General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

GASP-PL/I SIMULATION OF INTEGRATED AVIONIC SYSTEM PROCESSOR ARCHITECTURES

by
Glen A. Brent
BS

A thesis submitted in partial
fulfillment of the requirements for
the Master of Science Degree in Engineering.

Performed under NASA grant NASA-NSG-2238.

Department of Electrical Sciences and Systems Engineering
in the Graduate School
Southern Illinois University
September 1978

ACKNOWLEDGEMENTS

This research effort was funded in part by the National Aeronautics and Space Administration, Ames Research Center under grant NASA-NSG-2238, entitled "Development of GASP-PL/I Simulation." The NASA Technical Officer for this grant is Mr. George Callas, Ames Research Center, National Aeronautics and Space Administration, Moffett Field, California 94035.

The author wishes to express his sincere appreciation to Dr. Thomas McCalla, advisor, for his assistance and support throughout this project. Many others have contributed in one way or another. Their names are too numerous to mention, but their efforts are very much appreciated.

TABLE OF CONTENTS

FIGURES

TABLES

1. INTRODUCTION

General Aviation electronics has traditionally used separate, single-function instruments and avionics, making the cockpit a maze of dials and gauges. Recently , NASA sponsored two developmental studies aimed at integrating and improving general aviation avionics. One was performed at Southern Illinois University under NASA contract NAS2-9310.[1] Completed in July 1977, this study proposed a complete integration of all aircraft instrumentation into a single modular system. Instead of using the current single-function aircraft instruments, computers compile and display inflight information for the pilot's use. Having all the inflight information available to a medium capability computer allows many new possibilities in navigation, inflight diagnositics, autopilots, etc.

A new processor architecture called the Team Architecture was also proposed. This is a hardware/software approach to high-reliability computer systems. Safety requirements for aircraft require that the system be fault tolerant. The Team Architecture uses hardware and software redundancy, plus state-of-the-art technology in a no-critical element system. The goal is a system that never has a complete failure.[1].

The research presented is a follow-up study of the proposed Team Architecture. Some questions arose as to the efficiency and capabilities of the new architecture. In this study, GASP-PL/I simulation models are used to evaluate the operating characteristics

of the Team Architecture.

Following sections present the problem, model development, simulation programs and results at length. The appendices contain program input formats, outputs, and listings, not including the GASP-PL/I programs.

2 PROBLEM DESCRIPTION

The advent of microprocessors resulted in significant changes in the way many design problems are approached. Applications from home computers to automobiles have been found for these inexpensive, small, and powerful computers. Microprocessors are expanding into many areas where computer processing has been impractical.

One new application is in general aviation aircraft avionics. Currently, some systems available use microprocessors for frequency storage and navigation calculations. Taking a closer look at the aircraft, the possibilities for aircraft instrumentation applications are unlimited. Aircraft are essentially large instrumentation systems in which everything from airspeed to engine oil temperature is monitored. Since inflight information is collected continually, why not input it to a medium capability computer system? Calculations and analyses could be performed by the computer to present a concise picture of the aircraft situation to the pilot. This is the basic concept behind the Integrated Avionic System (IAS).[1]

Designed to replace all of the current instruments and avionics in general aviation aircraft, the IAS uses microprocessors and state of the art technology. High reliability is obtained through hardware redundancy and a unique Team Architecture concept. The Team Achitecture is a hardware/software system which is designed to prevent complete failures.[1]

On the hardware side of the Team Architecture is the IAS Central Processor (CP). Since a large amount of computing power is required,

a multi-processing approach is used. Four microprocessor systems comprise the CP. Using four microprocessors increases both computing power and reliability. Each microprocessor subsystem in the CP is called a Central Processor Subsystem (CPS). In the Team Architecture, every CPS has access to any device in the system and is capable of running the IAS independently of the others. CPSs share jobs in the IAS and check each others performance.[1]

The Team Operating System (TOS) is the software half of the Team architecture. Software for the IAS must be reliable, fast, and accurate. The TOS must insure that all IAS jobs are performed within the correct time frame. Aircraft instrumentation is a real-time problem and the IAS must keep up with the various data rates. Diagnostics must also be included to insure that the IAS is running properly.[1]

Reference 1 describes the IAS in detail. The Team architecture concept is a new approach to high reliability computer systems. On paper only, much needs to be learned about hardware and software configurations before a prototype IAS can be built. Many CP configurations must be studied to obtain the best mix of capability, reliability, and cost.

The developmental system study (reference 1), proposes a modular system configuration using the IEEE 488 standard instrumentation bus. A table-driven operating system was proposed in which the IAS jobs are executed by cycling through a set of table entries by levels. Jobs in the IAS cover a wide range of execution times, I/O requirements, and data rates.

This research project is a computer simulation of the IAS CP using GASP-PL/I models. A general-purpose simulation model is developed, and its use is demonstrated by studying the IAS CP. Specifically, four operating characteristics of the Team Architecture are evaluated:

1. The computing overhead required to manage and operate the Team Operating System. Because the TOS uses four CPS units, this is evaluated for the individual CPS units, and the entire system.

2. The load on the system bus and how long a CPS must wait to get a bus.

3. The time a CPS spends waiting to update the Team Operating System. This is part of the operating system overhead, but is directly affected by the system bus load.

4. The affect of different types of programs on the operating system. Factors which increase or decrease the operating system overhead.

Hardware configurations of the CP, CPS, system bus, and the TOS programs are described in the next sections.

3 THE INTEGRATED AVIONIC SYSTEM

The IAS is a modular, bus-oriented system. The major system components are the subsystems and the system bus (Figure 3.1). Each subsystem performs one of the functions required in the aircraft, such as, navigation, communication, display processing. Four of the IAS subsystem are called Central Processor Subsystems (CPS). These four subsystems comprise the Central Processor (CP), which is the controller for the entire IAS.[1]

The CP uses a unique approach to high reliability computer architecture called the Team Architecture concept. This is a hardware/software architecture which is designed to provide high reliability without sacrificing computing power. The main components of the Team Architecture are the CPS units and the Team Operating System (TOS). Each CPS has equal authority in the system. A dedicated controller is not used, because this would be a critical link that could cause the entire IAS to fail.[1]

The TOS is a table-driven operating system which is resident in all of the CPS units. Each CPS runs the TOS when needed, and all four share the system workload. IAS jobs are run by cycling through the TOS tables and running jobs in a specific order and frequency. The IAS is a real-time application and all jobs must be run above a minimum frequency to insure that data is not lost.

The CPS units are Motorola M6800 microprocessor systems. The main components in each CPS are a microprocessor, a block of memory, and a Bus Interface Module (BIM). The BIM interfaces the CPS to the

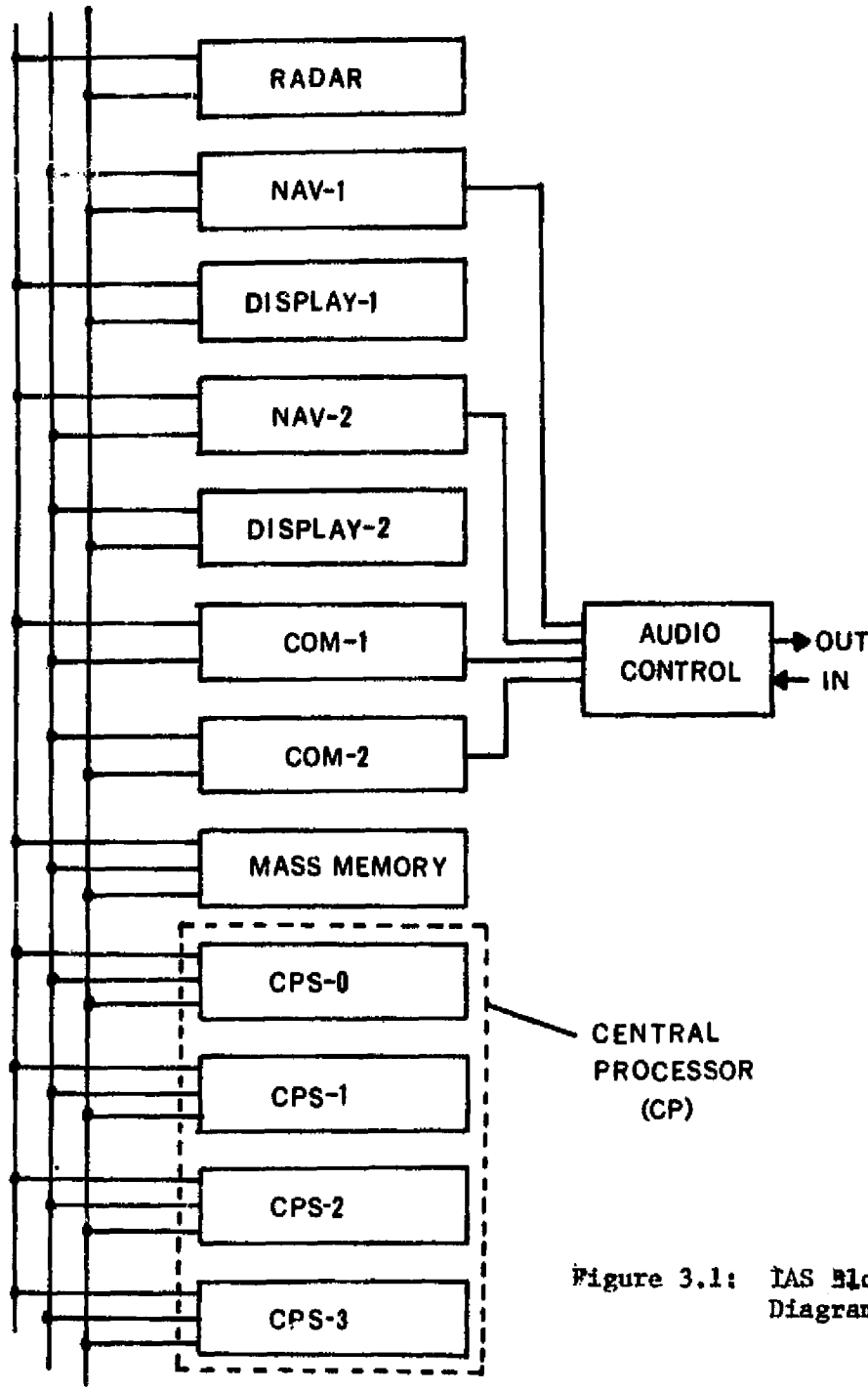


Figure 3.1: IAS Block Diagram.

system bus. The CPS shares a block of memory with the BIM which contains the TOS tables and a input/output buffer area. All messages to other subsystems from the CPS are sent through the BIM using a buffer. This allows the CPS to load a buffer, then let the BIM send the data to the receiving device(s), leaving the CPS free to perform other calculations.

The system bus is a triple-redundant IEEE 488 standard instrumentation bus. All system communication is transferred over the bus using the interface functions and handshakes specified by the standard: IEEE 488-1975.[2] Thus each BIM must be capable of implementing any interface function that is required to transfer information to and from the CPS and other subsystems.

The system design, as described in reference 1, covers the basic structure of the IAS and the Team Architecture. However, much of the CPS organization had to be defined in greater detail before being simulated. A major change was the use of the Motorola M6800 microprocessor instead of the Intel 3000.

A wide range of family devices, widespread use in other applications, and the large amount of documentation and support software available are the major reasons why the M6800 family was chosen. The M6800 is a versatile microprocessor, having six addressing modes and 72 instructions. Accessory devices include parallel I/O chips, serial I/O chips, and many others.[3]

3.1 CPS HARDWARE CONFIGURATION

The CPS is essentially a standard Motorola M6800 microprocessor system. M6800 family components are used exclusively, eliminating the need for special one-of-a-kind circuitry. The basic CPS is a M6800 microprocessor which is connected via an 8-bit data bus, and a 16-bit address bus to a block of memory and one M6821 peripheral interface adaptor (PIA) (Figure 3.2). The PIA is used to send commands to the BIM to control I/O on the system bus.

Although the final IAS CPS may have a different configuration, the memory was configured as follows (Figure 3.3). Each CPS has 10K bytes of memory, including 2K bytes of ROM for the TOS programs. The lower 2K bytes of RAM (address 0000 to 07FF hexadecimal)* are shared with the BIM and contains two of the TOS tables, status words, input buffers, and output buffers. Addresses 0800 to 0803 contain the CPS PIA which interfaces with the I/O processor in the BIM.

The remaining two TOS tables are contained in the block of memory starting at 0810 and ending at 08FF. This area also includes a group of working registers used by the TOS and IAS programs. Standard locations are picked for storing different information to simplify the operating system programs.

The TOS programs are contained in 2K bytes of ROM starting at

---

*All memory addresses will be written in hexadecimal from this point on, as will all commands and messages used in the TOS.

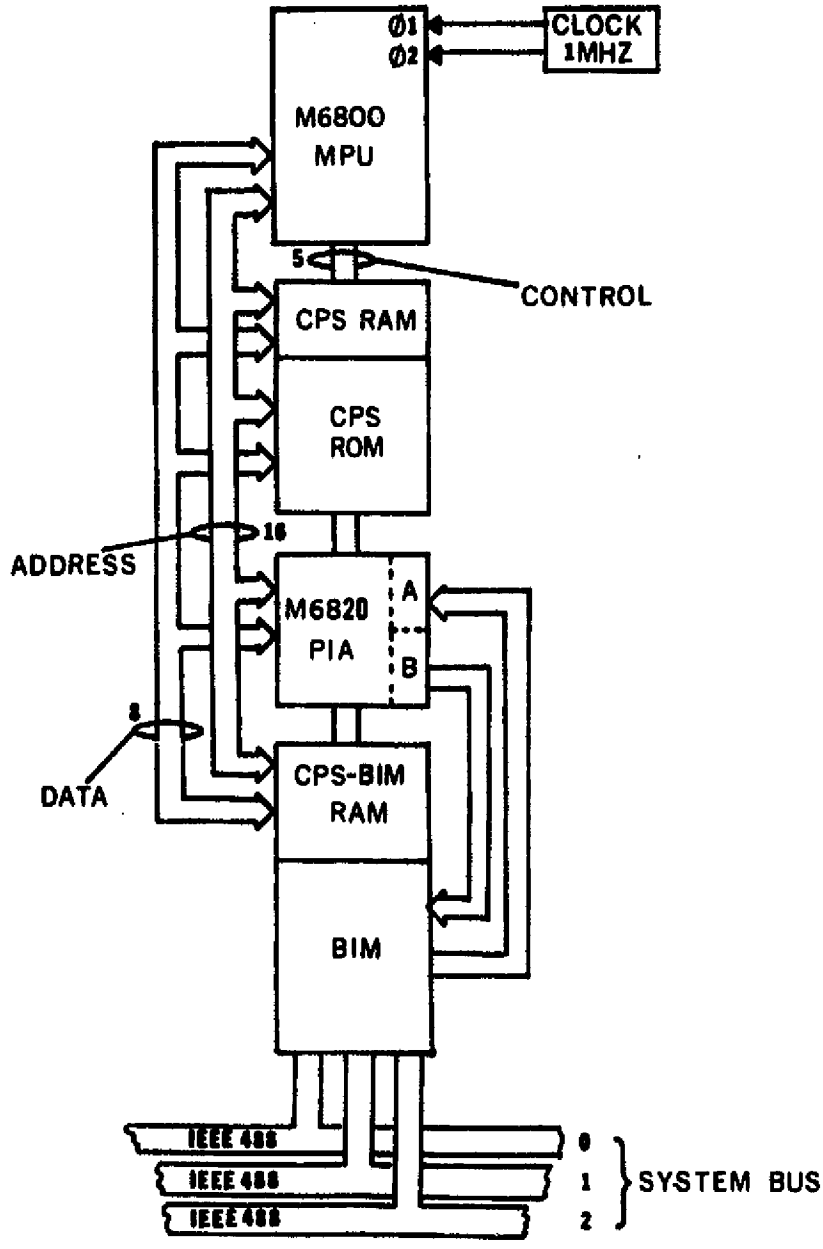


Figure 3.2: CPS Block Diagram.

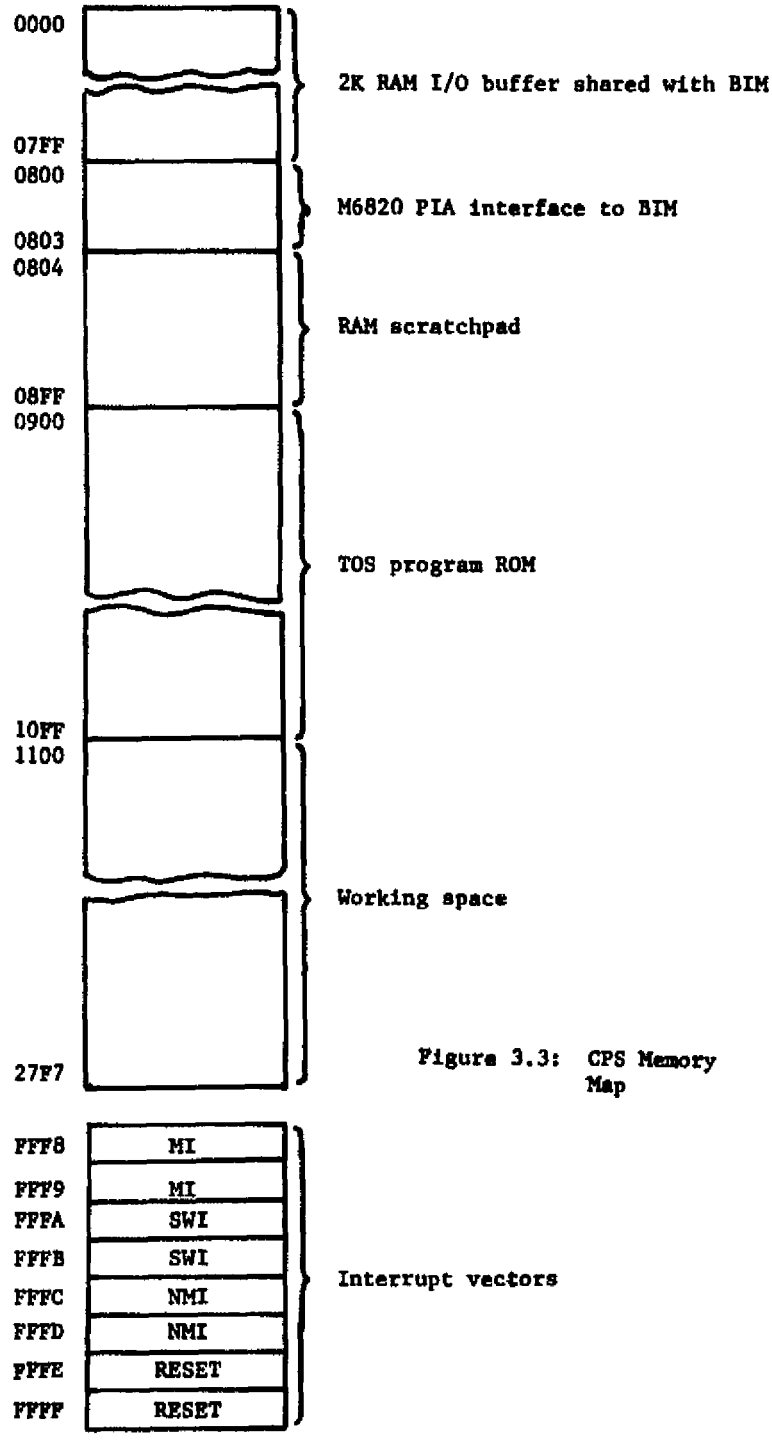


Figure 3.3: CPS Memory Map

address 0900 and ending at 10FF. The balance of the memory is used for working space for the IAS programs. Memory configuration will be discussed further in section 3.3 when the TOS is described in detail.

All four CPS units are configured as shown in Figure 3.2. In the team architecture, all CPS units have equal capabilities and equal access to all subsystems. By configuring all four exactly alike, any of the CPS units can run any of the software. The IAS does not have a dedicated controller which could cause the entire system to fail.

3.2 THE BUS INTERFACE MODULE

Each CPS is interfaced to the system bus using a stand-alone unit called a Bus Interface Module (BIM). The BIM is also a microprocessor controlled system and shares 2K bytes of RAM with the CPS. Using a microprocessor controlled interface as an I/O channel allows the CPS to continue with IAS programs and not handle time consuming I/O.

The system bus is a triple-redundant IEEE 488 standard instrumentation bus.[1] The IEEE 488 bus provides a uniform method for interfacing different devices together in a multi-purpose instrumentation system. The standard provides not only the hardware specifications, but also the interface functions required for reliable operation of the bus. Interface functions are provided for any type of device requirements.[2] The IEEE 488 bus has a maximum data rate of one mega-byte per second, although the BIM limits the bus to a lower rate. A closer look at the BIM requirements and tasks shows which interface functions are needed to perform every CPS task.

The BIM has two main subdivisions, the I/O processor and the interface unit (Figure 3.4). The I/O processor communicates with both the CPS and the interface unit. Commands from the CPS and the bus (via the interface unit) are decoded by the I/O processor which generates responses.

The interface unit is a hardware implementation of a subset of the interface functions described in IEEE standard 488-1975. It would be possible to implement any interface functions with software using a microprocessor, if the standard did not specify response to the ATN and IFC commands be completed in less than 200 nano-seconds. However, since a microprocessor cannot respond that quickly, a hardware implementation must be used.

To eliminate confusion, the following convention will be used to describe messages in the IAS. Messages sent between the CPS and BIM will be call CPS messages. Data and commands sent over the system bus will be called interface messages. These are defined by the standard. Messages exchanged between the I/O processor and the interface unit will be called local messages.

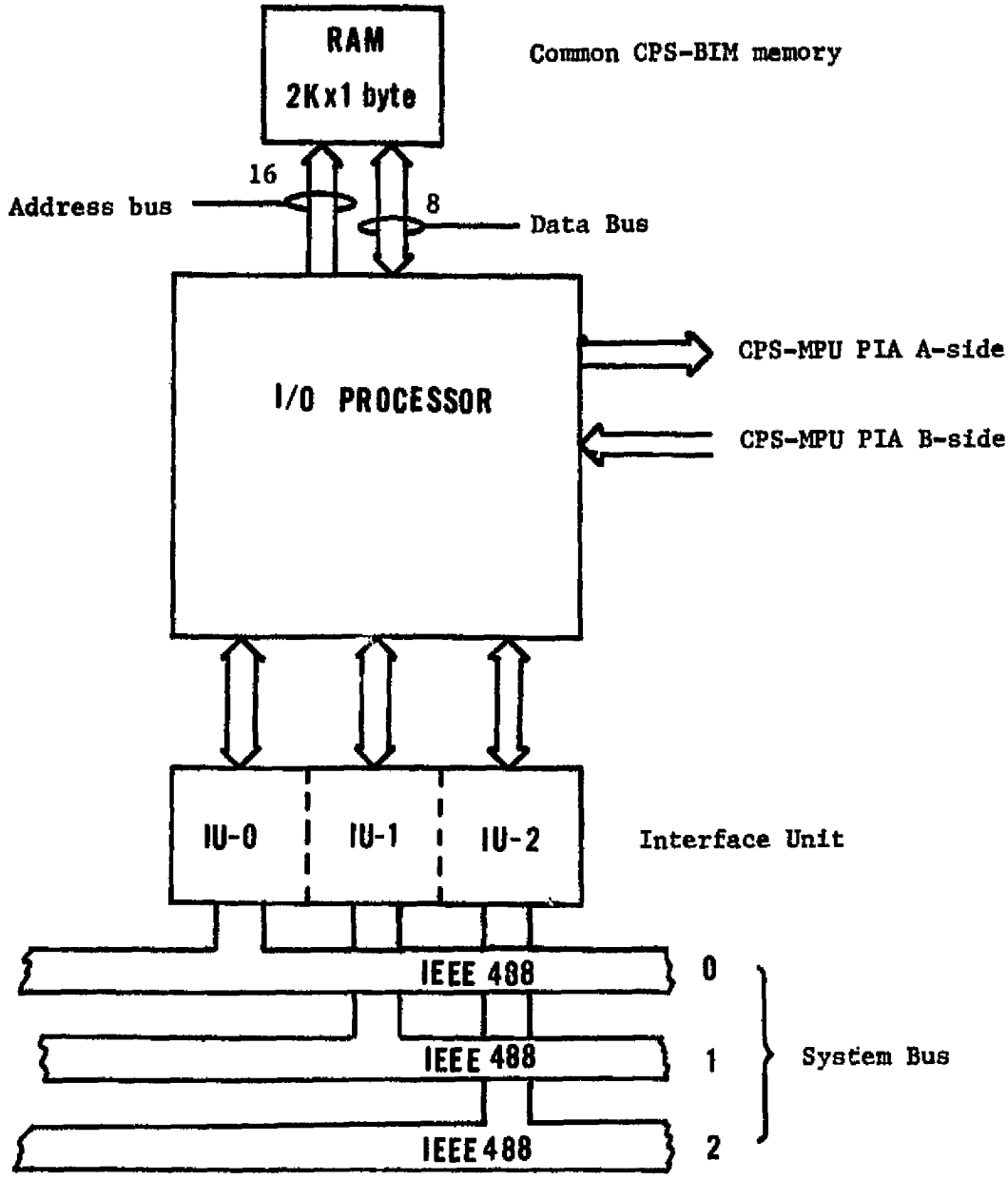


Figure 3.4: Bus Interface Module (BIM) Block Diagram.

The BIM must perform the following tasks on request of either the CPS or the system:

1. Clear its internal registers and reset to the idle state. All current tasks are stopped and any I/O in process is lost.

2. Send the output buffer to one or more receiving devices. The BIM must first get control of one of the system busses and then address all the listening devices before the transfer is performed.

3. Get control of one of the three system busses to perform any of the required actions. This includes resetting the system, addressing devices, servicing requests, running serial bus polls, and transferring data buffers.

4. Get control of a system bus and inform the other BIMs that the TOS is being updated. To prevent conflicts in the operating system, only one CPS may update the TOS at a time.

5. Send TOS table update information to the other CPS BIMs.

6. The BIM must be capable of receiving a data buffer from another CPS BIM or a subsystem.

7. The BIM must respond to all bus commands and service requests from other subsystems.

8. The BIM must be able to receive TOS table update information from the other CPS BIMs.

9. The BIM must send its status and other requested information to the CPS.

All of the above functions are performed by the BIM on command from either the CPS or the system bus. It should be noted that all data transfers across the system bus are performed by the BIM independently of the CPS. The CPS initiates the transfer by commands, and the BIM responds with a completion message when through.

To accomplish the above tasks, the following interface functions as defined by the IEEE 488 bus standard are required:[2]

1. Source Handshake (SH) function.
2. Acceptor Handshake (AH) function.
3. Talker (T) function including the serial poll capability.
4. Listener (L) function.
5. Service Request (SRQ) function.
6. Controller (C) function including:
    a. Controller service request states.
    b. System controller states.
    c. System control interface clear states.

Along with the interface functions listed above, the I/O processor and the interface unit must be capable of generating the local and interface messages required. The above list of interface functions is a subset of the entire set defined in the standard. Table 3.1 lists the interface functions, the subset numbers (see reference 2), and the messages required.

For this study, the interface unit will be treated as a black box which performs the above interface functions. For the final system, the interface would be a sequential logic circuit capable of performing the state diagrams as described in reference 2 for the functions listed in Table 3.1. A detailed sequential logic design is not required because of the strict definition of the IEEE 488 standard instrumentation bus.

Table 3.1: IEEE 488 Bus Interface Functions Implemented in the Interface Unit.[2]

FUNCTION	SUBSETS	MESSAGES REQUIRED
SOURCE HANDSHAKE	SH1	ATN, RFD, DAC
ACCEPTOR HANDSHAKE	AH1	ATN, DAV
TALKER*	T6	MLA, STB, DAB, EOS, END, RQS, IFC, ATN, MTA, SPE, SPD, OTA
LISTENER	L4	IFC, ATN, UNL, MLA, MTA
SERVICE REQUEST	SR1	SRQ
CONTROLLER**	C1, 2, 4, 7	IFC, ATN, TCT, rsc, gts, tcs, sic

* Includes the Serial Poll states.
** Includes the System Controller and Service Request states.

The speed at which data is transferred across the system bus is a function of the I/O processors and not the interface units. To conform to the standard, the interface unit must be capable of a one mega-byte per second data rate.[2] For example, if the I/O processor is a Motorola M6800, with a minimum clock cycle-time of one microsecond and a minimum instruction execution length of two cycles, the maximum data rate the processor could generate is less than 500 kilo-bytes per second. Since the interface unit must be capable of performing the SH-AH cycle at one mega-byte per second, it is the I/O processor and not the interface unit which limits the system bus data rate.

To perform the BIM tasks listed earlier a set of CPS messages

were defined. These are one-byte words which request an action to be taken by the BIM. Table 3.2 is a summary of BIM commands. Most of these result in the BIM returning completion of task messages (Table 3.3). It should be noted that the messages defined here are software functions and are not fixed.

Table 3.2: BIM Commands.

MNEMONIC	MEANING	BIM ACTION
SND	SEND DATA BUFFER	The BIM will get control of a system bus and send the buffer starting at the address specified in the Output Buffer Starting Address (OBSA). A 'TRS' message is sent to the CPS MPU when the transfer is started and the 'TRC' is sent when the transfer is complete. The CPS MPU must not alter the OBSA until the 'TRS' is received.
GTC	GET CONTROL	The BIM will get control of a system bus and inform the other CPS MPUs that it will be sending TOS update information. The 'BSC' message is sent to the CPS MPU when the BIM is ready to send the TOS update information.
STU	SEND TOS UPDATE	This message must have been preceded by a 'GTC.' The BIM, upon receipt of the 'STU' will send the TOS table update information to the other BIMs. The 'TUS' message is sent to the CPS MPU when the transfer is started and a 'TUC' is sent when completed.
BCR	BIM CLEAR	The BIM stops all I/O in process and returns to the idle mode. This command is generally used to reset the BIM during start-up.

Reference 1 did not specify the exact hardware and software configurations of the BIM. The following description was developed and modeled. Other configurations are possible, but were not studied in detail. Figure 3.5 shows the hardware configuration of the BIM. A Motorola M6800 microprocessor is used as the I/O processor. Included are 2K bytes of RAM shared with the CPS MPU, a block of scratchpad RAM, and a block of ROM which contains the BIM control programs. Interrupts are prioritized using a hardware priority circuit. This reduces the response time of the BIM to inputs.

Table 3.3: BIM Responses.

MNEMONIC	MEANING	REMARKS
BER	BUS ERROR	During the last operation a system bus error occurred.
BFE	BUFFER ERROR	The output buffer was formatted incorrectly, the data was not sent.
BSC	BUS CONTROL OBTAINED	The BIM has obtained control of one of the system busses.
DIN	DATA BUFFER RECEIVED	The BIM has just received a data buffer from another device.
TRC	TRANSFER COMPLETE	An output buffer has just been sent without any errors.
TRS	TRANSFER STARTED	The output data buffer is being sent. The CPS can now modify the OBSA.
TUC	TABLE UPDATE COMPLETE	The BIM has just completed transferring the system table update information.
TUS	TABLE UPDATE STARTED	The BIM has started sending the system table update.

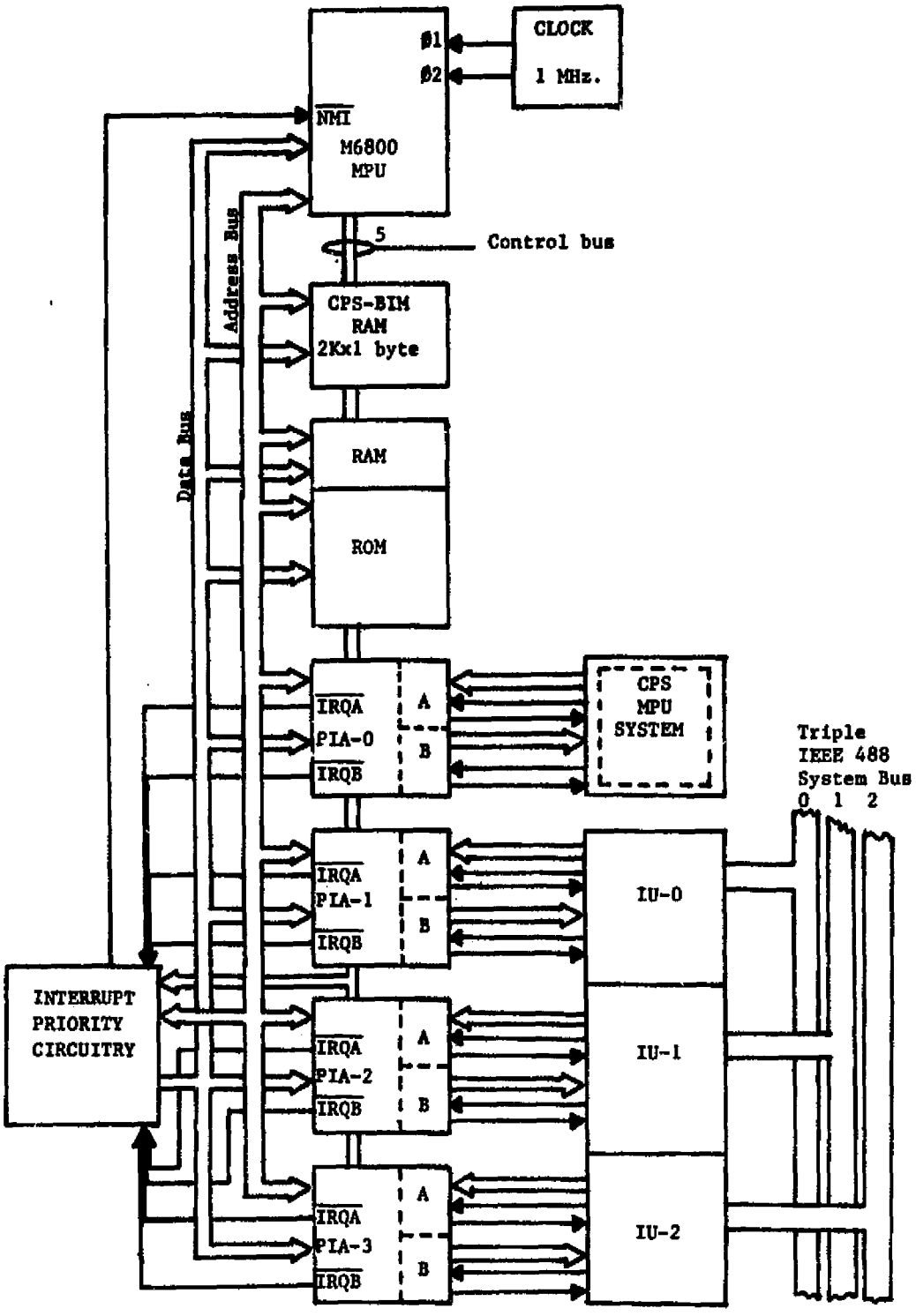


Figure 3.5: BIM Hardware Configuration.

Interfacing the I/O processor to the CPS MPU is a M6821 Peripheral Interface Adapter (PIA). This is connected as shown in Figure 3.6. On both PIAs, the A-side is used for input, and the B-side is used for output. After initialization, both PIAs are programmed as follows:

A-side: Handshake input mode.
Interrupt on negative transition of CA1.

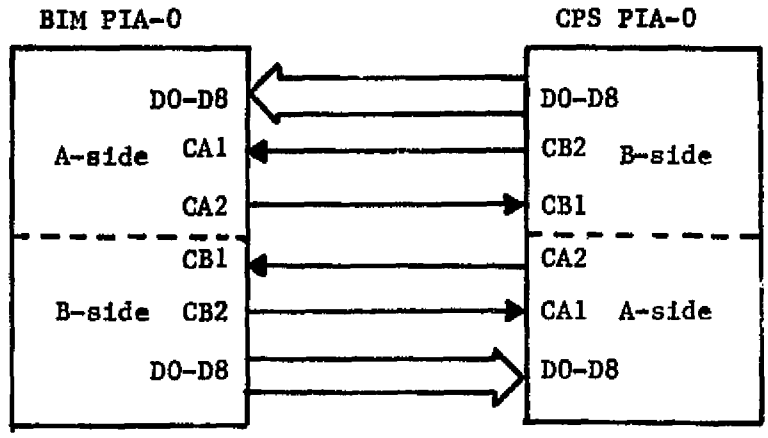
B-side: Pulse output mode.
Interrupts are disabled.

The interface unit is three identical IEEE 488 bus interfaces. Each one is connected to the I/O processor through a PIA. All I/O on the system bus, and local messages pass through these three PIAs. The three interfaces are not interconnected to provide redundancy.

Software for the BIM must be as fast as possible, while still performing all of the BIM functions reliably. A four-mode software operating system was proposed for the BIM (Figure 3.7).

During power-up, the BIM automatically starts in the idle mode. In this mode, the BIM is waiting for a command from either the CPS MPU or the system bus. If the BIM receives a "BCR" command while in any mode, it will immediately return to the idle mode. All current operations are lost and the BIM is re-initialized. This allows the CPS to reset the BIM at any time.

All inputs are processed using interrupts. The BIM knows the current mode, the current interface unit state, and how each input is to be processed. To execute the input in the fastest possible time a hardware interrupt priority circuit is used.



Initial status:

CRA= 00100101
CRB= 00101100
DDRA= 00000000
DDRB= 11111111

A-side:

Handshake input
Interrupt on negative CA1 transition

B-side:

Pulse-mode output
Interrupts disabled

Figure 3.6: CPS to BIM PIA Interface Configuration.

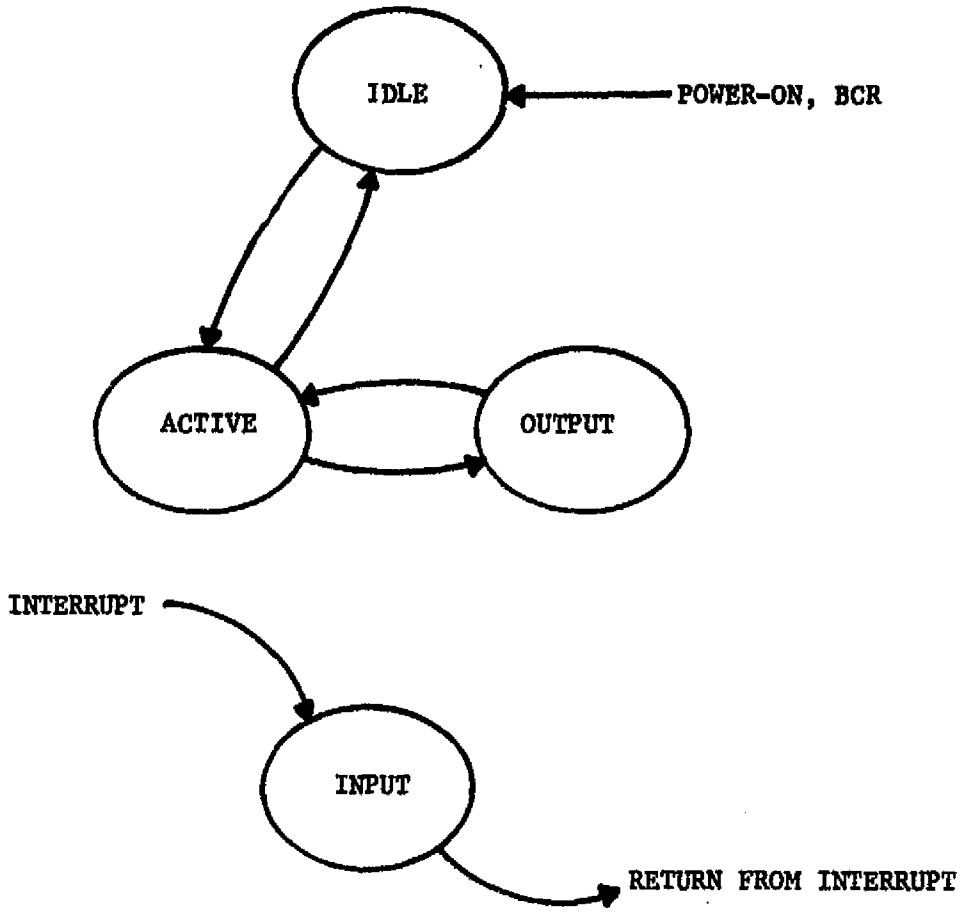


Figure 3.7: BIM Software Modes.

While in the active mode, the BIM will execute any commands received and return to the idle mode. The BIM commands, "SND" and "GTC", cause the BIM to enter the active mode. Three local messages also cause the BIM to enter the active mode. The messages "tac" (talker active), "spa" (serial poll active), and "srq" (service request pending), all require the I/O processor to receive or send messages over the system bus. See Table 3.4 for meanings of each of the commands.

The "SND" command informs the BIM that a data buffer is ready to be sent. The starting address of the buffer is in locations 0700 and 0701. Output buffers are formatted according to Figure 3.8, with the first address containing the number of bytes to be sent. The system bus addresses of the receiving devices are in the next higher

Table 3.4: Interface Messages Used.[2]

MESSAGE	MEANING
ATN	attention
DAB	data byte
DAC	data accepted
DAV	data valid
END	end of message
IDY	identify
IFC	interface clear
MLA	my listen address
MTA	my talk address
OTA	other talk address
RFD	ready for data
RQS	request service
SPD	serial poll disable
SPE	serial poll enable
SQR	service request
STB	status byte
TCT	take control
UNL	unlisten

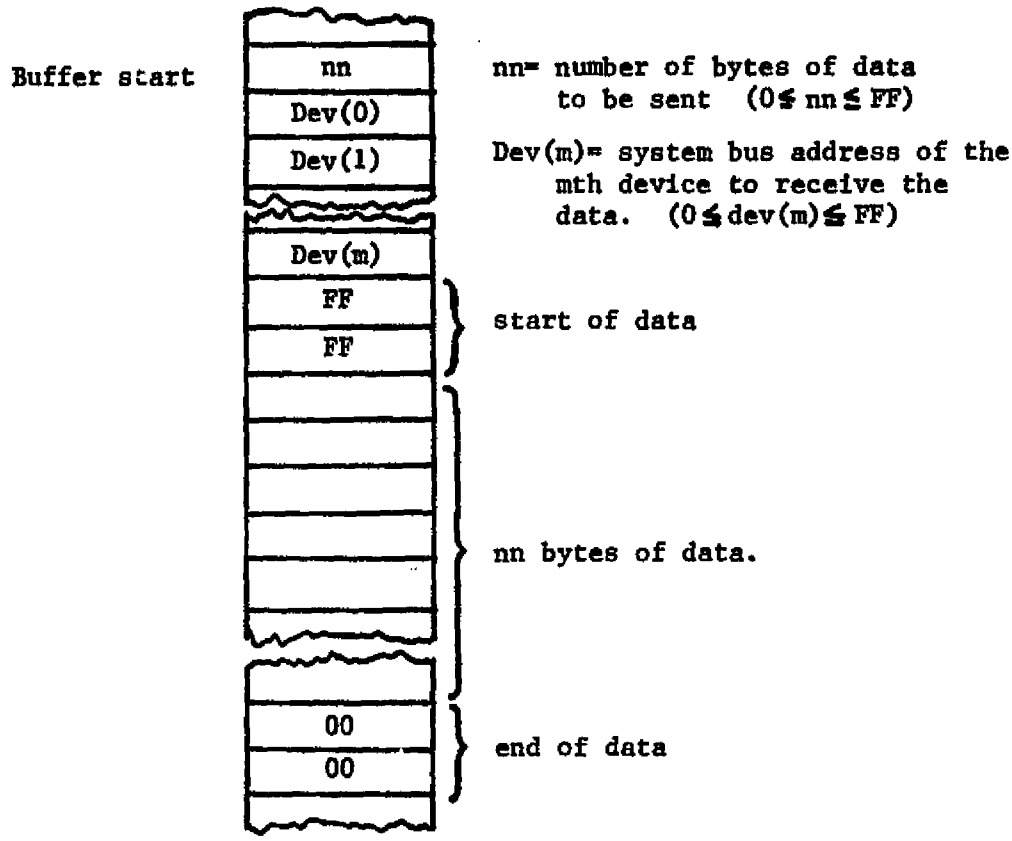


Figure 3.8: BIM Output Data Buffer Format.

locations. Each address is one byte long, and the last address is followed by two locations containing FF. This is an invalid address and is only used to signify the start of the data. The data to be sent is next, followed by two locations containing the null value, 00.

When the I/O processor receives the "SND" command, it will first get control of a system bus, if it does not already have control of one (Figure 3.9). Once control is obtained it will read the buffer start address, from scratchpad memory. A "TRS" message is sent to the CPS MPU, and the transfer is started. The CPS MPU must not change the buffer start address until the "TRS" is received.

Next, the BIM will address all listeners, and go to the output mode. In the output mode the data buffer is sent over the system bus using the SH-AH handshake cycle. When the buffer has been sent, the BIM re-enters the active mode, de-addresses all of the listeners, and sends the "TRC" (TRANSFER COMPLETE) message to the CPS MPU. After completing all pending commands, the BIM returns to the idle mode.

