:
PUSH ES
MOV SI,2[RX]
MOV BX,[RX]
MOV AL,AWAIT_IND
PUSH AX
MOV AL,[RX]
PUSH AX
MOV AX,[SI]
PUSH AX
PUSH AX
CALLF GATFKEPEG
POP ES
RET

*** ADVANCE *** ADVANCE *** ADVANCE *** ADVANCE *** ADVANCE **********/

ADVANCE:
PUSH ES
MOV BX,[RX]
MOV AL,ADVANCE_IND
PUSH AX
MOV AL,[RX]
PUSH AX
PUSH AX
PUSH AX
PUSH AX
CALLF GATFKEPFF
POP ES
RET

*** CREATE_EVC *** CREATE_EVC *** CREATE_EVC **********/

CREATE_EVC:
PUSH FS
MOV BX,[RX] ;BX <-- PTR TO NAME OF EVENT
MOV AL,CREATE_EVC_IND
PUSH AX ;IN <-- CREATE_EVC INDICATOR
MOV AL,[RX]
PUSH AX
MOV AL,[RX]
PUSH AX
PUSH AX
CALLF GATFKFFFFFF
POP FS

PFT

;*** CREATE_SEQ *** CREATE_SEQ *** CREATE_SEQ ****************************/

CREATE_SEQ:
PUSH FS
MOV BX,[RX] ;BX <-- PTR TO NAME OF SEQ
MOV AL,CREATE_SEQ_IND
PUSH AX ;IN <-- CREATE_SEQ INDICATOR
MOV AL,[RX]
PUSH AX
MOV AL,[RX]
PUSH AX
PUSH AX
CALLF GATFKFFFFFF
POP FS

PFT

;*** TICKET *** TICKET *** TICKET *** TICKET *** TICKET *** TICKET ***/

TICKET:
PUSH ES
PUSH FS
MOV CX,SP
MOV BX,[RX]
MOV AL,TICKET_IND
PUSH AX
MOV AL,[RX]
PUSH AX
PUSH AX
PUSH SS
PUSH CX
CALLF GATFKFFFFFF
PUSH BX
POP ES

PFT

107
MCRTFX = TFX/TIC [code[ar[82]], data[ah[439]]]

CIPROC input option file

clproc = sysini1 [code[ab[439]], data[ab[82]], m[0], ad[82]], map[all]],
        sysdev,
        asmrout,
        satemod

TRACKER input option file

tracker = trkinit [code[ab[439]], data[ab[6ff], m[0], ad[82]], map[all]],
          trkdeletc,
          satemod

MSLFACT input option file

MSLFACT = slract [code[ab[439]], data[ab[412], m[0], a1[82]], map[all]],
          slract,
          satemod

121
APPENDIX F

LINK66 Input Option Files

The INPUT option directs LINK66 to obtain further command line input from an indicated file. This reduces the amount of interactive typing needed to link various modules together. In essence, the input file is a batch file scanned by LINK66. For example, the modules shown in CIPROC.INP are linked with the command: LINK66 CIPROC[I], where I denotes that CIPROC.INP contain the actual files to be linked. The name appearing on the lefthand side of the equal sign in the LINK66 option files is the name assigned to the CMD module. Therefore, LINK66 CIPROC[I] produces the CMD module CIPROC.CMD. Details concerning this procedure may be found in [Ref. 18].
if ( (binary(buffer_ub) -
  binary(buffer_lb)) >= buffer_length) then
  do:
    k = add2bit16(buffer_lb, one);
    call await (MISSILE_ORDER_OUT, k);
  end:
end; /* do i */

end mslorder;

******************************************************************************
* TPKPRT is the main module of a process at Cluster 2                  *
* that simulates the consumption of track detection data.              *
* It signals its consumption by advancing *eventcount                  *
* TRACK_OUT.  This module is linked as shown in                        *
* C2USEP'S.INP or C2USER/T.INP of Appendix F.                          *
* PL/I-26 Source File Name : TRKRPRT.PLI                               *
******************************************************************************

trkrprt: procedure ;
%replace

    infinity by 32767,
    one by '0001'b4;

%include 'sysdef.pli';

DECLARE

  i fixed bin (15),
  k bit (16) static init ('0000'b4);

/* end DECLARATIONS */

/* main */
do i = 0 to infinity:
  k = add2bit16(k, one);
  call await (TRACK_IN, k);
/* consume */
call advance (TRACK_OUT);
end: /* do i */
end trkrprt;
MSLORDER is the main module of a producing process at Cluster 2 that simulates issuing missile orders. It signals the next iteration of missile orders by advancing eventcount MISSILE_ORDER_IN. The consumer is MSLREACT at cluster 1. This module is linked as shown in C2USERS.INP or C2USER/T.INP in Appendix F.

PL/I-86 Source File Name: MSLORDER.PLI

Contained in:

MCO/TEX Command Module Name: C2USERS.CMD
MXTR:CF Command Module Name: C2USER/T.CMD

**********************************************************
mslorder:  procedure:
%

%replace

.infinity by 32767,
one by '0021'b4,
.buffer_length by 50;

#include 'sysdef.pli';

DECLARE

i fixed bin (15),
(k,buffer_ub,buffer_lb) bit (16);

/* end DECLARATIONS */

/* main */
do i = 2 to infinity:

/* simulation of missile order */

call advance (MISSILE_ORDER_IN);
buffer_ub = read('MISSILE_ORDER_IN');
put skip(2) edit ('Eventcount value = ',
.buffer_ub(a.h4(5)));
buffer_lb = read ('MISSILE_ORDER_OUT');
put skip(2) edit ('Eventcount value = ',
.buffer_lb(a.h4(5)));
if (binary(buffer_ub) -
    binary(buffer_lb)) >= buffer_length then
  do:
    k = add2bit1F(buffer_lb.one);
    call await (TRACK_OUT, k);
  end;
end; /* do FOPFRVF */
end trkdetect;

******************************************************************************
* C2UNIT is the initialization module for the Cluster 2
* processes that are multiplexed on SBC 2. This module is
* linked as shown in C2USFRS.INP or C2USFR/T.INP in
* Appendix F.
* PL/I-96 Source File Name: C2UNIT.PLI
*
******************************************************************************
c2_users_init: procedure options (main);

   %include 'sysdef.pli';

   /* begin */
   /* missile order */
   call create_proc ('03'b4, 'fc'b4,
                    '020'b4, '06ff'b4, '029'b4,
                    '0439'b4, '06ff'b4, '06ff'b4);
   call create_proc ('04'b4, 'fc'b4,
                    '040'b4, '06ff'b4, '06de'b4,
                    '0439'b4, '06ff'b4, '06ff'b4);
   call await ('fe'b4, '01'b4);
end c2_users_init;
trkdetect: procedure;

%replace

FOREVER
one
buffer_length

%include 'sysdef.pli';

DECLARE

1 fixed bin (15),
(k,buffer_ub,buffer_lb) bit (16);

end DECLARATIONS */

/* main */

d0 i = 0 to 32000;

/* simulation of track input data*/

/* Input from real-time sensor here */

call advance (TRACK_IN);
buffer_ub = read (TRACK_IN);
put skip(2) edit ('Eventcount value = ',
buffer_ub)(a,b4(5));
buffer_lb = read (TRACK_OUT);
put skip(2) edit ('Eventcount value = ',
buffer_lb)(a,b4(5));
DECLAIM

i fixed sin (16),
! bit '16) static init ('066b4);

	/* end DECLARATIONS */

	/* main */

do i = 0 to infinity:

	k = add2bit16(k, one);

call await (MISSILE_ORDER_IN, k);

	/* consume() */

call advance (MISSILE_ORDER_OUT);

doi; /* do i */
end msltrain;

*******************************************************************************
* TRKDINIT is the initialization module for the process *
* TACKER (CMD filename). It is linked as shown in *
* TRACKER.INP or TRKFR/T.INP of Appendix F. *
* *
* PL/I-P6 Source File Name : TRKDINIT.PLI *
* *
*******************************************************************************

trkdinit: procedure options (main);

#include 'sysdef.pli';

	/* begin */

call create_proc ('01'b4, '06b4,
	'0900'b4, '06ff'b4, '0023'b4,
	'0439'b4, '06ff'b4, '06ff'b4);

call await ('fe'b4, '01'b4);
end trkdinit;
MSLTINIT is the initialization module for the process that simulates the training of a missile launcher as a result of orders received from the MSLODF process of Cluster 2. This module is linked as shown in MSLFACT.INP or LAUNCH/T.INP in Appendix F.

PL/I-86 Source File Name: MSLINIT.PLL

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

msltinit: procedure options (main);

%include 'sysdef.pll';

/* begin */
call create_proc ("92'b4, "fc'b4, 0600'b4, "4d8'b4, "023'b4, 0439'b4, 74d8'b4, "04d'b4);
call await ("fe'b4, "01'b4);

end msltinit;

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

MSLTRAIN is the main module of a process that responds to commands issued by MSLODF. It is a consumer of missile orders. It signals its use of a command by advancing distributed eventcount MISSILE_ORDER_OUT. It is linked as shown in MSLFACT.INP or LAUNCH/T.INP of Appendix F.

PI/I-86 Source File Name: MSLTRAIN.PLL

MCO-TEX Command Module Name: MSLFACT.CMD

MXPACE Command Module Name: LAUNCH/T.CMD

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

msltrain: procedure:

%replace

infinity by 32767,
one by '2001'b4;

%include 'sysdef.pll';
Both Clusters - there is no actual computations being done by any MCO:TEX process, so Ethernet Request Packets and Ethernet Packets are being generated at the fastest possible rate. Any possible timing problems would be exposed by this demonstration process. None were noted, and the processes performed as expected.

The system console shown in Figure 11 is used to monitor using DPT96 changes in GLOBAL data and shared memory structures. A process that automatically provides diagnostic and display support is under development for RTC STAR. This process will execute under CP/M-86 on single board computer 1 in each cluster. Source code for the demonstration model, except for link86 input option files and the NI3013 Driver, follows. The input option files and NI3010 Driver are contained in Appendices F and Y respectively.
FIGURE 11 - Demonstration Model
APPENDIX E

DEMONSTRATION PROGRAM SOURCE CODE

The model of processes that demonstrates the distributivevity of MCORTFX over Ethernet is illustrated in Figure 11. The interactions that are occurring at each cluster are as follows.

Cluster 1 - The MSLFACT (missile launcher reaction) process and TRACKER (target tracking) process get a ticket through the kernel (SYSTEM$IO) to write to the Ethernet Request Block. This corresponds to user transparent simultaneous requests for Ethernet access. The NI3216 Driver and Packet Processor is processing Ethernet Request Packets and Ethernet packets. All processes are competing for access to GLOBAL data via the kernel.

Cluster 2 - the MSLOPER (generates missile orders) and TRKRPRT (track reporting) processes are multiplexed on the real processor and are scheduled and blocked based on the interaction with the Cluster 1 processes. The NI3216 Driver and Packet Processor performs the same function as that in Cluster 1. The code is identical with the exception of the initialization module. Recall that this module in each cluster is responsible for the creation of eventcounts and sequencers, as well as calls to PRFINESCLUSTER and DISTRIBUTIONS$P, which are cluster and eventcount-dependent distribution of procedures.
MOV AL, DISTRIBUTION MAP_IND
PUSH AX ; IN <-- DISTRIBUTION MAP_IND
PUSH AX ; BYT <-- UNUSED WORD
PUSH AX ; WORD <-- UNUSED WORD
PUSH SS ; MAP_PTR_SFO <-- SS
PUSH CX ; MAP_PTR_OFFSET <-- DATA_PTR
CALLF GATPACK
ADD SP, 4
POP FS
RET

;*** ADD2BIT16 *** ADD2BIT16 *** ADD2BIT16 *** 'DD2BIT16 **/
ADD2BIT16:
MOV SI, [BX] ; SI <-- PTR TO BIT(16)#1
MOV BX, 2[BP] ; BX <-- PTR TO BIT(16)#2
MOV BX, [BP] ; BX <-- BIT(16)#2
ADD BX, [SI] ; BX <-- BIT(16)#1 + BIT(16)#2
POP FS
END
POP FS
RET

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

PREEMPT:

PUSH ES
MOV BX, [BX] ;BX <-- PTR TO NAME OF PROCESS
MOV AL, PREEMPT_IND
PUSH AX
MOV AL, [BX] ;N <-- PREEMPT INDICATOR
PUSH AX
PUSH AX ;BYTE <-- PREEMPT PROCESS NAME
PUSH AX ;WORDS <-- UNUSED WORD
PUSH AX ;PTR_SEG <-- UNUSED WORD
PUSH AX ;PTR_OFFSET <-- UNUSED WORD
CALL GATEKEEPER
POP ES
RET

;*** DFFINF_CLUSTER *** DFFINF_CLUSTER *** *** */

DEFINE_CLUSTER:

PUSH ES
MOV BX, [BX] ;BX <-- PTR TO LOCAL$CLUSTER$ADDR
MOV AL, DFFINF_CLUSTER_IND
PUSH AX ;N <-- DEFINE_CLUSTER_IND
PUSH AX ;BYTE <-- UNUSED WORD
PUSH WORD PTR [BX] ;WORDS <-- LOCAL$CLUSTER$ADDR
PUSH AX ;PTR_SEG <-- UNUSED WORD
PUSH AX ;PTR_OFFSET <-- UNUSED WORD
CALL GATEKEEPER
POP ES
RET

;*** DISTRIBUTION_MAP *** DISTRIBUTION_MAP *** ***/

DISTRIBUTION_MAP:

PUSH ES
MOV SI, 4[BP] ;SI <-- PTR TO GROUP ADDRESS
PUSH WORD PTR [SI] ;STACK THE GROUP ADDRESS
MOV SI, 2[BP] ;SI <-- PTR TO ID OF MAP_TYPE
MOV AH, [SI]
MOV SI, [BX] ;SI <-- PTR TO MAP_TYPE
MOV AL, [SI] ;AL <-- MAP_TYPE
PUSH AX ;STACK IT AND MAP_TYPE
MOV CX, SP ;POINTER TO DATA
PUSH ES
PUSH ES ; EVENT COUNT DUMMY STORAGE
MOV CX,SP ; POINTER TO EVENT COUNT
MOV RX,[BX] ; BX <-- PTR TO EVENT NAME
MOV AL,READ_IND
PUSH AX ; N <-- READ INDICATOR
MOV AL,[RX]
PUSH AX ; BYT <-- EVENT NAME
PUSH AX ; BYT <-- UNUSED WORD
PUSH SS ; PTR TO EVENT NAME
PUSH CX ; PTR_OFFSET <-- EVENT COUNT POINTER
CALLF GATFPKFPFR
POP BX ; RETRIEVE EVENT COUNT
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,10[FX] ; 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[FX] ; SI <-- PTR TO PROCESS SP
PUSH WORD PTR [SI] ; STACK PROCESS SP
MOV SI,2[FX] ; SI <-- PTR TO PROCESS PRIORITY
MOV AL,[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 ; PTR TO DATA
MOV AL,CREATE_PROC_IND
PUSH AX ; N <-- CREATE PROCESS IND
PUSH AX ; BYT <-- UNUSED WORD
PUSH AX ; WORDS <-- UNUSED WORD
PUSH SS ; PROC_PTR SEGMENT <-- STACK SEG
PUSH CX ; PROC_PTR OFFSET <-- DATA POINTER
CALLF GATPKPFPR
ADD SP,14 ; REMOVE STACKED DATA

108
C2PROC input option file

C2PROC = [code=ab[439]], data=ab[902], m[0], a1[921], map[all], sysdev, asmout, qatemod

C2USERS input option file

C2users = [code=ab[439]], data=ab[6ff], m[0], a1[921], map[all], mslorder, trkrprrt, zatemod

MXTACEF input option file

MXTACEF = TFX/TAC [code=ab[46C]], data=ab[4BC]]

C1PROC/T input option file

c1proc/t = [code=ab[439]], data=ab[803], m[?], a1[521], map[all], sysdev, asmrout, qatemod
TPFL/T input option file

```fortran
trkerr/t =
trkinit[cd[439]],data[ab[6ff],m[0],ad[92]],map[all11],
trkdetect,
matetr
```

LAUNCH/T input option file

```fortran
launch/t =
msltinit[cd[439]],data[ab[439],m[2],ad[92]],map[all1],
mslttrain,
matetr
```

C2PROC/T input option file

```fortran
c2proc/t =
sysinit2[cd[439]],data[ab[6ff],m[0],ad[92]],map[all1],
sysdev,
asmrout,
matetr
```

C2USPH/T input option file

```fortran
c2user/t =
c2uinit[cd[439]],data[ab[6ff],m[9],ad[92]],map[all1],
msorder,
trkfrpt,
matetr
```

123
APPENDIX G

LEVEL II MCORTEX SOURCE CODE

The LFVEL II source code, written in PL/M-86, is contained in file LFVEL2.SRC. Due to the conditional compilation switches contained in the code, it should be compiled for either the MCORTEX version or MXTACE version. Files are provided to be used with the SUBMIT utility [Ref. 19]. The MCORTEX version of LFVEL II is compiled by using the SUBMIT file L2CMPM.CSD (LFVEL2 compile, MCORTEX).

LLEVEL II is one of the relocatable code modules shown in the SUBMIT file LNKKM.CSD, which is used to link the modules together for KORE.CPS. After linking, the resultant file must be located using the LOCS6 utility. This is provided for in the SUBMIT file LOCKM.CSD (locate MCORTEX). The file KORE is created and becomes KORE.OPS after transfer to the multi-user CP/M-86 system. KORE.OPS is loaded by MCORTEX.COM under the CP/M-86 operating system. Memory maps for KORE.OPS and KORE.TEC are provided at the end of Appendix G. The map information comes from KORE.MP2 after compiling, linking, and locating the applicable files.

In the source listing for LFVEL2.SRC, the executable code must begin in column 7 (see L2CMPM.CST). It appears left justified in this listing due to thesis format requirements.
REM#S:  

!!! CAUTION !!! !!! CAUTION !!! !!! CAUTION!!!  
IF NEW USER SERVICES ARE ADDED TO THIS MODULE  
OF CHANGES ARE MADE TO EXISTING ONES, MAKE  
SURE THE LOCATOR MAP (FILE: KORE.MP2) IS CHECKED TO SEE IF THE LOCATION OF 'GATESKEEPER' HAS NOT CHANGED. THE ABSOLUTE ADDRESS OF THIS PROCEDURE HAS BEEN SUPPLIED TO THE GATESMODULE IN FILE: GATF.SRC. IF IT HAS CHANGED THE NEW ADDRESS SHOULD BE UPDATED IN FILE: GATF.SRC AND RECOMPILED. ALL USER PROCESSES WILL HAVE TO BE RELINKED WITH FILE: GATE.OBJ AND RELOCATED.

LITERAL DECLARATIONS GIVEN AT THE BEGINNING OF SOME 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 LOCATESVC AND CREATESVC IS AN EXAMPLE. TO CHANGE IT IN ONE MODULE AND NOT THE OTHER WOULD KILL THE CREATION OF ANY NEW EVENTCOUNTS BY THE OS.

