AD-A059 601 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A REAL-TIME OPERATING SYSTEM FOR SINGLE BOARD COMPUTER BASED DI--ETC(U)
JUN 78 W NIEMANN

UNCLASSIFIED NL

1 OF 4
AD
A059 601

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A REAL-TIME OPERATING SYSTEM FOR SINGLE BOARD COMPUTER BASED DISTRIBUTED NAVAL TACTICAL DATA SYSTEMS.

Wolfgang Niemann

Master's thesis,

June 1978

347 p.

Thesis Advisor: Professor U. R. Kodres

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Real-time Operating System for Single Board Computer Based Distributed Naval Tactical Data Systems.		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1978 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) LT Wolfgang Niemann		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1978 13. NUMBER OF PAGES 346
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed computer systems, Single Board Computers, Naval Tactical Data Systems, real-time operating system

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The microprocessor revolution has produced a capable computer on a single printed circuit board.

The design and development of a real-time operating system for a distributed system of Single Board Computers is presented in this paper.

There are user manuals and program descriptions for the operating system, a debug module, a CRT module and a line printer module.

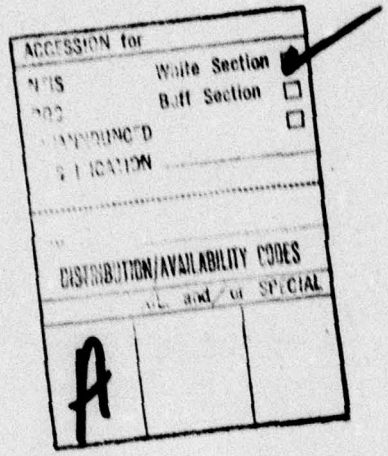
The operating systems has been developed for a Multibus system with three INTEL Single Board Computers SBC80/20-4 and 64K bytes of common memory.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

78 10 06

The system has been designed specifically for Naval Tactical Data Systems applications and the feasibility of such applications are evaluated with respect to currently available Single Board Computers and with respect to Single Board Computers that should be available in the near future.



Approved for public release; distribution unlimited.

A REAL-TIME OPERATING SYSTEM FOR SINGLE BOARD COMPUTER BASED DISTRIBUTED NAVAL TACTICAL DATA SYSTEMS

by

Wolfgang Niemann
Lieutenant, Federal German Navy

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1978

Author: Wolfgang Niemann

Approved by: Uno R. Kodres
Thesis Advisor

J. Bursbrea
Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences

ABSTRACT

The microprocessor revolution has produced a capable computer on a single printed circuit board.

The design and development of a real-time operating system for a distributed system of Single Board Computers is presented in this paper.

There are user manuals and program descriptions for the operating system, a debug module, a CRT module and a line printer module.

The operating system has been developed for a Multibus system with three INTEL Single Board Computers SBC80/20-4 and 64K bytes of common memory.

The system has been designed specifically for Naval Tactical Data Systems applications and the feasibility of such applications are evaluated with respect to currently available Single Board Computers and with respect to Single Board Computers that should be available in the near future.

TABLE OF CONTENTS

I. INTRODUCTION

An examination of currently installed Naval Tactical Data Systems reveals that the heart of the system, the computer, does not represent today's advanced technology. Even in recently implemented systems large 'space, weight, power and maintenance cost consuming' second generation computers are found. The use of second generation technology is caused by lengthy lead times in systems acquisition, system conversion and especially software conversion cost, educational cost etc. However, in the age of microcomputers, which provides features like low hardware cost, high degree of versatility and therefore a potential for standardization, high reliability, low maintenance cost and low power, space and weight consumption, it should be the time to develop new systems in order to make use of these features which seem to be tailored specifically for military applications.

The real-time operating system developed in this thesis should be understood as a step in the direction of making use of the new technology. The operating system is designed for a distributed system of concurrently operating Single Board Computers.

After identifying typical computing requirements for Naval Tactical Data Systems, a distributed computer system using Single Board Computers and a real-time operating system are developed. Three user modules, a debug, CRT, and a line printer module, are introduced. In the conclusion of this paper a comparison between the identified requirements and the developed system is made and possible extensions are indicated.

II. COMPUTING REQUIREMENTS FOR NAVAL TACTICAL DATA SYSTEMS

In this section the basic requirements for a computer system which is to drive a Naval Tactical Data System are considered.

In general it can be said that the computer system has to be able to handle the workload dictated by the operational specifications and to cope with peak situations without a system failure.

Typically, Naval Tactical Data Systems are dedicated systems. Because of this fact, system requirements for the CPU, memory and input/output are known. It is therefore not necessary to carry a vast amount of overhead in order to be prepared for unknown worst cases. However, when deciding on a computer system, future extensions of the system with hardware consequences should be taken into account. Unfortunately it is very difficult to predict the possible enhancements during the life time of a Naval Tactical Data System. Therefore, the chosen computer system should be expandable.

Since Naval Tactical Data Systems are rather complex systems with many different system functions, the equipment used in an implementation is produced by many different manufacturers. Although there are some standard interfaces

for Naval Tactical Data System, the computer, which is to drive the peripheral hardware has to be versatile in order to be connected to different devices with different interfaces.

In order to reduce cost by large series and simplified maintenance a standardization between different systems is highly desirable.

The instruction repertoire of the computer system has to provide instructions which allow the efficient programming of typical operations in Naval Tactical Data Systems. Typical operations are

- the solution of complex mathematical problems in a reasonable time (quasi real-time)
- bit manipulations
- fast data base access
- complex and fast input/output operations
- extensive interrupt handling.

Many command and control operations and decisions depend on the proper functioning of the Naval Tactical Data System. High reliability, even under extreme physical conditions, are therefore a top requirement for this kind of system. In case of a breakdown, the tactical information often is lost or it takes some time to recreate a valid tactical situation. Because of the importance of the Naval Tactical Data System within a larger system (ship or

aircraft), a complete back-up for the computer system is desirable.

The major requirement to be met by the operating system is to run the system under real-time conditions. Dependant on the computer system, this leads to operating systems of differing sizes where the ratio of time used by the operating system to total time should be optimized.

Basically, the operating system has to support an overall program structure which simplifies

- software development
- software maintenance
- software extensions
- use of components from other systems (standardization).

III. A COMPUTER SYSTEM USING SINGLE BOARD COMPUTERS

In this section the hardware concept of a computer system consisting of Single Board Computers is developed. Basic building block of this computer system is a INTEL Single Board Computer, SBC80/20-4.

A. INTEL'S SINGLE BOARD COMPUTER SBC80/20-4

INTEL's Single Board Computer, SBC80/20-4, represents a complete microcomputer on a single printed circuit board.

The SBC80/20-4 includes:

- 8080A CPU
- 4K static Random Access Memory (RAM)
- up to 8K Read Only Memory (ROM)
- 48 programmable parallel input/output lines
- a programmable serial input/output interface
- programmable interval timers
- programmable, eight priority level, vectored interrupt structure
- bus interface for external system bus.

An in-depth description of hardware, function and programming of the SBC80/20-4 is given in the SBC80 manual [INTEL SBC80/20 HARDWARE REFERENCE MANUAL 98-317c].

1. 8080A CPU

The 8080A is a single LSI chip CPU. It has six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or as register pairs, providing 16-bit operations.

A 16-bit stack pointer controls the addressing of an external stack which can be located in general memory.

The 8080A has an address range of 64K bytes.

The CPU Set consists of the 8080A CPU,clock generator and a system controller. It performs all system processing functions and provides a stable timing reference for all other circuitry on the board.

2. Random Access Memory/Read Only Memory

The 4 K Random Access Memory on the Single Board Computer can be jumper assigned to the top address space in any one of the four 16K address blocks.

The Read Only Memory is located starting at address 0000 and has a size of 4K or 8K depending on the type of memory devices used.

The full CPU capability of addressing 64K bytes can be utilized by adding external memory boards which are accessible via the system bus. The respective parts in this memory are 'shadowed' by the on-board memory. The CPU set is capable of determining whether an addressed

memory location is on-board or not.

3. Parallel Input/Output

The SBC provides 48 input/output lines which can be configured by software in combinations of uni-directional or bi-directional input/output ports.

4. Serial Input/Output

The SBC includes a programmable synchronous/asynchronous RS232C communication interface which is capable of operating with all common communication frequencies.

5. Interval Timers

The SBC contains two fully programmable and independent BCD or 16-bit binary interval timers/event counters.

6. Interrupt Structure

The on-board interrupt controller provides vectoring for up to eight interrupt leves in four different priority processing modes. Operating mode and priority assignment are under software control.

7. System Bus Interface

This interface is compatible with INTEL's Multibus system and allows the combination of several Single Board

Computers,memory and other utility boards on one system bus.

Two bus priority systems are available: serial (up to three master controller) and parallel (up to sixteen master controller).

B. SYSTEM CONCEPT

The development of a computer system consisting itself of rather independent Single Board Computers automatically leads to the concept of a distributed system. A distributed system in this context is a computer system in which several processors are working on more or less independent tasks connected by a common system bus. The computer system to be developed is a direct realization of this concept.

Basically the system consists of Single Board Computers and 64K bytes of Random Access Memory external to the Single Board Computers. The external memory resides on four printed circuit boards which are bus compatible with the Single Board Computers. This memory represents a common memory to all Single Board Computers attached to the same bus. This concept in connection with the on-board memory of the Single Board Computers opens up software possibilities which are explored in Chapter IV.

Information interchange between Single Board Computers can be realized using three different concepts.

Concept (1):

Storage of information in common memory and periodic checks of the agreed upon 'mail boxes'. This concept can be used in systems where all processing is organized in a hierarchical 'producer - consumer' structure. In this structure basic processing is local to a Single Board Computer. Data is gathered and processed at a high rate. The output, reduced data and updates of common data bases, is required by others processors at a lower frequency. Since external events can occur asyncronously, an extra data path through common memory for such messages has to be provided.

Concept (2):

Storage of information in common memory and interruption of the information receiving Single Board Computer. The addition of interrupts to concept (1) allows both synchronous and asynchronous transfers of data sets between Single Board Computers. The disadvantage of this concept is the number of interrupt lines required with an increasing number of participating Single Board Computers in order to address each other. The alternative of having only one interrupt common to all Single Board Computers would cause an interruption of all processors and requires a polling scheme to determine the receiving Single

Board Computer.

Concept (3):

Direct transfer of information between Single Board Computers using input/output ports and interrupts. This concept can be used to exchange time critical information between processors. Since it directly involves the CPU of both the sender and receiver the amount of data passed has to be kept small. However, large data sets can be transferred with the use of pointers to common memory. The limitations of concept (2), extended for data lines between input/output ports, also apply for concept (3).

The input/output structure of the Single Board Computers allow the connection of a variety of peripheral devices to the computer system. Each connection represents a hardware modification or specialization of a Single Board Computer, i.e. the dedication of a part of the computer system to a special task.

C. SYSTEM BUS ORGANIZATION

The system bus, which is the main communication line between several Single Board Computers, common memory and other utility printed circuit boards, is implemented using INTEL's Multibus. This bus system allows master-master and master-slave relationships between various system hardware modules.

Transfers via this bus proceed asynchronously, i.e. the transfer speed is dependent on the speed of the transmitting and receiving devices. Once a module has gained control of the bus, transfers can proceed with a maximum rate of 5 million bytes/second.

Master modules can gain access to the bus using a serial or parallel priority resolving scheme. The serial scheme is limited to three masters because of the signal propagation delay, but up to sixteen masters may be connected to the bus using the parallel priority scheme.

The bus system provides an override facility which allows a hardware module to keep control of the bus until the operation is completed. This feature can be used under software control.

D. MEMORY ORGANIZATION

The total memory of the computer system consists of

- common Random Access Memory
- on-board Random Access Memory
- on-board Read Only Memory.

The on-board Random Access Memory portions are located in the same region on all Single Board Computers. This leaves the respective portion in common memory unused, however, it is now possible to run identical programs on the Single Board Computers. Identical programs allow the access

of common data and execution of shared code. The location of on-board Random Access Memory can be changed because all programs are relocatable.

All variable data has to be kept in on-board memory for two reasons:

1) Faster access by CPU and increased overall performance because use of the system bus is avoided.
   On-board memory access does not require WAIT states of the CPU.

2) No data conflicts in the execution of shared code. Although the variable data has the same logical address space, its physical representation is local to the Single Board Computers and therefore 'invisible' to other CPUs.

On-board Read Only Memory contains the executive and frequently executed program parts. Other program parts, especially when they are identical in all Single Board Computers are located in common memory in order to allow shared execution.

E. INPUT/OUTPUT FACILITIES

Each Single Board Computer provides serial and parallel input/output facilities which can be completely driven by interrupts.

The serial interface is ready to be used with a CRT. With the addition of an adapter a TTY can also be connected to a Single Board Computer.

Each Single Board Computer provides six bi-directional 8-bit ports.These ports can be configured by software to be 8-bit data input/output ports or 4-bit bit addressable input/output ports. The latter can be used to receive and send status bit information in conjunction with the 8-bit input/output ports.

The hardware provides three basic modes of operation that can be selected by software:

- Mode 0 : Basic Input/Output
- Mode 1 : Strobed Input/Output
- Mode 2 : Bi-directional Bus

Mode 0 can be used for simple, status driven device interfaces. Since this kind of interface is not compatible with the real-time requirements it is not considered here.

Mode 1 and Mode 2 generally require the support of interrupts. Mode 1 provides a means for transferring data to or from a specific port in conjunction with strobe or

'hand-shake' signals. This mode can be used for a variety of different peripheral devices.

Mode 2 provides communication with a peripheral device on a 8-bit bus for both receiving and transmitting data. 'Hand-shake' are provided to maintain proper bus flow in a similar manner to Mode 1.

The driver/termination connections of the parallel input/output section are left open and have to be inserted depending on the actual implementation.

The primary user considerations in determining how to use each of the six input/output ports are:

- choice of operating mode
- direction of data flow
- choice of driver/terminator networks
- jumper configurations
- mutual port restrictions.

F. INTERRUPT STRUCTURE

The interrupt controller accepts jumper selectable interrupt requests from

- parallel input/output
- serial input/output
- interval timers
- system bus

- directly from external devices.

The controller resolves priority among the eight possible interrupt levels according to an algorithm which is selectable by software. The priority assignments and algorithms can be changed dynamically at any time during system operation.

Of the four possible interrupt modes only the 'fully nested' mode is considered here. In this mode priorities are fixed such that level 0 has the highest priority and level 7 the lowest.

The interrupt hardware provides vectored interrupts where the interrupt vector is not fixed in its location and size (4 or 8 bytes/interrupt). The interrupt controller has to be programmed with location and step width of the interrupt vector.

The interrupt controller allows the setting of a one byte interrupt mask by software. This feature allows the inhibition of not wanted interrupts in any combination of the eight interrupt levels.

IV. A REAL-TIME OPERATING SYSTEM FOR SINGLE BOARD COMPUTERS

The objectives for the operating system to be developed in this section are:

1) The operating system has to be able to control NIDS applications under real-time conditions.

2) The host computer system is the multi processor microcomputer system described in the previous section.

3) The existing Microcomputer Development System and the associated support software (ISIS-II,PL/M-80) has to be used for program development.

4) The operating system must provide debugging tools that allow system debugging and system testing under real-time conditions.

This chapter describes the operating system in more general terms. An in-depth description is given in the user's manual (Appendix A) and in the program description of the operating system (Appendix B).

A TASK is a part of the program which handles a specific function of the system and consists of one or more procedures.

A MODULE is a separate task or separate group of related tasks which are considered to be independent in terms of software development and system integration.

The EXECUTIVE is the kernel of the operating system and controls the scheduling and execution of tasks.

The OPERATING SYSTEM consists of executive, system calls and system data.

A. SYSTEM CONCEPT

In contrast to an operating system for general usage in which the nature of the running tasks is not predictable this real-time operating system drives a dedicated system. Dedicated, in this context, means that the implemented tasks do not change at run time of the system. This concept allows dropping much of the book-keeping overhead which is typical for operating systems for general usage. Tasks in this system are not physically moved in memory, they are only activated and suspended.

Execution of tasks is under control of the kernel of the operating system, the executive. The executive controls the CPU assignment to the various tasks activated by interrupts, to process a message from another task or at a preset point in time.

B. MODULAR STRUCTURE

The operating system supports a strictly modular program structure. Modules are integral parts of the overall program which usually handle a specific function or a group of related functions. This organization not only eases program development and maintenace, it also allows easy-to-implement changes of the system. Typically, a module is compiled separately and then integrated into the rest of the system. Because of the independent nature of modules they can be removed or replaced without influencing the remaining system. Not only user programs represent modules, the operating system itself consists of independent modules.

Modules are identified by a number. The number depends on the number of the Single Board Computer which hosts the module. Maximum number of modules per Single Board Computer is 8. Modules in Single Board Computer 1 have numbers 0 to 7, in Single Board Computer 2 numbers 8 to 15 etc. The lowest module number in a Single Board Computer is reserved for the executive of the operating system while the highest number represents the debug module.

C. FACILITIES OF THE OPERATING SYSTEM

1. Priority Tasks

Priority tasks represent the highest priority level a task can have in the system. Priority tasks are executed as soon as processor control is returned from the current

active task.

A priority task has to be entered into the list of priority tasks with a system call. A single execution of that priority task can then be scheduled with another system call.

The primary purpose of priority calls is the process of interrupts. The call of the priority task has to be scheduled in the interrupt service routine. The high priority of that task ensures that it is executed as soon as the processor becomes available.

2. Communication between Modules

Since modules are separate and independent parts of the system the operating system has to provide a function which allows the tasks or modules to communicate with each other. This communication uses the form of messages which are sent from one module to another. A message is sent with a system call and entered into a FIFO list. The receiving module's message entry is called if no priority task is pending and the message is to be processed next in the FIFO list.

A message consists of message control block and possibly data bytes. The message control block identifies receiving module, sending module, message number and length of the message. The message number depends entirely on an agreement between sending and receiving module and serves

the purpose of identifying the message. If the message control block itself is not sufficient for the transmission of information, data bytes can be added to the message.

A message is always sent from one module to another module regardless in which Single Board Computer they reside. The operating system decodes the number of the receiving module and routes the message to another Single Board Computer if necessary.

3. Time Dependent and Periodic Tasks

Real-time environments and especially Naval Tactical Data Systems require the execution of certain tasks at a predefined point in time or periodically with a specified time interval. In this operating system these tasks range in the priority hierarchy below the priority tasks and the process of messages.

Periodic tasks have to be identified to the operating system with a system call. With this system call the task and the specified time interval is entered into the list of periodic tasks. The executive uses the system's real-time clock to determine the exact time of the call. A periodic task can be suspended or the specified time interval can be changed with system calls.

4. Background Tasks

Background tasks represent the lowest priority level in the system. They are executed only if no priority task is pending, no message is to be processed and no periodic call is necessary, i.e. the processor is idling. This idle time can be used to perform hardware checks on a time slice basis or to perform data reductions for statistic and test purposes.

If a Single Board Computer is completely interrupt driven, (i.e. not periodically activated) the processor can enter the HALT state to free the system bus. The next interrupt will 'awake' the processor and the executive.

5. System Calls

The operating system provides system calls for task management, communication between modules and functions which are commonly used by several modules.

6. Real-time Clock and Count Down Clock

The operating system provides two different clocks, a continuously running real-time clock and a count down clock which can be started with a specified run time. There is only one real-time clock in common memory which is used by all Single Board Computers in the system. The real-time clock is driven by the Single Board Computer with the number 1. Both real-time clock and count down clock make use of

the interval timers on the Single Board Computers.

A count down clock is implemented in each Single Board Computer. The count down clock can be started at any time and generates an interrupt when the specified time is elapsed. The count down clock can be used to control time critical processes. It has a size of 16 bits where the least significant bit represents 1.86 micro seconds.

The real-time clock has a size of 4 bytes, the least significant bit has the value of 1 milli second and the maximum run time is approximately 50 days.

D. REAL-TIME EXECUTIVE

The real-time executive represents the kernel of the operating system. The executive continuously checks for pending tasks on four priority levels:

1. priority tasks
2. messages to be processed
3. periodic tasks
4. background tasks.

The processor is assigned to the next task with highest priority in this scheme. The executive only proceeds to the next lower level if no task is pending at the current or a higher level.

Each Single Board Computer runs its own, identical executive. An executive is tailored to its environment with special compile parameters.

The executives in different Single Board Computers communicate with each other using normal messages via an exchange in an absolutely located buffer in common memory.

Because the code of the executive is executed most often it is located in on-board memory.

E. INTERRUPT HANDLING

All interrupts are handled entirely by the operating system. The operating system initializes the interrupt controller and sets up the interrupt vector.

There are four special interrupts which are handled by the operating system: real-time clock interrupt, count down clock interrupt, system restart interrupt and an interrupt which causes the system to enter the monitor, if implemented.

User modules can activate interrupts with a system call and passing the requested interrupt level and the index of a priority task to process the interrupt. In case of an interrupt, a call of the priority task is scheduled by the operating system. The de-activation of an interrupt is also performed with a system call.

F. SYSTEM MONITOR

The system monitor is implemented as a system call. It is called when internal program limits are exceeded. An address value and two byte values are passed with the system call. The address can point to the location of the error and the two byte values can further specify the nature of the error.

The system monitor generates the display of an appropriate message for the operator.

This feature is primarily designed as an aide in the program development and test phase and to cover undefined program states.

G. SYSTEM INTEGRATION

User modules to run under the operating system are independent with respect to other user modules and the operating system. In order to provide the necessary links to the operating system, a user module needs to be compiled together with some system data and external declarations of the system calls. Furthermore the module's entry for the process of messages has to be identified to the operating system.

The linking of a module to the operating system is implemented using the public/external declaration feature of PL/M-80 and the LINK program in ISIS-II. Basically, a

system integration is the execution of this LINK program. Included in the linking process are the operating system itself and the user modules of the special configuration to be integrated.

Depending on the application, the linked and located code can be kept externally and loaded into Random Access Memory or transferred into Erasable and Programmable Read Only Memory.

V. AVAILABLE USER MODULES

Three user modules have been developed and are available to run in the presented system configuration: a CRT module, a line printer module and a debug module. The software documentation, a program description of each module and a user's manual for the debug functions, is part of the Appendix.

A. CRT MODULE

This module is a typical user module. It drives a CRT connected to a Single Board Computer. The CRT, via the CRT module, may be used by any other module in the system, regardless in which Single Board Computer it is located. Messages are used for the communication between the CRT module and a 'CRT user' module.

The CRT module provides two different kinds of outputs to the CRT and the facility of obtaining inputs from the connected keyboard.

B. LINE PRINTER MODULE

The line printer module is the driver for a line printer. Any module in the system can transfer text with a message to the line printer module in order to have it printed.

C. DEBUG MODULE

This module allows the debugging of the entire system under real-time conditions, i.e. without influencing system operation during the debugging process. The provided debug functions are designed to ease debugging of malfunctions which are typically encountered in real-time systems and specifically in Naval Tactical Data Systems.

The debugging concept allows several users to debug different program parts at the same time.Any Single Board Computer can be debugged from any other Single Board Computer with the restriction that only one user is allowed to debug a Single Board Computer at a time.

Input/output media for the debug module are a CRT or equivalent device and a high speed printer for output only. Both devices are driven by modules described in previous sections.

VI. CONCLUSIONS

In this section a comparison is made between the proposed computer system (hardware concept and operating system) and the requirements established in Chapter II.

It is emphasized that no attempt is made to examine the capabilities of the previously described computer system to drive a typical Naval Tactical Data System. Obviously the CPU, INTEL's 8080A, fails to qualify for this task because of its restricted instruction repertoire (especially the lack of arithmetic instructions), eight bit word size and 64K byte address space.

With the recent introduction of a single chip 16-bit microprocessor with

- instructions to operate on 8-,16- and 32 bit quantities
- signed and unsigned arithmetic operations including multiplications and divisions
- address space of 1 megabyte

the proposed model with minor adaptive changes in the operating system becomes realistic, provided that the concept of Single Board Computers will be kept. Considering the success of INTEL's SBC80 series, this is very likely.

A. HARDWARE DESIGN

1. Standardization and Cost

The proposed hardware design, distributed system with Single Board Computers on a common bus system, is very flexible. It can be used in Naval Tactical Data System applications which require extensive computing power as well as in very small and simple systems. Because of this flexibility, the proposed concept would reduce the number of different computer architectures currently in use. This reduction is equivalent to an increase in standardization which besides lower hardware cost has a strong influence on cost in terms of software development, maintenance and training of personnel.

2. Expandability

The proposed concept has special advantages as far as future changes or extensions of the system are concerned. A hardware change is kept local to one or a few Single Board Computers. Extensions are accomplished easily by adding Single Board Computers to the system. However, it should be noted that the utilization of the system bus may turn out to be the 'bottleneck' of such a system. The capacity of the system bus clearly dictates the limits for the proposed computer design.

A solution to this problem is thinkable in form of a 'super bus' structure consisting of two or more of the

previously developed systems and a connecting 'super' bus system. This structure would lead to a strictly hierarchical system concept in which information is reduced before being transferred to the next higher bus level.

3. Interfaces with Peripheral Equipment

The implemented input/output interfaces in Single Board Computers are not fixed. The input/output and interrupt controller are software programmable and, in connection with easy to implement jumper connections, allow the tailoring of the input/output and interrupt facilities according to the application.

4. Maintenance and Reliability

The most astonishing effects of the new technology used in Single Board Computers can be found in the area of maintenance and reliability. A computer system consisting of Single Board Computers is practically maintenance-free. Although reliability tests of the new technology are still in progress, it is already known that there is a great increase in reliability. In case of a failure parts of the computer would no longer be replaced: the computer would be replaced itself.

5. Physical Requirements

The proposed computer system drastically reduces weight, space and environmental (power, cooling etc.)

requirements of currently implemented Naval Tactical Data Systems. This reduction is especially important for airborne systems.

6. System Redundancy

Up to now a complete back-up computer system is not found in Naval Tactical Data System applications. Reasons for this are mainly costs and sometimes space and weight. With an implementation of the proposed system concept these constraints could be eliminated. Although reliability is already greatly increased a redundant system design can be considered.

B. OPERATING SYSTEM

1. Naval Tactical Data System Requirements

The proposed operating system has been designed specifically for an application in Naval Tactical Data Systems. It provides facilities which ensure the real-time operation of the entire system. 'Real-time' is a relative expression with respect to Naval Tactical Data Systems. An operator expects a system reaction in real-time after completion of his inputs. In this case the system has to provide an 'instantenous' reaction in terms of human speed. However, in the case of a high frequency radar interface, 'instantenous' represents a considerably shorter time between input and reaction. The structure of the operating

system with different priority levels supports this interpretation of 'real-time' as well as other commonly found Naval Tactical Data System operations.

2. Software Development,Maintenance and Extensions

Because of the modular structure supported by the operating system, it is possible to run test vehicles early in the program development phase. The modules in the test vehicle are replaced by the original modules as soon as they are completed or the simulated equipment is installed. This concept of a continuous test of the growing system involves some simulation overhead but it avoids 'big bang' tests and leads to a better tested system.

The modular structure also eases program maintenance, because modules are easily changed and re-integrated into the system.

Extensions to the system are implemented by adding modules. Hardware facilities of the computer system can be extended by increasing the number of Single Board Computers. In order to find an optimal distribution of the extended program it may be necessary to re-distribute the modules over the Single Board Computers.

3. Standardization

The structure of the operating system basically reflects a similar structure used in various real-time Naval

Tactical Data Systems. This allows the use of already developed software and knowledge.

4. Simulation

Simulation in Naval Tactical Data Systems is needed mainly for three reasons: software development, software test and operator training. The modular software structure in connection with the distributed Single Board Computer concept allow simulation not only of missing modules or equipment, it allows the simulation of the operation of entire Single Board Computers as well. No hardware changes are necessary since the simulation takes place entirely inside the computer system.

APPENDIX A

PROGRAM DESCRIPTION OF OPERATING SYSTEM

TABLE OF CONTENTS

1 General

1.1 Introduction

This segment of text describes the function of the operating system. The use of the facilities is documented in the user's manual for the operating system.

In cases where system functions are well explained in the program listing itself no description is given here.

The executive is considered to be a module. It has the lowest possible module number in a SBC and the module identification EX.

1.2 Abbreviations and Conventions

All numbers in this segment of text are decimal except as otherwise indicated.

Task - part of the program which handles a specific function of the system and consists of one or more procedures

Module - part of the program which consists of one or more (related) tasks and can be compiled separately

Executive - part of the operating system which controls the scheduling and execution of tasks

Operating System - consists of executive, system calls and system data

System - consists of operating system and all integrated user modules

SBC - Single Board Computer

MDS - Microcomputer Development System (INTEL)

MCB - Message Control Block
RMN - Receiving Module Number
SMN - Sending Module Number
MN - Message Number
ML - Message Length

EX - executive, module number in SBC 1/2/3 : 00/08/16
LP - line printer module, module number in SBC 1/2/3 : 05/13/21
CS - CRT module, module number in SBC 1/2/3 : 06/14/22
DB - debug module, module number in SBC 1/2/3 : 07/15/23

RTC - Real Time Clock
CDC - Count Down Clock

2 Executive

The executive is the kernel of the operating system. It controls the process of

- priority tasks
- real-time messages
- time dependent (periodic) tasks
- background tasks.

2.1 Inputs

EX receives two real-time messages from any debug module,'start message extraction' and 'stop message extraction'.

Format:

```
RMN  :  EX
SMN  :  any debug module
MN   :  10 - start
        11 - stop
ML   :  04
```

This message controls the state of the flag MSGEXTRACTION. It is set to 'TRUE' upon receipt of the 'start message extraction' message and reset to 'FALSE' when 'stop message extraction' is received.

2.2 Function

2.2.1 System Initialization

The system is initialized with a call of procedure EXSTART at system start. Prior to this call the variables SAVESTACKPTR and RESTART are set.

SAVESTACKPTR contains the value of the stack pointer at initial system start. It is saved for a system restart without loading and system RESET.

RESTART is set to 'FALSE' at the initial start of the system. In case a system restart is initiated with INT6, RESTART is set to 'TRUE'. At the same time the stack pointer is reset to the saved value.

RESTART is used by all modules to determine whether the current start is a system start with or without hardware reset.

In order to prevent overlapping program action caused by erroneous interrupts, the interrupts are locked out for the time of system initialization.

All system tables are reset and a 'start' message for each module in the same SBC is packed into the system's message buffer.

EXSTART ends with the initialization of the interrupt controller and an ENABLE instruction.

SBC1 has additional tasks at system initialization. It resets the RTC, starts the RTC update and gives the start signal to all other SBCs in the system.
All other modules in the system check their specific start variable START1, START2 etc. to take on a value other than 0.
After completion of the initialization, SBC1 sets the respective SBC number into START1, START2 etc. and all SBCs start their initialization.

2.2.2 Priority Calls

In the context of priority calls there are two relevant items of system data: PRIORLIST and PRIORSCHEDULE.

PRIORLIST is an address vector of length 8 and contains the addresses of priority procedures entered.

PRIORSCHEDULE is a byte variable which indicates the scheduled priority calls,e.g. bit 0 = 1 means that a priority call of the priority procedure in PRIORLIST(0) has been scheduled.
Prior to the call of this procedure the respective bit in PRIORSCHEDULE is reset to 0.

The executive keeps checking PRIORSCHEDULE until all calls have been made before proceeding to the process of real-time messages.

2.2.3 Communication Between Modules

Communication between modules uses real-time messages.
These messages can be sent from any module to any other module in the system.
A message is considered to be 'internal' if it is sent to a module in the same SBC. An 'external' message is sent to a module in an other SBC.
Internal and external messages have the same format,only the treatment by the executive is different.

A message is sent with a call of SEND. In this system procedure the message is placed into MSGBUFFER, a circular FIFO list.

MSGBUFFER is controlled by two pointers: MSGIN (next to fill) and MSGOUT (next to process).
The variable NUMMSG contains the number of messages currently in MSGBUFFER.

2.2.3.1 Internal

The executive checks NUMMSG for a message to be processed.
If NUMMSG = 0 then the executive proceeds to check for external messages.
MSGOUT points to the next message to be processed.
The executive computes the index of the next message in MSGBUFFER (new MSGOUT = MSGOUT + ML of current message).
After this update, the executive examines RMN of the message.
If the receiving module is in the same SBC the procedure MSGENTxx is called (xx = relative module number in a SBC : 0 - 7).
This procedure is the message entry of the receiving module.

2.2.3.2 External

If the receiving module of a message is not in the own SBC, the executive calls SENDEXT to process this message.

There is a buffer for external messages (EXTMSGBUFFER) which has a very similar structure to MSGBUFFER.
The only exception is that the receiver of the message is the number of the SBC which hosts the receiving module.

An external message is kept into EXTMSGBUFFER until it is processed by the respective SBC and transferred into the local message buffer.

Since all SBCs operate in EXTMSGBUFFER there is a lock mechanism that prevents two SBCs from working in EXTMSGBUFFER at the same time.

Every time a message is taken out of EXTMSGBUFFER or when the first message is written into the empty EXTMSGBUFFER the variable EXTMSG is set to the number of the receiving SBC of the message currently at the top of EXTMSGBUFFER.
If EXTMSG = 0 then no external message is waiting to be processed.

After checking the external messages PRIORSCHEDULE is examined again.

2.2.4 Periodic Calls

All activated periodic calls are kept in PERLIST.
PERLIST is a vector of records.
The variable NUMPER contains the number of activated periodic calls.

PERXTBL, a list of pointers to PERLIST, is always kept compact, i.e. if a periodic call is suspended, entries in PERXTBL are moved to become compact again. This technique reduces execution time when the executive is checking for necessary periodic calls.

If the executive finds a 'next call time' <= RTC then a new 'next call time' is computed (RTC + time interval) and the periodic procedure is called.

If no periodic procedure is to be called, the executive proceeds to the background tasks.
Otherwise the periodic procedure is called and upon return of program control the priority calls are checked again.

2.2.5 Background Tasks

Background tasks represent the lowest priority level within the executive.
They are executed only if no other task is pending, i.e. the processor is idling.

2.2.6 Message Extraction

Before processing a real-time message, the executive calls EXMSGEXTR if the flag MSGEXTRACTION is 'TRUE'.
In this procedure the current message is checked against the message control block contained in DEBUGMCB.
If DEBUGMCB matches the current message, the message entry of the debug module is called with a faked 'message extraction' message.
Upon return the current message is processed.

2.3 Outputs

At system start, after its own initialization, EX sends 'start' messages to all modules in the same SBC.

Format:

RMN - all modules in own SBC
SMN - EX
MN - 00
ML - 04

Apart from the 'start' message, EX sends an 'extraction' message to DB if message extraction is activated and a matching message was detected. This message is 'sent' by directly calling the message entry of DB.
This special procedure is chosen in order to avoid an excessive load of the system's message buffer since each extracted message would be represented twice: as original message and as data bytes of the 'extraction' message.

3 Interrupt Handling

All interrupts are handled entirely by the operating system.

There are four system interrupts and three user interrupts. The system interrupts are:

- RTC interrupt
- CDC interrupt
- system restart
- enter monitor.

The RTC interrupt (INT2) is activated in SBC 1 only.
This interrupt is generated by counter 0 of the interval timer arriving at the terminal count of 0.
Counter 0 is first set in procedure EXSTART to the equivalent of 1 msec and started.
Upon occurrence of the interrupt the RTC is updated and the counter is started again with a time interval of 1 msec.

A CDC interrupt may be initiated in each SBC. The CDC interrupt is started with the system call SETCDC.
At terminal count the interrupt (INT0) is generated and the address passed with the system call is called.

The system can be restarted without RESET by generating INT6 at the front panel of the MDS. From the interrupt process the procedure SYSRESTART is called where the restart is initiated.

The SBC monitor can be entered at any time by pressing INT2 at the front panel.
This interrupt is jumpered on the SBCs to cause an interrupt on level 1.
Upon occurrence, the monitor is entered at location 0740H.
The same procedure when activating the monitor with RESET applies, i.e. typing capital 'U' to initialize the USART.

Note: Since the monitor changes locations in on-board Random Access Memory, the system cannot be restarted without loading!

User interrupts can be activated on levels 3,4 and 5. They are activated and de-activated with system calls (ENTERINT and REMINT).

The interrupt vector is located at 3000H and has a length of 64 bytes, i.e. each interrupt occupies 8 bytes.
This structure is compatible with PL/M-80.

The interrupt routines are written in PL/M-80 and therefore located at 0000 - 003FH.
After SBC 1 is loaded, the loader transfers the code for interrupt processing to its final location in the interrupt vector.
Since the user interrupts are restricted by PL/M-80 to INT3 - INT7, only 5 interrupts can be programmed this way.
These are the three user interrupts and RTC and CDC interrupt.
The code for the process of monitor and restart interrupt is written into the interrupt vector in procedure EXSTART.

The book-keeping of user activated interrupts takes place in table INTTBL.
INTTBL(i) contains FFH if interrupt i is not activated.
An activated interrupt is indicated by INTTBL(j) = k, where j is the interrupt level and k is the index of the priority task to be scheduled upon occurrence of the interrupt.

4 System Data

System data are divided in two parts:

- data to be compiled with each module
- data to be compiled with the executive, system calls and the debug module.

The first data set is in the source files SYDATP.SRC and SYDATE.SRC.
The executive has to be compiled with the PUBLIC declarations of this data in SYDATP.SRC whereas all other components of the system are compiled with the EXTERNAL declarations in SYDATE.SRC.

Since this data set is well explained in the program listing (see Section 11 in the operating system user's manual) it is not described here.

The second data set is in the source files EXDATP.SRC and EXDATE.SRC.
It contains all data necessary for the operation of the executive and the system calls.
The executive has to be compiled with the PUBLIC declarations in EXDATP.SRC. All other components which need to operate on these data (system calls, interrupt handling, debug module) can be compiled with the EXTERNAL declarations in EXDATE.SRC.

All operational data of the system are listed and described in Section 9.

5 System Calls

The code of the system calls has been split up into seven parts in order to ease program development and maintenance under ISIS-II. These program parts are named SCPUB1 - SCPUB7.

The object code of the system calls is kept in the object library SC.LIB.

Each module can be compiled with the set of EXTERNAL declarations of all system calls in the source file SCEXT.SRC.
The matching of the PUBLIC and EXTERNAL declarations takes place in the LINK step during system generation where the object library SC.LIB has to be included.

The function of the system calls is explained in the source program listing.

6 Real-time Clock and Count Down Clock

RTC and CDC are implemented using counter 0 and 1 of the on-board interval timer.

Both counters are 'down counters' with a terminal count of 0 and driven by a clock input of 1.86 micro seconds.

The 'terminal count' output line is jumpered to the interrupt controller.
Counter 0 generates a level 2 interrupt while counter 1 is tied to INT0.

Counter 0 (RTC) is driven by SBC 1 only. SBC 1 loads counter 0 with the equivalent of 1 msec (LSB = 1.86 micro seconds) and updates the RTC (4 byte vector in common memory) by 1 upon occurrence of the interrupt, i.e.terminal count of counter 0.

A CDC interrupt is implemented in each SBC. Its initialization is by a system call (SETCDC).
In the process of this system call the following actions take place:

- save the address to be called at CDC interrupt
- enable interrupt level 0
- load counter 1 with the transferred value (LSB = 1.86 micro seconds).

Upon occurrence of the CDC interrupt, the interrupt on level 0 is disabled and the indicated address is called.

7 Loader

The system is loaded and started under control of a separate loader.

The loader runs in the MDS and is started together with the SBCs. It interacts with the SBCs through four absolutely located variables:

- LOADSBC (F1F0H)
- START1 (F1F1H)
- START2 (F1F2H)
- START3 (F1F3H).

At start all four variables are reset to 0. The SBCs continuously check LOADSBC to take on the value of their SBC number (1 - 3).
The loader loads the code for SBC1 from the disk with an offset(bias) of 6000H.An ISIS-II system call is used to load the file LOAD1.
After completion of the loading LOADSBC is set to 1.The loader now waits until LOADSBC becomes 0 again.

SBC1 detects the '1' in LOADSBC and starts moving the code from the temporary storage into on-board memory for which it was located before by the LOCATE program of ISIS-II.
After completion of the move, LOADSBC is set to 0 and then the SBC waits for the variable START1 to become 1.

The loader now repeats the process for SBC2 and SBC3.

After having loaded all SBCs, the loader stores a 1 in START1. This is the signal for SBC1 to start.
After initializing system data and the RIC the variables START2 and START3 are set to 2 and 3 respectively.
This effects the start of the entire system.

The loader issues informative messages on the CRT as it proceeds through the loading process.

It should be noted that a loading process is not necessary in a practical application because the program would reside in Read Only Memory.

8 Memory Map

In this section all absolutely located data and code is described. Furthermore all PROM changes of the SBC monitor are listed.

8.1 Absolute Data

Absolutely located data is necessary for communication between SBCs and loading and starting of the SBCs.

Absolute system data are located in the region F000H - F1EFH. These data include:

- module status table (MODSTATUS)
- real-time clock (RTC)
- message buffer and message control variables for external message exchange.

```
F000
 ..     MODSTATUS
F017
F018
 ..     RTC
F018
F01C    EXTMSGLOCK
F01D    EXTMSG
F01E    EXTMSGIN
F01F    EXTMSGOUT
F020    NUMEXTMSG
F021    EXTLASTMSG
F022
 ..     EXTMSGBUFFER
F119
```

Absolute data for loading and starting are located in the region F1F0 - F1F7:

```
F1F0    LOADSBC
F1F1    START1
F1F2    START2
F1F3    START3
F1F4
 ..     key for the system operation (key is 0ABCDH)
F1F5
```

Since the snapshot function is implemented with the trap instruction RST4, it is necessary to define the entrance of the snapshot execution in an absolute location. The entry address is stored in 3040H and 3041H.

8.2 Absolute Code

Apart from the code for the SBC monitor,which is located absolutely because it resides in ROM,the interrupt vector has to be located absolutely.

The interrupt controller is programmed to expect the interrupt vector at 3000H with 8 bytes/interrupt.
The code for INT0,INT2,INT3,INT4 and INT5 is moved into the vector by the loader. The rest of the vector is set in the start routine (EXSTART). These are INT1 (entry of monitor) and INT6 (system restart).
INT7 currently is not implemented.

```
3000
 ..      CDC interrupt
3007
3008
 ..      'enter SBC monitor' interrupt
300F
3010
 ..      RTC interrupt
3017
3018
 ..      INT3 (user interrupt)
301F
3020
 ..      INT4 (user interrupt)
3027
3028
 ..      INT5 (user interrupt)
302F
3030
 ..      'system restart' interrupt
3037
3038
 ..      INT7 (not implemented)
303F
```

8.3 Changes of the SBC Monitor

This section lists and comments the changes of the SBC monitor in PROM.