When a "GTC" (GET CONTROL) command is received, the BIM will first determine if any CPS is updating its TOS tables (Figure 3.10). If one is, the BIM will enter a wait loop until the current TOS updater is finished. The BIM will then get control of a bus, and send a message to the other CPS BIMs that it is now updating the TOS. Once a bus is obtained, the "BSC" (BUS CONTROL OBTAINED) message is sent to the CPS MPU, and the BIM stays in the active mode, waiting for the

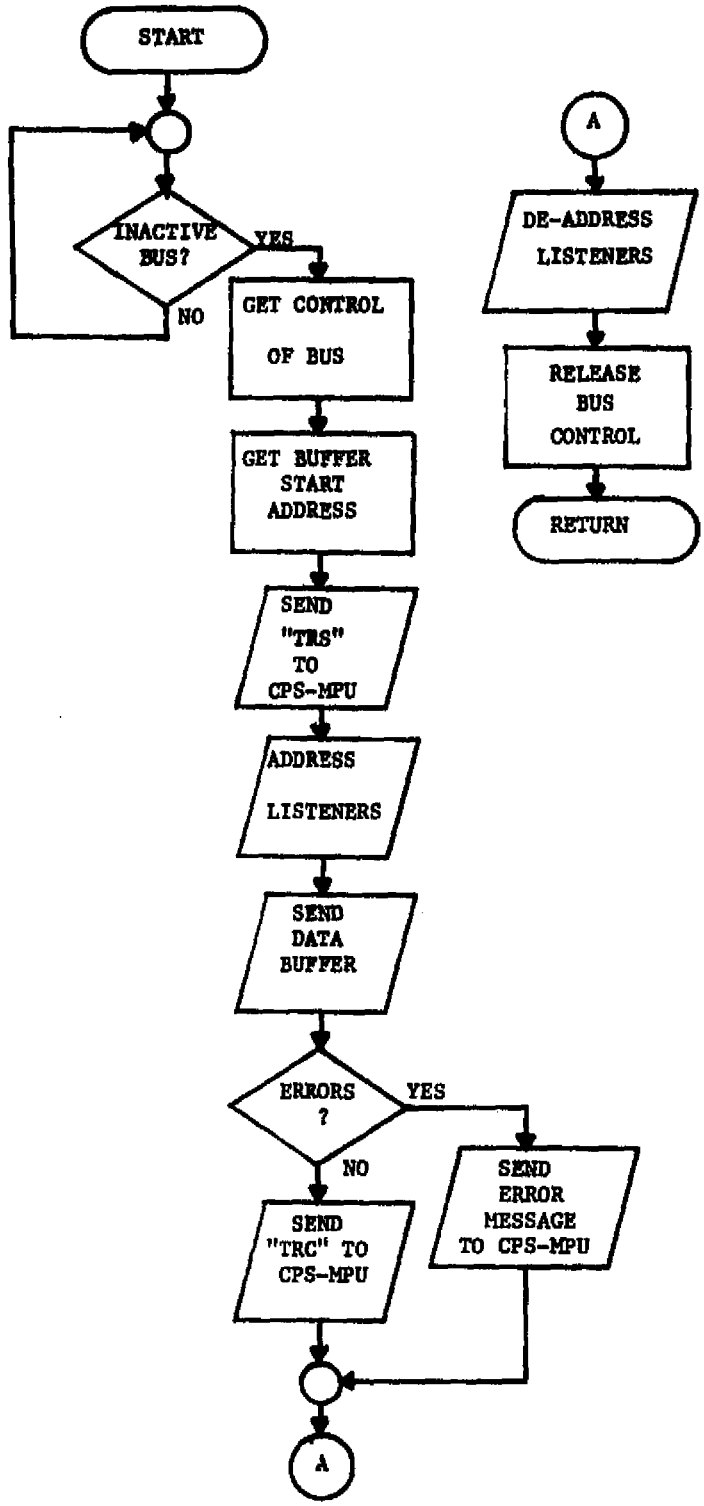


Figure 3.9: SND Command Execution Flowchart.

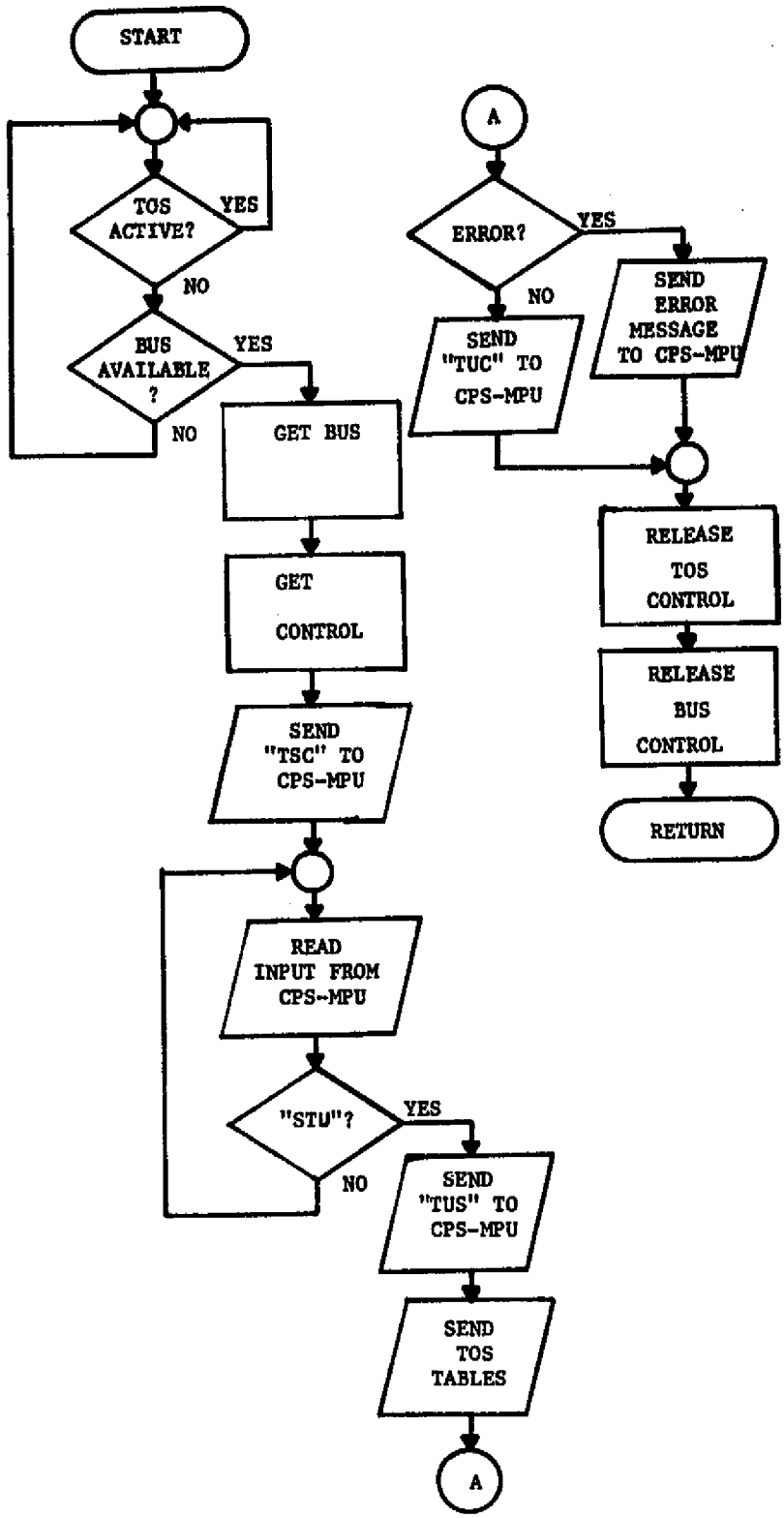


Figure 3.10: GTC Command Execution Flowchart.

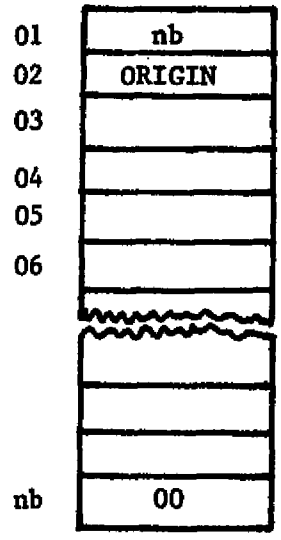
"STU" (SEND TABLE UPDATE) command. The BIM will ignore any other command from the CPS MPU, except a "BCR."

On receipt of the "STU" command, the BIM sends the TOS table entries to the other CPS BIMs. When finished, the bus is released, the "TUC" (TABLE UPDATE COMPLETE) message is sent to the CPS MPU, and the BIM returns to the idle mode. The "STU" command must always be preceded by a "GTC," or the BIM will send an error message and reset.

The BIM can only be controller on one bus at a time, and only be a talker on one bus at a time. However, because all inputs are specified to be performed under interrupt control, the BIM is capable of being a listener on more than one bus at a time.

Input data is stored in the input buffer area. The first data byte received contains the number of bytes to be inputted, and the second contains the origin address. The I/O processor will reserve a block of the input buffer area for the inputs, and fill it as the data comes in. Once complete the BIM sends the "DIN" message to the CPS MPU. The input buffer uses the same format as the order in which the data is received (Figure 3.11). The maximum blocksize is 255 bytes, including the origin address and the number of bytes.

Local messages are sent between the I/O processor and the interface unit. Table 3.5 lists all the local message defined. These were defined to facilitate the estimation of BIM function execution times.



nb -The number of bytes in the message including the ORIGIN address and number of bytes word.

ORIGIN -The address of the device which sent the message.

Figure 3.11: Input data buffer format.

Table 3.5: Local Messages.[2]

MESSAGE	MEANING
gts	go to standby
nba	new byte available
rdy	ready
rsc	request system control
rsv	request service
sic	send interface clear
tac	talker active
tcs	take control synchronously

3.3 TEAM OPERATING SYSTEM

The Team Operating System (TOS) is a multiprocessing operating system designed for high reliability. Instead of a dedicated controller, the TOS allows any of the "team members," the CPS units, to update the operating system when needed. The CPS units share both system and TOS workload on an equal basis. When a CPS completes a task, it determines which task it will execute next. Before beginning execution it schedules the subsequent task, and sends this information to the other CPSs. The next CPS to complete a task will update the system again.[1]

A CPS first determines if another CPS is updating the TOS. If not, then the CPS sends the "GTC" command to the BIM. While waiting for the BIM to get control of a bus, the CPS performs diagnostics on its own system, or doing other low priority jobs.

Data rates in the IAS vary. Fuel supply, altitude, and engine

oil pressure change relatively slowly compared to airspeed, and aircraft atitude. The TOS must run the tasks that update high data rate parameters at a higher frequency than the low data rate jobs. A method is needed to assure that all jobs are executed often enough to prevent any loss of information.

Reference 1 proposes a table-driven operating system for the IAS. This operating system, hereafter referred as the TOS, uses three tables: The Task Table (TTBL), the Task Queue Table (TQTBL), and the Level Table (LTBL). In addition, a Next Task Register (NTR) is used to indicate the next job to be executed (Figure 3.12). Each CPS has a copy of the TOS tables and the TOS programs resident in its memory.

The TOS tables were designed for a 16-bit microprocessor. Implementing them in the M6800, which is an 8-bit machine, requires some minor changes, but the operation is the same.

The TTBL is a list of the IAS tasks and starting addresses by task ID numbers. Three memory bytes are required to store each entry. IAS tasks are given an ID number by their position in the TTBL. This defines their position in both the TTBL and the TQTBL. Each TTBL entry contains the task number in the least si gnificant seven bits of the first byte, plus either the starting address of the routine which performs the task or a file number. If the most significant bit of the first entry is 0, the entry contains an address and the routine is in memory. If it is a 1 the program is not in memory, and the second two bytes contain the file number under which the task can be found.

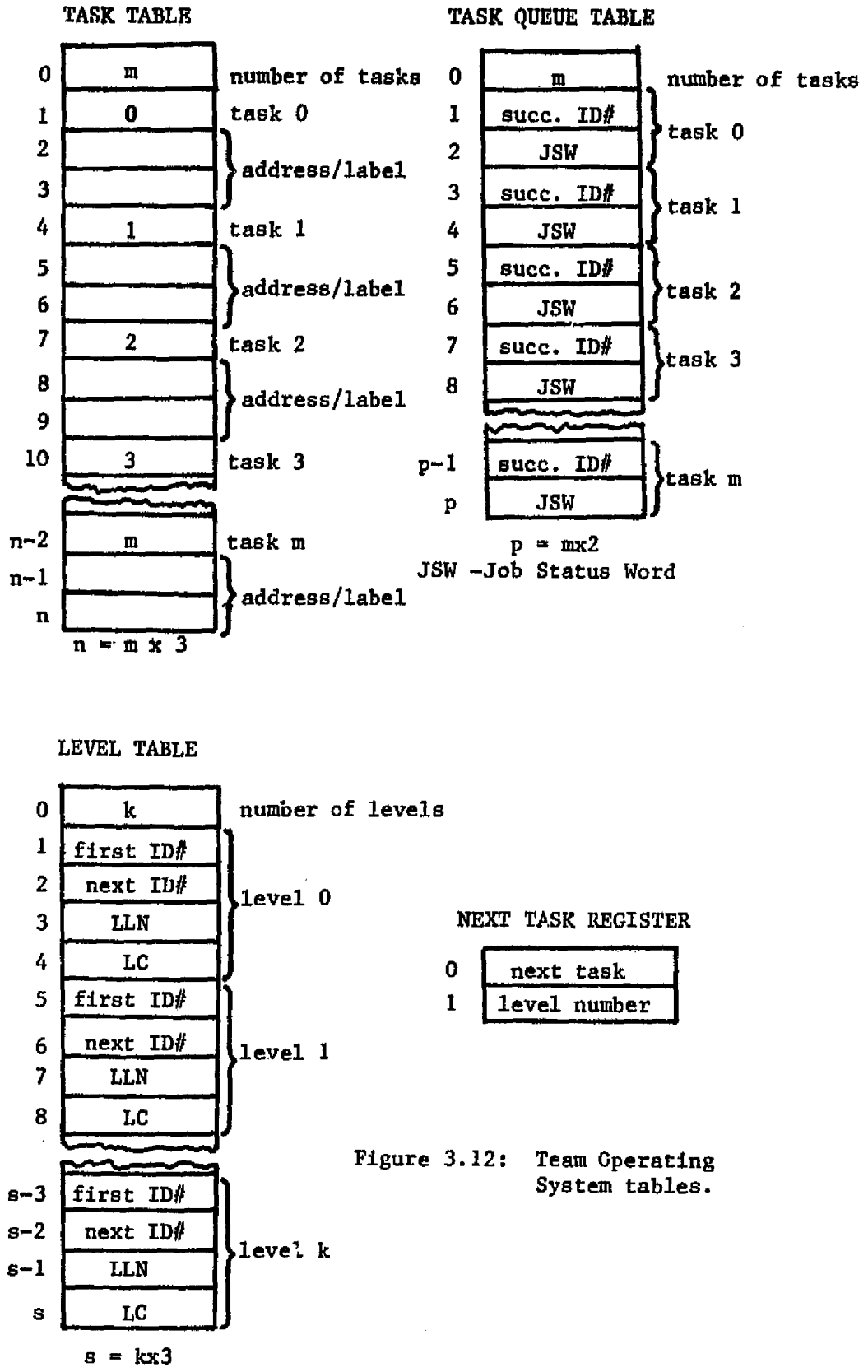


Figure 3.12: Team Operating System tables.

The CPS must then get the program from the system mass memory.

The TQTBL requires two bytes per entry. Tasks are linked to an entry in the TQTBL by the ID number as defined by their TTBL entry position. Each TQTBL entry contains the successor ID number in the first byte, and a job status word (JSW) in the second byte. The successor ID points to the job that is to be run after the current task. Valid ID numbers are 00 to 7F, with FF in the successor ID entry signifying that the current task is the end of the chain.

The Job Status Word (JSW) contains bit flags used by the TOS. For this study, only the least significant bit will be used. This is the Operational Status Bit (OSB). When the OSB is '1' the job is to be executed when its turn comes up. When the OSB is '0' the job is skipped. Thus, the OSB is used to dynamically modify the jobs in the system.

The third TOS table is the LTBL. Each entry in the LTBL is a level containing jobs which have to be run at approximately the same frequency. The lowest level contains jobs that must be run at the highest frequency. Tasks are linked in a chain by levels in the TQTBL.

Four bytes of memory are required by each LTBL entry. The first byte contains the task number of the first job to be executed in the level. The next task to be executed in the level is stored in the second byte. The level limit number (LLN) is stored in the third byte and the level counter is stored in the fourth. Tasks are linked together in the TQTBL by levels, the first task entry in the LTBL contains the ID number of the first task in a chain of tasks in the

TQTBL.

Each level is executed LLN times. The level counter (LC) is used to count how many times a level has been cycled through. When a level has been executed LLN times, the TOS goes to the next higher level and executes a job from this level.

Finally, the Next Task Register (NTR) is two bytes of memory which contains the ID number of the next task to be executed in the first byte and the level of this task in the second byte.

Figure 3.13 is a flowchart of the algorithm used in the TOS. Each CPS performs this algorithm every time it finishes an IAS task. To illustrate the algorithm, Figure 3.14 constructs an example of a hypothetical system, and shows the order in which the tasks are run. The purpose of the TOS tables is to create a crude order of magnitude separation between the frequency of execution of tasks in one level and the next.

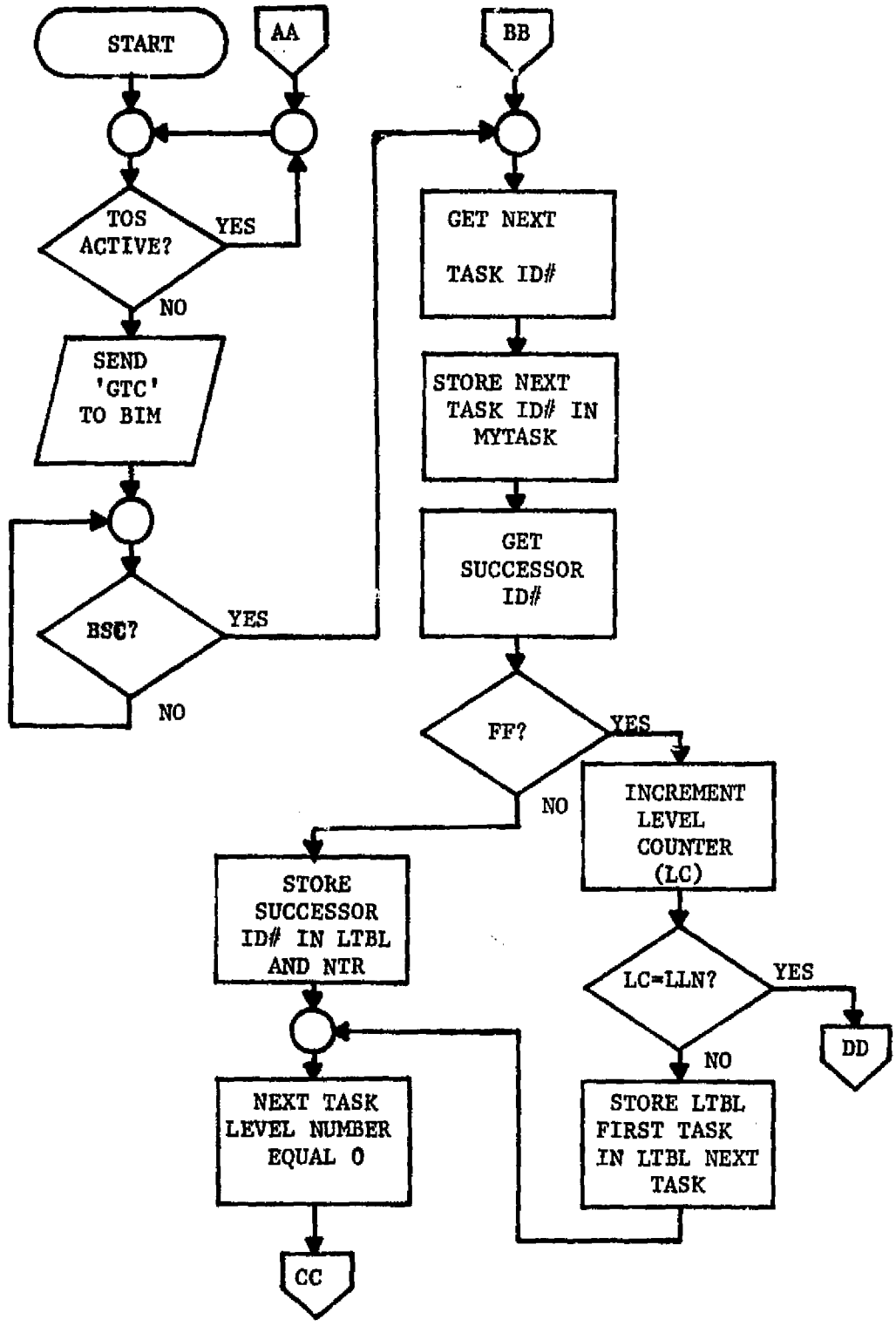


Figure 3.13: Team Operating System algorithm flowchart.

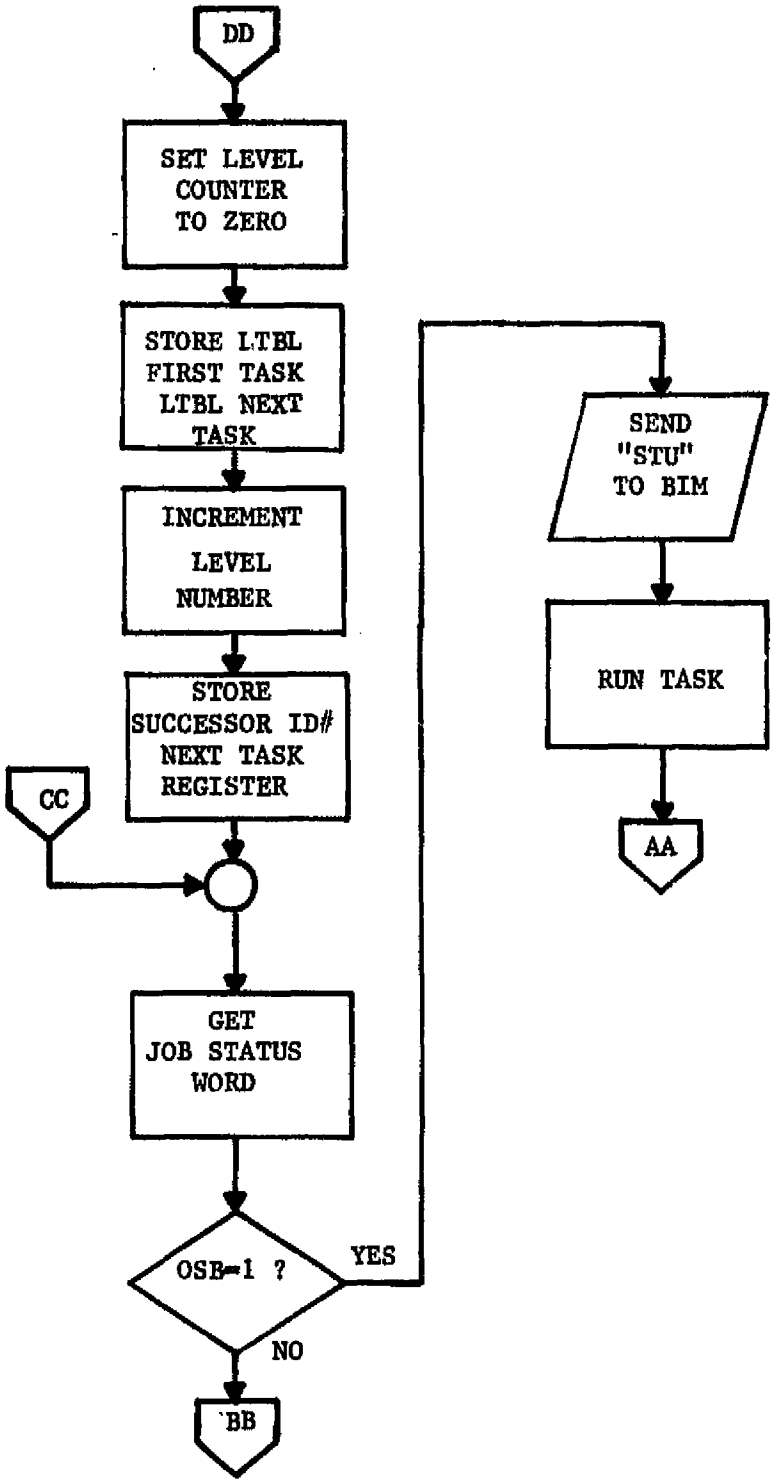


Figure 3.13: Team Operating system algorithm flowchart (continued).

System Configuration:

Number of levels: 3.

Level Number	0	1	2
Task ID Numbers	0, 1, 2	5, 6	3, 4
Execution Order	1, 0, 2	5, 6	3, 4
Level Limit Number	2	1	1

Initial System Status: Level 0; Task 1.

Execution Sequence:

Sequence	Task ID	Level	Sequence	Task ID	Level
1	1	0	16	1	0
2	0	0	17	0	0
3	2	0	18	2	0
4	1	0	19	1	0
5	1	0	20	0	0
6	2	0	21	2	0
7	5	1	22	5	1
8	1	0	23	1	0
9	0	0	24	0	0
10	2	0	25	2	0
11	1	0	26	1	0
12	0	0	27	0	0
13	2	0	28	2	0
14	6	1	29	6	1
15	4	2	30	4	2

Figure 3.14: TOS Example.

4 SIMULATION MODEL

The IAS is a complex system, having many subsystems with an infinite number of interactions possible between them. The lack of a strict definition of the system configuration and interactions could make the simulation model too complex to be studied effectively. In an effort to reduce the complexity, the problem was defined and simplified as the model was developed.

4.1 SIMULATION OBJECTIVES

Basically, the simulation was a study of the Team Architecture concept. This architecture is not restricted to avionic systems alone, it could be used in any application requiring high reliability. Since the purpose was to study the architecture and not spend months writing programs, actual IAS programs were not written for this simulation.

To write IAS programs would required a major programming effort. In addition, models for each of the subsystems would be needed, requiring even more development time. However, as much information is gained by writing programs which take less development time, but represent a wide range of execution times and I/O requirements. The objective is to determine what has the greatest affect on the efficiency.

Instead of writing a simulation program that could only be used for the IAS alone, a multi-purpose simulator was developed. At the

onset it was thought that a simulator might be used for hardware and software evaluation for a prototype system. To facilitate model changes, the program was written so that the models are constructed from inputs. The program allows any of the model parameters to be specified by inputs. Not only does this concept result in a complex model initialization procedure, but model verification becomes more critical also. The resulting program is more complex than a single-use simulator, but also very versatile.

Any Motorola M6800 microprocessor system may be simulated on this simulator without changing any program code. For specialized devices which are connected to the I/O ports of the M6800s a subroutine must be written to handle the device model. This follows the concept used in GASP-PL/I. The basic framework is provided, but the user must write any specialized models to complete the model.

4.2 MODEL DESCRIPTION

The simulation model is a deterministic model of the logical operation of the Motorola M6800 microprocessor and the Bus Interface Module (BIM). Using standard M6800 components in standard configurations eliminates the need for a wire-by-wire simulation. The actions and responses of all the components used in the system are fixed. These were defined by the manufacturer and cannot be changed.*

---

*This also includes the Bus Interface Module, which was defined for this study in section 3.2.

The M6800 data and address busses are simulated, since their functions and operations are defined by the microprocessor.

State changes within the microprocessor, in the memory, and in the peripheral interfaces all occur at time intervals specified by a clock signal. Any external inputs or changes are sensed only during transitions of the clock. M6800 instructions take a finite number of clock cycles to execute. The entire system is digital, and can be simulated using a discrete model.

GASP-PL/I uses an event-scheduling method for discrete simulations. State changes within the model are scheduled as events. The system is modeled by stepping through time from event to event, updating model parameters at event times.[12]

At first thought, the microprocessor would seem to be a very complex system to model. Actually, because of the very strict definition of the components, the model is relatively simple. The basic system has three components: the microprocessor unit (MPU); the memory; and the peripheral interfaces. The M6800 uses memory-mapped I/O and the peripheral interfaces could be considered as part of the memory. However, because of their functional complexity, they are treated separately.

MPU busses do not have to be simulated, the signals sent over these are defined by the components. Bus timing is handled by the MPU, and is fixed. Only four input control lines need be modeled: $\overline{\text{RESET}}$, $\overline{\text{HALT}}$, $\overline{\text{IRQ}}$, and $\overline{\text{NMI}}$. These are negative true input lines to the MPU which cause a reset, MPU halt, maskable interrupt, and non-maskable interrupt, respectively.[3]

The MPU is actually a sequential logic circuit which goes through state changes defined by the current state and the next instruction to be performed. The state at a specific time is a function of the contents of the int. nal registers. The model is then reduced to the MPU registers and control lines.

Six registers are internal to the M6800 MPU: program counter, index register, stack pointer, two accumulators, and a condition code register. The execution of an instruction alters the contents of one or more of the registers, and possibly a memory location. The four control lines mentioned above also cause state changes within the MPU.[3]

Entities within the MPU model are the six registers and the four control lines (Figure 4.1). Attributes which describe these entities are:

1. The contents of the registers or the current status of the control lines.
2. The instruction set of the MPU including both the operation performed and the number of cycles reguired to execute each individual instruction.

The M6800 is an eight-bit microprocessor and can address up to 65K bytes of memory. Memory is modeled as a block of locations addressed exactly as they would be in a normal system. Two types of memory are modeled: random access memory (RAM), and read-only memory (ROM). These can be configured in any manner in the model. Write operations can only be performed on RAM, while a read operation is legal for all types. If the MPU tries to write on ROM, the data will be lost.

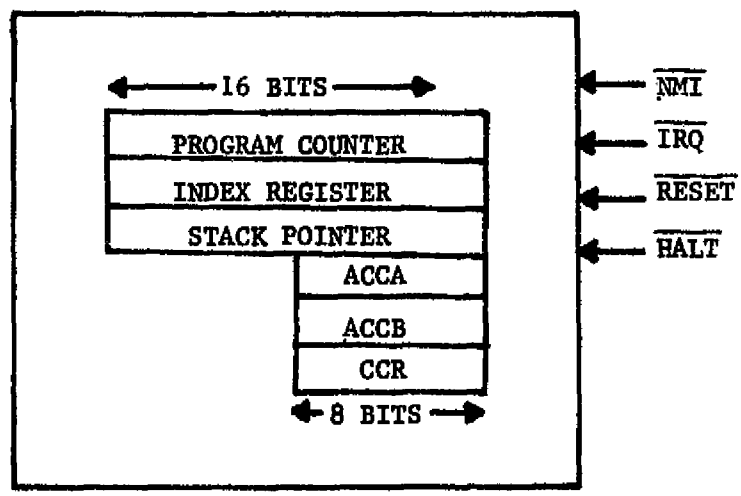


Figure 4.1: M6800 MPU model.

Although other standard devices are now available, only two are modeled: A parallel device, the M6821 Peripheral Interface Adapter (PIA); and, a serial device, the M6850 Asynchronous Communications Interface Adapter (ACIA). Both are programmable LSI components which perform the functions necessary to interface the MPU to many external systems.

Like the MPU, these devices can also be modeled as registers and control lines. The functions performed by the device, depend on the type, the device to which the interface is connected, and the way connections are made. Responses to inputs vary depending on the way the devices are programmed.

The PIA is used to interface the MPU to external devices requiring parallel I/O. Two eight-bit data busses are used, one from each side of the PIA. Lines may be programmed individually as either inputs or outputs. In addition, four control lines, two for each side, are used to handshake with external devices.[3]

Six registers are used, two control registers (CR), two data direction registers (DDR), and two peripheral data registers (PDR). Each side of the PIA is configured alike. The CR is used to control interrupts and handshakes. The DDR is used to specify whether a data line is used as an input or an output. Each line can be individually programmed. All input and output passes through the PDR.[3]

The functional characteristics are described in detail in reference 3. Both the A-side and B-side function alike except that the B-side will only handshake during outputs, and the A-side will only handshake during inputs. Each CR determines how each side

responds to control line changes, and what types of handshakes are used. PIAs are programmed by the MPU to perform functions as required. They must be initialized before they are used, but the configuration can be changed by reprogramming at any time.

Serial communications are performed by the ACIAs. This device is set up to communicate with a modem, using request-to-send, clear-to-send, and data-carrier-detect lines to control the transfers. Like the PIA, the ACIA is a programmable LSI module.

ACIAs contain four internal registers: transmit data register (TDR), receiver data register (RDR), status register (SR), and a control register (CR). To the MPU, the ACIA looks like two memory locations. The CR and SR share the first location. A read operation on the first location will read the SR, which is read only. A write will access the CR, which is write only. Outputs are transmitted through the TDR, while inputs are loaded into the RDR. These share the second location and are write only and read only, respectively.

Both the PIA and ACIA are modeled as registers and control lines (Figures 4.2 and 4.3). The functions performed by these devices are modeled by the programs which perform memory read and write operations in the simulation. Control line state changes and data transfers are modeled by a program which handles all of the I/O functions.

Three main categories of events can occur in the MPU system:

1. Instruction execution.
2. Program interrupts.
3. Input or output to or from a peripheral device.

Every MPU in the system uses the same instructions, interrupt

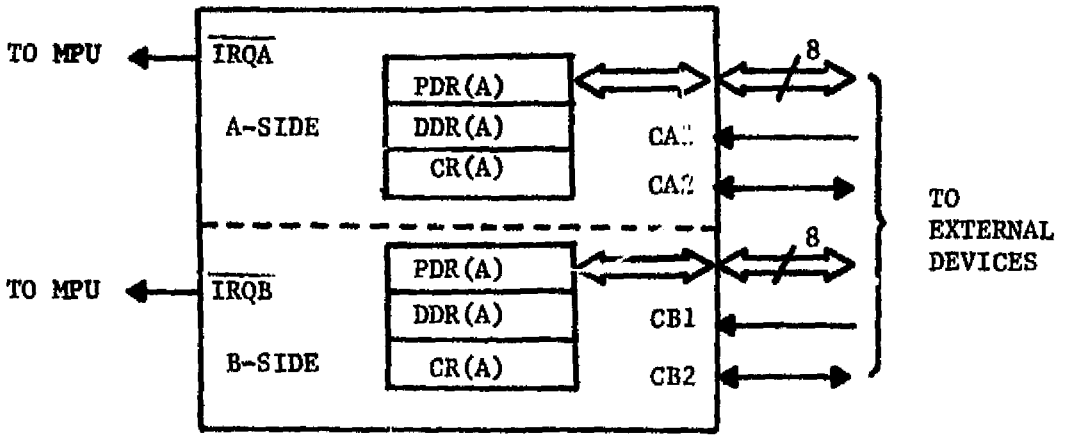


Figure 4.2: Peripheral Interface Adapter (PIA) model.

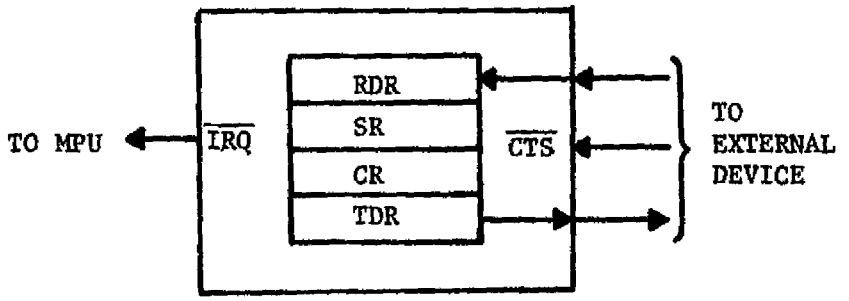


Figure 4.3: Asynchronous Communications Interface Adapter (ACIA) model.

sequences, and I/O functions.

Instruction events work within the MPU registers, the memory, and the peripheral interface registers. The memory contains the program to be run, and the MPU program counter is incremented from instruction to instruction. Seventy-two different instructions, each using up to six different addressing modes can be executed. Instructions may use up to three memory locations. Execution times vary from two cycles to twelve cycles depending on the addressing mode and operation performed. The logic sequence followed by the MPU during instruction execution is shown in Figure 4.4.[3] The model uses the flowchart shown in Figure 4.5 for instruction events.

Unlike the MPU, which must be started by a reset, the simulation model may be started at a specified address. This allows simulation runs to be made on sections of programs without doing the entire program. The two entry points in Figure 4.5 show this.

The M6800 instruction set is very versatile. Immediate, direct, indexed, extended, relative and inherent addressing modes are all available. Subroutine calls are also implemented with the stack pointer. Out of the 256 possible instruction codes, 192 are used.[3]

Four types of program interrupts are possible: halt, reset, maskable interrupts, and non-maskable interrupts. These are caused by a logic zero being present on the $\overline{\text{HALT}}$, $\overline{\text{RESET}}$, $\overline{\text{IRQ}}$, and $\overline{\text{NMI}}$ input lines, respectively.[3] The MPU only executes interrupts at the end of each instruction cycle, and the simulation model checks for interrupts after the instruction has been performed. MPU interrupts are executed according to the flowchart in Figure 4.6. Note that in the simulator

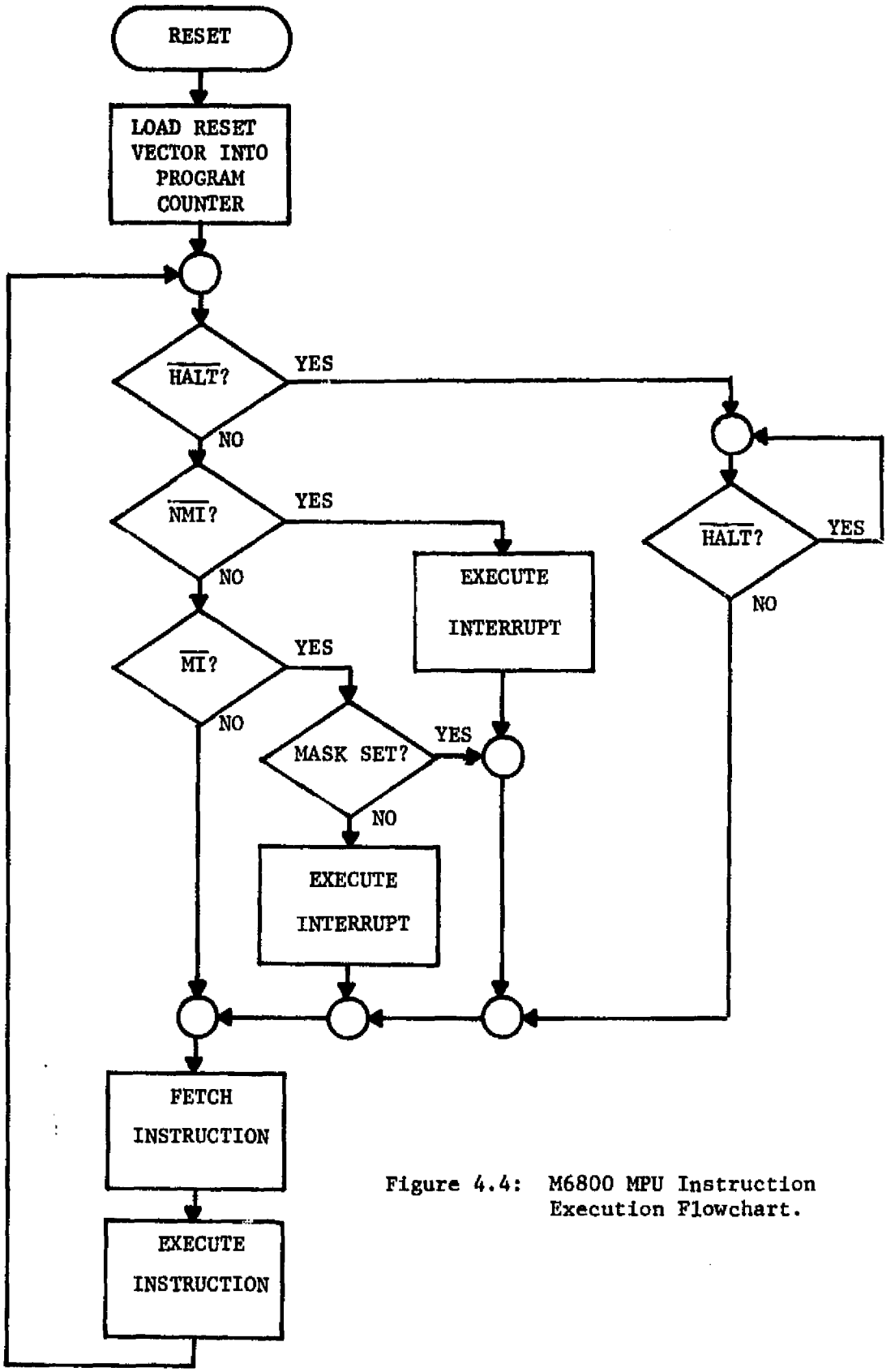


Figure 4.4: M6800 MPU Instruction Execution Flowchart.

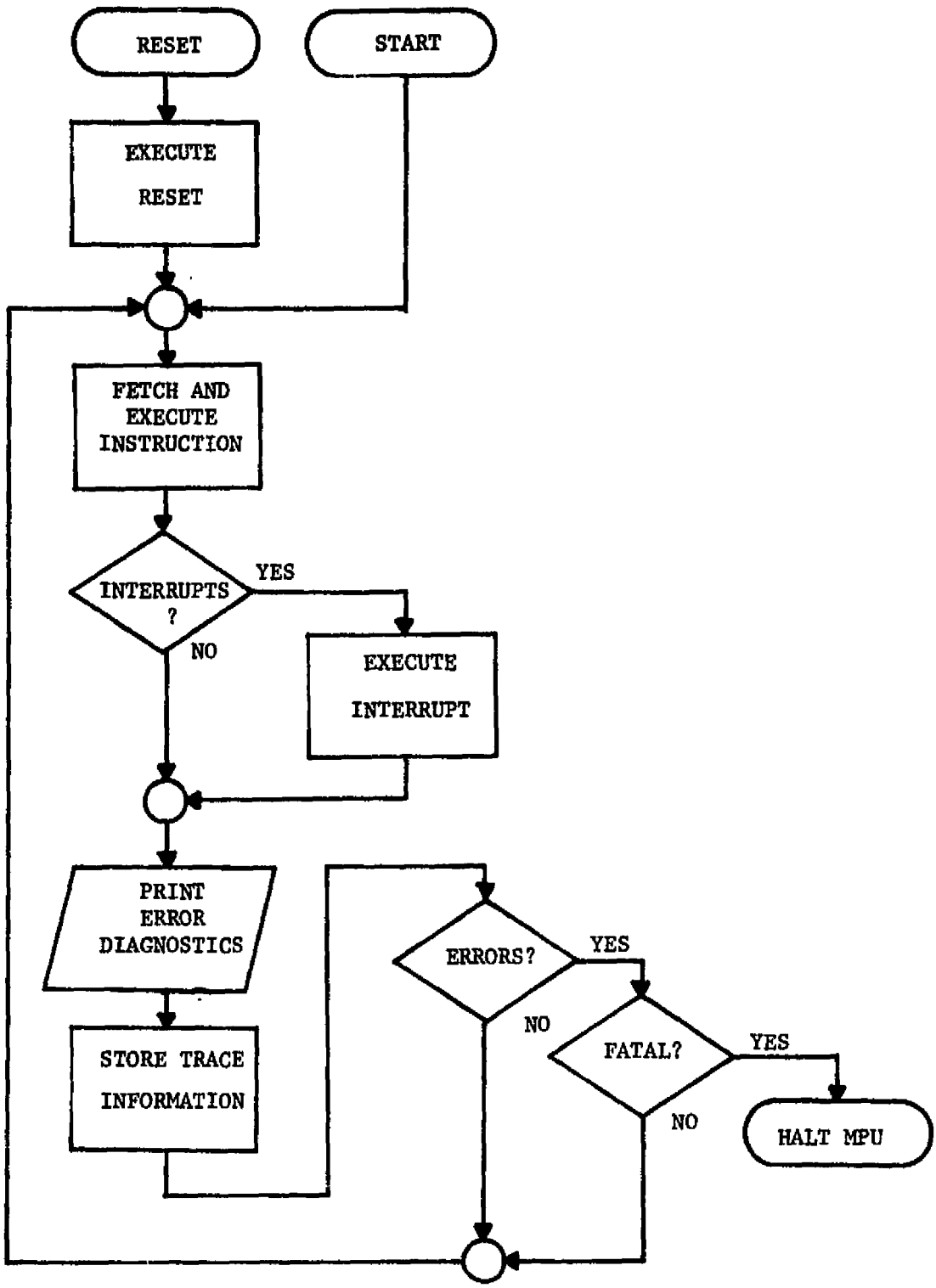


Figure 4.5: Simulation Model Instruction Event Execution Flowchart.