CONDITIONAL COMPIILATION COMMANDS ARE USED TO PRODUCE TWO VERSlONS OF THE MCORTEX OPERATING SYSTEM. 'MCORTEX' IS THE VERSION WITHOUT ANY I/O PERTAINING TO ENTRY OF OS PRIMITIVES. WITH THIS VERSION IT IS EXPECTED THAT THE USER HAS COMPLETED DEBUGGING OF USER PROCESS CODE AND THIS IS NO LONGER NECESSARY. IN CONFRST, THE COFF BRACKETED BY 'NOT MCORTEX' IS THE CODE
FOR THE TRACIE VERSION OF MCORTEX KNOWN AS
'MXTRACE.' THIS VERSION PROVIDES DIAGNOSTIC
'HOOKS' INTO THE OS AND SHOULD BE USED DURING
THE CODE DEVELOPMENT STAGES.

 L23MODULF: DO:

DECLARE
MAX$CPU LITERALLY '10',
MAX$VPS$CPU LITERALLY '10',
MAX$CPU$M$VPS$CPU LITERALLY '10',
FALSE LITERALLY '0',
READY LITERALLY '1',
RUNNING LITERALLY '3',
WAITING LITERALLY '7',
TRUE LITERALLY '119',
NOT$FOUND LITERALLY '255',
PORT$CA LITERALLY '9CAH',
RESET LITERALLY '0',
EN&T LITERALLY '0',
FRBS$BLOCK$LENGTH LITERALLY '20',
FVC$TYPE LITERALLY '0',
FRBS$HEAD LITERALLY '0CH',
FRBS$WRITE LITERALLY '2F8H',
FRBS$WRITE$REQUEST LITERALLY 'CF8H',
INT$RETURN LITERALLY '77H';

DECLARE P-D S STRUCTURE
(CPU$NUMBER BYTE,
VPS$START BYTE,
VPS$END BYTE,
127
DECLARE VP为您提供CPU
LAST$CPU
COUNTER
EXTERNAL;

DECLARE VPM(MAX$CPU$MAX$VPS$CPU) STRUCTURE
(VPSID BYTE,
STATE BYTE,
VPSPRIORITY BYTE,
EVC$HEADER BYTE,
EVCSAVEVALUE WORD,
SPSRG WORD,
SS$REG WORD) EXTERNAL;

DECLARE LOCAL$CLUSTP$ADDR WORD EXTERNAL;

DECLARE EVENTS BYTE EXTERNAL;

DECLARE EVCS$T$P$T (100) STRUCTURE
(EVCS$NAME BYTE,
VALUE WORD,
REMOTE$ADDR WORD,
THREAD BYTE) EXTERNAL;

DECLARE SECUF$ENRS BYTE EXTERNAL;

DECLARE SEQF$ENRS (100) STRUCTURE
(SEQ$NAME BYTE,
SEQ$VALUE WORD) EXTERNAL;

DECLARE NR$VPS MAX$CPU BYTE EXTERNAL,
NR$VPS BYTE EXTERNAL,
HD$INT$FLAG MAX$CPU) BYTE EXTERNAL,
GLOBAL$LOCK BYTE EXTERNAL;

/*宣言外部过程引用*/

VPSCHEDULE: PROCEDURE EXTERNAL; END;
/*在文件'SCHD.ASM'*/

$1F NOT $C0RTEX

DECLARE

MSG16($) BYTE INITIAL('ENTERING PREEMPT',13,10,')',
MSG17($) BYTE INITIAL('ISSUING INTERRUPT!',13,10,')',
MSG18($) BYTE INITIAL('ENTERING WAIT',10,13,')',
MSG19($) BYTE INITIAL('ENTERING ADVANCE',10,13,')',
MSG21($) BYTE INITIAL('ENTERING CREATE$RFC FOR %'),
MSG23($) BYTE INITIAL('ENTERING HEAD FOR EvC: %'),
MSG24($) BYTE INITIAL('ENTERING TICKET',13,10,')',
MSG25($) BYTE INITIAL('ENTERING CREATE$FO %'),
MSG26($) BYTE INITIAL('ENTERING CREATE$FOC',10,13,')',
MSG27($) BYTE INITIAL('ENTERING GATEKEEPER N= %');

DECLARE

C LITERALLY 'ODF',
LF LITERALLY 'OAH';

$END}$

/*8201***************************************************************************/
/*8201***************************************************************************/
/*8201***************************************************************************/
/** GATEKEEP$PP Procedure BREWER 7-18-84 ***/
/***********************************************************************************************/
/* THIS PROCEDURE IS THE ENTRY INTO THE OPERATING SYSTEM FROM THE USER DOMAIN. THIS IS THE */
/* ACCESS POINT TO THE UTILITY/SERVICE ROUTINES AVAILABLE TO THE USER. THIS PROCEDURE IS CALLED BY THE */
/* GATE MODUL$ WHICH IS LINKED WITH THE USER PROGRAM. */
/* IT IS THE GATE MODUL$ WHICH PROVIDES TRANSLATION FROM THE USER DESIRED FUNCTION TO THE FORMAT REQUIRED */
/* FOR THE GATEKEEPER. THE GATEKEEPER CALLS THE */
/* DESIRED 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 IN-* /
/* DIRECT CALL. * /
/* * /
/* THE PARAMETER LIST IS PROVIDED FOR CONVENIENCE AND * /
/* REPRESENTS NO FIXED MEANING, EXCEPT FOR 'N'. * /
/* N FUNCTION CODE PROVIDED BY GATE * /
/* BYTE BYTE VARIABLE FOR TRANSLATION * /
/* WORDS WORD " " " " * /
/* PTR POINTER VARIABLE FOR TRANSLATION * /
/* */

GATEKEEPER: PROCEDURE(N, PYT, WORDS, PTR) REENTRANT PUBLIC;

DECLARE

(IN, BYT) PYT;
WORDS WORD,
PTR POINTER;

/* 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 NOT MCEOTEX

IF N < 10 THEN DO;
CALL OUT$LINE(MSG27);
CALL OUT$NUM(N);
CALL OUT$CHAR(CR);
CALL OUT$CHAR(LF);
END;

ENDIF

DO CASE ":
CALL AWAIT(PYT, WORDS); /* N */
CALL ADVANCE(PYT); /* V */
CALL CREATE$VEC(PYT); /* 1 */
CALL CREATE$SEQ(PYT); /* 2 */
CALL TICKET(PYT, PTR); /* 3 */
CALL READ(PYT, PTR); /* 4 */
CALL CREATE$PROC(PYT); /* 5 */
CALL PREEMPT(PYT); /* 6 */
CALL DEFINESCLUSTER(WORDS); /* 7 */
CALL DISTRIBUTIONSMAP(PTR); /* 8 */

SIF NOT MCEOTEX

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE *****/

136
### MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******

- CALL OUT$CHAR(PTK);
- CALL OUT$LINE(PTK);
- CALL OUT$NUM(PTK);
- CALL OUT$DNUM(WORDS);
- CALL IN$CHAR(PTK);
- CALL IN$NUM(PTK);
- CALL IN$DNUM(PTK);

**ENDIF**

**FND:** /* CASE */

**FND:** /* GATE*KEFPFR */

### **283**

/* CREATE EVENTCOUNTER PROCEDURE FEIKE: $-16-$84 */

/* CREATE EVENTCOUNTER FOR INFR-PROCESS SYNCHRONIZATION. */
/* EVENTCOUNTER IS INITIALIZED TO 0 IN THE EVENTCOUNTER TABLE. */
/* CREATE EVENTCOUNTER: PROCEDURE (NAME) REENTRANT PUBLIC: */

**DEFINE** NAME BYTE;

**IFDEF** NOT MCORTEX

### MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******

- CALL OUT$LINE(GM$21);
- CALL OUT$NUM(NAME);
- CALL OUT$CHAR(CR);
- CALL OUT$CHAR(LF);

**ENDIF**

/* ASSERT GLOBAL LOCK */

DO WHILE LOCKSET(GLOBAL$LOCK, 119): END;

**IFDEF** /* THE EVENTCOUNTER DOES NOT ALREADY EXIST */

LOCATE EVENTCOUNTER = NOT$FOUND THEN DO;
/* CREATE THE EVENTCOUNTER ENTRY BY ADDING THE */
/* NEW EVENTCOUNTER TO THE END OF THE EV$TABLE */
EVENT$TABLE(EVENTS).NAME = NAME;
EVENT$NAME = NAME;
EVENT$VALUE = 0;
EVENT$ADDR = LOCAL$CLUSTER$ADDR;
EVENT$ADDRESS = 255;
/* INCREMENT THE SIZE OF THE EV$TABLE */
EVENTS = EVENTS + 1;
END; /* CREATE THE EVENTCOUNTER */
/* RELEASE THE GLOBAL LOCK */
**READ PROCEDURE**

BREWER 8-16-24

**-------------**

/* THIS PROCEDURE ALLOWS USERS TO READ THE PRESENT VALUE */
/* OF THE SPECIFIED EVENT'S COUNT WITHOUT MAKING ANY */
/* CHANGES. 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. */
**-------------**

**READ:**

*PROCEDURE:* (EVCS$NAME, RET$SPTR) EXTERNAL PUBLIC

DECLARE:

  EVCS$NAME BYTE,
  EVC$NAME INDEX BYTE,
  RET$SPTR POINTER,
  EVCS$VALUE$SET BASED RET$SPTR WORD;

/* INITIALIZE gloBal LOCK */
DO WHILE LOCK=SET( @GLOBAL$LOCK, 119); END;

SIF NOT MCORTEX

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/
  CALL OUT$LINE( @MSG23 );
  CALL OUT$NAME( EVCS$NAME );
  CALL OUT$CHAR( CR );
  CALL OUT$CHAR( LF );

ENDIF

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

/* OBTAIN VALUE */
  EVCS$VALUE$SET = FVC$TBL( EVC$NAME INDEX ).VALUE;

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

/*9368*******************************************************************/
/* WAIT PROCEDURE
/*-------------------------------*/
/* CENTTERNAL ACCES.S 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(EVC$ID, AWAITED$VALUE) REENTRANT PUBLIC:

    DFCLR
    AWAITED$VALUE    WORD,
    (EVC$ID, NEED$SCHED, RUNNING$VP, EVCTBL$INDEX) BYTF;

    $IF NOT MCCRTFX

    /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****
    /*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****

    CALL OUTLINE(#MSG18);

SENDIF

    /* LOCK GLOBAL LOCK */
    DC WHILE LOCK$SET(@GLOBAL$LOCK, 119); END;
    NEED$SCHED = TRUF;

    /* DETERMINE THE RUNNING VIRTUAL PROCESSOR */
    RUNNING$VP = NET$VP;

    /* GET FVC INDEX */
    EVCTBL$INDEX = LOCATE$FVC(EVC$ID);

    /* DETERMINE IF CURRENT VALUE IS LESS THAN THE */
    /* AWAITED VALUE */
    IF EVCTBL(EVCTBL$INDEX).VALUE < AWAITED$VALUE THEN DO:
        /* LOCK PROCESS */
        VPM(RUNNING$VP).FVC$THRD$AD = EVCTBL(EVCTBL$INDEX).THRD;
        VPM(RUNNING$VP).FVC$VALUE = AWAITED$VALUE;
        VPM(EVCTBL$INDEX).THRD = RUNNING$VP;
        DISABLE;
        PRDS.LAST$RUN = RUNNING$VP;
        VPM(RUNNING$VP).STATE = WAITING;
    END; /* LOCK PROCESS */

    ELSE /* DO NOT LOCK PROCESS */
    NEED$SCHED = FALSE;

    /* SCHEDULE THE VIRTUAL PROCESSOR */
    IF NEED$SCHED = TRUE THEN
        CALL VPSCHEDULER; /* NO RETURN */

    /* UNLOCK GLOBAL LOCK */
    GLOBAL$LOCK = 0;

133
ADVANCE: PROCEDURE (FVC$ID) REENTRANT PUBLIC;

DECLARE
   (FVC$ID, FVCTRL$INDEX)            BYTE,
   (SAVE, RUNNING$VP, DUMMY$VAR, I)   BYTE,
   CLUSTER$ADD
      @WORD;

$IF NOT MCORTEX
   /*********************MXTRACE********MXTRACE********MXTRACE********MXTRACE********/
   /********MXTRACE********MXTRACE********MXTRACE********MXTRACE********/

   CALL OUT$LINE(@MSG19);

$ENDIF

/* LOCK THE GLOBAL LOCK */
TO WHILE LOCKSET @GLOBAL$LOCK, 119); END;

RUNNING$VP = KET$VP;
FVCTRL$INDEX = LOCATE$FVC(FVC$ID);
FVC$STBL(FVCTRL$INDEX).VALUE = FVC$STBL(FVCTRL$INDEX).VALUE + 1;
IF FVCTRL$FVC(FVCTRL$INDEX).REMOTE$ADD < > LOCAL$CLUSTER$ADD.
   THEN DO;

   /* REMOTE COPY IS NEEDED - THE CONVENTION IS:
      AN EVENTCOUNT THAT HAS A REMOTE COPY WILL
      NOT HAVE ITS "REMOTE$ADD" FIELD EQUAL TO THE
      LOCAL$CLUSTER$ADD. */

134
CLUSRS AND EVCSTBL(EVCSTBLINDEX), REMOTESADD?
XOR LOCALS CLUSTERSADD;
GLOBAL$LOCK = #;
CALL SYSTEMSIO(FMT,EVCSTYPE,EVC$ID,
EVCSTBL(EVCSTBLINDEX),VALUE,CLUSTERSADDER);
DO WHILE LOCKSFT(@GLOBAL$LOCK,119); END;

END; /* ITD */
SAVE = 255;
I = EVCSTBL(EVCSTBLINDEX).THREA;
DO WHILE I <> 255;
IF VPM(I).EVC$AW$VALUE <= EVCSTBL(EVCSTBLINDEX).VALUE
THEN DO; /* AWAKEN THE PROCESS */
VPM(I).STATE = READY;
VPM(I).EVC$AW$VALUE = 0;
IF S$AVE = 255 THEN DO; /* THIS FIRST ONE IN LIST */
  DUMMY$VAR = VPM(I).EVC$THREAD;
  EVCSTBL(EVCSTBLINDEX).THREA = DUMMY$VAR;
  VPM(I).EVC$THREAD = 255;
  I = EVCSTBL(EVCSTBLINDEX).THREA;
END; /* IF FIRST */
ELSE DO; /* THEN THIS NOT FIRST IN LIST */
  VPM(S$AVE ).EVC$THREAD = VPM(I).EVC$THREAD;
  VPM(I).EVC$THREAD = 255;
  I = VPM(S$AVE ).EVC$THREAD;
END; /* IF NOT FIRST */
END; /* IF AWAKEN */
ELSE DO; /* DO NOT AWAKEN THIS PROCESS */
  S$AVE = I;
  I = VPM(I).EVC$THREAD;
END; /* IF NOT AWAKEN */
FND; /* DO WHILE */
PHDS.LASTRUN = RUNNINGSVP;
VPM(RUNNINGSVP).STATE = READY;
CALL VPSCHEDULE; /* NO RETURN */
/* UNLOCK THE GLOBAL LOCK */
GLOBAL$LOCK = 0;
RETURN;
FND; /* ADVANCE PROCEDURE */

/* THIS PROCEDURE AWAKENS A HI 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 VPSID IS FE OR THE MONITOR PROCESS, IT WILL */
/* MAKE IT READY WHERE-EVER IT IS IN THE VPM. THE FOLLOW-*/
DECLARE (CH1, INCH) BYTE;

/* CHECK PORT STATUS */
INCH = (INPUT(ODAH) AND 0F0H);
IF INCH = 13H THEN
  DO WHILE ('INCH' <> 11H);
    IF (((INPUT(ODAH) AND 12H) < 0) THEN
      INCH = (INPUT(ODAH) AND 0FFH);
    END;
  DO WHILE (INPUT(ODAH) AND 01H) = 2;
  END;
OUTPUT(OD9H) = CH1;
RETURN;
END;

/* INCHEX PROCEDURE */
BEGIN 6-1-1-4 */

/* GETS 2 HEX CHAR FROM THE SERIAL PORT AND IgNEORS 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: FFCV$CHAR */
/* INCHEX: PROCEDURE BYTE RECENTIANT PUBLIC: */

DECLARE ASCII (*) BYTE DATA ('0123456789ABCDEF'),
ASCIIIL(*) BYTE DATA ('0123456789', 'A', 'B', 'C', 'D', 'E', 'F',
                      'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'),
INCHAR, HEXCHAR, 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 */
  INCHAR = FFCV$CHAR;
  H = 0;
  STOP = 0;
  /* COMPARE CHAR TO HEX CHAR SET */
  DO WHILE NOT STOP;
    IF (INCHAR = ASCII (H)) OR (INCHAR = ASCIIIL (H)) THEN TO;
    STOP = 0;
    FOUND = OFFH;
    CALL SEND$CHAR; INCHAR; /* TO ECHO IT */
  END;
END;

/* END */
DISABLE:
SEND = HIGH ( DNUM );
CALL OUTSHEX ( SEND );
SEND = LOW ( DNUM );
CALL OUTSHEX ( SEND );
END;
RETURN;

收回字符: 程序 RYTF 重入公开:

DECLARE
CH  RYTF;

/* 检查 PORT 状态位 2 为接收-准备信号 */
DO WHILE ( INPUT(OD1H) AND OD2H ) = 0;
    CH = ( INPUT(OD3H) AND OD7H );
END;
RETURN;

发送字符: 程序 (CHAR) 重入公开:
OUTSNUM: PROCEDURE ( NUM ) REENTRANT PUBLIC;

DECLARE NUM BYTE;
DISABLE;
CALL OUT$HEX( NUM );
ENABLE;
RETURN;
END;

/*---------------------------------------------*/
/* INSNUM PROCEDURE BREWER 8-1P-84 */
/* GETS FOUR ASCII FROM SERIAL PORT TO FORM WORD VALUE. */
/* CALLS MADE TO: IN$HEX */
/*---------------------------------------------*/

INSNUM: PROCEDURE ( RETSPTR ) REENTRANT PUBLIC;

DECLARE
RETSPTR POINTER,
NUM BASED RETSPTR WORD,
(H, L) WORD;
DISABLE;
H = INS$HEX;
H = SHL( H, 8 );
L = INS$HEX;
NUM = (H OR L);
ENABLE;
RETURN;
END;

/*---------------------------------------------*/
/* OUTFNUM PROCEDURE TREF$FX 8-1A-84 */
/* OUTPUTS A WORD VALUE NUMBERED VIA THE SERIAL PORT */
/* CALLS MADE TO: OUT$HEX */
/*---------------------------------------------*/

OUTSNUM: PROCEDURE ( NUM ) REENTRANT PUBLIC;

DECLARE
ENUM WORD,
SN$ND BYTE;

117
OUT$CHAR: PROCEDURE (CHAR) REENTRANT PUBLIC:

DECLARE CHAR BYTE;
DISABLE;
CALL SEND$CHAR (CHAR);
FNAPIF;
RETURN;
END;