The monitor needs to be modified in order to allow the automatic loading of the system.
At system start the monitor checks a key which is an address value located in F1F4H.
For operation of operating system the user has to store the the hexadecimal value ABCD in this location. If the contents is different from this value the monitor will operate in normal mode.

```
0000    JMP 0700            C3 00 07    jump to 0700H at RESET

0700    LXI H,KEY           21 F4 F1    check key
0703    MVI A,0ABH          3E AB
0705    CMP M               BE
0706    JNZ 0740            C2 40 07    start monitor
0709    INX H               23
070A    MVI A,0CDH          3E CD
070C    CMP M               BE
070D    JNZ 0740            C2 40 07    start monitor
0710    DI                  F3          disable interrupts
0711    LXI H,LOADSBC       21 F0 F1    wait until ready to load
0714    MVI A,SBC           3E nn       SBC nn
0716    CMP M               BE
0717    JNZ 0716            C2 16 07    not ready yet
071A    LXI B,9000          01 00 90    begin of temporary code
071D    LXI D,3042          11 42 30    begin on-board RAM
0720    LDAX B              0A
0721    STAX B              12          move code into own RAM
0722    INX B               03
0723    INX D               13
0724    MVI A,0EFH          3E EF       last byte to be moved
0726    CMP C               B9
0727    JNC 0720            D2 20 07    continue move
072A    MVI A,0             3E 00
072C    STA LOADSBC         32 F0 F1    reset LOADSBC
072F    LXI H,STARTn        21 Fn F1    check for start SBC n
0732    MVI A,0             3E 00
0734    CMP M               BE
0735    JZ 0734             CA 34 07    no start yet
0738    STA STARTn          32 Fn F1    reset STARTn
073B    JMP 3042            C3 42 30    start SBC

0740    MVI A,4E            3E 4E       replaced monitor
0742    OUT ED              D3 ED       instructions
0744    JMP INUST           C3 DH 03    0000 - 0006
```

```
0020    DI              F3          execute snapshot
0021    PUSH H          E5          upon execution of
0022    PUSH D          D5          a RST4 instruction
0023    PUSH B          C5
0024    PUSH PSW        F5          save registers
0025    LHLD 3040H      2A 40 30    address of snapshot
0028    PCHL            E9          transfer control
```

9 Listing of Executive Data

```
                $EJECT
                $ INCLUDE(:F1:EXDATP.SRC)
            =   /*
            =           UPDATE: 1
            =   */
            =   /*
            =   ************************************************************************
            =   ************************************************************************
            =   **
            =                           EXECUTIVE DATA
            =   ***
            =   */
I   10   1  =   DCL MODSTATUS (MAXSYSMOD) BYTE PUBLIC AT (BEGINCM);
            =           /*
            =           MODSTATUS CONTAINS THE STATUS OF ALL MODULES
I           =           IN THE SYSTEM. INDEX IS MODULE NUMBER.
            =           MODSTATUS IS UPDATED BY EXEC WITH SYS MSG 4
            =           (BIT 0 = LSB)
            =           BIT 0 - BIT 7 = 0: MODULE DOES NOT EXIST
            =           BIT 0: 1 - MODULE EXISTS
            =           BIT 1: 0 - MODULE NOT ACTIVATED
            =                  1 - MODULE ACTIVATED
            =           BIT 2: 0 - MODULE MAY NOT BE SUSPENDED
            =                  1 - MODULE MAY BE SUSPENDED
            =           BIT 3: 0 - MODULE NOT AUTHORIZED TO SUSPEND/
            =                     ACTIVATE OTHER MODULES
            =                  1 - MODULE AUTHORIZED TO SUSPEND/ACTIVATE
            =                     OTHER MODULES
            =           */
    11   1  =   DCL ADRSTAT ADDRESS PUBLIC,
            =       (MODSTAT BASED ADRSTAT) (MAXMOD) BYTE;
            =           /*
```

18

60

```
                =           MODSTAT IS AN OVERLAY FOR MODSTATUS AND COVERS
                =           THE ENTRIES FOR ONE COMPUTER.
                =           THE BASE VARIABLE ADRSTAT IS SET AT START.
                =           */
  12    1       =   DCL BEGABSDATA BYTE PUBLIC AT (BEGINCM + MAXSYSMOD),
                =           /*
                =           BEGIN OF ABSOLUTE DATA IN COMMON MEMORY
                =           */
                =       RTC (4) BYTE PUBLIC AT (.BEGABSDATA),
                =           /*
                =           SYSTEM REALTIME CLOCK
                =           LEAST SIGNIFICANT BYTE = RTC(3)
                =           LEAST SIGNIFICANT BIT : 1 MSEC
                =           */
                =       EXTMSGLOCK BYTE PUBLIC AT (.BEGABSDATA + 4),
                =           /*
                =           0 IF EXT. MSG BUFFER UNLOCKED
                =           OTHERWISE NUMBER OF CPTR CURRENTLY WORKING IN BUFFER
                =           */
                =       EXTMSG BYTE PUBLIC AT (.BEGABSDATA + 5),
                =           /*
                =           0 IF NO EXTERNAL MSG TO PROCESS
                =           OTHERWISE NUMBER OF RECEIVING CPTR
                =           */
                =       EXTMSGIN BYTE PUBLIC AT (.BEGABSDATA + 6),
                =       EXTMSGOUT BYTE PUBLIC AT (.BEGABSDATA + 7),
                =           /*
                =           POINTER TO EXTERNAL MSG BUFFER */
                =       NUMEXTMSG BYTE PUBLIC AT (.BEGABSDATA + 8),
                =           /*
                =           NUMBER OF EXT. MSG IN EXT. MSG BUFFER
                =           */
```

```
                =       EXTLASTMSG BYTE PUBLIC AT (.BEGABSDATA + 9),
                =       EXTMSGBUFFER (250) BYTE PUBLIC AT (.BEGABSDATA + 10),
                =           /*
                =           EXTERNAL MESSAGE BUFFER
                =           */
                =       ENDABSDATA BYTE PUBLIC AT (.BEGABSDATA + 258);
                =           /*
                =           END OF ABSOLUTE DATA IN COMMON MEMORY
                =           */
  13    1       =   DCL (LOADSBC,START1,START2,START3) BYTE PUBLIC AT (0F1F0H);
                =           /*
                =           ABSOLUTE LOCATED VARIABLES USED FOR LOADING AND
                =           START OF SBC'S
                =           */
  14    1       =   DCL ILLMSG (36) BYTE PUBLIC INITIAL
                =                       ('ILLEGAL MSG:RMN 00SMN 00 MN 00 ML 00');
                =           /*
                =           TEXT FOR CRT MSG 'ILLEGAL MSG'
                =           */
  15    1       =   DCL MONMSG (4) BYTE PUBLIC INITIAL (CS,EX,10,30),
                =       MONTEXT (26) BYTE PUBLIC INITIAL ('SYSTEM MONITOR: 0000 00 00'
                    );
                =           /*
                =           SYSTEM MONITOR MSG TO CS
                =           */
  16    1       =   DCL SAVESTACKPTR ADDRESS PUBLIC;
                =           /*
                =           STACK POINTER AT SYSTEM START
                =           */
  17    1       =   DCL EXTRMSG (4) BYTE PUBLIC INITIAL (DB,EX,27,0);
                =           /*
                =           EXTRACTION MSG TO DEBUG MODULE
```

```
                =           */
  18  1         =   DCL CDCADR ADDRESS PUBLIC;
                =           /*
                =           ADDRESS TO BE CALLED AT CDC INTERRUPT
                =           */
  19  1         =   DCL RSTADR ADDRESS PUBLIC,RSTFL BYTE PUBLIC;
                =           /*
                =           ADDRESS OF PROCEDURE SYSRESTART
                =           */
  20  1         =   DCL SNAPINTADR ADDRESS PUBLIC AT (0F1F6H);
                =           /*
                =           ADDRESS OF SNAPSHOT INTERRUPT ENTRY
                =           */
  21  1         =   DCL EXCLEARBEG BYTE PUBLIC;
                =           /*
                =           BEGIN OF DATA TO BE CLEARED AT START
                =           */
  22  1         =   DCL INTTBL(8) BYTE PUBLIC;
                =           /*
                =           TABLE CONTAINS INDICES OF PRIORITY PROCEDURES TO
                =           BE SCHEDULED IN CASE OF AN INTERRUPT
                =           */
  23  1         =   DCL CDCACTIVE BYTE PUBLIC;
                =           /* TRUE IF CDC INT ACTIVATED */
  24  1         =   DCL CODESAVE ADDRESS PUBLIC,
                =           /*
                =           SYSTEM MONITOR CODE FOR MSG BUFFER OVERFLOW
                =           */
                =       EXPERFL BYTE PUBLIC,
                =       EXPERADR ADDRESS PUBLIC,
                =       EXPERX BYTE PUBLIC;
                =           /*
```

```
                =           EXEC PERIODIC DATA
                =           */
     25   1     =   DCL MSGEXTRACTION BYTE PUBLIC;
                =           /*
                =           TRUE IF MSG EXTRACTION IS ACTIVATED
                =           */
     26   1     =   DCL MSGBUFFER (MSGBUFLEN) BYTE PUBLIC,
                =       OVERFLOWTEST ADDRESS PUBLIC,
                =       (MSGIN, MSGOUT) ADDRESS PUBLIC,
                =       NUMMSG BYTE PUBLIC,
                =       LASTMSG ADDRESS PUBLIC;
                =           /*
                =           MSGBUFFER IS A CIRCULAR FIFO LIST CONTAINING THE
                =             MSG WHICH ARE WAITING TO BE PROCESSED.
                =           MSGIN POINTS TO LOCATION OF NEXT MSG TO FILL IN.
                =           MSGOUT POINTS TO MSG TO BE PROCESSED NEXT.
                =           NUMMSG IS INCREMENTED WHEN A MSG IS SENT AND
                =             DECREMENTED WHEN A MSG IS PROCESSED.
                =             NUMMSG = 0: MSGBUFFER IS EMPTY.
                =           LASTMSG CONTAINS INDEX OF LAST BYTE OF LAST MSG
                =             IN MSGBUFFER + 1.
                =           */
     27   1     =   DCL PRIORLIST (MAXPRIOR) ADDRESS PUBLIC,
                =       PRIORSCHEDULE BYTE PUBLIC;
                =           /*
                =           PRIORLIST CONTAINS THE ADDRESSES OF THE ACTIVATED
                =           PRIORITY TASKS.
                =           A PRIORITY TASK IS CALLED WHEN IT IS SCHEDULED BY
                =           SETTING THE RESPECTIVE BIT IN PRIORSCHEDULE,
                =           E.G.  IF BIT 3 (LSB = BIT 0) IN PRIORSCHEDULE IS
                =           SET, THE ADDRESS IN PRIORLIST(3) IS CALLED.
                =           */
```

```
28   1  =   DCL (XB1,XB2,XB3) BYTE PUBLIC,(XA1,XA2,XA3) ADDRESS PUBLIC;
        =            /*
        =            EXEC WORK VARIABLES
        =            */
29   1  =   DCL PERXTBL (MAXPER) BYTE PUBLIC;
        =            /*
        =            PERXTBL CONTAINS POINTER TO PERLIST IN COMPACT FORM
        =            0FFH - PERIODIC NOT ACTIVATED
        =            */
30   1  =   DCL PERLIST (MAXPER) STRUCTURE (
        =              FREE BYTE,           /* TRUE - ITEM IS FREE */
        =              PERADR ADDRESS,      /* ADDRESS OF PERIODIC */
        =              PERINTADR ADDRESS,   /* ADDRESS OF TIME INTERVAL */
        =              PERTIME (4) BYTE) PUBLIC,  /* NEXT CALL TIME (RTC FORMAT) */
        =            /*
        =            PERLIST CONTAINS ALL SIGNIFICANT DATA OF AN
        =            ACTIVATED PERIODIC TASK
        =            */
        =          (PERINT BASED XA1) (4) BYTE,
        =            /*
        =            OVERLAY FOR TIME INTERVAL OF A PERIODIC
        =            */
        =          PERX BYTE PUBLIC,
        =            /*
        =            PERX IS SEARCH INDEX FOR PERLIST
        =            */
        =          NUMPER BYTE PUBLIC;
        =            /*
        =            NUMBER OF ACTIVATED PERIODICS
        =            */
31   1  =   DCL EXCLEAREND BYTE PUBLIC;
        =            /*
```

```
=           END OF DATA TO BE CLEARED AT START
=           */
```

APPENDIX B

USER'S MANUAL FOR OPERATING SYSTEM

TABLE OF CONTENTS

1 General

1.1 Introduction

This manual describes the use of the real-time operating system for dedicated multi processor micro computers.
The operating system is designed to use INTEL's single board computers SBC80/20-4,therefore it is assumed that the reader is familiar with

- single board computer hardware
- PL/M-80
- operating system ISIS-II for INTEL's MDS.

1.2 Abbreviations and Conventions

All numbers in this segment of text are decimal except as otherwise indicated.

Task - part of the program which handles a specific function of the system and consists of one or more procedures

Module - part of the program which consists of one or more (related) tasks and can be compiled separately

Executive - part of the operating system which contols the scheduling and execution of tasks

Operating System - consists of executive,system calls and system data

System - consists of operating system and all integrated user modules

SBC - Single Board Computer

MDS - Microcomputer Development System (INTEL)

MCB - Message Control Block
RMN - Receiving Module Number
SMN - Sending Module Number
MN - Message Number
ML - Message Length

EX - executive, module number in SBC 1/2/3 : 00/08/16
LP - line printer module,module number in SBC 1/2/3 : 05/13/21
CS - CRT module, module number in SBC 1/2/3 : 06/14/22
DB - debug module, module number in SBC 1/2/3 : 07/15/23

RTC - Real Time Clock
CDC - Count Down Clock

1.3 Facilities of the Operating System

The operating system is specifically designed to control complex real-time environments.
In order to meet these requirements the operating system provides:

- priority task scheduling
- communication between modules
- time dependent (periodic) scheduling of tasks
- real-time clock and count down clock
- system monitor
- commonly needed functions in form of system calls
- interrupt handling.

The operating system is able to control all possible SBC hardware configurations. INTEL's multibus system can handle up to 16 master controller, theoretically all SBCs.

Code in memory may be shared by several SBCs, however, care should be taken that there is no interference with data access. By simply keeping all data in on-board memory no data conflicts can occur.

For efficiency reasons the executive and time critical functions should be kept in on-board memory, whereas the code of the system calls can be located in common memory where it can be shared.

A module can take on the identities of several modules.
The body of the module is shared in common memory. The kernel of the module, i.e. the entry for real-time message, is special in each SBC as expressed by different module numbers.

Communication between modules allow the sending of messages from a module to another module, regardless of where these modules are located.
Since message between SBCs as well as code executed from common memory make use of the system bus the actual distribution of the modules in the entire computer system should be such that system bus access is minimized.

All executives in the SBCs are identical, the binding of an executive to the host SBC takes place during the compile with two compile paremeters: number of SBC and module number of first module in the SBC.

2 System Organization

2.1 Modular Structure

The operating system supports a strictly modular structure. Modules are integral parts of the overall system which usually handle a specific task or a group of related tasks. This organization not only eases program development and maintenance it also allows easy-to-implement changes of the system.

The links between a module and the rest of the system are the system data and the entry for real-time messages of the module.
These links require

- the inclusion of system data in the compilation of a module
- the marking of the message entry procedure as a PUBLIC procedure and a predefined name to allow access by the executive.

The example of a complete user module is given in Section 9.

A module has a unique number which is used to identify it in the system.
The number of a module depends on the number of the SBC which hosts the module. Each SBC can have a maximum of 8 modules (0-7), e.g. module 3 in SBC 2 has the module number 11.
The lowest module number in each SBC is reserved for the executive. Implementing the executive as a module allows user modules to send message to the executive.

Each possible module in the system has a reserved byte in the system table MODSTATUS. This table contains the status of all modules. If the entry is 0 for a module the module is not existent.
MODSTATUS is updated with the system call UPDSTAT (see Section 6.12).

Each module is in one of three states: nonexistent, existent or activated.

2.2 Priority Tasks

Priority tasks consists of one or more procedures with one entry procedure. The call of this procedure can be scheduled. A scheduled priority task will be considered by the executive as soon as the processor becomes available.

Usually the process of an interrupt is implemented as a priority task where the scheduling is done in the interrupt procedure.

In connection with priority tasks there are four related system calls: PRIORITY, PRIORITYINT, ENTERPRIOR and REMPRIOR (see Section 6 for formats).

A priority task has to be entered into the list of priority calls prior to its first scheduling. This is done with the system call ENTERPRIOR. ENTERPRIOR returns an index which is used to schedule a call of the task (PRIORITY or PRIORITYINT) or remove the task from the priority list (REMPRIOR).

PRIORITYINT and PRIORITY perform the same function, however, PRIORITY may not be called from an interrupt procedure and vice versa. This prevents erroneous program action in the case that the processor is interrupted while executing the priority scheduling system call.

2.3 Communication Between Modules

Since modules are separate and independent parts of the system, the operating system has to provide a function which allows the modules to communicate with each other.
This communication takes place in form of a message exchange. Messages are sent and received by modules and the sender and receiver are identified by their module numbers.

A message consists of a message control block (MCB) and, if necessary, data bytes.
The MCB has a length of 4 bytes and the following structure:

```
**********
*   RMN    *   Receiving Module Number
**********
*   SMN    *   Sending Module Number
**********
*   MN     *   Message Number
**********
*   ML     *   Message Length
**********
```

The MN is an agreed upon number between sender and receiver of the message in order to identify the meaning of the message.
ML determines the total number of bytes of the message (MCB + data bytes) and has a minimum value of 4 (message without data bytes).

A message is sent with the system call SEND. This routine requires the address of the first byte of the MCB to be passed to it.
SEND returns a 'TRUE' if the message has been sent and a 'FALSE' otherwise. The latter can happen when the receiving module is not activated or not existent.
If the message has been sent, data bytes may be stored in the vector MSGDATA. SEND has reserved space for as many data bytes as indicated by ML in the MCB.
In the case that more data bytes than specified are written into MSGDATA they will be disregarded.
The message is inserted into a circular FIFO list.
The executive checks the top of the list and transfers control to the message entry of the module specified by RMN in the MCB.
Prior to calling the module the message to be processed is set into the vectors MSG and MSGDATA to allow access by the processing module.

2.4 Time Dependent (Periodic) Scheduling of Tasks

A common requirement of real-time applications is the time dependent execution of tasks, e.g. a display has to be updated every two seconds or a process has to be triggered once after a certain amount of time has elapsed.
The operating system provides the functions for executing tasks of this kind.

The time dependent scheduling of tasks is implemented with three system calls: PERACT, PERCHG and PERSUSP (see Section 6 for formats).

A procedure can be entered into the list of periodic tasks by calling PERACT and passing 2 parameters: the address of the procedure entry and the time interval.
PERACT returns an index which identifies the task in the system calls PERCHG and PERSUSP.

The time interval can be changed with the system call PERCHG and a periodic task can be removed from the list with a call of PERSUSP.

The executive continously checks the list of periodic tasks and calls the respective procedure when the specified time is less than or equal to the value in the real-time clock.

.5 Background Tasks

In order to utilize the processor in case that no priority call is scheduled, no message is to be to processed and no periodic call is necessary, background tasks can be called by the executive.
Typically, not time dependent hardware checks are performed as a background tasks on a time slice basis.

3 Interrupt Handling

All interrupts are handled by the operating system.
User interrupts currently can be activated on three levels: INT3, INT4 and INT5.

An interrupt is activated with the system call ENTERINT. Parameters for this system call are the requested interrupt level and the index of the priority task to be scheduled upon occurrence of the interrupt.

An activated interrupt can be de-activated with the system call REMINT.

4 System Data

The set of EXTERNAL system data declarations as listed in Section 11 has to be included in the compilation of a module. The PUBLIC definitions of the same data are compiled together with the executive.

Care should be taken in the use of the system work variables. Since all modules are allowed to use them (except in interrupt routines) their contents is not preserved from one call to the next.

5 Executive

This section describes the function of the executive and the interface between executive and user module.

The executive of the operating system is a program loop which checks for pending tasks on 4 levels:

- priority tasks
- messages to be processed
- periodic tasks
- background tasks.

The executive only proceeds to the next lower level if no task is pending at the current or a higher level.

The only link between the executive and a user module is the message entry of the module.
The executive is compiled with all possible message entries declared as EXTERNAL procedures while the respective PUBLIC declaration is part of the module.
During the linking process of ISIS-II the two declarations are matched and the ececutive can call the message entry of the module at system start. It is up to the module to initialize itself, enter priority calls, activate periodic calls and set its own status.

Prior to calling a module's message entry, the executive 'sets' the message into the based vectors MSG and MSGDATA where MSG contains the MCB and MSGDATA the data bytes, if any.

Each SBC runs its own, identical executive. The only difference between the ececutives is the module number.

6 System Calls

This section describes format and function of the system calls.

All system calls are given in the form

NAME TYPE (parameter list).

A TYPE included after the name means that the system call returns a value of the indicated type.

An/Bn in the parameter list indicate address/byte values.

6.1 ENTERPRIOR

Format : ENTERPRIOR BYTE (A1)

Input : A1 - address of priority procedure

Output : . index of the priority task
(may be used for scheduling and removing the task)
. FF if the task could not be entered into the list of priority tasks (overflow)

Function : The indicated priority procedure is entered into the list of priority tasks. The index to this list is returned. The list can hold up to 8 tasks/SBC.

6.2 PRIORITYINT/PRIORITY

Format : PRIORITYINT/PRIORITY (B1)

Input : B1 - index of priority task

Function : A single call of the indicated priority procedure is initiated by the executive. The call occurs as soon as the CPU becomes available.

Remarks : PRIORITYINT is to be called from interrupt procedures only and PRIORITY is not to be called from interrupt procedures!
Prior to scheduling a priority task it has to be entered with ENTERPRIOR, otherwise erroneous program action will occur.

6.3 REMPRIOR

Format : REMPRIOR (B1)

Input : B1 - index of priority task

Function : The indicated priority task is removed from the list of priority tasks and can no longer be scheduled.

6.4 SEND

Format : SEND BYTE (A1)

Input : A1 - address of first byte of MCB

Output : TRUE if message sent
FALSE otherwise
MSGDATA is set to contain the data bytes of the message.

Function : A message is sent to another module with this system call.
If the output is TRUE the message is sent and the data bytes can be written into MSGDATA, if any.
If the output is FALSE the message could not be sent. This may be caused by a not activated receiving module or an overflow in the system's message buffer.

6.5 PERACT

Format : PERACT BYTE (A1,A2)

Input : A1 - address of entry procedure for periodic task

A2 - address of first byte of time interval
(The time interval is expected to be in RTC format.)

Output : . index of periodic task
. FF if task could not be activated because of an overflow in the list

Function : The call of the indicated periodic task is activated with the input time interval.

Remark : The task is called as soon as possible and from then on with the time given interval.

6.6 PERCHG

Format : PERCHG (B1,A1)

Input : B1 - index of periodic task
A1 - address of first byte of time interval (in RTC format)

Function : The time interval of the periodic task is changed.

Remark : The periodic task is called next at RTC + new time interval.

6.7 PERSUSP

Format : PERSUSP (B1)

Input : B1 - index of periodic task

Function : The periodic task is removed from the list of periodic tasks and is not called any more.

6.8 ACTIVATE

Format : ACTIVATE BYTE (B1,B2)

Input : B1 - number of module to be activated
B2 - number of calling module

Output : FALSE if module to be activated does not exist or calling module not authorized
TRUE otherwise

Function : The indicated module is to be activated.
The module receives a 'restart' message to indicate that it has to reinitialize itself.

6.9 ACTIVE

Format : ACTIVE BYTE (B1)

Input : B1 - module number

Output : TRUE if the indicated module is active
FALSE otherwise

Function : The system call checks the status of the input module and returns a TRUE if the module is active.

6.10 SUSPEND

Format : SUSPEND BYTE (B1,B2)

Input : B1 - number of module to be suspended
B2 - number of calling module

Output : FALSE if the module cannot be suspended or calling module not authorized
TRUE otherwise

Function : The indicated module receives a 'suspend' message. Its status is changed from 'active' to 'existent'.

6.11 PROCADR

Format : PROCADR ADDRESS

Output : first address of calling procedure

Function : This system call is used to determine the location of a procedure at run time.

Remark : Since PROCADR examines the system stack, it only returns a correct value if called with the following sequence of code:

```
XYZ:  PROCEDURE;
      IF FLAG = 0 THEN
        DO;
        XYZADR = PROCADR;
        FLAG = 1;
        RETURN;
        END;
      ..
      ..
      procedure body
      ..
      END XYZ;
```

6.12 UPDSTAT

Format : UPDSTAT (B1,B2)

Input : B1 - module number
B2 - status
bit 0 : 0 - module does not exist
1 - module exists
bit 1 : 0 - module not acticated
1 - module activated
bit 2 : 0 - module may not be suspended
1 - module may be suspended
bit 3 : 0 - module not authorized to suspend/activate others
1 - module authorized to suspend/activate others

Function : The current status of module B1 is replaced by B2.

Remark : The status of a module has to be set when the module receives the 'start' message in order to change its status from 'not existent' to 'existent' or 'activated'.

6.13 CLEARDATA

Format : CLEARDATA (A1,A2)

Input : A1 - address of first byte
A2 - address of last byte

Function : Clear data from A1 to A2.

6.14 BIT

Format : BIT BYTE (B1,B2)

Input : B1 - number of bit
B2 - byte to be checked

Output : TRUE if bit B1 in B2 is set
FALSE otherwise

6.15 CLEARBIT

Format : CLEARBIT (B1,A1)

Input : B1 - number of bit
A1 - address of a memory location

Function : Bit B1 in memory location A1 is set to 0.

6.16 SETBIT

Format : SETBIT (B1,A1)

Input : B1 - number of bit
A1 - address of memory location

Function : Bit B1 in memory location A1 is set to 1.

6.17 SETCDC

Format : SETCDC BYTE (A1,A2)

Input : A1 - time interval (LSB = 1.86 micro sec)
A2 - address to be called at CDC interrupt

Output : FALSE if CDC already activated
TRUE otherwise

6.18 ILLEGALMSG

Format : ILLEGALMSG

Function : Process undefined message for a module.
An asynchronous message giving the MCB of the undefined message is initiated at the CRT.
The undefined message is taken out of common vector MSG.

6.19 ENTERINT

Format : ENTERINT BYTE (B1,B2)

Input : B1 - interrupt level
B2 - index of priority task

Output : FALSE if same interrupt already occupied
TRUE otherwise

Function : The interrupt on level B1 is enabled.
Upon occurrence of the interrupt the call of priority task with index B2 is scheduled.

6.20 REMINT

Format : REMINT (B1)

Input : B1 - interrupt level

Function : The enabled interrupt on level B1 is disabled.

7 System Monitor

The system monitor is implemented as a system call and may be called when an internal error condition occurs, program limits are exceeded etc.

The system monitor generates an 'asynchronous output' message to a CRT module to indicate nature and location of the special condition.

Format : CALL SYSMON (A1,B1,B2);

A1 any address value
B1 and B2 are any byte values

The generated asynchronous CRT output has the following format:

*** SYSTEM MONITOR : XXXX YY ZZ

where XXXX = (A1)
YY = (B1)
ZZ = (B2)

It is recommended to set A1 to the stack pointer when calling the system monitor. In this case the displayed value XXXX will point to the memory location from where the system monitor was called.
B1 and B2 can be used to further specify the condition.

Up to now the system monitor is called with B1 representing the module number and B2 a specification within the module.

B1	B2	condition
0	1	internal message buffer overflow
0	2	priority list overflow
0	3	periodic list overflow
0	4	use of illegal periodic index
0	5	use of illegal priority index
0	6	external message buffer overflow
0	7	double occupation of interrupt level
5	0	line printer output buffer overflow

8 Real-time Clock and Count Down Clock

The operating sytem provides two system clocks:

Real-time Clock (RTC):

The RTC is a public 4 byte vector. The least significant byte is RTC(3).
The value of the least significant bit is 1 millisecond.
Maximum time value is 4294967 sec = 71582 min = 1193 hrs = approx. 50 days.
The RTC is located in common memory and updated by SBC 1.

Count Down Clock (CDC):

The operating system provides a CDC for each SBC.
The CDC can be activated by any module with the system call SETCDC.
The CDC has a length of two bytes and the least significant bit is 1.86 micro sec.

9 System Loader and System Start

The system is started automatically after the loading is completed.

To load and start the system the following steps have to be performed:

- RESET and load ISIS-II
- enter monitor and change locations F1F4 and F1F5 to 0ABH and 0CDH respectively
- RESET and load ISIS-II
- load and execute the loader by typing 'LOADER'
- the loader issues informative messages as it proceeds in the loading process.

The loader expects the code to be loaded into the SBCs to be in the files LOAD1,LOAD2 and LOAD3 respectively.
These files are read from disk drive 0.

In case a file cannot be found, a warning message is typed and and the loading process continued.

The loader terminates if file LOAD1 cannot be found because the system cannot be started without SBC1 in this configuration. (In a general application, however, any number of SBCs can be loaded and started in any sequence.)

It should be noted that in a practical application of the operating system the on-board code would reside in ROM.

10 Examole of a User Module

This section describes an example of a typical user module.

The example is given in form of a source listing in PL/M-80 to be compiled under ISIS-II.

```
DEMO:       DO;

$INCLUDE(:F1:SYDATE.SRC)
    /* include external definitions of system data */

/*
*****************************************************************
**
**          DEMO SUBSTITUTIONS
*/
DCL         DM LIT '03',    /* module number of DEMO */
            ..

            ..
            ENDSUBST LIT '0';

/*
*****************************************************************
**
**          DEMO VARIABLE DATA
*/
DCL         VARDATABEG BYTE,
            (DMPERX,DMPRIORX) BYTE,
                /* storage for DM periodic and priority
                   indices*/

            (DMPERFL,DMPRIORFL) BYTE,
                /* flags controlling execution of DMPER
                   and DMPRIOR */

            (DMPERADR,DMPRIORADR) ADDRESS,
                /* storage for entry addresses of periodic
                   and priority procedures */

            ..

            ..
            VARDATAEND BYTE;
```

```
/*
*****************************************************************
**
**          DEMO CONSTANT DATA
*/
DCL         DMTIMEINT (4) BYTE INITIAL (0,0,3,0E8H),
                /* time interval for DM periodic : 4 sec */

            DMMSG (4) BYTE INITIAL (4,DM,10,7),
                /* MCB for message to module 4 */
            ..

            ..
            DMENDCONST BYTE;

$INCLUDE(:F1:SCEXT.SRC)
    /* include external definitions of system calls */

/*
*****************************************************************
**
**          DEMO UTILITY ROUTINES
*/
DMUT1:      PROCEDURE;
            ..
            ..
            END DMUT1;

            ..
            ..

DMUTn:      PROCEDURE;
            ..
            ..
            END DMUTn;
/*
*****************************************************************
**
**          DEMO PRIORITY PROCEDURE(S)
*/
DMPRIOR:    PROCEDURE;

            IF DMPRIORFL = 0 THEN
                /* first call,find entry address of DMPRIOR */
              DO;
              DMPRIORADR = PROCADR;
              DMPRIORFL = 1;
              RETURN;
              END;
            ..

            ..
            END DMPRIOR;
```

```
/*
**********************************************************************
**
**          DEMO PERIODIC PROCEDURE(S)
*/
DMPER:      PROCEDURE;

            IF DMPERFL = 0 THEN
                /* first call,find entry address of DMPER */
              DO;
              DMPERADR = PROCADR;
              DMPERFL = 1;
              RETURN;
              END;
            ..

            ..
            END DMPER;
/*
**********************************************************************
**
**          DEMO START PROCEDURE
*/
DMSTART:    PROCEDURE;

            CALL CLEARDATA(.VARDATABEG,.VARDATAEND);
                /* clear variable data */

            CALL DMPRIOR;
            CALL DMPER;
                /* determine entry addresses of procedures */

            DMPRIORX = ENTERPRIOR(DMPRIORADR);
                /* enter priority call of DMPRIOR */

            DMPERX = PERACT(DMPERADR,.DMTIMEINT(0));
                /* activate periodic call of DMPER */

            CALL UPDSTAT(DM,3);
                /* set module's status
                    3 - existent and activated */

            IF NOT SEND(.DMMSG(0)) THEN RETURN;
                /* send demo message to module 4 with data
                   bytes 0,1,2 */
            DO B1 = 0 TO 2;
              MSGDATA(B1) = B1;
              END;
            ..

            ..
            END DMSTART;
```

```
/*
**********************************************************************
**
**          DEMO REALTIME MESSAGE ENTRY
*/
MSGENT03:   PROCEDURE PUBLIC;

            IF MSG(MN) = 0 THEN
                /* start message */
              DO;
              CALL DMSTART;
              RETURN;
              END;
            ..
            ..

            /* process of other defined real-time message */

            ..
            ..
            END MSGENT03;
```

11 Listing of System Data

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EXECUTIVE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:EXEC.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1          EXECUTIVE:   DO;
              $ INCLUDE(:F1:SYDATP.SRC)
         =    /*
         =                    UPDATE: 1
         =    */
         =    /*
         =    ****************************************************************************
         =    ****************************************************************************
         =    **
         =                      SYSTEM DATA DECLARATIONS
         =    ***
         =    */
   2   1 =    DECLARE LIT LITERALLY 'LITERALLY',
         =            DCL LIT 'DECLARE';
         =
   3   1 =    DCL TRUE LIT '0FFH',
         =        FALSE LIT '0',
         =        RMN LIT '0',
         =        SMN LIT '1',
         =        MN LIT '2',
         =        ML LIT '3',
         =        MAXCPTR LIT '3',          /* MAX NUMBER OF COMPUTERS */
         =        MAXMOD LIT '8',           /* MAX NUMBER OF MODULES IN 1 CPTR */
         =        MAXSYSMOD LIT '24',       /* MAX NUMBER OF MODULES IN SYSTEM */
         =        MSGBUFLEN LIT '500',      /* LENGTH OF MSG BUFFER */
         =        MAXPRIOR LIT '8',         /* MAX NUMBER OF PRIORITY CALLS */
```

```
                =       MAXPER LIT '8',             /* MAX NUMBER OF PERIODIC CALLS */
                =       BEGINCM LIT '0F000H',       /* BEGIN OF COMMON MEMORY */
                =       INTVECTOR LIT '3000H',    /* INTERRUPT VECTOR */
                =       CPTR LIT '1',               /* NUMBER OF HOST COMPUTER */
                =       FIRSTMN LIT '0',            /* MN OF FIRST MODULE IN CPTR */
                =       LASTMN LIT 'FIRSTMN+7',   /* MN OF LAST MODULE IN CPTR */
                =       EX   LIT 'FIRSTMN',         /* MN OF EXEC MODULE */
                =       LP LIT 'FIRSTMN+5',         /* MN OF LINE PRINTER MODULE */
                =       CS LIT 'FIRSTMN+6',         /* MN OF CRT MODULE */
                =       DB    LIT 'FIRSTMN+7';      /* MN OF DEBUG MODULE */
                =
                =   /*
                =   ***
                =   */
  4     1       =   DCL (B1,B2,B3) BYTE PUBLIC,
                =       (A1,A2,A3,A4) ADDRESS PUBLIC,
                =       (BU,BL) BYTE PUBLIC AT (.A4);
                =             /*
                =             SYSTEM WORK VARIABLES
                =             BU(UPPER) AND BL(LOWER) OVERLAY ADDRESS VARIABLE A4
                =             */
  5     1       =   DCL (ADRMSG,ADRMSGDATA) ADDRESS PUBLIC,
                =       (MSG BASED ADRMSG) (4) BYTE,
                =       (MSGDATA BASED ADRMSGDATA) (100) BYTE;
                =             /*
                =             WORK AREAS FOR MSG CONTROL BLOCK (MSG) AND MSG
                =             DATA BYTES (MSG DATA)
                =             ADRMSG AND ADRMSGDATA ARE SET BY EXEC BEFORE THE
                =             CALL OF A MODULES MSG HANDLER
                =             */
  6     1       =   DCL RESTART BYTE PUBLIC;
                =           /*
```

```
                =           FALSE IF START WITH SYSTEM RESET
                =           TRUE  IF RESTART WITHOUT SYSTEM RESET
                =           */
  7   1         =   DCL INTMASK BYTE PUBLIC;
                =           /*
                =           MASK OF CURRENTLY ACTIVATED INTERRUPTS
                =           */
  8   1         =   DCL DEBUGMCB (4) BYTE PUBLIC;
                =           /*
                =           MCB FOR DEBUG PURPOSES
                =           */
                =
                =   /*
                =   ************************************************************************
                =   **
                =                                   SYSTEM MESSAGES
                =   ***
                =   */
  9   1         =   DCL SYSMN0(4) BYTE PUBLIC INITIAL (0,EX,0,4),
                =            /*
                =           INTERNAL START MSG, SENT TO ALL MODULES IN CPTR
                =           */
                =       SYSMN1(4) BYTE PUBLIC INITIAL (0,EX,1,4),
                =           /*
                =           RESTART MSG, SENT TO MODULE TO BE ACTIVATED
                =           */
                =       SYSMN5(4) BYTE PUBLIC INITIAL (0,EX,5,4);
                =           /*
                =           SUSPEND MSG, SENT BY EXEC TO MODULE TO BE SUSPENDED
                =           */
```

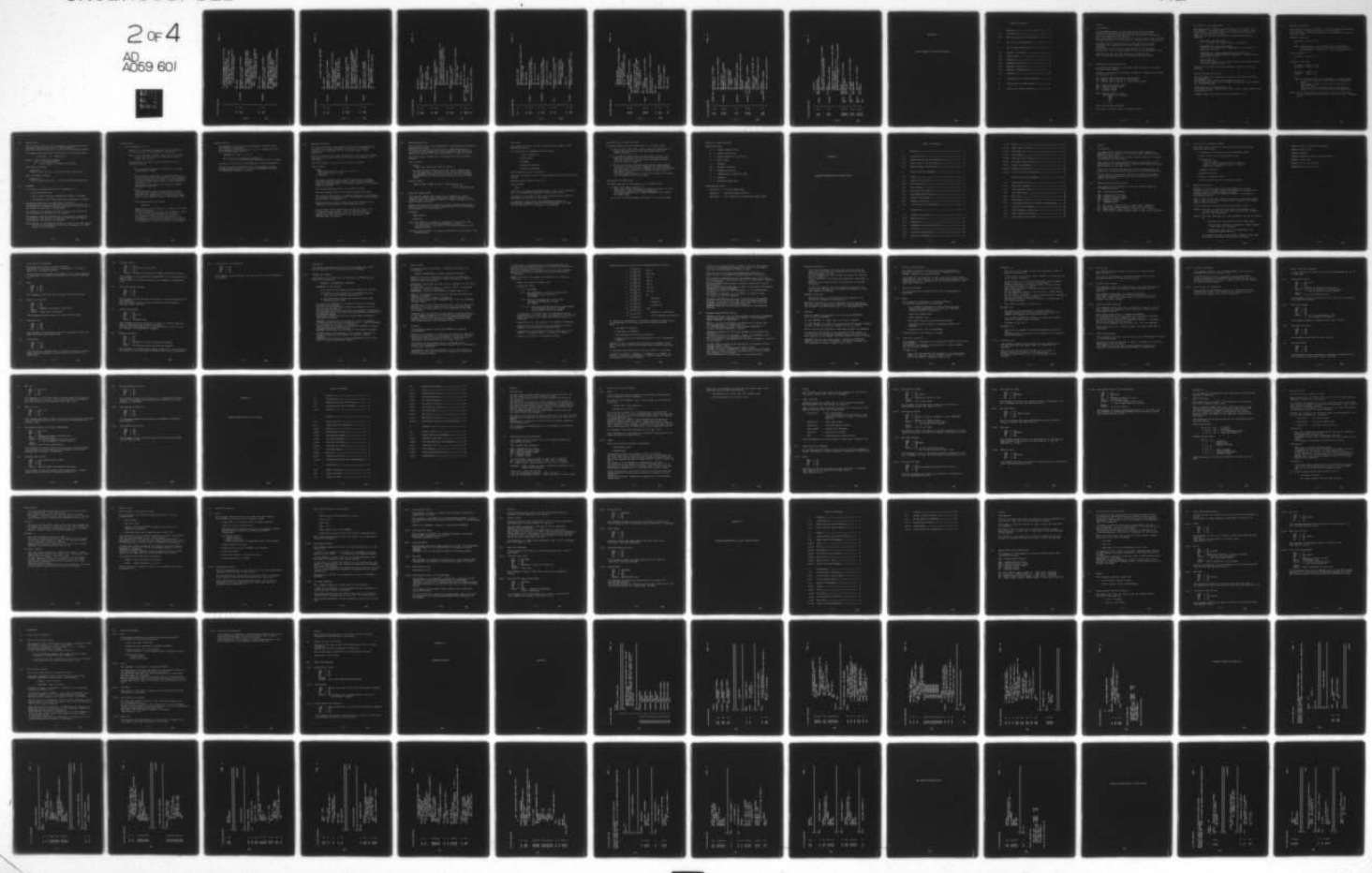
12 Listing of External System Call Declarations

```
                $EJECT
                $ INCLUDE(:F1:SCEXT.SRC)
          =     /*
          =     ****************************************************************************
          =     ****************************************************************************
          =     **
          =                                       SYSTEM CALLS
          =     **
          =     ***
          =     */
   30     1 =   CLEARDATA:  PROCEDURE (P1,P2) EXTERNAL;
   31     2 =               DECLARE (P1,P2) ADDRESS;
          =               /*
          =               CLEAR MODULES VARIABLE DATA REGION
          =               P1 - ADDRESS OF FIRST BYTE
          =               P2 - ADDRESS OF LAST BYTE
          =               */
   32     2 =               END CLEARDATA;
   33     1 =   SENDEXT:    PROCEDURE BYTE EXTERNAL;
          =               /*
          =               SEND MSG TO MODULE IN OTHER CPTR
          =               */
   34     2 =               END SENDEXT;
   35     1 =   RECEXT:     PROCEDURE EXTERNAL;
          =               /*
          =               TRANSFER EXTERNAL MSG INTO OWN MSG BUFFER
          =               */
   36     2 =               END RECEXT;
   37     1 =   SEND:       PROCEDURE (P1) BYTE EXTERNAL;
   38     2 =               DECLARE P1 ADDRESS;
          =               /*
          =               SEND A MSG TO ANOTHER MODULE IN THE SYSTEM
```

AD-A059 601 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A REAL-TIME OPERATING SYSTEM FOR SINGLE BOARD COMPUTER BASED DI--ETC(U)
JUN 78 W NIEMANN

UNCLASSIFIED NL

2 OF 4
AD A059 601



```
                =                   P1 CONTAINS THE ADDRESS OF THE 1ST BYTE
                =                      OF THE MSG CONTROL BLOCK.
                =                   SEND RETURNS AN ADDRESS VALUE:
                =                     - THE ADDRESS TO STORE THE 1ST DATA
                =                       BYTE OF THE MSG
                =                     - 0 IF THE RECEIVING MODULE DOES NOT
                =                         EXIST OR IS NOT ACTIVATED.
                =                   */
     39    2    =                   END SEND;
                =
     40    1    =   ACTIVATE:       PROCEDURE (P1,P2) BYTE EXTERNAL;
     41    2    =                   DECLARE (P1,P2) BYTE;
                =                   /*
                =                   ACTIVATE A CURRENTLY SUSPENDED MODULE.
                =                   P1 - NUMBER OF MODULE TO BE ACTIVATED
                =                   P2 - NUMBER OF CALLING MODULE
                =                   ACTIVATE RETURNS A BYTE VALUE
                =                     TRUE - OPERATION O.K.
                =                     FALSE - CALLING MODULE NOT AUTHORIZED
                =                             OR MODULE DOES NOT EXIST
                =                   */
     42    2    =                   END ACTIVATE;
                =
     43    1    =   SUSPEND:        PROCEDURE (P1,P2) BYTE EXTERNAL;
     44    2    =                   DECLARE (P1,P2) BYTE;
                =                   /*
                =                   SUSPEND A CURRENTLY ACTIVE MODULE.
                =                   P1 - NUMBER OF MODULE TO BE SUSPENDED
                =                   P2 - NUMBER OF CALLING MODULE
                =                   SUSPEND RETURNS A BYTE VALUE
                =                     TRUE  - OPERATION O.K.
                =                     FALSE - CALLING MODULE NOT AUTHORIZED
```

```
                =                       OR MODULE CANNOT BE SUSPENDED
                =                   */
     45    2    =                   END SUSPEND;
                =
     46    1    =   ACTIVE:         PROCEDURE (P1) BYTE EXTERNAL;
     47    2    =                   DECLARE P1 BYTE;
                =                   /*
                =                   CHECK WHETHER A MODULE IS ACTIVATED
                =                   P1 - MODULE NUMBER
                =                   ACTIVE RETURNS A BYTE VALUE
                =                     TRUE  - MODULE IS ACTIVE
                =                     FALSE - MODULE IS NOT ACTIVE
                =                   */
     48    2    =                   END ACTIVE;
                =
     49    1    =   PERACT:         PROCEDURE (P1,P2) BYTE EXTERNAL;
     50    2    =                   DECLARE (P1,P2) ADDRESS;
                =                   /*
                =                   ACTIVATE PERIODIC CALL OF A PROCEDURE
                =                   P1 - ADDRESS OF PERIODIC PROCEDURE
                =                   P2 - ADDRESS OF TIME INTERVAL
                =                   PERACT RETURNS THE INDEX OF THE PERIODIC
                =                   */
     51    2    =                   END PERACT;
                =
     52    1    =   PERCHG:         PROCEDURE (P1,P2) EXTERNAL;
     53    2    =                   DECLARE P1 BYTE,P2 ADDRESS;
                =                   /*
                =                   CHANGE THE TIME INTERVAL OF A PERIODIC
                =                   P1 - PERIODIC INDEX
                =                   P2 - ADDRESS OF NEW TIME INTERVAL
                =                   */
```

```
 54   2  =                END PERCHG;
         =
 55   1  =   PERSUSP:     PROCEDURE (P1) EXTERNAL;
 56   2  =                DECLARE P1 BYTE;
         =                /*
         =                SUSPEND A PERIODIC
         =                    P1 - PERIODIC INDEX
         =                */
 57   2  =                END PERSUSP;
         =
 58   1  =   ENTERPRIOR: PROCEDURE (P1) BYTE EXTERNAL;
 59   2  =                    DECLARE P1 ADDRESS;
         =                    /*
         =                    ENTER NEW PRIORITY CALL IN PRIORITY LIST.
         =                    P1 - ADDRESS OF PRIORITY PROCEDURE
         =                    ENTERPRIOR RETURNS THE PRIORITY INDEX
         =                    */
 60   2  =                    END ENTERPRIOR;
         =
 61   1  =   PRIORITY:    PROCEDURE (P1) EXTERNAL;
 62   2  =                    DECLARE P1 BYTE;
         =                    /*
         =                    REQUEST CALL OF A PRIORITY PROCEDURE
         =                    P1 - PRIORITY INDEX
         =                    */
 63   2  =                    END PRIORITY;
         =
 64   1  =   PRIORITYINT: PROCEDURE (P1) EXTERNAL;
 65   2  =   DCL      P1 BYTE;
 66   2  =            END PRIORITYINT;
         =
 67   1  =   REMPRIOR:PROCEDURE (P1) EXTERNAL;
```