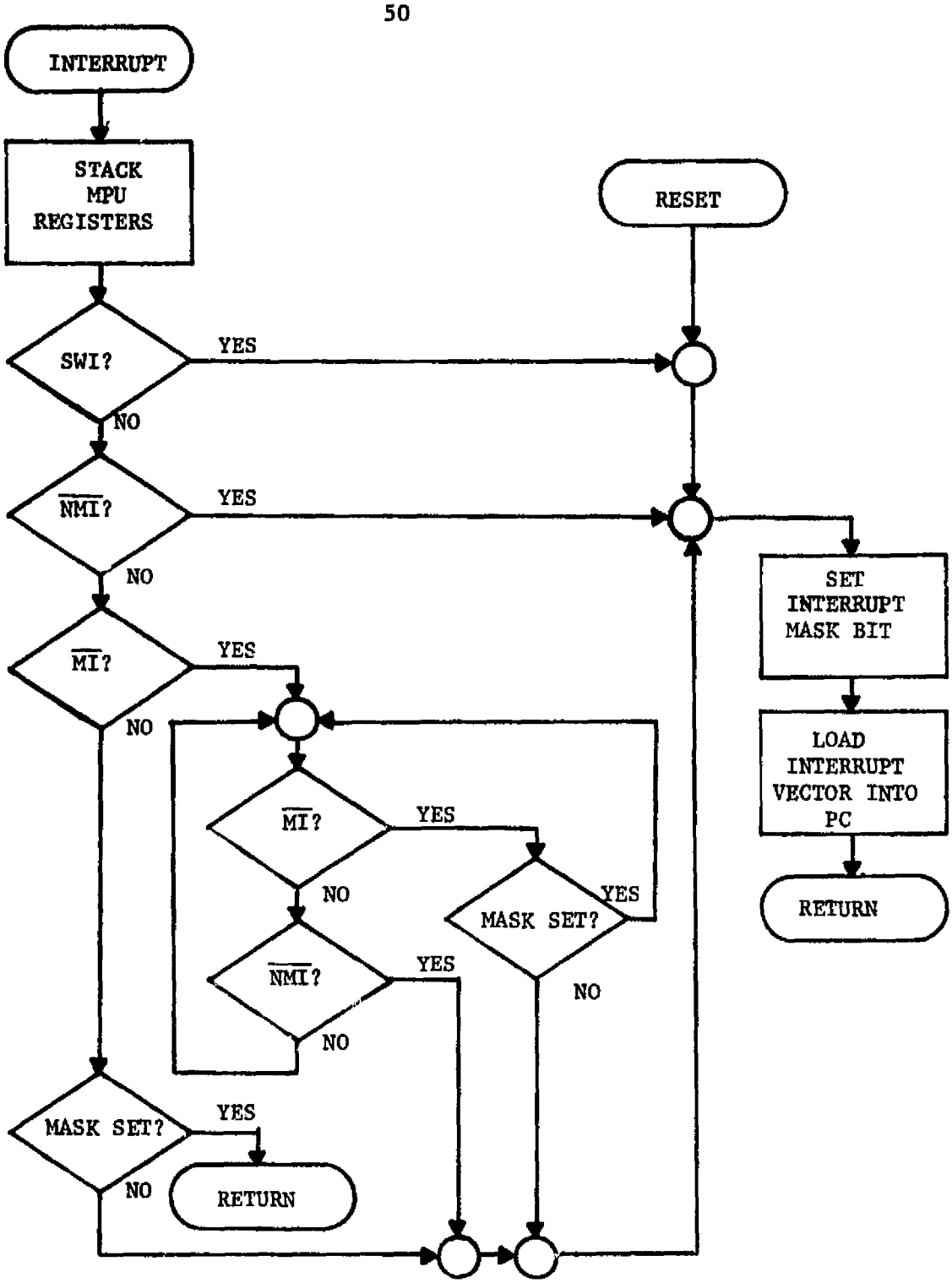


Figure 4.6: MPU Interrupt Execution Flowchart.

interrupts are detected during instruction events. However, any of the four interrupts may be scheduled at any time.

When a $\overline{\text{HALT}}$ signal is true (logic 0), the MPU will complete the current instruction and then halt processing. The halt state is an idle state in which the MPU is waiting for the $\overline{\text{HALT}}$ line to go high again. When this occurs, processing will resume. A halt event is executed in the model by setting a flag specifying that the MPU is in the halt state.[3]

MPU resets are normally used to start the MPU at a specific location after power-up. The starting address is stored in locations FFFE and FFFF of the memory. A reset cannot occur until after eight clock cycles have passed since the power came up on the MPU. Three clock cycles are required to complete the reset, after which program execution continues as usual.[3]

Once the $\overline{\text{RESET}}$ line is pulled to a logic zero, the following sequence of events takes place (Figure 4.6). The MPU fetches the interrupt vector from addresses FFFE and FFFF, and loads it into the program counter. The interrupt mask bit is set, then on the next cycle the instruction at the address in the program counter is fetched and program execution begins.[3] In the simulation model, resets are performed in the same manner and can be used as starting events for model execution.

Maskable and non-maskable interrupts both cause the same sequence of events to occur within the MPU. When an interrupt is detected, the contents of the MPU registers are pushed onto the stack, and the interrupt vector is loaded into the program counter. The interrupt

mask bit is set in the condition code register and program execution resumes at the address just loaded into the program counter.[3]

Maskable interrupts are caused by the $\overline{IRQ}$ line going low. These will not be executed when the mask bit, "I," of the condition code register is set. The $\overline{IRQ}$ line is connected in a wired-OR configuration to the interrupt request outputs of any peripheral devices. Thus when more than one device requests an interrupt, the $\overline{IRQ}$ line will remain low until all of the interrupts have been cleared. Maskable interrupts use the locations FFF8 and FFF9 for the interrupt vector.[3]

Interrupts are not prioritized on the M6800 unless the user designs a prioritizing circuit. The simulation model does not implement any type of prioritizing circuit, however, this could be done rather easily. In the model, when a peripheral device causes an interrupt, an interrupt counter is incremented. When the interrupt is subsequently cleared, the interrupt counter is decremented. Interrupts will not be lost, even when the interrupt mask is set.

Non-maskable interrupts are executed and modeled in the same manner as maskable interrupts. These interrupts cannot be masked, however, and are caused by the $\overline{NMI}$ line going to a logic zero. Non-maskable interrupts use the locations FFFC and FFFD as an interrupt vector.[3]

The simulation model executes both types of hardware interrupts in the same way. It is assumed that any PIAs or ACIAs in the system cause maskable interrupts only. The capability was not included to specify the type of interrupt.

Two types of software interrupts are possible. The SWI (software interrupt) instruction uses the interrupt vector at locations FFFA and FFFB. Execution of the SWI instruction requires 12 clock cycles to perform the same sequence as the hardware interrupts. The WAI (wait for interrupt) instruction pushes the MPU registers onto the stack and then goes into an idle loop until a hardware interrupt occurs. The WAI is used to provide faster response to hardware interrupts when they occur.[3]

The third event category, called I/O events, is the most complex type to model. although all BIM functions are modeled as I/O events, there are only two types of I/O events possible for the MPU system: data transfers and control line changes. Both must be handled differently for PIAs and ACIAs. BIM functions modeled as I/O events will be discussed in section 4.3.

To simplify the program requirements for the model, I/O events are only scheduled for the device on which the state change is to occur. All state changes are made, and then any further changes on other devices are scheduled as I/O events. This can be thought of as outputting a signal (the state change) to another device.

For PIAs, five I/O events are possible:

1. Data transfers.
2. CA1(CB1) set.
3. CA1(CB1) reset.
4. CA2(CB2) set.
5. CA2(CB2) reset.

For both sides of the PIA, the execution of each event is the same. When CA2 and CB2 are used as outputs, the two sides do not perform alike. The A-side will only perform a handshake during a read operation, while the B-side will only perform the handshake when used as an output.

The peripheral data registers (PDR) are connected to the data lines that interface the PIA to the external device. When the data byte on the lines changes, the value in the PDR changes. This does not cause an interrupt or change within the control register. For the PIA, the data transfer event causes a data byte to be loaded into the peripheral data register.

The control line set/reset events can result in interrupts, other control line changes, and bits in the CR being set. The contents of the CR determines what happens when a control line is set or reset. Thus, the program must perform the event, then check to see what mode the PIA has been programmed in and perform any further changes.

Two event types are possible for ACIAs: data transfers, and a $\overline{\text{CTS}}$ (clear to send) set/reset. Data transfers cause a data byte to be loaded into the receiver data register if it is empty. If it is not, the data byte is not loaded into the RDR. If a second data-transfer occurs before the data is read by the MPU, an overrun occurs and the first data byte is lost.

The $\overline{\text{CTS}}$ is an input line from the modem which will inhibit transmit interrupts. When the $\overline{\text{CTS}}$ is high, the transmit data register empty (TDRE) bit of the status register will be forced low, and will stay low until the $\overline{\text{CTS}}$ signal goes low.[3]

Unless the user writes external models to interface with the simulator, all I/O events are transparent. MPU trace prints will not contain I/O events, however, they will show any interrupts that result.

4.3 BIM MODEL

For this study, the Bus Interface Module (BIM) was treated as a black box. Although the BIM, as described in section 3.2, is specified to be a M6800 microprocessor and a hardware interface unit, it was only simulated functionally. It could have been simulated as another MPU system, with the interface unit being external, but this is more detail than required. BIM functions are limited by the specifications in section 3.2. Response times are a function of how many bytes of data have to be transferred and the current system bus activity. It is simpler to write a PL/I program which works within the simulator, computes response times, manages the system bus, transfers data, and schedules all responses than to write the assembly language programs for the BIM.

In the model, all response times for the BIM are calculated according to the type of operation, how much interaction with the system bus has to be performed, and how much activity is occurring within the BIM. Estimations are based on the I/O processor having a clock period of one microsecond, and the interface unit always executing its state changes before the I/O processor does. This is a reasonable assumption, considering that the I/O processor must execute

a number of instructions to change states. With an average execution time of about 3.5 microseconds, it will stay far behind the interface unit.

Since the IEEE 488 bus standard specifies that all I/O on the bus is performed using handshakes, the data transfers across the bus will only occur as fast as the slowest device (that is participating in the handshake) can operate. Thus, data rates are not only affected by the talker's capability, but the listeners' also.[2]

BIM timing estimations are based on the amount of time it would take to execute the function required. The following functions were analyzed to estimate BIM response times:

1. Source handshake timing in which the I/O processor is sending a data buffer.
2. Acceptor handshake timing in which the I/O processor is receiving a data buffer.
3. Getting control of one of the system busses.
4. Performing local BIM functions that do not require system bus interaction.

The handshake timing is the most important for the accuracy of the model. A small error when summed over a large data buffer will be magnified greatly. This applies to both inputs and outputs.

Consider the source handshake, which could occur in the following situations:

case 1: The BIM is sending a buffer and is sequentially stepping through by incrementing addresses.

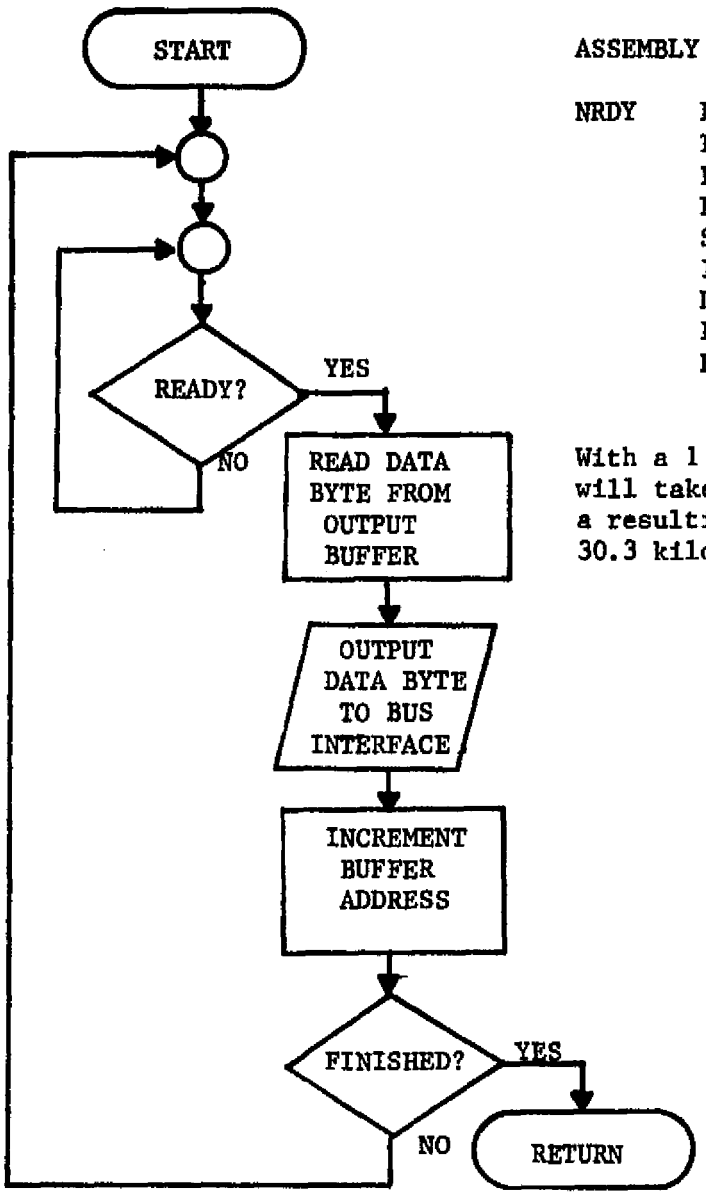
case 2: The BIM is responding to a serial poll when it has been addressed and is sending out only one data byte. This would occur even if the BIM had not sent a service request (SRQ) message on the bus.

case 3: The BIM is currently in control of the bus and is sending messages. For example, when the bus is being set up for a buffer transfer, the BIM will be sending listen addresses across the bus.

Figure 4.7 shows the program steps that would have to be executed to cause the transfers (case 1). Assuming that the interface unit is ready before the I/O processor (it has 15 microseconds to do so) the cycle takes 30 micro-seconds. This is a data rate of 30.3 kilo-bytes per second. Each time the interface unit is not ready, the execution time increases by ten microseconds. If it was not read on the first check continually, the data rate would drop to 25 kilo-bytes per second. This is a drop of 17.5 %. The maximum data rate across a bus is limited to 30.3 kilo-bytes per second by the I/O processor.

Figure 4.8 is the sequence that would occur for case 2. Note that this case is initiated by an interrupt, since all BIM inputs are handled using interrupts. At the minimum, using a hardware prioritized interrupt configuration the I/O processor would respond to the interrupt 14 microseconds after the current instruction was completed.

It still must decode the message and perform one source handshake cycle (case 1) before returning to its original task. The decode



ASSEMBLY CODE			CYCLES
NRDY	LDAA	PIACRB	4
	BITA	#$80	2
	BEQ	NRDY	4
	LDAA	$0,X	5
	STAA	PIAORB	5
	INX		4
	DECB		2
	BGT	NRDY	4
	RTS		30 total

With a 1 MHz. clock, the loop will take 30 microseconds, with a resulting data rate of 30.3 kilobytes/second.

Figure 4.7: Case 1, Output Buffer I/O Loop.

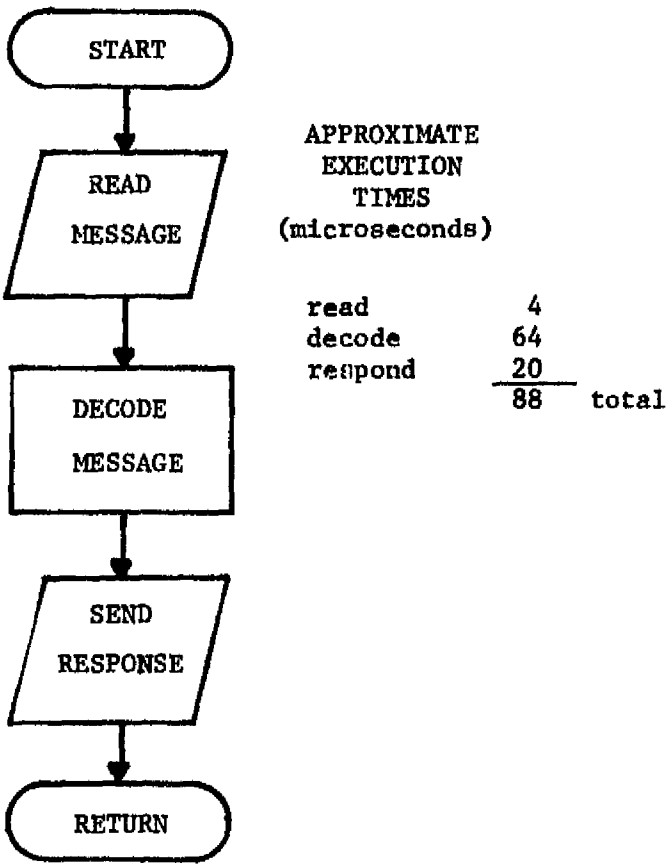


Figure 4.8: Case 2 Timing Sequence.

would reguire a skip chain. Since there are 12 local messages defined, by putting the most likely messages first in the skip chain, the command would on the average, be decoded before half of the messages were checked. Ten microseconds are required for each check, plus the jump requires another four microseconds. The total time is 88 microseconds after the interrupt has occurred. Including the interrupt, the response time will be at least 102 microseconds.

Case 3 can be assumed to be essentially the same response time as case 1. The I/O processor will be reading listener addresses from the memory and sending them out to the bus.

The acceptor handshake can occur in any of the following cases:

Case 4: The BIM has been addressed as a listener and is inputting data bytes sequentially into an input buffer.

Case 5: The BIM is inputting one byte from the interface. The message is expected, and the BIM is in a wait loop.

Figure 4.9 illustrates the steps involved for case 4. If this loop is performed sequentially without interrupts, it would require 29 microseconds per cycle. Add the 14 microseconds for the interrupt and the response time is 45 microseconds. This is an input data rate of 22 kilo-bytes per second.

Case 5 represents a situation where the BIM has received a message from the interface and another message is to follow immediately (Figure 4.10). The minimum time before the message is read is 14 microseconds.

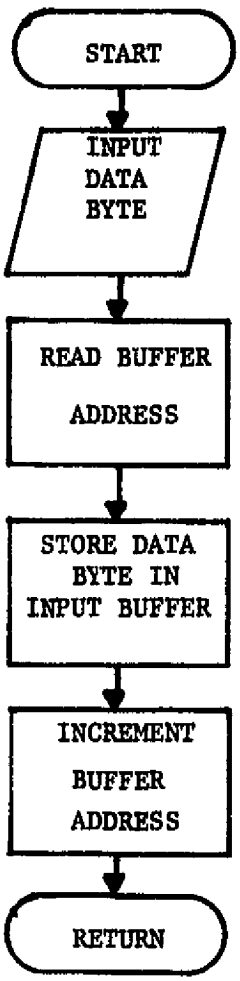
From the analyses of cases 1 and 4, when the BIM is sending a buffer to another device, the maximum data rate possible is 30.3 kilo-bytes per second, worst case is 22 kilo-bytes per second. Case 5

will only occur in special operations.

The BIM will operate in the range from 22 to 30.3 kilo-bytes per second. Considering that the IEEE 488 bus is a one mega-byte per second bus, data rates across the bus will be very slow using this implementation. Table 4.1 lists the five cases and the timing estimations. These are the data rates used in the BIM model.

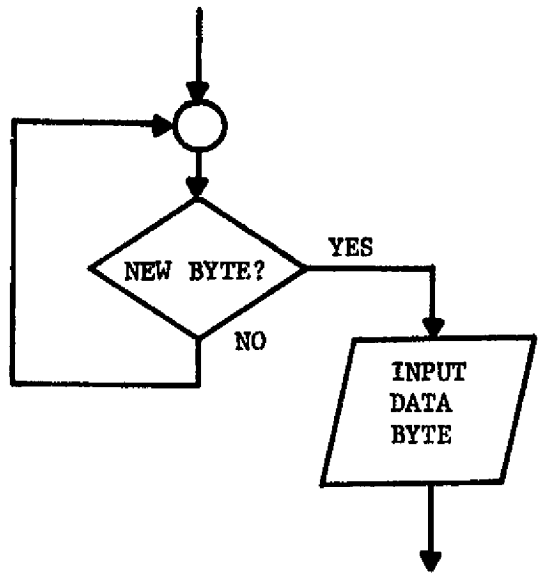
Table 4.1: Summary of Bim timing estimations.

CASE	DATA RATE	OPERATION	MINIMUM RESPONSE TIME
1	$\leq$ 30.3 KB/sec.	output	30 microseconds
2	$\leq$ 11 KB/sec.	output	88 microseconds
3	$\leq$ 30.3 KB/sec.	output	30 microseconds
4	$\leq$ 22 KB/sec.	output	45 microseconds
5	$\leq$ 71 KB/sec.	input	14 microseconds



ASSEMBLY CODE		CYCLES
LDAA	PIAORA	4
LDX	BUFFAD	5
STAA	$00,X	6
INC	BUFFAD	6
RTI		10
		31
interrupt		14
		45 total

Figure 4.9: Case 4 Timing sequence.



ASSEMBLY CODE			CYCLES
LOOP	LDAA	PIACRA	4
	BITA	$80	2
	BEQ	LOOP	4
	LDAA	PIAORA	4
			14 total

Figure 4.10: Case 5 Timing Sequence.

5 SIMULATION PROGRAM

The simulation program is written in PL/I. A simulation package, called GASP-PL/I is used as the framework around which the program was written. Although PL/I is not considered a simulation language, like SIMSCRIPT or GPSS, it is a versatile general scientific programming language. PL/I has computational facilities for floating point arithmetic, bit-string handling, and many built-in functions, allowing any type of computation to be performed. In addition, extensive input/output facilities are available for use including print editing, sequential and direct access file management. PL/I is well suited for simulation programming because of its many capabilities.

GASP-PL/I is a PL/I based simulation package (not a language) derived from GASP-IV (Fortran-based). Discrete, continuous, or combined discrete-continuous simulations can be performed using GASP-PL/I.[1] GASP-PL/I was chosen for this study because previous work on related projects has shown it to be well-suited for digital systems simulations.[1][16]

Throughout the simulation program, structured programming techniques are used. This method of program writing results in a hierarchial program which has a minimum of redundant coding, and is easily debugged. Structured programming follows a top-down method of both development and testing. As a result, by the time the lowermost programs in the hierarchy are tested the entire simulation is running.[6]

The model developed in chapter 4 could be implemented in many

ways. However, a table driven environment was found to be the most efficient. This method uses a group of system tables, whose size depends on the number of MPUs, the number of peripherals, and the size of the memory. Elements in the tables describe the system and control the simulation execution.

5.1 GASP-PL/I

GASP-PL/I is a package of PL/I programs which perform the time functions, data-file management, and support facilities required for a simulation. Each GASP-PL/I program performs a unique function in the simulation, and can be called by any user program.

The simulation problem is divided into two dimensions in a GASP-PL/I simulation: the state-space dimension; and, the time dimension.[12] GASP-PL/I performs the time dimension and the user writes the state-space dimension. The state-space dimension is the behavioral model of the simulated system. Discrete, continuous, or combined discrete-continuous systems can be modelled using GASP-PL/I.[12][13][14]

On closer examination, all of the programs in a GASP-PL/I simulation can be divided into four main functional areas:[14]

1. Executive.
2. Dynamic behavior description.
3. Data storage.
4. Support facilities.

The support facilities are a group of specialized single-function programs which fall into these areas:

1. Initialization of the model.
2. Initialization and run-time error diagnostics.
3. Random deviate generators.
4. Statistical calculations.
5. Event monitoring.
6. Outputs.

Both GASP-PL/I and user-written programs are used in all but the data storage functions. Figure 5.1 illustrates the division of programs by functional areas. The GASP-PL/I programs listed here represent approximately 2,900 lines of code, a fairly large programming effort which the user does not have to do.[14]

Probably the biggest advantage in using GASP-PL/I is that the user is not concerned with the simulation time management, but concentrates exclusively on the construction of the model. PL/I allows the user a high degree of flexibility in writing the simulation model. Combining this with the many support facilities available, shows that GASP-PL/I is a very powerful simulation tool.[12][13][14]

Many of the advantages in using GASP-PL/I are a result of the programming language, PL/I. On the other hand, many of the disadvantages are also a result of using PL/I. PL/I is a very high level language, and thus is very inefficient. Programs requiring many iterations of complex calculations will consume large amounts of CPU time. PL/I requires large amounts of core to run in. This restricts any GASP-PL/I simulations to machines with fairly large main

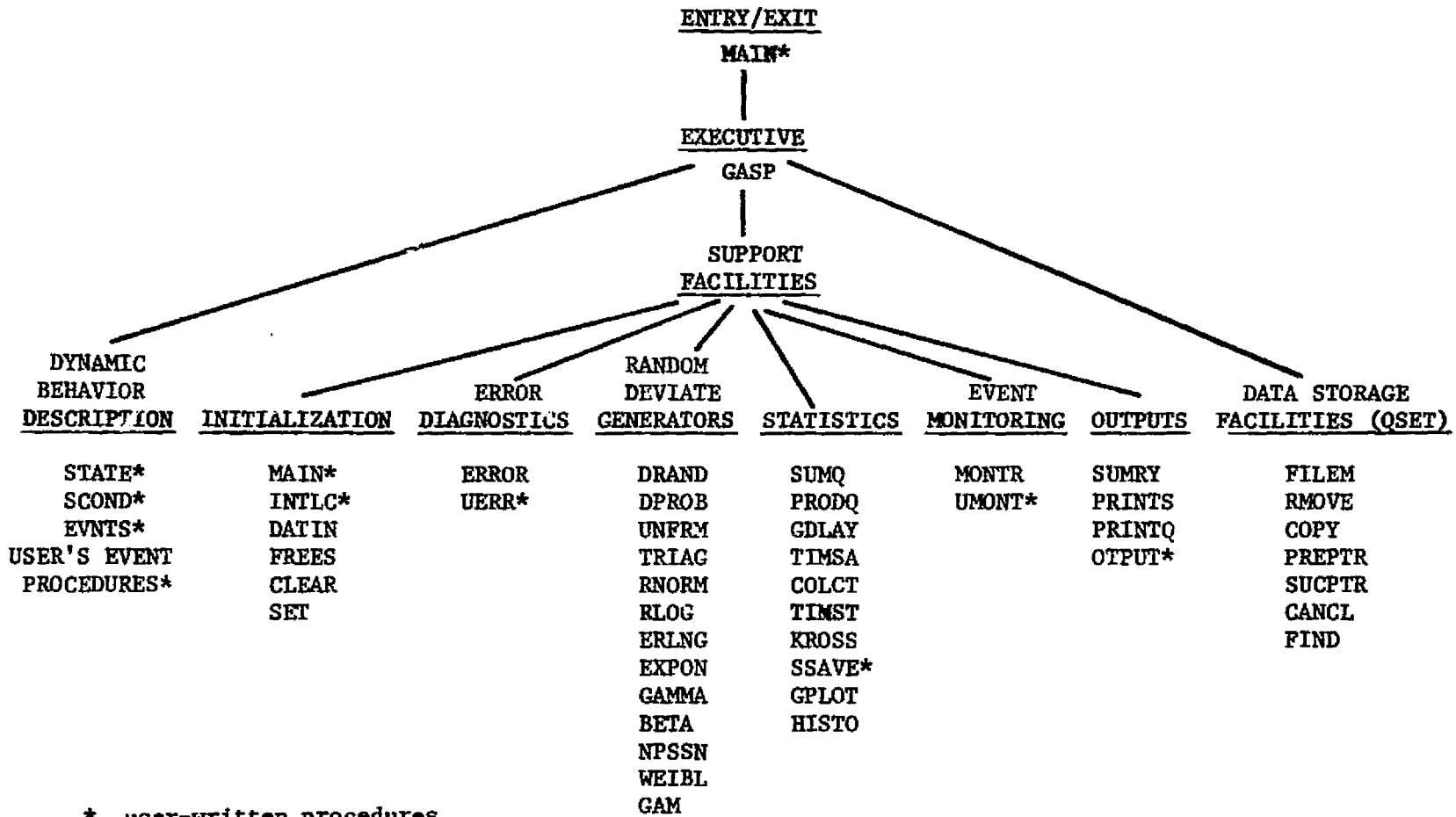


Figure 5.1: The Functional Divisons of a GASP-PL/I Simulation Program.

storage even when using overlaying techniques.

After working with GASP-PL/I for the past year, several deficiencies were noted. All variables used by GASP-PL/I programs are declared external, resulting in over 200 external names from the GASP-PL/I programs alone. As a result, loader table overflows may occur. Round-off errors can be a problem, because GASP-PL/I variables are declared single precision. The GASP-PL/I reference lacked much of the information needed to write programs which interface with GASP-PL/I programs. To overcome this, the actual program listing was used extensively.

5.2 STRUCTURED PROGRAMMING

Structured programming is a technique by which programs can be written that are concise and easily debugged. Sometimes called "go-to-less" programming, structured programming techniques help reduce program redundancy and complexity.[6] The final program product is hierarchial in structure, with all higher-level procedures determining when to call lower-level procedures.

By dividing the program task into different functions, each leg of the hierarchy can perform a different function required in the total task. This makes program errors easier to isolate, and changes easier to make. The resulting programs are smaller in size, and more manageable.

Programs written for this project follow structured programming techniques consistently, except in cases where it was found that a

considerable amount of execution time could be saved otherwise. Time is a major disadvantage of structured programming. Program calls, especially in PL/I with dynamic storage allocation, are time consuming. A compromise was sought between a strict single function, non-redundant program structure, and the less time-consuming, sequential programs.

5.3 MODEL REALIZATION

The models developed in chapter 4 were realized in two parts. The first and most complex was the M6800 model. Much of the complexity results not from the M6800, but because a multiprocessing model was developed. In keeping with the general purpose simulator concept, the model was realized so that many of the parameters are specified by inputs. The BIM model was developed as an external addition to the general purpose model. In the general simulator, the entire system configuration is specified by inputs. Inputs are used to specify the following parameters:

1. The number of MPUs in the system.
2. The memory size and configuration for each MPU in the system. Memory may be specified as any combination of RAM or ROM up to the maximum 65K addressible by the M6800.
3. The number of peripheral units used with each MPU in the system, and the connections used.

Every MPU in the system is a M6800, and thus uses the same instruction set, same types of registers, and functions. Because of this, a table structure was found to be an efficient means of modeling a number of identical MPUs at once. Tables are used for three purposes:

1. To model the registers in the MPUs, PIAs, and ACIAs.
2. To describe the system configuration.
3. To control the execution of the simulator.

Tables are allocated during initialization to have an entry for each MPU in the system.

All of the MPU registers are modeled in the tables. If there are three MPUs in the system, then there are three program counters, three stack pointers, etc. Figure 5.2 shows the format of the tables which model these MPU registers: program counter (PC), stack pointer (SP), accumulator-A (ACCA), accumulator-B (ACCB), the index register (INR), and the condition code register (CCR).

Peripheral devices are modeled using a multi-function structure, PERTBL (Figure 5.3). The PERTBL is allocated to have one entry for each ACIA and two for each PIA in the system. All of the peripheral registers in the system are modeled, plus the interconnections between devices are described in PERTBL.

Two tables are used to describe and control model execution: SYSTBL and TRCE (Figures 5.4 and 5.5). The basic description of each MPU system is contained in SYSTBL plus two elements are used for run-time diagnostics. The TRCE table is used to control trace prints. Four trace options are used: disabled, address range, branch only,

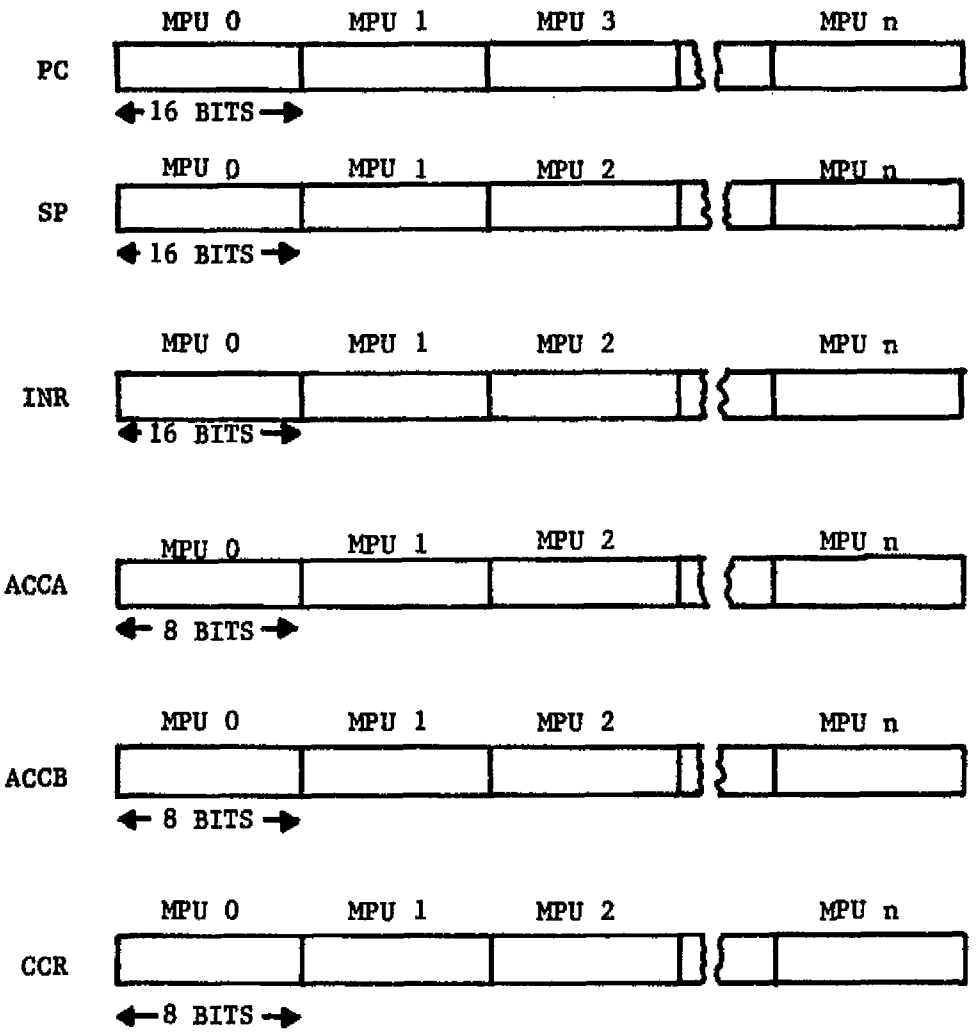


Figure 5.2: MPU Register Tables.

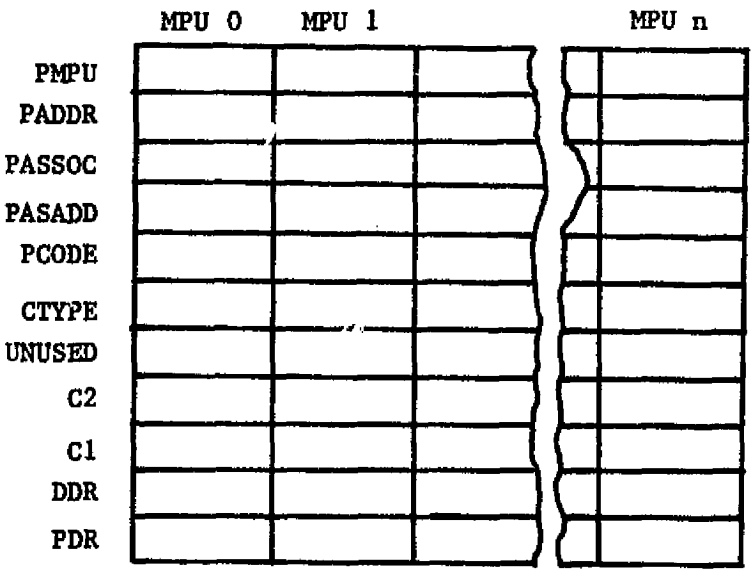


Figure 5.3: PERTBL Structure.

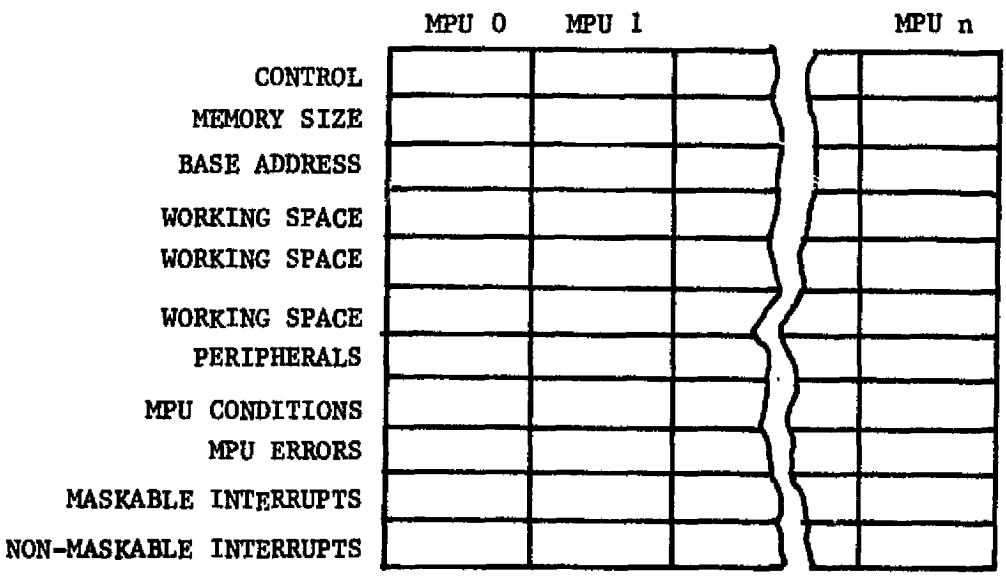


Figure 5.4: SYSTBL Structure.

time, and enabled.

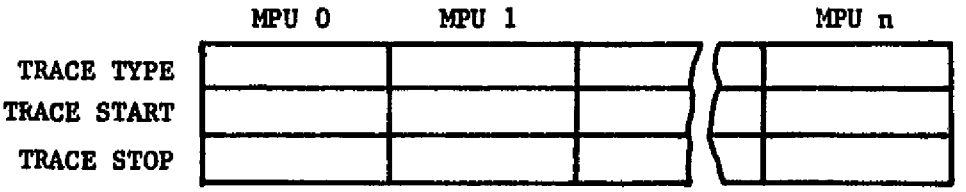
The table structure just described provides an efficient method for initializing and controlling the simulation model. During initialization, each input alters an element in one of the tables. During the execution phase, the programs work within the tables to perform model functions. All of the programs have access to the tables, allowing complete control of the model.

Up to 65K of memory can be specified for each MPU. Four types of memory locations are possible:

1. RAM.
2. ROM.
3. Peripheral device first location.
4. Peripheral device second location.

ACIAs require two locations and PIAs require four, two per side. Two PERTBL entries are required per PIA, one for each side. Because of this, PIA sides are treated separately. However, to be consistent with hardware requirements, PIAs must occupy four contiguous memory locations.

The M6800 is an eight bit machine and thus each memory location must be 8 bits wide. To simulate the different types of memory, another eight bits is used. Peripherals require a link to the PERTBL, and this is stored in the memory location. Simulated memory location formats are shown in Figure 5.6.



Trace type code:

0 -disabled.
1 -address trace.
2 -time trace.
3 -branch trace.
4 -enabled.

Figure 5.5: TRCE Table Structure.

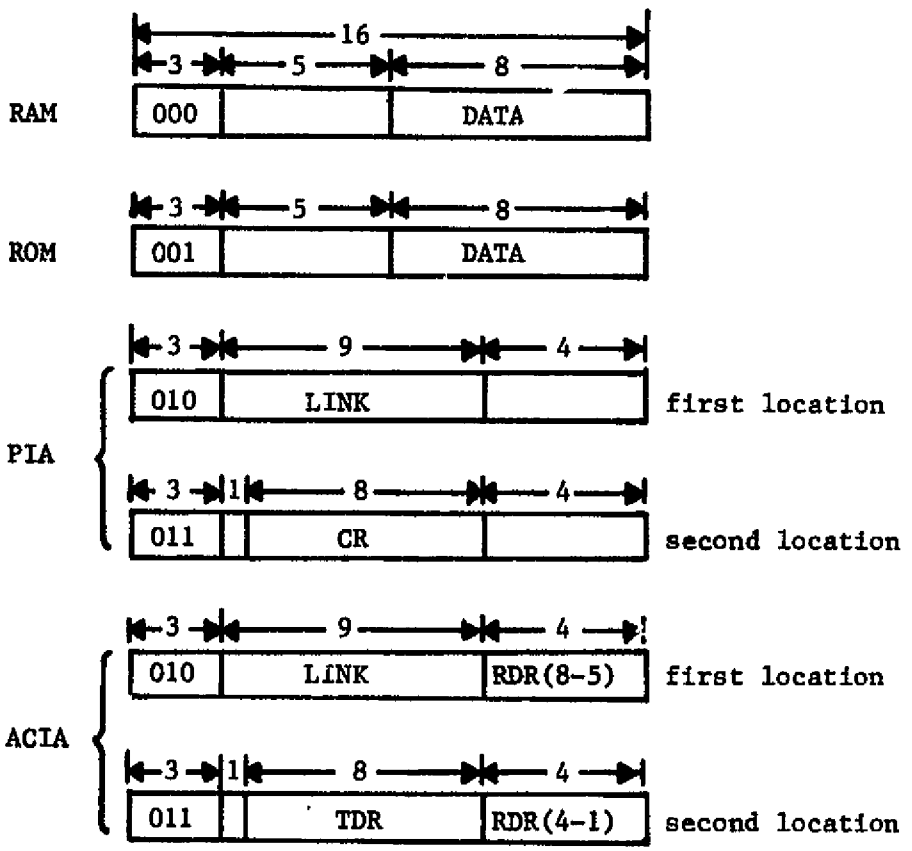


Figure 5.6: Simulated Memory Formats.

The simulator is capable of simulating up to twenty MPUs, all with 65K of memory and 128 total peripheral devices. Since it takes two bytes of memory to simulate one, a quick calculation shows that each MPU requires 2x65,536=131,072 bytes to simulate 65K. With twenty MPUs all having 65K of simulated memory it would require over 2.6 million bytes of memory. This is why two methods of storing the simulated memory are built in to the simulator: in core and in a disk file.

For smaller amounts of memory, using main storage results in a much faster execution time. Disk files are slower, even when using direct access data sets, but larger amounts of memory can be simulated. The method of storage is transparent to all but the initialization and memory access programs in the simulator. A buffer is used to transfer the contents to the simulation programs.

Memory is allocated as one block for the entire system. Each MPU is given a base address which is used to locate all memory locations. This base addressing scheme is used to allow different sizes of memories to be specified for the MPUs in the system.

The simulator automatically assumes the eight highest locations of memory are addressed as FFF8 to FFFF. The MPU uses these addresses as interrupt vectors and they must be present. For example, if 10K (10,240) loocations of memory were specified, the valid addresses are 0000 to 27F7 and FFF8 to FFFF. Note that the simulator also assumes that all memory is contiguous, except for the interrupt vector locations.

When a write operation is performed on ROM, the data will be

lost. When an invalid address is accessed, either an indeterminate number will be read, or the data will be lost if a write is performed. In any case, the simulator flags any invalid memory access operations and prints an error message.

There are three phases to each simulation run: initialization, execution, and the final summary. Because of the format used in this simulator, the initialization phase is the most critical. All model parameters are specified by inputs and the initialization program must check all of the inputs for validity, and flag any errors. Without the confidence that all illegal inputs will be flagged as errors, one can not be sure the simulator is executing a valid model. All of the inputs which specify model parameters are checked twice. The first check is to see if the input is valid according to the type of input expected. A second check is performed to see if the input conflicts with the previously specified inputs.

There are two parts to the initialization of the system. The first part contains the GASP-PL/I initialization inputs, as specified by reference 12. A minimum value is required by the simulation program for some of the GASP-PL/I inputs. These are specified in Appendix 1.

The second part of the initialization is the specification for the M6800 system. This is performed by procedure INTLC, and follows a format similar to GASP-PL/I (see Appendix 1). Eight card types are used, two of which are required. The required cards are:

'SYS' - This card specifies the number of MPUs and the system disposition. It must be the first card to be read after the GASP-PL/I inputs.