OUT$LINE: PROCEDURE (LINE$PTR) REENTRANT PUBLIC:

DECLARE
LINE$PTR PINTER,
LINE BASED LINE$PTR (90) BYTE,
II BYTE;
DISABLE;
DO II = 0 TO 99;
IF LINEF (II) = 'K' THEN GO TO DONE;
CALL SEND$CHAR (LINE (II));
FND;
DONE: FNABLE;
RETURN;
END;

OUT$NUM: PROCEDURE BREWER 8-18-84 */
*/
/* CALLS MADE TO: OUT$FEX */
*******************************************************************************/
/* INPUT TO SERIAL PORT VIA SER861 DOWN LOAD PROGRAM MAY */
/* NOT BE ACCEPTED. */
/* POINTER IS PROVIDED BY USER SO HE CAN BE RETURNED THE */
/* CHARACTER. */
/* CALLS MADE TO: RECVSCCHAR */
/* ***********************************************************************************/

INSCHAR: PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;
DECLARE
    RET$PTR POINTER,
    INCHR BASED RET$PTR BYTE;
DISABLE;
    INCHR = RECVSCCHAR;
    ENABLE;
    RETUPN;
END; /* INSCHAR */

/*1317***********************************************************************************/
/* INSNUM PROCEDURE    BREWER 8-18-94 */
/* ***********************************************************************************/
/* GETS TWO ASCII CHAR FROM THE SERIAL PORT, EXAMINES */
/* THEM TO SEE IF THEY ARE IN THE SET A..F HEX AND FORMS */
/* A BYTF VALUE. EACH VALID HEX DIGIT IS ECHOED TO THE */
/* CRT. IMPROPER CHAR ARE IGNORED. NO ALLOWANCES ARE */
/* MADE FOR WRONG DIGITS. GET IT RIGHT THE FIRST TIME. */
/* IF YOU ARE INDIRECTLY ACCESSING THE SERIAL PORT VIA */
/* THE SER861 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. */
/* ***********************************************************************************/
/* CALLS MADE TO: IN$HEX */
/* ***********************************************************************************/

INSNUM: PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;
DECLARE
    RET$PTR PTRFP,
    NUM BASED RET$PTR BYTE;
DISABLE;
    NUM = IN$HEX;
    ENABLE;
    RETURN;
END; /* INSNUM */

/*1245***********************************************************************************/
/* *************************************************************************************

145
END: /* DEFINE CLUSTER */

/* DISTRIBUTIONSMAP PROCEDURE BREWER 8-18-84 */

/* THIS PROCEDURE ASSIGNS GROUP ADDRESSES TO THE */
/* REMOTEADDR FIELD OF THE DISTRIBUTED ENTITY. THIS IS */
/* SYSTEM MANAGEMENT DECISION - THE USER (ALTHOUGH SYSTEM)*/
/* PROCESSES DO NOT MAKE CALLS TO THIS PROCEDURE. */

/* DISTRIBUTIONSMAP: PROCEDURE (MAP$PTR) REENTRANT PUBLIC; */

DECLARE
MAP$PTR POINTER,
TBL$INDEX BYTE,
MAP$TABLE BASED MAP$PTR STRUCTURE
(MAP$TYPE BYTE,
ID BYTE,
CLUSTPR$ADDR WORD);

DO CASE MAP$TABLE.MAP$TYPE;

DO; /* FVEXTCOUNT TYPE */
TBL$INDEX = LOCATESVC (MAP$TABLE.ID);
EVCS$TBL.TPL$INDEX).RFMT$ADDR=MAP$TABLE.CLUSTPR$ADDR;
END;

DO;
/* STUB */
END;

FND: /* DO CASE */

FND; /* DISTRIBUTIONSMAP */

IF NOT MXTRACE

/* CONDITIONAL COMPI LATION OF PROCEDURES */
ASSOCIATED WITH *** MXTRACE *** */

/ *** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/

/ /* INSCHAR PROCEDURE BREWER 8-18-84 */
/ /* GETS A CHAR FROM THE SERIAL PORT. CHAR IS !!!NOT!!! */
/ /* ECHOED. THAT IS RESPONSIBILITY OF USER IN THIS CASE. */

144
FNZ:

/* SLOT OPEN SO WRITE TO FRB */
FRSINDEX = I "MOD FRB$BLOCK$LENGTH;
FRB (FRSINDEX).COMMAND = REQUEST$TYPE;
FRB (FRSINDEX).TYPESNAME = NAME;
FRB (FRSINDEX).NAME$VALUE = VALUE;
FRB (FRSINDEX).REMOTE$ADDR = ADDR;
/* NEED TO ADVANCE THE VALUE OF FRB$WRITE */

DO WHILE LOCK$SFT(GLOBAL$LOCK, 119);
/* ASSERT LOCK */
END;
INDEX = LOCATE$FVC(FRB$WRITE);
FVC$TRL(INDEX).VALUE = FVC$TRL(INDEX).VALUE + 1;
GLOBAL$LOCK = 0; /* RELEASE */

/* NOTE THAT THIS AVOIDS THE UNNECESSARY OVER- */
/* HEAD OF THE ADVANCE PROCEDURE */
END: /* DO BLOCK */

END; /* REQUEST$TYPE */

END: /* PATH */

END; /* SYSTEM$IO */

END; 0923

DEFINES CLUSTER PROCEDURE BREWEn 9-10-64 */
/*-----------------------------------------------*/
/* THIS PROCEDURE IS CALLED ONLY ONE TIME AT EACH CLUSTER. */
/* ITS SOLE PURPOSE IS TO DEFINE THE LOCAL$CLUSTER ADDRESS. */
/* THIS PROCEDURE CALL MUST BE THE FIRST CALL IN THE INIT */
/* PROCESS BROUGHT UP IN EACH CLUSTER. */
/* */
/* */
/********************************************************************/

DEFINES CLUSTER: PROCEDURE (CLUSTER$ID) REENTRANT PUBLIC;

DECLARE CLUSTER$ID WORD,
I BYTE;

LOCAL$CLUSTER$ADDR = CLUSTER$ID;
/* FOR NOW OTHER ENTRIES FIELDS ARE UNINITIALIZED */
FVC$TRL(I).REMOTE$ADDR = CLUSTER$ID;
/* FIRST ENTRY IN TABLE IS A RESERVED SYSTEM EVENTCOUNT */

143
/* IS WRITTEN TO ALLOW FOR THE EXTENSION TO OTHER DATA */
/* COMMUNICATION MEDIA. */
/* */
/* FUNCTIONALITY: */
/* QUEUES UP REQUESTS IN AN ETHERNET REQUEST BLOCK */
/* (ERB) FOR CONSUMPTION BY THE ETHERNET COMMUNICATION */
/* CONTROLLER RCAPD (FCCP) DEVICE HANDLER. */
/* */
/* CALLS MADE TO: */
/* READ */
/* ADVANCE */
/* TICKET */
/* */

SYSTEMSIC: PROCEDURE (PATH, REQUEST$TYPE, NAME, VALUE, ADDR)
PUBLIC REENTRANT;

DECLARE (
    (PATH, REQUEST$TYPE, NAME, ERB$INDEX, IND$F) BYTE,
    VALUE, ADDR, I, J ) BYTE;

DECLARE ERB(ERB$BLOCK$LENGTH) STRUCTURE
    (COMMAND BYTE,
     TYPES$NAME BYTE,
     NAME$VALUE WORD,
     REMOTFSADDR WORD) AT (1000H);

IF PATH = ENET THEN
DO;

DO CASE REQUEST$TYPE:

DO; /* IT'S ETHERNET AND EVENTCOUNT */
    CALL TICKET(ERB$WHITE$REQUEST, (I));
    /* I NOW HAS THE VALUE OF TICKET RETURNED */
    CALL READ(ERB$WHITE, (OJ));
    /* J NOW HAS THE VALUE OF ERB$WRITE */
    DO WHILE (J < I);
        CALL TIME (10);
        /* 1 MS DELAY => REDUCE BUS CONTENTION */
        CALL READ(ERB$WRITE, (OJ));
    END;
    /* DO WHILE */
    /* WRITE TO ERB, IF IT'S NOT FULL */
    CALL READ(ERB$READ, (OJ));
    DO WHILE ((I-J) >= ERB$BLOCK$LENGTH);
    /* IT'S FULL SO DO A "BUSY WAIT" */
    CALL TIME (60);
    /* DELAY ON PACKET TRANSMISSION TIME */
    CALL READ(ERB$READ, (OJ));

142
/* TO SET UP PROC$STACK$PTR */
PS1 = PROC$TABLE.PROC$SP - 11H;
PS2 = PROC$TABLE.PROC$SS;

PROC$STACK$RT$TYPE = INT$RETURN;
PROC$STACK$RP = 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$STACK$EX = 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 */
DO WHILE LOCKSET(@GLOBAL$LOCK,119): END;

IF PRDS.VPS$PR$CPU < MAX$VPS$CPU THEN DO;
TEMP = PRDS.VPS$PER$CPU + PRDS.VPS$START;
VPM( TEMP ).VPSID = PROC$TABLE.PROC$ID;
VPM( TEMP ).STATE = 01; /* READY */
VPM( TEMP ).VPS$PRIORITY = PROC$TABLE.PROC$PRI;
VPM( TEMP ).VPS$THREAD = 255;
VPM( TEMP ).VPS$VAL$VALUE = 0;
VPM( TEMP ).VPS$SP$REG = PROC$TABLE.PROC$SP - 1AH;
VPM( TEMP ).VPS$SS$REG = PROC$TABLE.PROC$SS;

PRDS.VPS$PR$CPU = PRDS.VPS$PER$CPU + 1;
PRDS.VPS$END = PRDS.VPS$END + 1;
NR$VPS( PRDS.CPU$NUMBER ) =
NR$VPS( PRDS.CPU$NUMBER ) + 1;
END: /* DO */

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

/*0832********************************************************************
 /* SYSTEM$PROCEDURE BREWER 8-12-84 */
/*---------------------------------------------------------------*/
/* PROCESSES A REQUEST FROM THE ADVANCE PROCEDURE (AND */
/* OTHERS TO BE DEVELOPED) TO ADVANCE THE VALUE OF AN */
/* EVENTCOUNT THAT HAS A REMOTE COPY. ALTHOUGH THE */
/* CURRENT IMPLEMENTATION IS LIMITED TO THE ETHERNET AS */
/* THE MEDIUM FOR DISTRIBUTED EVENTCOUNTS, THE PROCEDURE */

141
/* PROCESS PRIORITY, THE DESIRED PROC STACK LOCATION. */
/* AND THE PROCESS CODE STARTING LOCATION WHICH IS */
/* IS THE ELEMENTS: THE IP REGISTER (OFFSET) AND THE */
/* CS REGISTER (CODE SEGMENT). */
/* CALLS MAIL TO: OUTFIT */
/* *********************************************** */

CREATE PROC: PROCOUTRF( PROCSPTR ) REENTRANT PUBLIC;

DECLARE
PROCSPTR POINTER,
PROCSTABLE BASED PROCSPTR STRUCTURE
(PROCSID BYTE,
PROCSPRI BYTE,
PROCSPR WORD,
PROCSSS WORD,
PROCSIP WORD,
PROCSSS WORD,
PROCSSD WORD,
PROCSPES WORD);

DECLARE (PS1, PS2) WORD,
TEMP BYTE;

DECLARE PROCSTACKSPTR POINTER AT(@PS1),
PROCSTACK BASED PROCSTACKSPTR STRUCTURE
.LENGTH(0FEEH) BYTE,
.MSTYP TYPE WORD,
BP WORD,
DI WORD,
SI WORD,
DS WORD,
DX WORD,
CX WORD,
AX WORD,
BX WORD,
FS WORD,
IP WORD,
CS WORD,
FL WORD);

$IFDEF NOT McORTEX

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  

CALL OUTSLINF(@MSG26);

$ENDIF

143
/* PRESENT VALUE OF SPECIFIED SEQUENCE AND INCREMENTS THE */
/* SEQUENCE. 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: OUTSLINE */

_TICKET: PROCEDURE(SEQSNAMF, RETST$TPW) REENTRANT PUBLIC:

DECLARE
SEQSNAMF BYTEF,
SEQTLSINDEX BYTEF,
RETS$TPW POINTERP,
SEQ$VALUE$RET POINTER RETS$TPW WORD:

/* ASSERT GLOBAL LOCK */
DO WHILE LOCKSET(@GLOBALSLOCK,119): END:

$IF NOT MCOPTEX

/* **** MTRACE *** MTRACE *** MTRACE *** MTRACE *** */
/* **** MTRACE *** MTRACE *** MTRACE *** MTRACE *** */
CALL OUTSLINE(#MSG24):

ENDIF

/* OBTAIN SEQSNAM INDEX */
SEQTLSINDEX = LOCATESEQ(SEQSNAMF);
/* OBTAIN SEQUENCER VALUE */
SEQ$VALUE$RET = SEQ$TABLE(SEQTLSINDEX).SEQ$VALUE;
/* INCREMENT SEQUENCER */
SEQ$TABLE(SEQTLSINDEX).SEQ$VALUE =
SEQ$TABLE(SEQTLSINDEX).SEQ$VALUE + 1 ;

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

/*0727*********************************************************************/
/* CREATE$PROC PROCEDURE BREWER 8-15-84 */
/*----------------------------------------------------------------------*/
/* THIS PROCEDURE CREATES A PROCESS FOR THE USER 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, */

139
CREATESSEQ:  PROCEDURE (NAME)  REENTRY PUBLIC;

DECLARE NAME BYTE;

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

IF NOT MCOTTEX

/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******

**MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******

CALL OUTSLINE (UMSG25);
CALL OUT$HFX (NAME);
CALL OUT$CPAR (CR);
CALL OUT$CHAR (LF);

SENDIF /* THE SEQUNCER DOES NOT ALREADY EXIST, IF */
LOCATFFSEQ (NAME) = NOT$FOUND THEN DO:
/* CREATE THE SEQUNCER ENTRY BY ADDING THE */
/* NEW SEQUNCER TO THE END OF THE SEQ$TABLE */
SEQ$TABLE (SEQUNCERS).SEQ$NAME = NAME;
SEQ$TABLE (SEQUNCERS).SEQ$VALUE = 0;
/* INCREMENT NUMBER OF SEQUNCERS */
SEQUNCERS = SEQUNCERS + 1;
FND; /* CREATE THE SEQUNCER */
/* RELASE THE GLOBAL LOCK */
GLOBAL$LOCK = 0;
RETURN;
FND; /* CREAT$SEQ PROCEDURE */

/*0672***************************************************************************************************************************/
/*  TICKET  PROCEDURE                         BEEWER  9-10-84 */
/****************************************************************************************************************************/
/* INTER-VIRTUAL PROCESSOR SEQUNCER PRIMITIVE FOR USER */
/* PROGRAM. SIMILAR TO 'TAKE A NUMBER AND WAIT.'  RETURNS*/

132
SNDIF

HW$INT$FLAG(CPU) = TRUE;
DISABLE;
OUTPUT(PORT$CA) = HCH;
CALL TIME(1);
OUTPUT(PORT$CA) = RESET;
ENABLE;
END;

END; /* NORMAL PRESENT */
ELSE DO; /* PRESENT THE MONITOR */
/* SEARCH VPM POP ALL ID´S OF OFFH */
SEARCH$ST = 0;
DO WHILE IOK$SET(GLOBAL$LOCK,119); END;
DO CPU = 0 TO (NP$PPS - 1):
SEARCH$END = SEARCH$ST + NP$PPS(CPU) - 1;
/* SET ALL INT$FLAGS EXCEPT THIS CPU´S */
IF PRDS.CPU$NUMBER <> CPU THEN
HW$INT$FLAG(CPU) = TJE;
DO INDEX = SEARCH$ST TO SEARCH$END:
IF VPM(INDEX).VP$ID = VP$ID THEN
VPM(INDEX).STAT$ = READY;
END; /* DO */
SEARCH$ST = SEARCH$ST + MAX$VP$PPS(CPU);
END; /* ALL MONITOR PROCESS SET TO READY */
/* INTERRUPT THE OTHER CPU´S AND RESCHEDULE THIS ONE */

SIF NOT MCOPTRX

/* MXTRACE *** MXTRACE *** MXTRACE *** MXTRACE *** */
/* MXTRACE *** MXTRACE *** MXTRACE *** MXTRACE *** */

CALL OUT$LINE(GMSG17);

SNDIF

DISABLE;
OUTPUT(PORT$CA) = HCH;
CALL TIME(1);
OUTPUT(PORT$CA) = RESET;
ENABLE;
INDEX = PPS$VP;
DISABLE;
PRDS.CPU$ST$UN = INDEX;
VPM(INDEX).STAT$ = READY;
CALL VPSC$FUL; /* NO RETURN */
END; /* ELSE */
/* UNLOCK GLOBAL MEMORY */
GLOBAL$LOCK = 0;
RETURN;
END; /* PREEMPT PROCEDURE */
/* ING CORE does not take advantage of the fact that */
/* currently it is the third entry in the VPM for each */
/* final process. */
/* calls made to: cutline, vpschedule */

Pragma: PROCEDURE VPSID ) REENTRANT PUBLIC;

Declare (VPSID, SEARCH$ST, SEARCH$END, CPU, INDEX) BYTE;

$IF NOT MCOREX

/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******
/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******

CALL OUT$LINE( @MSG16 );

ENDIF

IF VPSID <> 0 THEN DO; /* NORMAL PREEMT */
/* SEARCH VPM FOR INDEX FOR ID */

SEARCH$ST = ??;

DO CPU = 0 TO (NR$VPS - 1);
SEARCH$END = SEARCH$ST + NR$VPS( CPU ) - 1;

DO INDEX = SEARCH$ST TO SEARCH$END;

IF VPM( INDEX ).VPSID = VPSID 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 VP TO READY */

VPM( INDEX ).STATE = READY;
/* NEED HARDWARE INTR OR RE-SCHED */

IF ( CPU = PRDS.CPU$NUMBER ) THEN DO;
INDEX = HT$VP;
/* DETERMINE RUNNING PROCESS */
DISABLE;
PRDS.LAST$RUN = INDEX;
VPM( INDEX ).STATE = READY; /* SET TO READY */
CALL VPSCHEDULER; /* NO RETURN */
END;
ELSE DO; /* CAUSE HARDWARE INTERRUPT */

ENDIF

$IF NOT MCOREX

/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******
/*** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ****** MXTRACE ******

CALL OUT$LINE( @MSG17 );

136
FLSF DO:
  \n  I = H + 1;
  IF I = 10H THEN STOP = 0FFH;
END; /* FLSF */
END; /* DO WHILE */
H = SEL(H, 4);
END; /* DO WHILE */
FOUND = 0;
DO WHILE NOT FOUND;
  INCHR = RECVSCHAR;
  L = 0H;
  STOP = 0;
  DO WHILE NOT STOP;
    IF (INCHR=ASCII \"L\") OR (INCHR=ASCII \"L\") THEN LC;
    STOP = 0FFH;
    FOUND = 0FFH;
    CALL SEND$CHAR(INCHR);
  END;
  FLSF DO;
    L = L + 1;
    IF L = 10H THEN STOP = 0FFH;
  END; /* FLSF */
END; /* DO WHILE */
END; /* DO WHILE */
RETURN (\"L\" OR L);
END; /* IN$HEX */