```
 68   2  =                    DECLARE P1 BYTE;
         =                    /*
         =                    REMOVE PRIORITY CALL FROM PRIORITY LIST
         =                    P1 - PRIORITY INDEX
         =                    */
 69   2  =                    END REMPRIOR;
         =
 70   1  =   SETBIT:      PROCEDURE (P1,P2) EXTERNAL;
 71   2  =                    DECLARE P1 BYTE,P2 ADDRESS;
         =                    /*
         =                    SET BIT P1 IN BYTE P2
         =                    */
 72   2  =                    END SETBIT;
         =
 73   1  =   CLEARBIT:    PROCEDURE (P1,P2) EXTERNAL;
 74   2  =                    DECLARE P1 BYTE,P2 ADDRESS;
         =                    /*
         =                    CLEAR BIT P1 IN BYTE P2
         =                    */
 75   2  =                    END CLEARBIT;
         =
 76   1  =   BIT:          PROCEDURE (P1,P2) BYTE EXTERNAL;
 77   2  =                    DECLARE (P1,P2) BYTE;
         =                    /*
         =                    CHECK BIT P1 IN BYTE P2
         =                    RETURN TRUE IF BIT IS SET,
         =                           FALSE OTHERWISE
         =                    */
 78   2  =                    END BIT;
         =
 79   1  =   SYSMON:       PROCEDURE (P1,P2,P3) EXTERNAL;
 80   2  =                    DECLARE P1 ADDRESS,(P2,P3) BYTE;
```

```
                =                       /*
                =                       SYSTEM MONITOR
                =                       MAY BE CALLED WHEN AN INTERNAL ERROR
                =                       CONDITION EXISTS.
                =                       P1 - LOCATION OF ERROR
                =                       P2,P3 - SPECIFICATION OF ERROR
                =                       */
     81    2    =                       END SYSMON;
     82    1    =   TIMECHK:      PROCEDURE (P1) BYTE EXTERNAL;
     83    2    =             DECLARE P1 ADDRESS;
                =             /*
                =             CHECK RTC AGAINST INPUT TIME
                =             P1 - ADDRESS OF TIME IN RTC FORMAT
                =             RETURN TRUE IF INPUT TIME < RTC
                =                    FALSE OTHERWISE
                =             */
     84    2    =             END TIMECHK;
     85    1    =   SHFT:           PROCEDURE (P1) BYTE EXTERNAL;
     86    2    =             DECLARE P1 BYTE;
                =             /*
                =             RETURN A BYTE WITH A '1' SHIFTED INTO
                =             BIT POSITION P1
                =             */
     87    2    =             END SHFT;
                =
     88    1    =   PACKMCB:      PROCEDURE (P1) EXTERNAL;
     89    2    =                 DCL P1 ADDRESS;
                =                 /*
                =                 PACK MCB INTO MSGBUFFER
                =                 */
     90    2    =                 END PACKMCB;
                =
```

```
 91   1  =   SETPERTIME: PROCEDURE (P1) EXTERNAL;
 92   2  =                DCL P1 BYTE;
      =                   /*
      =                   SET NEXT CALL TIME OF A PERIODIC
      =                   */
 93   2  =                END SETPERTIME;
 94   1  =   UPDSTAT:     PROCEDURE (P1,P2) EXTERNAL;
 95   2  =                DECLARE (P1,P2) BYTE;
      =                   /*
      =                   UPDATE MODULE STATUS
      =                   P1 - MODULE NUMBER
      =                   P2 - STATUS
      =                   */
 96   2  =                END UPDSTAT;
 97   1  =   PROCADR:     PROCEDURE ADDRESS EXTERNAL;
      =                   /*
      =                   RETURN ADDRESS OF BEGIN OF CALLING
      =                   PROCEDURE
      =                   */
 98   2  =           END PROCADR;
 99   1  =   CONVASC:     PROCEDURE (P1) EXTERNAL;
100   2  =           DECLARE P1 BYTE;
      =                   /*
      =                   CONVERT 2 DIGIT HEX NUMBER INTO
      =                   2 ASCII CODES
      =                   P1 - NUMBER
      =                   OUTPUT IN SYSTEM VARIABLE BU,BL
      =                   */
101   2  =           END CONVASC;
102   1  =   ILLEGALMSG: PROCEDURE EXTERNAL;
      =                   /*
      =                   PROCESS ILLEGAL MSG RECEIVED BY
```

```
                =                   A MODULE
                =                   */
  103   2       =               END ILLEGALMSG;
  104   1       =   DEBLK:      PROCEDURE EXTERNAL;
                =                   /*
                =                   MOVE POINTER B1 TO 1ST NON-BLANK IN MSGDATA
                =                   STARTING AT LOCATION MSGDATA(B1)
                =                   */
  105   2       =               END DEBLK;
  106   1       =   GETNUM:         PROCEDURE BYTE EXTERNAL;
                =                   /*
                =                   CONVERT ASCII CODES IN MSGDATA TO HEX NUMBER
                =                   START AT MSGDATA(B1)
                =                   CONVERTED NUMBER IS LEFT IN A4
                =                   RETURN TRUE IF NUMBER IS O.K., FALSE OTHERWISE
                =                   */
  107   2       =               END GETNUM;
  108   1       =   SETCDC:     PROCEDURE (P1,P2) BYTE EXTERNAL;
  109   2       =   DCL             (P1,P2) ADDRESS;
  110   2       =               END SETCDC;
                =
  111   1       =   ENTERINT:       PROCEDURE (P1,P2) BYTE EXTERNAL;
  112   2       =   DCL             (P1,P2) BYTE;
  113   2       =               END ENTERINT;
                =
  114   1       =   REMINT:         PROCEDURE (P1) EXTERNAL;
  115   2       =   DCL             P1 BYTE;
  116   2       =               END REMINT;
  117   1           END;
```

APPENDIx C

USER'S MANUAL FOR DEBUG FUNCTIONS

TABLE OF CONTENTS

1 General

1.1 Introduction

The implemented debug functions are specifically designed to ease the debugging of user modules running under the control of the real-time operating system without interference with the operation of the system.
Since the debug module (DB) is a user module itself, the functions are available only if this module is integrated into the system.

The debug module communicates with the user using the CRT, therefore the integration of the CRT module (CS) is also necessary.
If the user wishes to obtain hard copies of the debugging results, the line printer module (LP) has to be integrated, too.

Inputs at the CRT are terminated with the RETURN key.
The line editing functions of the CS module can be used.

1.2 Abbreviations and Conventions

All inputs and outputs of the debug functions are in the hexadecimal number system.

Numbers in this text are decimal, hexadecimal numbers are trailed by the letter 'H'.

DB - module identification of debug module
CS - module identification of CRT module
LP - module identification of line printer module

RMN - receiving module number
SMN - sending module number
MN - message number
ML - message length

MCB - message control block
MCB consists of 4 bytes:
RMN
SMN
MN
ML

SBC - single board computer

hard copy - printer output of debug function

2 Initiation of the Debug Mode

The debug mode is entered with the input of 'Control-D' at the CRT keyboard. The debug module, if integrated, responds with 'DEBUG CPTR:' and expects the input of the number of the SBC which the user wishes to debug.
Depending on the input the following system reactions are possible:

- SBC number not specified:
  A '?' is typed and 'DEBUG CPTR:' is repeated.

- Requested SBC is not activated:
  'CPTR NOT ACTIVATED,DEBUG TERMINATED' is typed and the debug mode is left.

- Requested SBC already debugged from another SBC:
  'CPTR ALREADY DEBUGGED,DEBUG TERMINATED' is typed and the debug mode is left.

- Input accepted:
  The system responds with a new line and the prompt character of the debug function '>'.

Now any of the debug commands described in Section 3 may be entered.

Use of the Debug Functions

In this section the inputs and outputs of all debug functions are described.
Furthermore, some hints for the usage are given.
In the following description, system outputs are given prefaced by: 'SYSTEM OUTPUTS'.

Debug commands are prompted with '>'.
If a debug command requires more than 1 input, these commands are prompted with ':'.

Illegal inputs are reported with the output of '?'.

3.1 Inspect and Change

This function allows the display of byte and address values and is invoked by 'A' (for address) or 'B' (for byte).
The 'C' command allows the change of the address or byte value displayed last.

Address inspection:

```
>A XXXXcr   'AXXXX : YYYY
>'
```

Note: The contents of the address value is displayed in the format a user would read it. The internal representation in memory is in the reverse byte form.

Byte inspection:

```
>B XXXXcr   'BXXXX : YY
>'
```

Change of contents:

```
>A XXXXcr   'AXXXX : YYYY
>'CZZZZcr   'AXXXX : ZZZZ
>'

>B XXXXcr   'BXXXX : YY
>'CZZcr     'BXXXX : ZZ
>'
```

Note: A change command can be repeated, it always refers to the last inspected memory location (address or byte).
The change command is only legal following an inspect command.
The input of address values is in user readable format and not in internal machine representation.

Note: If the last command has been 'A', 'B' or 'C' then the input of RETURN only will display the next byte or address value.

3.2 Memory Dump

With this function the user can display a contiguous portion of memory specified by a begin and end address.
The dump allows the display of either byte or address values.

The range of the dump can be specified in 2 different ways:

DBXXXX,Z or DBXXXX-YYYY

where XXXX is the begin address
YYYY is the end address
Z is the number of values to be dumped

Dump of byte values:

```
DBXXXX,Zcr
'BXXXX : VV VV VV VV ... (up to 16 byte values/row)'
```

Dump of address values:

```
DAXXXX,Zcr
'AXXXX : VVVV VVVV VVVV ... (up to 8 address values/row)'
```

3.3 Snapshot

This function allows the display ('snapshot') of

- CPU registers/flags
- up to 16/32 contiguous address/byte memory locations
- independently specified address or byte memory locations

before the execution of a specified instruction.
Furthermore,the taking of the snapshot can be conditional in the sense that a specified memory location has to have a specified value at the time of the snapshot.

The taking of the snapshot and the display of the results do not influence the operation of the system.

The snapshot remains activated until terminated as described in Section 4. The function is executed each time the specified snapshot instruction is executed.

Note: Because of the snapshot method of inserting a trap instruction it is not possible to activate a snapshot in a ROM section of the program.

Input format:

S XXXX parameter list cr

where 'S' is the debug command and XXXX the address of the first byte of the snapshot instruction.

Note: If the snapshot address is not set to the first byte of an instruction, an erroneous program action may occur.

The parameter list can be any combination of the following inputs:

- R
  This parameter requests the display of all CPU registers and flags.

- D
  This parameter specifies a contiguous region in memory whose contents at the time of the snapshot is to be displayed. The format of the parameter is identical to the debug function 'dump' described in Section 3.2.
  The region to be dumped is limited to 10H/20H address/byte values.

- A or B
  These parameters have the form AXXXX or BXXXX and cause the display of address/byte value at location XXXX respectively. A maximum of 5 locations, either address or byte, can be specified.

- M
  This parameter has the format

  MXXXX-YY

  where XXXX is a byte location in memory and YY the specified contents.
  If this condition is set and the contents of XXXX is not YY, then the snapshot is not taken.
  This feature allows to check the state of the system at a specified iteration of a loop (M parameter is the index variable) or to monitor the occurence of a specified input to the system.

Output format:

The snapshot function has no outputs until the specified instruction is executed.
If no parameter was specified in the 'S' command then only the standard output occurs:

'SNAPSHOT AT XXXX : YY YY YY'

where XXXX is the snapshot address and
YY YY YY is the snapshot instruction (max 3 bytes).

The output initiated by the parameters is self-explanatory.

If during the output processing of the snapshot results another snapshot occurs, it will be suppressed. The number of suppressed snapshots is typed.

3.4 Message Simulation

This function allows the generation of real-time messages and is useful to trigger processes for which the initiating module is not integrated or the initiating input is not available.

After the input of the debug command 'MS', the function prompts the user for inputs in order to construct the message to be simulated.

The input format is shown by the following example.

Input format:

```
>MScr
  'RMN:'17cr 'SMN:'8cr 'MN:'10cr 'ML:'6cr
  'DATA BYTES:'10cr ':'20cr
  'MSG SENT
>'
```

The above commands would cause the sending of a message with the receiving module number 17H, the sending module number 8, message number 10H, message length 6 with two data bytes 10H and 20H.

The minimum message length is 4 (length of MCB).

Maximum message length is 24H: MCB and 20H data bytes.

The input of data bytes is prompted automatically depending on the specified message length. The message is sent as soon as the input is completed.

After the prompt symbol reapears the same message can be repeated with the input of the RETURN key only.

In the case of an illegal input during the input of the message, the same (wrong) input is prompted again.
If the specified message cannot be sent by the system because the receiving module is not activated, then 'MSG NOT SEND' is typed.

3.5 Message Extraction

This function allows the interception of any specified real-time message at any time.
The user can specify any combination of RMN,SMN,MN and ML.
If the specified message is to be processed by the receiving module,the message (including data bytes) is typed on the CRT.

The input/output formats are illustrated with the following example.

Input format:

```
>MXcr
  'RMN:'17cr 'SMN:'10cr 'MN:'cr 'ML:'cr
```

An input request answered with the input of RETURN only has the meaning of 'not specified'. In the above example all messages from module 10H to module 17H are extracted because MN and ML are not specified and can take on every value.

Output format:

```
'RMN: 17 SMN: 10 MN: 11 ML: 16  DATA: VV VV .....
                                      VV VV .....
                                      (up to 16 bytes/line)
```

3.6 Periodic Inspect/Dump

This feature repeats the output of an 'inspect' or 'dump' function at maximum system speed and allows therefore a continuous 'look' into memory in order to observe changes of variables, status bits etc.

These functions are initiated by the same commands described in Section 3.1 and 3.2 with the difference that they are preceded by the letter 'P' (for periodic).

Examples:

```
PDBXXXX,ZZcr

PDAXXXXcr
```

Note: The number of values to be dumped by the periodic dump is restricted to 10H/20H address/byte values.
A larger range is cut down to this maximum length by the debug function.

'Maximum system speed' is ususally determined by the speed of the output media.

3.7 Hard Copy

This function allows the user to obtain hard copies of the debugging results.

If a hard copy is requested, the outputs of

- memory inspection
- memory dump
- snapshot
- message extraction
- periodic functions

are directed to the line printer.

The user may switch at any time between the CRT and line printer output.

Default output device is the CRT.

Input format:

```
>Hcr
```

The first 'H' command switches output to the line printer, the next 'H' command switches back to CRT and so forth.

If output is switched to the line printer, the print head is positioned at the top of a new page.

In case the line printer is not connected properly, an asynchronous message 'LINE PRINTER NOT CONNECTED' is typed on the CRT and the 'H' command is disregarded.

4 Termination of a Debug Function

A selected debug function terminates in 3 different ways:

a) The function terminates itself after the inspection of a memory location.In this case the system shows the prompt character '>'.

b) If a debug function requires more than 1 input or is executed periodically,it can be terminated with the input of 'Control-I' at any time. The system responds with the prompt character.

c) At any time within the debug mode the CRT interruption 'Control-I' can be used to disconnect the debug module from the CRT. In this case, however, not only the currently activated debug function is stopped,the debug mode is terminated as well.
The only system reaction is the positioning of the cursor to the beginning of a new line.

5 Termination of Debug Mode

The debug mode can be terminated in 2 different ways:

a) Use of the debug command 'T'.
This input can be made when the prompt character is displayed and a regular command is expected.
The system responds with 'DEBUG TERMINATED' and leaves the debug mode.

b) At any time the procedure decribed in 4 c) can be used.

6 Summary of Debug Commands

Debug Functions:

A - inspect address value

B - inspect byte value

C - change address or byte value

D - memory dump

H - hardcopy

MS - message simulation

MX - message extraction

P - inspect/dump in periodic mode

S - snapshot

T - terminate debug mode

Other Debug Inputs:

Control-D - initiate debug mode

Control-T - terminate debug function

Control-I - CRT interruption (terminates debug mode)

APPENDIX D

PROGRAM DESCRIPTION OF DEBUG MODULE

TABLE OF CONTENTS

1 General

1.1 Introduction

The debug module (module identification: DB) allows the debugging of all modules in all single board computers under real-time conditions.
This means that the use of a debug function does not influence the operation of the system.

Since each single board computer has its own kernel of the debug module, it is possible that several users debug different program parts at the same time.

There are no special requirements for the integration of DB except that at least one CRT module has to be integrated, too.

The CRT and the line printer are the I/O media of the debug module. Real-time messages are used to communicate with the respective modules.

1.2 Abbreviations and Conventions

All numbers in this segment of text are decimal except as indicated otherwise.

SBC - single board computer

MCB - message control block
RMN - receiving module number
SMN - sending module number
MN - message number
ML - message length

CS - CRT module, module number in SBC 1/2/3 : 06/14/22
EX - executive, module number in SBC 1/2/3 : 00/08/16
DB - debug module, module number in SBC 1/2/3 : 07/15/23
LP - line printer module, module number in SBC 1/2/3 : 05/13/21

1.3 Facilities of the Debug Mocule

The debug module offers the following debug functions under real-time conditions:

- inspection and change of byte and address values
- memory dump of byte and address values
- snapshot with
  - . register dump
  - . dump of specified single memory locations
  - . dump of one contiguous region of memory
  - . specified condition
- message simulation
- message extraction
- periodic inspection/dump
- choice between CRT and line printer output

1.4 Module Urganization

Because of its size the DB module has been split up into 9 parts in order to ease program development and program maintenance under the ISIS-II operating system.
The single parts of DB are program modules in the sense of PL/M-80 and ISIS-II.

Each of the 9 parts has a seperate source file while the object code is kept in one object library which represents the object code of the DB module.

In the following the parts of DB are described briefly:

DBMUD1: This part is the entry for real-time messages in DB. It also includes the PUBLIC definitions of all DB data and the START procedure.

DBMUD2: This part processes all input messages from the CRT module. It

- performs the initialization of the debug mode
- relays debug requests to responsible debug modules in other SBC's if necessary
- distributes debug inputs and messages to the respective debug function.

The program and data organization of DBMUD2 allows easy to implement additions to the debug functions.

DBMOD3: Memory inspection and change

DBMOD4: Memory dump

DBMOD5: Snapshot

DBMOD6: Message simulation and extraction

DBMOD7: System test

DBMOD8: DB utility routines

DBMOD9: DB utility routines

2 Input Real-time Messages

DB obtains all inputs from the CRT module.
The format of the debug inputs is described in the user's manual for the debug functions.

In this section the format and contents of all input messages is described. The respective process can be found in Section 3.7.

2.1 Start

```
RMN   :  DB
SMN   :  EX
MN    :  00
ML    :  04
```

This message informs DB that the system has been started.

2.2 Input Text from CRT

```
RMN   :  DB
SMN   :  CS
MN    :  20
ML    :  depends on length of input text
DATA0-
DATAn :  ASCII input characters
```

This message transmits input text from the CRT to DB.

2.3 Terminate I/O

```
RMN   :  DB
SMN   :  CS
MN    :  22
ML    :  04
```

This message is received when the CRT interruption input was made during activated debug mode.

2.4 Start Debug

```
RMN   :  DB
SMN   :  CS
MN    :  23
ML    :  04
```

This message is received when a 'Control-D' input was legally made at the CRT keyboard: a user wishes to activate the debug mode.

2.5 External Start

RMN : DB
SMN : DB module in other SBC
MN : 24
ML : 05
DATA0 : number of CS module for debug inputs and outputs

This message is received when the host SBC is to be debugged from another SBC.
DATA0 contains the module number of a CS module to communicate with.

2.6 Terminate Debug Function

RMN : DB
SMN : CS
MN : 25
ML : 04

This message informs DB that a 'Control-T' input was made at the CRT keyboard.
With this input the user wishes to terminate the currently selected debug function.

2.7 Output Acknowledge

RMN : DB
SMN : CS or LP
MN : 26
ML : 05
DATA0 : tellback code

This message is sent by CS or LP in order to indicate that the output requested by DB has been completed.
The returned acknowledge code is the code sent to CS or LP with the 'output with acknowledge' message.

2.8 Message extraction

RMN : DB
SMN : EX
MN : 27
ML : depends on length of extracted message
DATA -
DATAn : MCB and data bytes of extracted message

This message is received when message extraction is activated and the executive has intercepted a message specified for extraction.

2.9 Line Printer not Connected

RMN : DB
SMN : LP
MN : 28
ML : 04

This message informs DB that the line printer is not connected to the system.

3 Processing

This section describes the function of the debug facilities and the process of the received real-time messages.

3.1 Inspect and Change

The process of this function is contained in DBMOD3 and consists of 3 procedures:

DBINSPECT, DBCHANGE and TYPEINSP.

DBINSPECT is called when

a) a A,B,PA or PB command has been entered at the CRT

b) RETURN only was entered at the keyboard and the last activated function was A,B or C

c) an acknowledge message was received and the last command was PA or PB.

In case a) the address of the memory location is decoded from the input message.
If the input was not legal, the error indication '?' is sent to the CRT module.
If a periodic output was requested, the cursor is positioned at the beginning of the input line and TYPEINSP is called with acknowledge request.
Otherwise TYPEINSP is called with the output followed by the prompt character.

DBCHANGE is called for the execution of the 'C' command.
If the previous input has not been A, B or C, then the change is not legal and an error is indicated.
Otherwise the new contents is stored and TYPEINSP is called to type address and new contents.
DBCHANGE terminates with the issue of the prompt character.

TYPEINSP performs the output of the current 'inspect and change' address and the respective contents (byte or address).
Depending on the state of the flag DBTELLBACK this is done with or without acknowledge request.

3.2 Memory Dump

The memory dump is performed by DBMOD4 and consists of 5 procedures:

DBDUMP, DUMPDECODE, INITDUMP, DUMPLINE and DUMP.

DBDUMP is the entry procedure. From here the respective routines to set up, execute or repeat the dump (in case of periodic dump) are called.

DUMPDECODE determines the mode, byte or address, and the range of the dump.
In case of an illegal input, a logical 'FALSE' is returned, otherwise a 'TRUE' is returned.
DUMPDECODE is called from DBDUMP and DBSNAP (see 3.3).

DUMP is the procedure which actually performs the dump from DUMPBEG to DUMPEND.
DUMP is called from DBDUMP and SNAPSHOT.
The procedure keeps calling DUMPLINE until the flag DUMPDONE becomes 'TRUE'.

INITDUMP sets up the format of the output line. It packs the address of the first value typed on a line and determines the number of values typed on a line depending on the mode of the dump.

DUMPLINE packs the contents of the memory locations starting at the current address. Up to ENDLINE values are packed for a line. DUMPLINE uses the utility procedures PACKBYTE and PACKADR to convert the contents of memory locations into ASCII codes and pack into DB output buffer.

3.3 Snapshot

The snapshot function constitutes DBMOD5 and consists 4 procedures.

Basically the snapshot is enabled by replacing the operation code of the instruction at the snapshot address by the trap instruction RST4. The actual snapshot is taken when this instruction is executed and an 'interrupt' occurs.

The display of the snapshot data is performed with a priority call scheduled during the process after the execution of the trap instruction.

The snapshot remains activated until it is terminated by a command. At this time the original instruction is restored at the snapshot address.

If the system is stopped with an activated snapshot and started again without new loading, the snapshot address still contains the RST4 instruction.
The start procedure of DB checks for an activated snapshot before system stop and restores the original instruction if necessary.

DBSNAP initiates the snapshot and performs the following functions:

- reset all relevant snapshot data

- process CRT input

  . 'R' - set REGFL

  . 'M' - set SNCONDADR to condition address and
  SNCONDSAVE to the condition
  set CONDFL

  . 'D' - determine address and type of dump
  (procedure DUMPDECODE is used)
  set DUMPFL

  . 'A'/'B' - determine address and type of the requested variables and store in SNVAR
  update SNVARX (index to SNVAR)

  In case that an illegal input is encountered during the process of the CRT input, the display of '?' and the prompt character is initiated and program control is returned.

- determination of the length of the snapshot instruction

- insertion of the instruction in the execution table SNEXTBL (see structure of SNEXTBL below)

- insertion of the address of the next instruction into SNEXTBL (= snapshot address + instruction length)

- insertion of RST4 at the snapshot address

Organization and function of the snapshot execution table:

```
          **********
0         *   E1   *        POP H
          **********
1         *   F1   *        POP PSW
          **********
2         *   C1   *        POP B
          **********
3         *   D1   *        POP D
          **********
4         *   E1   *        POP H
          **********
5         *   33   *        INX SP
          **********
6         *   33   *        INX SP
          **********
7         *   FB   *        EI
          **********
8         *   00   *             replaced
          **********
9         *   00   *             snapshot
          **********
10        *   00   *             instruction
          **********
11        *   C3   *        JMP
          **********
12        *        *             address of instruction
          **********
13        *        *             after snapshot instruction
          **********
```

At the end of the snapshot 'interrupt' process, program control is transferred to the first byte of the execution table where

- the stack is updated
- interrupts are enabled
- the replaced instruction is executed
- program control is transferred back to the 'interrupted' program.

SNAPINT is the procedure which services the snapshot interrupt. The return from this procedure takes place as described above.

If the specified condition is not, met control is returned.

If there is already a snapshot in progress, i.e. SNAPFL is 'TRUE', a counter is incremented in order to count multiple occurrences of the same snapshot.

If none of the above applies, SNAPFL is set and the current value of the requested parameters are saved if the respective flag (SNVARX,DUMPFL or REGFL) is set.

After saving of the specified values and scheduling a priority call for procedure SNAPSHOT,SNEXTBL is executed.

The procedure SNAPSHOT outputs the snapshot data to the CRT.
This is done with 'synchronous output with acknowledge' message to the CS module.
The acknowledge mode prevents a possible overflow of the CRT output buffer.
The returned acknowledge code is used to distribute the process sequentially through SNAPSHOT.
Since this process is straightforward, it is not described in detail.
At the end of SNAPSHOT the flag SNAPFL is reset to allow the output of the next snapshot.

Procedure SNAPTERM has the task of terminating the snapshot function. It is called when the 'terminate debug function' message is received or at system start when the system was stopped before with an activated snapshot.
SNAPTERM restores the original instruction at the snapshot address.

3.4 Message Simulation/Extraction

DBMOD6 consists of the functions message simulation and message extraction. These functions are handled in the procedures DBMSX, DBMSXINP and DBEXTRACT.

DBMSX is the entry procedure for DBMOD6 and is called from the message entry of the DB module.
If flag DBTELLBACK is 'TRUE' the procedure DBEXTRACT is called, otherwise DBMSXINP is called.

DBMSXINP processes the CRT input message for simulation and extraction .
In order to obtain all necessary inputs, the procedure initiates a dialog with the operator at the CRT.
All input/output is performed with real-time messages to and from the CS module.

In both cases, simulation and extraction, the dialog yields a message control block.
Illegal inputs are rejected and prompted again.
The MCB is stored in DEBUGMCB, a table located in the system data area to allow access by the executive in order to check for messages to be extracted.
After completion of the MCB input, the process splits.

Message simulation:

The dialog continues until the input of data bytes as specified in the MCB is completed. After this, the constructed message is sent.
If the message could not, be sent the output of 'MSG NOT SENT' is issued.
If the sending of the message was successful 'MSG SENT' is typed.
The function terminates with the sending of the prompt character.

If the last function has been 'message simulation' and the input of RETURN only is received, the sending of the same message is repeated.

Message extraction:

The extraction is initialized with the sending of an extraction message to the executive.

DBEXTRACT processes extraction messages from the executive. The CRT output is done using output with acknowledge messages. The process is straightforward and is not described here.

3.5 Hardcopy

The 'H' command is processed in the procedure DBHARDCOPY which is part of DBMOD8.

If flag HARDCOPY is 'TRUE' then it is set to 'FALSE'.

If flag HARDCOPY is 'FALSE' an 'initialize line printer' message is sent to the LP module and HARDCOPY is set to 'TRUE'.

Before leaving the procedure, a new debug command is prompted.

In case the line printer is not connected to the system, a 'line printer not connected' message will be received by DB. Upon receipt of this message the flag HARDCOPY is set to 'FALSE'.

HARDCOPY controls the output media in procedure DBSYNCPRH. Depending on the state of HARDCOPY, the receiving module for the 'synchronous output' message is LP or CS.

3.6 Periodic Inspect/Dump

The inspect and dump functions can also be executed as periodic functions, i.e. the output is repeated with the highest frequency the system allows.

The output of the functions is sent to LP or CS with acknowledge request. Upon receipt of the acknowledge message the same requested output is repeated with the current contents of the specified locations.

3.7 Real-time Messages

In this section the process of the real-time messages is described.

3.7.1 Start

This message is processed in procedure DBSTART.
The following operations are performed:

- if a snapshot has been activated before the last system stop (DBFUNCTION = 4), then call SNAPTERM (restore snapshot instruction)
- clear DB variable data
- clear snapshot data
- set module status (existing and activated)
- determine start address of procedures DBINPUT and SNAPSHOT
- enter priority call for SNAPSHOT.

The DB module is then ready to be used.

3.7.2 Input Text from CRT

This message is processed by the procedure DBINPUT which constitutes DBMOD2.
The execution of DBINPUT is controlled by a case statement and the variable CRTINPUT.

CRTINPUT = 0:

Input from CRT must be the initiation of the debug mode. 'CPTR:' is typed on the CRT, CRTINPUT is set to 1 and a 'CRT input request' message is sent to CS.

CRTINPUT = 1:

The input of the number of SBC which the user wishes to debug is expected now.

If the input is not legal, then '?-CPTR:' is typed and the input request repeated.

If the number is 0 or its own SBC number, the procedure DBREQ is called to initiate the debug mode.
In DBREQ the control variable is set to 2 and a prompt for a debug command is issued.
Otherwise the number of the debug module in the target SBC is computed.
If the respective module is not activated, 'DEBUG MODULE NOT ACTIVATED' is typed.
If the module is active, a 'start debug external' message is sent to the external debug module.
The module number of the own CS module is transmitted in the first data byte.

The control variable is reset to 0.

CRTINPUT = 2:

The input is the selection of a debug function.
DBINPUT decodes the function identifier and calls the respective procedure to process the input.

The variable DBFUNCTION contains an index which determines the selected debug function.

CRTINPUT is set to 3.

CRTINPUT = 3:

This input is caused by the selected debug function itself.
Program control is routed to the process of the currently activated function.

3.7.3 Terminate I/O

This message informs the DB module that the connection to the CRT has to be terminated (input of 'Control-I' at the CRT).

The input control variable is reset to 0 in order to accept a new initialization of the debug mode.
Any pending debug function and the debug mode are terminated.

3.7.4 Start Debug

This message informs DB that a 'Control-D' input has been made at the CRT.

This input is processed in procedure DBINPUT with control variable CRTINPUT = 0 (see Section 3.7.2).

3.7.5 External Start Debug

This message is sent by a debug module in an other SBC and informs DB that there is an external debug request from an other SBC.

The message is processed in procedure DBEXTREQ.
After saving the module number of the connected CRT in the other SBC in DBCRT procedure, DBREQ is called to further process the debug request.

3.7.6 Terminate Debug Function

This message is received after a 'Control-T' input at the CRT and it serves the purpose of terminating the currently activated debug function.

The process takes place in procedure DBTERMPER.
If no debug function is activated (i.e. the prompt character is displayed), an error message is sent to CS.
If the activated function is 'message extraction', a message is sent to EX to stop the extraction.
An activated snapshot is terminated with a call of SNAPTERM.

After resetting of all relevant flags, the prompt character is displayed.

3.7.7 Output Acknowledge

This message is received when an output with acknowledge initiated by DB is completed.

The returned acknowledge code is saved in DBTBCODE and the flag DBTELLBACK is set to 'TRUE'.
Then DBINPUT is called which passes program control to the activated function controlled by CRTINPUT and DBFUNCTION.

3.7.8 Extraction Message

This message transmits an extracted message from EX to the 'message extraction' function in DB.

If an extraction output is currently active (DBEXTRACT = 'TRUE') a counter is incremented and DBEXTRACT is not called. Otherwise DBEXTRACT is called in order to initiate the typing of the extracted message.

3.7.9 Line Printer not Connected

The process of this message consists of resetting the flag HARDCOPY to 'FALSE', i.e. output is not directed to the line printer because it is not in operation.

4 Output Real-time Messages

In this section the format of the real-time messages sent by DB is described.

4.1 Synchronous Output

```
RMN    :  CS or LP
SMN    :  DB
MN     :  11
ML     :  depends on length of output text
DATA0  :  0 - no roll screen/cr,lf after output
          1 - roll screen/cr,lf after output
DATA1-
DATAn  :  text to be typed/printed
```

This message is used to type text on the input line of the CRT or on the line printer.

4.2 CRT Input Request

```
RMN    :  CS
SMN    :  DB
MN     :  12
ML     :  05
DATA0  :  0 - no roll screen after input
          1 - roll screen after input
```

This message requests a keyboard input from the CRT.

4.3 Terminate CRT Input

```
RMN    :  CS
SMN    :  DB
MN     :  14
ML     :  04
```

This message disconnects DB from the CRT.

4.4 Activate Debug

```
RMN    :  CS
SMN    :  DB
MN     :  15
ML     :  04
```

This message is used to establish a connection between DB and the CRT at which the debug request was made.

4.5 New Line

```
RMN   : CS or LP
SMN   : DB
MN    : 16
ML    : 04
```

This message rolls the CRT screen once and places the cursor at the beginning of the input line or positions the print head of the line printer at the beginning of the next line.

4.6 Begin of Line

```
RMN   : CS or LP
SMN   : DB
MN    : 17
ML    : 04
```

This message positions the CRT cursor at the beginning of the input line or the print head of the line printer at the beginning of the next line.

4.7 Synchronous Output with Output Acknowledge

```
RMN   : CS or LP
SMN   : DB
MN    : 18
ML    : depends on length of output text
DATA0 : acknowledge code
DATA1 : 0 - no roll screen/cr,lf after output
        1 - roll screen/cr,lf after output
DATA2-
DATAn : text to be typed/printed
```

This message is used to type text on the input line of the CRT or to print on the line printer and requests an acknowledge message when the output is completed.

4.8 External Start Debug

```
RMN   : any external debug module
SMN   : DB
MN    : 24
ML    : 05
DATA0 : module number of connected CRT module
```

This message informs an external debug module about a debug request from the CRT module determined by DATA0.

4.9 Start Message Extraction

```
RMN  : Ex
SMN  : DB
MN   : 10
ML   : 04
```

This message informs the executive that a message extraction is started. The message to be extracted is described by the MCB currently in DEBUGMCB.

4.10 Stop Message Extraction

```
RMN  : Ex
SMN  : DB
MN   : 11
ML   : 04
```

This message causes the termination of message extraction by the executive.

4.11 Initialize Line Printer

```
RMN  : LP
SMN  : DB
MN   : 13
ML   : 04
```

This message is sent when the output of DB is to be switched to the line printer.

APPENDIX E

PROGRAM DESCRIPTION OF CRT MODULE

TABLE OF CONTENTS

1 General

1.1 Introduction

The CRT module (module identification: CS) is the driver for the CRT under the real-time operating system.
All I/O is interrupt driven in order to meet the real-time requirements.
The CS module 'connects' all other modules in the system to the CRT, i.e. the interface with the operator is controlled by the communicating module. The CS module merely passes data in and out for other modules.Except for some line editing functions CS does not have its own interface.
The interface between CS and a user module takes place with the use of real-time messages.

The module may be shared by several SBC's. In this case only the message entry procedure and the CS data have to be local to the SBC's.
Depending on the number of the host SBC the module numbers of the CS module vary.

CS is designed to communicate with a DATAMEDIA Elite 2500 CRT and keyboard.

1.2 Abbreviations and Conventions

All numbers in this segment of text are decimal except as indicated otherwise.

SBC - single board computer

MCB - message control block
RMN - receiving module number
SMN - sending module number
MN - message number
ML - message length

CS - CRT module, module number in SBC 1/2/3 : 06/14/22
EX - executive, module number in SBC 1/2/3 : 00/08/16
DB - debug module, module number in SBC 1/2/3 : 07/15/23

INACTMOD - module number of module currently connected to CRT
(INput ACTive MODule)

input line - last line on CRT
line for asynchronous outputs - line 7 on CRT
line for synchronous outputs - last line on CRT (=input line)

1.3 Facilities of the CS Module

1.3.1 Input

The CS module allows any module in the system to obtain data from an operator at the keyboard.

In order to ease keyboard input, CS provides two line editing functions:

- delete last input character
- delete entire input.

CRT input for a module is initiated with an 'activate CRT input' msg to CS. This msg is acknowledged with a msg sent back. The first data byte of this msg determines whether the request is acknowledged or not.
The case that CRT control is not granted can only occur if several modules attempt to obtain inputs at the same time.
Once the connection to the CRT is established, the user module has to send an 'input request' msg to CS. CS then sends the next input terminated with the RETURN key to the requesting module.

All keyboard inputs are displayed on the input line.

Upon completion of the inputs, the connected module has to release the CRT with a special msg.

1.3.2 Output

There are 2 distinct types of CRT output:

- asynchronous and
- synchronous.

Asynchronous outputs are typed on line 7 of the CRT.
The text typed here is sent to CS using the 'asynchronous output' msg. This msg may be sent by any module at any time and serves the purpose to report errors, special internal conditions etc.
The output of this message is accompanied by the bell.
For the time of the output all input is blocked and a 'f' is displayed at the position of the next input. After typing the asynchronous message the 'f' is erased with the next input.

Synchronous outputs are typed on the input line and allow the module currently in control of the CRT to interact with the operator.
'Synchronous output' messages are accepted from the connected module only.

There are two messages which control the screen itself. They cause the cursor to be positioned at

- the beginning of a new line (roll screen once)
- the beginning of the current line.

2 Inputs

The CS module receives inputs from the keyboard of the CRT and from other modules with real-time messages.

2.1 Input from CRT

Keyboard inputs are transmitted in form of ASCII characters. See DATAMEDIA Elite 2500 manual for details.

Apart from the normal character set there are inputs which have special meanings. They are listed below:

'Control-I'	-	CRT interruption This input forces the termination of any existing connection between a module and the CRT.
'Control-D'	-	enter debug mode
'Control-T'	-	terminate debug function
'Back Space'	-	delete last character
'Rub Out'	-	delete entire input
'CR'	-	termination of current input

The interpretation of these inputs is described in Section 3.2.

2.2 Input Real-time Messages

In this section the format of the incoming real-time messages is given. The process of these messages is described in Section 3.4.

2.2.1 Start

```
RMN  :  CS
SMN  :  EX
MN   :  00
ML   :  04
```

This msg is sent by the executive when the system is started. Upon receipt the CS module initializes itself (see Section 3.4.1).

2.2.2 Asynchronous Output

```
RMN    :  CS
SMN    :  any module
MN     :  10
ML     :  depends on length of text
DATA0-
DATAn  :  text to be typed
```

This message is received when a module is initiating an asynchronous output (see Section 3.4.2).

2.2.3 Synchronous Output

```
RMN    :  CS
SMN    :  module currently connected to CRT (INACTMOD)
MN     :  11
ML     :  depends on length of text
DATA0  :  0 - no roll screen after output
          1 - roll screen after output
DATA1-
DATAn  :  text to be typed
```

The module currently connected to the CRT transmits text to be typed on the input line with this message (see Section 3.4.3).

.2.4 CRT Input Request

```
RMN    :  CS
SMN    :  INACTMOD
MN     :  12
ML     :  05
DATA0  :  0 - no roll after next input
          1 - roll screen once after next input
```

This message is sent by the module currently connected to the CRT to request an input from the keyboard (see Section 3.4.4).

2.2.5 Activate CRT Input

```
RMN    :  CS
SMN    :  module requesting connection to the CRT
MN     :  13
ML     :  04
```

With this message each module can establish a connection to the CRT (see Section 3.4.5).

2.2.6 Terminate CRT Input

RMN : CS
SMN : INACTMOD
MN : 14
ML : 04

This message is used by the module currently connected to the CRT to release the CRT (see Section 3.4.6).

2.2.7 Activate Debug

RMN : CS
SMN : any debug module
MN : 15
ML : 04

With this message the user selected debug module connects itself to the CRT (see Section.4.7).

2.2.8 New Line

RMN : CS
SMN : INACTMOD
MN : 16
ML : 04

This message causes the roll of the screen by 1 line and the positioning of the cursor at the beginning of a new line (see Section 3.4.8).

2.2.9 Begin of Line

RMN : CS
SMN : INACTMOD
MN : 17
ML : 04

This message positions the cursor at the beginning of the current input line (see Section 3.4.9).

2.2.10 Synchronous Output with Acknowledge

```
RMN    :  CS
SMN    :  INACTMOD
MN     :  18
ML     :  depends on length of text
DATA0  :  acknowledge code
DATA1  :  0 - no roll screen after output
          1 - roll screen once after output
DATA2-
DATAn  :  text to be typed
```

This message initiates a synchronous output on the input line and requests an acknowledge message when this output is completed (see Section 3.4.10).

3 Processing

In this section the process of CRT input/output and real-time message is described.

3.1 USART Interface

The operation of the USART is controlled by a mode control word and a command control word.

After a hardware reset, the first output has to be a mode control word. After having received the mode control word the USART accepts any number of command control words.
If it is required to output a mode control word without a hardware reset, the USART can be reset with a special control word.

The formats of the control words and the input/output ports are given below.

Mode Control Word:

```
bit 7, 6 :  01  -  1 STOP BIT
 "  5, 4 :  00  -  NO PARITY
 "  3, 2 :  10  -  7 BIT CHARACTER LENGTH
 "  1, 0 :  10  -  BAUD RATE FACTOR 16
```

Command Control Word:

```
bit 7 :  0
 "  6 :  0  -  NO RESET
               RESET USART
 "  5 :  1  -  REQUEST TO SEND
 "  4 :  0
 "  3 :  0
 "  2 :  1  -  RECEIVE ENABLE
 "  1 :  1  -  DATA TERMINAL READY
 "  0 :  1  -  TRANSMIT ENABLE
```

Input/Output port for status information is 0EFH, for data 0EEH.

3.2 Input from CRT

All CRT input is interrupt driven.

Upon system start, the USART (CRT input) is initialized and the respective interrupt is activated.

If a key on the keyboard is hit, an interrupt occurs. The interrupt procedure decodes the interrupt and schedules a call of the the priority procedure CSINPUT.
After the call of CSINPUT by the executive, an input instruction is executed to obtain the input character.

Without any filtering the following inputs are considered in the the indicated sequence and processed:

- 'Control-I' : CRT interruption
- 'Control-D' : initialize debug mode
- 'Control-T' : terminate debug function

'Control-I':

This input forces the termination of the current connection between a module and the CRT.
If a module is currently connected to the CRT (INACTMOD <> FFH), then a 'terminate I/O' msg is sent to this module.
If no module is connected, no actions take place.

'Control-D':

This input initiates the debug mode.
If debugging from this CRT is already activated, the error indication '?' is written into the output buffer.
If the request is legal, any current connection between a module and the CRT is terminated with the 'terminate I/O' message and a 'start debug' message is sent to the debug module in the same SBC.

'Control-T':

This input causes the sending of a 'terminate debug function msg to the connected debug module. If not in debug mode, a '?' is written into the output buffer.

All other keyboard inputs are considered only if

- no output is active
- an 'input request' msg has been received.

'Back Space':

This input causes the deletion of the last input character on the screen and in the input buffer.
The cursor is moved back and the last input is erased.
An attempt to move the cursor past the beginning of the last input request is not accepted and answered with the bell.

'Rub Out':

With this key the entire input of the last input request can be erased. The cursor is moved back to the first position of the last input request and the rest of the line is deleted so that a correct input line can be generated.

'RETURN':

This key terminates the current input.
The input collected starting at the position of the cursor at the last input request is sent to the connected module with a 'transfer input text' message.
Depending on the specification in the last 'input request' message, the screen is rolled once or not.

All other inputs:

The input ASCII character is checked for legality. Legal codes have to be within the range of 1FH < input code < 5AH and within the length of the input line.
If the input is illegal a '?' is written into the output buffer, otherwise the input is stored, the input buffer pointers are updated and the input character is written into the output buffer.
Before returning program control to the executive, the output of the current output buffer contents is initiated.

3.2 Output to CRT

All CRT output is interrupt driven.

At system start the following codes are written into the output buffer:

- clear screen
- set roll mode
- output '*** SYSTEM START' and bell on the line for asynchronous outputs
- position cursor at the beginning of input line.

Then the output is activated by executing an output instruction for the first character in the output buffer.
Upon completion of the output, an interrupt occurs which is processed by the interrupt routine. After determining the source of the interrupt, a priority call of CSOUTPUT is scheduled.

If there are no more characters for output, the output buffer pointers are reset to 0 and an acknowledge message is sent to INACTMOD if requested.
Otherwise the next character from the output buffer is transferred to the USART and the buffer pointers are updated.

The output buffer CRTOUT has 2 pointers:

- OUTPTR (next character for output)
- OUTX (next character to fill).

CRTOUT is empty if OUTPTR >= OUTX. In this case both pointers are reset to 0.

3.4 Real-time Messages

.4.1 Start

This message informs CS that the system has been started. CS performs the following initialization steps:

- reset USART if no system reset has been performed
- clear all local variable data
- determine begin addresses of priority procedures CSINPUT and CSOUTPUT and store in CSINADR and CSOUTADR respectively
- set module status:
  - module exists
  - module activated
  - module authorized to suspend/activate other modules
- reset pointers and flags
- enter priority calls of CSINPUT and CSOUTPUT
- initialize USART
- pack 'start output' into output buffer
- initialize output to USART

3.4.2 Asynchronous Output

With this messages text to be typed on the line for asynchronous outputs (line 7 on CRT) is sent to CS.

For the duration of the output, an uparrow ('↑') is displayed at the current position of the cursor on the input line.

After completion of the asynchronous output, the cursor is returned to its last position on the input line. The next keyboard input erases the '↑'.

The following output is initiated:

- '↑'
- move cursor to the beginning of line 7
- clear line
- pack bell
- pack '*** '
- pack output text (from message)
- pack code to return cursor to the last input position.

After packing the output buffer, the output is initiated with the call of STARTOUT.

3.4.3 Synchronous Output

This message causes the typing of the transferred text on the input line.

If SMN of this message is different from INACTMOD, then the system routine ILLEGALMSG is called and the message rejected.

If the message is legal, the text of the message starting at data byte 1 is moved into the output buffer.

If data byte 0 is TRUE, then the roll of the screen after the output is requested and the respective characters are entered into the output buffer.
If no roll is requested, the position of the cursor on the input line is incremented by the number of characters in the transferred text.

The output to the CRT is initiated with a call of procedure STARTOUT.

3.4.4 CRT Input Request

This message requests an input from the CRT keyboard.

If SMN is not INACTMOD, then the system routine ILLEGALMSG is called and the request is rejected.

The cursor position and the input buffer pointer are saved in order to mark the beginning of this input for line editing.

The variable ROLLSCREEN is set according to data byte 0 of the msg.

3.4.5 Activate CRT Input

This message is sent by a module which wishes to establish a connection to the CRT.

This request is answered with an acknowledge message in which the first data byte indicates whether the request is acknowledged or not.

SMN of this message is saved in the variable INACTMOD.

3.4.6 Terminate CRT Input

This message terminates the connection between the sending module (SMN = INACTMOD) and the CRT.
CS resets INACTMOD to FFH.

3.4.7 Activate Debug

This message connects a debug module to the CRT. No acknowledge message is sent in this case since this request is always granted.
INACTMOD is set to the module number of the debug module.

3.4.8 New Line

This message is internally converted to a 'synchronous output' message and the procedure for synchronous output is called.

3.4.9 Beginning of Line

See Section 3.4.8

3.4.10 Synchronous Output with Acknowledge

The process of this message is essentially identical to the process of a synchronous message without acknowledge (see Section 3.4.3) with the exception that the sending module requests a message to indicate the completion of the initiated output.

This feature can be used to avoid overwriting of text and buffer overflows.

The first data byte contains the acknowledge code which has to be returned with an acknowledge message after completion of the output.

4 Outputs

The CS module outputs data to the CRT and sends real-time messages to other modules in the system.

4.1 Output to CRT

Outputs to the CRT are transmitted in form of ASCII characters. See DATAMEDIA Elite 2500 manual for details.

The only ASCII exception is the use of the XY addressing feature of the DATAMEDIA.
This mode allows the arbitrary positioning of the cursor. The XY mode is initiated with '0CH'. The following two characters (coded according to the manual) are considered to be a location on the CRT. The first determines the position in a row and the second character determines the row.

If CS detects an illegal input character, then this is indicated with the output of '?'.

4.2 Real-time Messages

In this section the format of the real-time msg sent by CS is described.

4.2.1 Transfer Input Text