'END' - This card signals the end of the input deck. It must be the last card in the deck. If not, any further cards will be ignored.

The rest of the input cards are optional, since default values are assumed if the card is not specified. However, a program must be loaded into memory for the simulated MPU to run. The following is a list of the cards and there uses:

'MPU' - This card contains the specifications for the memory size and number of peripherals for each MPU. One 'MPU' card is used for each mpu in the system.

'START' - This card is used to specify the method of starting an MPU. An address can be specified, or a reset can be used.

'TRC' - Used to specify the trace type for a MPU. If no 'TRC' card is found for an MPU, the default is for trace prints to be enabled.

'MEM' - Used to specify the memory disposition and method of storage for the sytem. If used, this card must physically follow all of the 'MPU' cards and precede any 'LOAD' cards in the input deck, .

'LOAD' - Used to configure and input data into the simulated memory. The 'LOAD' card can be used more than once, but any that are used must follow all 'MPU' cards, and the 'MEM' card if used.

'SIM' - Used to specify simulation control parameters. These are defined by setting bits in a control word.

All cards are free format, starting on any column, inputs may be separated by blanks or commas, and the card may cover any number of physical cards. Inputs to the GASP-PL/I programs are not read by the same procedure, and must start on column one to be read correctly. Appendix 1 contains a complete description of the input formats for this simulator, including the required GASP-PL/I inputs. A sample input deck is shown in Figure 5.7.

```
GEN,'G. A. BRENT',20,,1,'NO','NO','NO',,*
STA,1,,,,*
LIM,0,1,120,3,6,0,0,0,0,*
PRI,2,'FIFO',1,*
COL,1,'ITIME-0',*
SEE,30,*
PRI,3,'FIFO',1,*
INI,'TTFIN','Y','Y','Y',0,0.05,*
ENT,2,10,0,0,0,0,0,0,0,*
ENT,1,0,1200,0,0,0,0,0,*
ENT,1,0.0499,100,0,0,0,0,0,*
FIN
'SYS','1','NEW','KEEP','*'
'MPU','0','3840','2','*'
'MEM','CORE','NEW','*'
'LOAD'
'MPU','0','ADDR','0000','00','00','01','00','00','02','01',
'06','06','02','00'
'08','08','02','00'
'ADDR','0704',
'00','00','ADDR','0810','08','20','ADDR','0814','08','50','ADDR','081A'
'00','ADDR','FFF8','09','00','ADDR','FFFE','09','20',
'ADDR','0700','00','80','04','00'
'ADDR','0800','PIAA','1','100','1','PIAB','0','100','1'
'ADDR','0850','08','01','01','02','01','03','01','FF','01','07','01',
'04','01','05','01','FF','01',
'09','01','0B','01','0A','01','FF','01'
'ADDR','0820','18','00','0A','00','01','0B','00','02','0C','00'
'03','0D','00','04','14','00','05','15','00'
'06','16','00','07','17','00'
'08','18','00','09','19','00','0A','1E','00','0B','1F','00'
'*'
'TRC','0','1','0900','09FF','*'
'SIM','0',00000000000000000000000000000000B,'*'
'END'
```

Figure 5.7: Sample Input Deck.

A number of optional special functions were built into the simulator and can be specified by inputs:

1. The initial system may be stored on a sequential file for further runs. This system may be reused and altered in later runs.

2. The facility is present to print the entire contents of the memory for any MPU in a dump format. This requires a large amount of CPU time and paper, but can be useful in certain cases. The memory dump can be printed before and after the execution of the simulation.

3. An echo-check of the initialization inputs is also available, providing a method of verifying the model parameters.

4. Although it is a run-time function, event monitoring can be specified by inputs as follows:
    1. Printing any conditions or errors which occur in the MPU model.
    2. Printing MPU status information at each event time.

The last option is basically a program diagnostic function and was built in for debugging purposes. The appendices contain examples of all the print functions.

Once the initialization is complete, the second phase starts: model execution. As was stated earlier, a discrete model is used. Events are executed by stepping from one to the next. Five basic event type codes were defined (Table 5.1). GASP-PL/I uses FILE(1) as the events file and all events are scheduled by placing them into the events file. Files are accessed using the ATRIB(*) buffer array and the GASP-PL/I programs: FILEM and RMOVE. ATRIB(1) is the time at which the event is to occur, and ATRIB(2) is the event code.

Table 5.1: Event Code Definitions.

EVENT CODE	EVENT TYPE
10≤ code ≤ 30	MPU instruction event. The MPU number is given by CODE-10.
100≤ code ≤120	MPU halt event. MPU number is given by CODE-100.
CODE=300	I/O event. These are decoded by procedure IOEVNT (see Table 5.2).
1000≤ CODE ≤1020	Maskable interrupt scheduling event. The MPU number is given by CODE-1000.
1100≤ CODE ≤1120	Non-maskable interrupt scheduling event. The MPU number is given by CODE-1100.
1200≤ CODE ≤1220	Reset or restart event. The MPU number is given by CODE-1200.

Events are loaded into the events file in ascending order according to the contents of ATRIB(1). If two events are scheduled to occur at the same time, they are loaded into the events file in first-in-first-out fashion. Note, an event cannot be scheduled to occur at a time that has already past. An error results and the simulation terminates.

The simulation model must be started in either of two ways. A reset can be scheduled to occur by storing an entry in the events file (inputted by GASP-PL/I) or by starting the MPU at a specific address, specifying the contents of all the registers at that time. The latter method was built in to allow parts of M6800 programs to be executed.

During the model execution, errors and conditions that occur for

each MPU may be printed if specified by inputs. The most useful run-time printout is the trace print. This print-out contains the contents of all the registers as a result of the instructions performed. Trace prints can be specified to:

1. Print all the MPU instructions.
2. Print any instruction executed in a certain address range.
3. Print instructions executed within a certain time range.
4. Print only instructions that result in program branches.

The trace type is specified for each MPU in the system independently.

Two methods are used for stopping the model execution. The first is to specify a time limit for the simulation to GASP-PL/I. The second is to halt all of the MPUs, at which time the final printouts will start. It is best to still include a time limit to insure that I/O events will not continue until the default GASP-PL/I time limit (1.0E+20). The simulation will also terminate on fatal errors in the MPU or the system, and if all of the MPUs are halted. If a program error occurs the final printout will be printed, except in extreme cases.

Final summary prints include the following:

1. Any trace entries remaining in the trace files due to buffering.
2. The final contents of all of the MPU registers.
3. The final contents of all the peripheral unit registers.
4. Simulation statistics.
5. The GASP-PL/I summary report if enabled by inputs.
6. Any memory dumps enabled by inputs.

The BIM was modeled as a black box, using one external procedure, ODEV, which performs all of the functions specified in chapter 4. This procedure also models the system bus. The bus does not have to be modeled separately, because only the BIMs and the other subsystems interact with it. All bus functions are transparent to the CPS MPU, only the results are transferred. Response times are calculated using the timing estimates discussed earlier.

BIMs are monitored for the following:

1. The time the BIM must wait to get control of a bus.
2. The number of output bytes sent over the bus.
3. The number of input bytes read from the system bus by each BIM.

5.4 PROGRAM DESCRIPTIONS

The programs developed follow the hierarchy in Figure 5.8. There are four functional legs:

1. Initialization (Figure 5.9).
2. Support routines (Figure 5.10).

3. Run-time (Figure 5.11).

4. Final summary outputs (Figure 5.12).

The highest program in the hierarchy is MAIN. This procedure serves as the entry and exit points for the entire program. The procedure GASP, is called by MAIN to perform the simulation. In Figures 5.8 through 5.12, all dashed line boxes are GASP-PL/I programs, solid line boxes are programs written for this study, and double lined boxes are subprograms which are used in more than one place in the simulation.

GASP is the executive and controller for the entire simulation. It decides when to call any of the programs at the top of the four functional legs. Each of these is a modular hierarchy and may include programs used in other legs. Table 5.2 is a complete list of the non-GASP-PL/I programs and procedures used in this simulation.

Table 5.2: Simulation Program Descriptions.

NAME	FUNCTION	DESCRIPTION

BBIN(DATA)	support	Function which converts the hexadecimal parameter to fixed binary. A zero is returned for any invalid hexadecimal numbers.
		
		Declaration:
		`DCL BBIN ENTRY(CHAR(6))`
		`RETURNS(FIXED BIN(16,0));`

BRANCH	run-time	Used by the conditional branch instructions to perform the relative address calculations required for the branch. Declaration: `DCL BRANCH ENTRY;`

CCRCHK(RESULT,A, B,CKH,CKI,CKN, CKZ,CKV,CKC)	run-time	Performs the logical operations to update the condition code register bits using the following code:
		
		CKx=0: Reset bit x.
		CKx=1: Set bit x.
		Ckx=2: No operation performed.
		CKx=3: The bit x is set or reset according to the three operands A, B, and RESULT using the standard logical operation.
		
		Declaration:
		`DCL CCRCHK ENTRY(FIXED BIN(8,0),`
		`FIXED BIN(8,0),FIXED BIN(8,0),`
		`FIXED DEC(1,0),FIXED DEC(1,0),`
		`FIXED DEC(1,0),FIXED DEC(1,0));`

CONPRT(STRING)	support	Used by ERR and TRCPRT to print MPU conditions that are flagged in SYSTBL(7,MPU). Declaration: `DCL CONPRT ENTRY(BIT(31));`
ECHO	initialization	Used to print the initial system configuration.

Table 5.2: Simulation Program Descriptions (continued).

NAME	FUNCTION	DESCRIPTION
		Declaration: DCL ECHO ENTRY;
ERR	support	Prints the run-time error diagnostics for system errors and conditions. Declaration: DCL ERR ENTRY;
ERRPRT(STRING)	support	Called by ERR and TRCPRT to print errors that are flagged in SYSTBL(8,MPU). Declaration: DCL ERRPRT ENTRY(BIT(31));
EVNTS	run-time	Used to decode the event codes and transfer control to the routines which perform the events. EVNTS also schedules subsequent instruction events. Declaration: DCL EVNTS ENTRY(FIXED BIN(31,0));
HHEX(DATA)	support	Complimentary of BBIN. HHEX converts binary inputs to a hexadecimal character string. Declaration: DCL HHEX ENTRY(FIXED BIN(31,0)) RETURNS(CHAR(6));
INCHK	run-time	Performs the interrupt checking and processing for the MPUs in the system. Declaration: DCL INCHK ENTRY;
INSCHD(CODE)	run-time	Used to schedule interrupts according to the following code: CODE= 10xx maskable interrupt. 11xx non-maskable interrupt.

Table 5.2: Simulation Program Descriptions (continued).

NAME	FUNCTION	DESCRIPTION
		12xx reset. where xx is the MPU number. Interrupts can be scheduled by loading the above codes in the events file. Declaration: DCL INSCHD ENTRY(FIXED BIN(31,0));
INSTR	run-time	This program contains all of the procedures which perform the MPU instructions. They were combined in one procedure because of common variables. Declaration: DCL INSTR ENTRY;
INTLC	initialization	Perfroms the system initialization by reading the inputs and constructing the system model. See Appendix 2 for the input specifications and formats. Declaration: DCL INTLC ENTRY;
IOEVNT	run-time	This program performs all of the I/O events on ACIAs and PIAs in the system. ODEV is called to model any non-MPU devices in the system. All user-defined events are scheduled as I/O events. The format for I/O events is: ATRIB(1)= event time. ATRIB(2)= 300. ATRIB(3)= device number. ATRIB(4)= event code. ATRIB(5)= data (data transfers only). ATRIB(6)= address of device. The event codes used are: 1 -data transfer event. 2 -PIA C1 set or ACIA CTS set. 3 -PIA C1 reset or ACIA CTS reset.

Table 5.2: Simulation Program Descriptions (continued).

NAME	FUNCTION	DESCRIPTION
		4 -PIA C2 set. 5 -PIA C2 reset. >5 -user-defined events. Device numbers from 0 to 19 are reserved for MPUs. Any time a device number greater than 19 is found, ODEV is called to process the event. Declaration: DCL IOEVNT ENTRY;
MAIN	entry/exit	Used to call GASP to start the simulation. MAIN cannot be called by another routine.
MDUMP	support	When called, MDUMP will print a hexadecimal format dump of the entire contents of the memory for the MPU number specified in the variable MPU. Declaration: DCL MDUMP ENTRY;
MFIND(ADDRESS, MCODE)	run-time	This procedure manages the memory model. MFIND returns the location of the addressed memory location in the array LOC(0:31). Invalid addresses return a negative value. MCODE specifies the following operation: 0 -Test to see if the memory location address is valid, but do not access the memory. 1 -Test the memory address for for validity, and access the Declaration: DCL MFIND ENTRY(FIXED BIN(16,0), FIXED BIN(31,0)) RETURNS(FIXED BIN(31,0));
MPHLT(CODE)	run-time	Halts the MPU specified by code. MPU halts can be scheduled as events by using an event code of 100+MPU.

Table 5.2: Simulation Program Descriptions (continued).

NAME	FUNCTION	DESCRIPTION
		Declaration: DCL MPHLT ENTRY(FIXED BIN(31,0));
ODEV	run-time	Performs all external device functions in the simulation. IOEVNT calls ODEV whenever an I/O event with a device number greater than 19 is found. For this simulation, ODEV modeled the BIMs and the system bus. Declaration: DCL ODEV ENTRY;
OTPUT	summary	OTPUT prints the final summary for the system. USUM is called to print any external device results. Declaration: DCL OTPUT ENTRY;
READ(ADDRESS)	run-time	Read performs all MPU memory read operations on the simulated memory. The external variable, MPU, must be set to the number of the MPU whose memory is to be accessed before calling READ. ADDRESS must be a valid memory address for the MPU. READ returns the contents of the memory location. Declaration: DCL READ ENTRY(FIXED BIN(16,0)) RETURNS(FIXED BIN(8,0));
TRACE	support	Stores the trace print entries for each MPU. The file which is used to store the entries is numbered MPU+3. Trace will call TRCPRT to print a trace table when the filing array is full. The file storage array must be specified large enough to hold at least 60 trace entries per MPU to print full page trace prints.

Table 5.2: Simulation Program Descriptions (continued).

NAME	FUNCTION	DESCRIPTION
		Declaration: DCL TRACE ENTRY;
TRCPRT	support	Prints trace prints for the MPUs. The MPU number is given in the variable MPU. Declaration: DCL TRCPRT ENTRY;
UERR(CODE)	support	Called by both GASP-PL/I programs and user programs to print non-GASP-PL/I error codes. The error codes are user defined, and must be less than 100. All error codes greater than 100 are reserved for GASP-PL/I error codes. Declaration: DCL UERR ENTRY(FIXED BIN(31,0));
UMONT(IX)	support	Prints event monitoring information during the model execution if enabled by scheduling an event with the event code 0. Declaration: DCL UMONT ENTRY(FIXED BIN(31,0));
USUM	summary	Called by OTPUT to print external device statistics. In this simulation USUM printed all BIM and system bus statistics. Declaration: DCL USUM ENTRY;
WRITE(DATA, ADDRESS)	run-time	Complement of READ. WRITE performs all MPU memory write operations on the simulated memory, including peripheral device functions. Declaration: DCL WRITE ENTRY(FIXED BIN(8,0)), FIXED BIN(16,0));

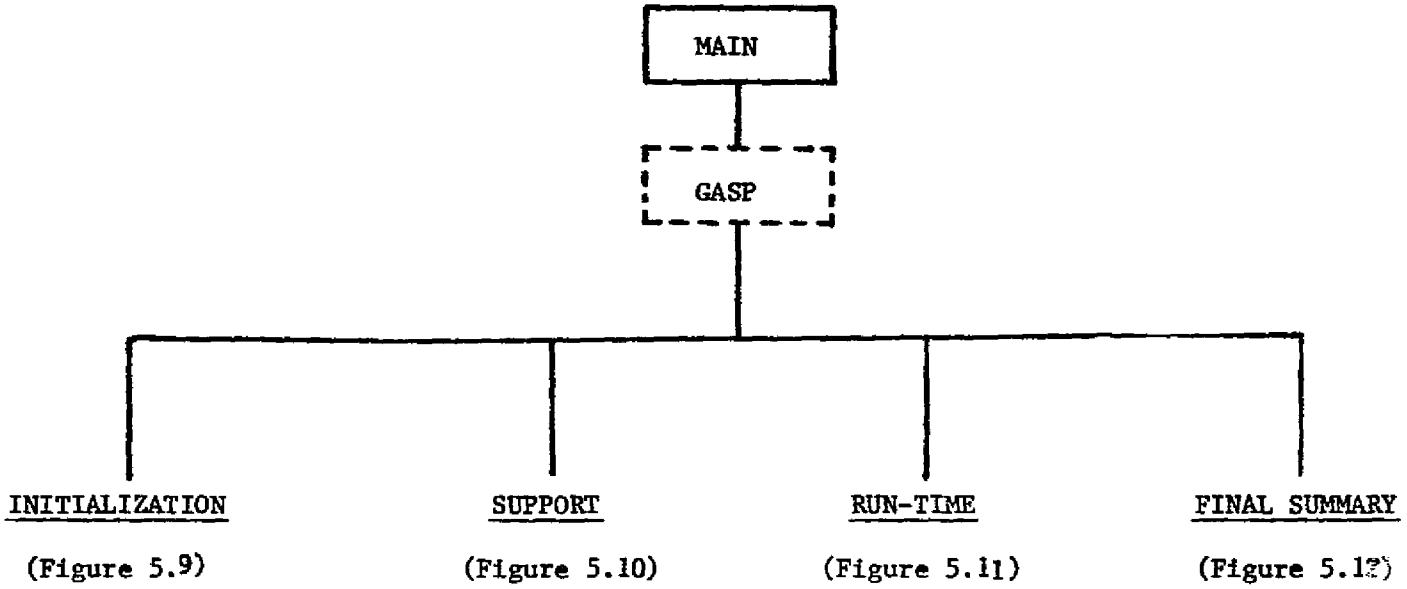
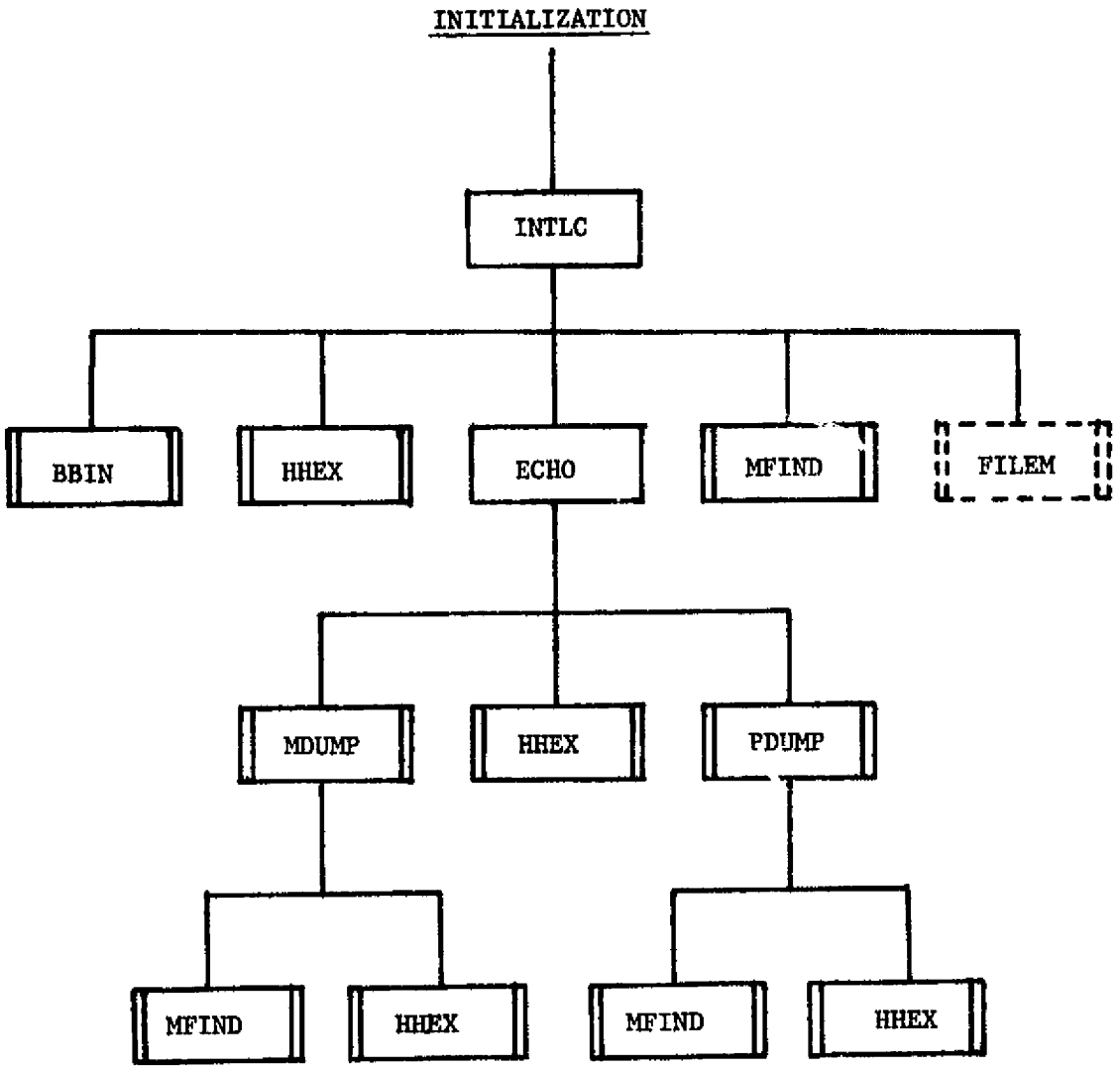


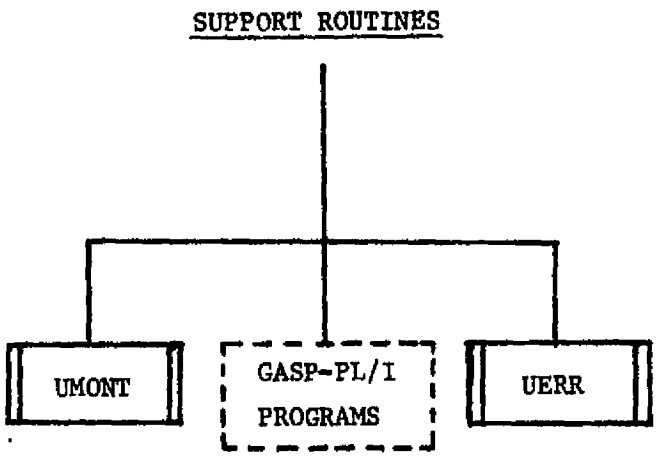
Figure 5.8: Simulation Program Hierarchy.



----- GASP-PL/I programs

_____ programs written by the author

Figure 5.9: Initialization Program Hierarchy.



----- GASP-PL/I programs

_____ programs written by the author

Figure 5.10: Support Routines Hierarchy.

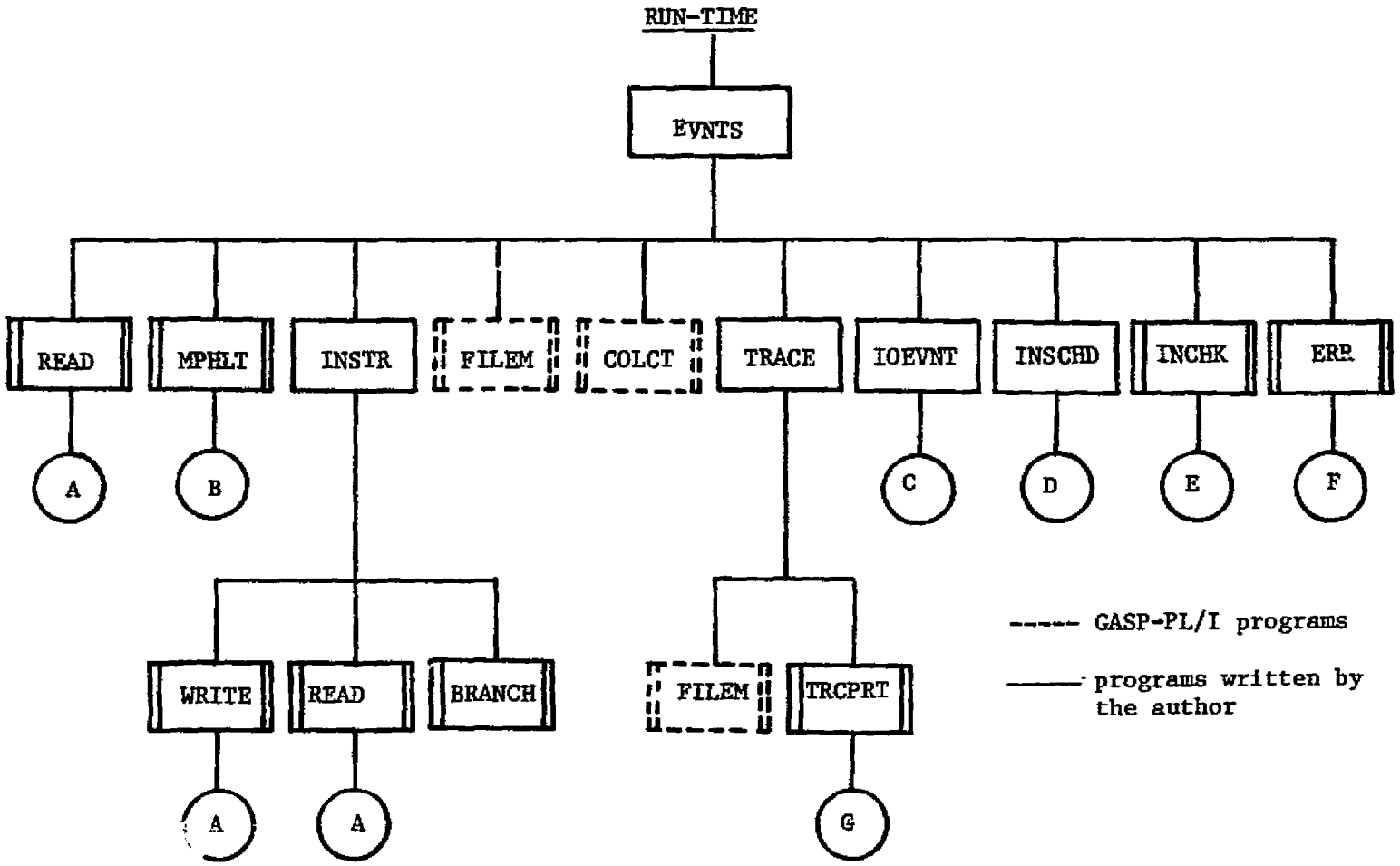
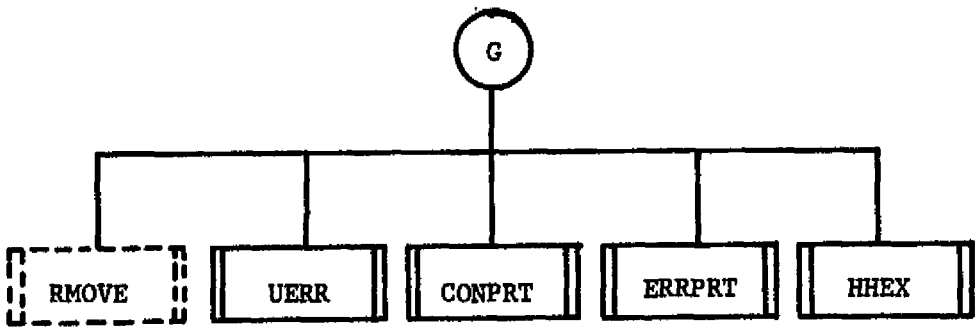


Figure 5.11: Run-Time Program Hierarchy.



----- GASP-?L/I programs

——— programs written by the author

Figure 5.11: Run-Time Program Hierarchy (continued).

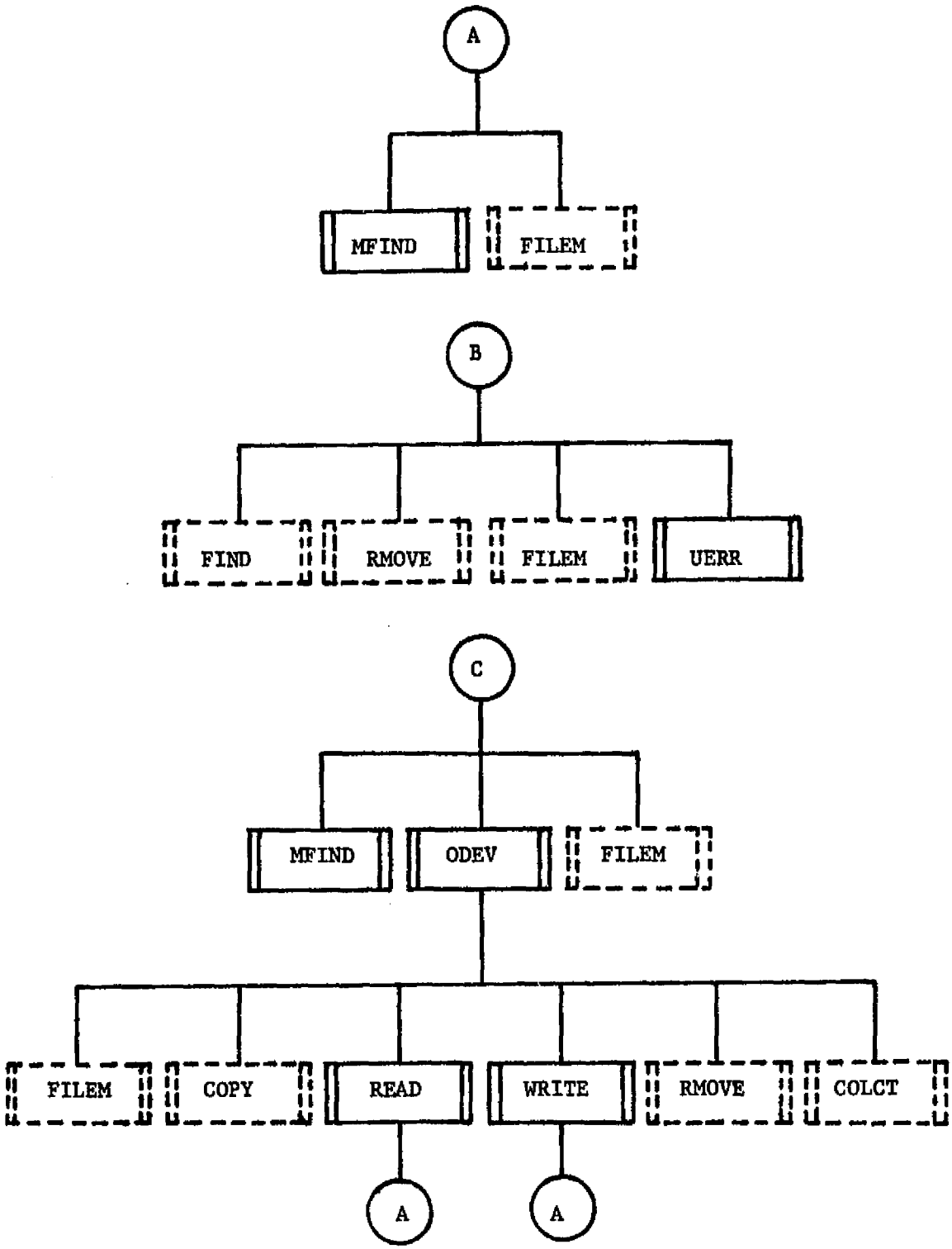
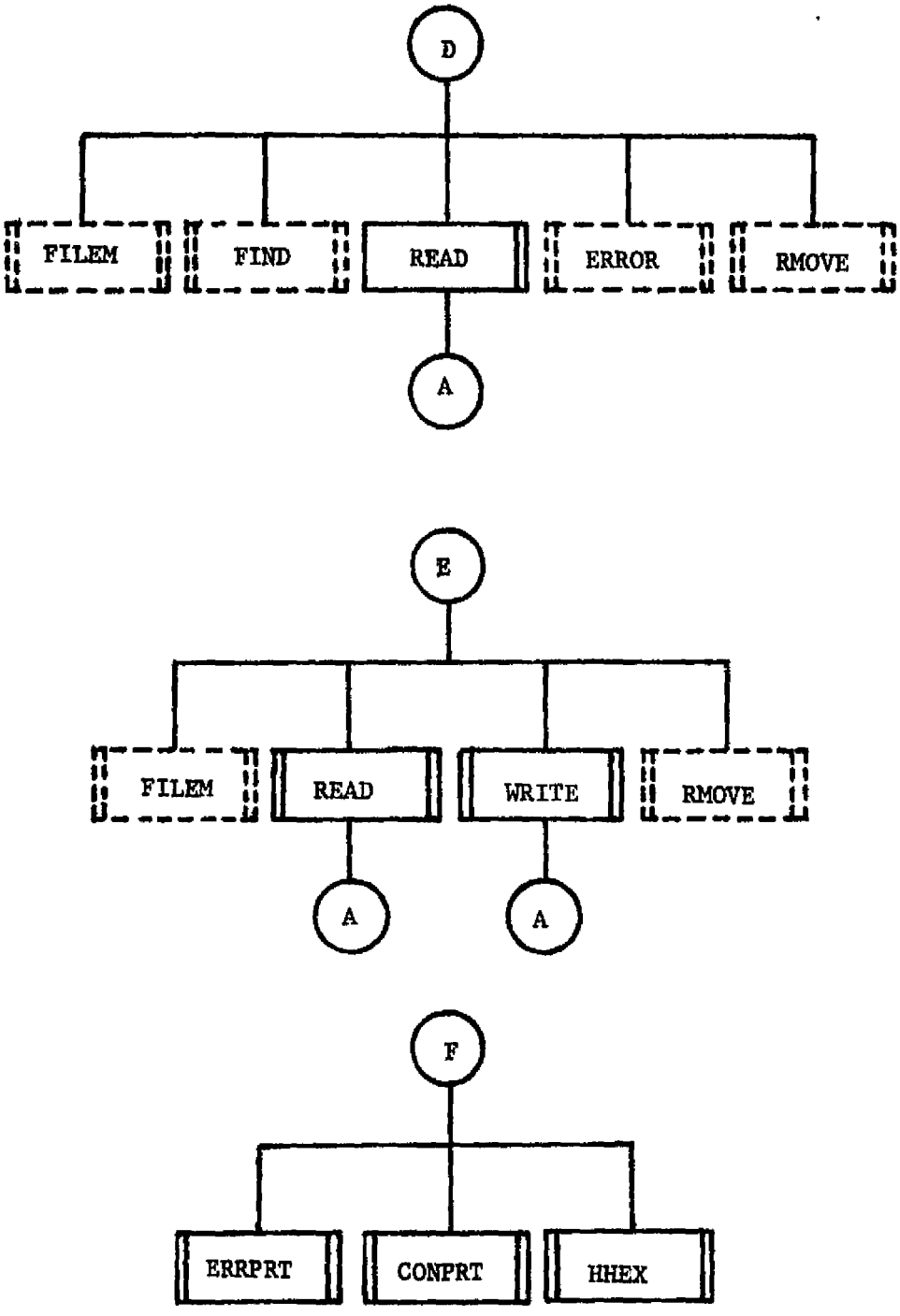


Figure 5.11: Run-Time Program Hierarchy (continued).



——— programs written by the author

----- GASP-PL/I programs

Figure 5.11: Run-Time Program Hierarchy (continued).

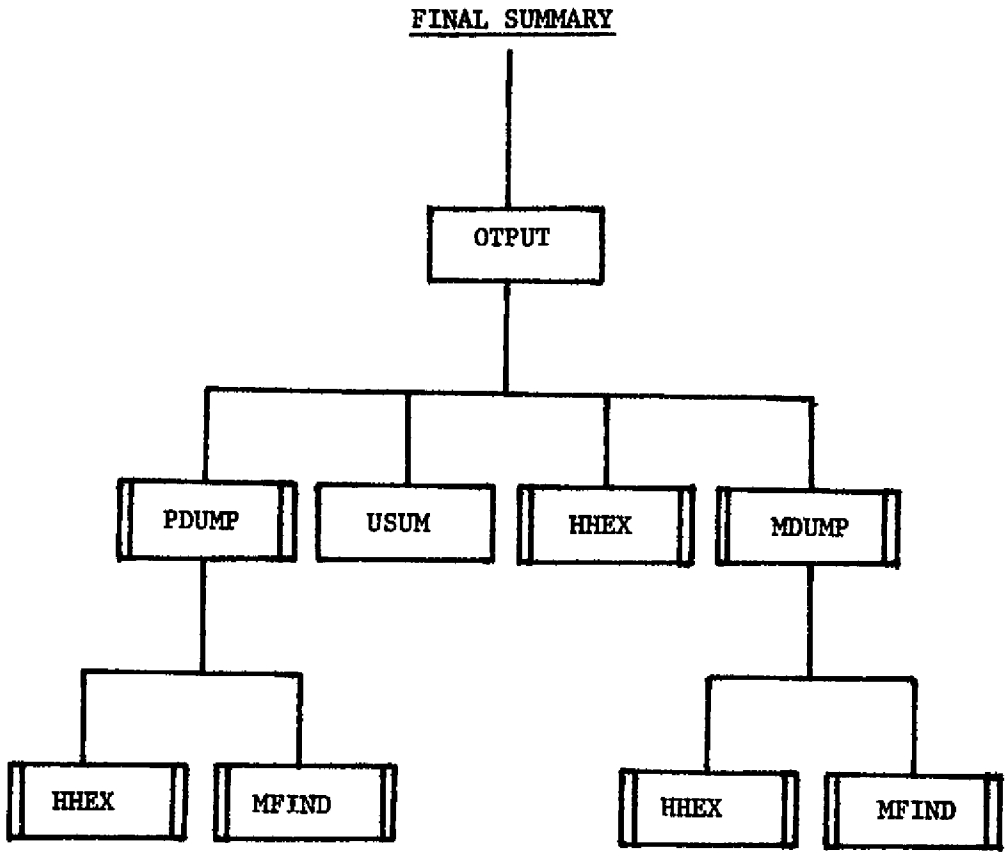


Figure 5.12: Final Summary Program Hierarchy.

6 METHODS AND RESULTS

The basic purpose of the simulation presented here is to study and evaluate the characteristics of the Team Architecture described in the previous chapters. The operating characteristics evaluated were defined as:

1. The computing overhead required to manage the Team Operating System (TOS). Included in the overhead are:
   a. The time spent waiting to get control of a system bus.
   b. The time spent waiting to update the operating system.
   c. Computation performed while updating the operating system.
2. The affect of different program types on the operating system performance.
3. The system bus load.

Team Operating System programs were written according to the description given in chapter 3. The program used is simpler than what would be implemented in the real system, but still performs the operating system functions. The TOS program listed in Appendix 2 was used throughout all of the tests. Task programs scheduled and run by the operating system were varied, however, to exercise the architecture.

Task programs can be classified in two basic types: programs with little or no system bus I/O, or ones with a large amount of I/O. Since the system bus forms the central element of the Team Architecture, program I/O would appear to limit system efficiency. The tests were structured to evaluate the affects of both varying program lengths and I/O on the system.

With this in mind, three series of tests were run. In all three, each CPS MPU contained identical copies of the TOS and test programs. Data was collected on bus waits, TOS waits, operating system overhead, and program execution times.

A major handicap was encountered which prevented long test runs. As with many simulation programs, the ratio of CPU time used to simulated time is far greater than 1:1. This simulator, depending on the MPU programs being simulated, varies from approximately 5,000:1 for one MPU to 15,000:1 for four MPUs. This can be attributed to the program complexity, to the inefficiency inherent in PL/I, and to the modular program structure. As a result, all tests could only be performed for very short intervals of simulated time. This severely limited the evaluations.

The first series of tests contained test programs without any system bus I/O. The only system bus I/O resulted from the TOS updates. Program execution times were varied to determine the affect on the system. Table 6.1 and Figures 6.1 through 6.4 show the result of these tests. The average job execution time was varied from approximately 200 microseconds to 20 milliseconds. Again the time restriction limits the lengths. The number of levels was also varied.

The second series of tests contained programs which produced varying amounts of system bus I/O. Again, program execution times and the number of levels was varied. The results are tabulated in Table 6.2 and plotted as a function of the average execution time in Figures 6.5 and 6.6.

The third series varied system bus I/O, program execution times,

and the number of levels in the system. In addition, the programs run were constructed to dynamically modify which jobs were run by setting and resetting the Operational Status Bits (OSBs) of the programs in the system tables. The results are tabulated in Table 6.3.

Table 6.1: Series I Test Results.

RUN NUMBER	NUMBER OF TASKS	NUMBER OF LEVELS	AVERAGE TASK TIME (usec.)	AVERAGE OS TIME PER TASK (usec.)	OVERHEAD (%)	MEAN BUS WAIT (usec.)	MEAN TOS WAIT (usec.)	AVERAGE SYSTEM BUS DATA RATE (Kb/sec.)
1	110	1	173.8	1667.1	90.5	1194.3	1302.2	14.8
2	107	1	385.6	1498.3	79.5	1027.7	1738.6	14.6
3	105	1	789.7	1085.5	57.9	657.6	774.7	14.3
4	91	1	1383.3	762.6	35.5	304.0	424.0	12.6
5	57	1	2699.5	648.1	19.4	211.9	335.1	7.8
6	31	1	5323.6	565.5	9.6	156.9	290.3	4.3
7	18	1	9530.0	500.2	4.9	133.3	261.4	2.5
8	107	2	197.6	1645.5	89.3	1230.8	1340.0	22.7
9	104	2	373.5	1510.4	80.2	1111.1	1223.4	22.2
10	101	2	716.1	1217.1	63.0	585.0	881.0	21.7
11	93	2	1326.8	778.1	36.9	313.0	430.4	21.1
12	58	2	2669.4	653.9	19.6	224.0	346.6	12.5
13	31	2	5258.1	620.0	10.5	172.0	312.5	6.8
14	16	2	10116.8	502.1	4.7	153.5	279.7	3.5
15	97	3	703.5	1315.3	65.2	878.5	1005.7	28.5
16	88	3	1363.6	854.2	38.5	398.0	511.0	25.8
17	57	3	2569.4	840.3	24.6	235.4	354.2	16.8

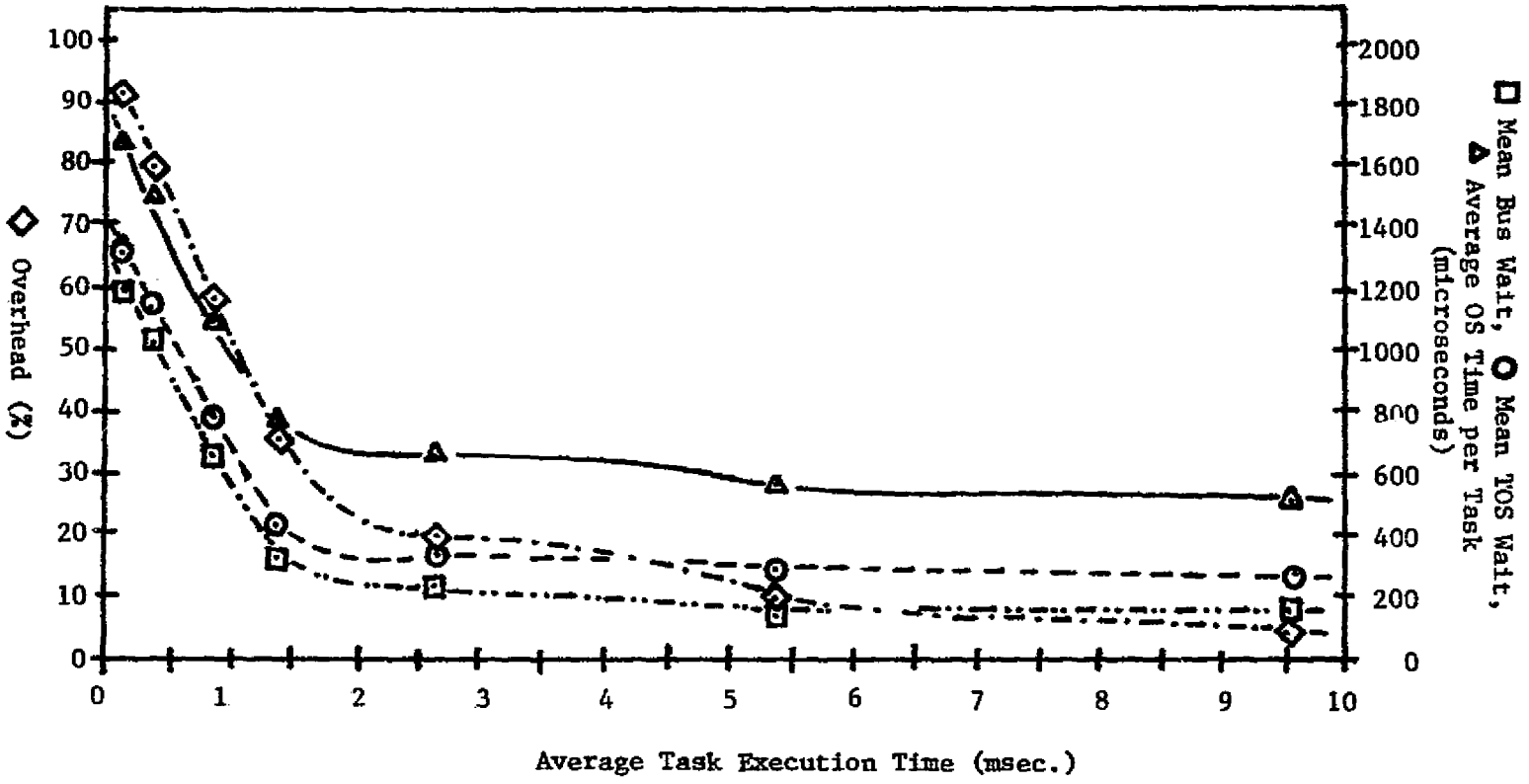


Figure 6.1: Series I Results: One Level.