/*1230*******************************************************************/
/* OUT$HEX  PROCEDURE Brewer :=-16-(44 */
/*------------------------------------------------------------------------*/
/* TRANSLATES BYTE VALUES TO ASCII CHARACTERS AND OUTPUTS */
/* THEM THROUGH THE SERIAL PORT */
/*----------------------------------------------- */
/* CALLS MADE TO: SEND$CHAR */
/************************************************************************************/

OUT$HEX: PROCEDURE(B) REENTRANT PUBLIC:

DECLARE R BYTE;
DECLARE ASCII(*) BYTE DATA ('212346789ABCDEF');

CALL SEND$CHAR(ASCII(SH:(3,4) AND 0FH));
CALL SEND$CHAR(ASCII(B AND 0FH));
RETURN;
END;

/* END CONDITIONAL COMPIILATION OF PROCEDURES NEEDED FOR */

MXTRACE

157
$ENDIF

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

END: /* L2$MODULE */

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

/*****************************/
APPENDIX H

LEVEL I MCORTEX SOURCE CODE

The LEVEL I source code, written in PL/M-86, is contained in file LFVFL1.SRC. The SUBMIT utility [Ref. 19] is used to compile either MCORTEX or MXTRACE versions of KOPF. The MCORTEX version of LEVEL I is compiled by using the SUBMIT file LICMPM.CSD. LEVEL I is one of the relocatable code modules shown in the SUBMIT file LNKKM.CSD in Appendix G. The SUBMIT file LOCKM.CSD is used to locate the various modules to file KORF. After transfer to the multi-user CP/M-86 system, the code is saved as KORE.OPS (as described in Appendix A). Analogous files are provided to generate KORF.TRC. The memory maps created by the linker and locator are included at the end of this appendix.
LICMPM.CSD SUBMIT file

: F1:PLM6 : F1:LEVEL1.SRC SET(MCOSER) NOCOND LEFTMARG1N(?)
LARGE

LICMPM.CSD SUBMIT file

: F1:PLM6 : F1:LEVEL1.SRC SET(MCOSER) NOCOND LEFTMARG1N(?)
LARGE

LNKKT.CSD SUBMIT file

: F1:LINKA6 : F1:LEVEL1.OBJ,: F1:LEVEL2.OBJ,: F1:SCHED.OBJ,:),
: F1:INTK.OBJ,: F1:GLOBAL.OBJ TO : F1:KORE.LNK

LNKKT.CSD SUBMIT file

: F1:LINKA6 : F1:LEVEL1.OBJ,: F1:LEVEL2.OBJ,: F1:SCHED.OBJ,),
: F1:INTK.OBJ,: F1:GLOBAL.OBJ TO : F1:KORE.LNK

LOCKM.CSD SUBMIT file

: F1:LOCM6 : F1:KORE.LNK ADDRESSFS SEGMENTS(
STACK(0C550H),
INITMOD_CODE(04590H),
GLOBALMOD. DATA(0F539FFH))
SGSIZER(STACK.75FF)
FFFSRR3F(0H TO 06FFH)

---

STACK(0C550H),
INITMOD_CODE(04590H),
GLOBALMOD. DATA(0F539FFH))
SGSIZER(STACK.75FF)
FFFSRR3F(0H TO 06FFH)
LOCKT.CSD  SUBMIT file

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

:F1:LOCAT6 :F1:KORE.LNK  ADDRESSSES(SEGMENTS(&
STACK(025H),&
INITMOD_CODE(04350H),&
GLOBALMODULE_DATA(02530H)))&
SEG:SIZEF(STACK(75H))&
RESERVE(0H TC 0AFFF)H)

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

LEVEL1.SRC  file

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

FILE: LEVEL1.SRC
VERSION: BPFWER B-I 4-84
PROCEDURES
DEFINED: RET$VP       RDYTHISVP
GETWORK    LOCATESVRC
LOCATFSSFO IDLESVRC
SAVESCONTEXT GET$SP
MONITOR$PROC

REMARKS:
(1) WARNING: SEVERAL OF THE LITERAL DECLARATIONS BELOW
HAVE A SIMILAR MEANING IN OTHER MODULES. THAT MEAN-
ing IS COMMUNICATED ACROSS MODULES BOUNDARIES. BE
CAREFUL WHEN CHANGING THEM.

(2) CONDITIONAL COMPILATION FACILITIES ARE USED TO
PRODUCE TWO OS VERSIONS. "MCORTFX" PROVIDES NO
DIAGNOSTIC ASSISTANCE, WHEREAS "MYTRACE" PROVIDES
DISPLAY MESSAGES ANNOUNCING THE ENTRY INTO VARIOUS
OS PRIMITIVES.

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

L1$MODULO: DO;

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

:'LEVEL1.SRC'

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

LOCAL DECLARATIONS

DECLARE
MAX$CPU         LITFRALLY '10'

154
MAX$VPS$CPU
MAX$CPU$SS
MAX$VPS$CPU
FALSE
FAIL
RUNNING
WAITING
TRUE
NOTFOUND
PORT5C0
PORT5C2
PORT5CF
PORT5CA
RESET
INTS
RETURN

*IF MCORTEX

/***** MCORTEX ***** MCORTEX ***** MCORTEX ***** MCORTEX *****
/***** MCORTEX ***** MCORTEX ***** MCORTEX ***** MCORTEX *****

*/FLSF

/***** MXTACF ***** MXTACF ***** MXTACF ***** MXTACF *****
/***** MXTACF ***** MXTACF ***** MXTACF ***** MXTACF *****

*/ENDIF

155
/* PROCESS DATA SEGMENT TABLE */
/* INFORMATION RELEVANT TO THE PARTICULAR PHYSICAL */
/* PROCESS OR WHICH IT IS RESIDENT */
/* */
/* CPU$NUMBER: UNIQUER SEQUENTIAL NUMBER ASSIGNED TO */
/* THIS REAL PROCESS */
/* */
/* VP$START: VPN INDEX OF THE FIRST VIRTUAL */
/* PROCESS ASSIGNED TO THIS REAL CPU */
/* */
/* VP$END: INDEX IN VPN OF LAST VIRTUAL... */
/* */
/* VP$CPU: THE NUMBER OF VP ASSIGNED TO THIS */
/* REAL CPU. MAX IS 10 */
/* */
/* LAST$RUN: VPN INDEX OF THE PROCESS MOST */
/* RECENTLY SWITCHED FROM RUNNING TO */
/* EITHER READY OR WAITING */
/* */
/* COUNTER: AN ARBITRARY MEASURE OF PERFORMANCE */
/* count made while in idles stat. */
/* */

DFCLARE PRDS STRUCTURE
(CPU$NUMBER BYTEF,
 VP$START BYTEF,
 VP$END BYTEF,
 VP$CPU BYTEF,
 LAST$RUN BYTEF,
 COUNTER WORD) PUBLIC INITIAL(0,0,0,0,0,0);

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

DFCLARE VPM( MAX$CPU$CPU$CPU$CPU ) STRUCTURE
(VPSID BYTEF,
 STATE BYTEF,
 VP$PRIORITY BYTEF,
 VPS$PRIORITY BYTEF,
 VPS$SVALUE WORD,
 SPECF WORD,
 SS$SVALUE WORD) EXTERNAL;

DFCLARE
CPU$INIT BYTEF EXTERNAL,
HDWSINT$FLAG( MAX$CPU ) BYTEF EXTERNAL,
NR$VPS( MAX$CPU ) BYTEF EXTERNAL,
M$VPS BYTEF EXTERNAL,
GLOBAL$LOCK BYTEF EXTERNAL;
DECLARE EVENTS BYTE EXTERNAL,
    FVC$TLP(10?) STRUCTURE
        (FVC$NAME BYTE,
         VALUE WORD,
         FVC$TLP Id WORD,
         THREAD BYTE) EXTERNAL;

DECLARE SEQUENCERS BYTE EXTERNAL,
    SEOST$TLP(10?) STRUCTURE
        (SEO$NAME BYTE,
         SEO$VALUE WORD) EXTERNAL;

/**148*******************************************************************/
/** DECLARATION OF EXTERNAL PROCEDURE REFERENCES */
/** THE FILE AND MODULE WHERE THEY ARE DEFINED ARE */
/** LISTED. */

INITIAL$PROC: PROCEDURE EXTERNAL; END;
    /* IN FILE:   INIT$S.C */
    /* IN MODULE: INIT$MOD */

AWAIT: PROCEDURE (FVC$ID, AWAITED$VALUE) EXTERNAL;
    DECLARE FVC$ID BYTE, AWAITED$VALUE WORD;
    END;

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

DECLARE INTO$SEQC LA$T EXTERNAL;
    /* IN FILE: SCHED.ASM */

DECLARE INTO$VECTOR POINTER AT(0110H) INITIAL(0INTVAC);
    /* IN FILE: SCHED.ASM */

/**171*******************************************************************/
/** THESE DIAGNOSTIC MESSAGES MAY EVENTUALLY BE REMOVED. */
/** THE UTILITY PROCEDURES, HOWEVER, ARE ALSO USED BY THE */
/** MONITOR PROCESS. THEY SHOULD NOT BE REMOVED. */

SIF NOT MCORTEX

/**** MXT$FACE **** MXT$ACEF **** MXT$FACE **** MXT$FACE **** MXT$FACE *****/
/**** MXT$ACEF **** MXT$ACEF **** MXT$FACE **** MXT$FACE **** MXT$FACE *****/

157
DECLARE

MSG1(*) BYTE INITIAL ('ENTERING PETSVP', '13,10','\x'),
MSG1A(*) BYTE INITIAL ('ENTERING RUNNINGSVP INDEX', '10','\x'),
MSG4(*) BYTE INITIAL ('ENTERING DYTHISVP', '13,10','\x'),
MSG4A(*) BYTE INITIAL ('SET VP TO READY: VP = \x'),
MSG7(*) BYTE INITIAL ('ENTERING GTFWORK', '13,10','\x'),
MSG7A(*) BYTE INITIAL ('SET VP TO RUNNING: VP = \x'),
MSG7B(*) BYTE INITIAL ('SELECTED = \x'),
MSG10(*) BYTE INITIAL ('ENTERING IDLFSTVP', '13,10','\x'),
MSG11(*) BYTE INITIAL ('UPDATE IDLE COUNT', '13,10','\x'),
MSG12(*) BYTE INITIAL ('ENTERING KERNELINIT', '10,13','\x'),
MSG20(*) BYTE INITIAL ('ENTERING LOCATESVC', '10,13','\x'),
MSG22(*) BYTE INITIAL ('ENTERING LOCATESVC', '10,13','\x'),
MSG23(*) BYTE INITIAL ('FOUND', '10,13','\x'),
MSG24(*) BYTE INITIAL ('NOT FOUND', '10,13','\x');

DECLARE CP LITERALLY '\x0d\x0a',
LF LITERALLY '\x0a';

OUTSCAR: PROCEDURE CHAR EXTERNAL;
  DECLARE CHAR PTR;
END;

OUTSLINE: PROCEDURE LINFSPTR EXTERNAL;
  DECLARE LINFSPTR POINTER;
END;

OUTSNUM: PROCEDURE NUM EXTERNAL;
  DECLARE NUM BYTE;
FND;

OUTSDNUM: PROCEDURE DNUM EXTERNAL;
  DECLARE DNUM WORD;
END;

OUTSHEX: PROCEDURE B EXTERNAL;
  DECLARE B BYTE;
END;

INSCAR: PROCEDURE RETSPTR EXTERNAL;
  DECLARE RETSPTR POINTER;
END;

INSDNUM: PROCEDURE RETSPTR EXTERNAL;
  DECLARE RETSPTR POINTER;
END;

IN$NUM: PROCEDURE RETSPTR EXTERNAL;
  DECLARE RETSPTR POINTER;
END;

158
DECLARE IDLE$STACK STRUCTURE
LENTH(?32H) WORD,
RET$TYPE WORD,
BP WORD,
DI WORD,
SI WORD,
DS WORD,
DX WORD,
CX WORD,
AX WORD,
FX WORD,
FS WORD,
START POINTER, /* IP,CS */
FL WORD) AT(IDLE$STACK$SAES)

DECLARE INIT$STACK STRUCTURE
LENTH(?32H) WORD,
RET$TYPE WORD,
BP WORD,
DI WORD,
SI WORD,
DS WORD,
DX WORD,
CX WORD,
AX WORD,
FX WORD,
FS WORD,
START POINTER, /* IP,CS */
FL WORD) AT(INIT$STACK$SAES)

IF NOT MCOTRXY
*/ 200% SETS THE IF FLAG */

159
DECLARE MONITOR$STACK STRUCTURE
(LIGHT(030H) WORD,
FIFTYPE WORD,
DP WORD,
DI WORD,
S WORD,
D WORD,
D WORD,
C WORD,
D WORD,
I WORD,
V WORD,
F WORD,
START POINTER,  
)AT(MONITOR$STACK$AES)

INITIAL(
O,O,O,O,O,O,O,O,O,O,O,O,O,O,
INT$RETURN,7AH,O,O,O,O,O,O,O,O,
)MONITOR$PROC,29PH)

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
$ENDIF

/******* MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
/******* MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
/* PET$VP PROCEDURE BREWER 8-12-84 */ 
/**---------------------------------------------------------------**/ 
/* USED BY THE SCHEDULER TO FIND OUT WHAT IS THE CURRENT */ 
/* RUNNING PROCESS. IT'S INDX IN VPM IS RETURNED. */ 
/**---------------------------------------------------------------**/ 
/* CALLS MADE TO: OUT$HEX OUT$CHAIN */ 
/******* MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
/******* MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
RPT$VP: PROCEDURE BYTE REENTRANT PUBLIC;
DECLARE RUNNING$VP$POINTEX BYTE;
SIF NOT MCO$TPX
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/ 
CALL OUT$LINE(@MSG1):
$/ENDIF

160
/* SEARCH THE VP MAP FOR RUNNING PROCESS INDEX */
DO RUNNINGSVPINDEX = PRDS.VPSTART TO PRDS.VP END:
IF VPM( RANNNINGSVPINDEX ).STATE = RUNNING THEN GO TO FOUND:
END: /* DO */
RUNNINGSVPINDEX = PRDS.LASTS_RUN:

FOUND:

$IF NOT MCCortex

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  
CALL OUTSLINE(*MSG1A):  
CALL OUTCHAR(RUNNINGSVPINDEX);  
CALL OUTCHAR(CR);  
CALL OUTCHAR(LF):
SENDIF

RETURN RUNNINGSVPINDEX;
END: /* RTSVP PROCEDURE */

/**366***************************************************************************/
/** HDYTHISVP PROCEDURE REENTRANT PUBLIC; BRIEFER 9-18-84 */
/** CHANGES A VIRTUAL PROCESSOR STATE TO READY */
/** CALLS MADE TO: OUTCHAR OUTSTFF */
*******************************************************************************/

HDYTHISVP: PROCEDURE REENTRANT PUBLIC;

$IF NOT MCCortex

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  
CALL OUTSLINE(*MSG4):
SENDIF

PRDS.LASTS_RUN = RTSVP; /* SAVE INDEX */

$IF NOT MCCortex

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE ****/  
CALL OUTSLINE(*MSG4,1):

161
CALL OUTSHPX( PRDS.LAST$RUN );
CALL OUTSCHAR( CR );
CALL OUTSCHAR( LF );

ENDIF

VPM( PRDS.LAST$RUN ).STATE = READY;
RETURN;
END; /* EDYTHISVP PROCEDURE */

/***** SAIECONTECT PROCEDURE *****************************/
/* SAVFCONTEXT PROCEDURE *****************************/
/* SAVFCONTEXT PROCEDURE (STACK$PTR, STACK$SEG) REENTRANT
PUBLIC;

DECLARF 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$RUN ).SS$SEG = STACK$SEG; /* STATE */
END;

END;

/***** GET$SP PROCEDURE *****************************/
/* GET$SP PROCEDURE *****************************/
/* GET$SP PROCEDURE WORD REENTRANT PUBLIC;

DECLARF N PYTE;

N = RETSVIP; /* GET CURRENT RUNNING VIRTUAL PROCESSOR */
RETURN VPM( N ).SP$REG; /* RETURN NEW VP STACK POINTER */
END:
NETWORK: PROCEDURE WORD REFERENT PUBLIC:

DECLARE (PHI,N,I) PFB;
DECLARE SELECTED DBL WORD;
DECLARE DISPLAY PFB;

$IFDEF NOT MCORTEX

/****** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****
/****** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****

CALL OUTSLINE(OMSG7):

ENDIF

PRI = 255;
DO /* SEARCH VPM FOR ELIGIBLE VIRTUAL PROCESSOR TO RUN */
   I = PREGS.VP$START TO PREGS.VP$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;
      FND: /* IF */
   FND: /* DO LOOP SEARCH OF VPM */

/* SET SELECTED VIRTUAL PROCESSOR */
VPM(N).STATE = RUNNING;
SELECTEDDBR = VPM(N).SS$FP7;

$IFDEF NOT MCORTEX

/****** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****
/****** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****

CALL OUTSLINE(OMSG7A);
CALL OUTSLINE(OMS7A);
CALL OUTSLINE(CP);
CALL OUTSLINE(NP);