```
RMN    :  INACTMOD
SMN    :  CS
MN     :  20
ML     :  depends on length of input text
DATA0-
DATAn  :  input text
```

This message is used to send input text to the requesting module.

4.2.2 Activate CRT Input Acknowledge

```
RMN    :  INACTMOD
SMN    :  CS
MN     :  21
ML     :  05
DATA0  :  TRUE  - request acknowledged
          FALSE - otherwise
```

This message informs the module which wants to activate CRT input whether the request is granted or not.

4.2.3 Terminate I/O

RMN : INACTMOD
SMN : CS
MN : 22
ML : 04

This message is sent to the module currently in control of the CRT when the CRT interruption input (Control-I) is detected.

4.2.4 Start Debug

RMN : DB
SMN : CS
MN : 23
ML : 04

This msg informs the debug module that the legal input 'Control-D' has been generated.

4.2.5 Terminate Debug Function

RMN : DB
SMN : CS
MN : 25
ML : 04

This message informs the debug module that the input of 'Control-I' has been detected.

4.2.5 Acknowledge

RMN : INACTMOD
SMN : CS
MN : 26
ML : 05
DATA0 : acknowledge code

This message indicates the completion of an output with acknowledge request.
The acknowledge code is identical to the code received with the 'synchronous output with acknowledge' message.

APPENDIX F

PROGRAM DESCRIPTION OF LINE PRINTER MODULE

TABLE OF CONTENTS

1 General

.1 Introduction

The line printer module (LP) is the driver for the CENTRONIX 101 line printer under the real-time operating system.

All output is interrupt driven in order to meet the real-time requirements.

The code of the LP module may be shared by several single board computers if more than 1 line printer is to be connected to the system.

All modules in the system can initiate printouts on the line printer by transferring text to LP with a real-time message.

1.2 Abbreviations and Conventions

All numbers in this segment of text are decimal except when indicated otherwise.

SBC - single board computer

MCB - message control block
RMN - receiving module number
SMN - sending module number
MN - message number
ML - message length

CS - CRT module, module number in SBC 1/2/3 : 06/14/22
EX - executive, module number in SBC 1/2/3 : 00/08/16
DB - debug module, module number in SBC 1/2/3 : 07/15/23
LP - line printer module, module number in SBC 1/2/3 : 05/13/21

1.3 Facilities of the LP Module

In order to allow the switching of output between CRT and line printer, the usage of the line printer is very similar to the CRT output, i.e. is controlled by the same messages. (Exception: the 'beginning of line' message is interpreted as 'new line' message.)

LP allows to print any text sent to the module with the 'print' message. The text is transferred in the data bytes of the message and has to be ASCII coded.
This text can be printed with or without the generation of a acknowledge msg to indicate the completion of the printout.

Apart from the 'print' and 'print with acknowledge' message there are 3 form control message:

- new page
- new line
- beginning of line (interpreted as new line).

LP accepts an 'initialize line printer' message. Upon receipt of this message the proper connection of the line printer is checked.
If the check is positive, the print head is positioned at the beginning of a new page.
Otherwise a warning message is typed on the CRT and a 'line printer not connected' message is sent to the module which requested the check.

2 Inputs

The LP module receives inputs from

- line printer (status inputs)
- other modules (real-time messages).

2.1 Status Inputs from Line Printer

Two status bits from the line printer are made available for LP on input port E6:

- bit 4 : 'SELECT'
- bit 3 : 'LPT BUSY'.

2.2 Input Real-time Messages

In this section the format of the incoming real-time message is given.
The process of these messages is described in Section 3.4.

2.2.1 Start

```
RMN   : LP
SMN   : Ex
MN    : 00
ML    : 04
```

This message is sent by the executive when the system has been started.
Upon receipt the LP module initializes itself (see Section 3.4.1).

2.2.2 Print

```
RMN   : LP
SMN   : any module
MN    : 11
ML    : depends on length of text to be printed
DATA0 : 0 - no new line after output
        1 - new line after output
DATA1-
DATAn : ASCII characters to be printed
```

The text transferred with this message is to be printed on the line printer (see Section 3.4.2).

2.2.3 New Page

```
RMN   : LP
SMN   : any module
MN    : 12
ML    : 04
```

This message requests the positioning of the print head at the beginning of the first line on a new page (see Section 3.4.3).

2.2.4 Initialize Line Printer

```
RMN   : LP
SMN   : any module
MN    : 13
ML    : 04
```

This message causes LP to check the status of the line printer (see Section 3.4.4).

2.2.5 New Line

RMN : LP
SMN : any module
MN : 16
ML : 04

This message positions the print head at the beginning of a new line (see Section 3.4.5).

2.2.6 Beginning of Line

RMN : LP
SMN : any module
MN : 17
ML : 04

This message is processed as being a 'new line' msg (see Section 2.2.5).

2.2.7 Print with Acknowledge

RMN : LP
SMN : any module
MN : 18
ML : depends on length of text
DATA0 : acknowledge code
DATA1 : 0 - no new line after print
1 - new line after print
DATA2-
DATAn : ASCII characters to be printed

The transferred text of this message is to be printed and upon completion of the printing an acknowledge message with DATA0 has to be returned to the sending module (see Section 3.4.6).

3 Processing

.1 Line Printer Interface

3.2 Line Printer Status Input

Upon receipt of the 'initialize line printer' message, LP reads the 'SELECT' status bit of the line printer.
If the line printer is connected ('SELECT' bit = 'FALSE') the procedure LPNEWPAGE is called.
If the line printer is not connected properly:

- an 'asynchronous output' msg is sent to CS to type a warning '*** LINE PRINTER NOT CONNECTED'
- a 'line printer not connected' msg is sent to the module which sent the 'initialize line printer' msg.

3.3 Line Printer Output

All line printer output is interrupt driven.

The output characters are written into the output buffer LPOUTBUF. LPOUTBUF is controlled by two pointers:

- LPOUTX (next to fill)
- LPOUTPTR (next to print).

LPOUTBUF is empty if LPOUTPTR >= LPOUTX. In this case both pointers are reset to 0.

An overflow occurs if LPOUTX > last index of LPOUTBUF. This condition causes a call of the system monitor and output characters are rejected until there is space in LPOUTBUF.

When all text is transferred from the 'print' message into the output buffer, or an overflow occurs, the output is started by calling the output procedure.

When the output of a character is completed, an interrupt is generated and the priority call of the procedure LPOUTPUT is scheduled by the interrupt procedure.
After LPOUTPUT called by the executive, the next character is is put on the output line or the printout is terminated (if LPOUTBUF is empty).
In the latter case the flag LPTELLBACK is checked and an acknowledge message is generated if necessary.

3.4 Real-time Messages

3.4.1 Start

This message informs LP that the system has been started.
LP performs the following operations:

- clear all local variables
- determine start address of procedure LPOUTPUT
- enter priority call for LPOUTPUT
  (to be called if a line printer output interrupt occurs)
- set module status
  - module exists
  - module activated.

3.4.2 Print

This message is processed in procedure LPPRINT.

The data bytes of the msg are moved into the output buffer until all characters are moved or an overflow occurs.
In case of an overflow, the move of characters is terminated and the output is initiated.
When the remaining space is sufficient and all characters are moved, then DATA0 of the msg controls the insertion of code for a 'carriage return' and a 'line feed'. Then the output is initiated.

3.4.3 New Page

The code for 'form feed' is packed into the output buffer and the output is activated.

3.4.4 Initialize Line Printer

LP reads the 'SELECT' status bit and determines whether output is possible or not.
If no printing is possible, i.e. the printer is not connected properly, the typing of a warning message at the CRT is generated with an 'asynchronous output' msg to CS.
Otherwise, the 'new page' procedure is executed
(see Section 3.4.3).

3.4.5 New Line

The code for 'carriage return' and 'line feed' is packed into the output buffer and the output is activated.

3.4.6 Print with Acknowledge

This message is processed by the procedure LPPRINT (see 3.4.2) after the acknowledge code has been removed from the msg.
The code and the sending module is saved.
After completion of the output an acknowledge message is sent back together with the requested acknowledge code.

4 Outputs

The LP module outputs data to the line printer and sends messages to other modules in the system.

4.1 Output to the Line Printer

Outputs to the line printer are transmitted in form of ASCII characters.
Letters are printed as capital letters only.

See the available CENTRONIX 101 interface specification.

Output port is port E4H.

4.2 Real-time Messages

4.2.1 Asynchronous Output

RMN : CS
SMN : LP
MN : 10
ML : 30
DATA0-
DATA25: text 'LINE PRINTER NOT CONNECTED'

4.2.2 Acknowledge

RMN : module which sent 'print with acknowledge' message
SMN : LP
MN : 26
ML : 05
DATA0 : acknowledge code transferred with 'print with acknowledge' message

4.2.3 Line Printer not Connected

RMN : module which sent 'initialize line printer' message
SMN : LP
MN : 28
ML : 04

This message informs the requesting module that the line printer is not connected properly to the system.

APPENDIX G

PROGRAM LISTINGS

EXECUTIVE

```
                $EJECT
                $ INCLUDE(:F1:MSGENT.SRC)
            =   /*
            =   ***************************************************************************
            =   ***************************************************************************
            =   **
            =       DECLARATIONS OF MSG ENTRY PROCEDURES FOR ALL POSSIBLE
            =       MODULES IN COMPUTER
            =       IF A MODULE IS ADDED THE RESPECTIVE MSGENT PROCEDURE
            =       HAS TO BE MADE 'EXTERNAL' AND THE 'RETURN' HAS TO
            =       BE TAKEN OUT
            =   **
            =   ***************************************************************************
            =   */
  119   1   =   MSGENT01:PROCEDURE;
  120   2   =           RETURN;
  121   2   =           END;
  122   1   =   MSGENT02:PROCEDURE;
  123   2   =           RETURN;
  124   2   =           END;
  125   1   =   MSGENT03:PROCEDURE;
  126   2   =           RETURN;
  127   2   =           END;
  128   1   =   MSGENT04:PROCEDURE;
  129   2   =           RETURN;
  130   2   =           END;
  131   1   =   MSGENT05:PROCEDURE EXTERNAL;
  132   2   =           END;
  133   1   =   MSGENT06:PROCEDURE EXTERNAL;
  134   2   =           END;
  135   1   =   MSGENT07:PROCEDURE EXTERNAL;
  136   2   =           END;
```

```
                $ EJECT

137   1         EXMSGENT:   PROCEDURE EXTERNAL;
138   2                   END EXMSGENT;

139   1         EXSTART:    PROCEDURE EXTERNAL;
140   2                   END EXSTART;

141   1         EXMSGEXTR:  PROCEDURE EXTERNAL;
142   2                   END EXMSGEXTR;

                /*
                ****************************************************************
                ****************************************************************
                **
                            EXECUTIVE
                **
                ****************************************************************
                */
143   1         EXEC:       PROCEDURE PUBLIC;

144   2         EXEC0:      CALL EXSTART;
                            /* INITIALIZE */
                /*
                ****************************************************************
                **
                    PROCESS PRIORITY TASKS OF MODULES
                */
145   2         EXEC1:      DO WHILE PRIORSCHEDULE <> 0;
                            /* DO AS LONG AS PRIORITY CALLS ARE SCHEDULED */
146   3                     XB2 = 1;
147   3                     DO XB1 = 1 TO MAXPRIOR;
```

```
                            /* FIND HIGHEST PRIORITY SCHEDULED */
148   4                    XB3 = SCR(PRIORSCHEDULE,XB1);
149   4                    IF CARRY THEN
150   4                      DO;
151   5                      PRIORSCHEDULE = PRIORSCHEDULE XOR XB2;
                                 /* RESET PRIORITY BIT */
152   5                      XA1 = PRIORLIST(XB1-1);
153   5                      CALL XA1;
                                 /* CALL PRIORITY PROCEDURE */
154   5                      GO TO EXEC11;
155   5                      END;
156   4                    XB2 = ROL(XB2,1);
157   4                    END;
158   3      EXEC11:         ;
159   3                  END;
                  /*
                  ***************************************************************************
                  **
                      PROCESS PENDING REAL TIME MESSAGES
                  */
160   2       EXEC2:       IF NUMMSG = 0 THEN GO TO EXEC21;
                           /* NO MSG TO PROCESS */
162   2                  IF MSGOUT = LASTMSG THEN LASTMSG,MSGOUT = 0;
                           /* NEXT MSG AT TOP OF MSGBUFFER */
164   2                  IF MSGEXTRACTION THEN CALL EXMSGEXTR;
                           /* MSG EXTRACTION IS ACTIVATED */
166   2                  XA1 = MSGOUT;
167   2                  MSGOUT = XA1 + MSGBUFFER(XA1+ML);
                           /* COMPUTE BEGIN OF NEXT MSG */
168   2                  NUMMSG = NUMMSG - 1;
                           /* UPDATE NUMBER OF MSG */
169   2                  ADRMSG = .MSGBUFFER(XA1);
```

```
170   2                 ADRMSGDATA = ADRMSG + 4;
                            /* 'SET' MSG INTO WORKAREA */
171   2                 XB1 = MSG(RMN) - FIRSTMN;
                          /* COMPUTE INTERNAL MODULE NUMBER */
172   2                 IF XB1 >= MAXMOD
                        THEN XB2 = SENDEXT;
                             /* MODULE NOT IN OWN COMPUTER
                                CALL EXTERNAL COMMUNICATION */
174   2                   ELSE DO CASE XB1;
175   3                     CALL EXMSGENT;
176   3                     CALL MSGENT01;
177   3                     CALL MSGENT02;
178   3                     CALL MSGENT03;
179   3                     CALL MSGENT04;
180   3                     CALL MSGENT05;
181   3                     CALL MSGENT06;
182   3                     CALL MSGENT07;
183   3                     END;
184   2                 GO TO EXEC1;
                          /* CHECK PRIORITY CALLS AGAIN */
185   2         EXEC21:    IF EXTMSG <> CPTR THEN GOTO EXEC3;
                          /* NO EXTERNAL MSG TO PROCESS */
187   2                 CALL RECEXT;
                          /* TRANSFER MSG INTO OWN MSG BUFFER */
188   2                 GO TO EXEC1;
                          /* CHECK PRIORITY CALLS AGAIN */
                  /*
                  ************************************************************************
                  **
                      CHECK FOR PERIODIC CALLS OF MODULES
                  */
189   2         EXEC3:        ;
```

```
190   2                 IF NUMPER = 0 THEN GO TO EXEC4;
                            /* NO PERIODIC CALLS ACTIVE */
192   2                 XB1 = NUMPER - 1;
                            /* SET LIMIT FOR 1 SEARCH CYCLE */
193   2                 DO XB2 = 0 TO XB1;
                            /* CHECK ALL ACTIVATED PERIODICS */
194   3                   PERX = PERXTBL(XB2);
195   3                   IF TIMECHK(.PERLIST(PERX).PERTIME(0)) THEN
                              /* CALL TIME REACHED */
196   3                     DO;
197   4                     CALL SETPERTIME(PERX);
                                /* SET NEXT CALL TIME */
198   4                     XA1 = PERLIST(PERX).PERADR;
199   4                     CALL XA1;
                                /* CALL PERIODIC */
200   4                     GO TO EXEC1;
                                /* CHECK PRIORITY CALLS */
201   4                     END;
202   3                   END;
                  /*
                  ***************************************************************************
                  **
                      BACKGROUND TASKS
                  */
203   2           EXEC4:      ;
204   2                   GO TO EXEC1;
205   2                   RETURN;
206   2                   END EXEC;

                  /*
                  ***************************************************************************
                  ***************************************************************************
```

```
                **
                        MAIN MODULE PROGRAM ENTRY
                **
                */
 207    1       SYSSTART: SAVESTACKPTR = STACKPTR;
                             /* SAVE STACKPOINTER FOR RESTART */
 208    1                 RESTART = FALSE;
                             /* START WITH SYSTEM RESET */
 209    1                 CALL EXEC;
 210    1             END;
```


MODULE INFORMATION:

```
     CODE AREA SIZE      = 01CDH     461D
     VARIABLE AREA SIZE  = 02EBH     747D
     MAXIMUM STACK SIZE  = 0004H       4D
     705 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

MESSAGE HANDLER OF EXECUTIVE

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EXMSG
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:EXMSG.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            EXMSG:   DO;
                /*
                            UPDATE: 1
                */
                $ NOLIST
                /*
                ****************************************************************************
                ****************************************************************************
                **
                **                      EXEC MSG HANDLER
                **
                ****************************************************************************
                */

 117   1        EXEC:       PROCEDURE EXTERNAL;
 118   2                  END EXEC;

 119   1        MSGENT07:   PROCEDURE EXTERNAL;
 120   2                  END;

                /*
                ****************************************************************************
                ****************************************************************************
                **
```

```
                    EXECUTIVE PERIODIC
                ;*
                ;*************************************************************************
                ;*/
121   1         EXPER:      PROCEDURE;

122   2                 IF EXPERFL = 0 THEN
                            /* FIRST CALL, GET ADDRESS */
123   2                   DO;
124   3                   EXPERADR = PROCADR;
125   3                   EXPERFL = 1;
126   3                   RETURN;
127   3                   END;
128   2                 CALL SYSMON(CODESAVE,0,1);
                            /* SEND MSG OVERFLOW TO SYSTEM MONITOR */
129   2                 CALL PERSUSP(EXPERX);
130   2                 EXPERX = 0FFH;
131   2                 RETURN;
132   2                 END EXPER;
                /*
                ;*************************************************************************
                ;*************************************************************************
                ;*
                                                                        EXMSGEXTR

                    EXTRACT CURRENT MSG IF REQUIRED
                ;*
                ;*************************************************************************
                ;*/
133   1         EXMSGEXTR: PROCEDURE PUBLIC;

134   2                 ADRMSG = .MSGBUFFER(MSGOUT);
```

```
135   2             DO XB3 = 0 TO 3;
                       /* CHECK MCB OF CURRENT MSG */
136   3               IF (DEBUGMCB(XB3) <> 0FFH) AND
                        (DEBUGMCB(XB3) <> MSG(XB3)) THEN RETURN;
                          /* NO EXTRACTION REQUIRED */
138   3               END;
139   2             EXTRMSG(ML) = MSG(ML) + 4;
140   2             ADRMSG = .EXTRMSG(0);
141   2             ADRMSGDATA = .MSGBUFFER(MSGOUT);
142   2             CALL MSGENT07;
143   2             RETURN;
144   2             END;
          /*
          ************************************************************************
          ************************************************************************
          **
                                                                    EXMSGENT

              PROCESS REAL TIME MESSAGE
          **
          ************************************************************************
          */
145   1       EXMSGENT:   PROCEDURE PUBLIC;
146   2               IF MSG(MN) > 9 THEN
147   2                 DO;
148   3                 XB3 = MSG(MN) - 10;
149   3                 IF XB3 > 1 THEN RETURN;
151   3                 DO CASE XB3;
152   4                   MSGEXTRACTION = TRUE;
153   4                   MSGEXTRACTION = FALSE;
154   4                   END;
155   3                 END;
```

```
156   2             RETURN;
157   2             END EXMSGENT;
                    /*
                    ***************************************************************************
                    ***************************************************************************
                    **
                                                                          SYSRESTART

                          SYSTEM RESTART
                    **
                    ***************************************************************************
                    */
158   1       SYSRESTART: PROCEDURE;

159   2       DCL       (START BASED A1) BYTE;

160   2                 IF RSTFL = 0 THEN
161   2                   DO;
                              /* FIRST CALL, FIND ADDRESS OF SYSRESTART */
162   3                   RSTADR = PROCADR;
163   3                   RSTFL = 1;
164   3                   RETURN;
165   3                   END;

166   2                 IF CPTR <> 1 THEN
167   2                   DO;
168   3                   IF CPTR = 2
                            THEN A1 = .START2;
170   3                     ELSE A1 = .START3;
171   3                   DO WHILE START <> 0;
                              /* WAIT UNTIL SBC 1 STARTED */
172   4                     END;
```

```
 173   3                 START = CPTR;
 174   3                 END;

 175   2               RESTART = TRUE;
                           /* START WITHOUT RESET */

 176   2               STACKPTR = SAVESTACKPTR;
                           /* RESET STACK POINTER */

 177   2               CALL EXEC;

 178   2               END SYSRESTART;
                 /*
                 ***************************************************************************
                 ***************************************************************************
                 **
                                                                              START

                     INITIALIZE COMPUTER
                 **
                 ***************************************************************************
                 */
 179   1         EXSTART:    PROCEDURE PUBLIC;

 180   2               DISABLE;
 181   2               CALL CLEARDATA(.EXCLEARBEG,.EXCLEAREND);
                           /* CLEAR EXEC DATA */
 182   2               DO B1 = 0 TO LAST(PERLIST);
                           /* INITIALIZE PERIODIC LIST */
 183   3                 PERLIST(B1).FREE = TRUE;
 184   3                 END;
 185   2               ADRSTAT = .MODSTATUS(0) + FIRSTMN;
```

```
                        /* LOCATE OVERLAY FOR HOST COMPUTER
                           IN MODSTATUS TABLE */
186   2                 MODSTAT(0) = 00001011B;
                        /* SET EXEC SLOT */
187   2                 DO B1 = 1 TO LAST(MODSTAT);
                        /* INITIALIZE MODSTAT WITH 0
                           I.E.  NO MODULE EXISTS
                           PACK INTERNAL START MSG INTO MSGBUFFER */
188   3                   MODSTAT(B1) = 0;
189   3                   SYSMN0(0) = B1 + FIRSTMN;
190   3                   CALL PACKMCB(.SYSMN0(0));
191   3                   END;
192   2                 NUMMSG = MAXMOD - 1;
                        /* SET NUMBER OF MSG IN MSGBUFFER */

193   2                 CALL EXPER;
                        /* OBTAIN ADDRESS OF EXPER */
194   2                 EXPERX = 0FFH;

195   2                 A4 = INTVECTOR;
196   2                 B1 = BU AND 11100000B;
197   2                 OUTPUT(0DAH) = B1 OR 00010010B;    /* ICW1 */
198   2                 OUTPUT(0DBH) = BL;    /* ICW2 */
                        /* INITIATE INTERRUPT CONTROLLER, FULLY NESTED MODE */

199   2                 IF CPTR = 1
                          THEN INTMASK = 01111001B;
201   2                   ELSE INTMASK = 01111101B;
202   2                 OUTPUT(0DBH) = INTMASK;
                        /* INITIALIZE INTERRUPTS
                           INT7 - SYSTEM START
                           INT2 - RTC (SBC 1 ONLY)
```

```
                        INT1 - ENTER MONITOR (FRONT PANEL INT2)
                     */

203   2              OUTPUT(0DFH) = 00110000B;
                         /* COUNTER 0 MODE CONTROL WORD */
204   2              OUTPUT(0DCH) = 1AH;
205   2              OUTPUT(0DCH) = 2;
                         /* LOAD COUNTER 0 (RTC),LEAST SIGNIFICANT BYTE FIRST
                            1 MSEC = 538 X 1860 NSEC <=> 21AH */

206   2              IF CPTR = 1 THEN
207   2                DO;
208   3                CALL CLEARDATA(.BEGABSDATA,.ENDABSDATA);
209   3                IF RESTART
                         THEN DO;
211   4                     START2 = 0;
212   4                     START3 = 0;
213   4                     END;
214   3                   ELSE DO;
215   4                     START2 = 2;
216   4                     START3 = 3;
217   4                     END;
                              /* START SBC2 AND SBC3 */
218   3                END;

219   2              ENABLE;

220   2              RETURN;
221   2              END EXSTART;
222   1      END EXMSG;
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE INT
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:INT.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1             INT:    DO;
                $ NOLIST
                /*
                ****************************************************************
                **
                **                  INTERRUPT PROCESSING
                */
                /*
                ****************************************************************
                */
117   1         INTRESET:   PROCEDURE;

118   2                     OUTPUT(0DAH) = 00100000B;
119   2                     RETURN;
120   2                     END INTRESET;
                /*
                ****************************************************************
                */
121   1         INT0:       PROCEDURE INTERRUPT 6;

                          /* CDC INTERRUPT */

122   2                     IF NOT CDCACTIVE THEN RETURN;
124   2                     DISABLE;
125   2                     CALL CDCADR;
                            /* CALL PROCESS OF INTERRUPT */
```

```
126   2             INTMASK = INTMASK OR 1;
127   2             OUTPUT(0DBH) = INTMASK;
                        /* DISABLE INT0 */
128   2             CALL INTRESET;
129   2             CDCACTIVE = FALSE;
130   2             ENABLE;
131   2             RETURN;
132   2             END INT0;
            /*
            ***************************************************************************
            */
133   1       INT2:       PROCEDURE INTERRUPT 7;

                      /* RTC INTERRUPT */

134   2       DCL   INCRONE BYTE DATA (1);

135   2             DISABLE;

136   2             RTC(3) = RTC(3) + INCRONE;
137   2             RTC(2) = RTC(2) PLUS 0;
138   2             RTC(1) = RTC(1) PLUS 0;
139   2             RTC(0) = RTC(0) PLUS 0;
                        /* UPDATE RTC */

140   2             OUTPUT(0DFH) = 00110000B;
141   2             OUTPUT(0DCH) = 1AH;
142   2             OUTPUT(0DCH) = 2;
                        /* SET CTR 0 TO 1 MSEC IN MODE 0 */

143   2             CALL INTRESET;
144   2             ENABLE;
```

```
145   2                 RETURN;
146   2                 END INT2;
                  /*
                  ***************************************************************************
              -    *
                  */
147   1           INT3:       PROCEDURE INTERRUPT 3;

148   2                 DISABLE;
149   2                 CALL PRIORITYINT(INTTBL(3));
                            /* SCHEDULE PRIORITY CALL */
150   2                 CALL INTRESET;
151   2                 ENABLE;
152   2                 RETURN;
153   2                 END INT3;
                  /*
                  ***************************************************************************
              -    *
                  */
154   1           INT4:       PROCEDURE INTERRUPT 4;

155   2                 DISABLE;
156   2                 CALL PRIORITYINT(INTTBL(4));
                            /* SCHEDULE PRIORITY CALL */
157   2                 CALL INTRESET;
158   2                 ENABLE;
159   2                 RETURN;
160   2                 END INT4;
                  /*
                  ***************************************************************************
                  */
161   1           INT5:       PROCEDURE INTERRUPT 5;
```

This page intentionally blank

```
 162   2                 DISABLE;
 163   2                 CALL PRIORITYINT(INTTBL(5));
                             /* SCHEDULE PRIORITY CALL */
 164   2                 CALL INTRESET;
 165   2                 ENABLE;
 166   2                 RETURN;
 167   2                 END INT5;
                   /*
                   ***************************************************************************
                   */
 168   1         END INT;
```


MODULE INFORMATION:

```
     CODE AREA SIZE     = 00ADH    173D
     VARIABLE AREA SIZE = 0000H      0D
     MAXIMUM STACK SIZE = 000AH     10D
     571 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

PUBLIC DECLARATIONS OF SYSTEM CALLS

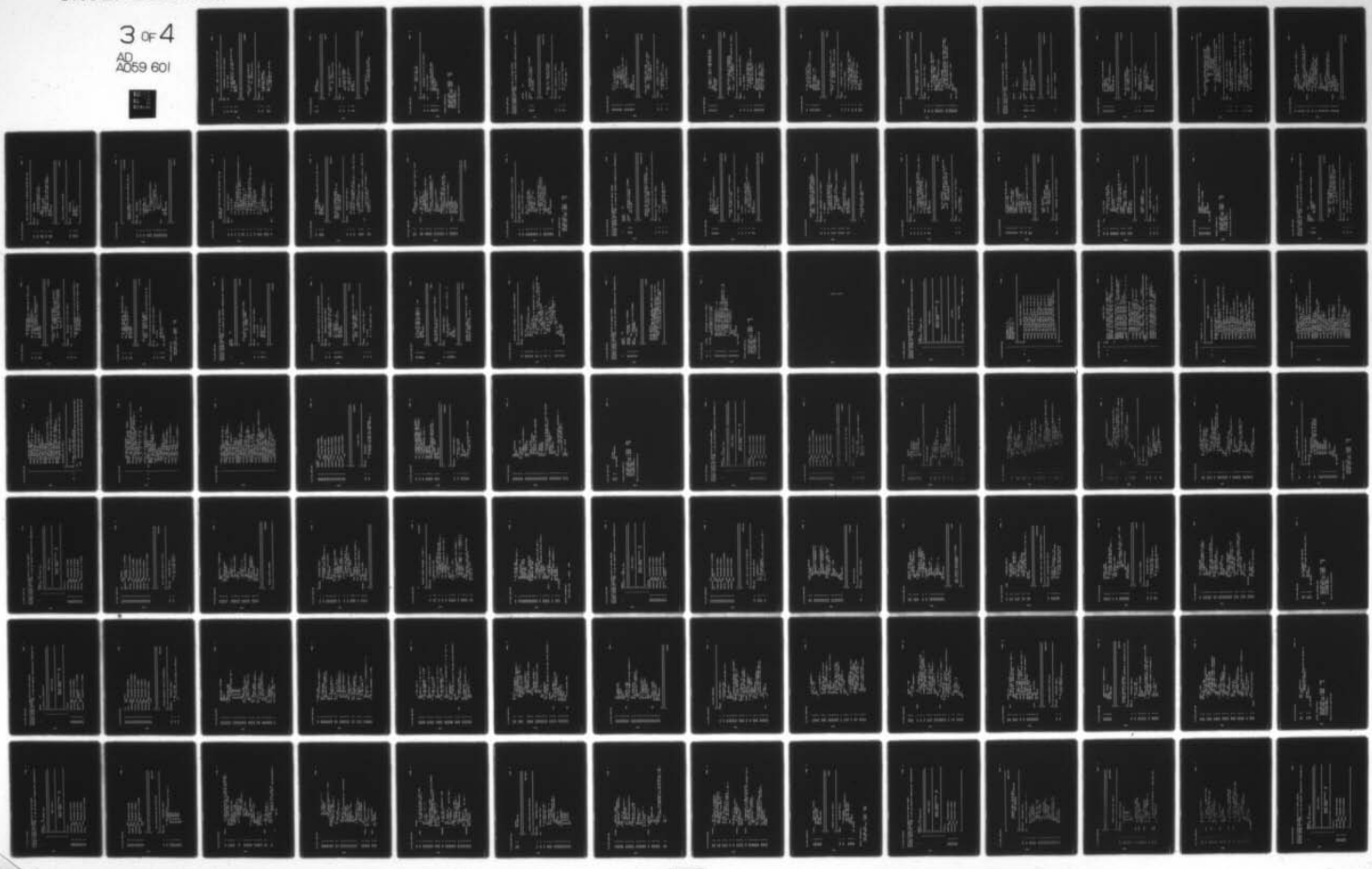
```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB1
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SCPUB1.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            SCPUB1:    DO;
                $ NOLIST

  30    1       SYSMON:   PROCEDURE (P1,P2,P3) EXTERNAL;
  31    2       DCL      (P2,P3) BYTE,P1 ADDRESS;
  32    2                END SYSMON;

                /*
                **************************************************************************
                **************************************************************************
                **
                                                                        CLEARDATA

                    CLEAR MODULES VARIABLE DATA REGION
                        INPUT : P1 - ADDRESS OF FIRST BYTE
                                P2 - ADDRESS OF LAST BYTE
                **
                **************************************************************************
                */
  33    1       CLEARDATA:   PROCEDURE (P1,P2) PUBLIC;

  34    2       DCL      (P1,P2) ADDRESS,(I,J) ADDRESS,
                         (DAT BASED P1) (1) BYTE;

  35    2                I = P2 - P1;
  36    2                DO J = 0 TO I;
```

```
37   3                DAT(J) = 0;
38   3                END;
39   2              RETURN;
40   2              END CLEARDATA;
               /*
               ***************************************************************************
               ***************************************************************************
               **
                                                                          SHFT

                   RETURN BYTE WITH A 1 SHIFTED INTO POSITION P1
                        INPUT  : P1 - BIT POSITION
                        OUTPUT : P1TH POWER OF 2
               **
               ***************************************************************************
               */
41   1       SHFT:       PROCEDURE (P1) BYTE PUBLIC;

42   2       DCL         P1 BYTE;

43   2                   IF P1 = 0 THEN RETURN 1;
45   2                   RETURN ROL(1,P1);
46   2                   END;
               /*
               ***************************************************************************
               ***************************************************************************
               **
                                                                          BIT

                        CHECK A BIT IN A SPECIFIED BYTE
                        INPUT  : P1 - BIT
                                 P2 - BYTE TO CHECK
```

AD-A059 601 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A REAL-TIME OPERATING SYSTEM FOR SINGLE BOARD COMPUTER BASED DI--ETC(U)
JUN 78 W NIEMANN

UNCLASSIFIED NL

3 OF 4
AD
A059 601



```
                        OUTPUT : TRUE IF BIT IS SET, FALSE IF NOT
                    **
                    ************************************************************************
                    */
 47   1             BIT:        PROCEDURE (P1, P2) BYTE PUBLIC;

 48   2             DCL         (P1, P2) BYTE;

 49   2                         IF (P2 AND SHFT(P1)) = 0 THEN RETURN FALSE;
                                 /* SPECIFIED BIT NOT SET */
 51   2                         RETURN TRUE;
 52   2                         END BIT;
                    /*
                    ************************************************************************
                    ************************************************************************
                    **
                                                                          CLEARBIT

                        CLEAR A BIT IN A SPECIFIED BYTE
                            INPUT : P1 - BIT
                                    P2 - ADDRESS OF BYTE
                    **
                    ************************************************************************
                    */
 53   1             CLEARBIT:   PROCEDURE (P1, P2) PUBLIC;

 54   2             DCL         P1 BYTE, P2 ADDRESS,
                                (WORD BASED P2) BYTE;

 55   2                         XB1 = SHFT(P1);
 56   2                         WORD = (WORD AND XB1) XOR XB1;
                                 /* CLEAR BIT */
```

```
 57   2              RETURN;
 58   2              END CLEARBIT;
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        SETBIT

                    SET A BIT IN A SPECIFIED BYTE
                        INPUT : P1 - BIT
                                P2 - ADDRESS OF BYTE
                **
                **********************************************************************
                */
 59   1       SETBIT:     PROCEDURE (P1,P2) PUBLIC;

 60   2       DCL             P1 BYTE,P2 ADDRESS,
                         (WORD BASED P2) BYTE;

 61   2              WORD = WORD OR SHFT(P1);
                         /* SET BIT */
 62   2              RETURN;
 63   2              END SETBIT;
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        LEGAL

                    CHECK P1 AGAINST P2
                        INPUT   : P1,P2 VALUES TO CHECK
                                  P3 ERROR CODE FOR SYSMON
```

```
                        OUTPUT : TRUE IF P1 < P2
                                 FALSE OTHERWISE
                    **
                    ******************************************************************************
                    */
  64    1           LEGAL:      PROCEDURE (P1,P2,P3) BYTE PUBLIC;

  65    2           DCL         (P1,P2,P3) BYTE;

  66    2                       IF P1 < P2 THEN RETURN TRUE;
  68    2                       CALL SYSMON(STACKPTR,0,P3);
  69    2                       RETURN FALSE;
  70    2                       END LEGAL;
  71    1           END SCPUB1;
```