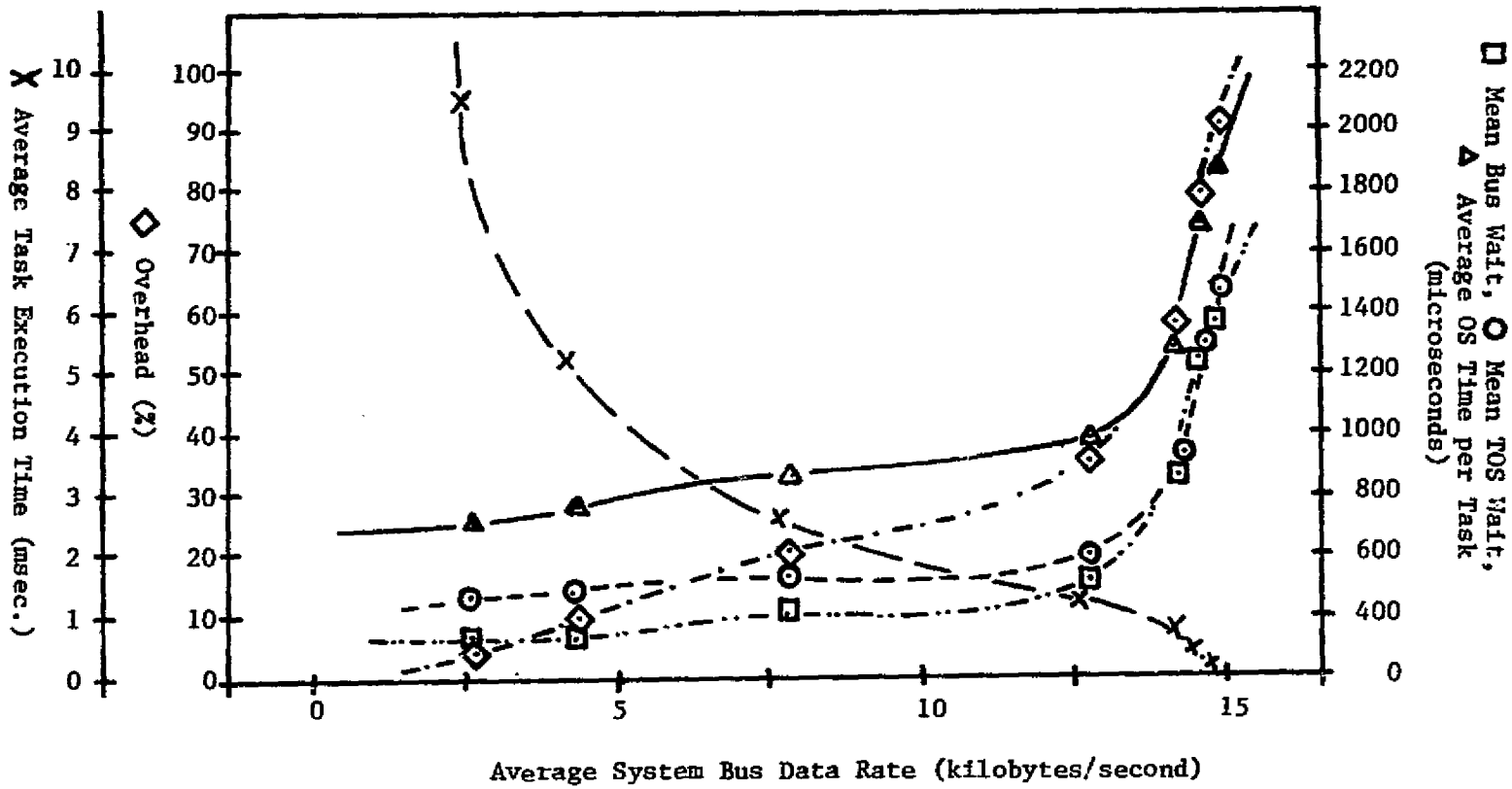


Figure 6.2: Series I Results: One Level.

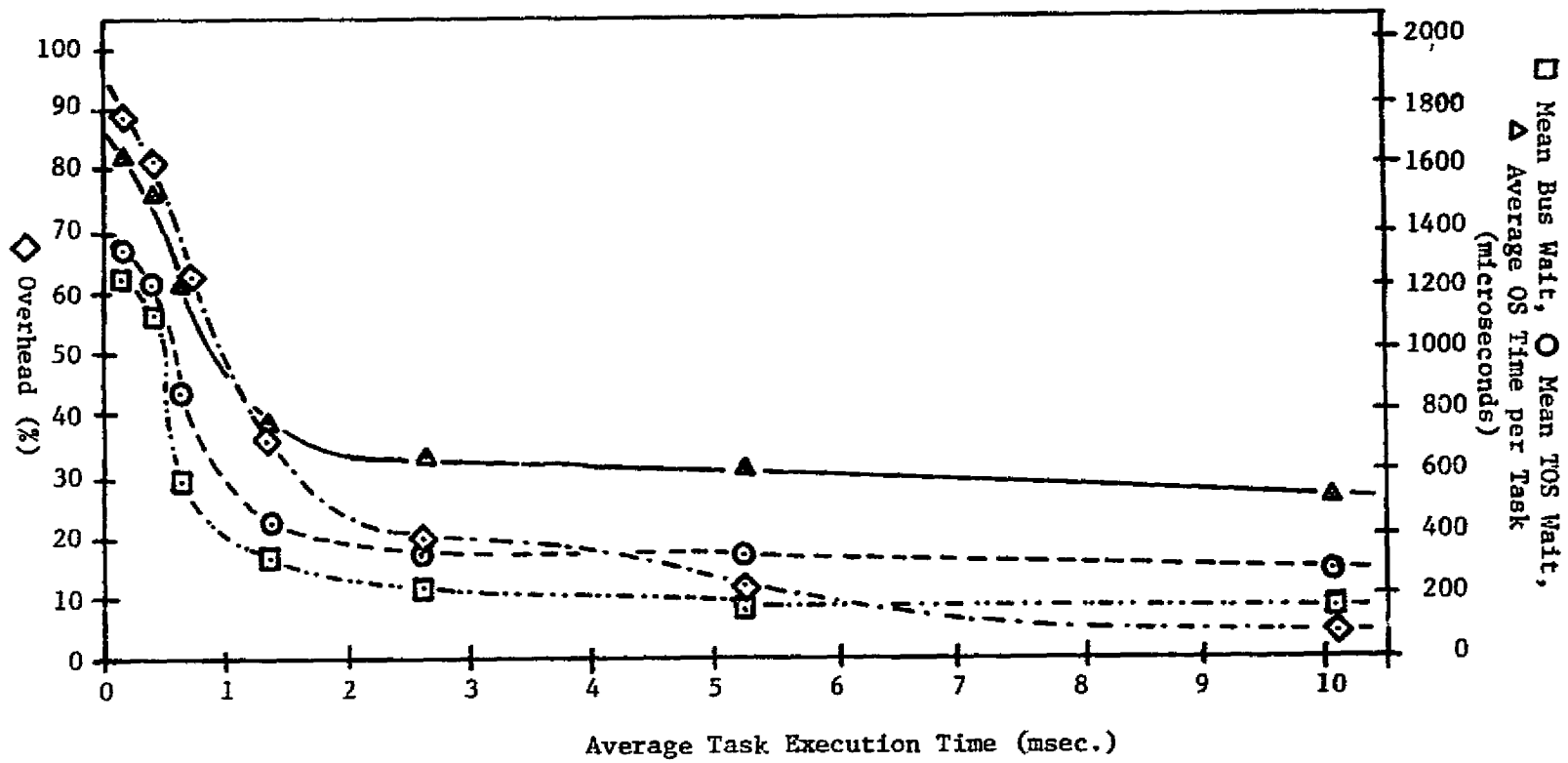


Figure 6.3: Series I Results: Two Levels.

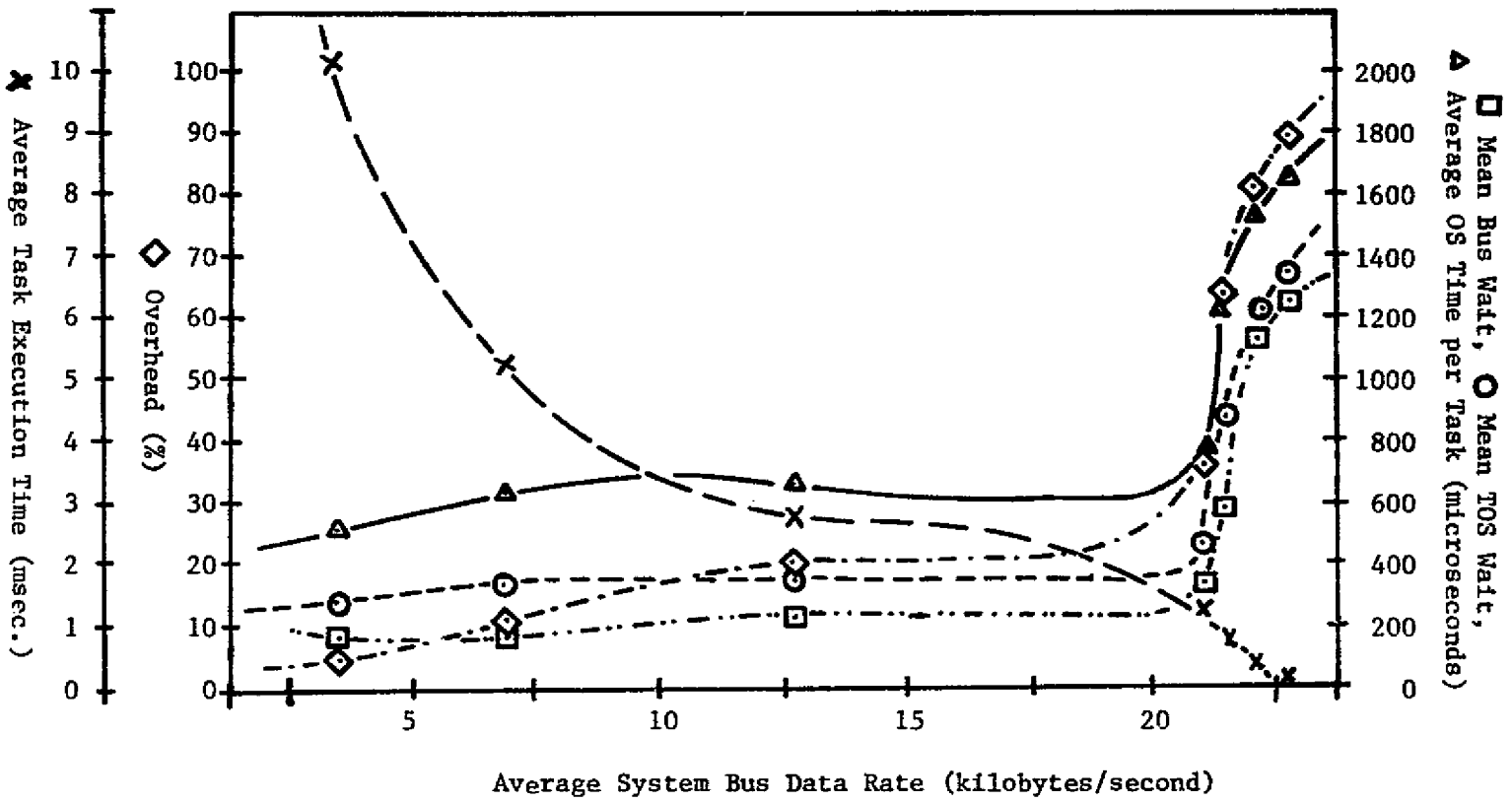


Figure 6.4: Series I Results: Two Levels.

Table 6.2: Series II Test Results.

RUN NUMBER	NUMBER OF TASKS	NUMBER OF LEVELS	AVERAGE TASK TIME (usec.)	AVERAGE OS TIME PER TASK (usec.)	OVERHEAD (%)	MEAN BUS WAIT (usec.)	MEAN TOS WAIT (usec.)	AVERAGE SYSTEM BUS DATA RATE (Kb/sec.)
20	73	1	3460.2	681.0	16.4	547.0	313.0	35.7
21	81	2	3350.8	876.6	20.7	553.0	508.0	43.3
22	96	3	3071.1	1034.6	25.2	592.0	711.0	51.7
23	54	1	5614.1	1407.7	20.0	1084.0	1141.0	47.1
24	55	2	5626.6	1292.4	18.7	868.0	988.0	53.0
25	52	3	5944.7	1438.9	19.5	1015.0	1156.0	52.0
26	29	1	10716.9	2232.6	17.2	2041.0	2081.0	47.3
27	25	2	10885.4	2627.8	19.4	1789.0	1802.0	50.7
28	31	3	10027.2	1381.0	12.1	1180.0	1034.0	55.8
29	106	1	15341.5	3007.3	16.4	2835.0	2760.0	50.7
30	106	2	15493.3	3049.3	16.4	2964.0	2819.0	50.2
31	58	3	15360.4	2373.5	13.4	2549.0	2112.0	47.9

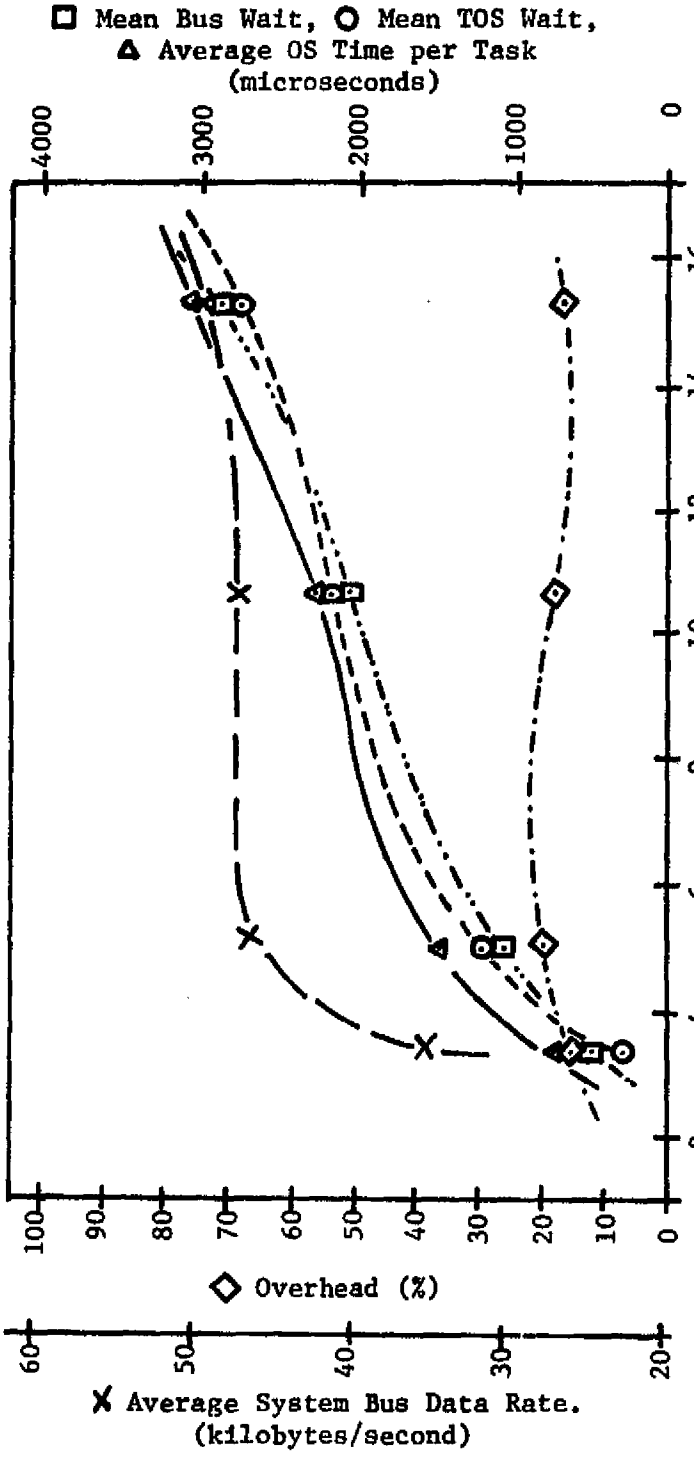


Figure 6.5: Series II Results: One Level.

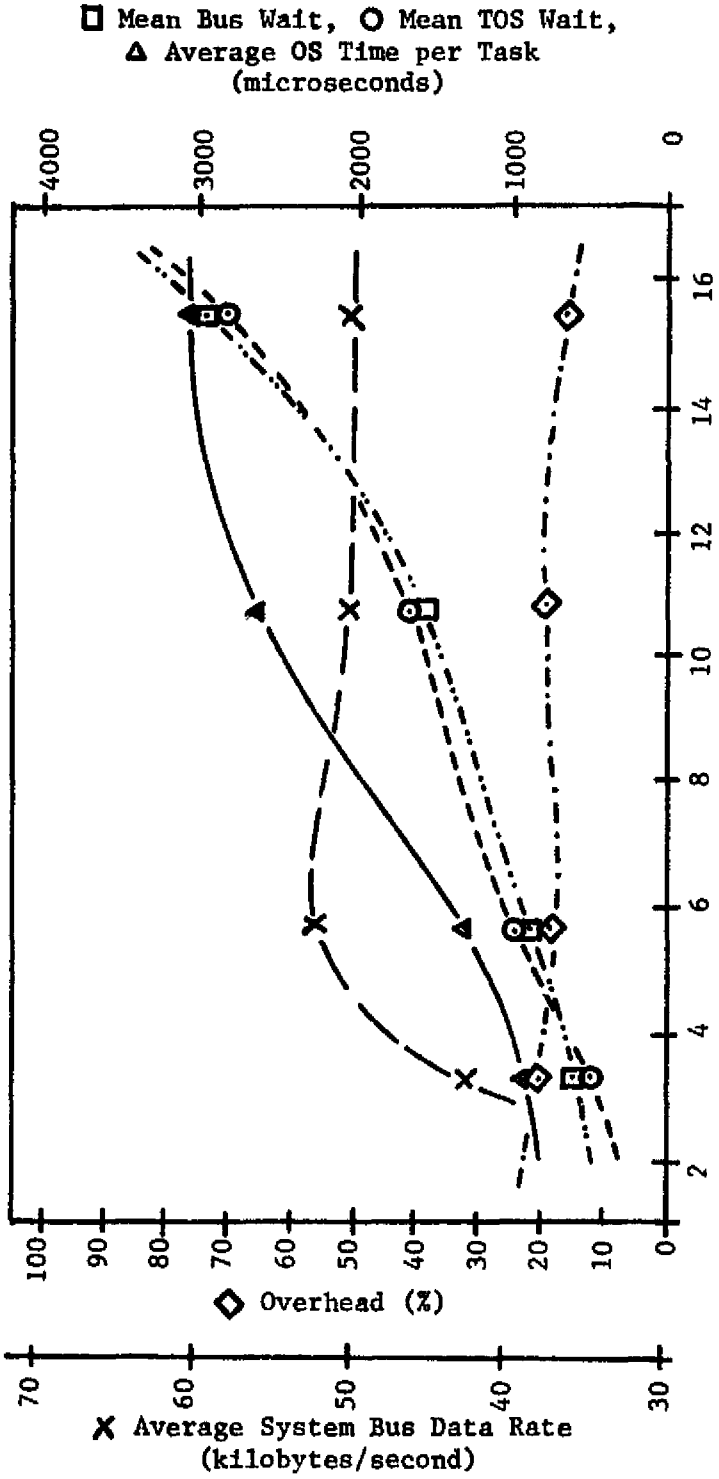


Figure 6.6: Series II Results: Two Levels.

Table 6.3: Series III Test Results.

RUN NUMBER	NUMBER OF TASKS	NUMBER OF LEVELS	AVERAGE TASK TIME (usec.)	AVERAGE OS TIME PER TASK (usec.)	OVERHEAD (%)	MEAN BUS WAIT (usec.)	MEAN TOS WAIT (usec.)	AVERAGE SYSTEM BUS DATA RATE (Kb/sec.)
32	74	2	3556.9	1360.3	34.7	395.0	495.0	31.6
33	62	1	2569.7	1135.6	30.6	374.0	324.0	26.8
34	76	3	2159.2	1185.6	35.4	398.0	523.0	36.2

7 CONCLUSIONS

The graphs in chapter 6 show the characteristics of the Team Architecture when programs up to 20 milliseconds in length are run. Unfortunately CPU time usage prevented longer tests. However, the graphs do show some interesting trends and system characteristics.

Series I used programs without system bus I/O, varying the average task execution time from 200 microseconds to around 10 milliseconds. Although the actual system will probably contain very few programs of this length, they are one of the possible extremes. Very short programs (less than 2 milliseconds) increase the system overhead greatly. This results from more time being required to update the operating system than it takes to run each job.

The test programs run in Series I require only operating system I/O to be sent over the system bus. This is the minimum possible for the Team Architecture. The operating system update time stabilizes between 500 and 600 milliseconds for programs greater than 2 milliseconds in length. This can be expected to continue for longer programs than tested, since system bus I/O decreases as task execution times increase. Each CPS sends only a fixed amount of data each time the operating system is updated.

Series II tests were performed to analyze the affect of program I/O on the operating system. The test programs generated increasing amounts of system bus I/O and also had execution times proportional to the amount of I/O generated. The maximum data buffer size that can be

sent at one time by the BIM is 255 bytes. The amount of data sent by each group of test programs was increased until all were sending and receiving 255 byte buffers.

The graphs for Series II (Figures 6.5 and 6.6) show the average system bus data rate increasing until about 50 kilobytes per second where it levels off. The maximum system bus data rate appears to be between 50 to 55 kilobytes per second. Even at this system bus load, The system overhead stayed between 15% to 20%. However, the TOS update time increases with the average execution time as does the mean bus wait and the mean TOS wait.

Series III contained test programs which varied both execution time and the number of data bytes sent over the system bus. Also, the programs executed were altered by setting and resetting the Operational Status Bits (OSB) during the program run.

The results of Series III runs showed an increase in system overhead to approximately 30%. This resulted from the TOS skipping jobs which had the OSB set to zero. Bus waits were not high averaging 374 to 400 microseconds. Since the system bus data rate was less than 37 kilobytes per second the average operating system update time can be expected to remain fairly constant if the average task execution time increases. This would result in a decrease in system overhead.

It should be noted that the system characteristics are dependent on the software. Many software configurations are possible for the Team Operating System, and the programs used perform the basic algorithm. In the final system, software would be more complex and probably require more time to execute. Very little effort was made to

try to optimize the operating system software, and some efforts in this area could help.

It was surprising to note that the average system bus data rate stabilized at approximately 50 kilobytes per second, less than two-thirds of the theoretical maximum of 90.9 kilobytes per second as described in chapter 3. This can be attributed to a number of factors, the most important of which is the software configuration. The CPS software was written so that it would obtain control of a system bus and not release it until completely finished with the current task. This allows the bus to sit idle while the CPS generates the data to be sent. A more efficient method might improve the system bus data rate.

Another area of possible improvement is the Bus Interface Module (BIM). The realization described in Chapter 3 is slow compared to the capabilities of the system bus. Using a microprocessor with a faster clock period than 1 microsecond would increase the system bus data rate greatly, since it is the microprocessor which limits the speed. Although more expensive in developmental costs, a specialized hardware BIM that executes the transfer faster than the one used could improve data rates greatly. If the system could run all the busses at their maximum data rate of 1 megabyte per second, the system bus load would not be a limiting factor on efficiency.

The results just discussed represent a detailed study of the lower end of the task execution times for the Team Architecture. CPU time usage restricts the size of programs tested and this should be evaluated to complete the system characteristics. This simulation

program is not suitable for studying very long programs. A more gross simulation would be more appropriate for studying the longer programs.

8 RECOMMENDATIONS

As a result of the major effort required to develop the simulation model and the programs which implemented it, the tests performed were limited to just a few areas. Some further tests came to mind during the course of the evaluation presented in the previous chapter.

The software used in the TOS tests is only one of many possible configurations. Efficiency could be improved by optimizing the operating system programs. Much of the update time was spent waiting on the BIM to obtain control of a system bus. This time could be more effectively spent running tasks, instead of in a wait loop. This would reduce the overhead, plus allow the CPS to do some local programs such as diagnostic tests.

Another interesting point which could be investigated further is the frequency at which the tasks are executed. This depends on the number of levels, the number of tasks in each level, the length of time required to execute each task, and the number of times the level is cycled through before going to the next level. An equation was not derived for this, and it would be interesting to see if one could be. Obviously, the relationship would not be simple, since the frequency depends on so many factors and would vary dynamically as the system ran.

Although the system was designed to utilize a combination hardware and software, software is almost the sole cause of time

delays in the system. The most noticeable point is the BIM. According to the IEEE 488 bus standard, each system bus must be capable of operating at one mega-byte per second. With three busses, an overall data rate of over two mega-bytes per second should be easily attainable. However, the I/O processor in the BIM limits the average data rate to around 55 kilobytes per second. Only about 2% of the maximum. An interesting thought would be to develop an entirely hardware BIM which could run at one mega-byte per second. The system would run somewhat faster, but the cost of such a design might be prohibitive.

Now that the system has been simulated at the instruction level, further tests could be performed using a higher level model. The basic operating characteristics were found in this study, and these could be used to create a higher level model. A model which simulates the system at the subsystem level would execute faster, allowing more comprehensive tests.

REFERENCES

1. T. M. McCalla, Jr., et. al., "Preliminary Candidate Advanced Avionic System for General Aviation Aircraft," NASA-CR-152025, for NASA Ames Research Center, Southern Illinois University, 1977.

2. "IEEE Standard Digital Interface for Programmable Instrumentation," IEEE Std. 488-1975, The Institute of Electrical and Electronics Engineers, Inc., 1975.

3. "M6800 Microprocessor Applications Manual," Microcomputer Applications Engineering, Motorola Semiconductor Products, Inc., Pheonix, Arizona, 1975.

4. "M6800 Programming Reference Manual," Motorola Semiconductor Products, Inc., Pheonix, Arizona, 1976.

5. "M6800 Microcomputer System Design Data," Motorola Semiconductor Products, Inc., Pheonix, Arizona, 1976.

6. Joan K. Hughes and Jay I. Michtom, "A Structured Approach to Programming," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

7. Joan K. Hughes, "PL/I Programming," John Wiley and Sons, Inc., New York, New York, 1973.

8. "PL/I (F) Programmers Guide," IBM publication: C28-6594, International Business Machines, White Plains, New York.

9. "OS PL/I Optimizing Compiler: Programmers Guide," IBM publication: SC33-006-4, International Business Machines, White Plains, New York, 1976.

10. Robert E. Shannon, "Systems Simulation, the Art and Science," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

11. Thomas H. Naylor, et al., "Computer Simulation Techniques," John Wiley and Sons, Inc., New York, 1966.

12. A. Alan B. Pritsker and Robert E. Young, "Simulation With GASP-PL/I," John Wiley and Sons, Inc., New York, 1975.

13. A. Alan B. Pritsker and Robert E. Young, "The GASP-PL/I Users Manual," Pritsker and Associates, Lafayette, Indiana, 1974.

14. Glen A. Brent, "GASP-PL/I Reference Manual," Southern Illinois University at Carbondale, Carbondale, Illinois, 1978.

15. Glen A. Brent and Thomas M. McCalla, "Exploring Team Avionics Systems by Simulation," Eleventh Annual Simulation Symposium, Tampa, Florida, March 1978.

16. M. S. Jayakumar and Thomas M. McCalla, "Simulation of Microprocessor Emulation Using GASP-PL/I," Computer, April, 1977, pp. 20-26.

17. John B. Peatman, "Microcomputer-Based Design," McGraw-Hill Book Company, New York, 1977.

18. Abd-elfattah Abd-alla and Arnold Meltzer, "Principles of Digital Computer Design," Volume 1, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

19. Harold S. Stone, et al., "Introduction to Computer Architecture," Science Research Associates, Inc., Chicago, 1975.

20. Robert L. Morris and John R. Miller, "Designing With TTL Integrated Circuits," McGraw-Hill Book Company, New York, 1971.

21. Clarence B. Germain, "PL/I for the IBM 360," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

APPENDICES

APPENDIX 1: INPUT FORMATS.

GASP-PL/I uses a "card" format for inputs. Each "card" is used to specify different control or run-time parameters for the simulation. Each "card" can cover more than one physical card, but must follow the format specified in reference 12.

To be read correctly, GASP-PL/I inputs must begin on column one. Inputs may be separated by a comma, or one or more blanks. All character string inputs must be surrounded by single quotes except for the card type and the asterisk which is the last input on each card.

Default values are assumed for all inputs, but to perform a simulation many of the inputs are required. GASP-PL/I requires two cards, GEN and FIN, be included. Sixteen card types may be used as inputs to the GASP-PL/I part of the simulation.

For this simulation, many of the GASP-PL/I variables must be initialized to minimum values according to the initial simulation model configuration. Table A1.1 lists the minimum values which must be specified.

Table A1.1 Minimum initialization values required for GASP-PL/I variables.

CARD	VARIABLE	VALUE
STA	NNCLT	Number of MPUs in the system.
LIM	NNTRY	Large enough to handle the events to be scheduled and the trace files. A good rule of thumb is to make it 100 times the number of MPUs in the system.
LIM	NNATR	Must be at least 8 for the correct operation of the simulator.
LIM	NNFIL	Number of MPUs plus two.
COL	I,'LABEL'	There must be one 'COL' card for each MPU. 'I' is the MPU number and 'LABEL' is the label to be associated with this MPU in the final summary.
INI	MSTOP	This card must be inserted to specify the method of halting the simulation.

Inputs to the M6800 simulator follow the basic format of GASP-PL/I inputs. Card types are used to specify various parameters of the system. All cards are free format. They can start on any column, be separated by a comma or any number of blanks, and cover any number of physical cards. Default values are assumed for all but the two required cards, 'SYS' and 'END.'

The description of the cards will follow a common format. Note that almost all inputs are surrounded by single quotes, and each card ends in an asterisk ('*'). Any errors found will terminate the simulation run, although the entire input deck will usually be

processed.

The following formats are used to describe the inputs:

1. Inputs that are required in each card, will be surrounded by single quotes (i. e. 'SYS').

2. Optional inputs are indicated by square brackets [].

3. Inputs surrounded by braces {} denote hexadecimal numbers. These are inputted as character strings and must be surrounded by single quotes.

4. Inputs surrounded by parenthesis () are decimal numbers. Like the hexadecimal numbers, they too are inputted as character strings and must be surrounded by single quotes.

5. Inputs that are not character strings are not surrounded by single quotes.

6. If the default value is desired a comma must be used to indicate a null input.

7. All of the cards except the 'LOAD' card have a fixed number of fields. Errors will result if the right number of fields is not found.

8. Two cards are required:
   a. 'SYS'
   b. 'END'
   All others are optional, but the simulator will generally use more.

10. The order of the cards is only specified for a few cases. This is because some of the parameters in a card rely on previous inputs on other cards.

11. Underlined inputs are default values.

'SYS' card. (required)

Format:

'SYS',['(NMPU)' / '1'],['NEW' / 'OLD'],['DELETE' / 'KEEP'],'*'

Number of fields: 5.

'(NMPU)' The number of MPUs in the system. Range: 0≤ NMPU ≤20.

'NEW' or 'OLD' System initialization specification. If 'OLD' is specified the system will be initialized from the the file named SIMSYS. The system can be updated in any way when old is specified, including changing memory sizes, number of MPUs, and number of peripheral devices. 'NEW' causes the system to be initialized from inputs only.

'DELETE' or 'KEEP' If 'KEEP' is specified the initial system configuration will be stored on the file SIMSYS after the initialization is complete. However, if an initialization error occurs, the configuration will not be kept. 'DELETE' causes the simulator to not write the initial configuration on the file SIMSYS.

'*' End of card.

'MPU' card. (optional)

Format:

'MPU', ['(MPU number)' '0'], ['(memory size)' '1024'], ['(peripherals)' '2'], '*'

Number of input fields: 5.

'(MPU number)' The number of the MPU to which these inputs are to be applied. If no MPU number is specified, the default is 0.

'(memory size)' The amount of memory to be allocated to this MPU. If this field is left out the default is 1024 bytes of memory.

'(peripherals)' The number of peripherals that are to be initialized for this MPU. This input causes space to be allocated in the tables for the peripherals. The peripherals must be initialized in the 'LOAD' card to be used. The default value is two peripherals.

'*' End of card.

Notes:

1. MPUs are numbered starting a 0.
2. Maximum memory size is 65536 (64K).
3. The last eight locations of memory are automatically addressed as FFF8 to FFFF to simulate the interrupt vector locations.
4. All 'MPU' cards must be read before the 'MEM' card, if used.

'MEM' card. (optional)

Format:

'MEM' ,['CORE' / 'DISK'] ,['NEW' / 'OLD'], '*'

Number of fields: 4.

'NEW' or 'OLD' Specifies if the memory is to be initialized from the file SIMSYS ('OLD'), or if new inputs will be used only ('NEW').

'CORE' or 'DISK' Specifies if main storage ('CORE') or the direct access file MEM1 ('DISK') is to be used to simulate the MPU memory. Main storage provides faster execution times, but large memories may not be simulated. Using the disk file, MEM1, any amount of memory can be simulated, but the execution will be much slower.

'*' end of card.

Notes:

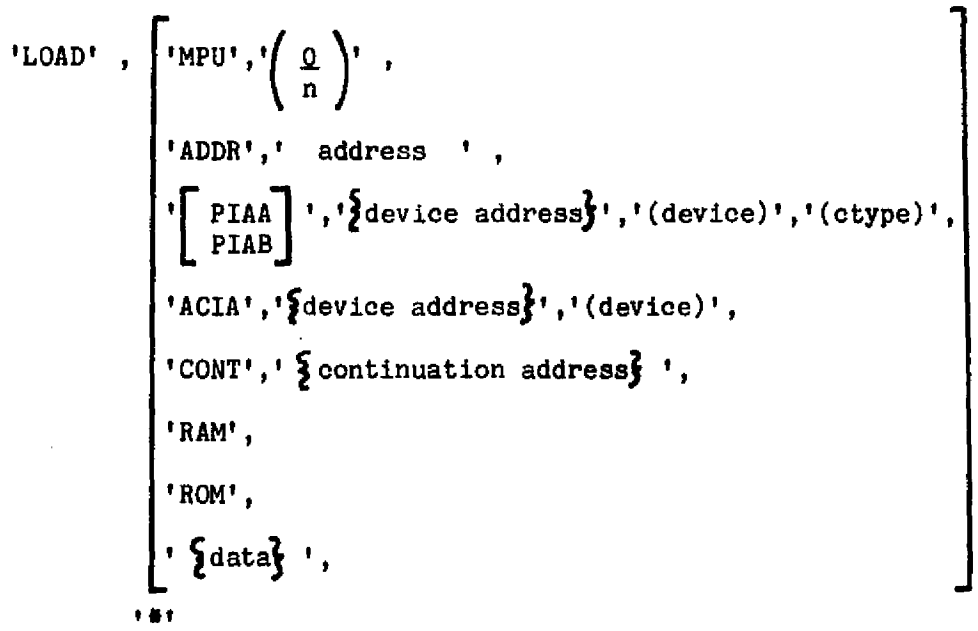
1. The 'MEM' card must follow any 'MPU' cards and precede any 'LOAD' cards in the input stream.

2. Memory is initialized to RAM unless specified as otherwise in the 'LOAD' card.

3. Specifying 'OLD' causes the entire memory to be read in from the file SIMSYS, but changes can be made by using a 'LOAD' card.

'LOAD' card. (optional)

Format:



Minimum number of fields: 2.

'MPU' The current MPU number is to be set to the value of the next input. When a 'MPU' keyword is found the current memory type is reset to RAM and the current address is reset to 0.

'ADDR' The current address is to be set to the value of the next input.

'PIAA' or 'PIAB' The current location is to be initialized as a PIA location. 'PIAA' causes the location to be initialized as an A-side. 'PIAB' causes the location to be initialized as a B-side. When these codes are found, INTLC looks for three more inputs:

'{device address}' This is the address of the device to which the PIA side is connected. The address must be a valid address if the PIA is connected to another MPU system.

'(device)' This is the number of the device to which the PIA side is connected. Device numbers

between 0 and 19 are reserved for MPU devices.

'(ctype)' This is the connection type used for the PIA side. If 0 is specified the devices will be modeled with C1 and C2 of this device connected to C1 and C2 of the associated device, respectively.

If a 1 is found, the PIA side will be modeled with C1 and C2 of this device connected to C2 and C1 of the associate device, respectively.

Both locations that are occupied by the PIA side are automatically initialized. To be consistent with hardware requirements, both sides of the PIA unit must be in two contiguous locations of memory.

'ACIA' The current location is initialized as an ACIA unit. This must be followed by the device address and device number as described for the PIA input key words. Both locations used by the ACIA unit are automatically initialized.

'CONT' The current type of memory is to be continued to the address specified by the next input.

'RAM' All succeeding locations are to be initialized as random access memory. This is the default memory type.

'ROM' All succeeding locations will be initialized as read only memory until a 'RAM' keyword is found.

'{data}' This is a two-digit hexadecimal word which is to be loaded in the current memory address.

'*' End of card.

Notes:

1. After a peripheral device is initialized, the memory returns to the previous type.

2. Default specifications for the 'LOAD' card are:

   Address = 0000.
   MPU = 0.
   Type = RAM.

'START' card. (optional)

Format:

'START' , '(MPU)' , ['RESET' | '{start address}'] , '*'

Number of fields: 4.

'(MPU)' The number of the MPU to which this card applies.

'RESET' The MPU is to be started by a reset at the beginning of the simulation exectuion.

'{start address}' The MPU is to be started at this address at the beginning of the simulation model execution.

'*' End of card.

Notes:

1. If the start card is omitted, an entry must be initialized in the events file (FILE(1)) to start the MPU. Without this the MPU will not be started.

'TRC' card. (optional)

Format:

'TRC , '(MPU)' ,['(trace type)'] , ['{start address}' ,
(start time)

['{stop address}'] , '#'
(stop time)

Number of fields: 6.

'(MPU)' Number of the MPU to which these inputs apply.

'(trace type)' Specifies the type of trace to be performed according to the following code:

0 -trace disabled.
1 -address start and stop.
2 -time start and stop.
3 -branch trace only.
4 -always enabled.

Even if the trace print function is disabled, the simulator will record and print trace entries for instructions which caused MPU errors.

'{start address}' Starting address of the range of addresses to be traced if an address trace is specified.

'{stop address}' Stop address of the range of addresses to be traced if an address trace is specified.

(start time) Time at which the tracing is to be started if a time trace isspecified.

(stop time) Time at which the trace printing is to be stopped if a time trace is specified.

'#' End of card.

Notes:

1. Start and stop addresses or times do not have to be specified if the trace type is 0 or 4. However, two input fields must be found and two commas will accomplish this.

'SIM' card. (optional)

Format:

'SIM' , '(MPU)' , b0...b31B , '*'

Number of fields: 4.

b0 ... b31B — This is a thirty-one bit binary word which specifies control parameters in the simulation according to the following:

- b0=1 -Print a memory dump during the initialization echo check for this MPU.
- b1=1 -Print a memory dump for this MPU during the final summary print.
- b2=1 -Print errors and conditions as they occur during the run.
- b3...b31 -Unused.

A zero in any bit location disables the option. The default value is all zeros.

'*' — End of card.

Notes:

1. Bit five of the control word is used to specify tracing of I/O events only.
2. The option is on when the associated bit set to 1 and off when the it is set to 0.

INPUT ERROR CODES

0	'SYS' card not found.
1	Illegal number of MPUs specified.
2	'*' not found when expected.
3	Illegal card type.
4	Illegal MPU number specified.
5	Illegal memory size specified.
6	Illegal address specified.
7	Illegal number of peripheral units specified.
9	End-of-file found on file SIMSYS.
10	Illegal associate device specified.
11	Illegal system initialization specification found.
12	Illegal system disposition specification.
13	Undefined file condition occurred on file SIMSYS.
14	Illegal input.
15	End-of-file found on file MEM1.
16	'MPU' card found after a 'LOAD' or 'MEM' card.
17	Incorrect file format on file SIMSYS.
18	End-of-file found on file SYSIN.
19	Illegal peripheral associate device.
20	Attempt to initialize an illegal memory address.
21	Illegal memory initialization specification.
23	Illegal trace type specified.
24	NNATR<8. It must be at least 8.
25	Peripheral linked to a nonexistent device.
27	Attempt to initialize more peripherals than specified.
28	Illegal 'CONT' address.
30	Key error occurred on file MEM1.
32	Illegal associate device address.
33	Illegal associate device number.
34	Illegal card type.
35	Attempt to initialize a peripheral when none were specified.

APPENDIX 2: TEAM OPERATING SYSTEM PROGRAMS.