*FNDIF
### Symbol Table of Module LIMODULE

<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>0C26H</td>
<td>0026H</td>
<td>PUB</td>
<td>PADS</td>
<td>CAC2H</td>
<td>0266H</td>
<td>PUB</td>
<td>LOCATEP</td>
</tr>
<tr>
<td>0A2CH</td>
<td>042CH</td>
<td>PUB</td>
<td>IDLPROC</td>
<td>CAC2H</td>
<td>03FDH</td>
<td>PUB</td>
<td>LIDTP</td>
</tr>
<tr>
<td>0A3CH</td>
<td>052CH</td>
<td>PUB</td>
<td>LOCATEDPC</td>
<td>CAC2H</td>
<td>0236H</td>
<td>PUB</td>
<td>SAVECOUNTYT</td>
</tr>
<tr>
<td>0A6CH</td>
<td>062CH</td>
<td>PUB</td>
<td>GETSP</td>
<td>CAC2H</td>
<td>0165H</td>
<td>PUB</td>
<td>GETUP</td>
</tr>
<tr>
<td>0A7CH</td>
<td>072CH</td>
<td>PUB</td>
<td>GFXTHISVP</td>
<td>CAC2H</td>
<td>0234H</td>
<td>PUB</td>
<td>GINGP</td>
</tr>
<tr>
<td>0A4CH</td>
<td>082CH</td>
<td>PUB</td>
<td>OUTHEX</td>
<td>CAC2H</td>
<td>0359H</td>
<td>PUB</td>
<td>IPYCHAR</td>
</tr>
<tr>
<td>0A5CH</td>
<td>092CH</td>
<td>PUB</td>
<td>SENDCHAR</td>
<td>CAC2H</td>
<td>044AH</td>
<td>PUB</td>
<td>INOUT</td>
</tr>
<tr>
<td>044CH</td>
<td>002CH</td>
<td>PUB</td>
<td>OUTNUM</td>
<td>CAC2H</td>
<td>048AH</td>
<td>PUB</td>
<td>OUTCHAR</td>
</tr>
<tr>
<td>045CH</td>
<td>012CH</td>
<td>PUB</td>
<td>INCHAP</td>
<td>CAC2H</td>
<td>049CH</td>
<td>PUB</td>
<td>DISTRIB</td>
</tr>
<tr>
<td>046CH</td>
<td>022CH</td>
<td>PUB</td>
<td>REDEFCLUSTER</td>
<td>048CH</td>
<td>068BH</td>
<td>PUB</td>
<td>SYSTEMIC</td>
</tr>
<tr>
<td>047CH</td>
<td>032CH</td>
<td>PUB</td>
<td>CREATEPROC</td>
<td>048CH</td>
<td>07AFH</td>
<td>PUB</td>
<td>TICKET</td>
</tr>
<tr>
<td>048CH</td>
<td>042CH</td>
<td>PUB</td>
<td>CREATEPC</td>
<td>048CH</td>
<td>0613F</td>
<td>PUB</td>
<td>PHREJECT</td>
</tr>
<tr>
<td>049CH</td>
<td>052CH</td>
<td>PUB</td>
<td>ADVANCE</td>
<td>048CH</td>
<td>0231H</td>
<td>PUB</td>
<td>WAIT</td>
</tr>
<tr>
<td>04ACH</td>
<td>062CH</td>
<td>PUB</td>
<td>HEAD</td>
<td>048CH</td>
<td>0145H</td>
<td>PUB</td>
<td>CREATPC</td>
</tr>
<tr>
<td>04BCH</td>
<td>072CH</td>
<td>PUB</td>
<td>CATKEEPER</td>
<td>048CH</td>
<td>0342H</td>
<td>PUB</td>
<td>VPSCHEDULE</td>
</tr>
<tr>
<td>04CH</td>
<td>082CH</td>
<td>PUB</td>
<td>PLOTS</td>
<td>048CH</td>
<td>0439H</td>
<td>PUB</td>
<td>INITIATECC</td>
</tr>
<tr>
<td>04DCH</td>
<td>092CH</td>
<td>PUB</td>
<td>TP</td>
<td>048CH</td>
<td>0878H</td>
<td>PUB</td>
<td>CLUSTERCC</td>
</tr>
<tr>
<td>04ECH</td>
<td>0A2CH</td>
<td>PUB</td>
<td>SEQUENCERS</td>
<td>048CH</td>
<td>0324H</td>
<td>PUB</td>
<td>DISTRIB</td>
</tr>
<tr>
<td>04FCH</td>
<td>0B2CH</td>
<td>PUB</td>
<td>SYSTEM</td>
<td>048CH</td>
<td>0244H</td>
<td>PUB</td>
<td>LOCALI-</td>
</tr>
<tr>
<td>050CH</td>
<td>0C2CH</td>
<td>PUB</td>
<td>EVENTS</td>
<td>048CH</td>
<td>0642H</td>
<td>PUB</td>
<td>CLUSTERCC</td>
</tr>
<tr>
<td>051CH</td>
<td>0D2CH</td>
<td>PUB</td>
<td>UPPS</td>
<td>048CH</td>
<td>0532H</td>
<td>PUB</td>
<td>LOCALI-</td>
</tr>
<tr>
<td>052CH</td>
<td>0E2CH</td>
<td>PUB</td>
<td>GLOBALLOCK</td>
<td>048CH</td>
<td>0643H</td>
<td>PUB</td>
<td>LOCALI-</td>
</tr>
</tbody>
</table>

177
ISIS-II MCS-66 LINKER, V1.1, INVOKE BY:
:FL:LINK66, F1:LEVEL1.OBJ, F1:LEVEL2.OBJ, F1:SCHED.OBJ, A
:FL:INITK.OBJ, F1:GLOBAL.OBJ TC: F1:KOPF.LNK
LINK MAP FOR: F1:KOPF.LNK(1,MODULE)

LOGICAL SEGMENTS INCLUDED:

LENGTH ADDRESS SEGMENT CLASS
0066H ------ 11MODULE_CODE CODE
0133H ------ 11MODULE_DATA DATA
00E2H ------ STACK STACK
028FH ------ MEMORY MEMORY
2DE8H ------ 22MODULE_CODE CODE
27FH ------ 22MODULE_DATA DATA
3027H ------ ??SEG
C52FH ------ SCHEDULED
061AH ------ INITMOD_CODE CODE
2721H ------ INITMOD_DATA DATA
2788H ------ GLOBALMODULE_C CODE
7087H ------ GLOBALMODULE_P DATA

INPUT MODULES INCLUDED:
F1:LEVEL1.OBJ(11MODULE)
F1:LEVEL2.OBJ(22MODULE)
F1:SCHED.OBJ(SCPFD)
F1:INITK.OBJ(INITMOD)
F1:GLOBAL.OBJ(GLOBALMODULE)
# Memory Map of Module L1module

**READ FROM FILE:** F1:KORE.LIB
**WRITTEN TO FILE:** F1:KORE

**MODULE START ADDRESS** : PARAGRAPHS = 2372H OFFSET = 0072H

<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>00112H</td>
<td>00113H</td>
<td>0014H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
</tr>
<tr>
<td>0435CH</td>
<td>0434BH</td>
<td>001AH</td>
<td>W</td>
<td>INITMODY CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>03928H</td>
<td>03927H</td>
<td>001AH</td>
<td>W</td>
<td>LIMODULE CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>02A24H</td>
<td>02A23H</td>
<td>0014H</td>
<td>W</td>
<td>L2MODULE CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>06498H</td>
<td>06498H</td>
<td>000AH</td>
<td>W</td>
<td>GLOBALMODULE_C CODE</td>
<td>CODE</td>
</tr>
<tr>
<td>06498H</td>
<td>0649CH</td>
<td>0009H</td>
<td>W</td>
<td>LIMODULE_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>06498H</td>
<td>0649AH</td>
<td>0009H</td>
<td>W</td>
<td>L2MODULE_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>06498H</td>
<td>06499H</td>
<td>0009H</td>
<td>W</td>
<td>INITMODY_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>06498H</td>
<td>06499H</td>
<td>0009H</td>
<td>G</td>
<td>??SGF</td>
<td></td>
</tr>
<tr>
<td>06498H</td>
<td>0649AH</td>
<td>0009H</td>
<td>G</td>
<td>SCHEDULER</td>
<td></td>
</tr>
<tr>
<td>06555H</td>
<td>06555H</td>
<td>0097H</td>
<td>W</td>
<td>STACK</td>
<td>STACK</td>
</tr>
<tr>
<td>06555H</td>
<td>06555H</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
</tr>
<tr>
<td>06555H</td>
<td>06555H</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
</tr>
<tr>
<td>03F27H</td>
<td>03F27H</td>
<td>000AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td></td>
</tr>
<tr>
<td>05398H</td>
<td>05398H</td>
<td>000AH</td>
<td>W</td>
<td>GLOBALMODULE_D DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>05398H</td>
<td>05398H</td>
<td>000AH</td>
<td>W</td>
<td>MEMORY</td>
<td>MEMORY</td>
</tr>
</tbody>
</table>

175
** **

```plaintext
//** *** WRTACE *** WRTACE *** WRTACE *** WRTACE *** WRTACE *** WRTACE ***
// ** *** WRTACE *** WRTACE *** WRTACE *** WRTACE *** WRTACE ***
// ** *** WRTACE *** WRTACE *** WRTACE *** WRTACE *** WRTACE ***

SENDIF

"DWINTSFIAO, PRTS,CPUSUMFPR \ = 3 ;
ENDF;

PRTS.LASTRUN = 255; /* INDIATE START ENTRY TO
SCHEDULE */
CALL UPSCHEDULE;
*/
***/
END; /* LIMODULE */

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

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

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

*** MCOPTEX  MCOPTEX  MCOPTEX

ISIS-II MCS-86 LINKER, 1.1, INVOKED BY:
:F1:LINK26 :F1:LEVFL1.OPJ, :F1:LEVFL2.OPJ, :F1:SCHD.OPJ,
:F1:INITK.OPJ, :F1:GLOBAL.OPJ TO: :F1:KORF.LNK
LINK MAP FOR :F1:KORF.LNK(LIMODULE)

LOGICAL SEGMENTS INCLUDED:
LENGTH  ADDRESS  SEGMENT  CLASS
2265H ----  LIMODULE_CODE  CODE
089EH ----  LIMODULE_DATA  DATA
0C4CH ----  STACK  STACK
2227F ----  MEMORY  MEMORY
004CH ----  L2MODULE_CODE  CODE
00C7H ----  L2MODULE_DATA  DATA
0077H ----  L2SEG
0097F ----  SCHEDULE
001AH ----  INITMOC_CODE  CODE
0021H ----  INITMOC_DATA  DATA
0020H ----  GLOBALMODULE_C  CODF
0737F ----  GLOBALMODULE_D  DATA

INPUT MODULES INCLUDED:
:F1:LEVFL1.OPJ(LIMODULE)
:F1:LEVFL2.OPJ(LIMODULE)
:F1:SCHD.OPJ(SCHD)
:F1:INITK.OPJ(INITMOD)
:F1:GLOBAL.OPJ(GLOBALMODUL

177
```
/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE *****/
PRS.VPSP = PRS.VPSTART + 2;
PRDs.VPS_PSRP$U = 3;
/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE *****/
/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE *****/

ENDIF

/* INITIALIZE THE VP MAP FOR IDLE AND INIT PROC */
/* AND MONITOR PROCESS */
VM(PPDS.VPSTART).VPSP = 255;
VM(PPDS.VPSTART).STATE = 1;
VM(PPDS.VPSPRIORITY = 0;
VM(PPDS.VPthread = 255;
VM(PPDS.VPSAwSVALUE = 0;
VM(PPDS.VPSPREG = 60H;
VM(PPDS.VPSTART).SSSREG = INIT$STACK$SEG;
VM(PPDS.VPSTART+1).VPSP = 255;
VM(PPDS.VPSTART+1).STATE = 1;
VM(PPDS.VPSPRIORITY = 255;
VM(PPDS.VPthread = 255;
VM(PPDS.VPSAwSVALUE = 0;
VM(PPDS.VPSTART+1).SSSREG = 60H;
VM(PPDS.VPSTART+1).SSSREG = IDLE$STACK$SEG;

IF NOT MCORTEX

/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE *****/
/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE *****/
VM(PPDS.VPSTART+2).VPSP = 0FFH;
VM(PPDS.VPSTART+2).STATE = 7;
VM(PPDS.VPSPRIORITY = 0;
VM(PPDS.VPthread = 255;
VM(PPDS.VPSAwSVALUE = 0;
VM(PPDS.VPSTART+2).SSSREG = 60H;
VM(PPDS.VPSTART+2).SSSREG = MONITOR$STACK$SEG;

/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE *****/
/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE *****/

ENDIF

NV$RPS = VS$RPS + 1;

IF MCORTEX

/**** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX *****/
/**** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX *****/
NS$VPS = PRS.CPUNUMBER = 2;
/**** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX **** MCORTEX *****/

$FLSE

172
/* STARTING POINT OF THE OPERATING SYSTEM */

/* ROUTINE INITIALIZES THE CS AND IS NOT REPEATED. */

/* TO INITIALIZE THE PRDS TABLE FOR THIS CPU */

DECLARE CPU$STR_POINTER DATA (GPRDS.CPUSNUMBER),
22 BYTE;

DISABLE;

SIF NOT MCORETEX

**** MIXTRACE ***** MIXTRACE ***** MIXTRACE ***** MIXTRACE ***** /
**** MIXTRACE ***** MIXTRACE ***** MIXTRACE ***** MIXTRACE ***** /
CALL OUTSLINE (MO$G12);
SENDIF

/* INITIALIZE PPI AND PIC */
OUTPUT (PORTCC) = FCH; /* PPI - MICROPOLIS - MCORETEX */
OUTPUT (PORTCC) = 15H; /* PIC - ICW1 - EDGE TRIGGERED */
OUTPUT (PORTCC) = 40H; /* PIC-ICW2-VECTOR TABLE ADDRESS */
OUTPUT (PORTCC) = 0Fh; /* PIC-ICW4-MCS86 MODE, AUTO FCI */
OUTPUT (PORTCC) = 0AH; /*PIC-MASK ALLOWING INT. 4 & 6 */

/* ESTABLISH UNIQUE SEQUENTIAL NUMBER FOR THIS CPU */
/* SET GLOBALLOCK */
DO WHILE LOC$SET (GLOBALLOCK, 119) END;
PRDS.CPUSNUMBER = CPU$INIT;
CPU$INIT = CPU$INIT + 1;

/* ELSE SET GLOBAL LOCK */
GLOBALLOCK = 2;

/* SET UP INITIAL START AND END FOR PROC TABLE */
PRDS.VP$START = 0;
DO ZZ = 1 TO PRDS.CPUSNUMBER;
PRDS.VP$START = PRDS.VP$START + MAX$VPS$CPU;
END;

SIF MCORETEX

**** MOCORETEX ***** MCORETEX ***** MCORETEX ***** MCORETEX ***** /
**** MCORETEX ***** MCORETEX ***** MCORETEX ***** MCORETEX ***** /
PRDS.VP$END = PRDS.VP$START + 1;
PRDS.VP$START$CPU = 21;
**** MCORETEX ***** MCORETEX ***** MCORETEX ***** MCORETEX ***** /
ELSE

171
CALL OUTSCHAR(CR);  
CALL OUTSCHAR(LF);  
ADDR.OFFSET = ADDR.OFFSET + 1;  
PTR = PTP3;  
CALL OUTSNUM(ADDR.BASE);  
CALL OUTSCHAR(' ');  
CALL OUTSNUM(ADDR.OFFSET);  
CALL OUTSCHAR('-');  
CALL OUTSNUM(CONTENTS);  
ENDIF;  /* IF SKIP FOR NEXT SUB */  
IF (INCHR = ' ') THEN DO;  
CALL OUTSCHAR(' ');  
CALL IN$CHAR(INCHR);  
ENDIF;  /* IF GO TO NEXT ADDR */  
ENDIF;  /* IF CHANGE CONTENTS */  
INCHR = 'X';  /* RINITIALIZE CMD */  
ENDIF;  /* LOOP, CONTINUOUS SUR CMD */  
ENDIF;  /* SUBSTITUTE COMMAND SECTION */  
END;  /* IF */  
GO TO LOOP;  
ENDIF;  /* MONITOR PROCESS */  
/**** MXTRACF ***** MXTRACF ***** MXTRACF ***** MXTRACF *****/  
/**** MXTRACF ***** MXTRACF ***** MXTRACF ***** MXTRACF *****/  
ENDIF
CALL INSNUM(ADDR,OFFSET);
PTR2 = ADDR;
PTR = PTR3;
/* CONTENTS SHOULD NOW BE SET */
DO WHILE (INCHR<>'C') AND (INCHR<>'23a');
   CALL INSCAR(CHAR);
END; /* DO WHILE */
IF INCHR = 'C' THEN DO;
   CALL OUTSCHAR('');
   CALL OUTSNUM(CONTENTS);
   CALL OUTSCHAR(CR);
   CALL OUTSCHAR(LF);
END; /* IF NORMAL */
IF INCHR = '23a' THEN DO;
   COUNT = 0;
   CALL OUTSCHAR('');
   CALL INSNUM(QUANTITY);
   DO WHILE QUANTITY > 0;
      CALL OUTSCHAR(CR);
      CALL OUTSCHAR(LF);
      CALL OUTSNUM(ADDR,BASE);
      CALL OUTSCHAR('');
      CALL OUTSNUM(ADDR,OFFSET);
      LINCOMPLET = FALSE;
      DO WHILE LINCOMPLET = FALSE;
         CALL OUTSCHAR('');
         CALL OUTSNUM(CONTENTS);
         ADDR.OFFSET = ADDR.OFFSET + 1;
         PTR = PTR3;
         QUANTITY = QUANTITY - 1;
         IF ((ADDR.OFFSET AND $00FH)=0) OR
         (QUANTITY = 0) THEN LINCOMPLET = TRUE;
      END; /* DO WHILE LINE NOT COMPLETE */
   END; /* DO WHILE */
   END; /* IF MULTI */
END; /* DISPLAY COMMAND SECTION */
IF INCHR='S' OR (INCHR='23a') THEN DO;
   /* SUBSTITUTE COMMAND SECTION */
   CALL INSNUM(ADDR,FASP);
   CALL OUTSCHAR('');
   CALL INSNUM(ADDR.OFFSET);
   CALL OUTSCHAR('');
PT2 = ADDR;
PTR = PTR3;
/* CURRENT CONTENTS SHOULD NOW BE AVAILABLE */
CALL OUTSNUM(CONTENTS);
LOOP2 = TF;
DO WHILE LOOP2 = TRUE;
   DO WHILE (INCHR<>'.' AND (INCHR<>' ')
      AND (INCHR<>'C'));
      CALL INSCAR(char);
   END;
IF (INCHR = CP) THEN LOOP2 = FALSE;
IF (INCHR = '. ') THEN DO:
PRTS.COUNTER = PRTS.COUNTER + 1
GO TO LOOP;
FND; /* TPTRSPROC */

FLICT VP PRPS.