MODULE INFORMATION:

```
     CODE AREA SIZE     = 00D9H    217D
     VARIABLE AREA SIZE = 0014H     20D
     MAXIMUM STACK SIZE = 0006H      6D
     356 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB2
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SCPUB2.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1             SCPUB2:     DO;

                $ NOLIST

 48    1        SYSMON:     PROCEDURE (P1,P2,P3) EXTERNAL;
 49    2        DCL      (P2,P3) BYTE,P1 ADDRESS;
 50    2                  END SYSMON;

                /*
                **************************************************************************
                **************************************************************************
                **
                                                                      SETPERTIME

                    SET NEXT CALL TIME FOR PERIODIC
                        INPUT: P1 - PERIODIC INDEX
                **
                **************************************************************************
                */
 51    1        SETPERTIME: PROCEDURE (P1) PUBLIC;

 52    2        DCL      (P1,I,J,K,CY) BYTE;

 53    2                  XA1 = PERLIST(P1).PERINTADR;
 54    2                  CY = 0;
                        /* CLEAR CARRY */
```

```
 55   2              I = 3;
 56   2              DO J = 0 TO LAST(RTC);
 57   3                K = CY + RTC(I);
 58   3                IF CARRY THEN CY = 1;
 60   3                         ELSE CY = 0;
 61   3                K = K + PERINT(I);
                            /* COMPUTE NEXT CALL TIME */
 62   3                IF CARRY THEN CY = 1;
 64   3                PERLIST(P1).PERTIME(I) = K;
 65   3                I = I - 1;
 66   3                END;
 67   2              RETURN;
 68   2              END SETPERTIME;
          /*
          ***************************************************************************
          ***************************************************************************
          **
                                                                        TIMECHK

              CHECK RTC AGAINST INPUT TIME
                  INPUT  : ADDRESS OF TIME (RTC FORMAT)
                  OUTPUT : TRUE IF INPUT TIME < RTC
                           FALSE OTHERWISE
          **
          ***************************************************************************
          */
 69   1   TIMECHK:    PROCEDURE (P1) BYTE PUBLIC;

 70   2   DCL         P1 ADDRESS, I BYTE,
                     (TIME1 BASED P1) (4) BYTE;

 71   2              DO I = 0 TO LAST(RTC);
```

```
 72  3                IF TIME1(I) < RTC(I) THEN RETURN TRUE;
 74  3                IF TIME1(I) > RTC(I) THEN RETURN FALSE;
 76  3                END;
 77  2              RETURN FALSE;
 78  2              END TIMECHK;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                          PERACT

                ACTIVATE PERIODIC CALL OF A PROCEDURE
                    INPUT  : P1 - ADDRESS OF PROCEDURE
                             P2 - ADDRESS OF TIME INTERVAL(RTC FORMAT)
                    OUTPUT : INDEX OF PERIODIC
            **
            ***************************************************************************
            */
 79  1      PERACT:     PROCEDURE (P1,P2) BYTE PUBLIC;

 80  2      DCL         (P1,P2) ADDRESS;

 81  2              DO XB1 = 0 TO LAST(PERLIST);
                     /* SEARCH FOR FREE SLOT */
 82  3                IF PERLIST(XB1).FREE THEN GO TO PERACT1;
                        /* FREE SLOT FOUND */
 84  3                END;
 85  2              CALL SYSMON(STACKPTR,0,3);
 86  2              RETURN 0FFH;
 87  2      PERACT1: PERLIST(XB1).FREE = FALSE;
 88  2              PERLIST(XB1).PERADR = P1;
 89  2              PERLIST(XB1).PERINTADR = P2;
```

```
 90   2                DO XB2 = 0 TO LAST(RTC);
                           /* SET NEXT CALL TIME */
 91   3                  PERLIST(XB1).PERTIME(XB2) = RTC(XB2);
 92   3                  END;
 93   2                PERXTBL(NUMPER) = XB1;
 94   2                NUMPER = NUMPER + 1;
 95   2                RETURN XB1;
 96   2                END PERACT;
                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                          PERCHG

                    CHANGE THE TIME INTERVAL OF A PERIODIC CALL
                        INPUT: P1 - PERIODIC INDEX
                               P2 - ADDRESS OF NEW TIME INTERVAL
                **
                ***************************************************************************
                */
 97   1       PERCHG:      PROCEDURE (P1,P2) PUBLIC;

 98   2       DCL          P1 BYTE,P2 ADDRESS;

 99   2                    IF NOT LEGAL(P1,MAXPER,4) THEN RETURN;
                               /* ILLEGAL PERIODIC INDEX */
101   2                    PERLIST(P1).PERINTADR = P2;
                           /* SET NEW TIME INTERVAL */
102   2                    CALL SETPERTIME(P1);
                           /* SET NEW CALL TIME */
103   2                    RETURN;
104   2                    END PERCHG;
```

```
                /*
                **************************************************************************
                **************************************************************************
                **
                                                                        PERSUSP

                    SUSPEND THE PERIODIC CALL OF A PROCEDURE
                        INPUT: PERIODIC INDEX
                **
                **************************************************************************
                */
105   1         PERSUSP:    PROCEDURE (P1) PUBLIC;

106   2         DCL         P1 BYTE;

107   2                     IF NOT LEGAL(P1,MAXPER,4) THEN RETURN;
                                /* ILLEGAL PERIODIC INDEX */
109   2                     PERLIST(P1).FREE = TRUE;
110   2                     NUMPER = NUMPER - 1;
111   2                     IF NUMPER = 0 THEN GO TO SUSP1;
113   2                     XB2 = FALSE;
114   2                     DO XB1 = 0 TO NUMPER;
                                /* REMOVE P1 FROM PERXTBL AND COMPACT PERXTBL */
115   3                       IF NOT XB2 AND PERXTBL(XB1) = P1 THEN XB2 = TRUE;
117   3                       IF XB2 THEN PERXTBL(XB1-1) = PERXTBL(XB1);
119   3                       END;
120   2         SUSP1:          PERXTBL(NUMPER) = 0FFH;
121   2                     RETURN;
122   2                     END;
123   1         END SCPUB2;
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB3
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SCPUB3.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            SCPUB3:     DO;
                /*
                            UPDATE: 2
                */
                $NOLIST

  48    1       SYSMON: PROCEDURE (P1,P2,P3) EXTERNAL;
  49    2       DCL     (P2,P3) BYTE,P1 ADDRESS;
  50    2               END SYSMON;

  51    1       PERACT: PROCEDURE (P1,P2) BYTE EXTERNAL;
  52    2       DCL     (P1,P2) ADDRESS;
  53    2               END PERACT;

                /*
                ****************************************************************
                ****************************************************************
                **
                                                              MSGOVERFLOW

                    HANDLE MSG OVERFLOW
                **
                ****************************************************************
                */
  54    1       MSGOVERFLOW:    PROCEDURE;
```

```
55   2              IF EXPERX <> 0FFH THEN RETURN;
57   2              BU = MSG(MN);
58   2              BL = MSG(SMN);
59   2              CODESAVE = A4;
60   2              EXPERX = PERACT(EXPERADR,0);
61   2              RETURN;
62   2              END MSGOVERFLOW;
          /*
          ***************************************************************************
          ***************************************************************************
          **
                                                                        PACKMCB

              PACK MCB INTO MSGBUFFER
                   (NOT A PUBLIC PROCEDURE)
                   INPUT: ADDRESS OF MCB
          **
          ***************************************************************************
          */
63   1     PACKMCB:     PROCEDURE (A) PUBLIC;

64   2     DCL          A ADDRESS,B BYTE,
                        (MCB BASED A) (4) BYTE;

65   2              DO B = 0 TO LAST(MCB);
66   3                MSGBUFFER(MSGIN) = MCB(B);
67   3                MSGIN = MSGIN + 1;
68   3                END;
69   2              RETURN;
70   2              END PACKMCB;
          /*
          ***************************************************************************
```

```
                ************************************************************************
                **
                                                                        SEND

                    SEND A MESSAGE TO ANOTHER MODULE IN THE SYSTEM
                        INPUT  : ADDRESS OF MSG CONTROL BLOCK(MCB)
                                 MCB : RMN  * RECEIVING MODULE NUMBER
                                       SMN  * SENDING MODULE NUMBER
                                        MN  * MSG NUMBER
                                        ML  * MSG LENGTH
                        OUTPUT : FALSE IF RECEIVING MODULE IS NOT ACTIVATED
                                   OR DOES NOT EXIST OR RMN IS ILLEGAL
                                 TRUE OTHERWISE
                                 ADRMSGDATA IS SET TO ADDRESS OF FIRST DATA BYTE
                **
                ************************************************************************
                */
  71    1       SEND:       PROCEDURE (P1) BYTE PUBLIC;

  72    2       DCL     (P1) ADDRESS, SAVEMSGIN ADDRESS,
                        (MSG BASED P1) (4) BYTE;

  73    2                 IF NOT BIT(1, MODSTATUS(MSG(RMN))) OR MSG(RMN) > MAXSYSMOD
          -   - 1
                        THEN RETURN FALSE;
                            /* RECEIVING MODULE NOT ACTIVATED OR DOES NOT
                               EXIST */
  75    2                 IF MSGOUT = MSGIN AND NUMMSG <> 0 THEN GO TO SEND2;
                            /* MSG BUFFER OVERFLOW */
  77    2                 IF MSGOUT <= MSGIN THEN
  78    2                   DO;
  79    3                   IF (LAST(MSGBUFFER) - MSGIN) >= (MSG(ML) - 1) THEN
```

```
 80   3                    GO TO SEND1;
                             /* MSG FITS INTO REST OF BUFFER */
 81   3                  LASTMSG = MSGIN;
 82   3                  MSGIN = 0;
 83   3                  END;
 84   2                IF (MSGOUT - MSGIN) < MSG(ML) THEN GO TO SEND2;
                          /* NOT ENOUGH SPACE FOR MSG, OVERFLOW */
 86   2       SEND1:     SAVEMSGIN = MSGIN;
 87   2                CALL PACKMCB(P1);
                          /* TRANSFER MCB */
 88   2                MSGIN = SAVEMSGIN + MSG(ML);
                          /* COMPUTE BEGIN FOR NEXT MSG */
 89   2                IF MSGIN > LAST(MSGBUFFER) THEN
                          /* MSGIN IS OUTSIDE MSGBUFFER */
 90   2                  DO;
 91   3                  LASTMSG = MSGIN;
 92   3                  MSGIN = 0;
 93   3                  END;
 94   2                NUMMSG = NUMMSG + 1;
 95   2                ADRMSGDATA = .MSGBUFFER(SAVEMSGIN + 4);
                          /* SET ADDRESS OF FIRST DATA BYTE */
 96   2                RETURN TRUE;
 97   2       SEND2:       ADRMSG = P1;
 98   2                CALL MSGOVERFLOW;
 99   2                RETURN FALSE;
100   2                END SEND;
                  /*
                  **************************************************************************
                  **************************************************************************
                  **
                                                                          SETLOCK
```

```
                    WAIT FOR EXTMSGLOCK TO BECOME UNLOCKED AND LOCK
                **
                ************************************************************************
                */
 101    1       SETLOCK:    PROCEDURE;

 102    2       SETL1:      DO WHILE EXTMSGLOCK <> 0;
                            /* EXT MSG BUFFER LOCKED */
 103    3                 END;
 104    2               EXTMSGLOCK = CPTR;
                            /* SET LOCK */
 105    2               IF EXTMSGLOCK <> CPTR THEN GOTO SETL1;
                            /* ALREADY LOCKED BY OTHER COMPUTER */
 107    2               RETURN;
 108    2               END SETLOCK;
                /*
                ************************************************************************
                ************************************************************************
                **
                                                                          UNLOCK

                    UNLOCK EXTERNAL MSG BUFFER
                **
                ************************************************************************
                */
 109    1       UNLOCK:     PROCEDURE;

 110    2               EXTMSGLOCK = 0;
 111    2               RETURN;
 112    2               END UNLOCK;
                /*
                ************************************************************************
```

```
                ************************************************************************
                **
                                                                    SETEXTMSG

                    COMPUTE NUMBER OF RECEIVING COMPUTER AND SET
                    VARIABLE EXTMSG
                **
                ************************************************************************
                */
  113   1       SETEXTMSG:  PROCEDURE;

  114   2       DCL       (X1,X2,X3) BYTE;

  115   2                 IF NUMEXTMSG = 0
                      THEN EXTMSG = 0;
  117   2                 ELSE DO;
  118   3                   X1 = 8;
  119   3                   X2 = EXTMSGBUFFER(EXTMSGOUT);
                                /* RMN */
  120   3                   DO X3 = 1 TO MAXCPTR;
  121   4                     IF X2 < X1 THEN GOTO SET1;
  123   4                     X1 = X1 + 8;
  124   4                     END;
  125   3       SET1:           EXTMSG = X3;
  126   3                   END;
  127   2                 RETURN;
  128   2                 END SETEXTMSG;
                /*
                ************************************************************************
                ************************************************************************
                **
                                                                    RECEXT
```

```
                    TRANSFER MSG FROM EXTERNAL MSG BUFFER INTO OWN
                    MSG BUFFER
                **
                ************************************************************************
                */
  129   1       RECEXT:     PROCEDURE PUBLIC;

  130   2       DCL       (X1,X2,X3) BYTE;

  131   2                 CALL SETLOCK;
                              /* SET EXTMSGBUFFER LOCK */
  132   2                 IF EXTMSGOUT = EXTLASTMSG THEN EXTLASTMSG,EXTMSGOUT = 0;
                              /* NEXT MSG AT TOP OF BUFFER */
  134   2                 X1 = EXTMSGOUT;
  135   2                 X2 = EXTMSGBUFFER(X1 + ML);
                              /* LENGTH OF MSG */
  136   2                 EXTMSGOUT = X1 + X2;
                              /* COMPUTE BEGIN OF NEXT MSG */
  137   2                 IF NOT SEND (.EXTMSGBUFFER(X1)) THEN GOTO REC1;
                              /* OVERFLOW */
  139   2                 IF X2 = 4 THEN GOTO REC1;
                              /* NO DATA BYTES */
  141   2                 X1 = X1 + 4;
  142   2                 X2 = X2 - 5;
  143   2                 DO X3 = 0 TO X2;
                              /* TRANSFER DATA BYTES */
  144   3                   MSGDATA(X3) = EXTMSGBUFFER(X1);
  145   3                   X1 = X1 + 1;
  146   3                   END;

  147   2       REC1:       NUMEXTMSG = NUMEXTMSG - 1;
```

```
148   2                 CALL SETEXTMSG;
                            /* SET NUMBER OF RECEIVING CPTR OF NEXT MSG */
149   2                 CALL UNLOCK;
150   2                 RETURN;
151   2                 END RECEXT;
                  /*
                  ***************************************************************************
                  ***************************************************************************
                  **
                                                                          SENDEXT

                      SEND MSG TO EXTERNAL MODULE
                      ADRMSG POINTS TO FIRST BYTE OF MSG
                      OUTPUT: TRUE IF MSG SENT
                              FALSE OTHERWISE
                  **
                  ***************************************************************************
                  */
152   1           SENDEXT:    PROCEDURE BYTE PUBLIC;

153   2                 CALL SETLOCK;
                          /* SET EXT MSG BUFFER LOCK */
154   2                 IF EXTMSGIN = EXTMSGOUT AND NUMEXTMSG <> 0 THEN GOTO SEXT2
          -   ;
                            /* OVERFLOW */
156   2                 IF EXTMSGOUT <= EXTMSGIN THEN
157   2                   DO;
158   3                     IF LAST(EXTMSGBUFFER) - EXTMSGIN >= MSG(ML) THEN GOTO SE
          -   XT1;
                              /* MSG FITS INTO REST OF BUFFER */
160   3                   EXTLASTMSG = EXTMSGIN;
161   3                   EXTMSGIN = 0;
```

```
 162   3                 END;
 163   2               IF EXTMSGOUT - EXTMSGIN < MSG(ML) THEN GOTO SEXT2;
                           /* OVERFLOW */

 165   2       SEXT1:      XB2 = MSG(ML) - 1;
 166   2               DO XB3 = 0 TO XB2;
                           /* TRANSFER MSG */
 167   3                 EXTMSGBUFFER(EXTMSGIN) = MSG(XB3);
 168   3                 EXTMSGIN = EXTMSGIN + 1;
 169   3                 END;
 170   2               IF EXTMSGIN > LAST(EXTMSGBUFFER) THEN
                           /* EXTMSGIN OUTSIDE BUFFER */
 171   2                 DO;
 172   3                 EXTLASTMSG = EXTMSGIN;
 173   3                 EXTMSGIN = 0;
 174   3                 END;
 175   2               NUMEXTMSG = NUMEXTMSG + 1;
 176   2               IF NUMEXTMSG = 1 THEN CALL SETEXTMSG;
                           /* SET RECEIVING CPTR OF NEXT MSG */
 178   2               CALL UNLOCK;
                           /* UNLOCK EXT MSG BUFFER */
 179   2               RETURN TRUE;
 180   2       SEXT2:  CALL MSGOVERFLOW;
 181   2               CALL UNLOCK;
 182   2               RETURN FALSE;
 183   2               END SENDEXT;
               /*
               ************************************************************
               ************************************************************
               **
                                                          ILLEGALMSG
```

```
                        PROCESS ILLEGAL MSG RECEIVED BY A MODULE
                    **
                    ************************************************************************
                    */
 184    1           ILLEGALMSG: PROCEDURE PUBLIC;

 185    2           DCL     PRSYNC(4) BYTE DATA (CS,EX,10,40);

 186    2                   XB2 = 16;
 187    2                   DO XB1 = 0 TO LAST(MSG);
 188    3                     CALL CONVASC(MSG(XB1));
 189    3                     ILLMSG(XB2) = BU;
 190    3                     ILLMSG(XB2 + 1) = BL;
 191    3                     XB2 = XB2 + 6;
 192    3                     END;
                                /* CONVERT RMN,SMN,MN,ML */
 193    2                   IF NOT SEND(.PRSYNC(0)) THEN RETURN;
                                /* SEND SYNC PRINT MSG TO CRT */
 195    2                   DO XB1 = 0 TO LAST(ILLMSG);
 196    3                     MSGDATA(XB1) = ILLMSG(XB1);
 197    3                     END;
 198    2                   RETURN;
 199    2                   END ILLEGALMSG;
 200    1           END;


MODULE INFORMATION:

     CODE AREA SIZE      = 03B1H    945D
     VARIABLE AREA SIZE  = 000DH     13D
     MAXIMUM STACK SIZE  = 0006H      6D
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB4
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SCPUB4.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            SCPUB4:     DO;
                $NOLIST

  48    1       SYSMON:     PROCEDURE (P1,P2,P3) EXTERNAL;
  49    2       DCL       (P2,P3) BYTE,P1 ADDRESS;
  50    2                 END SYSMON;

                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                        PRIORITYINT

                    SCHEDULE THE CALL OF A PRIORITY PROCEDURE
                    (PRIORITYINT TO BE CALLED FROM INT ROUTINES ONLY)
                        INPUT: PRIORITY INDEX
                **
                ***************************************************************************
                */
  51    1       PRIORITYINT: PROCEDURE (P1) PUBLIC;

  52    2       DCL       P1 BYTE;

  53    2                 IF P1 >= MAXPRIOR THEN RETURN;
                            /* ILLEGAL PRIORITY */
  55    2                 IF P1 <> 0
```

202

```
                        THEN P1 = ROL(1,P1);
 57   2                 ELSE P1 = 1;
 58   2               PRIORSCHEDULE = PRIORSCHEDULE OR P1;
 59   2               RETURN;
 60   2               END PRIORITYINT;
                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                        PRIORITY

                    SCHEDULE THE CALL OF A PRIORITY PROCEDURE
                        INPUT: PRIORITY INDEX
                **
                ***************************************************************************
                */
 61   1         PRIORITY:   PROCEDURE (P1) PUBLIC;

 62   2         DCL         P1 BYTE;

 63   2                     IF P1 >= MAXPRIOR THEN RETURN;
                                /* ATTEMPT TO SCHEDULE ILLEGAL PRIORITY */
 65   2                     CALL SETBIT(P1,.PRIORSCHEDULE);
                                /* SET SCHEDULE BIT */
 66   2                     RETURN;
 67   2                     END PRIORITY;
                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                        ENTERPRIOR
```

```
                    ENTER NEW PRIORITY CALL IN PRIORITY LIST
                        INPUT  : ADDRESS OF PRIORITY PROCEDURE
                        OUTPUT : PRIORITY INDEX TO PRIORLIST
                    **
                    ************************************************************************
                    */
 68   1             ENTERPRIOR: PROCEDURE (P1) BYTE PUBLIC;

 69   2             DCL      P1 ADDRESS;

 70   2                      DO XB1 = 0 TO LAST(PRIORLIST);
                             /* SEARCH FOR FREE SLOT IN PRIORLIST */
 71   3                        IF PRIORLIST(XB1) = 0 THEN
 72   3                          DO;
 73   4                          PRIORLIST(XB1) = P1;
                                     /* ADDRESS OF PRIORITY PROCEDURE */
 74   4                          RETURN XB1;
 75   4                          END;
 76   3                        END;
 77   2                      CALL SYSMON(STACKPTR,0,2);
                             /* PRIORITY LIST OVERFLOW */
 78   2                      RETURN 0FFH;
 79   2                      END ENTERPRIOR;
                    /*
                    ************************************************************************
                    ************************************************************************
                    **
                                                                        REMPRIOR

                        REMOVE PRIORITY CALL FROM PRIORLIST
                            INPUT: PRIORITY INDEX
                    **
```

```
                /*****************************************************************************
                */
   80   1       REMPRIOR:   PROCEDURE (P1) PUBLIC;

   81   2       DCL     P1 BYTE;

   82   2               IF NOT LEGAL(P1,MAXPRIOR,6) THEN RETURN;
                     /* ATTEMPT TO REMOVE PRIORITY WITH ILLEGAL INDEX */
   84   2               CALL CLEARBIT(P1,.PRIORSCHEDULE);
                       /* CLEAR SCHEDULE BIT IF SET */
   85   2               PRIORLIST(P1) = 0;
                       /* REMOVE PRIORITY ADDRESS */
   86   2               RETURN;
   87   2               END REMPRIOR;
                /*
                *****************************************************************************
                *****************************************************************************
                **
                                                                        SETCDC

                    SET CDC INTERRUPT
                    INPUT : P1 - TIME INTERVAL (LSB = 1.86 MICRO SEC
                        P2 - ADDRESS TO BE CALLED AT CDC INT
                    OUTPUT: FALSE IF CDC ALREADY ACTIVE
                            TRUE OTHERWISE
                **
                *****************************************************************************
                */
   88   1       SETCDC:     PROCEDURE (P1,P2) BYTE PUBLIC;

   89   2       DCL     (P1,P2) ADDRESS,
                        (P1H,P1L) BYTE AT (.P1);
```

```
 90   2              IF CDCACTIVE THEN RETURN FALSE;
 92   2              CDCACTIVE = TRUE;
 93   2              DISABLE;
 94   2              INTMASK = INTMASK XOR 1;
 95   2              OUTPUT(0DBH) = INTMASK;
 96   2              ENABLE;
 97   2              OUTPUT(0DFH) = 01110000B;
                         /* CTR 1 IN MODE 0 */
 98   2              OUTPUT(0DDH) = P1H;
 99   2              OUTPUT(0DDH) = P1L;
                         /* LOAD CTR 1 */
100   2              CDCADR = P2;
                         /* SAVE ADRRESS TO BE CALLED AT CDC INT */
101   2              RETURN TRUE;
102   2              END SETCDC;
                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                          ENTERINT

                        ENTER INTERRUPT
                        INPUT : P1 - INT LEVEL
                                P2 - PRIORITY INDEX
                        OUTPUT: FALSE IF INT ALREADY SET
                                TRUE OTHERWISE
                **
                ***************************************************************************
                */
103   1         ENTERINT:   PROCEDURE (P1,P2) BYTE PUBLIC;
```

```
104   2       DCL     (P1,P2) BYTE;

105   2               IF INTTBL(P1) <> 0FFH THEN
                          /* INT LEVEL ALREADY SET */
106   2                 DO;
107   3                 CALL SYSMON(STACKPTR,0,7);
108   3                 RETURN FALSE;
109   3                 END;
110   2               INTTBL(P1) = P2;
                          /* SET PRIORITY INDEX FOR INT LEVEL */
111   2               INTMASK = INTMASK XOR SHFT(P1);
112   2               DISABLE;
113   2               OUTPUT(0DBH) = INTMASK;
                          /* ENABLE INT LEVEL P1 */
114   2               ENABLE;
115   2               RETURN TRUE;
116   2               END ENTERINT;
                /*
                ***************************************************************
                **
                                                                    REMINT

                        DISABLE INTERRUPT
                        INPUT : INTERRUPT LEVEL
                **
                ***************************************************************
                */
117   1       REMINT:     PROCEDURE (P1) PUBLIC;

118   2       DCL     P1 BYTE;

119   2               INTTBL(P1) = 0FFH;
```

```
 120   2              INTMASK = INTMASK OR SHFT(P1);
 121   2              DISABLE;
 122   2              OUTPUT(0DBH) = INTMASK;
 123   2              ENABLE;
 124   2              RETURN;
 125   2              END REMINT;
 126   1      END SCPUB4;


MODULE INFORMATION:

     CODE AREA SIZE     = 0173H    371D
     VARIABLE AREA SIZE = 000CH     12D
     MAXIMUM STACK SIZE = 0004H      4D
     434 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB5
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SCPUB5.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1             SCPUB5:     DO;
                $NOLIST
 48    1        SEND:           PROCEDURE (P1) ADDRESS EXTERNAL;
 49    2        DCL       P1 ADDRESS;
 50    2                  END SEND;

                /*
                **************************************************************************
                **************************************************************************
                **
                                                                          ACTIVATE

                    ACTIVATE A SUSPENDED MODULE
                        INPUT  : P1 - NUMBER OF MODULE TO BE ACTIVATED
                                 P2 - NUMBER OF CALLING MODULE
                        OUTPUT : FALSE IF CALLING MODULE NOT AUTHORIZED
                                       OR IF MODULE DOES NOT EXIST
                                 TRUE IF O.K.
                **
                **************************************************************************
                */
 51    1        ACTIVATE:   PROCEDURE (P1,P2) BYTE PUBLIC;

 52    2        DCL       (P1,P2) BYTE;

 53    2                  IF NOT BIT(3,MODSTATUS(P2)) OR MODSTATUS(P1) = 0
```

```
                        THEN RETURN FALSE;
                          /* CALLING MODULE NOT AUTHORIZED OR MODULE DOES
                                NOT EXIST */
 55   2                 CALL SETBIT(1,.MODSTATUS(P1));
                            /* SET ACTIVE BIT */
 56   2                 IF NOT SEND(.SYSMN1(0)) THEN RETURN FALSE;
                            /* SEND RESTART MSG TO MODULE */
 58   2                 RETURN TRUE;
 59   2                 END ACTIVATE;
                  /*
                  **************************************************************************
                  **************************************************************************
                  **
                                                                          SUSPEND

                      SUSPEND A CURRENTLY ACTIVE MODULE
                           INPUT  : P1 - NUMBER OF MODULE TO BE SUSPENDED
                                    P2 - NUMBER OF CALLING MODULE
                           OUTPUT : FALSE IF CALLING MODULE NOT AUTHORIZED
                                          OR IF MODULE CANNOT BE SUSPENDED
                                    TRUE IF O.K.
                  **
                  **************************************************************************
                  */
 60   1       SUSPEND:    PROCEDURE (P1,P2) BYTE PUBLIC;

 61   2       DCL         (P1,P2) BYTE;

 62   2                   IF NOT BIT(3,MODSTATUS(P2)) OR NOT BIT(2,MODSTATUS(P1))
                            THEN RETURN FALSE;
                              /* CALLING MODULE NOT AUTHORIZED OR MODULE CANNOT
                                 BE SUSPENDED */
```

```
 64   2              IF NOT SEND(.SYSMN5(0)) THEN RETURN FALSE;
                         /* SEND SUSPEND MSG */
 66   2              CALL CLEARBIT(1,.MODSTATUS(P1));
                         /* CLEAR ACTIVE BIT */
 67   2              RETURN TRUE;
 68   2              END SUSPEND;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                          ACTIVE

                CHECK ACTIVE STATUS OF MODULE
                    INPUT  : MODULE NUMBER
                    OUTPUT : TRUE IF MODULE IS ACTIVATED,FALSE IF NOT
            **
            ***************************************************************************
            */
 69   1      ACTIVE:     PROCEDURE (P1) BYTE PUBLIC;

 70   2      DCL         P1 BYTE;

 71   2              RETURN BIT(1,MODSTATUS(P1));
 72   2              END ACTIVE;
 73   1      END SCPUB5;
```


MODULE INFORMATION:

```
    CODE AREA SIZE     = 00B5H    181D
    VARIABLE AREA SIZE = 0005H      5D
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB6
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SCPUB6.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1             SCPUB6:     DO;
                $NOLIST
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                          UPDSTAT

212                 UPDATE MODULE STATUS IN MODSTATUS TABLE
                        INPUT: P1 - MODULE NUMBER
                               P2 - STATUS
                **
                ****************************************************************************
                */
 30     1       UPDSTAT:    PROCEDURE (P1,P2) PUBLIC;

 31     2       DCL         (P1,P2) BYTE;

 32     2                   MODSTATUS(P1) = P2;
 33     2                   RETURN;
 34     2                   END UPDSTAT;
                /*
                ************************************************************
                ************************************************************
                **
                                                              PROCADR
```

```
                    RETURN ADDRESS OF BEGIN OF CALLING PROCEDURE
                **
                ************************************************************
                */
  35    1       PROCADR:    PROCEDURE ADDRESS PUBLIC;

  36    2       DCL         (A1,A2) ADDRESS,(A2U,A2L) BYTE AT (.A2),
                            (STACKTOP BASED A1) (2) BYTE;

  37    2                   A1 = STACKPTR;
  38    2                   A2U = STACKTOP(0) - 11;
  39    2                   A2L = STACKTOP(1) MINUS 0;
  40    2                   RETURN A2;
  41    2                   END PROCADR;
                /*
                ************************************************************
                ************************************************************
                **
                                                                    CONVASC

                    CONVERT 2 DIGIT HEX NUMBER INTO 2 ASCII CODES
                        INPUT  : NUMBER (BYTE)
                        OUTPUT : ASCII CODES IN A4
                **
                ************************************************************
                */
  42    1       CONVASC:    PROCEDURE (P1) PUBLIC;

  43    2       DCL         P1 BYTE;

  44    2                   BL = (P1 AND 00001111B) + 30H;
```

```
45   2              IF BL > 39H THEN BL = BL + 7;
47   2              BU = ROR((P1 AND 11110000B),4) + 30H;
48   2              IF BU > 39H THEN BU = BU + 7;
50   2              RETURN;
51   2              END CONVASC;
               /*
               ***************************************************************
               ***************************************************************
               **
                                                              DEBLK

                   MOVE POINTER B1 TO 1ST NOM-BLANK IN MSGDATA
                   STARTING AT MSGDATA(B1)
               **
               ***************************************************************
               */
52   1         DEBLK:      PROCEDURE PUBLIC;

53   2              DO WHILE MSGDATA(B1) = ' ' AND B1 <= MSG(ML) - 5;
54   3                B1 = B1 + 1;
55   3                END;
56   2              RETURN;
57   2              END DEBLK;
               /*
               ***************************************************************
               ***************************************************************
               **
                                                              GETNUM

                   CONVERT ASCII CODES IN MSGDATA TO HEX NUMBER
                   STARTIN AT MSGDATA(B1)
                   CONVERTED NUMBER IS LEFT IN A4
```

```
                    RETURN TRUE IF NUMBER IS O.K., FALSE OTHERWISE
                **
                ************************************************************
                */
   58   1       GETNUM: PROCEDURE BYTE PUBLIC;

   59   2                A4 = 0;
   60   2                XB3 = MSG(ML) - 5;
   61   2                IF B1 > XB3 THEN RETURN TRUE;
   63   2                XB2 = B1 + 3;
   64   2                DO B1 = B1 TO XB3;
                             /* CONVERT TO END OF MSGDATA, BUT MAX.  4 DIGITS */
   65   3                  XB1 = MSGDATA(B1);
   66   3                  IF XB1 = ' ' OR XB1 = ',' OR XB1 = '-' THEN RETURN TRUE;
                               /* ' ' AND ',' AND '-' ARE LEGAL DELIMITER */
   68   3                  IF B1 > XB2 THEN RETURN FALSE;
                               /* NUMBER TOO LARGE */
   70   3                  A4 = SHL(A4,4);
   71   3                  IF XB1 >= '0' AND XB1 <= '9'
                            THEN XB1 = XB1 - '0';
   73   3                            ELSE IF XB1 < 'A' OR XB1 > 'F'
                                           THEN RETURN FALSE;
                                               /* ILLEGAL CHARACTER */
   75   3                                   ELSE XB1 = XB1 - 37H;
   76   3                  A4 = A4 + XB1;
   77   3                  END;
   78   2                RETURN TRUE;
   79   2                END GETNUM;
   80   1       END SCPUB6;
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB7
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SCPUB7.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            SCPUB7:   DO;
                $NOLIST

  30    1       CONVASC:  PROCEDURE (P1) EXTERNAL;
  31    2       DECLARE     P1 BYTE;
  32    2                 END;
  33    1       SEND:       PROCEDURE (P1) BYTE EXTERNAL;
  34    2       DECLARE     P1 ADDRESS;
  35    2                 END SEND;

                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                          SYSMON

                    SYSTEM MONITOR
                    MAY BE CALLED WHEN AN INTERNAL ERROR CONDITION OCCURS,
                    PROGRAM LIMITS ARE EXCEEDED ETC.
                    SYSMON GENERATES AN ASYNCHRONOUS PRINTOUT TO INDICATE
                    THE NATURE AND LOCATION OF ERROR CONDITION.
                        INPUT:  P1 - LOCATION OF ERROR OR OTHER ADDRESS VALUE
                                P2,P3 - BYTE VALUES TO SPECIFY ERROR
                **
                ***************************************************************************
                */
```

```
  36   1       SYSMON:     PROCEDURE (P1,P2,P3) PUBLIC;

  37   2       DECLARE     P1 ADDRESS,(P2,P3,I) BYTE,
                          (STACKTOP BASED P1) (2) BYTE;

  38   2                  CALL CONVASC(STACKTOP(1));
  39   2                  MONTEXT(16) = BU;MONTEXT(17) = BL;
  41   2                  CALL CONVASC(STACKTOP(0));
  42   2                  MONTEXT(18) = BU;MONTEXT(19) = BL;
  44   2                  CALL CONVASC(P2);MONTEXT(21) = BU;MONTEXT(22) = BL;
  47   2                  CALL CONVASC(P3);MONTEXT(24) = BU;MONTEXT(25) = BL;
  50   2                  IF NOT SEND(.MONMSG(0)) THEN RETURN;
                              /* SEND ASYNC PRINT MSG TO CRT */
  52   2                  DO I = 0 TO LAST(MONTEXT);
  53   3                    MSGDATA(I) = MONTEXT(I);
  54   3                    END;
  55   2                  RETURN;
  56   2                  END SYSMON;
  57   1       END SCPUB7;
```


MODULE INFORMATION:

```
     CODE AREA SIZE      = 0092H    146D
     VARIABLE AREA SIZE  = 0005H      5D
     MAXIMUM STACK SIZE  = 0002H      2D
     265 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

DEBUG MODULE

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD1
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB1.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1          DBMOD1:      DO;
             $ INCLUDE(:F1:DBDATP.SRC)
        =    /*
        =    ***************************************************************************
        =    ***************************************************************************
        =    **
        =    **                        DEBUG MODULE
        =    **
        =    ***************************************************************************
        =    **
        =    **                  MODULE IDENTIFICATION : DB
        =    **                  MODULE NUMBER         : 07
        =    **
        =    ***************************************************************************
        =    */
        =    $ NOLIST
        =    /*
        =    ***************************************************************************
        =    **
        =    **                             DB DATA
        =    **
        =    ***************************************************************************
        =    **
        =    **                           SUBSTITUTIONS
        =    */
 11   1  =    DCL PROMPT LIT '3EH',                  /* PROMPT CHARACTER '>' */
```

```
=       INITDEBUG LIT '0',
=       SELECTCPTR LIT '1',
=       DEBUGCOMMAND LIT '2',
=       FUNCTIONINPUT LIT '3',
=       ENDSUBST LIT '0';
=   /*
=   ****************************************************************************
=   **
=   **                  MESSAGE CONTROL BLOCKS
=   */
12  1   DCL DBMN10 (4) BYTE PUBLIC INITIAL (0,DB,10,0),
=           /* ASYNC PRINT MSG */
=       DBMN11 (4) BYTE PUBLIC INITIAL (0,DB,11,0),
=           /* SYNC PRINT MSG */
=       DBMN12 (4) BYTE PUBLIC INITIAL (0,DB,12,5),
=           /* INPUT REQUEST MSG */
=       DBMN14 (4) BYTE PUBLIC INITIAL (0,DB,14,4),
=           /* TERMINATE I/O MSG */
=       DBMN15 (4) BYTE PUBLIC INITIAL (0,DB,15,4),
=           /* ACTIVATE DEBUG MSG TO CRT */
=       DBMN16 (4) BYTE PUBLIC INITIAL (0,DB,16,4),
=           /* NEW LINE MSG TO CRT */
=       DBMN17 (4) BYTE PUBLIC INITIAL (0,DB,17,4),
=           /* BEGIN OF INPUT LINE MSG */
=       DBMN18 (4) BYTE PUBLIC INITIAL (0,DB,18,0),
=           /* SYNC PRINT MSG,TELLBACK REQUIRED */
=       DBMN24 (4) BYTE PUBLIC INITIAL (0,DB,24,5),
=           /* DEBUG REQUEST MSG TO EXT. DEBUG MODULE */
=       ENDMCB LIT '0';
=   /*
=   ****************************************************************************
=   **
```

```
          =   **                          CONSTANT DATA
          =   */
  13   1  =   DCL SELCP1 (12) BYTE PUBLIC INITIAL (0,'DEBUG CPTR:'),
          =       SELCP2 (8) BYTE PUBLIC INITIAL (0,'?-CPTR:'),
          =       PROMPTMSG (2) BYTE PUBLIC INITIAL (0,PROMPT),
          =       NOTACTMSG (26) BYTE PUBLIC INITIAL (0,' DEBUG MODULE NOT ACTIV
           -   E,'),
          =       ERRMSG (3) BYTE PUBLIC INITIAL (0,' ?'),
          =       ERRMSG1 (22) BYTE PUBLIC INITIAL (1,'CPTR ALREADY DEBUGGED'),
          =       TERMMSG (18) BYTE PUBLIC INITIAL (1,' DEBUG TERMINATED'),
          =       INSPBUF (17) BYTE PUBLIC INITIAL (0,0,'          :       '),
          =       DBPERINT (4) BYTE PUBLIC INITIAL (0,0,0,0),
          =           /* TIME INTERVAL FOR PERIODIC FUNCTIONS */
          =       FUNCTBL (10) BYTE PUBLIC INITIAL ('A','B','C','D','S','M','H',
           -   'Y',0,'T'),
          =           /* DEBUG COMMAND IDENTIFICATIONS
          =              INDEX IS DBFUNCTION */
          =       EXTRACTSTART (4) BYTE PUBLIC INITIAL (EX,DB,10,4),
          =           /* START EXTRACTION MSG TO EXEC */
          =       EXTRACTSTOP (4) BYTE PUBLIC INITIAL (EX,DB,11,4),
          =           /* STOP EXTRACTION MSG TO EXEC */
          =       MSGTEXT (40) BYTE PUBLIC INITIAL
          =           (1,0,'  RMN:    SMN:    MN:    ML:     DATA:'),
          =       MSGTEXTX (6) ADDRESS PUBLIC INITIAL (0,0602H,060AH,0512H,0519H
           -   ,0820H),
          =       SKIPMSG (18) BYTE PUBLIC INITIAL (3,0,' ',0,0,'MSG NOT TYPED')
           -   ,
          =       SYSTTEXT1 (12) BYTE PUBLIC INITIAL (0,'  FUNCTION:'),
          =       SYSTTEXT2 (14) BYTE PUBLIC INITIAL (0,' SUBFUNCTION:'),
          =       SYSTTEXT3 (7) BYTE PUBLIC INITIAL (0,' DATA:'),
          =       SYSTMSG (4) BYTE PUBLIC INITIAL (3,DB,50H,0),
          =
```

```
=       ENDCONST LIT '0';
=   /*
=   ************************************************************************
=   **
=   **                          VARIABLE DATA
=   */
14  1 =   DCL DBCLEARBEG BYTE PUBLIC,
=
=       DBFUNCTION BYTE PUBLIC INITIAL (0),
=       CRTINPUT BYTE PUBLIC,
=           /* CRTINPUT AND DBFUNCTION CONTROL PROGRAM FLOW
=              IN PROCEDURE DBINPUT */
=       DBCRT BYTE PUBLIC,
=           /* MODULE NUMBER OF CRT MODULE CONNECTED TO DB */
=       HARDCOPY BYTE PUBLIC,
=           /* TRUE IF HARDCOPY REQUESTED */
=       DBTBCODE BYTE PUBLIC,
=           /* CODE OF TELLBACK MSG */
=       DBTELLBACK BYTE PUBLIC,
=           /* TRUE IF TELLBACK MSG RECEIVED */
=       BYTEMODE BYTE PUBLIC,
=           /* TRUE IF DEBUG FUNCTION IN BYTE MODE */
=       DBIN ADDRESS PUBLIC,
=       (DBINH, DBINL) BYTE PUBLIC AT (.DBIN),
=           /* INPUT IN MACHINE FORMAT */
=       SNFL BYTE PUBLIC,
=           /* IF 0, FIRST CALL OF SNAPSHOT */
=       SNAPSHOTADR ADDRESS PUBLIC,
=           /* START ADDRESS OF SNAPSHOT PROCEDURE */
=       DEBUG BYTE PUBLIC,
=           /* TRUE IF CPTR ALREADY DEBUGGED */
=       EXTDB BYTE PUBLIC,
```

```
=           /* MODULE NUMBER OF EXT.  DEBUG MODULE */
=       DBFL BYTE PUBLIC,
=           /* IF 0,FIRST CALL OF DBINPUT */
=       DBINPUTADR ADDRESS PUBLIC,
=           /* START ADDRESS OF DBINPUT */
=       PERFUNCTION BYTE PUBLIC,
=           /* TRUE IF PERIODIC FUNCTION ACTIVE */
=       TERMPERFUNCT BYTE PUBLIC,
=           /* TRUE IF TERMINATE PERIODIC FUNCTION MSG RECEIVED */
=       DBADR ADDRESS PUBLIC,
=       CONTBYTE BASED DBADR BYTE,
=       CONTADR BASED DBADR ADDRESS,
=       (DBADRL,DBADRH) BYTE PUBLIC AT (.DBADR),
=           /* ADDRESS OF DEBUG LOCATION */
=       DBFUNCTSAVE BYTE PUBLIC,
=           /* PREVIOUS FUNCTION */
=       DBOUTBUF (160) BYTE PUBLIC,
=           /* OUTPUT BUFFER */
=       DBPERX BYTE PUBLIC,
=           /* INDEX OF DEBUG PERIODIC */
=       SNPRIORX BYTE PUBLIC,
=           /* SNAPSHOT PRIORITY INDEX */
=       (DUMPBEG,DUMPEND) ADDRESS PUBLIC,
=           /* DUMP RANGE */
=       ENDLINE BYTE PUBLIC,
=           /* LAST BYTE TO TYPE IN A DUMP LINE */
=       DUMPDONE BYTE PUBLIC,
=           /* TRUE IF DUMP DONE */
=       (DUMPCONTA BASED A2) (1) ADDRESS,
=       (DUMPCONTB BASED A2) (1) BYTE,
=       DUMPBEGTMP ADDRESS PUBLIC,
=       DBMSGDATA (20H) BYTE PUBLIC,
```

```
            =           /* DATA OF SIMULATED MSG */
            =       DBMSWITCH BYTE PUBLIC,
            =           /* CONTROLS INPUT OF 'M' COMMAND */
            =       MSGSIM BYTE PUBLIC,
            =           /* TRUE IF MSG SIMULATION */
            =       MSGEX BYTE PUBLIC,
            =           /* TRUE IF MSG EXTRACTION */
            =       MSGDATAX BYTE PUBLIC,
            =           /* INDEX TO DBMSGDATA */
            =       EXTRIND BYTE PUBLIC,
            =       DBEXTRFL BYTE PUBLIC,
            =           /* TRUE IF A MSG EXTRACTION IN PROCESS */
            =       DBMSGCOUNT BYTE PUBLIC,
            =           /* NUMBER OF SAME MSG DURING THE PROCESS OF EXTRACTION */
            =       DBSYSTSW BYTE PUBLIC,
            =           /* SWITCH FOR SYSTEM TEST INPUTS */
            =       DBSTBUF (20H) BYTE PUBLIC AT (.DBMSGDATA(0)),
            =       DBSTBUFX BYTE PUBLIC,
            =           /* BUFFER AND INDEX FOR SYSTEM TEST INPUTS */
            =
            =       DBCLEAREND BYTE PUBLIC;
            =   /*
            =   ***************************************************************************
            =   **
            =   **                  SNAPSHOT DATA
            =   */
 15   1     =   DCL INSTRTBL (44) BYTE PUBLIC INITIAL
            =           (0D3H,6,16H,26H,36H,0C6H,0D6H,0E6H,0F6H,0DBH,0EH,1EH,2EH,3
             -    EH,
            =            0CEH,0DEH,0EEH,0FEH,1,11H,21H,31H,0C2H,0D2H,0E2H,0F2H,22H
             -    ,32H,
            =            0C3H,0C4H,0D4H,0E4H,0F4H,0CAH,0DAH,0EAH,0FAH,2AH,3AH,0CCH
```

```
                -   ,
                =           0DCH, 0ECH, 0FCH, 0CDH),
                =       SNDECTBL (5) BYTE PUBLIC INITIAL ('RMDAB'),
                =       SNEXTBL (14) BYTE PUBLIC INITIAL
                =             (0E1H, 0F1H, 0C1H, 0D1H, 0E1H, 33H, 33H, 0FBH, 0, 0, 0, 0, 0, 0),
                =       SNEXTBLA (7) ADDRESS PUBLIC AT (.SNEXTBL(0)),
                =           /* EXECUTION TABLE FOR REPLACED INSTRUCTION */
                =       SNTEXT1 (25) BYTE PUBLIC INITIAL
                =           (0,'SNAPSHOT AT      :       '),
                =       SNTEXT3 (66) BYTE PUBLIC INITIAL
                =         (0,0,'  A:   B:   C:   D:   E:   H:   L:   M:   STACK:
                -              '),
225             =       SNTEXT4 (28) BYTE PUBLIC INITIAL
                =           (4,0,'     SNAPSHOT(S) NOT TYPED');
     16   1     =   DCL SNCLEARBEG BYTE PUBLIC,
                =       SNREG (12) BYTE PUBLIC,
                =       SNREGA (6) ADDRESS PUBLIC AT (.SNREG(0)),
                =           /* TEMP REGISTER STORAGE */
                =       SNVAR (5) STRUCTURE (
                =         ADR ADDRESS,
                =         BYTETYPE BYTE,
                =         CONTA ADDRESS,
                =         CONTB BYTE) PUBLIC,
                =           /* TEMP VARIABLE STORAGE */
                =       SNDUMP (20H) BYTE PUBLIC,
                =           /* TEMP MEMORY DUMP STORAGE */
                =       REGFL BYTE PUBLIC,
                =           /* TRUE - TYPE REGISTER CONTENTS */
                =       DUMPFL BYTE PUBLIC,
                =           /* TRUE - TYPE DUMP */
                =       SNAPFL BYTE PUBLIC,
                =           /* TRUE - SNAPSHOT IN PROGRESS */
```

```
=       CONDFL BYTE PUBLIC,
=           /* TRUE - CONDITION SPECIFIED */
=       SNADR ADDRESS PUBLIC,
=       (SNADRCONT BASED SNADR) (1) BYTE,
=           /* SNAPSHOT ADDRESS AND CONTENTS */
=       MSGDATL BYTE PUBLIC,
=           /* LAST BYTE IN MSGDATA */
=       SNVARX BYTE PUBLIC,
=           /* INDEX FOR SNVAR */
=       SNCONDADR ADDRESS PUBLIC,
=       SNCONDSAVE BYTE PUBLIC,
=       (SNCOND BASED SNCONDADR) BYTE,
=           /* CONDITION ADDRESS AND CONTENTS */
=       SNCTR BYTE PUBLIC,
=           /* SNAPSHOT COUNTER */
=       TEMPINT0 (8) BYTE PUBLIC AT (30H),
=           /* TEMP STORAGE FOR 0000-0007 */
=       (MEMINT0 BASED A1) (1) BYTE,
=       (MEMINT5 BASED A2) (1) BYTE,
=           /* OVERLAYS FOR INT 0 AND INT 5 */
=       INSTRL BYTE PUBLIC,
=           /* INSTRUCTION LENGTH */
=       (SNB1,SNB2) BYTE PUBLIC,(SNA1,SNA2) ADDRESS PUBLIC,
=           /* SNAP INT WORK VARIABLES */
=       (SNCONTADR BASED SNA1) ADDRESS,
=       (SNCONTBYTE BASED SNA1) BYTE,
=       (DUMPDATA BASED DUMPBEG) (1) BYTE,
=       SNNUMDUMP BYTE PUBLIC,
=           /* NUMBER OF LOCATIONS TO DUMP */
=       (SNSTACK BASED SNA2) (12) BYTE,
=           /* OVERLAY FOR STACK */
=       SNCLEAREND BYTE PUBLIC;
```

```
                $ EJECT
 104    1       DBINPUT:  PROCEDURE EXTERNAL;
 105    2                 END;
 106    1       DBREQ:       PROCEDURE EXTERNAL;
 107    2                 END;
 108    1       DBEXTREQ:PROCEDURE EXTERNAL;
 109    2                 END;
 110    1       DBTERMPER: PROCEDURE EXTERNAL;
 111    2                 END;
 112    1       SNAPSHOT:  PROCEDURE EXTERNAL;
 113    2                 END;
 114    1       SNAPINT:     PROCEDURE EXTERNAL;
 115    2                 END ;
 116    1       SNAPTERM:   PROCEDURE EXTERNAL;
 117    2                 END;
 118    1       DBEXTRACT:   PROCEDURE EXTERNAL;
 119    2                   END;

                /*
                ***************************************************************
                ***************************************************************
                **
                                                              DBSTART

                                    SYSTEM START
                **
                ***************************************************************
                */
 120    1       DBSTART:     PROCEDURE;;

 122    2                 IF DBFUNCTION = 4 THEN CALL SNAPTERM;
                             /* RESTART,RESTORE ACTIVATED SNAPSHOT */
```

```
124   2                 CALL CLEARDATA(.DBCLEARBEG,.DBCLEAREND);
                            /* CLEAR VARIABLE DATA REGION */
125   2                 CALL CLEARDATA(.SNCLEARBEG,.SNCLEAREND);
                            /* CLEAR SNAPSHOT DATA */
126   2                 CALL UPDSTAT(DB,3);
                            /* SET DB MODULES STATUS */
127   2                 CALL DBINPUT;
128   2                 CALL SNAPINT;
129   2                 CALL SNAPSHOT;
130   2                 SNPRIORX = ENTERPRIOR(SNAPSHOTADR);
                            /* ENTER SNAPSHOT PRIORITY CALL */
131   2                 DBFUNCTSAVE = 0FFH;
132   2                 RETURN;
133   2                 END DBSTART;
                  /*************************************************************************
                  **************************************************************************
                  **
                                                                        MSGENT07

                  **
                  **************************************************************************
                  */
134   1           MSGENT07:   PROCEDURE PUBLIC;

135   2                   IF MSG(MN) = 0 THEN
                            /* START MSG */
136   2                     DO;
137   3                     CALL DBSTART;
138   3                     RETURN;
139   3                     END;
140   2                   IF MSG(MN) < 20 OR MSG(MN) > 28 THEN
                            /* ILLEGAL MSG */
```