The following is a listing of the programs written for the Team Operating System tests. These programs implement the algorithm discussed in section 3.3. All of the simulation tests were run using these programs.

```
        NAM TOSPROG
NTRID   EQU     $0000     NEXT TASK REGISTER ID
NTRLVL  EQU     $0001     NEXT TASK REGISTER LEVEL
NLVLS   EQU     $0002     NUMBER OF LEVELS
OBSA    EQU     $0700     OUTPUT BUFFER START ADDRESS
IBSA    EQU     $0702     INPUT BUFFER START ADDRESS.
BMSTS   EQU     $0704     BIM STATUS WORD ADDRESS.
CPSTS   EQU     $0705     CP STATUS WORD ADDRESS.
PIA1OA  EQU     $0800     PIA-1 OUTPUT REGISTER A ADDRESS
PIA1CA  EQU     $0801     PIA-1 CONTROL REGISTER A ADDRESS
PIA1OB  EQU     $0802     PIA-1 OUTPUT REGISTER B ADDRESS
PIA1CB  EQU     $0803     PIA-1 CONTROL REGISTER B ADDRESS
STTADD  EQU     $0810     TTBL STARTING ADDRESS WORD ADDRESS.
TTAD    EQU     $0812      TTBL ENTRY ADDRESS WORD.
STQADD  EQU     $0814     TQTBL STARTING ADDRESS WORD.
TQADD   EQU     $0816     TQTBL ENTRY ADDRESS WORD
LTADD   EQU     $0818     LTBL ENTRY ADDRESS WORD.
MYTSK   EQU     $081A      CURRENT CPS JOB WORD
INREG   EQU     $081B     INPUT REGISTER.
GTJOB   EQU     $2000     STARTING ADDRESS OF GTJOB.
        ORG     $0920
START   LDS     #$08FF    INITIALIZE THE SP.
        LDAA    #$FF
        STAA    PIA1OB    SET B-SIDE UP AS OUTPUT
        LDAA    #$25      INITIALIZE PIA.
* * * * * * * * * * * * *
*
*  THE PIA IS INITIALIZED TO THE FOLLOWING:
*
*             A-SIDE:
*                       HANDSHAKE INPUT.
*                       INTERRUPT ON NEGATIVE TRANSITION OF C1.
*
*             B-SIDE:
*                       HANDSHAKE OUTPUT.
*                       INTERRUPTS ARE MASKED.
*
* * * * * * * * * * * * *
        STAA    PIA1CA
        LDAA    #$2C
        STAA    PIA1CB
        LDAA    PIA1OA     CLEAR INTERRUPT FLAGS
        LDAA    PIA1OB
        LDX     #$0900    INITIALIZE MI VECTOR.
        STX     $FFF8
        LDAA    #$05      SEND 'BCh' TO BIM
        STAA    PIA1OB
        CLI               ENABLE INTERRUPTS
```

```
        LDAA    MYTSK       ACCA=MYTSK.
        JSR     TQCALC      CALCULATE TQTBL ADDRESS.
        LDAA    #$01
        BITA    $01,X       OSB='1'?
        BEQ     EOJ         NO, BRANCH TO EOJ.
        JMP     RUN         YES, RUN JOB.
TOS     LDAA    NTRID       ACCA=NEXT TASK
        STAA    MYTSK       GET NEW TASK
        JSR     TQCALC      CALCULATE TQTBL ADDRESS
        STX     TQADD       TQADD=INR
        LDAA    NTRLVL      ACCA=NRTLVL
        LDAB    $00,X       ACCB=SUCCESSOR ID
        CMPB    #$FF        ACCB=FF?
        BEQ     T5          YES, BRANCH TO T2
        JSR     LTCALC      NO, FINISH UPDATE.
        STAB    $01,X       NEXT TASK=SUCCESSOR ID.
T2      CLR     NTRLVL      NTRLVL=0.
        CLRA
T3      JSR     LTCALC      CALCULATE LTBL ADDRESS.
        LDAA    $01,X       ACCA=NEXT TASK.
        STAA    NTRID       NTRID=NEXT TASK
        LDX     TQADD
        LDAA    #$01
        BITA    $01,X       OSB='1'?
        BEQ     TOS         NO, START UPDATE OVER.
        LDAA    #$02        YES,  SEND 'STU'.
        STAA    PIA1OB
RUN     LDAA    #$20        STATUS:  RUNING JOB.
        STAA    CPSTS
        LDAA    MYTSK
        JSR     TTCALC
        LDAA    $00,X       IS JOB IN MEMORY?
        BGE     T4
        JSR     GTJOB       NO, GET JOB FROM MASS MEMORY.
T4      LDX     $01,X       YES, RUN JOB.
        JSR     $00,X
EOJ     LDAA    BMSTS       ACCA=BIM STATUS
        BITA    #$80        IS BIM ACTIVE?
        BNE     EOJ         YES, BRANCH TO EOJ.
        LDAA    #$04        NO,  SEND 'GTC'.
        STAA    PIA1OB
LO      WAI
        LDAA    INREG
        BITA    #$09        'BSC'?
        BNE     TOS
        BRA     LO
T5      JSR     LTCALC      CALCULATE LTBL ADDRESS.
        LDAA    $00,X       ACCA=FIRST TASK.
```

```
        STAA    $01,X       NEXT TASK = FIRST TASK.
        INC     $03,X       LC=LC+1.
        LDAA    $03,X       ACCA=LC.
        CMPA    $02,X       LLN=LC?
        BNE     T2
        CLR     $03,X       LC=0.
        INC     NTRLVL      NTRLVL=NTRLVL+1.
        LDAA    NLVLS
        CMPA    NTRLVL      NLVLS<NTRLVL?
        BGT     T6
        CLR     NTRLVL      YES, NTRLVL=0.
T6      LDAA    NTRLVL      ACCA=NTRLVL.
        BRA     T3
* * * * * * * * * * * * * * * *
*
*  THIS SUBROUTINE CALCULATES A LTBL
*  ADDRESS.
*       INPUT:  LEVEL NUMBER IN ACCA.
*       OUTPUT:  LTBL ADDRESS IN INR.
*
* * * * * * * * * * * * * * * *
LTCALC LDX      #$0003      INR=STARTING LTBL ADDRESS.
LTC1    DECA    ACCA=ACCA-1.
        BLT     LTC2        ACCA<0?
        INX     NO, INR=INR+4.
        INX
        INX
        INX
        BRA     LTC1
LTC2    RTS     YES,RETURN.
* * * * * * * * * * * * * * * *
*
*  THIS SUBROUTINE CALCULATES THE TQTBL
*  ADDRESS WHEN GIVEN A TASK ID NUMBER.
*
*       INPUT:  TASK ID NUMBER IN ACCA.
*       OUTPUT:  TQTBL ADDRESS IN INR.
*
* * * * * * * * * * * * * * * *
TQCALC LDX      STQADD      INR=STARTING TQTBL ADDRESS.
        INX
TQC1    DECA    ACCA=ACCA-1.
        BLT     TQC2        ACCA<0?
        INX     NO, INR=INR+2.
        INX
        BRA     TQC1
TQC2    RTS     YES, RETURN.
* * * * * * * * * * * * * * * *
```

```
*
*  THIS SUBROUTINE CALCULATES THE TTBL
*  ADDRESS FROM THE TASK ID NUMBER.
*
*
*       INPUT:  TASK ID NUMBER IN ACCA.
*       OUTPUT:  TTBL ADDRESS IN INR.
*
* * * * * * * * * * * * * * * * *
TTCALC LDX    STTADD    INR=STARTING TTBL ADDRESS.
       INX
TTC1   DECA              ACCA=ACCA-1.
       BLT    TTC2      ACCA<0?
       INX               NO,INR=INR+3.
       INX
       INX
       BRA    TTC1
TTC2   RTS               YES, RETURN
* * * * * * * * * * * * * * * * * *
*
*
*  THIS IS THE ROUTINE WHICH HANDLES THE MASKABLE
*  INTERRUPTS THAT RESULT FROM INPUTS FROM THE
*  BIM OR OTHER PERIPHERAL DEVICES.
*
*
* * * * * * * * * * * * * * * * * * * *
       ORG    $0900
       LDAA   PIA1CA
       BITA   #$80      IRQA1 SET?
       BEQ    M1        NO, GO TO M1
       LDAA   PIA1OA    YES, READ INPUT
       STAA   INREG
       RTI              STORE IN INREG AND RETURN
M1     RTI
       END
```

APPENDIX 3: SIMULATION PROGRAM LISTING.