*/
/
THE MONITOR PROCESS IS INITIALIZED BY THE OS LIKE
*/
INIT AND IDLE. IT HAS THE RESERVED ID OF 0F4 AND A
*/
PRIORITY OF 04. IT IS ALWAYS BLOCKED OR WAITING UNTIL*
*/
IT IS PREPARED COR THE USE.
*/
*/
CALLS MADE TO: OUTS LINE OUTS CHAR
*/
OUTSDNUM INS DNUM */
/*
*/
SIF NOT *CORTEX

**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****
**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
MONITMRSPROC: PROCEDURE REENTRANT PUBLIC;

DECLARF
PTR POINTER,
PTR2 POINTER,
PTR3 BASED PTR2 POINTER,
ADDR STRUCTURE (OFFSET WORD, BASE WORD),
CONTENTS BASED PTR BYTE;

DECLARF
(LINFCOMPLTE, LOOP2) BYTE,
(QUANTITY, COUNT) BYTE;
(INCHR, INDEX, VALIDSCMD) BYTE;

LOOP: VALIDSCMD = 0;
CALL OUTS CHAR (CP);
CALL OUTS CHAR (LP);
CALL OUTS CHAR (".");
DO WHILE NOT VALIDSCMD:
CALL INCHR (INCHR);
IF (INCHR = 'P') OR (INCHR = 'S') OR (INCHR = 'R') THEN
VALIDSCMD = OFFH;
IF (INCHR=6A) OR (INCHR=65) OR (INCHR=73) THEN
VALIDSCMD = OFFH;
IF VALIDSCMD = OFFH THEN CALL OUTS CHAR (INCHR);
END; /* TO WHILE */
IF (INCHR = 'R') OR (INCHR = 64) THEN DO;
/* DISPLAY COMMAND SECTION */
CALL INS DNUM (ADSR.BASE);
CALL OUTS CHAR (": ");

1FP
/* SYSTEM PROCESSES */
*/

/* IDLE PROCESS */
/* THIS PROCESS IS SCHEDULED IF ALL OTHER PROCESSES ARE BLOCKED. THE STARTING ADDRESS IS PROVIDED */
/* TO THE IDLESSTACK AND PLACED IN PRE bloginfo:bgf9-14 */
/* CALL TO THE SCHEDULE IS MADE EVERY 4 MS IN THE */
/* EVENT THAT AN ONBOARD PROCESS WAS LATCHED BY AN */
/* OFFBOARD OPERATION (ADVANCE). EVERY 250 ITERATIONS */
/* THE COUNT IS INCREMENTED BY ONE. THIS, THE COUNT IS */
/* INCREMENTED ONCE PER SECOND, THE COUNT IS MAINTAINED */
/* IN THE PRESC_TABLE AND IS A ROUGH MEASURE OF SYSTEM */
/* PERFORMANCE BY GIVING AN INDICATION OF THE AMOUNT OF */
/* TIME SPENT IN THE IDLE PROCESS. */
/*----------------------------------------------*/
/* CALLS MADE TO: PLM96 PROCEDURE 'TIME' */
/* OUTSLINE */

IDLE$PROC: PROCEDURE REENTRANT PUBLIC;
DECLARE I PYTF;

$IF NOT MCCRTFX

/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
CALL OUTSLINE(PROMSG10);

$ENDIF

LOOP: DO I = 1 TO 250;
    /* 4 MS DELAY */
    CALL TIME( 40 );
    IF WHILE LOCK$SET 'GLOBAL$LOCK', 110); /* ASSERT LOCK */
    END;
    CALL RLYTHISSUP;
    CALL VPSCHEDULE;
END; /* DO I */

$IF NOT MCCRTFX

/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
/**** MXTRACE **** MXTRACE **** MXTRACE **** MXTRACE ****
CALL OUTSLINE(PROMSG11);

$ENDIF
ENDIF

RETURN 3PCCTRLINDEX;
END; /* IF */
ELSE PC;

IF NOT MCO/TEX

**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
**** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE *****/
CALCOUTSLINE('MSG24');

ENDIF

RETURN NOTFOUND;
END; /* FISK */
END; /* LOCATRSSR PROCEDURE */
/* RETURN NOT FOUND CODE */

$IF NOT MCCORTEX

/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE **** /
/**** MTRACE **** MTRACE **** MTRACE **** MTRACE **** MTRACE **** /
CALL OUTSLINE(@MSG24);

$ENDIF

RETURN NOT FOUND;
END; /* ELSE */
$ENDIF /* LOCATE$SVC PROCEDURE */

/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE */
/***** LOCATE$SEQ PROCEDURE*/
/* FUNCTION CALL.  RETURNS THE INDEX IN EVENTCOUNTER TABLE */
/* OF THE EVENT NAME PASSED TO IT. */
/* CALLS MADE TO: OUT$CHAN OUT$LINE */
/****************************************************************************/

LOCATE$EVC:  PROCEDURE(EVENTSNAME) BYTE REENTRANT PUBLIC:

DECLARE EVENTSNAME BYTE:
DECLARE (MATCH, EVCTBL$INDEX) BYTE:

$IF NOT MCORTEX

**** MXTRACF **** MXTRACF **** MXTRACF **** MXTRACF ****
**** MXTRACF **** MXTRACF **** MXTRACF ****
CALL OUT$LINF(@MSG28);

$ENDIF

MATCH = FALSE;
EVCTBL$INDEX = 0;
/* SEARCH DOWN THE EVENTCOUNTER TABLE TO LOCATE THE */
/* DESIRED EVENT BY MATCHING THE NAMES */
DO WHILE (MATCH = FALSE) AND (EVCTBL$INDEX < EVENTS);
/* REACHED END OF THE TABLE */
IF EVENTSNAME = EVCTBL$INDEX.EVCTBL$INDEX THEN
  MATCH = TRUE;
ELSE
  EVCTBL$INDEX = EVCTBL$INDEX + 1;
END: /* WHILE */

$IF NOT MCORTEX

**** MXTRACF **** MXTRACF **** MXTRACF **** MXTRACF ****
**** MXTRACF **** MXTRACF **** MXTRACF ****
CALL OUT$LINF(@MSG28);

$ENDIF

RETURN EVCTBL$INDEX:
END:
Else DO:
### MEMORY MAP OF MODULE L1MODULP

**FILE FROM FILE: P1:FORP.LNK**

**WRITTEN TO FILE: P1:FORP**

<table>
<thead>
<tr>
<th>START</th>
<th>STOP</th>
<th>LENGTH</th>
<th>ALIGN</th>
<th>MAP CLASS</th>
<th>OFFSET</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C110H</td>
<td>0C113H</td>
<td>0004H</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>00000H</td>
<td>0000H</td>
</tr>
<tr>
<td>0C330H</td>
<td>0C333H</td>
<td>0003H</td>
<td>W</td>
<td>INITMOD_CODE</td>
<td>CODE</td>
<td>0000H</td>
</tr>
<tr>
<td>0CA20H</td>
<td>0CABEH</td>
<td>0DEH</td>
<td>W</td>
<td>L1MODULE_CODE</td>
<td>CODE</td>
<td>0000H</td>
</tr>
<tr>
<td>0B4CEH</td>
<td>0C55AEH</td>
<td>0DEH</td>
<td>W</td>
<td>L2MODULE_CODE</td>
<td>CODE</td>
<td>0000H</td>
</tr>
<tr>
<td>0C206H</td>
<td>0C2C6H</td>
<td>000AH</td>
<td>W</td>
<td>GLOBALMODULE_C</td>
<td>CODE</td>
<td>0000H</td>
</tr>
<tr>
<td>0C2C6H</td>
<td>0C306H</td>
<td>0133H</td>
<td>W</td>
<td>L1MODULE_DATA</td>
<td>DATA</td>
<td>0000H</td>
</tr>
<tr>
<td>0C320H</td>
<td>0C345H</td>
<td>025H</td>
<td>W</td>
<td>L2MODULE_DATA</td>
<td>DATA</td>
<td>0000H</td>
</tr>
<tr>
<td>0C412H</td>
<td>0C470H</td>
<td>058H</td>
<td>G</td>
<td>?3SEG</td>
<td>-</td>
<td>0000H</td>
</tr>
<tr>
<td>0C4F6H</td>
<td>0C536H</td>
<td>002H</td>
<td>G</td>
<td>SCHEDULE</td>
<td>-</td>
<td>0000H</td>
</tr>
<tr>
<td>0C536H</td>
<td>0C5F2H</td>
<td>036H</td>
<td>W</td>
<td>STACK</td>
<td>STACK</td>
<td>0000H</td>
</tr>
<tr>
<td>0C640H</td>
<td>0C6A8H</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>-</td>
<td>0000H</td>
</tr>
<tr>
<td>0C6BCH</td>
<td>0C722CH</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>-</td>
<td>0000H</td>
</tr>
<tr>
<td>0C730H</td>
<td>0C7AEH</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>-</td>
<td>0000H</td>
</tr>
<tr>
<td>10200H</td>
<td>10477H</td>
<td>007AH</td>
<td>A</td>
<td>(ABSOLUTE)</td>
<td>-</td>
<td>0000H</td>
</tr>
<tr>
<td>0538EH</td>
<td>05436H</td>
<td>078H</td>
<td>W</td>
<td>GLOBALMODULE_D</td>
<td>DATA</td>
<td>0000H</td>
</tr>
<tr>
<td>05A89H</td>
<td>05A99H</td>
<td>000AH</td>
<td>W</td>
<td>MEMORY</td>
<td>MEMORY</td>
<td>0000H</td>
</tr>
</tbody>
</table>
APPENDIX I

SCHEDULER & INTERRUPT HANDLER SOURCE CODE

The ASMEF code in file SCHFD.ASM is part of the LEVEL I module. Details pertaining to assembler invocation may be found in [Ref. 20] and [Ref. 21]. This module is linked into file K07E.LNK and its memory map is included in the map for K07F.
THE FOLLOWING ARE THE EXTERNAL PLUG IN PROCEDURES CALLED BY THIS MODULE.

EXTERN SAV$CONTEXT: FAR
EXTERN GET$P: FAR
EXTERN GET$WORK: FAR
EXTERN DYTHISV$P: FAR
EXTERN PRE$S: BYTE
EXTERN HWIN$FLAG: BYTE
EXTERN GLOBAL$LOCK: BYTE

SCHEDULER SEGMENT
PUBLIC VPSCHEDULER
PUBLIC INTTFC

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,OF

;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 RP
POP CY
MOV AX,SP
PUSH AX
PUSH SS
CALL SAVECONTEXT
CALL GET$WORK
PUSH AX
CALL GET$SP
POP SS
MOV SP, AX

;SWAP VIRTUAL PROCESSOR CONTEXT COMPLETE AT THIS POINT
;NOW OPERATING IN NEWLY SELECTED PROCESSOR STACK

167
POP CX ; GET INET_1ND FLAG
POP PP ; INSTALL NEW STACK BASE

; CHECK FOR RETURN TYPE, NORMAL OR INTERRUPT

CMP CX, 77H
JZ INTOMET

NORMAL RET: POP DS
; UNLOCK GLOBALSLOCK
MOV AX, SRG GLOBALSLOCK
MOV ES, AX
MOV ES:GLOBALSLOCK, 0
STI
HRET

VPSCHEUDULER PNIDP

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

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

; INTERRUPT_HANDLER PROC NFar

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

INTVEC: CLI
PUSH ES ; SAVE NEEDED REGS TO TEST INTERRUPT FLAG
PUSH BX
PUSH AX
PUSH CX
CALI HARDWARE_INT_FLAG
MOV AL, 0
XCHG AL, FS:HDWINTFLAG[BX]

CMP AL, 77H ; IS INT FLAG ON ?
JZ PUSH_REST_REGS ; IF 'YES' SAVE REST REGS
POP CX ; IF 'NO' RESUME PREVIOUS
POP AX
POP BX
POP ES
STI
HRET

PUSH_REST_REGS: PUSH DX ; FLAG WAS ON SO NEED
PUSH DS
PUSH SI

191
PUSH DI
MOV AX, SEG GLOBALLOCK
MOV ES, AX

; LOCK GLOBAL LOCK
LOCK XCHG FS:GLOBALLOCK, AL
TEST AL, 4L
JNZ CK
CALL REYTHISVP
MOV CX, 77H
; JUMP TO SCHEDULER
JMP INTJOIN

INTRET: POP DI
POP SI
POP DS
POP DX
POP CX
; RETURN FROM
JMP INTJOIN

; UNLOCK GLOBAL LOCK
MOV AX, SEG GLOBALLOCK
MOV ES, AX
MOV ES:GLOBALLOCK, 0
POP AX
POP BX
POP ES
STI
IRET

INTERRUPT_HANDLER FNDP

;**********************************************************************
;**********************************************************************
;* Hardware Interrupt Flag
;**********************************************************************
;**********************************************************************

; Hardware Interrupt Flag PROC NEAR
ASSUME CS: SCHEDULER
ASSUME DS: NOTHING
ASSUME SS: NOTHING
ASSUME ES: NOTHING

; Segment for Interrupt Handling
HW_DFLG: MOV 'S',SEG PINTFL
MOV FS, AX
MOV AX, 0FH
MOV CL, ES: PRTS [RX]
JMP CPU #
MOV CH, 0
RET IN EX
MOV BX, CX
MOV AX, SEG HDWINTFLAG
MOV ES, AX
; RET IN ES : SEG

; Hardware Interrupt Flag FNDP
SCHEDULER ENDS
END

182
APPENDIX J

GLOBAL DATA BASE AND INITIAL PROCESS CODE

Two files are contained in this appendix: GLOBAL.SHC AND INITK.SHC. They are separately compiled with the LARGE attribute. They are linked into the file: KOBE.LNK. They are represented in the memory map for KOBE presented at the end of Appendix F. INITK will be overwritten by an initialization module on each real processor.
GLOBAL$MODULE: DO:

DECLARE
MAX$CPU LITERALLY '10',
MAX$VPS$CPU LITERALLY '10',
MAX$CPU=MAX$VPS$CPU LITERALLY '100';

DECLAFF
GLOBEALSLOCK BYTE PUBLIC INITIAL(0);

/* THIS SHOULD REFLECT THE MAX$CPU ABOVE */

DECLAFF
NR$RPS BYTE PUBLIC INITIAL(0),
NR$VPS(MAX$CPU) BYTE PUBLIC INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

DECLAFF
HDWSINT$FLAG(MAX$CPU) BYTE PUBLIC;

DECLAFF
EVENTS BYTE PUBLIC INITIAL(1);

DECLAFF
LOCAL$CLUSTER$ADDR WORD PUBLIC;

DECLAFF
EVC$STR(100) STRUCTURE
(EVC$NAME BYTE,
VALUE WORD,
REMOTE$ADDR WORD, 
194
THREAD PUBLIC
INITIAL(0FFH,0,0FFFFH.255);
/* EVO "F" IS RESERVED FOR THE OP SYS */

DECLARE CPUINIT BYTE PUBLIC INITIAL(?);
DECLARE SUFFIXERS BYTE PUBLIC INITIAL(0);

DECLARE SEQUESTABLE(10?) STRUCTURE
SEQSNAMF BYTE,
SEQSVALUE WORD PUBLIC;

DECLARE VPM( MAX:CPUSSS$SMA:VPSSCPU ) STRUCTURE
VPSID BYTE,
VPSSTATE BYTE,
VPSPRIOCY BYTE,
EVCTHRFGD BYTE,
EVCSAW$VALUE WORD,
SP$PEG WORD,
SS$PEG WORD) PUBLIC;

END; /* MODULE */

******************************************************************************
**INIT** MODULE BREWER C-13-54

---

/* THE CODE SEGMENT OF THIS MODULE IS WHAT RESERVES SPACE */
/* BY THE CS FOR THE USER INITIAL PROCESS. THIS IS */
/* EXECUTABLE IN IT'S OWN RIGHT, THUS IF THE USER DOES */
/* NOT PROVIDE AN INITIAL PROCESS THIS ONE WILL EXECUTE, */
/* BLOCK ITSELF, AND IDLE THE CPU. THE ADDRESS OF THE */
/* INITIAL CODE SEGMENT IS PROVIDED TO LEVEL1 AND IT IS */
/* REFLECTED IN THE PLM LOCATE COMMAND. THE ADDRESSES */
/* PROVIDED MUST AGREE. THIS PROCESS HAS THE HIGHEST */
/* PRIORITY AND WILL ALWAYS BE SCHEDULED FIRST BY THE */
/* SCHEDULE. */
---

/* CALLS MADE TO: AWAIT */
/* INIT$MOD: TO: */

```plaintext
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/
/* DECLARE */
/* MSG13(*) BYTE INITIAL(10, 'ENTERING INITIAL PROCESS ', */
/* 13,10, ', ') ; */
/* OUT$LINE: PROCEDURE( PTR ) EXTERNAL; */
/* DECLARE PTR POINTER; */
/* END; */
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/
AWAIT: PROCEDURE( NAME, VALUE ) EXTERNAL; */
DECLARE NAME BYTE, VALUE WORD; */
END;

INITIAL$PROC: PROCEDURE PUBLIC:

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 MODULE. */
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/
/* CALL OUT$LINE(0(MSG13); */
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/
/**** MXTACE ***** MXTACE ***** MXTACE ***** MXTACE ***/

CALL :WAIT( OFFH, 1); */
END; /* INITIAL$PROC */
END; /* INIT$MOD */
```
APPENDIX K

NICNIC DEVICE DRIVER AND PACKET PROCESSOR SOURCE CODE

This code consists of PL/I-86 modules and 8086 assembly language modules. PL/I-86 is primarily an applications programming language, rather than a systems development language. As such, it does not have the language features to gain access to the 8086 processor or MULTIBUS hardware. In situations where it is necessary to access hardware-dependent components, RASMP6 (Ref. 18) modules are called. These assembly language routines are located in file ASMRUT.86 (assembly language routines), and are linked with the PL/I-86 modules.

As described in detail in Chapter IV, the Driver is a MCORTEX system process with a dedicated real processor. Its linking conventions and use of MCORTEX primitives are identical to any user process. The notable exception is the use of its initialization module to define the cluster address, create sequencers, create eventcounts, and distribute the eventcounts.

The Driver also reads a file called ADDRESS.DAT to determine its own physical Ethernet address and addresses to load into its multicast (or group) address table. Note the type of data in ADDRESS.DAT must be bit string for addresses, and fixed binary for the number of group addresses.
The SYSINIT1.PLI file is the initialization module for Cluster 1 and SYSINIT2.PLI is the initialization module for Cluster 2. These files and ADDRESS.DAT are the only system files that must be changed when new MCORTEX processes are added, causing a change in eventcount distributivity. MCORTEX processes may be readily ported in executable image form from one cluster to another. The eventcount distribution changes only require a change in the Driver initialization modules and the cluster ADDRESS.DAT files. The amount of recompilation and linking is kept to the absolute minimum with this schema.

The contents of SYSDEF.PLI (Appendix E), ADDRESS.DAT, and the Driver initialization modules reflect the current system configuration. This is the demonstration process described in Appendix F.

Due to thesis format requirements, the structure of the source code is slightly altered, i.e., PL/I statements are not necessarily compilable as illustrated.
sysinit1: proc options (main);

#include 'sysdef.pli';

#replace
EVC TYPE     by '00'b4;

/* main */

call define_cluster ('0001'b4); /* must be called prior to creating evc's */

/*** USER? ****/

call create_evc (TRACK_IN);
call create_evc (TRACK_OUT);
call create_evc (MISSILE_ORDER_IN);
call create_evc (MISSILE_ORDER_OUT);

/*** SYSTEM ***/

call create_evc (REB_READ);
call create_evc (FPR_WRITE);
call create_seq (FPR_WRITE_REQUEST);

/* distrib. map called after eventcounts have been created */

call distribution_map (EVC_TYPE, TRACK_IN, '0003'b4); /* local and remote copy of TRACK_IN needed */
call distribution_map (EVC_TYPE, MISSILE_ORDER_OUT, '0003'b4);
call create_orca ('fc'b4, 'pf'b4, '950'b4, '950'b4, '55f'b4, '55f'b4,
call await ("fe", "hl");

erd sysinit1:

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

### Cluster 2 ADD5PSS.DAT file

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

1.

```plaintext
"10000000b","7007010b",
"30000000b","20000010b"
```

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

### SYSINIT2.PLI file

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

sysinit2: proc options (main):

```plaintext
#include 'sysdef.pli';

replace

EVC_TYPE by '00'b4;

/* main */

call define_cluster ('3222'b4); /* must be called prior to creating evc's */

/***** USER ****/

call create_evc ('TRACK_IN');
call create_evc ('TRACK_OUT');
call create_evc ('MISSILE_ORD_IN');
call create_evc ('MISSILE_ORD_OUT');

/**** SYSTEM ****/

call create_evc ('PRP_UPAT');
call create_evc ('PRP_UPAT');
call create_seq ('PRP_UPAT_REQUEST');
```

190
/* distrib.map called after event counts have
 been created */
call distribution_map 'PVC_TYPE,TRACK_OUT'('V3', '14');
/* local and remote copy of TRACK_IN needed */
call distribution_map 'PVC_TYPE,MISSILE_OPDET_IN',
 '0773', 'b4');
/* local and remote copy of MISSILE_OPDET_IN needed */
call createProc ('fc', 'b4', '30', 'b4',
 '2958', 'b4', '2987', 'b4', '2953', 'b4',
 '0430', 'b4', '0280', 'b4', '2860', 'b4');
call await ('fc', 'b4', '0001', 'b4');
end sysinit;

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

NI3010.DCL file
***

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

/*replace

I/O port addresses
These values are specific to the use of the INTERLAN
NI3010 MULTIPUS to ETHERNET interface board. Any change
in the I/O port address of '22b8' hex (done so with a DIP
switch) will require a change to these addresses to
reflect that change.