228

```
141   2                 DO;
142   3                 CALL ILLEGALMSG;
143   3                 RETURN;
144   3                 END;
145   2               B1 = MSG(MN) - 20;
146   2               DO CASE B1;
                        /* BRANCH TO MSG PROCESS */
147   3                 CALL DBINPUT;
148   3                 CALL ILLEGALMSG;
149   3                 DO;
150   4                   DEBUG = FALSE;
151   4                   CRTINPUT = INITDEBUG;
152   4                   DBFUNCTION = 0FFH;
153   4                   IF PERFUNCTION THEN CALL DBTERMPER;
155   4                   END;
156   3                 CALL DBINPUT;
157   3                 CALL DBEXTREQ;
158   3                 CALL DBTERMPER;
159   3                 DO;
160   4                   IF TERMPERFUNCT OR DBFUNCTION = 0FFH THEN RETURN;
                            /* FUNCTION NO LONGER ACTIVE */
162   4                   DBTBCODE = MSGDATA(0);
163   4                   DBTELLBACK = TRUE;
164   4                   CALL DBINPUT;
165   4                   END;
166   3                 DO;
167   4                 IF DBEXTRFL THEN DBMSGCOUNT = DBMSGCOUNT + 1;
169   4                              ELSE CALL DBEXTRACT;
                        /* EXTRACTION MSG FROM EXEC */
170   4                 END;
171   3                 HARDCOPY = FALSE;
172   3                 END;
```

```
 173   2                RETURN;
 174   2                END MSGENT07;
 175   1        END DBMOD1;


MODULE INFORMATION:

     CODE AREA SIZE      = 0108H     264D
     VARIABLE AREA SIZE  = 0312H     786D
     MAXIMUM STACK SIZE  = 0004H       4D
     684 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD2
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB2.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            DBMOD2:      DO;
                $ INCLUDE(:F1:DBDATE.SRC)
            =   /*
            =   ***************************************************************
            =   ***************************************************************
            =   **
            =   **                      DEBUG MODULE
            =   **
            =   ***************************************************************
            =   **
            =   **                MODULE IDENTIFICATION : DB
            =   **                MODULE NUMBER         : 07
            =   **
            =   ***************************************************************
            =   */
            =   $ NOLIST
  104    1       DBINPREQ:    PROCEDURE EXTERNAL;
  105    2                   END;
  106    1       DBSYNCPR:    PROCEDURE EXTERNAL;
  107    2                   END;
  108    1       DBREQ:       PROCEDURE EXTERNAL;
  109    2                   END;
  110    1       DBTERM:      PROCEDURE EXTERNAL;
  111    2                   END;
  112    1       DBMSX:       PROCEDURE EXTERNAL;
  113    2                   END;
```

```
114   1       DBILLEGAL:  PROCEDURE EXTERNAL;
115   2               END;
116   1       DBINSPECT:  PROCEDURE BYTE EXTERNAL;
117   2               END;
118   1       DBCHANGE:   PROCEDURE EXTERNAL;
119   2               END;
120   1       DBDUMP:     PROCEDURE EXTERNAL;
121   2               END;
122   1       DBSYSTEST:  PROCEDURE EXTERNAL;
123   2               END;
124   1       DBHARDCOPY:  PROCEDURE EXTERNAL;
125   2                  END;
126   1       DBSNAP:     PROCEDURE EXTERNAL;
127   2               END;

                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                          DBINPUT

                    DISTRIBUTE CONTROL AFTER CRT INPUT
                **
                ****************************************************************************
                */
128   1       DBINPUT:    PROCEDURE PUBLIC;

129   2               IF DBFL = 0 THEN
                            /* FIRST CALL, FIND ADDRESS OF DBINPUT */
130   2                 DO;
131   3                 DBINPUTADR = PROCADR;
132   3                 DBFL = 1;
```

```
 133   3                RETURN;
 134   3                END;
 135   2              DO CASE CRTINPUT;
 136   3                GO TO DEBUGINIT;
 137   3                GO TO SELCPTR;
 138   3                GO TO SELFUNC;
 139   3                GO TO PROCESSFUNC;
 140   3                END;
                /*
                ***************************************************************************
                **
                **  INITIALIZE DEBUG
                */
 141   2      DEBUGINIT:
                DBCRT = CS;
 142   2              A1 = .SELCP1(0);
 143   2              B1 = LENGTH(SELCP1);
                        /* TYPE 'CPTR:' ON CRT */
 144   2      DBINP1:
                CALL DBSYNCPR;
 145   2              B1 = FALSE;
 146   2              CALL DBINPREQ;
                        /* REQUEST INPUT WITHOUT ROLL */
 147   2              CRTINPUT = SELECTCPTR;
                        /* PROCESS SELECT COMPUTER INPUT NEXT CALL */
 148   2              RETURN;
                /*
                ***************************************************************************
                **
                **  SELECT COMPUTER
                */
 149   2      SELCPTR:
```

```
                        B3 = MSGDATA(0) - 30H;
                            /* DECODE CPTR NUMBER */
150    2                IF MSG(ML) <> 5 OR B3 > MAXCPTR
                          THEN DO;
                              /* ILLEGAL INPUT */
152    3                   A1 = .SELCP2(0);
153    3                   B1 = LENGTH(SELCP2);
                               /* TYPE '?-CPTR:' ON CRT */
154    3                   GO TO DBINP1;
155    3                   END;
156    2                 ELSE DO;
                             /* INPUT O.K. */
157    3                   IF B3 = CPTR OR B3 = 0
                             THEN CALL DBREQ;
                                 /* DEBUG IN OWN COMPUTER */
159    3                   ELSE DO;
                                 /* DEBUG IN OTHER COMPUTER */
160    4                     EXTDB = 0;
161    4                     DO B2 = 1 TO B3;
162    5                       EXTDB = EXTDB + MAXMOD;
163    5                       END;
164    4                     EXTDB = EXTDB - 1;
                                 /* COMPUTE MODULE NUMBER OF DEBUG MODULE IN
                                    REQUESTED COMPUTER */
165    4                     IF NOT ACTIVE(EXTDB)
                               THEN DO;
                                   /* REQUESTED DEBUG MODULE NOT ACTIVE */
167    5                       B1 = LENGTH(NOTACTMSG);
168    5                       A1 = .NOTACTMSG(0);
169    5                       CALL DBSYNCPR;
                                    /* TYPE NOT ACTIVE MSG */
170    5                       CALL DBTERM;
```

```
                                        /* TERMINATE DEBUG */
 171   5                                RETURN;
 172   5                                END;
 173   4                              ELSE DO;
                                         /* REQUESTED DEBUG MODULE ACTIVE */
 174   5                                DBMN24(0) = EXTDB;
 175   5                              IF SEND(.DBMN24(0)) THEN MSGDATA(0) = CS;
                                        /* SEND DEBUG REQUEST MSG TO EXT.  DEBUG MO
           -   DULE
                                           DATA BYTE IS MODULE NUMBER OF CRT MODUL
           -   E */
 177   5                                CRTINPUT = INITDEBUG;
                                        /* RESET CONTROL VARIABLE */
 178   5                                END;
 179   4                              END;
 180   3                            END;
 181   2                    RETURN;
                      /*
                      ***************************************************************************
           -           *******
                      **
                      **  SELECT FUNCTION
                      */
 182   2                  SELFUNC:
                            TERMPERFUNCT = FALSE;
 183   2                          IF MSG(ML) > 4
                             THEN DO;
                                  /* CHARACTERS ENTERED */
 185   3                            IF MSGDATA(0) = 'P'
                                    THEN B1 = 1;
                                       /* PERIODIC FUNCTION */
 187   3                              ELSE B1 = 0;
```

```
                        /* NON PERIODIC FUNCTION */
188   3               B3 = MSGDATA(B1);
189   3               DO B2 = 0 TO LAST(FUNCTBL);
                          /* SEARCH FOR COMMAND */
190   4                 IF FUNCTBL(B2) = B3 THEN
191   4                   DO;
192   5                   DBFUNCTION = B2;
                              /* SET FUNCTION IDENTIFIER */
193   5                   CRTINPUT = FUNCTIONINPUT;
                              /* PROCESS FUNCTION INPUT NEXT */
194   5                   IF FUNCTBL(B2) <> 'C' THEN DBFUNCTSAVE = 0FFH;
196   5                   GO TO PROCESSFUNC;
197   5                   END;
198   4                 END;
199   3               END;
200   2             ELSE DO;
                        /* INPUT CR ONLY */
201   3               IF DBFUNCTSAVE < 3 THEN
                            /* INSPECT NEXT ADDRESS/BYTE */
202   3                   DO;
203   4                   B3 = DBINSPECT;
204   4                   RETURN;
205   4                   END;
206   3                 IF DBFUNCTSAVE = 5 AND MSGSIM THEN
                            /* REPEAT MSG SIMULATION */
207   3                   DO;
208   4                   CALL DBMSX;
209   4                   RETURN;
210   4                   END;
211   3                 END;
212   2             CALL DBILLEGAL;
                        /* NO CHARACTERS ENTERED
```

```
                                OR COMMAND NOT FOUND */
213   2                 RETURN;
                  /*
                  ************************************************************************
                  **
                  **  PROCESS FUNCTION INPUT
                  */
214   2           PROCESSFUNC:
                          DO CASE DBFUNCTION;
                              /* BRANCH TO SELECTED FUNCTION */
215   3                       IF NOT DBINSPECT THEN GO TO SELFUNC;
                              IF NOT DBINSPECT THEN GO TO SELFUNC;
                              CALL DBCHANGE;
220   3                       CALL DBDUMP;
221   3                       CALL DBSNAP;
222   3                       CALL DBMSX;
223   3                       CALL DBHARDCOPY;
224   3                       CALL DBSYSTEST;
225   3                       ;
226   3                       CALL DBTERM;
227   3                       END;
228   2                 RETURN;
229   2                 END DBINPUT;
230   1           END DBMOD2;
```


MODULE INFORMATION:

```
     CODE AREA SIZE      = 0220H     544D
     VARIABLE AREA SIZE  = 0000H       0D
     MAXIMUM STACK SIZE  = 0002H       2D
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD3
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB3.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            DBMOD3:    DO;
                $ INCLUDE(:F1:DBDATE.SRC)
            =   /*
            =   ***************************************************************
            =   ***************************************************************
            =   **
            =   **                    DEBUG MODULE
            =   **
            =   ***************************************************************
            =   **
            =   **                MODULE IDENTIFICATION : DB
            =   **                MODULE NUMBER         : 07
            =   **
            =   ***************************************************************
            =   */
            =   $ NOLIST

 104    1       DBSYNCPRH:  PROCEDURE EXTERNAL;
 105    2               END;
 106    1       DBNEWLINEH: PROCEDURE EXTERNAL;
 107    2               END;
 108    1       DBBEGLINEH: PROCEDURE EXTERNAL;
 109    2               END;
 110    1       DBSYNCPR:   PROCEDURE EXTERNAL;
 111    2               END;
 112    1       PACKADR:  PROCEDURE (P1) EXTERNAL;
```

```
113   2       DCL     P1 ADDRESS;
114   2               END;
115   1       PACKBYTE: PROCEDURE (P1) EXTERNAL;
116   2       DCL         P1 BYTE;
117   2               END;
118   1       GETADR:   PROCEDURE BYTE EXTERNAL;
119   2               END;
120   1       DBILLEGAL:  PROCEDURE EXTERNAL;
121   2               END;
122   1       DBNEWLINE: PROCEDURE EXTERNAL;
123   2               END;
124   1       DBBEGLINE:  PROCEDURE EXTERNAL;
125   2               END;
126   1       DBPROMPT: PROCEDURE EXTERNAL;
127   2               END;
128   1       GETINP:     PROCEDURE BYTE EXTERNAL;
129   2               END;

                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                        TYPEINSP

                    PACK INSPECTION OUTPUT (A OR B)
                **
                ****************************************************************************
                */
130   1       TYPEINSP:   PROCEDURE PUBLIC;

131   2               DO B1 = 0 TO LAST(INSPBUF);
                            /* PACK FIXED TEXT */
```

```
132   3                 DBOUTBUF(B1) = INSPBUF(B1);
133   3                 END;
134   2               B1 = 5;
135   2               CALL PACKADR(DBADR);
136   2               IF NOT BYTEMODE
                        THEN DO;
                             /* A */
138   3                   DBOUTBUF(4) = 'A';
139   3                   B1 = 12;
140   3                   CALL PACKADR(CONTADR);
141   3                   END;
142   2                 ELSE DO;
                             /* B */
143   3                   DBOUTBUF(4) = 'B';
144   3                   B1 = 12;
145   3                   CALL PACKBYTE(CONTBYTE);
146   3                   END;
147   2               IF DBTELLBACK
                        THEN A1 = .DBOUTBUF(0);
149   2                   ELSE A1 = .DBOUTBUF(1);
150   2               CALL DBSYNCPRH;
151   2               RETURN;
152   2               END TYPEINSP;
```

```
/*
*****************************************************************************
*****************************************************************************
**
                                                                    DBCHANGE

    PROCESS CHANGE COMMAND
**
*****************************************************************************
```

```
                *)
 153    1       DBCHANGE:   PROCEDURE PUBLIC;

 154    2                   IF DBFUNCTSAVE > 1 THEN
                                /* CHANGE AFTER INSPECT ONLY */
 155    2                     DO;
 156    3                     CRTINPUT = FUNCTIONINPUT;
 157    3                     GO TO DBCHG1;
 158    3                     END;
 159    2                   B1 = B1 + 1;
 160    2                   IF NOT GETINP THEN GOTO DBCHG1;
                                /* ILLEGAL INPUT */
 162    2                   IF NOT BYTEMODE
                              THEN CONTADR = DBIN;
                                   /* CHANGE ADDRESS VALUE */
 164    2                     ELSE DO;
                                   /* CHANGE BYTE VALUE */
 165    3                       IF DBINL <> 0 THEN GOTO DBCHG1;
                                    /* ILLEGAL INPUT */
 167    3                       CONTBYTE = DBINH;
 168    3                       END;
 169    2                   CALL TYPEINSP;
                                /* TYPE ADDRESS AND NEW CONTENTS */
 170    2                   CALL DBPROMPT;
                                /* SEND PROMPT CHARACTER */
 171    2                   RETURN;
 172    2       DBCHG1:         CALL DBILLEGAL;
 173    2                   RETURN;
 174    2                   END DBCHANGE;
                    /*
                    ************************************************************************
                    ************************************************************************
```

```
                **
                                                        DBINSPECT

                    PROCESS INSPECT COMMANDS ( A AND B )
                **
                ************************************************************************
                */
 175   1        DBINSPECT:  PROCEDURE BYTE PUBLIC;

 176   2                  IF PERFUNCTION THEN
 177   2                    DO;
                             /* PERIODIC INSPECT */
 178   3                    IF NOT DBTELLBACK THEN RETURN TRUE;
                                /* CURRENT TYPING NOT COMPLETED YET */
 180   3                    CALL DBBEGLINEH;
                                /* CURSOR TO BEGIN OF INPUT LINE */
 181   3                    CALL TYPEINSP;
                                /* TYPE CONTENTS */
 182   3                    RETURN TRUE;
 183   3                    END;
 184   2                  IF MSG(ML) = 4 THEN
 185   2                    DO;
                             /* INPUT CR, INSPECT NEXT ADDRESS */
 186   3                    IF NOT BYTEMODE
                               THEN DBADR = DBADR + 2;   /* INSPECT A */
 188   3                            ELSE DBADR = DBADR + 1;   /* INSPECT B */
 189   3                    GO TO INSP1;
 190   3                    END;
 191   2                  IF DBFUNCTSAVE <> 0FFH THEN RETURN FALSE;
                             /* NEW FUNCTION INPUT */
 193   2                  PERFUNCTION = B1;
 194   2                  B1 = B1 + 1;
```

```
 195   2                IF NOT GETADR THEN
                            /* ILLEGAL ADDRESS INPUT */
 196   2                  DO;
 197   3                  CALL DBILLEGAL;
 198   3                  DBFUNCTSAVE = 0FFH;
 199   3                  PERFUNCTION = FALSE;
 200   3                  RETURN TRUE;
 201   3                  END;
 202   2                DBFUNCTSAVE = DBFUNCTION;
 203   2                BYTEMODE = DBFUNCTION;
 204   2                IF PERFUNCTION THEN
 205   2                  DO;
 206   3                  CALL DBNEWLINEH;
                              /* CURSOR AT BEGIN OF NEW LINE */
 207   3                  DBTELLBACK = TRUE;
                              /* REQUEST TELLBACK MSG AFTER TYPING */
 208   3                  CALL TYPEINSP;
 209   3                  RETURN TRUE;
 210   3                  END;
 211   2       INSP1:     CALL TYPEINSP;
                            /* TYPE REQUESTED CONTENTS */
 212   2                CALL DBPROMPT;
                            /* SEND PROMPT CHARACTER */
 213   2                RETURN TRUE;
 214   2                END DBINSPECT;
 215   1       END DBMOD3;


MODULE INFORMATION:

     CODE AREA SIZE     = 0168H    360D
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD4
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB4.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            DBMOD4:        DO;
                $ INCLUDE(:F1:DBDATE.SRC)
            =   /*
            =   ***************************************************************************
            =   ***************************************************************************
            =   **
            =   **                      DEBUG MODULE
            =   **
            =   ***************************************************************************
            =   **
            =   **                MODULE IDENTIFICATION : DB
            =   **                MODULE NUMBER         : 07
            =   **
            =   ***************************************************************************
            =   */
            =   $ NOLIST
  104   1       DBSYNCPRH:  PROCEDURE EXTERNAL;
  105   2               END;
  106   1       DBNEWLINEH: PROCEDURE EXTERNAL;
  107   2               END;
  108   1       DBBEGLINEH: PROCEDURE EXTERNAL;
  109   2               END;
  110   1       DBSYNCPR:   PROCEDURE EXTERNAL;
  111   2               END;
  112   1       PACKADR:    PROCEDURE (P1) EXTERNAL;
  113   2       DCL     P1 ADDRESS;
```

```
114   2             END;
115   1       PACKBYTE:   PROCEDURE (P1) EXTERNAL;
116   2       DCL     P1 BYTE;
117   2             END;
118   1       DBNEWLINE:  PROCEDURE EXTERNAL;
119   2             END;
120   1       DBBEGLINE:  PROCEDURE EXTERNAL;
121   2             END;
122   1       GETINP:     PROCEDURE BYTE EXTERNAL;
123   2             END;
124   1       DBILLEGAL:  PROCEDURE EXTERNAL;
125   2             END;
126   1       DBPROMPT:   PROCEDURE EXTERNAL;
127   2             END;

              /*
              ***************************************************************************
              ***************************************************************************
              **
                                                                        DUMPLINE

                  DUMP 1 LINE (A OR B) UP TO ENDLINE
              **
              ***************************************************************************
              */
128   1       DUMPLINE:   PROCEDURE PUBLIC;

129   2             A2 = DUMPBEGTMP;
130   2             B1 = 5;
131   2             CALL PACKADR(DUMPBEG);
                        /* TYPE ADDRESS OF FIRST CONTENTS ON LINE */
132   2             B1 = 12;
```

```
133   2                 DO B2 = 0 TO ENDLINE;
134   3                   IF BYTEMODE
                            THEN DO;
136   4                       CALL PACKBYTE(DUMPCONTB(B2));
137   4                       DUMPBEGTMP = DUMPBEGTMP + 1;
138   4                       END;
139   3                     ELSE DO;
140   4                       CALL PACKADR(DUMPCONTA(B2));
141   4                       DUMPBEGTMP = DUMPBEGTMP + 2;
142   4                       END;
143   3                   DBOUTBUF(B1) = ' ';B1 = B1 + 1;
145   3                   IF DUMPBEGTMP > DUMPEND THEN
                              /* END OF DUMP */
146   3                     DO;
147   4                     DUMPDONE = TRUE;
148   4                     RETURN;
149   4                     END;
150   3                   END;
151   2                 RETURN;
152   2                 END DUMPLINE;
                  /*
                  ****************************************************************************
                  ****************************************************************************
                  **
                                                                          INITDUMP

                      INITIALIZE DUMP
                  **
                  ****************************************************************************
                  */
153   1           INITDUMP:   PROCEDURE PUBLIC;
```

```
154   2             DUMPDONE = FALSE;
155   2             DO B2 = 0 TO LAST(INSPBUF);
                        /* PRESET BUFFER */
156   3               DBOUTBUF(B2) = INSPBUF(B2);
157   3               END;
158   2             IF SNAPFL
                      THEN DUMPBEGTMP = .SNDUMP(0);
                           /* CALLED FROM SNAPSHOT */
160   2               ELSE DUMPBEGTMP = DUMPBEG;
                           /* ORDINARY DUMP */
161   2             IF BYTEMODE
                      THEN DO;
163   3                 ENDLINE = 0FH;
164   3                 DBOUTBUF(4) = 'B';
165   3                 END;
166   2               ELSE DO;
167   3                 ENDLINE = 7;
168   3                 DBOUTBUF(4) = 'A';
169   3                 END;
170   2             RETURN;
171   2             END INITDUMP;
          /*
          ************************************************************************
          ************************************************************************
          **
                                                                        DUMP

              DUMP CONTENTS FROM DUMPBEG TO DUMPEND
              MODE (A OR B) IN BYTEMODE
          **
          ************************************************************************
          */
```

```
172   1       DUMP:       PROCEDURE PUBLIC;

173   2                   CALL DUMPLINE;
174   2                   IF DBTELLBACK
                            THEN A1 = .DBOUTBUF(0);
176   2                         ELSE A1 = .DBOUTBUF(1);
177   2                   B1 = B1 - 1;
178   2                   CALL DBSYNCPRH;
                              /* TYPE 1 ROW */
179   2                   IF DUMPDONE THEN RETURN;
181   2                   DUMPBEG = DUMPBEG + 10H;
                              /* COMPUTE BEGIN ADDRESS OF NEW LINE */
182   2             RETURN;
183   2             END DUMP;
              /*
              ***************************************************************************
              ***************************************************************************
              **
                                                                    DUMPDECODE

                  DECODE DUMP INPUT, SET DUMPBEG, DUMPEND, BYTEMODE
              **
              ***************************************************************************
              */
184   1       DUMPDECODE: PROCEDURE BYTE PUBLIC;

185   2             B1 = B1 + 1;
186   2             BYTEMODE = MSGDATA(B1) - 'A';
187   2             IF BYTEMODE > 1 THEN RETURN FALSE;
189   2             B1 = B1 + 1;
190   2             IF NOT GETINP THEN RETURN FALSE;
                        /* ILLEGAL ADDRESS INPUT */
```

```
192   2             DUMPBEG = DBIN;
193   2             B3 = MSGDATA(B1);
194   2             B1 = B1 + 1;
195   2             IF NOT GETINP THEN RETURN FALSE;
                        /* ILLEGAL SECOND ADDRESS INPUT */
197   2             IF B3 = '-'
                      THEN DUMPEND = DBIN;
199   2                 ELSE IF B3 = ','
                              THEN DO;
201   3                               IF NOT BYTEMODE THEN DBIN = DBIN + DBIN;
203   3                               DUMPEND = DUMPBEG + DBIN - 1;
204   3                               END;
205   2                           ELSE RETURN FALSE;
206   2             RETURN TRUE;
207   2             END DUMPDECODE;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                          DBDUMP

                PROCESS DUMP COMMAND
            **
            ***************************************************************************
            */
208   1       DBDUMP:     PROCEDURE PUBLIC;

209   2             IF PERFUNCTION THEN
                        /* PERIODIC DUMP ACTIVE */
210   2               DO;
211   3               IF NOT DBTELLBACK THEN RETURN;
                          /* END OF DUMP */
```

```
213   3                 CALL DBBEGLINEH;
                            /* CURSOR TO BEGIN OF LINE */
214   3                 A2 = DUMPBEG; DUMPBEGTMP = DUMPBEG;
216   3                 CALL DUMP;
217   3                 RETURN;
218   3                 END;
219   2               IF DBTELLBACK THEN GO TO DUMPCONT;
                          /* DUMP ALREADY ACTIVATED, CONTINUE */
221   2               PERFUNCTION = B1;
222   2               IF NOT DUMPDECODE THEN
                          /* ERROR IN DUMP INPUT */
223   2                 DO;
224   3                 CALL DBILLEGAL;
225   3                 PERFUNCTION = FALSE;
226   3                 RETURN;
227   3                 END;
228   2               DBFUNCTSAVE = DBFUNCTION;
229   2               B3 = DUMPEND - DUMPBEG;
230   2               IF B3 > 0FH THEN DBTELLBACK = TRUE;
                          /* DUMP MORE THAN 1 ROW */
232   2               IF PERFUNCTION THEN
233   2                 DO;
234   3                 IF B3 > 0FH THEN DUMPEND = DUMPBEG + 0FH;
                              /* ADJUST DUMP REGION IF TOO LARGE */
236   3                 DBTELLBACK = TRUE;
237   3                 END;
238   2               CALL INITDUMP;

239   2       DUMPCONT:    ;
240   2               CALL DBNEWLINEH;
241   2               CALL DUMP;
242   2               IF PERFUNCTION THEN RETURN;
```

```
 244   2                IF NOT DUMPDONE THEN RETURN;
 246   2                CALL DBPROMPT;
                             /* SEND PROMPT FOR NEW DEBUG INPUT */
 247   2                RETURN;
 248   2                END DBDUMP;
 249   1       END DBMOD4;
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 0245H    581D
     VARIABLE AREA SIZE  = 0000H      0D
     MAXIMUM STACK SIZE  = 0006H      6D
     775 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD5
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB5.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            DBMOD5:      DO;

                $ INCLUDE(:F1:DBDATE.SRC)
              =  /*
              =  ***************************************************************
              =  ***************************************************************
              =  **
              =  **                      DEBUG MODULE
              =  **
              =  ***************************************************************
              =  **
              =  **                 MODULE IDENTIFICATION : DB
              =  **                 MODULE NUMBER         : 07
              =  **
              =  ***************************************************************
              =  */
              =  $ NOLIST

 104    1      DBNEWLINEH: PROCEDURE EXTERNAL;
 105    2              END;
 106    1      DBSYNCPRH:  PROCEDURE EXTERNAL;
 107    2              END;
 108    1      PACKADR:    PROCEDURE (P1) EXTERNAL;
 109    2      DCL      P1 ADDRESS;
 110    2              END;
 111    1      PACKBYTE:   PROCEDURE (P1) EXTERNAL;
```

```
112   2       DCL     P1 BYTE;
113   2               END;
114   1       DUMP:       PROCEDURE EXTERNAL;
115   2               END;
116   1       DUMPDECODE: PROCEDURE BYTE EXTERNAL;
117   2               END;
118   1       GETINP:     PROCEDURE BYTE EXTERNAL;
119   2               END;
120   1       DBSYNCPR:   PROCEDURE EXTERNAL;
121   2               END;
122   1       DBILLEGAL:  PROCEDURE EXTERNAL;
123   2               END;
124   1       DBNEWLINE:  PROCEDURE EXTERNAL;
125   2               END;
126   1       INITDUMP: PROCEDURE EXTERNAL;
127   2               END;

                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                        SNAPSHOT

                    PROCESS SNAPSHOT (PRIORITY PROCEDURE )
                **
                ***************************************************************************
                */
128   1       SNAPSHOT:   PROCEDURE PUBLIC;

129   2               IF SNFL = 0 THEN
                            /* FIRST CALL,FIND ADDRESS OF SNAPSHOT */
130   2                 DO;
```

```
131   3                 SNAPSHOTADR = PROCADR;
132   3                 SNFL = 1;
133   3                 RETURN;
134   3                 END;
135   2               IF DBTELLBACK THEN
136   2                 DO CASE DBTBCODE;
                            /* PROCESS TELLBACK MSG */
137   3                   GO TO TB0;
138   3                   GO TO TB1;
139   3                   GO TO TB2;
140   3                   GO TO TB3;
141   3                   GO TO TB4;
142   3                   END;
143   2               CALL DBNEWLINEH;
144   2               DO B3 = 0 TO LAST(SNTEXT1);
                        /* PACK HEADER */
145   3                 DBOUTBUF(B3) = SNTEXT1(B3);
146   3                 END;
147   2               B1 = 13;
148   2               CALL PACKADR(SNADR);
                        /* PACK SNAPSHOT ADDRESS */
149   2               B1 = 19;
150   2               DO B3 = 0 TO INSTRL - 1;
                        /* PACK INSTRUCTION */
151   3                 CALL PACKBYTE(SNEXTBL(B3+8));
152   3                 B1 = B1 + 1;
153   3                 END;
154   2               B1 = B1 - 1;
155   2               A1 = .DBOUTBUF(0);
156   2               CALL DBSYNCPRH;
                        /* TYPE FIRST LINE */
157   2               DBTELLBACK = TRUE;
```

```
158   2             IF REGFL THEN
                       /* TYPE REGISTER CONTENTS */
159   2               DO;
160   3               CALL DBNEWLINEH;
161   3               DO B3 = 0 TO LAST(SNTEXT3);
162   4                 DBOUTBUF(B3) = SNTEXT3(B3);
163   4               END;
164   3               B1 = 6;
165   3               CALL PACKBYTE(SNREG(1));
                          /* PACK A */
166   3               B1 = 11;
167   3               DO B3 = 2 TO 7 BY 2;
                          /* PACK B,C,D,E,H,L */
168   4                 CALL PACKBYTE(SNREG(B3+1));
169   4                 B1 = B1 + 3;
170   4                 CALL PACKBYTE(SNREG(B3));
171   4                 B1 = B1 + 3;
172   4                 END;
173   3               CALL PACKBYTE(SNREG(10));
                          /* PACK M */
174   3               B1 = B1 + 7;
175   3               CALL PACKADR(SNREGA(4));
                          /* PACK TOP OF STACK */
176   3               B1 = B1 + 1;
177   3               B3 = SNREG(0);
178   3               IF (B3 AND 1) <> 0 THEN
                          /* PACK CARRY */
179   3                 DO;
180   4                 DBOUTBUF(B1) = 'C';DBOUTBUF(B1+1) = 'Y';
182   4                 B1 = B1 + 3;
183   4                 END;
184   3               IF (B3 AND 10000000B) <> 0 THEN
```

```
                            /* PACK SIGN */
185   3                     DO;
186   4                     DBOUTBUF(B1) = 'S';B1 = B1 + 2;
188   4                     END;
189   3                   IF (B3 AND 01000000B) <> 0 THEN
                            /* PACK ZERO */
190   3                     DO;
191   4                     DBOUTBUF(B1) = 'Z';B1 = B1 + 2;
193   4                     END;
194   3                   IF (B3 AND 4) <> 0 THEN
                            /* PACK PARITY */
195   3                     DO;
196   4                     DBOUTBUF(B1) = 'P';B1 = B1 + 2;
198   4                     END;
199   3                   IF (B3 AND 00010000B) <> 0 THEN
                            /* PACK AUX. CARRY */
200   3                     DO;
201   4                     DBOUTBUF(B1) = 'A';DBOUTBUF(B1+1) = 'C';B1 = B1 + 2;
204   4                     END;
205   3                   A1 = .DBOUTBUF(0);
206   3                   CALL DBSYNCPRH;
                            /* TYPE AND CONTINUE AT TB0 WHEN TYPING COMPLETED */
207   3                   RETURN;
208   3                   END;
209   2       TB0:          IF SNVARX <> 0 THEN
                          /* TYPE REQUESTED VARIABLES */
210   2                   DO;
211   3                   CALL DBNEWLINEH;
212   3                   DBOUTBUF(0) = 1;DBOUTBUF(1) = 0;
214   3                   DO B3 = 2 TO 65;
215   4                     DBOUTBUF(B3) = ' ';
216   4                     END;
```

```
217   3                 B1 = 5;
218   3                 DO B3 = 0 TO SNVARX-1;
                            /* TYPE CONTENTS OF VARIABLES */
219   4                   CALL PACKADR(SNVAR(B3).ADR);
220   4                   DBOUTBUF(B1) = ':';B1 = B1 + 2;
222   4                   IF SNVAR(B3).BYTETYPE
                            THEN DO;
                                /* BYTE VARIABLE */
224   5                       DBOUTBUF(B1-7) = 'B';
225   5                       CALL PACKBYTE(SNVAR(B3).CONTB);
226   5                       END;
227   4                     ELSE DO;
                                /* ADDRESS VARIABLE */
228   5                       DBOUTBUF(B1-7) = 'A';
229   5                       CALL PACKADR(SNVAR(B3).CONTA);
230   5                       END;
231   4                   B1 = B1 + 2;
232   4                   END;
233   3                 A1 = .DBOUTBUF(0);
234   3                 CALL DBSYNCPRH;
                            /* TYPE AND CONTINUE AT TB1 WHEN TYPING COMPLETED */
235   3                 RETURN;
236   3                 END;
237   2       TB1:        SNA1 = DUMPBEG;
238   2                 IF NOT DUMPFL THEN GO TO TB3;
                            /* NO DUMP REQUESTED */
240   2                 CALL INITDUMP;
241   2                 CALL DBNEWLINEH;
242   2                 DBOUTBUF(0) = 2;
243   2                 CALL DUMP;
244   2                 RETURN;
245   2       TB2:          IF NOT DUMPDONE THEN
```

```
 246   2                    DO;
 247   3                    CALL DBNEWLINEH;
 248   3                    DBOUTBUF(0) = 3;
 249   3                    CALL DUMP;
 250   3                    RETURN;
 251   3                    END;
 252   2       TB3:           DUMPBEG = SNA1;
 253   2                   IF SNCTR <> 0 THEN
                               /* TYPE NUMBER OF SKIPPED SNAPSHOTS */
 254   2                    DO;
 255   3                    CALL DBNEWLINEH;
 256   3                    DO B3 = 0 TO LAST(SNTEXT4);
 257   4                      DBOUTBUF(B3) = SNTEXT4(B3);
 258   4                      END;
 259   3                    B1 = 4;
 260   3                    CALL PACKBYTE(SNCTR);
 261   3                    A1 = .DBOUTBUF(0);
 262   3                    B1 = LENGTH(SNTEXT4);
 263   3                    CALL DBSYNCPRH;
 264   3                    SNCTR = 0;
 265   3                    END;
 266   2       TB4:           SNAPFL = FALSE;
 267   2                   DBTELLBACK = FALSE;
 268   2                   CALL DBNEWLINEH;
 269   2                   RETURN;
 270   2                   END SNAPSHOT;
                     /*
                     ***************************************************************************
                     ***************************************************************************
                     **
                                                                                    DBSNAP
```

```
                        INITIATE SNAPSHOT
                    **
                    ************************************************************************
                    */
 271    1       DBSNAP:   PROCEDURE PUBLIC;

 272    2                 IF DBTELLBACK THEN
                              /* CALL CAUSED BY TELLBACK MSG */
 273    2                   DO;
 274    3                   CALL SNAPSHOT;
 275    3                   RETURN;
 276    3                   END;
 277    2                 CALL CLEARDATA(.SNCLEARBEG,.SNCLEAREND);
 278    2                 SNEXTBL(9) = 0;SNEXTBL(10) = 0;
                              /* RESET SNAPSHOT DATA */
 280    2                 B1 = B1 + 1;
 281    2                 MSGDATL = MSG(ML) - 5;
 282    2                 IF NOT GETINP THEN GO TO SNILL;
                              /* NON LEGAL SNAP ADDRESS */
 284    2                 SNADR = DBIN;
                              /* SET SNAP ADDRESS */
 285    2                 DO WHILE B1 <= MSGDATL;
                              /* DECODE INPUT */
 286    3                   CALL DEBLK;
 287    3                   B3 = MSGDATA(B1);
 288    3                   DO B2 = 0 TO LAST(SNDECTBL);
                                /* FIND INPUT PARAMETER CODE */
 289    4                     IF SNDECTBL(B2) = B3 THEN GOTO SN1;
 291    4                     END;
 292    3                   GO TO SNILL;
 293    3       SN1:              DO CASE B2;
 294    4                     DO;
```

```
                                    /* 'R' PARAMETER */
295   5                           REGFL = TRUE;
296   5                           B1 = B1 + 1;
297   5                           END;
298   4                         DO;
                                    /* 'M' PARAMETER */
299   5                           B1 = B1 + 1;
300   5                           IF NOT GETINP THEN GO TO SNILL;
302   5                           SNCONDADR = DBIN;
                                      /* SET CONDITION ADDRESS */
303   5                           CALL DEBLK;
304   5                           IF MSGDATA(B1) <> '-' THEN GO TO SNILL;
306   5                           B1 = B1 + 1;
307   5                           CALL DEBLK;
308   5                           IF NOT GETINP THEN GO TO SNILL;
310   5                           IF DBIN > 0FFH THEN GO TO SNILL;
                                      /* CONDITION NOT CORRECT */
312   5                           SNCONDSAVE = DBINH;
                                      /* SET CONDITION */
313   5                           CONDFL = TRUE;
314   5                           END;
315   4                         DO;
                                    /* 'D' PARAMETER */
316   5                           IF NOT DUMPDECODE THEN GOTO SNILL;
                                      /* INCORRECT DUMP INPUT */
318   5                           SNNUMDUMP = DUMPEND - DUMPBEG;
319   5                           IF SNNUMDUMP > LAST(SNDUMP) THEN
                                      /* DUMP RANGE TOO LARGE */
320   5                             DO;
321   6                             SNNUMDUMP = LAST(SNDUMP);
322   6                             DUMPEND = DUMPBEG + LAST(SNDUMP);
323   6                             END;
```

```
 324   5                    DUMPFL = TRUE;
 325   5                    END;
 326   4                  GO TO SNAB;
                             /* 'A' PARAMETER */

 327   4       SNAB:         DO;
                             /* 'B' PARAMETER */
 328   5                    IF SNVARX > LAST(SNVAR) THEN GO TO SN2;
                               /* MORE VARIABLES THAN ALLOWED */
 330   5                    SNVAR(SNVARX).BYTETYPE = MSGDATA(B1) - 'A';
                               /* SET VARIABLE TYPE */
 331   5                    B1 = B1 + 1;
 332   5                    IF NOT GETINP THEN GO TO SNILL;
 334   5                    SNVAR(SNVARX).ADR = DBIN;
                               /* SET ADDRESS OF VARIBLE */
 335   5                    SNVARX = SNVARX + 1;
 336   5                    END;
 337   4                  END;       /* DO CASE B2 */
 338   3       SN2:          B1 = B1 + 1;
 339   3                 END;       /* DO WHILE B1 <= MSGDATL */
 340   2               B3 = SNADRCONT(0);
 341   2               DO B2 = 0 TO LAST (INSTRTBL);
                          /* DETERMINE LENGTH OF SNAPSHOT INSTRUCTION */
 342   3                 IF INSTRTBL(B2) = B3 THEN
                             /* INSTR FOUND, SET LENGTH */
 343   3                   DO;
 344   4                   IF B2 < 18
                             THEN INSTRL = 2;
 346   4                     ELSE INSTRL = 3;
 347   4                   GO TO SN3;
 348   4                   END;
 349   3                 END;
```

```
 350   2                INSTRL = 1;
 351   2       SN3:         DO B2 = 0 TO INSTRL - 1;
                            /* MOVE SNAP INSTR INTO EXECUTION TABLE */
 352   3                  SNEXTBL(B2+8) = SNADRCONT(B2);
 353   3                  END;
 354   2                SNEXTBL(11) = 0C3H;
                            /* INSERT JUMP INSTRUCTION */
 355   2                SNEXTBLA(6) = SNADR + INSTRL;
                            /* SET ADDRESS OF INSTR AFTER SNAP INSTR */
 356   2                SNADRCONT(0) = 0E7H;
                            /* INSERT RST0 INSTR AT SNAP ADDRESS */
 357   2                DBFUNCTSAVE = DBFUNCTION;
 358   2                PERFUNCTION = TRUE;
 359   2                CALL DBNEWLINE;
 360   2                RETURN;
 361   2       SNILL:       CALL DBILLEGAL;
 362   2                RETURN;
 363   2                END DBSNAP;
                /*
                **************************************************************************
                **************************************************************************
                **
                                                                        SNAPTERM

                    TERMINATE SNAPSHOT FUNCTION
                **
                **************************************************************************
                */
 364   1       SNAPTERM:    PROCEDURE PUBLIC;

 365   2                DO B1 = 0 TO INSTRL-1;
                            /* RESET SNAPSHOT INSTRUCTION */
```

```
366   3                 SNADRCONT(B1) = SNEXTBL(B1+8);
367   3                 END;
368   2               SNAPFL = FALSE;
369   2               RETURN;
370   2               END SNAPTERM;
                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                          SNAPINT

                    PROCESS SNAPSHOT INTERRUPT
                    IF SNAPSHOT IS ACTIVE THIS PROCEDURE IS EXECUTED WITH RST4
                **
                ***************************************************************************
                */
371   1       SNAPINT:    PROCEDURE PUBLIC;

372   2               IF SNFL = 0 THEN
                            /* FIRST CALL */
373   2                 DO;
374   3                 SNAPINTADR = PROCADR;
375   3                 RETURN;
376   3                 END;
377   2               IF CONDFL AND SNCONDSAVE <> SNCOND THEN GO TO INTRET;
                            /* SPECIFIED CONDITION IS NOT MET */
379   2               IF SNAPFL THEN
                            /* ALREADY A SNAPSHOT IN PROGRESS */
380   2                 DO;
381   3                 SNCTR = SNCTR + 1;
382   3                 GO TO INTRET;
383   3                 END;
```

```
 384   2                SNAPFL = TRUE;
 385   2                SNB1 = 0;
 386   2                IF SNVARX <> 0 THEN
                            /* SAVE SPECIFIED VARIABLES */
 387   2                  DO WHILE SNB1 < SNVARX;
 388   3                    SNA1 = SNVAR(SNB1).ADR;
 389   3                    IF SNVAR(SNB1).BYTETYPE
                              THEN SNVAR(SNB1).CONTB = SNCONTBYTE;
 391   3                              ELSE SNVAR(SNB1).CONTA = SNCONTADR;
 392   3                    SNB1 = SNB1 + 1;
 393   3                    END;
 394   2                IF DUMPFL THEN
                            /* SAVE DUMP AREA */
 395   2                  DO SNB1 = 0 TO SNNUMDUMP;
 396   3                    SNDUMP(SNB1) = DUMPDATA(SNB1);
 397   3                    END;
 398   2                IF REGFL THEN
                            /* SAVE REGISTERS */
 399   2                  DO;
 400   3                  SNA2 = STACKPTR;
 401   3                  DO SNB2 = 0 TO 7;
                              /* SAVE PSW,B,D,H */
 402   4                  SNREG(SNB2) = SNSTACK(SNB2);
 403   4                  END;
 404   3                SNREG(8) = SNSTACK(10);SNREG(9) = SNSTACK(11);
 406   3                SNA1 = SNREGA(3);
                            /* H/L */
 407   3                SNREG(10) = SNCONTBYTE;
                            /* SAVE M */
 408   3                END;
 409   2                CALL PRIORITYINT(SNPRIORX);
 410   2      INTRET:       ;
```

```
411   2                 SNA1 = .SNEXTBL(0);
412   2                 CALL SNA1;
                            /* EXECUTE REPLACED INSTRUCTION AND RETURN TO
                               INTERRUPTED PROGRAM */
413   2                 RETURN;
414   2                 END SNAPINT;
415   1       END DBMOD5;


MODULE INFORMATION:

     CODE AREA SIZE      = 075DH   1885D
     VARIABLE AREA SIZE  = 0000H      0D
     MAXIMUM STACK SIZE  = 0004H      4D
     968 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD6
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB6.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            DBMOD6:    DO;
                $ INCLUDE (:F1:DBDATE.SRC)
            =   /*
            =   ***************************************************************************       /
            =   ***************************************************************************
            =   **
            =   **                      DEBUG MODULE
            =   **
            =   ***************************************************************************
            =   **
            =   **                MODULE IDENTIFICATION : DB
            =   **                MODULE NUMBER         : 07
            =   **
            =   ***************************************************************************
            =   */
            =   $ NOLIST

 104    1       DBSYNCPRH:  PROCEDURE EXTERNAL;
 105    2               END;
 106    1       DBNEWLINEH: PROCEDURE EXTERNAL;
 107    2               END;
 108    1       DBSYNCPR:   PROCEDURE EXTERNAL;
 109    2               END;
 110    1       DBPROMPT:   PROCEDURE EXTERNAL;
 111    2               END;
 112    1       GETINP:     PROCEDURE BYTE EXTERNAL;
```

```
113   2             END;
114   1       DBILLEGAL:  PROCEDURE EXTERNAL;
115   2             END;
116   1       DBNEWLINE:  PROCEDURE EXTERNAL;
117   2             END;
118   1       DBINPREQ:   PROCEDURE EXTERNAL;
119   2             END;
120   1       PACKBYTE:   PROCEDURE (P1) EXTERNAL;
121   2       DCL     P1 BYTE;
122   2             END;

              /*
              ***************************************************************
              ***************************************************************
              **
                                                              DBMSXINP

                  MSG SIMULATION AND MSG EXTRACTION
              **
              ***************************************************************
              */
123   1       DBMSXINP:       PROCEDURE PUBLIC;

124   2             DO CASE DBMSWITCH;
                        /* BRANCH TO PROCESSING OF INPUT */
125   3               GO TO DBM0;
126   3               GO TO DBM1;
127   3               GO TO DBM1;
128   3               GO TO DBM1;
129   3               GO TO DBM1;
130   3               GO TO DBM2;
131   3               END;
```

```
 132   2       DBM0:       IF MSGSIM AND MSG(ML) = 4 THEN GO TO DBMSEND;
                           /* CONTINUATION OF MSG SIMULATION, REPEAT MSG*/
 134   2                 MSGSIM = FALSE; MSGEX = FALSE;
 136   2                 B1 = B1 + 1;
 137   2                 DBFUNCTSAVE = DBFUNCTION;
 138   2                 IF MSGDATA(B1) = 'S'
                             THEN MSGSIM = TRUE;
                                 /* MSG SIMULATION */
 140   2                   ELSE IF MSGDATA(B1) = 'X'
                                     THEN MSGEX = TRUE;
                                         /* MSG EXTRACTION */
 142   2                                 ELSE DO;
 143   3                                 CALL DBILLEGAL;
 144   3                                 RETURN;
 145   3                                 END;
 146   2                 DO B3 = 0 TO LAST(DEBUGMCB);
                             /* RESET MCB */
 147   3                   DEBUGMCB(B3) = 0FFH;
 148   3                   END;
 149   2                 MSGDATAX = 0;
 150   2                 DBMSWITCH = 1;
 151   2                 GO TO DBM11;

 152   2       DBM1:       B1 = 0;
 153   2                 IF MSGEX AND MSG(ML) = 4
                             /* INPUT CR ONLY */
                           THEN DO; END;
 156   2                   ELSE IF NOT GETINP OR DBIN > 0FFH OR
                                   (DBMSWITCH = 4 AND DBIN > LENGTH(DBMSGDATA) +
           -     4)
                                 THEN DO;
```

```
 158   3                          B1 = 2; A1 = .(0,'?');
 160   3                          CALL DBSYNCPR;
 161   3                          GO TO DBM11;
 162   3                          END;
 163   2                        ELSE DEBUGMCB(DBMSWITCH-1) = DBIN;
 164   2                DBMSWITCH = DBMSWITCH + 1;
 165   2                IF DBMSWITCH = 5 THEN
                            /* INPUT MCB COMPLETED */
 166   2                  DO;
 167   3                  IF MSGEX THEN
                              /* MSG EXTRACTION */
 168   3                    DO;
 169   4                    DBMSWITCH = 0;
 170   4                    IF NOT SEND(.EXTRACTSTART(0)) THEN RETURN;
 172   4                    CALL DBNEWLINEH;
 173   4                    DBEXTRFL = FALSE;
 174   4                    DBMSGCOUNT = 0;
 175   4                    PERFUNCTION = TRUE;
 176   4                    RETURN;
 177   4                    END;
 178   3                  IF DEBUGMCB(ML) > 4
                              /* INPUT DATA BYTES */
                            THEN CALL DBNEWLINE;
 180   3                    ELSE GO TO DBMSEND;
 181   3                  END;
 182   2       DBM11:     A4 = MSGTEXTX(DBMSWITCH);
 183   2                  B1 = BL;
 184   2                  A1 = .MSGTEXT(BU);

 185   2       DBM3:          CALL DBSYNCPR;
 186   2                  B1 = FALSE;
 187   2                  CALL DBINPREQ;
```

```
188   2                 RETURN;

189   2       DBM2:       IF NOT GETINP OR DBIN > 0FFH THEN
                          /* ILLEGAL DATA BYTE INPUT */
190   2                   DO;
191   3                   B1 = 3; A1 = .(0,'?:');
193   3                   GO TO DBM3;
194   3                   END;
195   2                 DBMSGDATA(MSGDATAX) = DBIN;
196   2                 IF MSGDATAX >= DEBUGMCB(ML) - 5 THEN GOTO DBMSEND;
                            /* INPUT DATA BYTES COMPLETED */
198   2                 MSGDATAX = MSGDATAX + 1;
199   2                 B1 = 3; A1 = .(0,' :');
201   2                 GO TO DBM3;

202   2       DBMSEND:    IF NOT SEND(.DEBUGMCB(0)) THEN
                          /* SEND MSG */
203   2                   DO;
204   3                   CALL DBNEWLINE;
205   3                   B1 = 15; A1 = .(0,'  MSG NOT SENT');
207   3                   GO TO DBMSEND1;
208   3                   END;
209   2                 IF DEBUGMCB(ML) > 4 THEN
                            /* TRANSFER DATA BYTES */
210   2                   DO B3 = 0 TO MSGDATAX;
211   3                     MSGDATA(B3) = DBMSGDATA(B3);
212   3                     END;
213   2                 CALL DBNEWLINE;
214   2                 B1 = 11; A1 = .(0,'  MSG SENT');
216   2       DBMSEND1:   CALL DBSYNCPR;
217   2                 DBMSWITCH = 0;
218   2                 CALL DBPROMPT;
```

```
219   2             RETURN;
220   2             END DBMSXINP;
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                        DBEXTRACT

                    PROCESS EXTRACTION MSG FROM EXEC
                **
                ****************************************************************************
                */
221   1       DBEXTRACT:      PROCEDURE PUBLIC;

222   2       DCL     EXTRMSG (100) BYTE;

223   2               IF NOT DBTELLBACK THEN
                          /* FIRST CALL */
224   2                 DO;
225   3                 B2 = MSGDATA(ML) - 1;
226   3                 DO B1 = 0 TO B2;
                              /* SAVE EXTRACTED MSG */
227   4                   EXTRMSG(B1) = MSGDATA(B1);
228   4                   END;
229   3                 DBTBCODE = 0;
230   3                 END;
231   2               DO CASE DBTBCODE;
232   3                 GO TO EXTR0;
233   3                 GO TO EXTR1;
234   3                 GO TO EXTR2;
235   3                 GO TO EXTR3;
236   3                 END;
```

```
 237   2      EXTR0:      DBEXTRFL = TRUE;
 238   2                DO B2 = 0 TO LAST(MSGTEXT);
 239   3                  DBOUTBUF(B2) =  MSGTEXT(B2);
 240   3                  END;
 241   2                DO B2 = 2 TO 5;
                             /* PACK RMN, SMN, MN, ML */
 242   3                  A4 = MSGTEXTX(B2);
 243   3                  B1 = BU - 2;
 244   3                  CALL PACKBYTE(EXTRMSG(B2-2));
 245   3                  END;
 246   2                IF EXTRMSG(ML) = 4 THEN
                             /* NO DATA BYTES */
 247   2                  DO;
 248   3                  DBOUTBUF(0) = 2;
 249   3                  GO TO EXTR12;
 250   3                  END;
 251   2                B1 = 34;EXTRIND = 4;
 253   2                GO TO EXTR11;

 254   2      EXTR1:    IF HARDCOPY
                           THEN DO;
 256   3                    CALL DBNEWLINEH;
 257   3                    B1 = 2;
 258   3                    END;
 259   2                  ELSE DO;
 260   3                    DBOUTBUF(2) = 0CH;DBOUTBUF(3) = 64;DBOUTBUF(4) = 119;
                                /* CURSOR TO CHARACTER POSITION 36 ON INPUT LINE *
                         -   /
 263   3                    B1 = 5;
 264   3                    END;
```

```
265   2       EXTR11:     B3 = 0;
266   2                 DO WHILE EXTRIND < EXTRMSG(ML);
                            /* TYPE ALL DATA BYTES */
267   3                   CALL PACKBYTE(EXTRMSG(EXTRIND));
268   3                   DBOUTBUF(B1) = ' '; B1 = B1 + 1;
                              /* TYPE 1 DATA BYTE */
270   3                   B3 = B3 + 1;
271   3                   EXTRIND = EXTRIND + 1;
272   3                   IF B3 > 0FH THEN GO TO EXTR12;
                             /* 10H DATA BYTES TYPED ON 1 LINE */
274   3                   END;

275   2                 DBOUTBUF(0) = 2;
                            /* SET TELLBACK CODE */
276   2       EXTR12:     A1 = .DBOUTBUF(0);
277   2                 DBTELLBACK = TRUE;
278   2                 CALL DBSYNCPRH;
279   2                 RETURN;

280   2       EXTR2:      IF DBMSGCOUNT = 0 THEN GO TO EXTR3;
                            /* SAME MSG NOT OCCURRED DURING THIS PROCESS TIME */
282   2                 CALL DBNEWLINEH;
283   2                 DO B2 = 0 TO LAST(SKIPMSG);
284   3                   DBOUTBUF(B2) = SKIPMSG(B2);
285   3                   END;
286   2                 B1 = 3;
287   2                 CALL PACKBYTE(DBMSGCOUNT);
                            /* TYPE NUMBER OF SKIPPED MSG */
288   2                 B1 = LENGTH(SKIPMSG); A1 = .DBOUTBUF(0);
290   2                 CALL DBSYNCPRH;
291   2                 DBMSGCOUNT = 0;
292   2                 RETURN;
```

```
293   2       EXTR3:      DBTELLBACK = FALSE;
294   2                 DBEXTRFL = FALSE;
295   2                 CALL DBNEWLINEH;
296   2                 RETURN;
297   2                 END DBEXTRACT;
                  /*
                  ***************************************************************************
                  ***************************************************************************
                  **
                                                                              DBMSX

                      PROCESS MS AND MX INPUTS
                  **
                  ***************************************************************************
                  */
298   1       DBMSX:      PROCEDURE PUBLIC;

299   2                 IF DBTELLBACK
                            /* TELLBACK MSG RECEIVED */
                        THEN CALL DBEXTRACT;
301   2                   ELSE CALL DBMSXINP;
302   2                 RETURN;
303   2                 END DBMSX;
304   1       END DBMOD6;
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 0463H   1123D
     VARIABLE AREA SIZE = 0064H    100D
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD7
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB7.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1             DBMOD7:   DO;
                $INCLUDE(:F1:DBDATE.SRC)
            =   /*
            =   ***************************************************************************
            =   ***************************************************************************
            =   **
            =   **                      DEBUG MODULE
            =   **
            =   ***************************************************************************
            =   **
            =   **                MODULE IDENTIFICATION : DB
            =   **                MODULE NUMBER         : 07
            =   **
            =   ***************************************************************************
            =   */
            =   $ NOLIST

104   1       DBNEWLINE:  PROCEDURE EXTERNAL;
105   2               END;
106   1       DBSYNCPR:   PROCEDURE EXTERNAL;
107   2               END;
108   1       DBINPREQ:   PROCEDURE EXTERNAL;
109   2               END;

                /*
                ***************************************************************************
```

```
                ************************************************************************
                **
                                                                    DBSYSTINP

                            TRANSFER INPUT INTO DBSTBUF
                            RETURN FALSE IF INPUT NON NUMERICAL
                                   TRUE OTHERWISE
                **
                ************************************************************************
                */
110    1        DBSYSTINP:  PROCEDURE BYTE;

111    2        DCL     SYSTILL (3) BYTE DATA (' ? ');

112    2                B2 = MSG(ML) - 5;
113    2                DO B1 = 0 TO B2;
114    3                  B3 = MSGDATA(B1);
115    3                  IF B3 >= 'A'
                            THEN B3 = B3 - 37H;
117    3                            ELSE B3 = B3 - '0';
118    3                  IF B3 > 0FH THEN
119    3                    DO;
120    4                    A1 = .SYSTILL(0);
121    4                    B1 = LENGTH(SYSTILL);
122    4                    CALL DBSYNCPR;
123    4                    RETURN FALSE;
124    4                    END;
125    3                  DBSTBUF(DBSTBUFX) = B3;
126    3                  DBSTBUFX = DBSTBUFX + 1;
127    3                  END;
128    2                RETURN TRUE;
129    2                END DBSYSTINP;
```

```
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        DBSYSTEST

                        PROCESS SYSTEM TEST INPUTS
                **
                **********************************************************************
                */
130   1         DBSYSTEST:  PROCEDURE PUBLIC;

131   2                     DO CASE DBSYSTSW;
132   3                       GO TO SW0;
133   3                       GO TO SW1;
134   3                       GO TO SW2;
135   3                       GO TO SW3;
136   3                       END;

137   2         SW0:            PERFUNCTION = TRUE;
138   2                       DBFUNCTSAVE = DBFUNCTION;
139   2         SW01:           CALL DBNEWLINE;
140   2                       DBSTBUFX = 0;
141   2                       DBSYSTSW = 1;
142   2         SW02:           A1 = .SYSTTEXT1(0);
143   2                       B1 = LENGTH(SYSTTEXT1);
                                /* TYPE '  FUNCTION:' */
144   2                       GO TO SW4;

145   2         SW1:            IF NOT DBSYSTINP OR MSG(ML) = 4 THEN GOTO SW11;
147   2         SW11:           DBSYSTSW = 2;
148   2                       A1 = .SYSTTEXT2(0);
```