The following pages contain the entire listing of the programs written by the author for this project. Twenty-nine programs are included, although the GASP_PL/I programs are not listed here. The GASP_PL/I program package is copyrighted material and the source can be obtained by writing tha address specified in reference 12.

```
/*  BBIN VERSION-1                        10/2/77                */
BBIN:    PROCEDURE(STRING) RETURNS(FIXED BIN(16,0));
/*******************************************************/
/*                                                     */
/*  THIS PROCEDURE CONVERTS HEXADECIMAL STRINGS        */
/*  TO FIXED BINARY.                                   */
/*                                                     */
/*******************************************************/
   DCL STRING CHAR(6);
   DCL STR(0:5) BIT(4);
   DCL C(0:5) CHAR(1);
   DCL TBL(1:16) CHAR(1) INIT('0','1','2','3','4','5','6',
        '7','8','9','A','B','C','D','E','F');
   DCL BTBL(1:16) BIT(4) INIT('0000'B,'0001'B,'0010'B,'0011'B,
        '0100'B,'0101'B,'0110'B,'0111'B,'1000'B,'1001'B,
        '1010'B,'1011'B,'1100'B,'1101'B,'1110'B,'1111'B);
   DCL BNSTR FIXED BIN(16,0);
   DCL (I,J) FIXED DEC(3,0);
/*******************************************************/
/*                                                     */
/* IS THE STRING A LEGAL HEXADECIMAL NUMBER            */
/*                                                     */
/*******************************************************/
   IF VERIFY(STRING,' 0123456789ABCDEF')\=0 THEN BNSTR=0;
   ELSE DO;
        STR='0000'B;
        C=' ';
        DO I=0 TO 5;
             C(I)=SUBSTR(STRING,I+1,1);
             END;
        K=0;
        DO I=5 TO 0 BY -1;
             IF C(I)\=' ' THEN DO J=1 TO 16;
                  IF C(I)=TBL(J) THEN DO;
                       STR(K)=BTBL(J);
                       K=K+1;
                       END;
                  END;
             END;
        BNSTR=BIN(STR(3)||STR(2)||STR(1)||STR(0),16,0);
        END;
   RETURN(BNSTR);
   END BBIN;
```

```
/*     BRANCH     VERSION-1          1/5/78            */
BRANCH: PROCEDURE;
   DCL PC(*) FIXED BIN(16,0)CTL EXT;
   DCL STR BIT(31);
   DCL OPRND1 FIXED BIN(8,0) EXT;
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL MPU FIXED BIN(31,0) EXT;
   DCL 1 STAT(*) CTL EXT,
        2 INST FIXED BIN(31,0),
        2 INPT FIXED BIN(31,0),
        2 OTPT FIXED BIN(31,0),
        2 ERRS FIXED BIN(31,0),
        2 INTS FIXED BIN(31,0),
        2 WARN FIXED BIN(31,0),
        2 BRCH FIXED BIN(31,0),
        2 RTRN FIXED BIN(31,0),
        2 RSTS FIXED BIN(31,0),
        2 HALT FLOAT BIN(31),
        2 RUN FLOAT BIN(31),
        2 RESTRT FLOAT BIN(31),
        2 LSTHLT FLOAT BIN(31);
   IF OPRND1>127 THEN
        PC(MPU)=PC(MPU)+OPRND1-256;
   ELSE PC(MPU)=PC(MPU)+OPRND1;
   IF PC(MPU)<0 THEN PC(MPU)=PC(MPU)+256;
   STR=SYSTBL(8,MPU);
   IF  PC(MPU)>SYSTBL(1,MPU) THEN
        SUBSTR(STR,8,1)='1'B;
   SYSTBL(8,MPU)=STR;
   STR=SYSTBL(7,MPU);
   SUBSTR(STR,7,1)='1'B;
   SYSTBL(7,MPU)=STR;
   BRCH(MPU)=BRCH(MPU)+1;
   RETURN;
   END BRANCH;
```

```
/*    CCRCHK     VERSION-2          1/5/78                */
 CCRCHK: PROCEDURE(R,A,B,CKH,CKI,CKN,CKZ,CKV,CKC);
   DCL (A,B,R)FIXED BIN(8,0);
   DCL (CKH,CK_,CKN,CKZ,CKV,CKC) FIXED DEC(1,0);
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL STR1 BIT(31);
   DCL CCR(*) BIT(8) CTL EXT;
   DCL MPU FIXED BIN(31,0) EXT;
   IF CKH>1 THEN
        IF CKH=3 THEN
             SUBSTR(CCR(MPU),3,1)=(SUBSTR(A,5,1)^SUBSTR(B,5,1))
                  &\SUBSTR(R,5,1)^SUBSTR(A,5,1)&SUBSTR(B,5,1);
        ELSE;
   ELSE DO;
        IF CKH=1 THEN SUBSTR(CCR(MPU),3,1)='1'B;
        ELSE SUBSTR(CCR(MPU),3,1)='0'B;
        END;
   IF CKI<2 THEN
        IF CKI=1 THEN SUBSTR(CCR(MPU),4,1)='1'B;
        ELSE SUBSTR(CCR(MPU),4,1)='0'B;
   ELSE;
   IF CKN>1 THEN
        IF CKN=3 THEN
             IF R>128 THEN SUBSTR(CCR(MPU),5,1)='1'B;
             ELSE SUBSTR(CCR(MPU),5,1)='0'B;
        ELSE;
   ELSE DO;
        IF CKN=1 THEN SUBSTR(CCR(MPU),5,1)='1'B;
        ELSE SUBSTR(CCR(MPU),5,1)='0'B;
        END;
   IF CKZ>1 THEN
        IF CKZ=3 THEN
             IF R=0 THEN SUBSTR(CCR(MPU),6,1)='1'B;
             ELSE SUBSTR(CCR(MPU),6,1)='0'B;
        ELSE;
   ELSE DO;
        IF CKZ=1 THEN SUBSTR(CCR(MPU),6,1)='1'B;
        ELSE SUBSTR(CCR(MPU),6,1)='0'B;
        END;
   IF CKV>1 THEN
        IF CKV=3 THEN DO;
             SUBSTR(CCR(MPU),7,1)=SUBSTR(A,1,1)&SUBSTR(B,1,1)
                  &\SUBSTR(R,1,1)^\SUBSTR(A,1,1)&\SUBSTR(B,1,1)&
                  SUBSTR(R,1,1);
             STR1=SYSTBL(8,MPU);
             SUBSTR(STR1,3,1)=SUBSTR(CCR(MPU),7,1);
             SYSTBL(8,MPU)=STR1;
             END;
```

```
     ELSE;
ELSE DO;
     IF CKV=1 THEN SUBSTR(CCR(MPU),7,1)='1'B;
     ELSE SUBSTR(CCR(MPU),7,1)='0'B;
     IF CKV=1 THEN DO;
          STR1=SYSTBL(8,MPU);
          SUBSTR(STR1,3,1)='1'B;
          SYSTBL(8,MPU)=STR1;
          END;
     END;
IF CKC>1 THEN
     IF CKC=3 THEN
          SUBSTR(CCR(MPU),8,1)=SUBSTR(A,1,1)&SUBSTR(B,1,1)^
               (SUBSTR(A,1,1)^SUBSTR(B,1,1))&
               \SUBSTR(R,1,1);
     ELSE;
ELSE DO;
     IF CKC=1 THEN SUBSTR(CCR(MPU),8,1)='1'B;
     ELSE SUBSTR(CCR(MPU),8,1)='0'B;
     END;
CKH,CK1,CKN,CKV,CKZ,CKC=0;
RETURN;
END CCRCHK;
```

```
/*    CONPRT      VERSION-1                                          */
CONPRT: PROCEDURE(STRING);
/*****************************************************/
/*                                                   */
/*  THIS SUBPROGRAM PRINTS CONDITION DIAGNOSTICS     */
/*  ON THE MPUS DURING EXECUTION.                    */
/*                                                   */
/*****************************************************/
   DCL STRING BIT(31);
   DCL MESSAGE(31)  CHAR(50) VARYING INIT(
        'NON-MASKABLE INTERRUPT.',
        'MASKABLE INTERRUPT.',
        'RESET','PROCESSOR HALTED',
        'SOFTWARE INTERRUPT','WAIT-FOR-INTERRUPT',
       'PROGRAM BRANCH ',
        'RETURN-FROM-INTERRUPT ','RETURN-FROM-SUBROUTINE ',
        'DATA OUTPUT','DATA INPUT','RESET PENDING',
        'SUBROUTINE BRANCH');
   DCL I FIXED BIN(31,0);
   DO I=1 TO 31;
        IF SUBSTR(STRING,I,1)='1'B THEN DO;
             PUT SKIP EDIT((8)'*'^^' '^^MESSAGE(I))(A);
             END;
        END;
   END CONPRT;
```

```
/*  ECHO VERSION-3                              3/25/78       */
ECHO:  PROCEDURE;
/*****************************************************************/
/*                                                               */
/*    THIS PROGRAM PRINTS THE INITIAL CONFIGURATION              */
/*    OF THE SYSTEM AFTER INTLC IS FINISHED.                     */
/*    USED WITH INTLC VERSION-6.                                 */
/*                                                               */
/*****************************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (I,MAX) FIXED BIN(31,0);
   DCL (MPU,NMPU,NPER,PL) FIXED BIN(31,0) EXT;
   DCL (MDUMP,PDUMP) ENTRY;
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL TYPE(0:4) CHAR(10) INIT('  DISABLED',' ADDRESS ','    TIME  ',
       '   BRANCH  ',' ENABLED');
   DCL TRCE(*,*) FLOAT BIN(31) CTL EXT;
   IF LINENO(SYSPRINT)>PL-25 THEN PUT EDIT(' ')(LINE(PL-1),A);
   PUT SKIP(3) EDIT('INITIAL SYSTEM CONFIGURATION')
        (COL(16),A);
   PUT EDIT((28)'*')(COL(16),A);
   IF ALLOCATION(SYSTBL)='1'B THEN DO;
        DO I=0 TO NMPU-1 BY 4;
             IF NMPU-1<I+4 THEN MAX=NMPU-1;
             ELSE MAX=I+3;
             IF LINENO(SYSPRINT)>PL-15 THEN PUT EDIT(' ')(LINE(1),A);
             PUT SKIP(2) EDIT('MPU NUMBE..',(MPU DO MPU=I TO MAX))
                  (COL(2),A,COL(18),F(3,0),(3)(X(9),F(3,0)));
             PUT SKIP EDIT((60)'=')(A);
             PUT SKIP EDIT('MEMORY SIZE',((SYSTBL(1,MPU)+1)
                  DO MPU=I TO MAX))(COL(2),A,COL(17),F(6,0),
                  (3)(X(6),F(6,0)));
             PUT SKIP EDIT('PERIPHERALS',(SYSTBL(6,MPU)
                  DO MPU=I TO MAX))(COL(2),A,COL(18),F(3,0),
                  (3)(X(9),F(3,0)));
             PUT SKIP EDIT('TRACE TYPE',(TYPE(TRCE(1,MPU))
                  DO MPU=I TO MAX))(COL(2),A,COL(13),
                  (4)(X(2),A(10)));
             PUT SKIP EDIT('TRACE START')(COL(2),A(12));
             DO MPU=I TO MAX;
                  IF TRCE(1,MPU)=0 ^ TRCE(1,MPU)>2
                       THEN PUT EDIT((12)' ')(A(12));
                  IF TRCE(1,MPU)=1 THEN PUT EDIT(' '^^
                       HHEX(TRCE(2,MPU)))(X(2),A(10));
                  IF TRCE(1,MPU)=2 THEN PUT EDIT(TRCE(2,MPU))
                       (X(2),F(10,6));
                  END;
```

```
          PUT SKIP EDIT('TRACE STOP')(COL(2),A(12));
          DO MPU=I TO MAX;
               IF TRCE(1,MPU)=0 ^ TRCE(1,MPU)>2
                    THEN PUT EDIT((i2)' ')(A(12));
               IF TRCE(1,MPU)=1 THEN PUT EDIT(' '^^
                    HHEX(TRCE(3,MPU)))(X(2),A(10));
               IF TRCE(1,MPU)=2 THEN PUT EDIT(TRCE(3,MPU))
                    (X(2),F(10,6));
               END;
          PUT SKIP EDIT('TRACE FILE',((MPU+3) DO MPU=I TO MAX))
               (COL(2),A,COL(18),F(3,0),(3)(X(9),F(3,0)));
          PUT SKIP EDIT((60)'=')(A);
          END;
     PUT SKIP(2) EDIT('SIMULATION CONTROL SPECIFICATIONS:')
          (COL(10),A);
     PUT EDIT((I,SYSTBL(0,I)  DO I=0 TO NMPU-1))
          ((NMPU)(SKIP,COL(15),F(3,0),X(3),B(31)));
     CALL PDUMP;
     DO MPU=0 TO NMPU-1;
          IF SUBSTR(SYSTBL(0,MPU),1,1)='1'B THEN CALL MDUMP;
          END;
     END;
ELSE PUT SKIP(2) EDIT('SYSTEM INITIALIZATION NOT PERFORMED---',
     'SYSTEM TABLE NOT ALLOCATED.')(COL(10),A);
RETURN;
END ECHO;
```

```
/*    ERR      VERSION-2            1/25/78                    */
ERR:  PROCEDURE;
/*****************************************************************/
/*                                                               */
/*  THIS PROCEDURE CALLS THE PROCEDURES ERRPRT AND CONPRT        */
/*  TO PRINT DIAGNOSTICS FOR RUN TIME ERRORS AND CONDITIONS.     */
/*                                                               */
/*****************************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (CYCLES,MPU,PL) FIXED BIN(31,0) EXT;
   DCL EOI FLOAT BIN(31);
   DCL (STR1,STR2,STR3) BIT(31);
   DCL FLAG BIT(1)INIT(0);
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL 1 STAT(*) CTL EXT,
        2 INST FIXED BIN(31,0),
        2 INPT FIXED BIN(31,0),
        2 OTPT FIXED BIN(31,0),
        2 ERRS FIXED BIN(31,0),
        2 INTS FIXED BIN(31,0),
        2 WARN FIXED BIN(31,0),
        2 BRCH FIXED BIN(31,0),
        2 RTRN FIXED BIN(31,0),
        2 RSTS FIXED BIN(31,0),
        2 HALT FLOAT BIN(31),
        2 RUN FLOAT BIN(31),
        2 RESTRT FLOAT BIN(31),
        2 LSTHLT FLOAT BIN(31);
   DCL CONPRT ENTRY(BIT(31));
   DCL ERRPRT ENTRY(BIT(31));
   DCL I FIXED BIN(31,0);
   DCL TNOW FLOAT BIN EXT;
   DCL OPC FIXED BIN(16,0) EXT;
   STR1=SYSTBL(8,MPU);
   STR2=SYSTBL(7,MPU);
   STR3=SYSTBL(0,MPU);
   FLAG=SUBSTR(STR3,3,1);
   EOI=TNOW+0.000001*CYCLES;
   IF FLAG='1'B & SYSTBL(8,MPU)\=0  THEN DO;
/*****************************************************/
/*                                                   */
/*  PRINT ERROR DIAGNOSTICS                          */
/*                                                   */
/*****************************************************/
        PUT SKIP(2) EDIT((8)'*'^^'  MPU ',MPU,
             ' *****  ERROR AT TIME=',EOI,'SEC. ADDRESS='^^HHEX(OPC))
             (A,F(2,0),X(2),A,F(10,6),X(2),A);
        CALL ERRPRT(STR1);
```

```
        END;
        IF (SUBSTR(STR1,1,1)='1'B ^ SUBSTR(STR1,8,1)='1'B) &
             SUBSTR(STR2,3,1)='0'B THEN SUBSTR(STR2,4,1)='1'B;
   SYSTBL(7,MPU)=BIN(STR2,31,0);
   IF FLAG='1'B & STR2\=(31)'0'B THEN DO;
/*****************************************************/
/*                                                   */
/*  PRINT CONDITION DIAGNOSTICS.                     */
/*                                                   */
/*****************************************************/
       PUT SKIP(2) EDIT('******** MPU ',MPU,' ***** TIME=',
            EOI,'SEC.  ADDRESS='^^HHEX(OPC))
             (A,F(2,0),X(2),A,F(10,6),X(2),A);
        CALL CONPRT(STR2);
        END;
/*****************************************************/
/*                                                   */
/*   RECORD STATISITCS FOR THIS INSTRUCTION.         */
/*                                                   */
/*****************************************************/
   DO I=1 TO 31;
        IF SUBSTR(STR1,I,1)='1'B THEN DO;
             IF I=1 ^ I=8 THEN ERRS(MPU)=ERRS(MPU)+1;
             ELSE WARN(MPU)=WARN(MPU)+1;
             END;
        IF SUBSTR(STR2,I,1)='1'B THEN DO;
             IF I=1 ^ I=2 ^ I=5 ^ I=6 THEN INTS(MPU)=INTS(MPU)+1;
             IF I=3 THEN RSTS(MPU)=RSTS(MPU)+1;
             IF I=8 ^ I=9 THEN RTRN(MPU)=RTRN(MPU)+1;
             IF I=10 THEN OTPT(MPU)=OTPT(MPU)+1;
             IF I=11 THEN INPT(MPU)=INPT(MPU)+1;
             END;
        END;
   IF LINENO(SYSPRINT)>PL-5 THEN
        IF LINENO(SYSPRINT)<PL-1 THEN PUT EDIT(' ')(LINE(PL-1),A);
   RETURN;
   END ERR;
```

```
/*     ERRPRT    VERSION-1          1/23/78                    */
ERRPRT:  PROCEDURE(STRING);
/*                                                             */
/*  THIS PROCEDURE PRINTS ERROR DIAGNOSTICS ON ERRORS THAT     */
/*  ARE FLAGGED IN SYSTBL(5,MPU).                              */
/*                                                             */
   DCL STRING BIT(31);
   DCL MESSAGE(31)  CHAR(50) VARYING INIT(
        'ILLEGAL OPCODE (FATAL)',
        'NON-EXISTENT MEMORY ACCESSED.  (WARNING)',
        'OVERFLOW OCCURRED.  (WARNING)',
        ' ','ATTEMPT TO WRITE ON ROM (WARNING)',
        'BAD PERIPHERAL DEVICE ACCESSED (WARNING)',
        'BAD MEMORY LOCATION ACCESSED.  (WARNING)',
        'BRANCH TO NON-EXISTENT MEMORY.  (FATAL)');
   DCL I FIXED BIN(31,0);
   DO I=1 TO 31;
        IF SUBSTR(STRING,I,1)='1'B THEN DO;
             PUT SKIP EDIT((8)'*'^^' '^^MESSAGE(I))
                  (A);
             END;
        END;
   END ERRPRT;
```

```
/*   EVNTS    VERSION-3                    1/4/78          */
EVNTS:   PROCEDURE(TYPE);
   DCL TNOW BIN FLOAT EXTERNAL;
   DCL TYPE FIXED BIN(31,0);
   DCL (CYCLES,MPU,NMPU) FIXED BIN(31,0) EXT;
   DCL ATRIB(*) BIN FLOAT CTL EXT;
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL INSTR ENTRY;
   DCL READ ENTRY(FIXED BIN(16,0))RETURNS(FIXED BIN(8,0));
   DCL PC(*) FIXED BIN(16,0) CTL EXT;
   DCL OPC FIXED BIN(16,0) EXT;
   DCL TRACE ENTRY;
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (CODE,OPRND1,OPRND2) FIXED BIN(8,0) EXT;
   DCL (ABA,ADC,ADDD,AND,ASL,ASR,BCC,BCS,BEQ,BGE,BGT,BHI,
        BITT,BLE,BLS,BLT,BMI,BNE,BPL,BRA,BSR,BVC,
        BVS,CLC,CLI,CLR,CLV,CMP,COM,CPX,DAA,DES,
        DEX,DECC,EOR,INC,INS,INX,JMP,JSR,LDA,
        LDR,LSR,NEG,NOP,ORA,PSH,PUL,
        ROL,ROR,RTI,RTS,SBA,SBC,SEC,SEI,SEV,STA,
        STR,SUB,SWI,TAB,TAP,TST,TSX) ENTRY;
   DCL FLAG BIT(1);
   DCL ERR ENTRY;
   DCL INSCHD ENTRY(FIXED BIN(31,0));
   DCL (INCHK,IOEVNT) ENTRY;
   DCL COLCT ENTRY(FLOAT BIN,FIXED BIN(31,0));
   DCL MPHLT ENTRY(FIXED BIN(31,0));
   CODE=0;
   CYCLES=-1;
   FLAG='0'B;
   OPRND1,OPRND2=-1;
   IF TYPE>=10 & TYPE<=NMPU+10  THEN DO;
        MPU=TYPE-10;
        OPC=PC(MPU);
        SYSTBL(7,MPU)=BOOL(SYSTBL(7,MPU),
             '000100000001'B^^(19)'0'B,'0001'B);
        SYSTBL(8,MPU)=0;
        FLAG=SUBSTR(SYSTBL(7,MPU),4,1);
        IF FLAG='0'B THEN CODE=READ(PC(MPU));
        END;
   IF TYPE>99 &TYPE<120 THEN CALL MPHLT(TYPE);
   IF TYPE=300 THEN CALL IOEVNT;
   IF TYPE>=1000 & TYPE<1300 THEN
        CALL INSCHD(TYPE);
   IF CODE>0 & CODE<32 THEN DO;
        IF CODE=1 THEN CALL NOP;
        IF CODE=6 ^ CODE=7 THEN CALL TAP;
        IF CODE=8 THEN CALL INX;
```

```
    IF CODE=9 THEN CALL DEX;
    IF CODE=10 THEN CALL CLV;
    IF CODE=11 THEN CALL SEV;
    IF CODE=12 THEN CALL CLC;
    IF CODE=13 THEN CALL SEC;
    IF CODE=14 THEN CALL CLI;
    IF CODE=15 THEN CALL SEI;
    IF CODE=16 ^ CODE=17 THEN CALL SBA;
    IF CODE=22 ^ CODE=23 THEN CALL TAB;
    IF CODE=25 THEN CALL DAA;
    IF CODE=27 THEN CALL ABA;
    END;
IF CODE>31 & CODE<48 THEN DO;
    IF CODE=32 THEN CALL BRA;
    IF CODE=34 THEN CALL BHI;
    IF CODE=35 THEN CALL BLS;
    IF CODE=36 THEN CALL BCC;
    IF CODE=37 THEN CALL BCS;
    IF CODE=38 THEN CALL BNE;
    IF CODE=39 THEN CALL BEQ;
    IF CODE=40 THEN CALL BVC;
    IF CODE=41 THEN CALL BVS;
    IF CODE=42 THEN CALL BPL;
    IF CODE=43 THEN CALL BMI;
    IF CODE=44 THEN CALL BGE;
    IF CODE=45 THEN CALL BLT;
    IF CODE=46 THEN CALL BGT;
    IF CODE=47 THEN CALL BLE;
    END;
IF CODE>47 & CODE<64 THEN DO;
    IF CODE=48 ^ CODE=53 THEN CALL TSX;
    IF CODE=49 THEN CALL INS;
    IF CODE=50 ^ CODE=51 THEN CALL PUL;
    IF CODE=52 THEN CALL DES;
    IF CODE=54 ^ CODE=55 THEN CALL PSH;
    IF CODE=57 THEN CALL RTS;
    IF CODE=59 THEN CALL RTI;
    IF CODE=63 ^ CODE=62  THEN CALL SWI;
    END;
IF CODE>63 & CODE<128 THEN DO;
    IF CODE=64 ^ CODE=80 ^ CODE=96 ^ CODE=112 THEN CALL NEG;
    IF CODE=67 ^ CODE=83 ^ CODE=99 ^ CODE=115 THEN CALL COM;
    IF CODE=68 ^ CODE=84 ^ CODE=100 ^ CODE=116 THEN CALL LSR;
    IF CODE=70 ^ CODE=86 ^ CODE=102 ^ CODE=118 THEN CALL ROR;
    IF CODE=71 ^ CODE=87 ^ CODE=103 ^ CODE=119 THEN CALL ASR;
    IF CODE=72 ^ CODE=88 ^ CODE=104 ^ CODE=120 THEN CALL ASL;
    IF CODE=73 ^ CODE=89 ^ CODE=105 ^ CODE=121 THEN CALL ROL;
    IF CODE=74 ^ CODE=90 ^ CODE=106 ^ CODE=122 THEN CALL DECC;
```

```
    IF CODE=76 ^ CODE=92 ^ CODE=108 ^ CODE=124 THEN CALL INC;
    IF CODE=77 ^ CODE=93 ^ CODE=109 ^ CODE=125 THEN CALL TST;
    IF CODE=110 ^ CODE=126 THEN CALL JMP;
    IF CODE=79 ^ CODE=95 ^ CODE=111 ^ CODE=127 THEN CALL CLR;
    END;
IF CODE>127 ^ CODE<256 THEN DO;
    IF CODE=128 ^ CODE=144 ^ CODE=160 ^ CODE=176 ^
       CODE=192 ^ CODE=208 ^ CODE=224 ^ CODE=240
            THEN CALL SUB;
    IF CODE=129 ^ CODE=145 ^  CODE=161 ^ CODE=177 ^
       CODE=193 ^ CODE=209 ^ CODE=225 ^ CODE=241
            THEN CALL CMP;
    IF CODE=130 ^ CODE=146 ^ CODE=162 ^ CODE=178 ^
       CODE=194 ^ CODE=210 ^ CODE=226 ^ CODE=242
            THEN CALL SBC;
    IF CODE=132 ^ CODE=148 ^ CODE=164 ^ CODE=180 ^
       CODE=196 ^ CODE=212 ^ CODE=228 ^ CODE=244
            THEN CALL AND;
    IF CODE=133 ^ CODE=149 ^ CODE=165 ^ CODE=181 ^
       CODE=197 ^ CODE=213 ^ CODE=229 ^ CODE=245
            THEN CALL BITT;
    IF CODE=134 ^ CODE=150 ^ CODE=166 ^ CODE=182 ^
       CODE=198 ^ CODE=214 ^ CODE=230 ^ CODE=246
            THEN CALL LDA;
    IF CODE=151 ^ CODE=167 ^ CODE=183 ^
       CODE=215 ^ CODE=231 ^ CODE=247
            THEN CALL STA;
    IF CODE=136 ^ CODE=152 ^ CODE=168 ^ CODE=184 ^
       CODE=200 ^ CODE=216 ^ CODE=232 ^ CODE=248
            THEN CALL EOR;
    IF CODE=137 ^ CODE=153 ^ CODE=169 ^ CODE=185 ^
       CODE=201 ^ CODE=217 ^ CODE=233 ^ CODE=249
            THEN CALL ADC;
    IF CODE=138 ^ CODE=154 ^ CODE=170 ^ CODE=186 ^
       CODE=202 ^ CODE=218 ^ CODE=234 ^ CODE=250
            THEN CALL ORA;
    IF CODE=139 ^ CODE=155 ^ CODE=171 ^ CODE=187 ^
       CODE=203 ^ CODE=219 ^ CODE=235 ^ CODE=251
            THEN CALL ADDD;
    IF CODE=140 ^ CODE=156 ^ CODE=172 ^ CODE=188
            THEN CALL CPX;
    IF CODE=141 THEN CALL BSR;
    IF CODE=142 ^ CODE=158 ^ CODE=174 ^ CODE=190 ^
       CODE=206 ^ CODE=222 ^ CODE=238 ^ CODE=254
            THEN CALL LDR;
    IF CODE=159 ^ CODE=175 ^ CODE=191 ^
       CODE=223 ^ CODE=239 ^ CODE=255
            THEN CALL STR;
```

```
     IF CODE=173 ^ CODE=189 THEN CALL JSR;
     END;
IF TYPE>=10 & TYPE<30 & FLAG='0'B THEN DO;
     IF CYCLES=-1 THEN SYSTBL(8,TYPE-10)=BOOL(SYSTBL(8,TYPE-10),
          '1'B^^(30)'0'B,'0111'B);
     CALL INCHK;
     CALL TRACE;
     CALL ERR;
     FLAG=SUBSTR(SYSTBL(7,MPU),4,1);
     IF FLAG='1'B THEN DO;
          ATRIB(2)=MPU+100;
          SYSTBL(4,MPU)=1;
          END;
     ELSE ATRIB(2)=TYPE;
     IF CYCLES>=0 THEN DO;
          CALL COLCT(CYCLES,MPU+1);
          ATRIB(1)=TNOW+CYCLES*0.000001;
          END;
     ELSE ATRIB(1)=TNOW+0.000001;
     CALL FILEM(1);
     END;
RETURN;
END EVNTS;
```

```
/*  HHEX  VERSION-1                          10/2/77          */
HHEX:    PROCEDURE(BNSTR) RETURNS(CHAR(6));
/*******************************************************/
/*                                                     */
/*  THIS PROGRAM CONVERTS THE BINARY INPUT TO          */
/*  HEXADECIMAL....                                    */
/*                                                     */
/*******************************************************/
   DCL BNSTR FIXED BIN(16,0);
   DCL (I,J) FIXED BIN(31,0);
   DCL STR1 CHAR(6);
   DCL C(0:4) CHAR(1);
   DCL TBL(16) CHAR(1) INIT('0','1','2','3','4','5','6','7',
        '8','9','A','B','C','D','E','F');
   DCL BTBL(16) BIT(4) INIT('0000'B,'0001'B,'0010'B,'0011'B,
        '0100'B,'0101'B,'0110'B,'0111'B,'1000'B,'1001'B,
        '1010'B,'1011'B,'1100'B,'1101'B,'1110'B,'1111'B);
   DCL BSTR(4) BIT(4);
   BNSTR=ABS(BNSTR);
   C(0)=' ';
   DO I=1 TO 4;
        BSTR(I)=SUBSTR(BNSTR,4*I-3,4);
        END;
   DO I=1 TO 4;
        DO J=1 TO 16;
             IF BTBL(J)=BSTR(I) THEN C(I)=TBL(J);
             END;
        END;
   STR1=C(0)||C(1)||C(2)||C(3)||C(4);
   RETURN(STR1);
   END HHEX;
```

```
/*   INCHK    VERSION-1                    1/5/78       */
INCHK:  PROCEDURE;
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE DOES THE INTERRUPT SERVICE CHECK  */
/*  AND PROGRAM CONTROL BRANCH IF AN INTERRUPT IS    */
/*  PENDING.                                         */
/*                                                   */
/*****************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (CYCLES,MPU) FIXED BIN(31,0) EXT;
   DCL TPC FIXED BIN(16,0);
   DCL (SP(*),PC(*),INR(*)) FIXED BIN(16,0) CTL EXT;
   DCL STRING BIT(31);
   DCL (ACCA(*),ACCB(*)) FIXED BIN(8,0) CTL EXT;
   DCL CCR(*) BIT(8) CTL EXT;
   DCL READ ENTRY(FIXED BIN(16,0)) RETURNS(FIXED BIN(8,0));
   DCL WRITE ENTRY(FIXED BIN(8,0),FIXED BIN(16,0));
/*****************************************************/
/*                                                   */
/*   CHECK FOR INTERRUPTS PENDING                    */
/*                                                   */
/*****************************************************/
   IF SYSTBL(10,MPU)>0 ^ SYSTBL(9,MPU)>0 THEN DO;
        IF SYSTBL(10,MPU)>0 THEN DO;
/*****************************************************/
/*                                                   */
/*  NON-MASKABLE INTERRUPT PENDING                   */
/*                                                   */
/*****************************************************/
             SYSTBL(10,MPU)=SYSTBL(10,MPU)-1;
             TPC=READ(BIN(65532,16,0))*256+
                  READ(BIN(65533,16,0));
             STRING=SYSTBL(7,MPU);
             SUBSTR(STRING,1,1)='1'B;
             SYSTBL(7,MPU)=STRING;
             END;
        ELSE DO;
/*****************************************************/
/*                                                   */
/*  MASKABLE INTERRUPT PENDING                       */
/*                                                   */
/*****************************************************/
             IF SUBSTR(CCR(MPU),4,1)='1'B THEN RETURN;
             SYSTBL(9,MPU)=SYSTBL(9,MPU)-1;
             TPC=READ(BIN(65528,16,0))*256+
                  READ(BIN(65529,16,0));
             STRING=SYSTBL(7,MPU);
```

```
            SUBSTR(STRING,2,1)='1'B;
            SYSTBL(7,MPU)=STRING;
            END;
/*****************************************************/
/*                                                   */
/*  STACK REGISTER CONTENTS                          */
/*                                                   */
/*****************************************************/
        IF SUBSTR(STRING,6,1)='0'B THEN DO;
            CALL WRITE(BIN(SUBSTR(PC(MPU),9,8),8,0),SP(MPU));
            SP(MPU)=SP(MPU)-1;
            CALL WRITE(BIN(SUBSTR(PC(MPU),1,8),8,0),SP(MPU));
            SP(MPU)=SP(MPU)-1;
            CALL WRITE(BIN(SUBSTR(INR(MPU),9,8),8,0),SP(MPU));
            SP(MPU)=SP(MPU)-1;
            CALL WRITE(BIN(SUBSTR(INR(MPU),1,8),8,0),SP(MPU));
            SP(MPU)=SP(MPU)-1;
            CALL WRITE(ACCA(MPU),SP(MPU));
            SP(MPU)=SP(MPU)-1;
            CALL WRITE(ACCB(MPU),SP(MPU));
            SP(MPU)=SP(MPU)-1;
            CALL WRITE(BIN(CCR(MPU),8,0),SP(MPU));
            SP(MPU)=SP(MPU)-1;
            CYCLES=CYCLES+12;
            END;
        SUBSTR(CCR(MPU),4,1)='1'B;
        PC(MPU)=TPC;
        END;
   STRING=BIT(SYSTBL(7,MPU),31);
   IF SUBSTR(STRING,12,1)='1'B THEN DO;
/*****************************************************/
/*                                                   */
/*   RESET OCCURRED                                  */
/*                                                   */
/*****************************************************/
        SUBSTR(STRING,12,1)='0'B;
        SUBSTR(STRING,3,2)='10'B;
        SYSTBL(7,MPU)=BIN(STRING,31,0);
        END;
   RETURN;
   END INCHK;
```

```
/*    INSCHD    VERSION-1                1/4/78          */
INSCHD:  PROCEDURE(CODE);
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE SCHEDULES                         */
/*  INTERRUPTS ACCORDING TO THE FOLLOWING CODE:      */
/*                                                   */
/*       CODE             INTERRUPT TYPE             */
/*      10XX               MASKABLE                  */
/*      11XX               NON-MASKABLE              */
/*      12XX               RESET                     */
/*                                                   */
/*  WHERE "XX" IS THE MPU NUMBER                     */
/*                                                   */
/*****************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL PC(*) FIXED BIN(16,0) CTL EXT;
   DCL CCR(*) BIT(8) CTL EXT;
   DCL 1 STAT(*) CTL EXT,
        2 INST FIXED BIN(31,0),
        2 INPT FIXED BIN(31,0),
        2 OTPT FIXED BIN(31,0),
        2 ERRS FIXED BIN(31,0),
        2 INTS FIXED BIN(31,0),
        2 WARN FIXED BIN(31,0),
        2 BRCH FIXED BIN(31,0),
        2 RTRN FIXED BIN(31,0),
        2 RSTS FIXED BIN(31,0),
        2 HALT FLOAT BIN(31),
        2 RUN FLOAT BIN(31),
        2 RESTRT FLOAT BIN(31),
        2 LSTHLT FLOAT BIN(31);
   DCL CODE FIXED BIN(31,0);
   DCL STRING BIT(31);
   DCL FLAG BIT(1);
   DCL NXVNT PTR;
   DCL NULL BUILTIN;
   DCL FIND ENTRY(FLOAT BIN,FIXED BIN(31,0),PTR,FIXED BIN(31,0),
        FLOAT BIN) RETURNS(PTR);
   DCL RMOVE ENTRY(PTR,FIXED BIN(31,0));
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL ERROR ENTRY(FIXED BIN(31,0));
   DCL READ ENTRY(FIXED BIN(16,0)) RETURNS(FIXED BIN(8,0));
   DCL INCHK ENTRY;
   DCL (MPU,NMPU) FIXED BIN(31,0) EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL MFE(*) PTR CTL EXT;
   DCL TNOW FLOAT BIN EXT;
```

```
   IF CODE<1200 THEN
        IF CODE<1100 THEN DO;
             MPU=CODE-1000;
             CODE=1000;
             END;
        ELSE DO;
             MPU=CODE-1100;
             CODE=1100;
             END;
   ELSE DO;
        MPU=CODE-1200;
        CODE=1200;
        END;
   IF MPU<0 ^ MPU>=NMPU THEN DO;
        CALL ERROR(1);
        CODE=0;
        END;
/*******************************************************/
/*                                                     */
/* RUN ERROR(01)---- ILLLEGAL MPU NUMBER SPECIFIED     */
/*                                                     */
/*******************************************************/
/*                                                        */
   STRING=BIT(SYSTBL(7,MPU),31);
   IF CODE<1200 THEN DO;
        IF CODE=1000 THEN SYSTBL(9,MPU)=SYSTBL(9,MPU)+1;
        ELSE SYSTBL(10,MPU)=SYSTBL(10,MPU)+1;
        IF SYSTBL(4,MPU)=0 &
             SUBSTR(STRING,4,1)='1'B & SUBSTR(STRING,6,1)='1'B
             THEN DO;
             CALL INCHK;
             STRING=BIT(SYSTBL(7,MPU),31);
             IF SUBSTR(STRING,1,1)='1'B ^ SUBSTR(STRING,2,1)='1'B
                  THEN DO;
                  HALT(MPU)=HALT(MPU)+TNOW-LSTHLT(MPU);
                  RESTRT(MPU)=TNOW;
                  ATRIB(1)=TNOW+0.000003;
                  ATRIB(2)=MPU+10;
                  CALL FILEM(1);
                  SUBSTR(STRING,4,1)='0'B;
                  END;
             END;
        ELSE IF SYSTBL(4,MPU)=1 THEN DO;
             ATRIB(1)=TNOW+0.000004;
             ATRIB(2)=CODE+MPU;
             CALL FILEM(1);
             END;
        END;
```

```
IF CODE=1200 THEN DO;
     FLAG='0'B;
     IF SUBSTR(STRING,4,1)='1'B THEN DO;
          DO WHILE(FLAG='0'B);
               NXVNT=FIND(10+MPU,5,MFE(1),2,0);
               IF NXVNT=NULL THEN FLAG='1'B;
               ELSE CALL RMOVE(NXVNT,1);
               END;
          HALT(MPU)=HALT(MPU)+TNOW-LSTHLT(MPU);
          RESTRT(MPU)=TNOW;
          ATRIB(1)=TNOW;
          ATRIB(2)=MPU+10;
          SUBSTR(STRING,4,1)='0'B;
          END;
     ELSE DO;
          NXVNT=FIND(MPU+10,5,MFE(1),2,0);
          IF NXVNT\=NULL THEN CALL RMOVE(NXVNT,1);
          ELSE DO;
               ATRIB(1)=TNOW;
               ATRIB(2)=MPU+10;
               END;
          END;
     PC(MPU)=READ(65534)*256+READ(65535);
     SUBSTR(CCR(MPU),4,1)='1'B;
     SUBSTR(STRING,12,1)='1'B;
     SUBSTR(STRING,4,1)='0'B;
     ATRIB(1)=ATRIB(1)+0.000004;
     CALL FILEM(1);
     END;
SYSTBL(7,MPU)=BIN(STRING,31,0);
RETURN;
END INSCHD;
```

```
/*  INSTR   VERSION-3                10/4/77                */
INSTR:  PROCEDURE;
        DCL (ACCA(*),ACCB(*)) FIXED BIN(8,0) CTL EXT;
        DCL (PC(*),SP(*),INR(*)) FIXED BIN(16,0) CTL EXT;
        DCL CCR(*) BIT(8) CTL EXT;
        DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
        DCL WRITE ENTRY(FIXED BIN(8,0),FIXED BIN(16,0));
        DCL NEXT FIXED BIN(31,0);
        DCL ONLOC BUILTIN;
        DCL (CYCLES,MPU) FIXED BIN(31,0) EXT;
        DCL CCRCHK ENTRY(FIXED BIN(8,0),FIXED BIN(8,0),FIXED BIN(8,0),
             FIXED DEC(1,0),FIXED DEC(1,0),FIXED DEC(1,0),
             FIXED DEC(1,0),FIXED DEC(1,0),FIXED DEC(1,0));
        DCL ADDRESS FIXED BIN(16,0);
        DCL BRANCH ENTRY;
        DCL (CKH,CKI,CKN,CKV,CKZ,CKC) FIXED DEC(1,0);
        DCL STR1 BIT(31);
        DCL (TEMP,LHB,MHB,RESULT,TEMP2) FIXED BIN(8,0);
        DCL (CODE,OPRND1,OPRND2) FIXED BIN(8,0) EXT;
        DCL STRING BIT (8);
        DCL READ ENTRY(FIXED BIN(16,0)) RETURNS(FIXED BIN(8,0));
        DCL SWIH FIXED BIN(16,0);
NOP:    ENTRY;
        CYCLES=2;
        PC(MPU)=PC(MPU)+1;
        RETURN;
TAP:    ENTRY;
        IF CODE=6 THEN
             CCR(MPU)='11'B^^SUBSTR(ACCA(MPU),3,6);
        ELSE ACCA(MPU)=BIN(CCR(MPU),8,0);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
INX:    ENTRY;
        INR(MPU)=INR(MPU)+1;
        IF INR(MPU)>65535 THEN INR(MPU)=INR(MPU)-65536;
        IF INR(MPU)=0 THEN CKZ=1;
             ELSE CKZ=0.0;
        CALL CCRCHK(0,0,0,2,2,2,CKZ,2,2);
        PC(MPU)=PC(MPU)+1;
        CYCLES=4;
        RETURN;
DEX:    ENTRY;
        INR(MPU)=INR(MPU)-1;
        IF INR(MPU)<0 THEN INR(MPU)=65536+INR(MPU);
        IF INR(MPU)=0 THEN CKZ=1;
             ELSE CKZ=0;
        CALL CCRCHK(0,0,0,2,2,2,CKZ,2,2);
```

```
        PC(MPU)=PC(MPU)+1;
        CYCLES=4;
        RETURN;
CLV:    ENTRY;
        CALL CCRCHK(0,0,0,2,2,2,2,0,2);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
SEV:    ENTRY;
        CALL CCRCHK(0,0,0,2,2,2,2,1,2);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
CLC:    ENTRY;
        CALL CCRCHK(0,0,0,2,2,2,2,2,0);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
SEC:    ENTRY;
        CALL CCRCHK(0,0,0,2,2,2,2,2,1);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
CLI:    ENTRY;
        CALL CCRCHK(0,0,0,2,0,2,2,2,2);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
SEI:    ENTRY;
        CALL CCRCHK(0,0,0,2,1,2,2,2,2);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
SBA:    ENTRY;
        TEMP=256-ACCB(MPU);
        RESULT=ACCA(MPU)+TEMP;
        IF RESULT>255 THEN RESULT=RESULT-256;
        CKC=0;
        IF ACCB(MPU)>127 & ACCA(MPU)<128 THEN CKC=1;
        IF ACCB(MPU)>127 & RESULT>127 THEN CKC=1;
        IF RESULT>127 & ACCA(MPU)<128 THEN CKC=1;
        CALL CCRCHK(RESULT,ACCA(MPU),TEMP,2,2,3,3,3,3);
        IF CODE=16 THEN ACCA(MPU)=RESULT;
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
TAB:    ENTRY;
        IF CODE=22 THEN ACCB(MPU)=ACCA(MPU);
```

```
        ELSE ACCA(MPU)=ACCB(MPU);
        CALL CCRCHK(ACCB(MPU),0,0,2,2,3,3,0,2);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
DAA:    ENTRY;
        LHB=SUBSTR(ACCA(MPU),5,4);
        MHB=SUBSTR(ACCA(MPU),1,4);
        IF SUBSTR(CCR(MPU),8,1)='0'B THEN
             IF MHB<=9 THEN
                  IF SUBSTR(CCR(MPU),3,1)='1'B THEN
                       ACCA(MPU)=ACCA(MPU)+6;
                  ELSE IF MHB<9 THEN
                            IF LHB>9 THEN
                                 ACCA(MPU)=ACCA(MPU)+6;
                                 ELSE;
                       ELSE IF LHB>9 THEN DO;
                                 ACCA(MPU)=ACCA(MPU)+102;
                                 CKC=1;
                                 END;
                            ELSE;
             ELSE IF SUBSTR(CCR(MPU),3,1)='1'B THEN DO;
                            ACCA(MPU)=ACCA(MPU)+102;
                            CKC=1;
                            END;
                  ELSE DO;
                            ACCA(MPU)=ACCA(MPU)+96;
                            CKC=1;
                            END;
        ELSE IF SUBSTR(CCR(MPU),3,1)='1'B THEN ACCA(MPU)=ACCA(MPU)+102;
             ELSE IF LHB<=9 THEN
                            ACCA(MPU)=ACCA(MPU)+96;
                       ELSE ACCA(MPU)=ACCA(MPU)+102;
        CALL CCRCHK(ACCA(MPU),0,0,2,2,3,3,2,CKC);
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
ABA:    ENTRY;
        RESULT=ACCA(MPU)+ACCB(MPU);
        IF RESULT>255 THEN RESULT=RESULT-256;
        CALL CCRCHK(RESULT,ACCA(MPU),ACCB(MPU),3,2,3,3,3,3);
        ACCA(MPU)=RESULT;
        PC(MPU)=PC(MPU)+1;
        CYCLES=2;
        RETURN;
BRA:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
```

```
        CALL BRANCH;
        CYCLES=4;
        RETURN;
BHI:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),8,1)='0' & SUBSTR(CCR(MPU),6,1)='0'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BLS:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),8,1)^SUBSTR(CCR(MPU),6,1)='1'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BCC:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),8,1)='0'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BCS:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),8,1)='1'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BNE:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),6,1)='0'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BEQ:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),6,1)='1'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BVC:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
```

```
        IF SUBSTR(CCR(MPU),7,1)='0'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BVS:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),7,1)='1'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BPL:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),5,1)='0'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BMI:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF SUBSTR(CCR(MPU),5,1)='1'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BGE:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF BOOL(SUBSTR(CCR(MPU),5,1),SUBSTR(CCR(MPU),7,1),'0110'B)='0'B
             THEN CALL BRANCH;
        CYCLES=4;
        RETURN;
BLT:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF BOOL(SUBSTR(CCR(MPU),5,1),SUBSTR(CCR(MPU),7,1),'0110'B)='1'B
             THEN CALL BRANCH;
        CYCLES=4;
        RETURN;
BGT:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF (SUBSTR(CCR(MPU),6,1)^BOOL(SUBSTR(CCR(MPU),5,1),
             SUBSTR(CCR(MPU),7,1),'0110'B))='0'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
BLE:    ENTRY;
```

```
        OPRND1=READ(PC(MPU)+1);
        PC(MPU)=PC(MPU)+2;
        IF (SUBSTR(CCR(MPU),6,1)^BOOL(SUBSTR(CCR(MPU),5,1),
             SUBSTR(CCR(MPU),7,1),'0110'B))='1'B THEN
             CALL BRANCH;
        CYCLES=4;
        RETURN;
TSX:    ENTRY;
        IF CODE=48 THEN DO;
             INR(MPU)=SP(MPU)+1;
             IF INR(MPU)>65535 THEN INR(MPU)=INR(MPU)-65536;
             END;
        ELSE DO;
             SP(MPU)=INR(MPU)-1;
             IF SP(MPU)<0 THEN SP(MPU)=SP(MPU)+65536;
             END;
        PC(MPU)=PC(MPU)+1;
        CYCLES=4;
        RETURN;
INS:    ENTRY;
        SP(MPU)=SP(MPU)+1;
        IF SP(MPU)>65535 THEN SP(MPU)=SP(MPU)-65536;
        PC(MPU)=PC(MPU)+1;
        CYCLES=4;
        RETURN;
PUL:    ENTRY;
        SP(MPU)=SP(MPU)+1;
        IF SP(MPU)>65535 THEN SP(MPU)=SP(MPU)-65536;
        IF CODE=50 THEN ACCA(MPU)=READ(SP(MPU));
        ELSE ACCB(MPU)=READ(SP(MPU));
        PC(MPU)=PC(MPU)+1;
        CYCLES=4;
        RETURN;
DES:    ENTRY;
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
        PC(MPU)=PC(MPU)+1;
        CYCLES=4;
        RETURN;
PSH:    ENTRY;
        IF CODE=54 THEN CALL WRITE(ACCA(MPU),SP(MPU));
        ELSE CALL WRITE(ACCB(MPU),SP(MPU));
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
        PC(MPU)=PC(MPU)+1;
        CYCLES=4;
        RETURN;
RTS:    ENTRY;
```

```
        STR1=SYSTBL(7,MPU);
        SUBSTR(STR1,9,1)='1'B;
        SYSTBL(7,MPU)=STR1;
        SP(MPU)=SP(MPU)+2;
        IF SP(MPU)>65535 THEN SP(MPU)=SP(MPU)-65536;
        PC(MPU)=256*READ(SP(MPU)-1)+READ(SP(MPU));
        CYCLES=5;
        RETURN;
RTI:    ENTRY;
        STR1=SYSTBL(7,MPU);
        SUBSTR(STR1,8,1)='1'B;
        SYSTBL(7,MPU)=STR1;
        SP(MPU)=SP(MPU)+1;
        IF SP(MPU)>65535 THEN SP(MPU)=SP(MPU)-65536;
        CCR(MPU)=BIT(READ(SP(MPU)),8);
        SP(MPU)=SP(MPU)+1;
        IF SP(MPU)>65535 THEN SP(MPU)=SP(MPU)-65536;
        ACCB(MPU)=READ(SP(MPU));
        SP(MPU)=SP(MPU)+1;
        IF SP(MPU)>65535 THEN SP(MPU)=SP(MPU)-65536;
        ACCA(MPU)=READ(SP(MPU));
        INR(MPU)=256*READ(SP(MPU)+1)+READ(SP(MPU)+2);
        SP(MPU)=SP(MPU)+4;
        PC(MPU)=256*READ(SP(MPU)-1)+READ(SP(MPU));
        CYCLES=10;
        RETURN;
SWI:    ENTRY;
        PC(MPU)=PC(MPU)+1;
        CALL WRITE(SUBSTR(PC(MPU),9,8),SP(MPU));
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=65536+ SP(MPU);
        CALL WRITE(SUBSTR(PC(MPU),1,8),SP(MPU));
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
        CALL WRITE(SUBSTR(INR(MPU),9,8),SP(MPU));
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
        CALL WRITE(SUBSTR(INR(MPU),1,8),SP(MPU));
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
        CALL WRITE(ACCA(MPU),SP(MPU));
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
        CALL WRITE(ACCB(MPU),SP(MPU));
        SP(MPU)=SP(MPU)-1;
        IF SP(MPU)<0 THEN SP(MPU)=SP(MPU)+65536;
        CALL WRITE(BIN(CCR(MPU),8,0),SP(MPU));
        SP(MPU)=SP(MPU)-1;
```

```
        IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
        STR1=SYSTBL(7,MPU);
        IF CODE=62 THEN DO;
             SUBSTR(STR1,4,3)='101'B;
             CYCLES=9;
             END;
        ELSE DO;
             SUBSTR(STR1,4,3)='010'B;
             SUBSTR(CCR(MPU),4,1)='1'B;
             CYCLES=12;
             PC(MPU)=256*READ(65530)+READ(65531);
             END;
        SYSTBL(7,MPU)=BIN(STR1,31,0);
        RETURN;
NEG:    ENTRY;
        IF CODE=64 THEN DO;
             IF ACCA(MPU)\=0 THEN ACCA(MPU)=255-ACCA(MPU);
             CYCLES=2;
             NEXT=1;
             RESULT=ACCA(MPU);
             END;
        IF CODE=80 THEN DO;
             IF ACCB(MPU)\=0 THEN ACCB(MPU)=255-ACCB(MPU);
             CYCLES=2;
             NEXT=1;
             RESULT=ACCB(MPU);
             END;
        IF CODE=96 THEN DO;
             OPRND1=READ(PC(MPU)+1);
             ADDRESS=INR(MPU)+OPRND1;
             RESULT=255-READ(ADDRESS);
             CALL WRITE(RESULT,ADDRESS);
             CYCLES=4;
             NEXT=2;
             END;
        IF CODE=112 THEN DO;
             OPRND1=READ(PC(MPU)+1);
             OPRND2=READ(PC(MPU)+2);
             ADDRESS=256*OPRND1+OPRND2;
             CALL WRITE(255-READ(ADDRESS),ADDRESS);
             RESULT=READ(ADDRESS);
             CYCLES=6;
             NEXT=3;
             END;
        IF RESULT=128 THEN CKV=1;
             ELSE CKV=0;
        IF RESULT=0 THEN CKC=1;
             ELSE CKC=0;
```

```
          CALL CCRCHK(RESULT,0,0,2,2,3,3,CKV,CKC);
          PC(MPU)=PC(MPU)+NEXT;
          RETURN;
COM:      ENTRY;
          IF CODE=67 THEN DO;
               RESULT,ACCA(MPU)=255-ACCA(MPU);
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=83 THEN DO;
               RESULT,ACCB(MPU)=255-ACCB(MPU);
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=99 THEN DO;
               OPRND1=READ(PC(MPU)+1);
               ADDRESS=INR(MPU)+OPRND1;
               RESULT=225-READ(ADDRESS);
               CALL WRITE(RESULT,ADDRESS);
               CYCLES=7;
               NEXT=2;
               END;
          IF CODE=115 THEN DO;
               OPRND1=READ(PC(MPU)+1);
               OPRND2=READ(PC(MPU)+2);
               ADDRESS=256*OPRND1+OPRND2;
                    RESULT=255-READ(ADDRESS);
               CALL WRITE(RESULT,ADDRESS);
               NEXT=3;
               CYCLES=6;
               END;
          CALL CCRCHK(RESULT,0,0,2,2,3,3,0,1);
          PC(MPU)=PC(MPU)+NEXT;
          RETURN;
LSR:      ENTRY;
          IF CODE=68 THEN DO;
               TEMP=ACCA(MPU);
               ACCA(MPU)=ACCA(MPU)/2;
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=84 THEN DO;
               TEMP=ACCB(MPU);
               ACCB(MPU)=ACCB(MPU)/2;
               CYCLES=2;
               NEXT=1;
               END;
         IF CODE=100 ^ CODE=116 THEN DO;
```

```
          OPRND1=READ(PC(MPU)+1);
          IF CODE=100 THEN DO;
               ADDRESS=INR(MPU)+OPRND1;
               CYCLES=7;
               NEXT=2;
               END;
          ELSE DO;
               OPRND2=READ(PC(MPU)+2);
               ADDRESS=256*OPRND1+OPRND2;
               CYCLES=6;
               NEXT=3;
               END;
               TEMP=READ(ADDRESS);
          CALL WRITE(TEMP/2,ADDRESS);
          END;
     CKC=SUBSTR(TEMP,8,1);
     TEMP=TEMP/2;
     CALL CCRCHK(TEMP,0,0,2,2,0,3,CKC,CKC);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
ROR: ENTRY;
     IF CODE=70 THEN DO;
          CKC=SUBSTR(ACCA(MPU),8,1);
          ACCA(MPU)=ACCA(MPU)/2;
          IF SUBSTR(CCR(MPU),8,1)='1'B THEN
               ACCA(MPU)=ACCA(MPU)+10000000B;
          TEMP=ACCA(MPU);
          CYCLES=2;
          NEXT=1;
          END;
     IF CODE=86 THEN DO;
          CKC=SUBSTR(ACCB(MPU),8,1);
          ACCB(MPU)=ACCB(MPU)/2;
          IF SUBSTR(CCR(MPU),8,1)='1'B THEN
               ACCB(MPU)=ACCB(MPU)+10000000B;
          TEMP=ACCB(MPU);
          CYCLES=2;
          NEXT=1;
          END;
     IF CODE=102 ^ CODE=118 THEN DO;
          OPRND1=READ(PC(MPU)+1);
          IF CODE=102 THEN DO;
               ADDRESS=INR(MPU)+OPRND1;
               CYCLES=7;
               NEXT=2;
               END;
          ELSE DO;
               OPRND2=READ(PC(MPU)+2);
```

```
                    ADDRESS=256*OPRND1+OPRND2;
                    CYCLES=6;
                    NEXT=3;
                    END;
               TEMP=READ(ADDRESS);
               CKC=SUBSTR(TEMP,8,1);
               TEMP=TEMP/2;
               IF SUBSTR(CCR(MPU),8,1)='1'B THEN
                    TEMP=TEMP+10000000B;
               CALL WRITE(TEMP,ADDRESS);
               END;
          CALL CCRCHK(TEMP,0,0,2,2,3,3,2,CKC);
          SUBSTR(CCR(MPU),7,1)=BOOL(SUBSTR(CCR(MPU),8,1),
               SUBSTR(CCR(MPU),5,1),'0110'B);
          PC(MPU)=PC(MPU)+NEXT;
          RETURN;
ASR:      ENTRY;
          IF CODE=71 THEN DO;
               TEMP=ACCA(MPU)/2;
               CKC=SUBSTR(ACCA(MPU),8,1);
               STRING=BIT(TEMP,8);
               SUBSTR(STRING,1,1)=SUBSTR(ACCA(MPU),1,1);
               TEMP,ACCA(MPU)=BIN(STRING,8,0);
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=87 THEN DO;
               TEMP=ACCB(MPU)/2;
               CKC=SUBSTR(ACCB(MPU),8,1);
               STRING=BIT(TEMP,8);
               SUBSTR(STRING,1,1)=SUBSTR(ACCB(MPU),1,1);
               ACCB(MPU),TEMP=BIN(STRING,8,0);
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=103 ^ CODE=119 THEN DO;
               OPRND1=READ(PC(MPU)+1);
               IF CODE=103 THEN DO;
                    ADDRESS=INR(MPU)+OPRND1;
                    CYCLES=7;
                    NEXT=2;
                    END;
               ELSE DO;
                    OPRND2=READ(PC(MPU)+2);
                    ADDRESS=256*OPRND1+OPRND2;
                    CYCLES=6;
                    NEXT=3;
                    END;
```

```
          TEMP=READ(ADDRESS);
          CKC=SUBSTR(TEMP,8,1);
          STRING=BIT(TEMP/2,8);
          TEMP=BIN(STRING,8,0);
          CALL WRITE(TEMP,ADDRESS);
     END;
     CALL CCRCHK(TEMP,0,0,2,2,3,3,2,CKC);
     SUBSTR(CCR(MPU),7,1)=BOOL(SUBSTR(CCR(MPU),8,1),
          SUBSTR(CCR(MPU),5,1),'0110'B);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
ASL: ENTRY;
     IF CODE=72 THEN DO;
          CKC=SUBSTR(ACCA(MPU),1,1);
         TEMP,ACCA(MPU)=ACCA(MPU)*2;
          CYCLES=2;
          NEXT=1;
          END;
     IF CODE=88 THEN DO;
          CKC=SUBSTR(ACCB(MPU),1,1);
          TEMP,ACCB(MPU)=ACCB(MPU)*2;
          CYCLES=2;
          NEXT=1;
          END;
     IF CODE=104 ^ CODE=120 THEN DO;
          OPRND1=READ(PC(MPU)+1);
          IF CODE=104 THEN DO;
               ADDRESS=INR(MPU)+OPRND1;
               CYCLES=7;
               NEXT=2;
               END;
          ELSE DO;
               OPRND2=READ(PC(MPU)+2);
               ADDRESS=256*OPRND1+OPRND2;
               CYCLES=6;
               NEXT=3;
               END;
          TEMP=READ(ADDRESS);
          CKC=SUBSTR(TEMP,1,1);
          TEMP=TEMP/2;
          CALL WRITE(TEMP,ADDRESS);
          END;
     CALL CCRCHK(TEMP,0,0,2,2,3,3,2,CKC);
     SUBSTR(CCR(MPU),7,1)=BOOL(SUBSTR(CCR(MPU),8,1),
          SUBSTR(CCR(MPU),5,1),'0110'B);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
ROL: ENTRY;
```

```
          IF CODE=73 THEN DO;
               TEMP=ACCA(MPU)*2;
               CKC=SUBSTR(ACCA(MPU),1,1);
               IF SUBSTR(CCR(MPU),8,1)='1'B THEN TEMP=TEMP+1;
               ACCA(MPU)=TEMP;
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=89 THEN DO;
               TEMP=ACCB(MPU)*2;
               CKC=SUBSTR(ACCB(MPU),1,1);
               IF SUBSTR(CCR(MPU),8,1)='1'B THEN TEMP=TEMP+1;
               ACCB(MPU)=TEMP;
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=105 ^ CODE=121 THEN DO;
               OPRND1=READ(PC(MPU)+1);
               IF CODE=105 THEN DO;
                    ADDRESS=INR(MPU)+OPRND1;
                    CYCLES=7;
                    NEXT=2;
                    END;
               ELSE DO;
                    OPRND2=READ(PC(MPU)+2);
                    ADDRESS=256*OPRND1+OPRND2;
                    CYCLES=6;
                    NEXT=3;
                    END;
               TEMP=READ(ADDRESS)*2;
               CKC=SUBSTR(READ(ADDRESS),1,1);
               IF SUBSTR(CCR(MPU),8,1)='1'B THEN TEMP=TEMP+1;
               CALL WRITE(TEMP,ADDRESS);
               END;
          CALL CCRCHK(TEMP,0,0,2,2,3,3,2,CKC);
          SUBSTR(CCR(MPU),7,1)=BOOL(SUBSTR(CCR(MPU),8,1),
               SUBSTR(CCR(MPU),5,1),'0110'B);
          PC(MPU)=PC(MPU)+NEXT;
          RETURN;
DECC:     ENTRY;
          IF CODE=74 THEN DO;
               TEMP=ACCA(MPU);
               RESULT,ACCA(MPU)=ACCA(MPU)-1;
               IF RESULT<0 THEN RESULT,ACCA(MPU)=RESULT+256;
               CYCLES=2;
               NEXT=1;
               END;
          IF CODE=90 THEN DO;
```

```
          TEMP=ACCB(MPU);
          RESULT,ACCB(MPU)=ACCB(MPU)-1;
          IF RESULT<0 THEN RESULT,ACCB(MPU)= RESULT+256;
          CYCLES=2;
          NEXT=1;
          END;
     IF CODE=106 ^ CODE=122 THEN DO;
          OPRND1=READ(PC(MPU)+1);
          IF CODE=106 THEN DO;
               ADDRESS=INR(MPU)+OPRND1;
               CYCLES=7;
               NEXT=2;
               END;
          ELSE DO;
               OPRND2=READ(PC(MPU)+2);
               ADDRESS=256*OPRND1+OPRND2;
               CYCLES=6;
               NEXT=3;
               END;
          TEMP=READ(ADDRESS);
          RESULT=TEMP-1;
          IF RESULT<0 THEN RESULT=RESULT+256;
          CALL WRITE(RESULT,ADDRESS);
          END;
     IF RESULT=128 THEN CKV=1;
     ELSE CKV=0;
     CALL CCRCHK(RESULT,TEMP,255,2,2,3,3,CKV,2);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
INC: ENTRY;
     IF CODE=76 THEN DO;
          TEMP=ACCA(MPU);
          RESULT,ACCA(MPU)=ACCA(MPU)+1;
          IF RESULT>255 THEN RESULT,ACCA(MPU)=RESULT-255;
          CYCLES=2;
          NEXT=1;
          END;
     IF CODE=92 THEN DO;
          TEMP=ACCB(MPU);
          RESULT,ACCB(MPU)=ACCB(MPU)+1;
          IF RESULT>255 THEN RESULT,ACCB(MPU)=RESULT-255;
          CYCLES=2;
          NEXT=1;
          END;
     IF CODE=108 ^ CODE=124 THEN DO;
          OPRND1=READ(PC(MPU)+1);
          IF CODE=108 THEN DO;
               ADDRESS=INR(MPU)+OPRND1;
```

```
                CYCLES=7;
                NEXT=2;
                END;
           ELSE DO;
                OPRND2=READ(PC(MPU)+2);
                ADDRESS=256*OPRND1+OPRND2;
                CYCLES=6;
                NEXT=3;
                END;
           TEMP=READ(ADDRESS);
           RESULT=TEMP+1;
           IF RESULT>255 THEN RESULT=RESULT-256;
           CALL WRITE(RESULT,ADDRESS);
           END;
      CALL CCRCHK(RESULT,TEMP,1,2,2,3,3,3,2);
      PC(MPU)=PC(MPU)+NEXT;
      RETURN;
TST:  ENTRY;
      IF CODE=77 THEN DO;
           TEMP=ACCA(MPU);
           CYCLES=2;
           NEXT=1;
           END;
      IF CODE=93 THEN DO;
           TEMP=ACCB(MPU);
           CYCLES=2;
           NEXT=1;
           END;
      IF CODE=109 ^ CODE=125 THEN DO;
           OPRND1=READ(PC(MPU)+1);
           IF CODE=109 THEN DO;
                ADDRESS=INR(MPU)+OPRND1;
                CYCLES=7;
                NEXT=2;
                END;
           ELSE DO;
                OPRND2=READ(PC(MPU)+2);
                ADDRESS=256*OPRND1+OPRND2;
                CYCLES=6;
                NEXT=3;
                END;
           TEMP=READ(ADDRESS);
           END;
      CALL CCRCHK(TEMP,0,0,2,2,3,3,0,0);
      PC(MPU)=PC(MPU)+NEXT;
      RETURN;
JMP:  ENTRY;
      OPRND1=READ(PC(MPU)+1);
```

```
        IF CODE=110 THEN DO;
             ADDRESS=INR(MPU)+OPRND1;
             CYCLES=4;
             END;
        ELSE DO;
             OPRND2=READ(PC(MPU)+2);
             ADDRESS=256*OPRND1+OPRND2;
             CYCLES=3;
             END;
        PC(MPU)=ADDRESS;
        RETURN;
CLR:    ENTRY;
        IF CODE=79 THEN DO;
             ACCA(MPU)=0;
             CYCLES=2;
             NEXT=1;
             END;
        IF CODE=95 THEN DO;
             ACCB(MPU)=0;
             CYCLES=2;
             NEXT=1;
             END;
        IF CODE=111 ^ CODE=127 THEN DO;
             OPRND1=READ(PC(MPU)+1);
             IF CODE=111 THEN DO;
                  ADDRESS=INR(MPU)+OPRND1;
                  CYCLES=7;
                  NEXT=2;
                  END;
             ELSE DO;
                  OPRND2=READ(PC(MPU)+2);
                  ADDRESS=256*OPRND1+OPRND2;
                  CYCLES=6;
                  NEXT=3;
                  END;
             CALL WRITE(0,ADDRESS);
             END;
        CALL CCRCHK(0,0,0,2,2,0,1,0,0);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
SUB:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        NEXT=2;
        IF CODE=128 ^ CODE=192 THEN DO;
             TEMP=256-OPRND1;
             CYCLES=2;
             END;
        IF CODE=144 ^ CODE=208 THEN DO;
```

```
          TEMP=256-READ(BIN(OPRND1,16,0));
          CYCLES=2;
          END;
     IF CODE=160 ^ CODE=224 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          TEMP=256-READ(ADDRESS);
          CYCLES=5;
          END;
     IF CODE=176 ^ CODE=240 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=256-READ(ADDRESS);
          CYCLES=3;
          NEXT=3;
          END;
     IF CODE<192 THEN DO;
          TEMP2=ACCA(MPU);
          RESULT,ACCA(MPU)=TEMP2+TEMP;
          IF RESULT>255 THEN RESULT,ACCA(MPU)=RESULT-256;
          END;
     ELSE DO;
          TEMP2=ACCB(MPU);
          RESULT,ACCB(MPU)=TEMP2+TEMP;
          IF RESULT>255 THEN RESULT,ACCB(MPU)=RESULT-256;
          END;
     CKC=0;
     IF OPRND1>127 & TEMP2<128 THEN CKC=1;
     IF OPRND1>127 & RESULT>127 THEN CKC=1;
     IF RESULT>127 & TEMP2<128 THEN CKC=1;
     CALL CCRCHK(RESULT,TEMP2,TEMP,2,2,3,3,3,CKC);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
CMP: ENTRY;
     OPRND1=READ(PC(MPU)+1);
     TEMP=0;
     NEXT=2;
     IF CODE=129 ^ CODE=193 THEN DO;
          TEMP=256-OPRND1;
          CYCLES=2;
          END;
     IF CODE=145 ^ CODE=209 THEN DO;
          TEMP=256-READ(BIN(OPRND1,16,0));
          CYCLES=3;
          END;
     IF CODE=161 ^ CODE=225 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          TEMP=256-READ(ADDRESS);
          CYCLES=5;
```

```
          END;
        IF CODE=177 ^ CODE=241 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=256-READ(ADDRESS);
          CYCLES=4;
          NEXT=3;
          END;
        IF CODE<192 THEN TEMP2=ACCA(MPU);
        ELSE TEMP2=ACCB(MPU);
        RESULT=TEMP2+TEMP;
        IF RESULT>255 THEN RESULT=RESULT-256;
        CALL CCRCHK(RESULT,TEMP2,TEMP,2,2,3,3,3,3);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
SBC:    ENTRY;
        TEMP=0;
        OPRND1=READ(PC(MPU)+1);
        CKC=BIN(SUBSTR(CCR(MPU),8,1),2,0);
        NEXT=2;
        IF CODE=130 ^ CODE=194 THEN DO;
          TEMP=256-OPRND1-CKC;
          CYCLES=2;
          END;
        IF CODE=146 ^ CODE=210 THEN DO;
          TEMP=256-READ(BIN(OPRND1,16,0))-CKC;
          CYCLES=3;
          END;
        IF CODE=162 ^ CODE=226 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          TEMP=256-READ(ADDRESS)-CKC;
          CYCLES=5;
          END;
        IF CODE=178 ^ CODE=242 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=256-READ(ADDRESS)-CKC;
          CYCLES=4;
          NEXT=3;
          END;
        IF CODE<192 THEN DO;
          TEMP2=ACCA(MPU);
          RESULT,ACCA(MPU)=TEMP2+TEMP;
          IF RESULT>255 THEN RESULT,ACCA(MPU)=RESULT-256;
          END;
        ELSE DO;
          TEMP2=ACCB(MPU);
          RESULT,ACCB(MPU)=TEMP2+TEMP;
```

```
          IF RESULT>255 THEN RESULT,ACCB(MPU)=RESULT-256;
          END;
        CKC=0;
        IF OPRND1>127 & TEMP2<128 THEN CKC=1;
        IF OPRND1>127 & RESULT>127 THEN CKC=1;
        IF RESULT>127 & TEMP2<128 THEN CKC=1;
        CALL CCRCHK(RESULT,TEMP2,TEMP,2,2,3,3,3,CKC);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
AND:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        RESULT=0;
        NEXT=2;
        IF CODE=132 ^ CODE=196 THEN DO;
          CYCLES=2;
          TEMP=OPRND1;
          END;
        IF CODE=148 ^ CODE=212 THEN DO;
          TEMP=READ(BIN(OPRND1,16,0));
          CYCLES=3;
          END;
        IF CODE=164 ^ CODE=228 THEN DO;
          CYCLES=5;
          TEMP=READ(INR(MPU)+OPRND1);
          NEXT=3;
          END;
        IF CODE=180 ^ CODE=244 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=READ(ADDRESS);
          CYCLES=4;
          NEXT=3;
          END;
        IF CODE<192 THEN DO;
          TEMP2=ACCA(MPU);
          RESULT,ACCA(MPU)=BOOL(TEMP2,TEMP,'0001'B);
          END;
        ELSE DO;
          TEMP2=ACCB(MPU);
          RESULT,ACCB(MPU)=BOOL(TEMP2,TEMP,'0001'B);
          END;
        CALL CCRCHK(RESULT,0,0,2,2,3,3,0,2);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
BITT:   ENTRY;
        OPRND1=READ(PC(MPU)+1);
        NEXT=2;
        IF CODE=133 ^ CODE=197 THEN DO;
```

```
          TEMP=OPRND1;
          CYCLES=2;
          END;
     IF CODE=165 ^ CODE=213 THEN DO;
          TEMP=READ(BIN(OPRND1,16,0));
          CYCLES=3;
          END;
     IF CODE=165 ^ CODE=229 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          TEMP=READ(ADDRESS);
          CYCLES=5;
          END;
     IF CODE=181 ^ CODE=245 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=READ(ADDRESS);
          CYCLES=4;
          NEXT=3;
          END;
     IF CODE<192 THEN TEMP2=ACCA(MPU);
     ELSE TEMP2=ACCB(MPU);
     RESULT=BOOL(TEMP2,TEMP,'0001'B);
     CALL CCRCHK(RESULT,0,0,2,2,3,3,0,3);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
LDA: ENTRY;
     OPRND1=READ(PC(MPU)+1);
     NEXT=2;
     IF CODE=134 ^ CODE=198 THEN DO;
          TEMP=OPRND1;
          CYCLES=2;
          END;
     IF CODE=150 ^ CODE=214 THEN DO;
          TEMP=READ(BIN(OPRND1,16,0));