*/

command_register by 'b4',
command_status_register by 'b1',
transmit_data_register by 'b2',
interrupt_status_reg by 'b5',
interrupt_enable_register by 'b8',
high_byte_count_reg by ',
low_byte_count_reg by 'b4',

/* end of I/O port addresses */

/* Interrupt enable status register values */
disable ni3010 interrupts by 'b0',
i3010 interrupts disabled by 'b0',
receive_block_available by 'b4',
transmit dma done by 'b6',
receive_dma done by 'b7',

191
/* end register values */

/* Command Function Codes */

module_interface_loopback by '01'b4,
internal_loopback by '02'b4,
clear_loopback by '03'b4,
go_offline by '08'b4,
go_online by '09'b4,
onboard_diagnostic by '0a'b4,
clr_insert_source by '0e'b4,
load_transmit_data by '28'b4,
load_and_send by '29'b4,
load_group_addresses by '2a'b4,
reset by '3f'b4;

/* end Command Function Codes */

SYSDEV.PLI file

sysdev: procedure;

/* Date: */
1 SEPTEMBER 1984

Programmer: David J. BREWER

Module Function: To serve as the Ethernet Communication Controller Board (NI3010) device handler. This process is scheduled under MCORTEX and consumes Ethernet Requests Packets (EPP) generated by the SYSTEMSIO routine in LEVEL2.SRC. It also processes any inbound packets by analyzing the packet contents and making the appropriate MCORTEX calls.

/*

%replace

evo_type by '00'b4,
erb_block_len by 20,
erb_block_len_ml by 19,
infinity by 32767;

%include 'sysdef.pli';

192
DECLARE

1 erb(O:erb_block_len_m1) based (block ptr),
   2 command  bit (6),
   2 type_name bit (8),
   2 name_value bit (16),
   2 remote_addr bit (16);

DECLARE

1 transmit_data_block based (xmit_ptr),
   2 destination_address_a bit (8),
   2 destination_address_b bit (8),
   2 destination_address_c bit (8),
   2 destination_address_d bit (8),
   2 destination_address_e bit (8),
   2 destination_address_f bit (8),
   2 source_address_a bit (8),
   2 source_address_b bit (8),
   2 source_address_c bit (8),
   2 source_address_d bit (8),
   2 source_address_e bit (8),
   2 source_address_f bit (8),
   2 type_field_a bit (8),
   2 type_field_b bit (8),
   2 data (46) bit (8).

1 receive_data_block based (rcv_ptr),
   2 frame_status bit (8),
   2 null_byte bit (8),
   2 frame_length_lsb bit (8),
   2 frame_length_msb bit (8),
   193
destination_address_a bit (8),
destination_address_b bit (8),
destination_address_c bit (8),
destination_address_d bit (8),
destination_address_e bit (8),
destination_address_f bit (8),
source_address_a bit (8),
source_address_b bit (8),
source_address_c bit (8),
source_address_d bit (8),
source_address_e bit (8),
source_address_f bit (8),
type_field_a bit (8),
type_field_b bit (8),
data(46) bit (8),
crc_msb bit (8),
crc_upper_middle_byte bit (8),
crc_lower_middle_byte bit (8),
crc_lsb bit (8),

(xmit_ptr, rcv_ptr, block_ptr) pointer,
index fixed bin (15),
(addr_e, addr_f) bit (8),
address file,
copy_ie_register bit (8),
(cluster_addr, erb_write_value, i) bit (16),
(j,k) fixed bin (15),
reg_value bit (8),
write_io_port entry (bit (8), bit (8)),
read_io_port entry (bit (8), bit (8)),
initialize_cpu_interruption entry,
enable_cpu_interruption entry,
disable_cpu_interruption entry,
write_bar entry (bit(16));

/\* end module listing */

%replace

/\* codes specific to the Intel 8259a Programmable
Interrupt Controller (PIC) */

/* note that */ icw1_port_address by 'c0'b4,
/* icw2, icw4 */ icw2_port_address by 'c2'b4,
/* and now */ ocw_port_address by 'c2'b4,
/* use same */
/* port addr */

/* note: icw => initialization
control
word

ocw => operational
command
 word */

icw1 by '13'b4,

/* single PIC configuration, edge
triggered input */

icw2 by '40'b4,

/* most significant bits of vectoring
byte: for an interrupt 5, the effective address will be
(icw2 + interrupt #) * 4 which will be (40 hex + 5) * 4 =
114 hex */

icw4 by '0f'b4,

/* automatic end of interrupt
and buffered mode/master */

ocw1 by '8f'b4;

/* unmask interrupt 4 (bit 4), */
/* interrupt 5 (bit 5), and */
/* interrupt 6 (bit 6), mask all others */

/* end 8259a codes */

/* include constants specific to the NI3010
board */

#include 'ni3010.dcl';
/** Main Body **/

call write_io_port (interrupt_enable_register, disable_N13010_interruption);  
call initialize_PIC;  
call initialize_cpu_interruption;  
call read_io_port (command_status_register, reg_value);  
call perform_command (reset);  
call program_group_addresses;  
/* assignments to the source and destination address fields that will not change */  
call perform_command (clr_insert_source);  
/* N13010 performance is enhanced in this mode */  
unspec(block_ptr) = block_ptr_value;  
unspec(rcv_ptr) = rcv_ptr_value;  
unspec(xmit_ptr) = xmit_ptr_value;  
/* make one time assignments to transmit data block */  
transmit_data_block.destination_address_a = '03'b4;  
transmit_data_block.destination_address_b = '00'b4;  
transmit_data_block.destination_address_c = '00'b4;  
transmit_data_block.destination_address_d = '00'b4;  
transmit_data_block.source_address_a = '03'b4;  
transmit_data_block.source_address_b = '00'b4;  
transmit_data_block.source_address_c = '00'b4;  
transmit_data_block.source_address_d = '00'b4;  
/* set the local cluster address - file was opened in proc program_group_addresses */  
set file (address) list (addr_e, addr_f);  
transmit_data_block.source_address_e = addr_e;  
transmit_data_block.source_address_f = addr_f;  
cluster_addr = addr_e || addr_f;  
pit skip (2) edit ("*** CLUSTER \', cluster_addr, ' Initialization Complete ***")  
(col(15), a, b4(4), a);  
i = '0001'b4;  
call perform_command (go_online);  
/* at this point copy_ie_reg = PBA, but ie_reg on N13010 is actually disabled */  
call disable_cpu_interruption;
do k = 1 to infinity;
    /* note: interrupt not allowed during a
       call to MCO8TEX primitive */
    
erb_write_value = read(EBB_WRITE);
    /* In the MXTRACE version of the RTOS
       all primitive calls clear and
       set interrupts (diagnostic message
       routines), so the NI3010 interrupts
       must be disabled on entry to MXTRACE */
    do while (erb_write_value < 1);
        /* busy waiting */
       erb_write_value = read(EBB_WRITE);
copy_ie_register = receive_block_available;
call write_io_port(interrupt_enable_register,
    receive_block_available);
call enable_cpu_interrupts;
    /* if a packet has been received, this
       is when an interrupt may occur — can
       see that outbound packets are always
       favored. */
do j = 1 to 1000;
    /* interrupt window for packets received */
end; /* do j */
call disable_cpu_interrupts;
if (copy_ie_register = receive_dma_done) then
    do;
        /* receive DMA operation started, so let
           finish. */
call enable_cpu_interrupts;
do while (copy_ie_register = receive_dma_done);
end;
call disable_cpu_interrupts;
end; /* lift */
copy_ie_register = disable_ni3010_interrupts;
call write_io_port(interrupt_enable_register,
    disable_ni3010_interrupts);
end; /* busy */

/* F3B has an ERP in it, so process it */
/* no external interrupts (RBA) until
   the ERP is consumed and the packet
   gets sent */
index = mod((fixed(i) - 1),.erb_block_len);
    /* 32-bit limit on parameter to fixed fcn. */
transmit_data_block.data(1) = erb(index).command;
transmit_data_block.data(2) = erb(index).type_name;
transmit_data_block.data(3) = substr(erb(index).name_value, 9,8);
transmit_data_block.data(4) = substr(erb(index).name_value, 1,2);
transmit_data_block.destination_address_e = substr(erb(index).remote_addr, 1,2);
transmit_data_block.destination_address_f = substr(erb(index).remote_addr, 9,8);

call advance (FPB_READ); /* caution here !!!!!
an ADVANCE will result in a
call to VPSCHEDULEn, which
will set CPU interrupts on exit.
It's the reason N3010 interrupts
are disabled first in the
Do While loop above. */

/*/ packet ready to go, so send it */
call transmit_packet;
/* copy ie_register = RPA , but not actual register */
call disable_cpu_interruption;

/*/ setting up for next TCP consumption */
i = add2bit16(i, '0001'04);
end; /* do forever */
/*/ end main body */

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

initialize_pic: procedure;

DECLARE
write_io_port entry (bit (3), bit(8));
call write_io_port (icw1_port_address, icw1);
call write_io_port (icw2_port_address, icw2);
call write_io_port (icw4_port_address, icw4);
call write_io_port (ocw_port_address, ocw1);
end initialize_pic;

/*****************************/
perform_command:     procedure (command);

DECLARE
    command bit (8),
    reg_value bit (8),
    srf bit (8),
    write_io_port entry (bit (8), bit (8)),
    read_io_port entry (bit (8), bit (8));

    /* end declarations */

    srf = '0'b4;
    call write_io_port (command_register, command);
    do while ((srf & '01'b4) = '00'b4);
        call read_io_port (interrupt_status_reg, srf);
    end; /* do while */
    call read_io_port (command_status_register, reg_value);
    if (reg_value > '01'b4) then
        do:
            /* not (SUCCESS or SUCCESS with Retries) */

            put skip edit ("*** Ethernet Board Failure ***")
                (col(29),a);
            /* when this occurs, run the diagnostic 
            routine T3010/Cx, where r is the 
            current cluster number */

            stop;
        end; /* itd */

    end_perform_command;

transmit_packet: procedure external;

DECLARE
    srf bit (8),
    reg_value bit (8),
    write_io_port entry (bit (8), bit (8)),
    read_io_port entry (bit (8), bit (8)),
    enable_cpu_interrupts entry,
    disable_cpu_interrupts entry,
    write_bar entry (bit(16));
/* begin */

srf = '9'b;
call write_bar (xmit_ptr_value);
call write_in_port(high_byte_count_reg,'05'b4);
call write_in_port(low_byte_count_reg,'3c'b4);
copy_ie_register = transmit_dma_done;
call write_in_port(interrupt_enable_register, transmit_dma_done);
call enable_cpu_interrunts;
do while (copy_ie_register = transmit_dma_done);
end:
"* loop until the interrupt handler takes care of the TDC interrupt - it sets copy_ie_register = :H4 */
call perform_command (load_and_send);
end transmit_packet;

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

UL_interrupt_handler: procedure external;

/* This routine is called from the low level 8086 assembly language interrupt routine */

DECLARE

write_io_port entry (bit (A), bit (E)),
read_io_port entry (bit (E), bit (8)),
enable_cpu_interrupts entry,
disable_cpu_interrupts entry,
write_bar entry (bit(16));

/* begin */
call write_in_port(interrupt_enable_register,
disable_n13C18_interrupts);
if (copy_ie_register = receive_block_available)
then do:

call write_bar (rcv_ptr_value);
call write_io_port(high_byte_count_reg,'05'b4);
call write_in_port(low_byte_count_reg,'2b4');
"* initiate receive DMA */
copy_ie_register = receive_dma_done;
call write_in_port(interrupt_enable_register,
receive_dma_done);
end;  /* do */
else
  if (copy ie_register = receive_dma_done) then do:
    call process_packet;
    copy ie_register = receive_block_available;
    call write_in_port(interrupt_enable_register,
                       receive_block_available);
  end;  /* if then do */
else
  if (copy ie_register = transmit_dma_done)
    then do:

    copy ie_register = receive_block_available;
    /* MIU010 interrupts disabled on entry */
  end;  /* if then do */
end HL_interrupt_handler;

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

process_packet: procedure;

DECLARE

  local_evc_value bit (16),
  data_ptr pointer,
  remote_evc_value bit (16) based (data_ptr);

if (receive_data_block.data(1) = evc_type) then do:
  data_ptr = addr(receive_data_block.data(3));
  /* remote_evc_value now has a value */
  local_evc_value = read(receive_data_block.data(2));
  do while (local_evc_value < remote_evc_value);
    call advance (receive_data_block.data(2));
    local_evc_value = add2bit16(local_evc_value,
                                  "001"h4);
  end;
  call disable_cpu_interrupts;
  /* this must be done due to setting of
  cpu interrupts by calls to CORTEX'S
  VPSCHEDULER via ADVANCE */
end; /* iter */
  /* Only type packet in this limited implem. */
end process packet;

/*************************************************************/
program_group_addresses: procedure;

DECLARE

1 group_addr(40) based (group_ptr),
2 mc_group_field_a
  bit (3),
2 mc_group_field_b
  bit (9),
2 mc_group_field_c
  bit (9),
2 mc_group_field_d
  bit (8),
2 mc_group_field_e
  bit (8),
2 mc_group_field_f
  bit (8);

DECLARE

(group_ptr,p) pointer,
(field_e, field_f) bit (9),
bit_0_groups bit (9) based (p),
(i,num_groups,groups,times_6) fixed bin (7);

unspec(group_ptr) = rmit_ptr_value;
open file (address) stream input;
set file (address) list (num_groups);
do i = 1 to num_groups:

  group_addr(i).mc_group_field_a = '03'b4;
  group_addr(i).mc_group_field_b = '00'b4;
  group_addr(i).mc_group_field_c = '20'b4;
  group_addr(i).mc_group_field_d = '00'b4;
  set file (address) list (field_e,field_f);
  group_addr(i).mc_group_field_e = field_e;
  group_addr(i).mc_group_field_f = field_f;

end; /* do i */
call disable_cpu_interrupts;

202
call write_bar (xmit ptr value);
call write_io_port (high_byte_count_reg, '00' hi);
grups times 8 = 6 * num groups;
    v = addr (groups times 6);
call write_io_port (low_byte_count_reg, bit 8 groups);
copy ie_register = transmit_dma_done;
call write_io_port (interrupt_enable_register, 
    transmit_dma_done);
call enable_cpu_interrups;
do while (copy ie_register = transmit_dma_done);
end;  /* loop until the interrupt handler 
    takes care of the TDD interrupt 
    it sets COPY_IF_FG = FRA */
call perform_command (load_group_addresses);
end program_group_addresses;

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

end;  /* system device handler and packet processor */
operation = await_packet;
srf = 0;
call disable_cpu_interrupts;
copy ie_register = receive_block_available;
call write_in_port (interrupt_enable_register, receive_block_available);
call enable_cpu_interrupts;
call write_in_port (command_register, command);

do while (mod(srf, 2) = 0);
    call read_in_port (interrupt_status_register, srf);
end; /* do while */

/* status is available, so read it */
call read_in_port (command_status_register, status_code);
put edit (command_status_received(status_code)) (col(50), a);
call transmit_packet (transmit_data_block);

io while (operation = await_packet);
    /* handler will change */
end;

end perform_loopback;

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

transmit_packet: procedure (packet) external;

DECLAFF

srf fixed bin (7),
reg value fixed bin (7),
write io_port entry (fixed bin (7), fixed bin (7)),
read io_port entry (fixed bin (7), fixed bin (7)),
enable_cpu_interrupts entry,
disable_cpu_interrupts entry,
write_bar entry (pointer),

1 packet,

/* 2 destination_address_a

/* assigned */ 2 destination_address_b
DECLARE

command fixed bin (7),
reg_value fixed bin (7),
srf fixed bin (7),
write_io_port entry (fixed bin (7), fixed bin (7)),
read_io_port entry (fixed bin (7), fixed bin (7)),
command_status_codes entry (fixed bin (7))
returns (char(30) varying);

/* end declarations */

srf = 7;
call write_io_port (command_register, command);
do while (mod(srf, 2) = 0);
   call read_io_port (interrupt_status_reg, srf);
end; /* do while */
call read_io_port (command_status_register, reg_value);
if (command ^= reset) then
do:
   if (command ^= onboard_diagnostic) then
      put edit (command_status_codes(reg_value))
         (col(50), a);
   else
      put edit (diagnostic_codes(reg_value))
         (col(50), a);
   end:
end perform_command:

.perform_loopback: procedure (command);

DECLARE

write_io_port entry (fixed bin (7), fixed bin (7)),
read_io_port entry (fixed bin (7), fixed bin (7)),
initialize_cpu_interruptions entry,
enable_cpu_interruptions entry,
disable_cpu_interruptions entry,
write_bar entry (pointer),
command_status_codes entry (fixed bin (7))
returns (char(30) varying),
command fixed bin (7),
status_code fixed bin (7),
le_reg.value fixed bin (7),
srf fixed bin (7);

/* end declare */
/ \* end main body */
/*****************************/
/ \* procedures  */

fill_data_block: procedure;

DECLARE

i fixed bin (15) static initial (1),
end_of_file bit (1) static init ('0' b);

/ \* begin */
open file (test3010);
on endfile (test3010)
begin:
  end_of_file = '1'b;
end;
do while (end_of_file = 0);
get file test3010 edint (transmit_data_block.data_bytes(i))
  (a(1));
i = i + 1;
end: /* do while */
end:     /* fill_data_block */
/*****************************/
initialize_pic: procedure;

DECLARE

write io_port entry (fixed bin (7), fixed bin(7));
call write io_port (icw1_port_address, icw1);
call write io_port (icw2_port_address, icw2);
call write io_port (icw4_port_address, icw4);
call write io_port (ocw_port_address, ocw1);
end initialize_pic;
/*****************************/
perform_command: procedure (command);