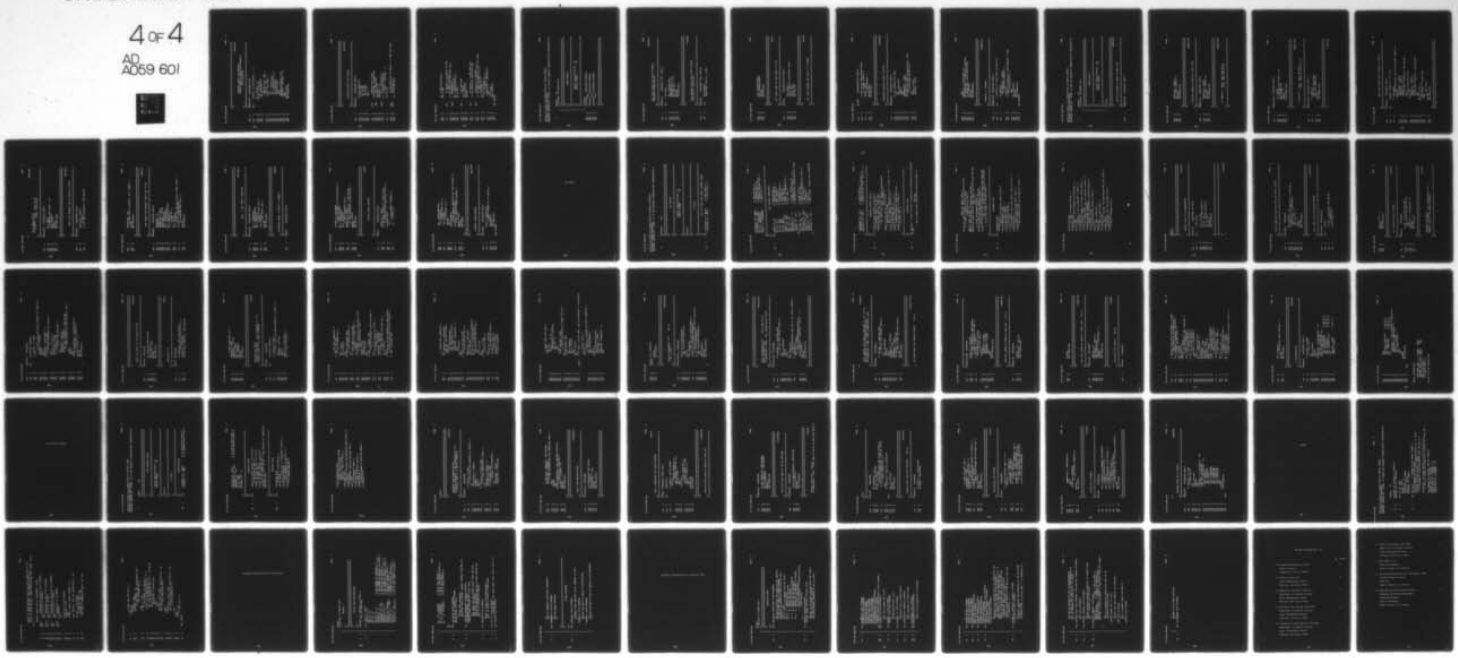
```
 149   2                 B1 = LENGTH(SYSTTEXT2);
 150   2                 GO TO SW4;

 151   2       SW2:          IF MSG(ML) = 4 THEN GO TO SW5;
                             /* INPUT CR ONLY */
 153   2                 IF NOT DBSYSTINP THEN GOTO SW11;
 155   2       SW21:         DBSYSTSW = 3;
 156   2                 A1 = .SYSTTEXT3(0);
 157   2                 B1 = LENGTH(SYSTTEXT3);
 158   2                 GO TO SW4;

 159   2       SW4:          CALL DBSYNCPR;
 160   2                 B1 = FALSE;
 161   2                 CALL DBINPREQ;
 162   2                 RETURN;

 163   2       SW3:          IF MSG(ML) = 4 THEN GOTO SW5;
 165   2                 IF NOT DBSYSTINP THEN GOTO SW21;

 167   2       SW5:          SYSTMSG(ML) = DBSTBUFX + 4;
 168   2                 IF NOT SEND(.SYSTMSG(0)) THEN GOTO SW01;
                             /* SEND MSG 50H TO T2 */
 170   2                 B2 = DBSTBUFX - 1;
 171   2                 DO B1 = 0 TO B2;
                             /* TRANSFER DATA */
 172   3                   MSGDATA(B1) = DBSTBUF(B1);
 173   3                   END;
 174   2                 GO TO SW31;
 175   2                 END DBSYSTEST;
 176   1       END DBM007;
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD7
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB7.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


                      '
  1                   DBMOD7:   DO;
                      $INCLUDE(:F1:DBDATE.SRC)
                =     /*
                =     ****************************************************************
                =     ****************************************************************
                =     **
                =     **                    DEBUG MODULE
                =     **
                =     ****************************************************************
                =     **
                =     **              MODULE IDENTIFICATION : DB
                =     **              MODULE NUMBER         : 07
                =     **
                =     ************************************************************
                =     */
                =     $ NOLIST

 104    1             DBNEWLINE:  PROCEDURE EXTERNAL;
 105    2                         END;
 106    1             DBSYNCPR:   PROCEDURE EXTERNAL;
 107    2                         END;
 108    1             DBINPREQ:   PROCEDURE EXTERNAL;
 109    2                         END;

                      /*
                      ****************************************************************
```

AD-A059 601 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
A REAL-TIME OPERATING SYSTEM FOR SINGLE BOARD COMPUTER BASED DI--ETC(U)
JUN 78 W NIEMANN

UNCLASSIFIED NL

4 OF 4
AD
A059 601



END
DATE
FILMED
12-78
DDC

```
                /*****************************************************************
                **
                                                              DBSYSTINP

                        TRANSFER INPUT INTO DBSTBUF
                        RETURN FALSE IF INPUT NON NUMERICAL
                               TRUE OTHERWISE
                **
                *****************************************************************
                */
  110   1       DBSYSTINP:  PROCEDURE BYTE;

  111   2       DCL     SYSTILL (3) BYTE DATA (' ? ');

  112   2               B2 = MSG(ML) - 5;
  113   2               DO B1 = 0 TO B2;
  114   3                 B3 = MSGDATA(B1);
  115   3                 IF B3 >= 'A'
                            THEN B3 = B3 - 37H;
  117   3                            ELSE B3 = B3 - '0';
  118   3                 IF B3 > 0FH THEN
  119   3                   DO;
  120   4                   A1 = .SYSTILL(0);
  121   4                   B1 = LENGTH(SYSTILL);
  122   4                   CALL DBSYNCPR;
  123   4                   RETURN FALSE;
  124   4                   END;
  125   3                 DBSTBUF(DBSTBUFX) = B3;
  126   3                 DBSTBUFX = DBSTBUFX + 1;
  127   3                 END;
  128   2               RETURN TRUE;
  129   2               END DBSYSTINP;
```

```
                /*
                ************************************************************************
                ************************************************************************
                **
                                                                        DBSYSTEST

                        PROCESS SYSTEM TEST INPUTS
                **
                ************************************************************************
                */
  130   1       DBSYSTEST:  PROCEDURE PUBLIC;

  131   2                 DO CASE DBSYSTSW;
  132   3                   GO TO SW0;
  133   3                   GO TO SW1;
  134   3                   GO TO SW2;
  135   3                   GO TO SW3;
  136   3                   END;

  137   2       SW0:            PERFUNCTION = TRUE;
  138   2                   DBFUNCTSAVE = DBFUNCTION;
  139   2       SW01:           CALL DBNEWLINE;
  140   2                   DBSTBUFX = 0;
  141   2                   DBSYSTSW = 1;
  142   2       SW02:           A1 = .SYSTTEXT1(0);
  143   2                   B1 = LENGTH(SYSTTEXT1);
                                /* TYPE '  FUNCTION:' */
  144   2                   GO TO SW4;

  145   2       SW1:            IF NOT DBSYSTINP OR MSG(ML) = 4 THEN GOTO SW11;
  147   2       SW11:           DBSYSTSW = 2;
  148   2                   A1 = .SYSTTEXT2(0);
```

```
149   2                 B1 = LENGTH(SYSTTEXT2);
150   2                 GO TO SW4;

151   2       SW2:          IF MSG(ML) = 4 THEN GO TO SW5;
                            /* INPUT CR ONLY */
153   2                 IF NOT DBSYSTINP THEN GOTO SW11;
155   2       SW21:         DBSYSTSW = 3;
156   2                 A1 = .SYSTTEXT3(0);
157   2                 B1 = LENGTH(SYSTTEXT3);
158   2                 GO TO SW4;

159   2       SW4:          CALL DBSYNCPR;
160   2                 B1 = FALSE;
161   2                 CALL DBINPREQ;
162   2                 RETURN;

163   2       SW3:          IF MSG(ML) = 4 THEN GOTO SW5;
165   2                 IF NOT DBSYSTINP THEN GOTO SW21;

167   2       SW5:          SYSTMSG(ML) = DBSTBUFX + 4;
168   2                 IF NOT SEND(.SYSTMSG(0)) THEN GOTO SW01;
                            /* SEND MSG 50H TO T2 */
170   2                 B2 = DBSTBUFX - 1;
171   2                 DO B1 = 0 TO B2;
                            /* TRANSFER DATA */
172   3                   MSGDATA(B1) = DBSTBUF(B1);
173   3                   END;
174   2                 GO TO SW01;
175   2                 END DBSYSTEST;
176   1       END DBMOD7;
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD8
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB8.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            DBMOD8:     DO;
                $ INCLUDE(:F1:DBDATE.SRC)
            =   /*
            =   ***************************************************************************
            =   ***************************************************************************
            =   **
            =   **                      DEBUG MODULE
            =   **
            =   ***************************************************************************
            =   **
            =   **                  MODULE IDENTIFICATION : DB
            =   **                  MODULE NUMBER         : 07
            =   **
            =   ***************************************************************************
            =   */
            =   $ NOLIST
 104    1       DBILLEGAL:  PROCEDURE EXTERNAL;
 105    2                  END;
 106    1       DBPROMPT:   PROCEDURE EXTERNAL;
 107    2                  END;
 108    1       SNAPTERM:   PROCEDURE EXTERNAL;
 109    2                  END;
                /*
                ***************************************************************************
                ***************************************************************************
                **
```

PACKBYTE

```
                    PACK BYTE VALUE INTO OUTPUT BUFFER
                    STARTING AT POSITION B1
                **
                ************************************************************************
                */
  110   1       PACKBYTE:    PROCEDURE (BYT) PUBLIC;

  111   2       DCL      BYT BYTE;

  112   2                CALL CONVASC(BYT);
  113   2                DBOUTBUF(B1) = BU;
  114   2                DBOUTBUF(B1+1) = BL;
  115   2                B1 = B1 + 2;
  116   2                RETURN;
  117   2                END PACKBYTE;
                /*
                ************************************************************************
                ************************************************************************
                **
```

PACKADR

```
                    PACK ADDRESS VALUE INTO OUTPUT BUFFER
                    STARTING AT POSITION B1
                **
                ************************************************************************
                */
  118   1       PACKADR:     PROCEDURE (ADR) PUBLIC;

  119   2       DCL      ADR ADDRESS,
                           (ADRH, ADRL) BYTE AT (.ADR);
```

```
120   2             CALL PACKBYTE(ADRL);
121   2             CALL PACKBYTE(ADRH);
122   2             RETURN;
123   2             END PACKADR;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                        GETINP

                GET INPUT FROM CRT AND LEAVE IT IN A4
            **
            ***************************************************************************
            */
124   1       GETINP:     PROCEDURE BYTE PUBLIC;

125   2             CALL DEBLK;
126   2             IF NOT GETNUM THEN RETURN FALSE;
128   2             DBIN = A4;
129   2             RETURN TRUE;
130   2             END GETINP;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                        GETADR

                GET INPUT ADDRESS AND LEAVE IT IN DBADR
            **
            ***************************************************************************
            */
```

```
131   1     GETADR:     PROCEDURE BYTE PUBLIC;

132   2             IF NOT GETINP THEN RETURN FALSE;
                        /* ILLEGAL INPUT */
134   2             DBADR = DBIN;
                        /* SET ADDRESS */
135   2             RETURN TRUE;
136   2             END GETADR;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                        DBTERMPER

                TERMINATE PERIODIC DEBUG FUNCTION
            **
             ***************************************************************************
            */
137   1     DBTERMPER:  PROCEDURE PUBLIC;

138   2             IF NOT PERFUNCTION THEN
139   2               DO;
140   3               CALL DBILLEGAL;
141   3               RETURN;
142   3               END;
143   2             TERMPERFUNCT = TRUE;
144   2             PERFUNCTION = FALSE;
145   2             IF MSGEX AND SEND(.EXTRACTSTOP(0)) THEN
                        /* MSG EXTRACTION ACTIVE */
146   2               DO;
147   3               MSGEX = FALSE;
148   3               DBEXTRFL = TRUE;
```

```
149   3                 END;
150   2               DBMSWITCH = 0; DBSYSTSW = 0;
152   2               IF DBFUNCTION = 4 THEN CALL SNAPTERM;
154   2               IF DBFUNCTION = 0FFH THEN RETURN;
156   2               CALL DBPROMPT;
157   2               RETURN;
158   2               END DBTERMPER;
                /*
                **************************************************************************
                **************************************************************************
                **
                                                                        DBHARDCOPY

                    PROCESS 'H' COMMAND
                **
                **************************************************************************
                */
159   1       DBHARDCOPY: PROCEDURE PUBLIC;

160   2       DCL     LPINITMSG (4) BYTE DATA (LP, DB, 13, 4);

161   2               IF HARDCOPY
                          THEN HARDCOPY = FALSE;
                            /* TERMINATE HARDCOPY */
163   2                 ELSE DO;
164   3               IF SEND(.LPINITMSG(0)) THEN HARDCOPY = TRUE;
                            /* INITIALIZE LP */
166   3                 END;
167   2               CALL DBPROMPT;
168   2               RETURN;
169   2               END DBHARDCOPY;
170   1       END DBMOD8;
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE DBMOD9
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:DB9.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1             DBMOD9:     DO;
                $ INCLUDE(:F1:DBDATE.SRC)
          =     /*
          =     ***************************************************************
          =     ***************************************************************
          =     **
          =     **                      DEBUG MODULE
          =     **
          =     ***************************************************************
          =     **
          =     **                  MODULE IDENTIFICATION : DB
          =     **                  MODULE NUMBER         : 07
          =     **
          =     ***************************************************************
          =     */
          =     $ NOLIST
                /*
                ***************************************************************
                ***************************************************************
                **                                                    DBNEWLINE

                    POSITION CURSOR AT BEGIN OF NEW LINE
                **
                ***************************************************************
                */
  104   1       DBNEWLINE:  PROCEDURE PUBLIC;
```

```
105   2             DBMN16(0) = DBCRT;
106   2             B1 = SEND(.DBMN16(0));
107   2             RETURN;
108   2             END DBNEWLINE;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                    DBBEGLINE

                POSITION CURSOR AT BEGIN OF INPUT LINE
            **
            ***************************************************************************
            */
109   1       DBBEGLINE:  PROCEDURE PUBLIC;

110   2             DBMN17(0) = DBCRT;
111   2             B1 = SEND(.DBMN17(0));
112   2             RETURN;
113   2             END DBBEGLINE;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                    DBNEWLINEH

                IF HARDCOPY = TRUE : NEW LINE ON LP
                              FALSE: NEW LINE ON CRT
            **
            ***************************************************************************
            */
```

```
114   1       DBNEWLINEH: PROCEDURE PUBLIC;

115   2               B3 = DBCRT;
116   2               IF HARDCOPY THEN DBCRT = LP;
118   2               CALL DBNEWLINE;
119   2               DBCRT = B3;
120   2               RETURN;
121   2               END DBNEWLINEH;
              /*
              ***************************************************************
              ***************************************************************
              **
                                                          DBBEGLINEH

                  IF HARDCOPY = TRUE : NEW LINE ON LP
                               FALSE: BEGIN OF LINE ON CRT
              **
              ***************************************************************
              */
122   1       DBBEGLINEH: PROCEDURE PUBLIC;

123   2               IF HARDCOPY
                        THEN CALL DBNEWLINEH;
125   2                       ELSE CALL DBBEGLINE;
126   2               RETURN;
127   2               END DBBEGLINEH;
              /*
              ***************************************************************
              ***************************************************************
              **
                                                          DBSYNCPR
```

```
                        SEND SYNC PRINT MSG TO MODULE IN DBCRT
                  **
                  ************************************************************************
                  */
  128   1         DBSYNCPR:   PROCEDURE PUBLIC;

  129   2         DCL      (MSGTEXT BASED A1) (1) BYTE;

  130   2                  IF DBTELLBACK
                             /* SEND PRINTSYNC MSG WITH TELLBACK REQUEST */
                           THEN DO;
  132   3                     DBTELLBACK = FALSE;
  133   3                     DBMN18(0) = DBCRT;
  134   3                     DBMN18(3) = B1 + 4;
  135   3                     IF NOT SEND(.DBMN18(0)) THEN RETURN;
  137   3                     END;
                             /* SEND PRINTSYNC MSG */
  138   2                  ELSE DO;
  139   3                     DBMN11(0) = DBCRT;
  140   3                     DBMN11(3) = B1 + 4;
  141   3                     IF NOT SEND(.DBMN11(0)) THEN RETURN;
  143   3                     END;
  144   2                  B1 = B1 - 1;
  145   2                  DO B2 = 0 TO B1;
  146   3                    MSGDATA(B2) = MSGTEXT(B2);
  147   3                    END;
                             /* SEND MSG */
  148   2                  RETURN;
  149   2                  END DBSYNCPR;
                  /*
                  ************************************************************************
                  ************************************************************************
```

```
                **
                                                                DBSYNCPRH

                    PRINT DEBUG OUTPUT
                    IF HARDCOPY = TRUE : PRINT ON LP
                                 FALSE: TYPE ON CRT
                **
                ************************************************************************
                */
  150   1       DBSYNCPRH:  PROCEDURE PUBLIC;

  151   2               B3 = DBCRT;
  152   2               IF HARDCOPY THEN DBCRT = LP;
  154   2               CALL DBSYNCPR;
  155   2               DBCRT = B3;
  156   2               RETURN;
  157   2               END DBSYNCPRH;
                /*
                ************************************************************************
                ************************************************************************
                **
                                                                DBINPREQ

                        SEND INPUT REQUEST MSG TO MODULE IN DBCRT
                **
                ************************************************************************
                */
  158   1       DBINPREQ:   PROCEDURE PUBLIC;

  159   2               DBMN12(0) = DBCRT;
                            /* RMN */
  160   2               IF NOT SEND(.DBMN12(0)) THEN RETURN;
```

```
162   2             MSGDATA(0) = B1;
                        /* SEND MSG, B1 CONTROLS 'ROLL SCREEN' */
163   2             RETURN;
164   2             END DBINPREQ;
              /*
              ***************************************************************************
              ***************************************************************************
              **
                                                                        DBPROMPT

                        SEND PROMPT AND REQUEST INPUT MSG TO CRT
              **
              ***************************************************************************
              */
165   1       DBPROMPT:   PROCEDURE PUBLIC;

166   2             DBTBCODE = 0;
167   2             DBMSWITCH = 0;
168   2             DBFUNCTION = 0FFH;
169   2             DBTELLBACK = FALSE;
170   2             CALL DBNEWLINE;
171   2             A1 = .PROMPTMSG(0);
172   2             B1 = LENGTH(PROMPTMSG);
173   2             CALL DBSYNCPR;
                        /* SEND MSG */
174   2             B1 = FALSE;
175   2             CALL DBINPREQ;
                        /* SEND INP REQ MSG, NO ROLL AFTER INPUT */
176   2             CRTINPUT = DEBUGCOMMAND;
                        /* DETERMINE NEXT INPUT */
177   2             RETURN;
178   2             END DBPROMPT;
```

```
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        DBILLEGAL

                                SEND '?' AND PROMPT CHARACTER
                **
                **********************************************************************
                */
  179   1       DBILLEGAL:  PROCEDURE PUBLIC;

  180   2                   A1 = .ERRMSG(0);
  181   2                   B1 = LENGTH(ERRMSG);
  182   2                   CALL DBSYNCPR;
                                /* SEND ILLEGAL MSG */
  183   2                   CALL DBPROMPT;
                                /* SEND PROMPT MSG */
  184   2                   RETURN;
  185   2                   END DBILLEGAL;
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        DBTERM

                    TERMINATE DEBUG / PROCESS T COMMAND
                **
                **********************************************************************
                */
  186   1       DBTERM:     PROCEDURE PUBLIC;
```

```
187   2             CRTINPUT = INITDEBUG;
                        /* NEXT DB INPUT IS INITIALIZATION */
188   2             B1 = LENGTH(TERMMSG);
189   2             A1 = .TERMMSG(0);
190   2             CALL DBSYNCPR;
                        /* SEND TERMINATE TEXT */
191   2             DBMN14(0) = DBCRT;
192   2             IF NOT SEND(.DBMN14(0)) THEN RETURN;
                        /* SEND TERMINATE I/O MSG TO CRT */
194   2             DEBUG = FALSE;
195   2             RETURN;
196   2             END DBTERM;
295           /*
              ***************************************************************************
              ***************************************************************************
              **
                                                                              DBREQ

                              PROCESS DEBUG REQUEST

              **
              ***************************************************************************
              */
197   1       DBREQ:      PROCEDURE PUBLIC;

198   2             DBMN15(0) = DBCRT;
199   2             IF NOT SEND(.DBMN15(0)) THEN RETURN;
                        /* SEND ACTIVATE DEBUG MSG TO CRT */
201   2             IF DEBUG THEN
202   2               DO;
                          /* CPTR ALREADY DEBUGGED */
203   3               A1 = .ERRMSG1(0);
```

```
 204   3                B1 = LENGTH(ERRMSG1);
 205   3                CALL DBSYNCPR;
                            /* SEND ERROR MSG */
 206   3                CALL DBTERM;
                            /* TERMINATE I/O */
 207   3                RETURN;
 208   3                END;
 209   2              CALL DBPROMPT;
                          /* SEND PROMPT CHARACTER AND INPUT REQUEST */
 210   2              CRTINPUT = DEBUGCOMMAND;
                         /* NEXT INPUT IS DEBUG COMMAND */
 211   2              DEBUG = TRUE;
 212   2              RETURN;
 213   2              END DBREQ;
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                          DBEXTREQ

                    EXTERNAL DEBUG REQUEST
                **
                ****************************************************************************
                */
 214   1        DBEXTREQ:   PROCEDURE PUBLIC;

 215   2              DBCRT = MSGDATA(0);
                            /* MODULE NUMBER OF CRT MODULE */
 216   2              CALL DBREQ;
 217   2              RETURN;
 218   2              END DBEXTREQ;
 219   1        END DBMOD9;
```

CRT MODULE

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE CRTMODULE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:CS.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1             CRTMODULE:      DO;
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                        CRT MODULE
                **
                ****************************************************************************
                **
                                    MODULE IDENTIFICATION: CS
                                    MODULE NUMBER        : 06
                **
                ****************************************************************************
                */
                $ NOLIST
                /*
                ****************************************************************************
                **
                                        CS DATA
                **
                ****************************************************************************
                **
                **                      CRT I/O PORTS AND USART CONTROL WORDS
                */
 11    1        DCL CRTDATAIN LIT '0EEH',       /* DATA INPUT */
                    CRTSTATIN LIT '0EFH',       /* STATUS INPUT */
```

```
                    CRTDATAOUT LIT '0EEH',      /* DATA OUTPUT */
                    CRTCMDOUT LIT '0EFH',       /* COMMAND OUTPUT */
                    MCW LIT '01001010B',        /* MODE CONTROL WORD */
                    CCW LIT '00100111B',        /* COMMAND CONTROL WORD */
                    RESET LIT '01000000B';      /* RESET CRT */
                /*
                ****************************************************************************
                **
                **                          CRT CONTROL CHARACTERS
                **
                */
  12   1        DCL HOME LIT '2',               /* HOME CURSOR */
                    FC LIT '1CH',               /* FORWARD CURSOR */
                    BC LIT '8',                 /* BACK CURSOR */
                    UC LIT '1AH',               /* UP CURSOR */
                    CR LIT '0DH',               /* CARRIAGE RETURN */
                    LF LIT '0AH',               /* LINE FEED */
                    CLEAR LIT '1EH',            /* CLEAR SCREEN */
                    ADRXY LIT '0CH',            /* START X/Y ADDRESSING */
                    ROLL LIT '1DH',             /* ROLL MODE */
                    BELL LIT '7',
                    DEBUGMODE LIT '4',           /* DEBUG (CONTROL D) */
                    INT LIT '9',                /* INTERRUPTION (CONTROL I) */
                    TERMPER LIT '14H',          /* TERMINATE PER. DEBUG FUNCTION (CONT
              -   ROL T) */
                    CLRLINE LIT '17H',          /* CLEAR LINE */
                    BOL LIT '96',               /* BEGIN OF LINE */
                    ASYNCLINE LIT '102',        /* LINE FOR ASYNCHRONOUS PRINTS */
                    INPLINE LIT '119',          /* INPUT LINE */
                    ARROW LIT '5EH',            /* UP ARROW */
                    BLANK LIT ''' ''',          /* BLANK */
                    DELCHAR LIT '5FH',          /* BACK SPACE - DELETE LAST CHARACTER
              -   */
```

```
                    LINELENGTH LIT '79',     /* LINE LENGTH */
                    CRTOUTLEN LIT '200',     /* LENGTH OF OUTPUT BUFFER */
                    DELINPUT LIT '7FH';      /* DELETE ENTIRE INPUT (RUB OUT) */
                /*
                ***************************************************************
                **
                **                  MESSAGE CONTROL BLOCKS
                **
                */
  13    1       DCL CSMN20(4) BYTE INITIAL (0,CS,20,0),
                       /* TRANSFER INPUT TEXT MSG */
                    CSMN21(4) BYTE INITIAL (0,CS,21,5),
                       /* INPUT REQUEST TELL BACK MSG
                            1ST DATA BYTE: TRUE - REQUEST ACKNOWLEDGED
                                           FALSE OTHERWISE */
                    CSMN22(4) BYTE INITIAL (0,CS,22,4),
                        /* TERMINATE I/O MSG
                           SEND TO MODULE CONNECTED TO CRT IF INPUT IS 'INT' */
                    CSMN23(4) BYTE INITIAL (DB,CS,23,4),
                        /* START DEBUG MSG */
                    CSMN25(4) BYTE INITIAL (0,CS,25,4),
                        /* TERMINATE PERIODIC DEBUG FUNCTION */
                    CSMN26(4) BYTE INITIAL (0,CS,26,5);
                        /* TELLBACK MSG */
                /*
                ***************************************************************
                **
                **                      CONSTANT DATA
                **
                */
  14    1       DCL ROLLSCR (8) BYTE INITIAL (CR,ADRXY,BOL,ASYNCLINE+1,CLRLINE,ADR
              -  XY,BOL,
                                               INPLINE),
                        /* ROLL SCREEN,CLEAR LINE AFTER ASYNCLINE,RETURN TO
```

300

```
                                BEGIN OF INPUT LINE */
                        DELTEXT(4) BYTE INITIAL (ADRXY,0,INPLINE,CLRLINE),
                            /* GO BACK TO BEGIN OF LAST INPUT,
                                CLEAR REST OF LINE */
                        NEWL(2) BYTE INITIAL (1,' '),
                        BEGL(4) BYTE INITIAL (0,ADRXY,BOL,INPLINE),
                        ASYNC1(10) BYTE INITIAL (ARROW,ADRXY,BOL,ASYNCLINE,CLRLINE,
                                                 BELL,'*** '),
                            /* START ASYNCHRONOUS PRINT */
                        ASYNC2(3) BYTE INITIAL (ADRXY,0,INPLINE),
                            /* RETURN FROM ASYNCHRONOUS PRINT */
                        INITCRT(25) BYTE INITIAL (CLEAR,ROLL,ADRXY,BOL,ASYNCLINE,
                                BELL,'*** SYSTEM START',ADRXY,BOL,INPLINE);
                            /* INITIALIZE SCREEN AT SYSTEM START */
                    /*
                    ***************************************************************************
                    **
                    **                          VARIABLE DATA
                    **
                    */
 15   1             DCL CSBEGDATA BYTE,
                        INACTMOD BYTE,
                            /* MODULE WITH CURRENT INPUT ACTIVE */
                        CRTIN(LINELENGTH) BYTE,
                            /* CRT INPUT BUFFER */
                        CRTOUT(CRTOUTLEN) BYTE,
                            /* CRT OUTPUT BUFFER */
                        (INX,OUTX) BYTE,
                            /* POINTER TO CRTIN AND CRTOUT (NEXT TO FILL) */
                        CURSORINP BYTE,
                            /* CURSOR POSITION ON INPUT LINE */
                        (CURSORSAVE,INXSAVE) BYTE,
                            /* SAVE CURSORINP AND INX AT BEGIN OF INPUT */
```

```
OUTACTIVE BYTE,
        /* TRUE IF OUTPUT ACTIVE */
INPREQUEST BYTE,
    /* TRUE IF INPUT REQUEST MSG RECEIVED */
OUTPTR BYTE,
    /* POINTER TO NEXT CHARACTER TO PRINT */
ROLLSCREEN BYTE,
    /* TRUE IF SCREEN IS TO ROLL AFTER INPUT */
(CSOUTFL,CSINFL) BYTE,
    /* IF 0 THEN FIRST CALL OF CSOUTPUT,CSINPUT */
(CSOUTADR,CSINADR) ADDRESS,
    /* ADDRESS OF PROCEDURES CSOUTPUT/CSINPUT */
(CSINX,CSOUTX) BYTE,
    /* CS INDICES FOR PRIORITY CALLS */
CHAR BYTE,
    /* INPUT CHARACTER */
DEBUG BYTE,
    /* TRUE IF DEBUG MODE */
CSTELLBACK BYTE,
    /* TRUE IF TELLBACK MSG REQUIRED AFTER SYNC OUTPUT */
CSTBCODE BYTE,
    /* REQUIRED CODE FOR TELLBACK MSG */
CSENDDATA BYTE;
    /* END OF VARIABLE DATA */
```

```
                $EJECT
                $NOLIST
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                        CSCONVXY

                    CONVERT INPUT TO XY-ADDRESSING
                **
                ****************************************************************************
                */
303   103   1   CSCONVXY:    PROCEDURE (B) BYTE;

      104   2   DCL      (A,B) BYTE;

      105   2            A =  B + 95;
      106   2            IF B < 33 THEN RETURN A;
      108   2            A = A - 64;
      109   2            IF B < 65 THEN RETURN A;
      111   2            A = A - 64;
      112   2            RETURN A;
      113   2            END CSCONVXY;
                /*
                ****************************************************************************
              -  *
                ****************************************************************************
              -  *
                **
                                                                        CSMOVEOU
              -  T
```

```
                        MOVE CHARACTERS FROM MSG INTO CRT OUTPUT BUFFER
                    **
                    ************************************************************************
                -    *
                    */
114     1           CSMOVEOUT:  PROCEDURE;

115     2                   B2 = MSG(ML) - 5;
116     2                   DO WHILE B1 <= B2 AND OUTX <= CRTOUTLEN;
117     3                     CRTOUT(OUTX) = MSGDATA(B1);
118     3                     OUTX = OUTX + 1;
119     3                     B1 = B1 + 1;
120     3                     END;
121     2                   RETURN;
122     2                   END CSMOVEOUT;
                    /*
                    ************************************************************************
                    ************************************************************************
                    **
                                                                          PACKCRTOUT

                        PACK INPUT INTO CRT OUTPUT BUFFER
                    **
                    ************************************************************************
                    */
123     1           PACKCRTOUT: PROCEDURE(B);

124     2           DCL     B BYTE;

125     2                   IF OUTX > CRTOUTLEN THEN RETURN;
                                /* OVERFLOW */
127     2                   CRTOUT(OUTX) = B;
```

```
128   2             OUTX = OUTX + 1;
129   2             RETURN;
130   2             END PACKCRTOUT;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                        PACKROLL
                PACK CODES FOR ROLL SCREEN
            **
            ***************************************************************************
            */
131   1     PACKROLL:   PROCEDURE;

132   2             DO B1 = 0 TO LAST(ROLLSCR);
133   3               CALL PACKCRTOUT(ROLLSCR(B1));
134   3               END;
135   2             CURSORINP = 1;CURSORSAVE = 1;
137   2             RETURN;
138   2             END PACKROLL;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                        CSOUTPUT

                CONTINUE CRT OUTPUT
                PRIORITY PROCEDURE , CALL SCHEDULED BY INT 3
                PROCESS LEVEL 3 CRT OUTPUT INTERRUPT
            **
            ***************************************************************************
            */
```

```
 139   1       CSOUTPUT:   PROCEDURE;

 140   2       DCL     OUTPUTTEST ADDRESS;