215
/* with a DS register value of 0870h in the link command, this will place packets in extended memory * /
transmit_data_block.destination_address_a = 2;
transmit_data_block.destination_address_b = 7;
transmit_data_block.destination_address_c = 1;
transmit_data_block.destination_address_d = 0;
if (cluster = cluster1) then
do:
  transmit_data_block.destination_address_e = 3;
  transmit_data_block.destination_address_f = -22;
  /* corresponds to 03-FF */
end;
else
do: /* it's cluster 2 */
  transmit_data_block.destination_address_e = 4;
  transmit_data_block.destination_address_f = 10;
  /* corresponds to 04-FF */
end;
transmit_data_block.type_field_a = 0;
transmit_data_block.type_field_b = 0;
do i = 1 to 1500;
  transmit_data_block.data_bytes(i) = ' ';
end;
call read_in_port (command_status_register, reg_value);
call fill_data_block;
call initialize_pic;
call initialize_cpu_interrups;
put skip edit ('Run Onboard Diagnostic') (col(5), a);
call perform_command (onboard_diagnostic);
put skip edit ('Perform Module Interface Loopback')
  (col(5), a);
call perform_loopback (module_interface_loopback);
do i = 1 to 1500;
  receive_data_block.data_bytes (i) = ' ';
end; /* in i */
put skip edit ('Perform Internal Loopback') (col(5), a);
call perform_loopback (internal_loopback);
do i = 1 to 1500;
  receive_data_block.data_bytes (i) = ' ';
end; /* in i */
put skip edit ('Perform External Loopback') (col(5), a);
call perform_loopback (go_online); /* external loopback */
put skip (2);
put edit ((border (1) do i = 1 to 80)) (a);
put skip (2);
call perform_command(reset);
will be \((40 \text{ hex} + 5) \times 4 = 114 \text{ hex}\)

icw4 by '0f'b4,

/* automatic end of interrupt
and buffered mode/master */

ocwl by '9f'b4,

/* unmask interrupt 5 (bit 5) and
interrupt 6. mask all others */

/* end 9259a codes */

cluster1 by 1,
cluster2 by 2,
packet received by 1,
await_packet by 0;

/* include constants specific to the 'NI3010' board */

%include 'ni3010.dcl';

/*****************************/
/* Main Body */

cluster = cluster2;
/* conditional to set up our address for loopbacks */
put list ('z'); /* clear screen */
p put skip;
put edit ('header (1) do i = 1 to 80) (a);
p put skip (2) edit ('NI3010 Diagnostic Routine')
(col(20),a);
p put skip (2);
p put skip edit ('Command Issued', 'Result')
(col:5),a,
(col(50),a);
p put edit ('**************', '********')
(col:5),a,
(col:50),a);
p put skip (2);
unspec(trans_blk_ptr) = '8000'h4;
unspec(rec_blk_ptr) = '8600'h4;
213
copy_command_status_register fixed bin (7),
(i..1) fixed bin (16),
reg_value fixed bin (7),
operation fixed bin (7),
cluster fixed bin (7),
border (80) char (1) static initial ".", 
"trans_blk_ptr, reg_blk_ptr) pointer.

/* Modules external to this module */

write_io_port entry (fixed bin (7), fixed bin (7)),
read_io_port entry (fixed bin (7), fixed bin (7)),
initialize_cpu_interruprs entry,
enable_cpu_interrups entry,
disable_cpu_interrups entry,
write_bar entry (pointer);

/* end module listing */

treplace

/* Codes specific to the Intel 8259a Programmable
   Interrupt Controller (PIC) */

   icw1_port_address by 'c'4'b4,
   icw2_port_address by 'c2'4'b4,
   icw4_port_address by 'c2'4'b4,
   ocw_port_address by 'c2'4'b4,
   use same */
   port addr */

   /* note: icw => initialization
      control
      word

      ocw => operational
      command
      word */

   icw1 by '13'b4,
   icw2 by '40'b4,
   icw2 by '40'b4,
   icw2 by '40'b4,

   /* most significant bits of vectoring
   byte; for an interrupt 5,
   the effective address will be:
   (icw2 + interrupt #) * 4 which

212
transmit_data_block based(trans_blk_ptr),

2 destination_address_a
fixed bin (7),
/* assigned */ 2 destination_address_b
by fixed bin (7),
/* XPROX */ 2 destination_address_c
fixed bin (7),
/* assigned */ 2 destination_address_d
by fixed bin (7),
/* INTFPLAN */ 2 destination_address_e
fixed bin (7),
2 type_field_a
fixed bin (7),
2 type_field_b
fixed bin (7),
2 data_bytes (1500)
char (1),

receive_data_block based (rec_blk_ptr),

2 frame_status bit(5),
2 null_byte fixed bin (7),
2 frame_length_lsb fixed bin (7),
2 frame_length_msb fixed bin (7),
2 destination_address_a fixed bin (7),
2 destination_address_b fixed bin (7),
2 destination_address_c fixed bin (7),
2 destination_address_d fixed bin (7),
2 destination_address_e fixed bin (7),
2 destination_address_f fixed bin (7),
2 source_address_a fixed bin (7),
2 source_address_b fixed bin (7),
2 source_address_c fixed bin (7),
2 source_address_d fixed bin (7),
2 source_address_e fixed bin (7),
2 source_address_f fixed bin (7),
2 type_field_a fixed bin (7),
2 type_field_b fixed bin (7),
2 data_bytes (1500) char (1),
2 crc_msb fixed bin (7),
2 crc_upper_middle_byte fixed bin (7),
2 crc_lower_middle_byte fixed bin (7),
2 crc_lsb fixed bin (7),

test3216 file,
copy_ie_register fixed bin (7).
This is a highly reliable packet switching implementation!
APPENDIX L

NI3010 DIAGNOSTIC CODE

In the event of an Ethernet board failure indication by the NI3010 Driver, the full range of NI3010 operations can be tested with this routine. Any changes to the port addresses of the NI3010 will have to be reflected in the NI3010.DCL file contained in Appendix K. This code will also have to be recompiled and relinked.

This routine is invoked with the CP/M-86 transient command: T3010/Cx, where x is the cluster to be tested. For example, T3010/C1 tests the NI3010 at Cluster 1. This diagnostic routine uses the factory default Ethernet physical address, so the boards should not be swapped between clusters without taking note of its physical address. The NI3010 Driver does not have this restriction.
The file ASMOUT.A86 is linked with the module to allow access to hardware port addresses and to allow a low level assembly language interrupt handler to call a PI/I-m8 interrupt handler. The LINK86 input option files are also included in this appendix.
: restore registers

pop es
pop fs
pop bp
pop di
pop si
pop dx
pop cx
pop bx
pop ax
sti
iret

end
enable_cpu_interrupts:

; Module Interface Specification:
; Caller: Ethertest(PL/I) Procedure
; Parameters: NONE

sti
ret

disable_cpu_interrupts:

; Module Interface Specification:
; Caller: Ethertest(PL/I) Procedure
; Parameters: none

cli
ret

interrupt_handler:

; IP, CS, and flags are already on stack
; save all other registers

push ax
push bx
push cx
push dx
push si
push di
push bp
push ds
push es

call hl_interrupt_handler ; high level source
routine
mov temp.es, es
mov dx, es
mov si, [bx]
mov ax, [si]
mov cl, 12
shr dx, cl
mov temp.es byte, dl
mov dx, temp.es
mov cl, 4
shl dx, cl
add ax, dx
jne no_add
add_1: inc temp.es byte
no_add: out l_bar_port, al
mov al, ah
out h_bar_port, al
mov al, temp.es_byte
out e_bar_port, al
pop si! pop dx! pop es! pop cx! pop ax! pop bx
ret

;-----------------------------

initialize_cpu_interrupts:

; Module Interface Specification:

; Caller: Ethertest(PL/I) Procedure

; Parameters: NONE

initmodule cseg common
org 114h
int5_offset rw 1
int5_segment rw 1

cseg
push bx
push ax
mov bx, offset interrupt_handler
mov ax, 0
push dx
mov dx, ax
mov ds:int5_offset, bx
mov bx, cs
mov ds:int5_segment, bx
pop ds
pop ax
pop bx
sti

206
readio_port:

; Parameter Passing Specification
;
; entry                exit
;
; parameter 1  <port address>        <unchanged>
; parameter 2  <meaningless>        <register value>

Cseg
push bx! push si! push dx! push ax
mov si, [bx]
mov al, [si]
mov port_address, al
mov si, 2[bx]
mov dl, port_address
mov dh, 0
in al, dx
mov [si], al
pop ax! pop dx! pop si! pop bx!
ret

write_bar:

; Parameter Passing Specification
;
; parameter 1 (and only): the address of the data block to be transmitted or received.

dseg

; This module computes a 24 bit address from a 32 bit address - actually it's a combination of the ES register and the IP passed via a parameter list.

push bx! push ax! push cx! push esi push dx! push si

mov dx, 0800h
mov es, dx
**asmout.486 file**

---

extrn hl_interrupt_handler : far

public write_io_port
public read_io_port
public write_bar
public initialize_cpu_interruption
public enable_cpu_interruption
public disable_cpu_interruption

---

write_io_port:

; Parameter Passing Specification:

; entry                           exit
; parameter 1                   parameter 2
; <port address>                <value to be outputted>
; <unchanged>                   <unchanged>

dseg
port_address rb 1
cseg

push bx! push si! push dx! push ax
mov si, [bx]
mov al, [si]
mov port_address, al
mov si, [bx]
mov al, [si]
mov dl, port_address
mov dh, 00h
out dx, al
pop ax! pop dx! pop si! pop bx

ret

---

204
by ----*/
/ * XPROX */ 2 destination_address_c
/ * ----*/ 2 destination_address_d
/ * assigned */ 2 destination_address_e
/ * by */
/ * INTERLAN */ 2 destination_address_f
/ * ----*/
/
/ * begin */
srf = 0;
call write_io_port (interrupt_enable_register, enable_m13012_interrups);
call write_io_port (addr(packet));
call write_io_port (high_byte_count_reg, 5); /* 1508 */
call write_io_port (low_byte_count_reg, -28);
copy_ie_register = transmit_dma_done;
call enable_cpu_interrups;
call write_io_port (interrupt_enable_register, transmit_dma_done);
do while (copy_ie_register = transmit_dma_done);
end; /* loop until the interrupt handler takes care of the TDP interrupt - it sets IE PF to 4 */
call write_io_port (command_register, load_and_send);
do while (mod(srf, 2) = 0);
call read_io_port (interrupt_status_reg, srf);
end; /* do while */
call read_in_port (command_status_register, reg_value);
end transmit_packet;

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

HL_interrupt_handler: procedure external;

/ * This routine is called from the low level 8086 assembly language interrupt routine */

DFCLAFF
write io port entry (fixed in \( \text{fixed in } '7' \)),
read io port entry (fixed in \( \text{fixed in } '7' \)),
enable cpu interrupts entry,
disable cpu interrupts entry,
write_bar entry (pointer),
match bit (1) static init ('1'h);

/* begin */
call disable_cpu_interrups;
call write_io_port(interrupt_enable_register,
                   disable_mio17_interrups);
if (copy_ie_register = receive_block_available)
then do:
call write_bar (addr(receive_data_block));
call write_io_port(high Byte_count_reg, 5):
/* 1522 bytes */
call write_io_port(low Byte_count_reg, -14);
/* initiate receive DMA */
call write_io_port(interrupt_enable_register,
                   receive_dma_done);
copy_ie_register = receive_dma_done;
end; /* do */
else
if (copy_ie_register = receive_dma_done)
then do:
do i = 1 to 1500:
if (transmit_data_block.data_bytes[i] =
    receive_data_block.data_bytes[i])
then
    match = 0;
end; /* iterative do */
if 'match = 2) then
do:
puts(skip(2) edit ('*** Warning ***')
     (col(30),a);
puts(skip edit ('*** Packet Error ***')
     (col'25),a);
end; /* if */
operation = packet_received;
end;
else
if (copy_ie_register = transmit_dma_done)
then do:
call write_io_port(interrupt_enable_register,
                   receive_block_available);
copy_ie_register = receive_block_available;
end; /* if */
end HL_interrupt_handler;

210
command_status_codes: procedure 'command_status';
    external returns (char (3)) varying;

DECLARE

    command_status fixed bin (7);

if command_status = 0 then
    return ('SUCCESS');
else
    if command_status = 1 then
        return ('SUCCESS WITH RETRIES');
    else
        if command_status = 2 then
            return ('ILLEGAL COMMAND');
        else
            if command_status = 3 then
                return ('INAPPROPRIATE COMMAND');
            else
                if command_status = 4 then
                    return ('FAILURE');
                else
                    if command_status = 5 then
                        return ('BUFFER SIZE EXCEEDED');
                    else
                        if command_status = 6 then
                            return ('FRAME TOO SMALL');
                        else
                            if command_status = 8 then
                                return ('EXCESSIVE COLLISIONS');
                            else
                                if command_status = 10 then
                                    return ('BUFFER ALIGNMENT ERROR');
                                else
                                    if command_status = 9 then
                                        return ('STOPPED');
                                    else
                                        return ('STOPPED');
                                    end if;
                                end if;
                            end if;
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end if;
else
    if command_status = 11 then
        return ('STOPPED');
    else
        if command_status = 12 then
            return ('STOPPED');
        else
            if command_status = 13 then
                return ('STOPPED');
            else
                if command_status = 14 then
                    return ('STOPPED');
                else
                    if command_status = 15 then
                        return ('STOPPED');
                    else
                        if command_status = 16 then
                            return ('STOPPED');
                        else
                            if command_status = 17 then
                                return ('STOPPED');
                            else
                                if command_status = 18 then
                                    return ('STOPPED');
                                else
                                    if command_status = 19 then
                                        return ('STOPPED');
                                    else
                                        if command_status = 20 then
                                            return ('STOPPED');
                                        else
                                            if command_status = 21 then
                                                return ('STOPPED');
                                            else
                                                if command_status = 22 then
                                                    return ('STOPPED');
                                                else
                                                    if command_status = 23 then
                                                        return ('STOPPED');
                                                    else
                                                        if command_status = 24 then
                                                            return ('STOPPED');
                                                        else
                                                            if command_status = 25 then
                                                                return ('STOPPED');
                                                            else
                                                                if command_status = 26 then
                                                                    return ('STOPPED');
                                                                else
                                                                    if command_status = 27 then
                                                                        return ('STOPPED');
                                                                    else
                                                                        if command_status = 28 then
                                                                            return ('STOPPED');
                                                                    end if;
                                                                end if;
                                                            end if;
                                                        end if;
                                                    end if;
                                                end if;
                                            end if;
                                        end if;
                                    end if;
                                end if;
                            end if;
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end if;
end command_status_codes;

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

diagnostic_codes: procedure (diag_status)
    external returns (char (30)) varying;

DECLARE

    diag_status fixed bin (7);

if diag_status = 0 then
    return ('SUCCESS');
else

if diag_status = 1 then
    return ('NM10 MICROPROCESSOR MEMORY ERROR');
else
    if diag_status = 2 then
        return ('NM10 DMA ERROR');
    else
        if diag_status = 3 then
            return ('TRANSMITTER ERROR');
        else
            if diag_status = 4 then
                return ('RECEIVER ERROR');
            else
                if diag_status = 5 then
                    return ('LOOPBACK FAILURE');
    end diagnostic_codes:

/*******************----------------------------------------------*/

end; /* procedure boardtest */
LIST OF REFERENCES


### INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Address</th>
</tr>
</thead>
</table>
| 1. | 2      | Defense Technical Information Center  
     |        | Cameron Station  
     |        | Alexandria, Virginia 22341 |
| 2. | 2      | Library, Code 0142  
     |        | Naval Postgraduate School  
     |        | Monterey, California 93943 |
| 3. | 1      | Department Chairman, Code 52  
     |        | Department of Computer Science  
     |        | Naval Postgraduate School  
     |        | Monterey, California 93943 |
| 4. | 1      | Dr. M. L. Cotton, Code 62Cc  
     |        | Department of Electrical Engineering  
     |        | Naval Postgraduate School  
     |        | Monterey, California 93943 |
| 5. | 3      | Dr. Uno P. Kodres, Code 52Kr  
     |        | Department of Computer Science  
     |        | Naval Postgraduate School  
     |        | Monterey, California 93943 |
| 6. | 1      | LT David J. Brewer  
     |        | 7659 Cherrywood Dr.  
     |        | Charleston Heights, South Carolina 29405 |
| 7. | 1      | Daniel Green, Code 20F  
     |        | Naval Surface Weapons Center  
     |        | Dahlgren, Virginia 22449 |
| 8. | 1      | CAPT J. Foreman, USN  
     |        | PMS 49085  
     |        | Naval Sea Systems Command  
<pre><code> |        | Washington, D.C. 20362 |
</code></pre>
<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>PCA AFIS Data Repository</td>
<td>3CA Corporation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Government Systems Division</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mail Stop 127-327</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Moorestown, New Jersey 08057</td>
</tr>
<tr>
<td>10</td>
<td>Library (Code E33-05)</td>
<td>Naval Surface Warfare Center</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dahlgren, Virginia 22449</td>
</tr>
<tr>
<td>11</td>
<td>Dr. M. J. Gralia</td>
<td>Applied Physics Laboratory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Johns Hopkins Road</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Laurel, Maryland 20707</td>
</tr>
<tr>
<td>12</td>
<td>Dana Small</td>
<td>Code 8242, NOSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>San Diego, California 92152</td>
</tr>
<tr>
<td>13</td>
<td>Loris C. Brewer</td>
<td>Route 2, Box 152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fair Play, Missouri 65649</td>
</tr>
<tr>
<td>14</td>
<td>Jon Wilson</td>
<td>Case Rixon Communications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2120 Industrial Parkway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Silver Spring, Maryland 20904</td>
</tr>
<tr>
<td>15</td>
<td>Lubna Fiaz</td>
<td>Systems Analyst</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suburban Bank</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6495 New Hampshire Avenue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hyattsville, Maryland 20783</td>
</tr>
<tr>
<td>16</td>
<td>Kevin Acoveno</td>
<td>Case Rixon Communications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2120 Industrial Parkway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Silver Spring, Maryland 20904</td>
</tr>
<tr>
<td>17</td>
<td>C.F. Hudson</td>
<td>Route 4, Box 224</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stockton, Missouri 65785</td>
</tr>
<tr>
<td>18</td>
<td>Philip D. Sevold</td>
<td>6413 Lomack Court</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Las Vegas, Nevada 89128</td>
</tr>
<tr>
<td>19</td>
<td>Karen Badner</td>
<td>11524 Colt Terrace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Silver Spring, Maryland 20902</td>
</tr>
</tbody>
</table>
20. Warrer Chir-Lee
    Sr. Systems Analyst
    Suburban Bank
    6495 New Hampshire Avenue
    Hyattsville, Maryland 20783

21. Jorge Ferrera
    Systems Programmer
    Suburban Bank
    6495 New Hampshire Avenue
    Hyattsville, Maryland 20783

22. Richard Zierit
    4707 Coachway Drive
    Rockville, Maryland 20852
END

FILMED

5-85

DTIC