 141   2               IF CSOUTFL = 0 THEN
 142   2                 DO;
                             /* FIRST CALL, FIND BEGIN ADDRESS OF CSOUTPUT */
 143   3                 CSOUTADR = PROCADR;
 144   3                 RETURN;
 145   3                 END;
 146   2               OUTPUTTEST = OUTPUTTEST + 1;
 147   2               IF OUTPTR >= OUTX THEN
                            /* NO MORE CHARACTERS FOR OUTPUT */
 148   2                 DO;
 149   3                 OUTPTR = 0; OUTX = 0;
 151   3                 OUTACTIVE = FALSE;
 152   3                 IF CSTELLBACK THEN
                             /* TELLBACK MSG REQUIRED */
 153   3                   DO;
 154   4                   CSTELLBACK = FALSE;
 155   4                   CSMN26(0) = INACTMOD;
 156   4                   IF NOT SEND(.CSMN26(0)) THEN RETURN;
                               /* SEND REQUIRED MSG AND TELLBACK CODE */
 158   4                   END;
 159   3                 RETURN;
 160   3                 END;
 161   2               OUTPUT(CRTDATAOUT) = CRTOUT(OUTPTR);
                            /* OUTPUT NEXT CHARACTER */
 162   2               OUTPTR = OUTPTR + 1;
 163   2               RETURN;
 164   2               END CSOUTPUT;
                 /*
```

```
                /*****************************************************************************
                ******************************************************************************
                **
                                                                        STARTOUT

                    START CRT OUTPUT IF NO OUTPUT ACTIVE
                **
                ******************************************************************************
                */
  165   1       STARTOUT: PROCEDURE;

  166   2               IF OUTACTIVE THEN RETURN;
  168   2               CALL CSOUTPUT;
  169   2               OUTACTIVE = TRUE;
  170   2               RETURN;
  171   2               END STARTOUT;
                /*
                ******************************************************************************
                ******************************************************************************
                **
                                                                        TERMIO

                    TERMINATE I/O
                **
                ******************************************************************************
                */
  172   1       TERMIO: PROCEDURE;

  173   2               IF INACTMOD = 0FFH THEN RETURN;
                            /* NO MODULE WITH ACTIVE I/O */
  175   2               CSMN22(0) = INACTMOD;
  176   2               IF NOT SEND(.CSMN22(0)) THEN RETURN;
```

```
                        /* SEND TERMINATE MSG */
178   2                 INACTMOD = 0FFH;
179   2                 INPREQUEST = FALSE;
180   2                 DEBUG = FALSE;
181   2                 CALL PACKROLL;
182   2                 CALL STARTOUT;
183   2                 RETURN;
184   2                 END TERMIO;
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                          CSINPUT

                    CONTINUE CRT INPUT
                    PRIORITY PROCEDURE , CALL SCHEDULED BY INT 3
                    PROCESS LEVEL 3 CRT INPUT INTERRUPT
                **
                ****************************************************************************
                */
185   1         CSINPUT:    PROCEDURE;

186   2         DCL         INPUTTEST ADDRESS;

187   2                     IF CSINFL = 0 THEN
                              /* FIRST CALL,FIND BEGIN ADDRESS OF CSINPUT */
188   2                       DO;
189   3                       CSINADR = PROCADR;
190   3                       RETURN;
191   3                       END;
192   2                     INPUTTEST = INPUTTEST + 1;
193   2                     CHAR = INPUT(CRTDATAIN);
```

```
                        /* GET NEXT CHARACTER */
194   2               IF CHAR = INT THEN
                        /* CRT INTERRUPTION */
195   2                 DO;
196   3                 CALL TERMIO;
197   3                 GO TO INITIN;
198   3                 END;
199   2               IF CHAR = DEBUGMODE THEN
                        /* START DEBUG REQUEST */
200   2                 DO;
201   3                 IF DEBUG THEN GO TO ILLCHK;
                            /* ALREADY IN DEBUG MODE */
203   3                 CALL TERMIO;
204   3                 IF NOT SEND(.CSMN23(0)) THEN GOTO INITIN;
                            /* SEND DEBUG REQUEST MSG */
206   3                 DEBUG = TRUE;
207   3                 INACTMOD = DB;
208   3                 GO TO INITIN;
209   3                 END;
210   2               IF CHAR = TERMPER THEN
                        /* TERMINATE PERIODIC DEBUG FUNCTION */
211   2                 DO;
212   3                 IF NOT DEBUG THEN GO TO ILLCHK;
                            /* ILLEGAL INPUT */
214   3                 CSMN25(0) = INACTMOD;
215   3                 B1 = SEND(.CSMN25(0));
                            /* SEND TERM PER FUNCTION MSG */
216   3                 GO TO INITIN;
217   3                 END;
218   2               IF OUTACTIVE OR NOT INPREQUEST THEN RETURN;
                        /* OUTPUT ACTIVE OR NO INPUT REQUEST */
220   2               IF CHAR = DELCHAR THEN
```

```
                        /* DELETE LAST CHARACTER */
 221    2               DO;
 222    3               IF CURSORINP <= CURSORSAVE
                          THEN CALL PACKCRTOUT(BELL);
 224    3                 ELSE DO;
 225    4                   CALL PACKCRTOUT(BC);
 226    4                   CALL PACKCRTOUT(CLRLINE);
 227    4                   INX = INX - 1;
 228    4                   CURSORINP = CURSORINP - 1;
 229    4                   END;
 230    3               GO TO INITOUT;
 231    3               END;
 232    2             IF CHAR = DELINPUT THEN
                          /* DELETE ENTIRE INPUT */
 233    2               DO;
 234    3               CURSORINP = CURSORSAVE;
 235    3               INX = INXSAVE;
 236    3               DELTEXT(1) = CSCONVXY(CURSORINP);
 237    3               DO B1 = 0 TO LAST(DELTEXT);
 238    4                 CALL PACKCRTOUT(DELTEXT(B1));
 239    4                 END;
 240    3               GO TO INITOUT;
 241    3               END;
 242    2             IF CHAR = CR THEN
                          /* END OF INPUT */
 243    2               DO;
 244    3               CSMN20(0) = INACTMOD;
                            /* SET RECEIVING MODULE */
 245    3               CSMN20(3) = INX + 3;
                            /* MSG LENGTH */
 246    3               INPREQUEST = FALSE;
 247    3               IF SEND(.CSMN20(0))
```

```
                        /* SEND INPUT TEXT MSG */
                      THEN DO;
249   4                   B2 = 0;
250   4                   DO B1 = 1 TO INX - 1;
251   5                     MSGDATA(B2) = CRTIN(B1);
252   5                     B2 = B2 + 1;
253   5                     END;
254   4                   END;
255   3                 ELSE INACTMOD = 0FFH;
                            /* RECEIVING MODULE NO LONGER ACTIVE */
256   3               INX = 1;
257   3               IF ROLLSCREEN THEN
258   3                 DO;
259   4                 CALL PACKROLL;
260   4                 GO TO INITOUT;
261   4                 END;
262   3               GO TO INITIN;
263   3               END;
264   2     ILLCHK:     IF CHAR < 20H OR CHAR > 5AH OR CURSORINP >= LAST(CRTIN
          -   );
                      /* ILLEGAL INPUT */
                    THEN DO;
                        /* PRINT '?' INSTEAD OF INPUT CHARACTER */
266   3                 CALL PACKCRTOUT('?');
267   3                 CALL PACKCRTOUT(BC);
268   3                 END;
269   2               ELSE DO;
270   3                 CALL PACKCRTOUT(CHAR);
271   3                 CRTIN(INX) = CHAR;
272   3                 INX = INX + 1;
273   3                 CURSORINP = CURSORINP + 1;
274   3                 END;
```

```
275   2       INITOUT:CALL STARTOUT;
276   2       INITIN:     ;
277   2               RETURN;
278   2               END CSINPUT;
              /*
              ***************************************************************************
              ***************************************************************************
              **
                                                                    CSPRINTASYNC

                  PRINT ASYNCHRONOUS TEXT     (MN 10)
                  ( MAX 1 LINE )
              **
              ***************************************************************************
              */
279   1       CSPRINTASYNC: PROCEDURE;

280   2               DO B1 = 0 TO LAST(ASYNC1);
281   3                 CALL PACKCRTOUT(ASYNC1(B1));
282   3                 END;
283   2               B1 = 0;
284   2               CALL CSMOVEOUT;
                          /* PACK MSG INTO OUTPUT BUFFER */
285   2               ASYNC2(1) = CSCONVXY(CURSORINP);
                          /* RETURN CURSOR TO LAST INPUT POSITION */
286   2               DO B1 = 0 TO LAST(ASYNC2);
287   3                 CALL PACKCRTOUT(ASYNC2(B1));
288   3                 END;
289   2               CALL STARTOUT;
290   2               RETURN;
291   2               END CSPRINTASYNC;
              /*
```

```
                ********************************************************************************
                ********************************************************************************
                **
                                                                        CSPRINTSYNC

                    PRINT SYNCHRONOUS TEXT     (MN 11 AND MN 18)
                    PRINT ON INPUT LINE
                    IF MSGDATA(0) = TRUE : ROLL SCREEN AFTER OUTPUT
                **
                ********************************************************************************
                */
 292    1       CSPRINTSYNC: PROCEDURE;

 293    2                IF MSG(SMN) <> INACTMOD THEN
                             /* MSG NOT FROM CORRECT MODULE */
 294    2                  DO;
 295    3                  CALL ILLEGALMSG;
 296    3                  RETURN;
 297    3                  END;
 298    2                B1 = 1;
 299    2                CALL CSMOVEOUT;
                             /* MOVE CHARACTERS FROM MSG TO CRT OUTPUT BUFFER */
 300    2                IF MSGDATA(0)
                           THEN CALL PACKROLL;
                                /* ROLL SCREEN AFTER OUTPUT */
 302    2                  ELSE CURSORINP = CURSORINP + MSG(ML) - 5;
 303    2                CALL STARTOUT;
 304    2                RETURN;
 305    2                END CSPRINTSYNC;
                /*
                ********************************************************************************
                ********************************************************************************
```

```
                **
                                                                CSINPREQ

                    INPUT REQUEST FROM MODULE CONNECTED TO CRT (MN 12)
                    SEND INPUT UP TO CR TO MODULE INACTMOD
                    IF MSGDATA(0) = TRUE : ROLL SCREEN AFTER INPUT
                **
                ********************************************************************************
                */
 306    1       CSINPREQ:   PROCEDURE;

 307    2               IF MSG(SMN) <> INACTMOD THEN
                             /* MSG NOT FROM CORRECT MODULE */
 308    2                 DO;
 309    3                 CALL ILLEGALMSG;
 310    3                 RETURN;
 311    3                 END;
 312    2               CURSORSAVE = CURSORINP;
 313    2               INXSAVE = INX;
 314    2               INPREQUEST = TRUE;
 315    2               ROLLSCREEN = MSGDATA(0);
                             /* TRUE IF NEW LINE AFTER INPUT */
 316    2               RETURN;
 317    2               END CSINPREQ;
                /*
                ********************************************************************************
                ********************************************************************************
                **
                                                              CSINPACTIVATE

                    ACTIVATE INPUT FOR SENDING MODULE     (MN 13)
                **
```

```
                ************************************************************************
                */
 318    1       CSINPACTIVATE: PROCEDURE;

 319    2               CSMN21(0) = MSG(SMN);
 320    2               IF NOT SEND(.CSMN21(0)) THEN RETURN;
                            /* SEND ACKNOWLEDGE MSG BACK */
 322    2               IF INACTMOD <> 0FFH
                          THEN MSGDATA(0) = FALSE;
                              /* CRT NOT FREE */
 324    2                 ELSE DO;
 325    3                   MSGDATA(0) = TRUE;
 326    3                   INACTMOD = MSG(SMN);
 327    3                   INPREQUEST = FALSE;
 328    3                   END;
 329    2               RETURN;
 330    2               END CSINPACTIVATE;
                /*
                ************************************************************************
                ************************************************************************
                **
                                                                        CSNEWLINE

                    POSITION CURSOR AT BEGIN OF NEW LINE  ( MN 16 )
                **
                ************************************************************************
                */
 331    1       CSNEWLINE:  PROCEDURE;

 332    2               MSG(ML) = 6;
 333    2               ADRMSGDATA = .NEWL(0);
 334    2               CALL CSPRINTSYNC;
```

```
335   2             RETURN;
336   2             END CSNEWLINE;
                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                        CSBEGLINE

                    POSITION CURSOR AT BEGIN OF INPUT LINE  ( MN 17 )
                **
                ***************************************************************************
                */
337   1       CSBEGLINE:  PROCEDURE;

338   2             MSG(ML) = 8;
339   2             ADRMSGDATA = .BEGL(0);
340   2             CALL CSPRINTSYNC;
341   2             CURSORINP = 1;CURSORSAVE = 1;
343   2             RETURN;
344   2             END CSBEGLINE;
                /*
                ***************************************************************************
                ***************************************************************************
                **
                                                                        CSSTART

                    INITIALIZE CRT MODULE    (MN 00)
                **
                ***************************************************************************
                */
345   1       CSSTART:   PROCEDURE;
```

```
 346   2              IF RESTART THEN OUTPUT(CRTCMDOUT) = RESET;
                          /* RESET USART IF NO SYSTEM RESET */
 348   2              CALL CLEARDATA(.CSBEGDATA,.CSENDDATA);
                          /* CLEAR VARIABLE DATA */
 349   2              CALL CSINPUT;
 350   2              CALL CSOUTPUT;
 351   2              CSOUTFL = 1;CSINFL = 1;
                          /* FIND PROCEDURE START ADRRESSES */
 353   2              CALL UPDSTAT(CS,00001011B);
                          /* SET MODULE STATUS */
 354   2              DO B1 = 0 TO LAST(INITCRT);
                          /* INITIALIZE SCREEN */
 355   3                CRTOUT(B1) = INITCRT(B1);
 356   3                END;
 357   2              OUTX = LENGTH(INITCRT);
 358   2              INX = 1;CURSORINP = 1;
 360   2              CURSORSAVE = 1;
 361   2              ROLLSCREEN = TRUE;
 362   2              INACTMOD = 0FFH;
 363   2              INPREQUEST = FALSE;
 364   2              DEBUG = FALSE;
 365   2              CSOUTX = ENTERPRIOR(CSOUTADR);
 366   2              CSINX = ENTERPRIOR(CSINADR);
                          /* ENTER PRIORITY CALLS */
 367   2              OUTPUT(CRTCMDOUT) = MCW;
                          /* MODE CONTROL WORD */
 368   2              B1 = 0;
 369   2              OUTPUT(CRTCMDOUT) = CCW;
                          /* COMMAND CONTROL WORD */
 370   2              B1 = ENTERINT(4,CSINX);
 371   2              B1 = ENTERINT(5,CSOUTX);
                          /* ENTER INTERRUPTS ON LEVELS 3 AND 4 */
```

```
372   2             CALL STARTOUT;
                        /* INITIALIZE START PRINTOUT */
373   2             RETURN;
374   2             END CSSTART;
            /*
            ***************************************************************************
            ***************************************************************************
            **
                                                                          MSGENT06

                CS MESSAGE ENTRY
            **
            ***************************************************************************
            */
375   1     MSGENT06:   PROCEDURE PUBLIC;

376   2             IF MSG(MN) = 0 THEN
                        /* START MSG */
377   2               DO;
378   3               CALL CSSTART;
379   3               RETURN;
380   3               END;
381   2             IF MSG(MN) < 10 OR MSG(MN) > 18
                    THEN CALL ILLEGALMSG;
383   2               ELSE DO CASE MSG(MN)-10;
384   3                 CALL CSPRINTASYNC;        /* MN 10 */
385   3                 CALL CSPRINTSYNC;         /* MN 11 */
386   3                 CALL CSINPREQ;            /* MN 12 */
387   3                 CALL CSINPACTIVATE;       /* MN 13 */
388   3                 DO;
389   4                 INACTMOD = 0FFH;         /* MN 14 */
390   4                 DEBUG = FALSE;
```

```
 391   4                    END;
 392   3                    INACTMOD = MSG(SMN);    /* MN 15 */
 393   3                    CALL CSNEWLINE;         /* MN 16 */
 394   3                    CALL CSBEGLINE;         /* MN 17 */
 395   3                    DO;                     /* MN 18 */
 396   4                      CSTELLBACK = TRUE;CSTBCODE = MSGDATA(0);
 398   4                      ADRMSGDATA = ADRMSGDATA + 1;
 399   4                      MSG(ML) = MSG(ML) - 1;
 400   4                      CALL CSPRINTSYNC;
 401   4                      END;
 402   3                    END;
 403   2                RETURN;
 404   2                END MSGENT06;
 405   1      END;
```

319

MODULE INFORMATION:

```
     CODE AREA SIZE     = 05ABH   1451D
     VARIABLE AREA SIZE = 0186H    390D
     MAXIMUM STACK SIZE = 000CH     12D
     1006 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

LINE PRINTER MODULE

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE LPMOD
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:LP.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


   1            LPMOD:       DO;
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                        LINE PRINTER MODULE
                **
                ****************************************************************************
                **
                        MODULE IDENTIFICATION: LP
                        MODULE NUMBER        : 05
                **
                ****************************************************************************
                */
                $ NOLIST
                /*
                ****************************************************************************
                **
                                    LP DATA
                **
                ****************************************************************************
                **
                **                      SUBSTITUTIONS
                */
  98    1       DCL     LPDATAOUT LIT '0FAH',       /* LP DATA OUTPUT PORT */
                        LPCMDOUT LIT '0FBH',        /* LP COMMAND OUTPUT PORT */
```

```
                        LPSTATIN LIT '0FBH',        /* LP STATUS INPUT PORT */
                        LPOUTBUFLEN LIT '150',      /* OUTPUT BUFFER LENGTH */
                        FORMFEED LIT '0CH',
                        CR LIT '0DH',
                        LF LIT '0AH',
                        ENDSUBST LIT '0';
                    /*
                    ***************************************************************
                    **
                    **                      CONSTANT DATA
                    **
                    */
 99   1             DCL     LPMN10 (4) BYTE INITIAL (CS,LP,10,0),
                            /* ASYNC PRINT MSG */
                        LPMN26 (4) BYTE INITIAL (0,LP,26,5),
                            /* TELLBACK MSG */
                        LPMN28 (4) BYTE INITIAL (0,LP,28,4),
                            /* LP NOT CONNECTED MSG */
                        NONOPMSG (26) BYTE INITIAL ('LINE PRINTER NOT CONNECTED'),
                        ENDCONST BYTE;
                    /*
                    ***************************************************************
                    **
                    **                       VARIABLE DATA
                    **
                    */
100   1             DCL     LPCLEARBEG BYTE,
                            /* BEGIN OF CLEAR REGION */
                        LPOUTFL BYTE,
                            /* 0 - FIRST CALL OF PROCEDURE LPOUTPUT */
                        LPOUTPUTADR ADDRESS,
                            /* START ADDRESS OF LPOUTPUT */
                        LPOUTBUF(LPOUTBUFLEN) BYTE,
                            /* LP OUTPUT BUFFER */
```

```
(LPOUTX,LPOUTPTR) BYTE,
    /* INDICES FOR LP OUTPUT BUFFER
       LPOUTX - NEXT TO FILL
       LPOUTPTR - NEXT TO PRINT */
LPTELLBACK BYTE,
    /* TRUE IF TELLBACK MSG REQUESTED AFTER OUTPUT */
LPTBCODE BYTE,
    /* REQUESTED CODE FOR TELLBACK MSG */
LPPRX BYTE,
    /* LP PRIORITY INDEX */
LPOUTACTIVE BYTE,
    /* TRUE IF OUTPUT ACTIVE */
LPMODSAVE BYTE,
    /* MODULE NUMBER OF PRINTING MODULE */
LPCLEAREND BYTE;
```

```
                $ EJECT
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                        LPOUTPUT

                    CONTINUE LINE PRINTER OUTPUT
                    PRIORITY PROCEDURE , CALL SCHEDULED BY INT 3
                    TO PROCESS LEVEL 3 LP OUTPUT INTERRUPT
                **
                ****************************************************************************
                */
323   101   1     LPOUTPUT:   PROCEDURE;

      102   2               IF LPOUTFL = 0 THEN
                                /* FIRST CALL,FIND ADDRESS OF LPOUTPUT */
      103   2                 DO;
      104   3                 LPOUTPUTADR = PROCADR;
      105   3                 LPOUTFL = 1;
      106   3                 RETURN;
      107   3                 END;
      108   2               IF LPOUTPTR >= LPOUTX THEN
                                /* NO MORE CHARACTERS TO PRINT */
      109   2                 DO;
      110   3                 LPOUTPTR = 0;LPOUTX = 0;
      112   3                 LPOUTACTIVE = FALSE;
      113   3                 IF LPTELLBACK THEN
                                    /* TELLBACK MSG REQUESTED */
      114   3                   DO;
      115   4                   LPTELLBACK = FALSE;
      116   4                   LPMN26(RMN) = LPMODSAVE;
```

```
 117   4                    IF NOT SEND(.LPMN26(0)) THEN RETURN;
 119   4                    MSGDATA(0) = LPTBCODE;
                                /* SEND TELLBACK MSG WITH REQUESTED CODE */
 120   4                    END;
 121   3                  RETURN;
 122   3                  END;
 123   2                OUTPUT(LPDATAOUT) = NOT LPOUTBUF(LPOUTPTR);
                            /* OUTPUT NEXT CHARACTER */
 124   2                LPOUTPTR = LPOUTPTR + 1;
 125   2                RETURN;
 126   2                END LPOUTPUT;
                  /*
                  ******************************************************************************
                  ******************************************************************************
                  **
                                                                        LPSTARTOUT

                      START LINE PRINTER OUTPUT
                  **
                  ******************************************************************************
                  */
 127   1          LPSTARTOUT: PROCEDURE;

 128   2                  IF LPOUTACTIVE THEN RETURN;
 130   2                  CALL LPOUTPUT;
 131   2                  LPOUTACTIVE = TRUE;
 132   2                  RETURN;
 133   2                  END LPSTARTOUT;
                  /*
                  ******************************************************************************
                  ******************************************************************************
                  **
```

LPPACK

```
                    PACK 1 CHARACTER INTO OUTPUT BUFFER
                ** 
                ************************************************************************
                */
 134    1       LPPACK: PROCEDURE (CHAR) BYTE;

 135    2       DCL     CHAR BYTE;

 136    2               IF LPOUTX > LAST(LPOUTBUF)
                   THEN DO;
                            /* BUFFER OVERFLOW */
 138    3                    CALL SYSMON(STACKPTR,5,0);
 139    3                    RETURN FALSE;
 140    3                    END;
 141    2                  ELSE DO;
                            /* PACK CHARACTER */
 142    3                    LPOUTBUF(LPOUTX) = CHAR;
 143    3                    LPOUTX = LPOUTX + 1;
 144    3                    END;
 145    2               RETURN TRUE;
 146    2               END LPPACK;
                /*
                ************************************************************************
                ************************************************************************
                **
```

LPNEWLINE

```
                    POSITION PRINT HEAD AT BEGIN OF NEW LINE
                **
                ************************************************************************
```

```
                */
147    1        LPNEWLINE:  PROCEDURE;

148    2                IF NOT LPPACK(CR) THEN RETURN;
150    2                IF NOT LPPACK(LF) THEN RETURN;
152    2                CALL LPSTARTOUT;
153    2                RETURN;
154    2                END LPNEWLINE;
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                    LPNEWPAGE

                    POSITION PRINT HEAD AT TOP OF NEW PAGE
                **
                **********************************************************************
                */
155    1        LPNEWPAGE:  PROCEDURE;

156    2                IF NOT LPPACK(FORMFEED) THEN RETURN;
158    2                CALL LPSTARTOUT;
159    2                RETURN;
160    2                END LPNEWPAGE;
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                    LPPRINT

                PROCESS PRINT MSG     (MN 11)
                IF MSGDATA(0) = TRUE : PRINT HEAD AT BEGIN OF NEW LINE AFTER 0
```

```
                -   UTPUT
                    **
                    ************************************************************************
                    */
     161    1       LPPRINT:    PROCEDURE;

     162    2               B3 = MSG(ML) - 5;
     163    2               IF B3 = 0 THEN RETURN;
     165    2               DO B2 = 1 TO B3;
                                /* MOVE CHARACTERS INTO OUTPUT BUFFER */
     166    3                 IF NOT LPPACK(MSGDATA(B2)) THEN GO TO PR1;
                                  /* OUTPUT BUFFER FULL */
     168    3                 END;
     169    2               IF MSGDATA(0) THEN CALL LPNEWLINE;
     171    2       PR1:        ;
     172    2               CALL LPSTARTOUT;
     173    2               RETURN;
     174    2               END LPPRINT;
                    /*
                    ************************************************************************
                    ************************************************************************
                    **
                                                                        LPPRINTTB

                        PROCESS PRINT MSG WITH TELLBACK     (MN 18)
                    **
                    ************************************************************************
                    */
     175    1       LPPRINTTB:  PROCEDURE;

     176    2               IF MSG(ML) < 7 THEN RETURN;
     178    2               LPTBCODE = MSGDATA(0);
```

```
                        /* SAVE TELLBACK CODE */
179   2                 LPTELLBACK = TRUE;
180   2                 ADRMSGDATA = ADRMSGDATA + 1;
181   2                 MSG(ML) = MSG(ML) - 1;
                        /* ADJUST MSG FOR PROCESS BY LPPRINT */
182   2                 LPMODSAVE = MSG(SMN);
                        /* SAVE MODULE NUMBER FOR TELLBACK MSG */
183   2                 CALL LPPRINT;
184   2                 RETURN;
185   2                 END LPPRINTTB;
                  /*
                  ****************************************************************************
                  ****************************************************************************
                  **
                                                                          LPINIT

                      INITIALIZE LINE PRINTER
                  **
                  ****************************************************************************
                  */
186   1           LPINIT:     PROCEDURE;

187   2                   IF (INPUT(LPSTATIN) AND 2) <> 0
                            THEN DO;
                                /* LP NOT CONECTED */
189   3                       LPMN28(RMN) = MSG(SMN);
190   3                       B1 = SEND(.LPMN28(0));
                                /* SEND 'LP NOT CONNECTED' MSG */
191   3                       LPMN10(ML) = LENGTH(NONOPMSG) + 4;
192   3                       IF NOT SEND(.LPMN10(0)) THEN RETURN;
                                /* SEND ASYNC PRINT MSG TO CS */
194   3                       DO B3 = 0 TO LAST(NONOPMSG);
```

```
195   4                    MSGDATA(B3) = NONOPMSG(B3);
196   4                    END;
197   3                  END;
198   2                ELSE CALL LPNEWPAGE;
                            /* LINE PRINTER CONNECTED */
199   2              RETURN;
200   2              END LPINIT;
               /*
               ***************************************************************************
               ***************************************************************************
               **
                                                                          LPSTART

                    INITIALIZE LP MODULE
               **
               ***************************************************************************
               */
201   1        LPSTART:    PROCEDURE;

202   2              CALL CLEARDATA(.LPCLEARBEG,.LPCLEAREND);
                        /* CLEAR VARIABLE DATA */
203   2              CALL LPOUTPUT;
                        /* FIND PROCEDURE START ADDRESS */
204   2              CALL UPDSTAT(LP,3);
                        /* SET MODULE STATUS */
205   2              LPPRX = ENTERPRIOR(LPOUTPUTADR);
                        /* ENTER PRIORITY CALL TO PROCESS INTERRUPT */
206   2              RETURN;
207   2              END LPSTART;
               /*
               ***************************************************************************
               ***************************************************************************
```

```
                **
                                                                MSGENT05

                    MESSAGE ENTRY OF LP MODULE
                **
                ************************************************************************
                */
 208   1        MSGENT05:   PROCEDURE PUBLIC;

 209   2                  IF MSG(MN) = 0 THEN
                             /* START MSG */
 210   2                    DO;
 211   3                    CALL LPSTART;
 212   3                    RETURN;
 213   3                    END;
 214   2                  IF MSG(MN) < 11 OR MSG(MN) > 18
                            THEN CALL ILLEGALMSG;
 216   2                    ELSE DO CASE MSG(MN) - 11;
 217   3                       CALL LPPRINT;
 218   3                       CALL LPNEWPAGE;
 219   3                       CALL LPINIT;
 220   3                       CALL ILLEGALMSG;
 221   3                       CALL ILLEGALMSG;
 222   3                       CALL LPNEWLINE;
 223   3                       CALL LPNEWLINE;
 224   3                       CALL LPPRINTTB;
 225   3                       END;
 226   2                  RETURN;
 227   2                  END MSGENT05;
 228   1        END;
```

LOADER

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE LOADER
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:LOADER.SRC NOOBJECT PAGEWIDTH(80) PAGELENGTH(35)


  1          LOADER:   DO;

  2   1      DECLARE LIT LITERALLY 'LITERALLY';
  3   1      DECLARE DCL LIT 'DECLARE';

  4   1      DCL CR LIT '0DH',
                 LF LIT '0AH',
                 NUMSBC LIT '3',
                 INTVECTOR LIT '3000H',
                 BIAS LIT '6000H',
                 BEGINRAM LIT '3000H';

  5   1      DCL SBC BYTE,      /* NUMBER OF SBC BEING LOADED */
                 (A1,A2) ADDRESS,(B1,B2) BYTE,
                 (MEMINTS BASED A1) (1) BYTE,(MEMINTD BASED A2) (1) BYTE,
                 FILENAME (6) BYTE INITIAL ('LOAD0/'),
                   /* FILENAMES OF FILES CONTAINING CODE FOR SBC'S */

                 INTSHFTTBL (10) BYTE INITIAL (3,3,4,4,5,5,6,0,7,2),
                     /* BYTE 0: LOADED INT LEVEL,BYTE 1:ACTUAL INT LEVEL ETC.  *
           -   /
                 KEY ADDRESS AT (0F1F4H) INITIAL (0ABCDH),
                 LOADSBC BYTE AT (0F1F0H),
                 START1 BYTE AT (.LOADSBC + 1),
                 START2 BYTE AT (.LOADSBC + 2),
                 START3 BYTE AT (.LOADSBC + 3),
```

```
                TEXT1 (*) BYTE INITIAL (CR,LF,LF,'SBC LOADER',CR,LF,LF),
                TEXT2 (*) BYTE INITIAL ('SBC LOADED',CR,LF,LF),
                TEXT3 (*) BYTE INITIAL ('LOAD FILE FOR SBC 0 NOT FOUND',CR,LF,
        -   LF),

                SBCRAM (4096) BYTE AT (BEGINRAM + BIAS);

 6   1      READ:       PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
 7   2      DCL       (P1,P2,P3,P4,P5) ADDRESS;
 8   2                END;
 9   1      WRITE:      PROCEDURE (P1,P2,P3,P4) EXTERNAL;
10   2      DCL       (P1,P2,P3,P4) ADDRESS;
11   2                END;
12   1      LOAD:       PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
13   2      DCL       (P1,P2,P3,P4,P5) ADDRESS;
14   2                END;
15   1      ERROR:      PROCEDURE (P1) EXTERNAL;
16   2      DCL       P1 ADDRESS;
17   2                END;

18   1                DO B1 = 0 TO LAST(SBCRAM);
19   2                SBCRAM(B1) = 0;
20   2                  END;
21   1                START1,START2,START3,LOADSBC = 0;

22   1                CALL WRITE(0,.TEXT1(0),LENGTH(TEXT1),.A1);

23   1                DO SBC = 1 TO NUMSBC;
                           /* LOAD ALL SBC'S */
24   2                  FILENAME(4) = SBC + 30H;
25   2                  CALL LOAD(.FILENAME(0),BIAS,0,.A1,.A2);
```

```
                        /* LOAD CODE FOR SBC N */
 26   2                IF A2 = 0
                         THEN DO;
 28   3                   LOADSBC = SBC;
 29   3                   DO WHILE LOADSBC <> 0;
                             /* WAIT UNTIL SBC TRANSFERRED CODE
                                INTO ON-BOARD RAM */
 30   4                     END;
 31   3                   IF SBC = 1 THEN
                             /* SET ACTUAL INT VECTOR */
 32   3                     DO B1 = 0 TO LAST(INTSHFTTBL) BY 2;
 33   4                       A1 = SHL(INTSHFTTBL(B1),3);
 34   4                       A2 = SHL(INTSHFTTBL(B1+1),3) + INTVECTOR;
 35   4                       DO B2 = 0 TO 7;
 36   5                         MEMINTD(B2) = MEMINTS(B2);
 37   5                         END;
 38   4                       END;
 39   3                   TEXT2(4) = SBC + 30H;
 40   3                   CALL WRITE(0,.TEXT2(0),LENGTH(TEXT2),.A2);
                             /* TYPE COMPLETION MSG */
 41   3                   END;
 42   2                 ELSE DO;
 43   3                   TEXT3(18) = SBC + 30H;
 44   3                   CALL WRITE(0,.TEXT3(0),LENGTH(TEXT3),.A2);
                             /* TYPE WARNING MSG */
 45   3                   END;
 46   2                END;
 47   1              START1 = 1;
                        /* START SBC 1 */
 48   1      END LOADER;
```

EXTERNAL DECLARATIONS OF SYSTEM DATA

```
                $EJECT
                $ INCLUDE(:F1:SYDATE.SRC)
          =     /*
          =                     UPDATE: 1
          =     */
          =     /*
          =     ****************************************************************************
          =     ****************************************************************************
          =     **
          =                       SYSTEM DATA DECLARATIONS
          =     ***
          =     */
336 2  1  =     DECLARE LIT LITERALLY 'LITERALLY',
          =             DCL LIT 'DECLARE';
          =
    3  1  =     DCL TRUE LIT '0FFH',
          =         FALSE LIT '0',
          =         RMN LIT '0',
          =         SMN LIT '1',
          =         MN LIT '2',
          =         ML LIT '3',
          =         MAXCPTR LIT '3',          /* MAX NUMBER OF COMPUTERS */
          =         MAXMOD LIT '8',           /* MAX NUMBER OF MODULES IN 1 CPTR */
          =         MAXSYSMOD LIT '24',       /* MAX NUMBER OF MODULES IN SYSTEM */
          =         MSGBUFLEN LIT '500',      /* LENGTH OF MSG BUFFER */
          =         MAXPRIOR LIT '8',         /* MAX NUMBER OF PRIORITY CALLS */
          =         MAXPER LIT '8',           /* MAX NUMBER OF PERIODIC CALLS */
          =         BEGINCM LIT '0F000H',     /* BEGIN OF COMMON MEMORY */
          =         INTVECTOR LIT '3000H',   /* INTERRUPT VECTOR */
          =         CPTR LIT '1',             /* NUMBER OF HOST COMPUTER */
          =         FIRSTMN LIT '0',          /* MN OF FIRST MODULE IN CPTR */
          =         LASTMN LIT 'FIRSTMN+7',   /* MN OF LAST MODULE IN CPTR */
```

```
            =       EX   LIT 'FIRSTMN',       /* MN OF EXEC MODULE */
            =       LP LIT 'FIRSTMN+5',       /* MN OF LINE PRINTER MODULE */
            =       CS LIT 'FIRSTMN+6',       /* MN OF CRT MODULE */
            =       DB    LIT 'FIRSTMN+7';    /* MN OF DEBUG MODULE */
            =
            =   /*
            =   ***
            =   */
  4   1     =   DCL (B1,B2,B3) BYTE EXTERNAL,
            =       (A1,A2,A3,A4) ADDRESS EXTERNAL,
            =       (BU,BL) BYTE  AT (.A4);
            =             /*
            =             SYSTEM WORK VARIABLES
            =             BU(UPPER) AND BL(LOWER) OVERLAY ADDRESS VARIABLE A4
            =             */
  5   1     =   DCL (ADRMSG,ADRMSGDATA) ADDRESS EXTERNAL,
            =       (MSG BASED ADRMSG) (4) BYTE,
            =       (MSGDATA BASED ADRMSGDATA) (100) BYTE;
            =             /*
            =             WORK AREAS FOR MSG CONTROL BLOCK (MSG) AND MSG
            =             DATA BYTES (MSG DATA)
            =             ADRMSG AND ADRMSGDATA ARE SET BY EXEC BEFORE THE
            =             CALL OF A MODULES MSG HANDLER
            =             */
  6   1     =   DCL RESTART BYTE EXTERNAL;
            =           /*
            =           FALSE IF START WITH SYSTEM RESET
            =           TRUE  IF RESTART WITHOUT SYSTEM RESET
            =           */
  7   1     =   DCL RTC (4) BYTE EXTERNAL;
            =             /* SYSTEM REAL TIME CLOCK LEAST SIGN. BYTE = RTC(3) */
  8   1     =   DCL INTMASK BYTE EXTERNAL;
```

```
                =           /*
                =           MASK OF CURRENTLY ACTIVATED INTERRUPTS
                =           */
  9   1         =   DCL DEBUGMCB (4) BYTE EXTERNAL;
                =           /*
                =           MCB FOR DEBUG PURPOSES
                =           */
                =
                =   /*
                =   ***************************************************************************
                =   **
                =                                   SYSTEM MESSAGES
                =   ***
                =   */
 10   1         =   DCL SYSMN0(4) BYTE EXTERNAL,
                =            /*
                =           INTERNAL START MSG, SENT TO ALL MODULES IN CPTR
                =           */
                =       SYSMN1(4) BYTE EXTERNAL,
                =           /*
                =           RESTART MSG, SENT TO MODULE TO BE ACTIVATED
                =           */
                =       SYSMN5(4) BYTE EXTERNAL;
                =           /*
                =           SUSPEND MSG, SENT BY EXEC TO MODULE TO BE SUSPENDED
                =           */
```

EXTERNAL DECLARATIONS OF EXECUTIVE DATA

```
                $EJECT
                $ INCLUDE(:F1:EXDATE.SRC)
          =     /*
          =     ****************************************************************
          =     ****************************************************************
          =     **
          =                              EXECUTIVE DATA
          =     ***
          =     */
   11   1 =     DCL MODSTATUS (MAXSYSMOD) BYTE EXTERNAL;
          =             /*
          =             MODSTATUS CONTAINS THE STATUS OF ALL MODULES
          =             IN THE SYSTEM. INDEX IS MODULE NUMBER.
          =             MODSTATUS IS UPDATED BY EXEC WITH SYS MSG 4
          =             (BIT 0 = LSB)
          =             BIT 0 - BIT 7 = 0: MODULE DOES NOT EXIST
          =             BIT 0: 1 - MODULE EXISTS
          =             BIT 1: 0 - MODULE NOT ACTIVATED
          =                    1 - MODULE ACTIVATED
          =             BIT 2: 0 - MODULE MAY NOT BE SUSPENDED
          =                    1 - MODULE MAY BE SUSPENDED
          =             BIT 3: 0 - MODULE NOT AUTHORIZED TO SUSPEND/
          =                       ACTIVATE OTHER MODULES
          =                    1 - MODULE AUTHORIZED TO SUSPEND/ACTIVATE
          =                       OTHER MODULES
          =             */
   12   1 =     DCL ADRSTAT ADDRESS EXTERNAL,
          =         (MODSTAT BASED ADRSTAT) (MAXMOD) BYTE;
          =             /*
          =             MODSTAT IS AN OVERLAY FOR MODSTATUS AND COVERS
          =             THE ENTRIES FOR ONE COMPUTER.
          =             THE BASE VARIABLE ADRSTAT IS SET AT START.
```

```
          =           */
   13  1  =    DCL BEGABSDATA BYTE EXTERNAL,
          =        EXTMSGLOCK BYTE EXTERNAL,
          =        EXTMSG BYTE EXTERNAL,
          =        EXTMSGIN BYTE EXTERNAL,
          =        EXTMSGOUT BYTE EXTERNAL,
          =        NUMEXTMSG BYTE EXTERNAL,
          =        EXTLASTMSG BYTE EXTERNAL,
          =        EXTMSGBUFFER(250) BYTE EXTERNAL,
          =        ENDABSDATA BYTE EXTERNAL;
   14  1  =    DCL (LOADSBC, START1, START2, START3) BYTE EXTERNAL;
   15  1  =    DCL ILLMSG (36) BYTE EXTERNAL;
          =           /*
          =           TEXT FOR ILLEGAL MSG TO CRT
          =           */
   16  1  =    DCL MONMSG (4) BYTE EXTERNAL,
          =        MONTEXT (26) BYTE EXTERNAL;
          =           /*
          =           SYSTEM MONITOR MSG TO CS
          =           */
   17  1  =    DCL EXTRMSG (4) BYTE EXTERNAL;
          =           /*
          =           EXTRACTION MSG TO DEBUG MODULE
          =           */
   18  1  =    DCL SAVESTACKPTR ADDRESS EXTERNAL;
          =           /*
          =           STACK POINTER AT SYSTEM START
          =           */
   19  1  =    DCL EXCLEARBEG BYTE EXTERNAL;
   20  1  =    DCL MSGEXTRACTION BYTE EXTERNAL;
          =           /*
          =           TRUE IF MSG EXTRACTION IS ACTIVATED
```

```
          =           */
   21   1 =    DCL CDCADR ADDRESS EXTERNAL,
          =        CDCACTIVE BYTE EXTERNAL,
          =        INTTBL(8) BYTE EXTERNAL;
   22   1 =    DCL RSTADR ADDRESS EXTERNAL,
          =        SNAPINTADR ADDRESS EXTERNAL,
          =        RSTFL BYTE EXTERNAL;
   23   1 =    DCL CODESAVE ADDRESS EXTERNAL,
          =        EXPERFL BYTE EXTERNAL,
          =        EXPERADR ADDRESS EXTERNAL,
          =        EXPERX BYTE EXTERNAL;
   24   1 =    DCL MSGBUFFER (MSGBUFLEN) BYTE EXTERNAL,
          =        (MSGIN, MSGOUT) ADDRESS EXTERNAL,
          =        NUMMSG BYTE EXTERNAL,
          =        LASTMSG ADDRESS EXTERNAL;
          =            /*
          =            MSGBUFFER IS A CIRCULAR FIFO LIST CONTAINING THE
          =              MSG WHICH ARE WAITING TO BE PROCESSED.
          =            MSGIN POINTS TO LOCATION OF NEXT MSG TO FILL IN.
          =            MSGOUT POINTS TO MSG TO BE PROCESSED NEXT.
          =            NUMMSG IS INCREMENTED WHEN A MSG IS SENT AND
          =              DECREMENTED WHEN A MSG IS PROCESSED.
          =              NUMMSG = 0: MSGBUFFER IS EMPTY.
          =            LASTMSG CONTAINS INDEX OF LAST BYTE OF LAST MSG
          =              IN MSGBUFFER + 1.
          =            */
   25   1 =    DCL PRIORLIST (MAXPRIOR) ADDRESS EXTERNAL,
          =        PRIORSCHEDULE BYTE EXTERNAL;
          =            /*
          =            PRIORLIST CONTAINS THE ADDRESSES OF THE ACTIVATED
          =            PRIORITY TASKS.
          =            A PRIORITY TASK IS CALLED WHEN IT IS SCHEDULED BY
```

```
                =           SETTING THE RESPECTIVE BIT IN PRIORSCHEDULE,
                =           E.G.  IF BIT 3 (LSB = BIT 0) IN PRIORSCHEDULE IS
                =           SET, THE ADDRESS IN PRIORLIST(3) IS CALLED.
                =           */
  26    1       =   DCL (XB1, XB2, XB3) BYTE EXTERNAL, (XA1, XA2, XA3) ADDRESS EXTERNAL;
                =           /*
                =           EXEC WORK VARIABLES
                =           */
  27    1       =   DCL PERXTBL (MAXPER) BYTE EXTERNAL;
                =           /*
                =           PERXTBL CONTAINS POINTER TO PERLIST IN COMPACT FORM
                =           0FFH - PERIODIC NOT ACTIVATED
                =           */
  28    1       =   DCL PERLIST (MAXPER) STRUCTURE (
                =              FREE BYTE,        .  /* TRUE - ITEM IS FREE */
                =              PERADR ADDRESS,     /* ADDRESS OF PERIODIC */
                =              PERINTADR ADDRESS,  /* ADDRESS OF TIME INTERVAL */
                =              PERTIME (4) BYTE) EXTERNAL,  /* NEXT CALL TIME (RTC FORMAT)
                 -    */
                =           /*
                =           PERLIST CONTAINS ALL SIGNIFICANT DATA OF AN
                =           ACTIVATED PERIODIC TASK
                =           */
                =          (PERINT BASED XA1) (4) BYTE,
                =           /*
                =           OVERLAY FOR TIME INTERVAL OF A PERIODIC
                =           */
                =          PERX BYTE EXTERNAL,
                =           /*
                =           PERX IS SEARCH INDEX FOR PERLIST
                =           */
                =          NUMPER BYTE EXTERNAL;
```

```
           =            /*
           =            NUMBER OF ACTIVATED PERIODICS
           =            */
  29   1  =    DCL EXCLEAREND BYTE EXTERNAL;
```

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22354	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
4. Professor U.R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
5. Professor N. Schneidewind, Code 52Ss Department of Computer Science Naval Postgraduate School Monterey, California 93940	1

6. LCDR F. Burckhead, Code 528g 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

7. Marineamt - A1 - 1
2940 Wilhelmshaven
Federal Republic of Germany

8. Dokumentationszentrale der Bundeswehr (See) 1
Friedrich-Ebert-Allee 34
5300 Bonn
Federal Republic of Germany

9. Kapitaenleutnant Wolfgang Niemann 1
Kommando Marinefuehrungssysteme
Wibbelhofstrasse 3
2940 Wilhelmshaven
Federal Republic of Germany