          CYCLES=3;
          END;
     IF CODE=166 ^ CODE=230 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          TEMP=READ(ADDRESS);
          CYCLES=5;
          END;
     IF CODE=182 ^ CODE=246 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=READ(ADDRESS);
          CYCLES=4;
          NEXT=3;
          END;
```

```
        IF CODE<192 THEN RESULT,ACCA(MPU)=TEMP;
        ELSE RESULT,ACCB(MPU)=TEMP;
        CALL CCRCHK(RESULT,0,0,2,2,3,3,0,2);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
STA:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        NEXT=2;
        IF CODE=151 ^ CODE=215 THEN DO;
             ADDRESS=OPRND1;
             CYCLES=4;
             END;
        IF CODE=167 ^ CODE=231 THEN DO;
             ADDRESS=INR(MPU)+OPRND1;
             CYCLES=6;
             END;
        IF CODE=183 ^ CODE=247 THEN DO;
             OPRND2=READ(PC(MPU)+2);
             ADDRESS=256*OPRND1+OPRND2;
             CYCLES=5;
             NEXT=3;
             END;
        IF CODE<192 THEN
             RESULT=ACCA(MPU);
        ELSE RESULT=ACCB(MPU);
        CALL WRITE(RESULT,ADDRESS);
        CALL CCRCHK(RESULT,0,0,2,2,3,3,0,2);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
EOR:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        NEXT=2;
        IF CODE=136 ^ CODE=200 THEN DO;
             TEMP=OPRND1;
             CYCLES=2;
             END;
        IF CODE=152 ^ CODE=216 THEN DO;
             TEMP=READ(BIN(OPRND1,16,0));
             CYCLES=3;
             END;
        IF CODE=168 ^ CODE=232 THEN DO;
             ADDRESS=INR(MPU)+OPRND1;
             TEMP=READ(ADDRESS);
             CYCLES=5;
             END;
        IF CODE=184 ^ CODE=248 THEN DO;
             OPRND2=READ(PC(MPU)+2);
             ADDRESS=256*OPRND1+OPRND2;
```

```
          TEMP=READ(ADDRESS);
          CYCLES=4;
          NEXT=3;
          END;
     IF CODE<192 THEN DO;
          TEMP2=ACCA(MPU);
          ACCA(MPU),RESULT=BOOL(TEMP2,TEMP,'0110'B);
          END;
     ELSE DO;
          TEMP2=ACCB(MPU);
          ACCB(MPU),RESULT=BOOL(TEMP2,TEMP,'0110'B);
          END;
     CALL CCRCHK(RESULT,TEMP2,TEMP,2,2,3,3,0,2);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
ADC: ENTRY;
     OPRND1=READ(PC(MPU)+1);
     CKC=SUBSTR(CCR(MPU),8,1);
     NEXT=2;
     IF CODE=137 ^ CODE=201 THEN DO;
          TEMP=OPRND1;
          CYCLES=2;
          END;
     IF CODE=153 ^ CODE=217 THEN DO;
          TEMP=READ(BIN(OPRND1,16,0));
          CYCLES=3;
          END;
     IF CODE=169 ^ CODE=233 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          TEMP=READ(ADDRESS);
          CYCLES=5;
          END;
     IF CODE=185 ^ CODE=249 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=READ(ADDRESS);
          CYCLES=4;
          NEXT=3;
          END;
     IF CODE<192 THEN DO;
          TEMP2=ACCA(MPU);
          RESULT,ACCA(MPU)=TEMP2+TEMP+CKC;
          IF RESULT>255 THEN RESULT,ACCA(MPU)=RESULT-256;
          END;
     ELSE DO;
          TEMP2=ACCB(MPU);
          RESULT,ACCB(MPU)=TEMP2+TEMP+CKC;
          IF RESULT>255 THEN RESULT,ACCB(MPU)=RESULT-256;
```

```
          END;
        CALL CCRCHK(RESULT,TEMP2,TEMP,3,2,3,3,3,3);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
ORA:    ENTRY;
        OPRND1=READ(PC(MPU)+1);
        NEXT=2;
        IF CODE=138 ^ CODE=202 THEN DO;
             TEMP=OPRND1;
             CYCLES=2;
             END;
        IF CODE=154 ^ CODE=218 THEN DO;
             TEMP=READ(BIN(OPRND1,16,0));
             CYCLES=3;
             END;
        IF CODE=170 ^ CODE=234 THEN DO;
             ADDRESS=INR(MPU)+OPRND1;
             TEMP=READ(ADDRESS);
             CYCLES=5;
             END;
        IF CODE=186 ^ CODE=250 THEN DO;
             OPRND2=READ(PC(MPU)+2);
             ADDRESS=256*OPRND1+OPRND2;
             TEMP=READ(ADDRESS);
             CYCLES=4;
             NEXT=3;
             END;
        IF CODE<192 THEN DO;
             TEMP2=ACCA(MPU);
             RESULT,ACCA(MPU)=BOOL(TEMP2,TEMP,'0111'B);
             END;
        ELSE DO;
             TEMP2=ACCB(MPU);
             RESULT,ACCB(MPU)=BOOL(TEMP2,TEMP,'0111'B);
             END;
        CALL CCRCHK(RESULT,0,0,2,2,3,3,0,2);
        PC(MPU)=PC(MPU)+NEXT;
        RETURN;
ADDD:   ENTRY;
        OPRND1=READ(PC(MPU)+1);
        NEXT=2;
        IF CODE=139 ^ CODE=203 THEN DO;
             TEMP=OPRND1;
             CYCLES=2;
             END;
        IF CODE=155 ^ CODE=219 THEN DO;
             TEMP=READ(BIN(OPRND1,16,0));
             CYCLES=3;
```

```
          END;
     IF CODE=171 ^ CODE=235 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          TEMP=READ(ADDRESS);
          CYCLES=2;
          NEXT=3;
          END;
     IF CODE=187 ^ CODE=251 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          TEMP=READ(ADDRESS);
          CYCLES=4;
          NEXT=3;
          END;
     IF CODE<192 THEN DO;
          TEMP2=ACCA(MPU);
          ACCA(MPU),RESULT=TEMP2+TEMP;
          END;
     ELSE DO;
          TEMP2=ACCB(MPU);
          ACCB(MPU),RESULT=TEMP2+TEMP;
          IF RESULT>255 THEN RESULT,ACCB(MPU)=RESULT-256;
          END;
     CALL CCRCHK(RESULT,TEMP2,TEMP,3,2,3,3,3,3);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
CPX: ENTRY;
     OPRND1=READ(PC(MPU)+1);
     LHB,MHB=0;
     NEXT=2;
     IF CODE=140 THEN DO;
          LHB,OPRND2=READ(PC(MPU)+2);
          MHB=OPRND1;
          CYCLES=3;
          NEXT=3;
          END;
     IF CODE=156 THEN DO;
          MHB=READ(BIN(OPRND1,16,0));
          LHB=READ(OPRND1+1);
          CYCLES=4;
          END;
     IF CODE=172 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          MHB=READ(ADDRESS);
          LHB=READ(ADDRESS+1);
          CYCLES=6;
          END;
     IF CODE=188 THEN DO;
```

```
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          MHB=READ(ADDRESS);
          LHB=READ(ADDRESS+1);
          CYCLES=5;
          NEXT=3;
          END;
     TEMP=SUBSTR(INR(MPU),1,8);
     TEMP2=SUBSTR(INR(MPU),9,8);
     LHB=LHB+1;
     IF LHB>255 THEN LHB=LHB-256;
     LHB=256-LHB;
     RESULT=TEMP2+LHB;
     IF RESULT>255 THEN DO;
          RESULT=RESULT-256;
          CKV=1;
          END;
     ELSE CKV=0;
     IF RESULT=0 THEN CKZ=1;
     ELSE CKZ=0;
     MHB=256-MHB+CKV;
     RESULT=TEMP+MHB;
     IF RESULT>255 THEN RESULT=RESULT-256;
     IF CKZ=1 & RESULT=0 THEN CKZ=1;
     ELSE CKZ=0;
     CALL CCRCHK(RESULT,TEMP,MHB,2,2,3,CKZ,3,2);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
BSR: ENTRY;
     STR1=SYSTBL(7,MPU);
     SUBSTR(STR1,13,1)='1'B;
     SYSTBL(7,MPU)=STR1;
     OPRND1=READ(PC(MPU)+1);
     PC(MPU)=PC(MPU)+2;
     CALL WRITE(SUBSTR(PC(MPU),9,8),SP(MPU));
     SP(MPU)=SP(MPU)-1;
     IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
     CALL WRITE(SUBSTR(PC(MPU),1,8),SP(MPU));
     SP(MPU)=SP(MPU)-1;
     IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
     CALL BRANCH;
     CYCLES=8;
     RETURN;
LDR: ENTRY;
     OPRND1=READ(PC(MPU)+1);
     NEXT=2;
     IF CODE=142 ^ CODE=206 THEN DO;
          OPRND2=READ(PC(MPU)+2);
```

```
          ADDRESS=256*OPRND1+OPRND2;
          CYCLES=3;
          NEXT=3;
          END;
     IF CODE=158 ^ CODE=222 THEN DO;
          ADDRESS=256*READ(BIN(OPRND1,16,0))+READ(OPRND1+1);
          CYCLES=4;
          END;
     IF CODE=174 ^ CODE=238 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          ADDRESS=256*READ(ADDRESS)+READ(ADDRESS+1);
          CYCLES=6;
          END;
     IF CODE=190 ^ CODE=254 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          ADDRESS=256*READ(ADDRESS)+READ(ADDRESS+1);
          CYCLES=5;
          NEXT=3;
          END;
     IF CODE<192 THEN SP(MPU)=ADDRESS;
     ELSE INR(MPU)=ADDRESS;
     IF ADDRESS=0 THEN CKZ=1;
     ELSE CKZ=0;
     IF SUBSTR(ADDRESS,1,1) THEN CKN=1;
     ELSE CKN=0;
     CALL CCRCHK(0,0,0,2,2,CKN,CKZ,0,2);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
STR: ENTRY;
     OPRND1=READ(PC(MPU)+1);
     NEXT=2;
     IF CODE=159 ^ CODE=223 THEN DO;
          ADDRESS=OPRND1;
          CYCLES=5;
          END;
     IF CODE=175 ^ CODE=239 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          CYCLES=7;
          END;
     IF CODE=191 ^ CODE=255 THEN DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          CYCLES=6;
          NEXT=3;
          END;
     IF CODE<192 THEN DO;
          TEMP=SUBSTR(SP(MPU),1,8);
```

```
          TEMP2=SUBSTR(SP(MPU),9,8);
          END;
     ELSE DO;
          TEMP=SUBSTR(INR(MPU),1,8);
          TEMP2=SUBSTR(INR(MPU),9,8);
          END;
     CALL WRITE(TEMP,ADDRESS);
     CALL WRITE(TEMP2,ADDRESS+1);
     IF TEMP=0 & TEMP2=0 THEN CKZ=1;
     ELSE CKZ=0;
     CALL CCRCHK(TEMP,0,0,2,2,3,CKZ,0,2);
     PC(MPU)=PC(MPU)+NEXT;
     RETURN;
JSR: ENTRY;
     STR1=SYSTBL(7,MPU);
     SUBSTR(STR1,13,1)='1'B;
     SYSTBL(7,MPU)=STR1;
     OPRND1=READ(PC(MPU)+1);
     IF CODE=173 THEN DO;
          ADDRESS=INR(MPU)+OPRND1;
          CYCLES=8;
          NEXT=2;
          END;
     ELSE DO;
          OPRND2=READ(PC(MPU)+2);
          ADDRESS=256*OPRND1+OPRND2;
          CYCLES=9;
          NEXT=3;
          END;
     PC(MPU)=PC(MPU)+NEXT;
     CALL WRITE(SUBSTR(PC(MPU),9,8),SP(MPU));
     SP(MPU)=SP(MPU)-1;
     IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
     CALL WRITE(SUBSTR(PC(MPU),1,8),SP(MPU));
     SP(MPU)=SP(MPU)-1;
     IF SP(MPU)<0 THEN SP(MPU)=65536+SP(MPU);
     PC(MPU)=ADDRESS;
     RETURN;
      END INSTR;
```

```
/*    INTLC   VERSION-6                    3/5/78    */
INTLC: PROCEDURE;
/*********************************************************/
/*                                                       */
/*  THIS PROCEDURE PERFORMS SYSTEM INITIALIZATION.       */
/*                                                       */
/*********************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL TRCE(*,*) FLOAT BIN(31) CTL EXT;
   DCL ITABL(*,*) CHAR(1) CTL EXT;
   DCL (PC(*),SP(*),INR(*)) FIXED BIN(16,0) CTL EXT;
   DCL (ACCA(*),ACCB(*)) FIXED BIN(8,0) CTL EXT;
   DCL CCR(*) BIT(8) CTL EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL MCT(*) FIXED BIN(31,0) CTL;
   DCL BBIN ENTRY(CHAR(6)) RETURNS(FIXED BIN(16,0));
   DCL IERR ENTRY(FIXED BIN(31,0));
   DCL MFIND ENTRY(FIXED BIN(31,0),FIXED BIN(31,0))
        RETURNS(FIXED BIN(31,0));
   DCL (DATE,TIME,ONKEY,ONSOURCE,ONLOC,ONCODE) BUILTIN;
   DCL OLDADD FIXED BIN(31,0) EXT;
   DCL ECHO ENTRY;
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL HHEX ENTRY(FIXED BIN(31,0)) RETURNS(CHAR(6));
   DCL HEX CHAR(17) INIT(' 0123456789ABCDEF');
   DCL DCML CHAR(11) INIT(' 0123456789');
   DCL DDATE CHAR(17) EXT;
   DCL TTIME CHAR(9);
   DCL (I,NTPER,PRCT,J,K,MAX,TEMP) FIXED BIN(31,0);
   DCL (MPU,NPER,NMPU,PL) FIXED BIN(31,0) EXT;
   DCL (CARD,END_OF_CARD,STR1) CHAR(6);
   DCL (FATAL,WARNING,ADDRESS) FIXED BIN(31,0);
   DCL MSTOP CHAR(5) EXT;
   DCL FLAG BIT(1);
   DCL TTBEG FLOAT BIN EXT;
   DCL TYPE BIT(3);
   DCL 1 STAT(*) CTL EXT,
        2 INST FIXED BIN(31,0),
        2 INPT FIXED BIN(31,0),
        2 OTPT FIXED BIN(31,0),
        2 ERRS FIXED BIN(31,0),
        2 INTS FIXED BIN(31,0),
        2 WARN FIXED BIN(31,0),
        2 BRCH FIXED BIN(31,0),
        2 RTRN FIXED BIN(31,0),
        2 RSTS FIXED BIN(31,0),
        2 HALT FLOAT BIN(31),
        2 RUN FLOAT BIN(31),
```

```
     2 RESTRT FLOAT BIN(31),
     2 LSTHLT FLOAT BIN(31);
DCL 1 CMEM(*) CTL EXT,
     2 CMPU FIXED BIN(31,0),
     2 CLOC(0:31) BIT(16);
DCL MEMTYPE CHAR(6) EXT;
DCL 1 TMEMREC,
     2 UNUSED BIT(16),
     2 TMEM_KEY CHAR(10),
     2 TLMPU FIXED BIN(31,0),
     2 TLOC(0:31) BIT(16);
DCL 1 MEMREC EXT,
     2 UNUSED BIT(16),
     2 MEM_KEY CHAR(10),
     2 LMPU FIXED BIN(31,0),
     2 LOC(0:31) BIT(16);
DCL 1 PERTBL(*) CTL EXT,
     2 PMPU FIXED BIN(31,0),
     2 PADDR FIXED BIN(31,0),
     2 PASSOC FIXED BIN(31,0),
     2 PASADD FIXED BIN(31,0),
     2 PCODE FIXED BIN(31,0),
     2 CTYPE BIT(1),
     2 UNUSED BIT(5),
     2 C2 BIT(1),
     2 C1 BIT(1),
     2 DDR BIT(8),
     2 PDR BIT(8);
DCL 1 TPERTBL(*) CTL,
     2 TMPU FIXED BIN(31,0),
     2 TADDR FIXED BIN(31,0),
     2 TASSOC FIXED BIN(31,0),
     2 TASADD FIXED BIN(31,0),
     2 TCODE FIXED BIN(31,0),
     2 TCTYPE BIT(1),
     2 TC2 BIT(1),
     2 TC1 BIT(1),
     2 TDDR BIT(8),
     2 TPDR BIT(8);
DCL 1 PERREC,
     2 UNUSED CHAR(9),
     2 PENTRY(3),
          3 RMPU FIXED BIN(31,0),
          3 RADDR FIXED BIN(31,0),
          3 RASSOC FIXED BIN(31,0),
          3 RASADD FIXED BIN(31,0),
          3 RCODE FIXED BIN(31,0),
          3 RCTYPE BIT(1),
```

```
            3 UNUSED BIT(5),
            3 RC2 BIT(1),
            3 RC1 BIT(1),
            3 RDDR BIT(8),
            3 RPDR BIT(8);
   DCL 1 SYSREC,
        2 SDATA(0:19) FIXED BIN(31,0);
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL SIMSYS FILE RECORD SEQUENTIAL ENV(F(800,80));
/*****************************************************/
/*                                                   */
/*      ON UNITS.................                    */
/*                                                   */
/*****************************************************/
   ON CONVERSION BEGIN;
        PUT SKIP(5) EDIT('CONVERSION ERROR....SOURCE='^^ONSOURCE^^
             'IN PROCEDURE '^^ONLOC)(A);
        ONSOURCE='0';
        FATAL=FATAL+1;
        END;
   ON ENDFILE(SYSIN) BEGIN;
        CALL IERR(18);
        GO TO EOI;
        END;
   FATAL,WARNING=0;
   STR1=DATE;
   TTIME=TIME;
   DDATE=SUBSTR(STR1,3,2)^^'/'^^SUBSTR(STR1,5,2)^^'/'^^
        SUBSTR(STR1,1,2)^^' '^^SUBSTR(TTIME,1,2)^^':'^^
        SUBSTR(TTIME,3,2)^^':'^^SUBSTR(TTIME,5,2);
/*****************************************************/
/*                                                   */
/*  READ 'SYS' CARD.                                 */
/*                                                   */
/*****************************************************/
   CARD='SYS';
   GET LIST(STR1);
   IF STR1='SYS' THEN CALL ISYS;
   ELSE DO;
        CALL IERR(18);
/*****************************************************/
/*                                                   */
/* ERROR(18)----  'SYS' CARD NOT FOUND. (FATAL)      */
/*                                                   */
/*****************************************************/
        GO TO EOI;
        END;
LOOP:  GET LIST(STR1);
```

```
   IF STR1='MPU' THEN DO;
        CALL IMPU;
        GO TO LOOP;
        END;
   IF STR1='START' THEN DO;
        CALL ISTART;
        GO TO LOOP;
        END;
   IF STR1='TRC' THEN DO;
        CALL ITRC;
        GO TO LOOP;
        END;
   IF STR1='MEM' THEN DO;
        CALL IMEM;
        GO TO LOOP;
        END;
   IF STR1='LOAD' THEN DO;
        CALL ILOAD;
        GO TO LOOP;
        END;
   IF STR1='SIM' THEN DO;
        CALL ISIM;
        GO TO LOOP;
        END;
   IF STR1='END' THEN GO TO EOI;
   CALL IERR(34);
/*******************************************************/
/*                                                     */
/* ERROR(30)---- ILLEGAL CARD TYPE. (FATAL)            */
/*                                                     */
/*******************************************************/
   GO TO LOOP;
EOI: IF ITABL(2,0)='X' THEN DO;
        ITABL(2,0)='L';
        CALL IMEM;
        END;
/*******************************************************/
/*                                                     */
/*  CALL IVAL TO VALIDATE SYSTEM CONFIGURATION.        */
/*                                                     */
/*******************************************************/
   CALL IVAL;
/*******************************************************/
/*                                                     */
/*  CALL ISTORE TO STORE THE SYSTEM INITIAL            */
/*   CONFIGURATION ON THE FILE: SIMSYS.                */
/*                                                     */
/*******************************************************/
```

```
        IF FATAL=0 & ITABL(5,0)='K' THEN CALL ISTORE;
/*****************************************************/
/*                                                   */
/*  CALL ECHO TO PRINT SYSTEM INITIALIZATION         */
/*  CONFIGURATION FOR USER VERIFICATION.             */
/*                                                   */
/*****************************************************/
   CALL ECHO;
   IF LINENO(SYSPRINT)>PL-15 THEN PUT EDIT(' ')(LINE(PL-1),A);
   PUT SKIP(5) EDIT('ERRORS=',FATAL,'WARNINGS=',WARNING)
        (COL(10),A,F(3,0),X(4),A,F(3,0));
   IF FATAL\=0 THEN DO;
        PUT SKIP(2) EDIT('******* SIMULATION NOT RUN BECAUSE '^^
        'OF FATAL ERRORS.')(A);
        PUT SKIP(2) EDIT('******* SYSTEM CONFIGURATION NOT WRITTEN '^^
             'ON FILE SIMSYS.')(A);
        MSTOP='STOP';
        END;
   ELSE DO;
        IF ITABL(5,0)\='K' THEN
             PUT SKIP(2) EDIT('SYSTEM CONFIGURATION NOT KEPT.')
             (COL(10),A);
        PUT SKIP(5) EDIT('******* SIMULATION EXECUTION '^^
             'BEGINS:')(A);
        PUT EDIT(' ')(LINE(PL-4),A);
        END;
/*****************************************************/
/*                                                   */
/*  INITIALIZATION SUBPROGRAMS.                      */
/*                                                   */
/*****************************************************/
ISYS: PROCEDURE;
/*****************************************************/
/*                                                   */
/*  ON UNITS.                                        */
/*                                                   */
/*****************************************************/
   ON ENDFILE(SIMSYS) BEGIN;
        CALL IERR(9);
/*****************************************************/
/*                                                   */
/* ERROR(09)---- END-OF-FILE FOUND ON FILE SIMSYS.   */
/*                                                   */
/*****************************************************/
        GO TO EOF_SIM;
        END;
   ON UNDEFINEDFILE(SIMSYS) BEGIN;
        CALL IERR(13);
```

```
/*****************************************************/
/*                                                   */
/* ERROR(13)---- FILE SIMSYS IS UNDEFINED.           */
/*                                                   */
/*****************************************************/
        GO TO EOF_SIM;
        END;
   STR1=' ';
   CARD='SYS';
   GET LIST(STR1);
   NMPU=1;
   IF VERIFY(STR1,DCML)\=0 THEN CALL IERR(14);
/*****************************************************/
/*                                                   */
/* ERROR(14)---- ILLEGAL INPUT.                      */
/*                                                   */
/*****************************************************/
   ELSE IF STR1\=' ' THEN NMPU=BIN(STR1,31,0);
   IF NMPU<1 ^ NMPU>20 THEN DO;
        CALL IERR(1);
        NMPU=1;
        END;
/*****************************************************/
/*                                                   */
/*  ERROR(1)---- ILLEGAL NUMBER OF MPUS SPECIFIED.   */
/*                                                   */
/*****************************************************/
   MAX=NMPU-1;
   ALLOCATE STAT(0:MAX),PC(0:MAX),SP(0:MAX),INR(0:MAX),ACCA(0:MAX),
        ACCB(0:MAX),CCR(0:MAX),SYSTBL(0:10,0:MAX),
        TRCE(3,0:MAX),ITABL(6,0:MAX),MCT(0:MAX);
   CALL DEFAULT;
   STR1='NEW';
   GET LIST(STR1);
   IF STR1\='OLD' THEN
        IF STR1\='NEW' THEN CALL IERR(11);
/*****************************************************/
/*                                                   */
/* ERROR(11)---- ILLEGAL SYSTEM INITIALIZATION       */
/*               SPECIFICATION.  DEFAULT ASSUMED.    */
/*                                                   */
/*****************************************************/
        ELSE;
   ELSE DO;
        ITABL(4,0)='0';
        OPEN FILE(SIMSYS) INPUT;
        READ FILE(SIMSYS) INTO(SYSREC);
        MAX=SDATA(0);
```

```
        DO I=0 TO 10;
             READ FILE(SIMSYS) INTO(SYSREC);
             DO MPU=0 TO MAX;
                  IF MPU<NMPU THEN DO;
                       SYSTBL(I,MPU)=SDATA(MPU);
                       ITABL(1,MPU)='0';
                       END;
                  END;
             END;
        END;
EOF_SIM: STR1='DELETE';
   GET LIST(STR1);
   IF STR1\='KEEP' THEN
        IF STR1\='DELETE' THEN CALL IERR(12);
/*******************************************************/
/*                                                     */
/* ERROR(12)----ILLEGAL SYSTEM DISPOSITION             */
/*              SPECIFICATION.  DELETE ASSUMED.        */
/*                                                     */
/*******************************************************/
        ELSE;
   ELSE ITABL(5,0)='K';
   GET LIST(STR1);
   IF STR1\='*' THEN CALL IERR(2);
/*******************************************************/
/*                                                     */
/* ERROR(02)---- '*' NOT FOUND.                        */
/*                                                     */
/*******************************************************/
   RETURN;
   END ISYS;
DEFAULT:  PROCEDURE;
/*******************************************************/
/*                                                     */
/*  THIS PROCEDURE INITIALIZES THE SYSTEM TO           */
/*  DEFAULT SPECIFICATIONS.                            */
/*                                                     */
/*******************************************************/
   SYSTBL=0;
   TRCE=0;
   DO I=0  TO NMPU-1;
        SYSTBL(6,I)=2;
        SYSTBL(7,I)=00010000000000000000000000000000B;
        SYSTBL(1,I)=1023;
        TRCE(1,I)=4;
        ITABL(1,I)='X';
        ITABL(6,I)='R';
        END;
```

```
   ITABL(2,0)='X';
   ITABL(4,0)='N';
   ITABL(5,0)='D';
   ITABL(3,0)='X';
   INR=0;
   INST,INPT,OTPT,ERRS,WARN,BRCH,INTS,RTRN,RSTS=0;
   HALT,RUN,RESTRT,LSTHLT=0;
   NTPER=0;
   PRCT=1;
   SP=0;
   ACCA,ACCB=0;
   CCR='11000000'B;
   PC=0;
   RETURN;
   END DEFAULT;
IMEM: PROCEDURE;
/*********************************************************/
/*                                                       */
/*  THIS PROCEDURE ALLOCATES AND PREFORMATS THE          */
/*  FILE 'MEM1' SO THAT IT MAY BE INITIALIZED            */
/*  IF OLD IS SPECIFIED, THEN THE FILE 'SIMSYS' IS       */
/*  READ AND TRANSFERED INTO 'MEM1.'                     */
/*                                                       */
/*  THIS SUBPROGRAM ALSO READS THE 'MEM' CARD.           */
/*                                                       */
/*  'MEM' CARD FORMAT:                                   */
/*                                                       */
/*       1.  'MEM'                                       */
/*       2.   '(INITIALIZATION SPECIFICATION)'           */
/*       3.   '*'                                        */
/*                                                       */
/*********************************************************/
   ON ENDFILE(SIMSYS) BEGIN;
        CALL IERR(8);
        GO TO ERROR_RET;
        END;
/*********************************************************/
/*                                                       */
/*  ERROR(08)---- EOF FOUND IN FILE 'SIMSYS.'            */
/*                                                       */
/*********************************************************/
   ON UNDEFINEDFILE(SIMSYS) BEGIN;
        CALL IERR(13);
        GO TO ERROR_RET;
        END;
/*********************************************************/
/*                                                       */
/*  ERROR(13)---- UNDEFINED FILE 'SIMSYS.'               */
```

```
/*                                                    */
/******************************************************/
   ON KEY(MEM1) BEGIN;
        STR1=ONKEY;
        CALL IERR(30);
        GO TO ERROR_RET;
        END;
/******************************************************/
/*                                                    */
/*  ERROR(30)---- KEY ERROR ON FILE(MEM1).            */
/*                                                    */
/******************************************************/
/*                                                    */
/*  CALCULATE THE TOTAL AMOUNT OF MEMORY AND THE      */
/*  TOTAL NUMBER OF PERIPHERALS IN THE SYSTEM.        */
/*                                                    */
/******************************************************/
   MAX=-1;
   CARD='MEM';
   MEMTYPE='CORE';
   IF ITABL(2,0)\='L' THEN GET LIST(MEMTYPE);
   IF MEMTYPE\='DISK' & MEMTYPE\='CORE' THEN DO;
        CALL IERR(31);
        MEMTYPE='CORE';
        END;
/******************************************************/
/*                                                    */
/*  ERROR(31)---- ILLEGAL MEMORY FILE TYPE SPEC.      */
/*                'CORE' ASSUMED.                     */
/*                                                    */
/******************************************************/
   DO MPU=0 TO NMPU-1;
        SYSTBL(2,MPU)=MAX +1;
        MAX=MAX+CEIL((SYSTBL(1,MPU)+1)/32);
        NPER=NPER+SYSTBL(6,MPU);
        END;
   LMPU=0;
   IF NPER>0 THEN DO;
/******************************************************/
/*                                                    */
/*  ALLOCATE THE PERIPHERAL TABLE.                    */
/*                                                    */
/******************************************************/
        ALLOCATE PERTBL(0:NPER);
        CTYPE,C1,C2='0';
        PCODE,PMPU,PADDR,PASADD,PASSOC=0;
        DDR,PDR='0';
        END;
```

```
   LOC='0'B;
   IF MEMTYPE='DISK' THEN DO;
        OPEN FILE(MEM1) SEQL OUTPUT;
        DO ADDRESS=0 TO MAX;
/*****************************************************/
/*                                                   */
/*  INITIALIZE THE MEMORY TO ALL RAM.                */
/*                                                   */
/*****************************************************/
             MEM_KEY=DEC(ADDRESS,7,0);
             IF LMPU<NMPU-1 THEN
                  IF ADDRESS=SYSTBL(2,LMPU+1) THEN LMPU=LMPU+1;
                  WRITE FILE(MEM1) FROM(MEMREC) KEYFROM(MEM_KEY);
             END;
        CLOSE FILE(MEM1);
        OPEN FILE(MEM1) DIRECT UPDATE;
        END;
   ELSE DO;
        ALLOCATE CMEM(0:MAX);
        CLOC=0;
        MPU=0;
        OLDADD=1;
        DO ADDRESS=0 TO MAX;
             CMPU(ADDRESS)=MPU;
             IF MPU<NMPU-1 THEN IF ADDRESS=SYSTBL(2,MPU+1)
                  THEN MPU=MPU+1;
             END;
        END;
   FLAG='0'B;
   STR1='NEW';
   IF ITABL(2,0)\='L' THEN  GET LIST(STR1);
/*****************************************************/
/*                                                   */
/*  IF 'OLD' IS SPECIFIED THEN THE MEMORY IS TO BE   */
/*  INITIALIZED FROM THE FILE 'SIMSYS.'              */
/*                                                   */
/*****************************************************/
   IF STR1='OLD' THEN DO;
        READ FILE(SIMSYS) INTO(TMEMREC);
        IF TMEM_KEY='***' THEN DO;
             MAX=TLMPU;  /*  TLMPU= NUMBER OF ENTRIES IN SIMSYS  */
             MCT=0;
             DO I=0 TO MAX;
                  READ FILE(SIMSYS) INTO(TMEMREC);
                  IF VERIFY(TMEM_KEY,DCML)=0 THEN DO;
                       IF TLMPU>=0 & TLMPU<NMPU THEN DO;
                            MPU=TLMPU;
                            J=MFIND(MCT(MPU),1);
```

```
                    IF J>=0 THEN DO J=0 TO 31;
                         LOC(J)=TLOC(J);
                         END;
                    MCT(TLMPU)=MCT(TLMPU)+32;
                    END;
               END;
          END;
     READ FILE(SIMSYS) INTO(TMEMREC);
     IF TMEM_KEY='***' THEN DO;
          MAX=TLMPU;
          DO MPU=0 TO MAX;
               READ FILE(SIMSYS) INTO(TMEMREC);
               K=0;
               DO ADDRESS=65528 TO 65535;
                    J=MFIND(ADDRESS,1);
                    LOC(J)=TLOC(K);
                    K=K+1;
                    END;
               END;
          END;
     READ FILE(SIMSYS) INTO(TMEMREC);
     IF TMEM_KEY='***' THEN DO;
/*****************************************************/
/*                                                   */
/*  GET OLD PERIPHERAL SPECS FROM THE FILE 'SIMSYS'  */
/*  AND LOAD THEM INTO THE TEMPORARY TABLE           */
/*  'TPERTBL.'                                       */
/*                                                   */
/*****************************************************/
          MAX=TLMPU;  /*  TLMPU= NUMBER OF ENTRIES  */
          ALLOCATE TPERTBL(MAX*3);
          K=1;
          NTPER=MAX*3;
          TMPU,TCODE=0;
          TCTYPE='0'B;
          DO I=1 TO MAX;
               READ FILE(SIMSYS) INTO(PERREC);
               DO J=1 TO 3;
                    IF K<=NTPER THEN DO;
                         TMPU(K)=RMPU(J);
                         TADDR(K)=RADDR(J);
                         TASSOC(K)=RASSOC(J);
                         TASADD(K)=RASADD(J);
                         TCODE(K)=RCODE(J);
                         TCTYPE(K)=RCTYPE(J);
                         TC2(K)=RC2(J);
                         TC1(K)=RC1(J);
                         TDDR(K)=RDDR(J);
```

```
                                    TPDR(K)=RPDR(J);
                                    K=K+1;
                                    END;
                               END;
                          END;
                     END;
                ELSE CALL IERR(17);
/*****************************************************/
/*                                                   */
/*  ERROR(17)---- INCORRECT FILE FORMAT ON FILE      */
/*                 'SIMSYS.'                         */
/*                                                   */
/*****************************************************/
ERROR_RET: END;
        ELSE CALL IERR(17);
/*****************************************************/
/*                                                   */
/*  ERROR(17)---- INCORRECT FILE FORMAT ON FILE      */
/*                 'SIMSYS.'                         */
/*                                                   */
/*****************************************************/
        CLOSE FILE(SIMSYS);
        END;
   ELSE IF STR1\='NEW' THEN CALL IERR(21);
/*****************************************************/
/*                                                   */
/*  ERROR(21)---- ILLEGAL MEMORY INITIALIZATION      */
/*                 SPECIFICATION.                    */
/*                                                   */
/*****************************************************/
   IF ITABL(2,0)\='L' THEN GET LIST(STR1);
   ITABL(2,0)='I';
   IF STR1\='*' THEN CALL IERR(02);
/*****************************************************/
/*                                                   */
/*  ERROR(02)---- '*' NOT FOUND WHEN EXPECTED.       */
/*                                                   */
/*****************************************************/
   RETURN;
   END IMEM;
ITRC:  PROCEDURE;
   CARD='TRC';
   STR1='X';
   GET LIST(STR1);
   FLAG='0'B;
   IF VERIFY(STR1,DCML)=0 THEN DO;
        MPU=BIN(STR1,31,0);
        IF MPU<0 ^ MPU>=NMPU THEN DO;
```

```
            CALL IERR(4);
            FLAG='1'B;
            END;
/*****************************************************/
/*                                                   */
/*  ERROR(04)---- ILLEGAL MPU NUMBER SPECIFIED.      */
/*                                                   */
/*****************************************************/
         END;
   ELSE DO;
         CALL IERR(14);
         END;
/*****************************************************/
/*                                                   */
/*  ERROR(14)---- ILLEGAL INPUT.                     */
/*                                                   */
/*****************************************************/
   STR1='X';
   GET LIST(STR1);
   IF VERIFY(STR1,DCML)=0 THEN DO;
         I=BIN(STR1,31,0);
         IF I<0 ^ I>4 THEN CALL IERR(23);
/*****************************************************/
/*                                                   */
/*  ERROR(23)---- ILLEGAL TRACE TYPE SPECIFIED.      */
/*                                                   */
/*****************************************************/
         ELSE IF FLAG='0'B THEN TRCE(1,MPU)=I;
         END;
   ELSE IF STR1\='X' THEN CALL IERR(14);
/*****************************************************/
/*                                                   */
/*  ERROR(14)---- ILLEGAL INPUT.                     */
/*                                                   */
/*****************************************************/
   IF TRCE(1,MPU)\=1 THEN
         IF TRCE(1,MPU)\=2 THEN
               GET LIST(STR1,STR1);
         ELSE GET LIST((TRCE(I,MPU) DO I=2 TO 3));
   ELSE DO I=2 TO 3;
         STR1=' ';
         GET LIST(STR1);
         IF VERIFY(STR1,HEX)\=0 THEN CALL IERR(6);
/*****************************************************/
/*                                                   */
/*  ERROR(06)---- ILLEGAL ADDRESS SPECIFIED.         */
/*                                                   */
/*****************************************************/
```

```
        ELSE TRCE(I,MPU)=BBIN(STR1);
        END;
   GET LIST(STR1);
   IF STR1\='*' THEN CALL IERR(2);
/*******************************************************/
/*                                                     */
/*  ERROR(02)---- '*' NOT FOUND WHEN EXPECTED.         */
/*                                                     */
/*******************************************************/
   RETURN;
   END ITRC;
ISTART: PROCEDURE;
   CARD='START';
   STR1='X';
   FLAG='0'B;
   ATRIB(1)=TTBEG;
   GET LIST(STR1);
   IF VERIFY(STR1,DCML)=0 THEN DO;
        MPU=BIN(STR1,31,0);
        IF MPU<0 ^ MPU>=NMPU THEN DO;
             CALL IERR(4);
             FLAG='1'B;
             END;
/*******************************************************/
/*                                                     */
/*  ERROR(04)---- ILLEGAL MPU NUMBER.                  */
/*                                                     */
/*******************************************************/
        END;
   ELSE DO;
        CALL IERR(14);
        FLAG='1'B;
        END;
/*******************************************************/
/*                                                     */
/*  ERROR(14)---- ILLEGAL INPUT.                       */
/*                                                     */
/*******************************************************/
   STR1='RESET';
   GET LIST(STR1);
   IF STR1\='RESET' & FLAG='0'B THEN
        IF VERIFY(STR1,HEX)\=0 THEN
             CALL IERR(6);
/*******************************************************/
/*                                                     */
/*  ERROR(06)---- ILLEGAL ADDRESS SPECIFIED.           */
/*                                                     */
/*******************************************************/
```

```
        ELSE DO;
             PC(MPU)=BBIN(STR1);
             ATRIB(2)=MPU+10;
             END;
ELSE IF FLAG='0'B THEN ATRIB(2)=1200+MPU;
   IF FLAG='0'B THEN CALL FILEM(1);
   GET LIST(STR1);
   IF STR1\='*' THEN CALL IERR(2);
/*******************************************************/
/*                                                     */
/*  ERROR(02)---- '*' NOT FOUND WHEN EXPECTED.         */
/*                                                     */
/*******************************************************/
   RETURN;
   END ISTART;
IMPU:  PROCEDURE;
   CARD='MPU';
   STR1='X';
   FLAG='0';
   GET LIST(STR1);
   MPU=0;
   IF VERIFY(STR1,DCML)=0 THEN DO;
        TEMP=BIN(STR1,31,0);
        IF TEMP<0 ^ TEMP>=NMPU THEN DO;
             CALL IERR(4);
             FLAG='1'B;
             END;
/*******************************************************/
/*                                                     */
/*  ERROR(04)---- ILLEGAL MPU NUMBER SPECIFIED.        */
/*                                                     */
/*******************************************************/
        ELSE MPU=TEMP;
        END;
   ELSE CALL IERR(14);
/*******************************************************/
/*                                                     */
/*  ERROR(14)---- ILLEGAL INPUT.                       */
/*                                                     */
/*******************************************************/
   STR1=' ';
   IF ITABL(2,0)='I' THEN DO;
        FLAG='1'B;
        CALL IERR(16);
        END;
/*******************************************************/
/*                                                     */
/*  ERROR(16)---- MPU CARD FOUND AFTER MEMORY          */
```

```
/*                INITIALIZED.                          */
/*                                                      */
/******************************************************/
   GET LIST(STR1);
   IF STR1\=' ' & FLAG='0'B THEN
        IF VERIFY(STR1,DCML)=0 THEN DO;
             TEMP=BIN(STR1,31,0);
             IF TEMP<1 ^ TEMP>65536 THEN CALL IERR(5);
/******************************************************/
/*                                                      */
/*  ERROR(5)---- ILLEGAL MEMORY SIZE SPECIFIED.         */
/*                                                      */
/******************************************************/
             ELSE SYSTBL(1,MPU)=TEMP-1;
             END;
        ELSE CALL IERR(14);
/******************************************************/
/*                                                      */
/*  ERROR(14)---- ILLEGAL INPUT.                        */
/*                                                      */
/******************************************************/
   ELSE;
   STR1=' ';
   GET LIST(STR1);
   IF STR1\=' ' & FLAG='0'B THEN
        IF VERIFY(STR1,DCML)=0 THEN DO;
             TEMP=BIN(STR1,31,0);
             IF TEMP<0 THEN CALL IERR(7);
/******************************************************/
/*                                                      */
/*  ERROR(07)---- ILLEGAL NUMBER OF PERIPHERALS         */
/*                 SPECIFIED.                           */
/*                                                      */
/******************************************************/
             ELSE SYSTBL(6,MPU)=TEMP;
                  END;
             ELSE CALL IERR(14);
/******************************************************/
/*                                                      */
/*  ERROR(14)---- ILLEGAL INPUT.                      */
/*                                                      */
/******************************************************/
   ELSE;
   GET LIST(STR1);
   IF STR1\='*' THEN CALL IERR(2);
/******************************************************/
/*                                                      */
/*  ERROR(02)---- '*' NOT FOUND WHEN EXPECTED.          */
```

```
/*                                                          */
/************************************************************/
   RETURN;
   END IMPU;
ILOAD:  PROCEDURE;
/************************************************************/
/*                                                          */
/*  THIS PROCEDURE LOADS THE MEMORY FROM THE                */
/*  LOAD CARD.                                              */
/*                                                          */
/************************************************************/
   CARD='LOAD';
   IF ITABL(2,0)='X' THEN DO;
        ITABL(2,0)='L';
        CALL IMEM;
        END;
   MPU,ADDRESS=0;
   FLAG='1';
   TYPE='000'B;
   END_OF_CARD='NO';
   DO WHILE(END_OF_CARD='NO');
   STR1=' ';
   I=1;
   GET LIST(STR1);
   IF STR1\='MPU' THEN
        IF STR1\='ADDR' THEN
             IF STR1\='RAM' THEN
                  IF STR1\='ROM' THEN I=-1;
                  ELSE TYPE='001'B;
             ELSE TYPE='000'B;
        ELSE DO;
             GET LIST(STR1);
             IF VERIFY(STR1,HEX)\=0 THEN CALL IERR(14);
/************************************************************/
/*                                                          */
/*  ERROR(14)---- ILLEGAL INPUT.                            */
/*                                                          */
/************************************************************/
             ELSE DO;
                  TEMP=BBIN(STR1);
                  J=MFIND(TEMP,0);
                  IF J<0 THEN CALL IERR(6);
/************************************************************/
/*                                                          */
/*  ERROR(06)---- ILLEGAL ADDRESS SPECIFIED.                */
/*                                                          */
/************************************************************/
                  ELSE ADDRESS=TEMP;
```

```
                END;
            END;
   ELSE DO;
        GET LIST(STR1);
        IF VERIFY(STR1,DCML)\=0 THEN CALL IERR(14);
/*****************************************************/
/*                                                   */
/*  ERROR(14)---- ILLEGAL INPUT.                     */
/*                                                   */
/*****************************************************/
        ELSE DO;
             TEMP=BIN(STR1,31,0);
             IF TEMP<0 ^ TEMP>=NMPU THEN CALL IERR(4);
/*****************************************************/
/*                                                   */
/*  ERROR(04)---- ILLEGAL MPU NUMBER SPECIFIED.      */
/*                                                   */
/*****************************************************/
             ELSE DO;
                  MPU=TEMP;
                  TYPE='000'B;
                  ADDRESS=0;
                  END;
             END;
        END;
   IF I<0 THEN
        IF STR1\='PIAA' & STR1\='PIAB' THEN
             IF STR1\='ACIA' THEN DO;
                  J,MAX=ADDRESS;
                  IF STR1='CONT' THEN DO;
                       GET LIST(STR1);
                       IF VERIFY(STR1,HEX)\=0 THEN CALL IERR(14);
/*****************************************************/
/*                                                   */
/*  ERROR(14)---- ILLEGAL INPUT.                     */
/*                                                   */
/*****************************************************/
                       ELSE DO;
                            MAX=BBIN(STR1);
                            IF MAX<ADDRESS ^ MAX>SYSTBL(1,MPU)-8 THEN
                                 DO;
                                 CALL IERR(28);
                                 MAX=ADDRESS;
                                 END;
/*****************************************************/
/*                                                   */
/*  ERROR(28)---- ILLEGAL CONTINUATION ADDRESS.      */
/*                                                   */
```

```
/*****************************************************/
                         STR1='000000';
                         END;
                    END;
               IF STR1\='*' THEN DO ADDRESS=J TO MAX;
                    TEMP=MFIND(ADDRESS,1);
                    IF TEMP<0 THEN CALL IERR(20);
/*****************************************************/
/*                                                   */
/*  ERROR(20)---- ATTEMPT TO INITIALIZE AN           */
/*                 ILLEGAL MEMORY ADDRESS.           */
/*                                                   */
/*****************************************************/
                    ELSE LOC(TEMP)=TYPE^^'00000'B^^
                         SUBSTR(BBIN(STR1),9,8);
                    END;
               ELSE END_OF_CARD='YES';
               END;
          ELSE DO;
/*************************************************************/
/*                                                           */
/*  INPUT ACIA LOCATION                                      */
/*                                                           */
/*  FORMAT:                                                  */
/*                                                           */
/*      1.  'ACIA'                                           */
/*      2.  '(ASSOCIATE ADDRESS)'  (IN HEX)                  */
/*      3.  '(ASSOCIATE NUMBER)'                             */
/*                                                           */
/*************************************************************/
               IF SYSTBL(6,MPU)=0 THEN DO;
                    CALL IERR(35);
                    GET LIST(STR1,STR1);
                    END;
               IF PRCT>NPER THEN DO;
                    CALL IERR(36);
/*************************************************************/
/*                                                           */
/*  ERROR(36)---- ATTEMPT TO INITIALIZE MORE PERIPHERALS     */
/*                 THAN SPECIFIED FOR THE SYSTEM.            */
/*                                                           */
/*************************************************************/
                    PRCT=NPER;
                    END;
               ELSE DO;
               TEMP=MFIND(ADDRESS,1);
               IF TEMP<=0 THEN DO;
                    CALL IERR(20);
```

```
                    GET LIST(STR1,STR1);
                    END;
/*****************************************************/
/*                                                   */
/*  ERROR(20)---- ATTEMPT TO INITIALIZE AN ILLEGAL   */
/*                 MEMORY ADDRESS.                   */
/*                                                   */
/*****************************************************/
                ELSE DO;
                    LOC(TEMP)='0100'B^^SUBSTR(BIT(PRCT,31),24,8)^^
                        '0000'B;
                    PADDR(PRCT)=ADDRESS;
                    ADDRESS=ADDRESS+1;
                    TEMP=MFIND(ADDRESS,1);
                    IF TEMP<0 THEN DO;
                        CALL IERR(20);
                        GET LIST(STR1,STR1);
                        END;
/*****************************************************/
/*                                                   */
/*  ERROR(20)---- ATTEMPT TO INITIALIZE AN           */
/*                 ILLEGAL MEMORY ADDRESS.           */
/*                                                   */
/*****************************************************/
                    ELSE DO;
                        LOC(TEMP)='011'B^^(13)'0'B;
                        ADDRESS=ADDRESS+1;
/*************************************************************/
/*                                                           */
/*  INPUT ASSOCIATE ADDRESS                                  */
/*                                                           */
/*************************************************************/
                        GET LIST(STR1);
                        IF VERIFY(STR1,HEX)\=0 THEN CALL IERR (32);
/*****************************************************/
/*                                                   */
/*  ERROR(32)---- ILLEGAL ASSOCIATE ADDRESS.         */
/*                                                   */
/*****************************************************/
                        ELSE TEMP=BBIN(STR1);
/*************************************************************/
/*                                                           */
/*  INPUT ASSOCIATE NUMBER                                   */
/*                                                           */
/*************************************************************/
                        GET LIST(STR1);
                        IF VERIFY(STR1,DCML)\=0 THEN DO;
                            CALL IERR(33);
```

```
                                    PASSOC(PRCT)=-1;
                                    END;
/*****************************************************/
/*                                                   */
/*   ERROR(33)----  ILLEGAL ASSOCIATE DEVICE.        */
/*                                                   */
/*****************************************************/
                               ELSE PASSOC(PRCT)=BIN(STR1,31,0);
                               PCODE(PRCT)=3;
                               PASADD(PRCT)=TEMP;
                               PMPU(PRCT)=MPU;
                               PRCT=PRCT+1;
                               END;
                          END;
                     END;
                     END;
          ELSE DO;
/*****************************************************/
/*                                                   */
/*  INPUT PIA LOCATIONS                              */
/*                                                   */
/*  FORMAT:                                          */
/*                                                   */
/*     1.  'PIAA' OR 'PIAB'                          */
/*     2.  '(ASSOCIATE ADDRESS)'                     */
/*     3.  '(ASSOCIATE NUMBER)'                      */
/*     4.  '(CONNECTION TYPE)'                       */
/*          0:  CA1<->CB1 & CA2<->CB2                */
/*          1:  CA2<->CB1 & CA1<->CB2                */
/*                                                   */
/*****************************************************/
               IF SYSTBL(6,MPU)=0 THEN DO;
                    CALL IERR(35);
                    GET LIST(STR1,STR1);
                    END;
               IF PRCT>NPER THEN DO;
                    CALL IERR(36);
/*****************************************************/
/*                                                   */
/*  ERROR(36)---- ATTEMPT TO INITIALIZE MORE         */
/*             PERIPHERALS THAN SPECIFIED.           */
/*                                                   */
/*****************************************************/
                    PRCT=NPER;
                    END;
               ELSE DO;
               IF STR1='PIAA' THEN PCODE(PRCT)=1;
               ELSE PCODE(PRCT)=2;
```

```
          TEMP=MFIND(ADDRESS,1);
          IF TEMP<0 THEN DO;
               CALL IERR(20);
               GET LIST(STR1,STR1);
               END;
/*********************************************************/
/*                                                       */
/*  ERROR(20)---- ATTEMPT TO INITIALIZE AN ILLEGAL       */
/*                 MEMORY LOCATION.                      */
/*                                                       */
/*********************************************************/
          ELSE DO;
               LOC(TEMP)='0100'B^^SUBSTR(BIT(PRCT,31),24,8)
                    ^^'0000B';
               PADDR(PRCT)=ADDRESS;
               ADDRESS=ADDRESS+1;
               TEMP=MFIND(ADDRESS,1);
               IF TEMP<0 THEN DO;
                    CALL IERR(20);
                    GET LIST(STR1,STR1);
                    END;
/*********************************************************/
/*                                                       */
/*  ERROR(20)---- ATTEMPT TO INITIALIZE AN ILLEGAL       */
/*                 MEMORY LOCATION.                      */
/*                                                       */
/*********************************************************/
               ELSE DO;
                    LOC(TEMP)='011011'B^^(10)'0'B;
                    ADDRESS=ADDRESS+1;
/*********************************************************/
/*                                                       */
/*  INPUT ASSOCIATE ADDRESS                              */
/*                                                       */
/*********************************************************/
                    GET LIST(STR1);
                    IF VERIFY(STR1,HEX)\=0 THEN
                         CALL IERR(32);
/*********************************************************/
/*                                                       */
/*  ERROR(32)---- ILLEGAL ASSOCIATE ADDRESS.             */
/*                                                       */
/*********************************************************/
                    ELSE TEMP=BBIN(STR1);
/*********************************************************/
/*                                                       */
/*  INPUT ASSOCIATE NUMBER                               */
/*                                                       */
```

```
/*********************************************************/
                        GET LIST(STR1);
                        IF VERIFY(STR1,DCML)\=0 THEN DO;
                             CALL IERR(33);
                             PASSOC(PRCT)=-1;
                             END;
/*********************************************************/
/*                                                       */
/*  ERROR(33)---- ILLEGAL ASSOCIATE DEVICE.             */
/*                                                       */
/*********************************************************/
                        ELSE PASSOC(PRCT)=
                             BIN(STR1,31,0);
                        PASADD(PRCT)=TEMP;
                        PMPU(PRCT)=MPU;
/*********************************************************/
/*                                                       */
/*  INPUT CONNECTION TYPE.                               */
/*                                                       */
/*********************************************************/
                        GET LIST(STR1);
                        IF VERIFY(STR1,DCML)\=0 THEN CALL IERR(14);
/*********************************************************/
/*                                                       */
/*  ERROR(14)---- ILLEGAL INPUT.                         */
/*                                                       */
/*********************************************************/
                        ELSE IF STR1='1' THEN CTYPE(PRCT)='1'B;
                        PRCT=PRCT+1;
                        END;
                   END;
              END;
         END;
   ELSE;
   END;
   END ILOAD;
ISTORE: PROCEDURE;
/*********************************************************/
/*                                                       */
/*  THIS PROCEDURE STORES THE INITIAL SYSTEM             */
/*  CONFIGURATION ON THE FILE: 'SIMSYS.'                 */
/*                                                       */
/*********************************************************/
/*                                                       */
/*      ON UNITS.....                                    */
/*                                                       */
/*********************************************************/
   ON KEY(MEM1) BEGIN;
```

```
        CALL IERR(30);
        PUT SKIP(2) EDIT('KEY='^^ONKEY^^';  ERROR CODE='^^
             ONCODE)(COL(20),A);
        END;
/*******************************************************/
/*                                                     */
/*  ERROR(30)---- KEY ERROR ON FILE MEM1.              */
/*                                                     */
/*******************************************************/
   MAX=SYSTBL(2,NMPU-1)+CEIL(SYSTBL(1,NMPU-1)/32);
PUT SKIP DATA(SYSTBL(1,NMPU-1));
/*******************************************************/
/*                                                     */
/*  WRITE SYSTEM CONFIGURATION ON FILE 'SIMSYS.'       */
/*                                                     */
/*******************************************************/
   OPEN FILE(SIMSYS) SEQL OUTPUT;
   SDATA(0)=NMPU-1;
   WRITE FILE(SIMSYS) FROM(SYSREC);
   DO I=0 TO 10;
        DO MPU=0 TO NMPU-1;
             SDATA(MPU)=SYSTBL(I,MPU);
             END;
        WRITE FILE(SIMSYS) FROM(SYSREC);
        END;
/*******************************************************/
/*                                                     */
/*  WRITE MEMORY CONFIGURATION ON FILE 'SIMSYS.'       */
/*                                                     */
/*******************************************************/
   TMEM_KEY='***';
   TLMPU=MAX;
   WRITE FILE(SIMSYS) FROM(TMEMREC);
   DO I=0 TO MAX;
        TMEM_KEY=DEC(I,7,0);
        IF MEMTYPE='DISK' THEN DO;
             READ FILE(MEM1) INTO(TMEMREC) KEY(TMEM_KEY);
             END;
        ELSE DO;
             TLMPU=CMPU(I);
             DO  J= 0 TO 31;
                  TLOC(J)=CLOC(I,J);
                  END;
             END;
        WRITE FILE(SIMSYS) FROM(TMEMREC);
        END;
/*******************************************************/
/*                                                     */
```

```
/*  WRITE INTERRUPT VECTOR LOCATIONS                    */
/*                                                      */
/******************************************************/
   TMEM_KEY='***';
   TLMPU=NMPU-1;
   WRITE FILE(SIMSYS) FROM(TMEMREC);
   DO MPU=0 TO NMPU-1;
        J=0;
        DO ADDRESS=65528 TO 65535;
             K=MFIND(ADDRESS,1);
             TLOC(J)=LOC(K);
             J=J+1;
             END;
        TLMPU=MPU;
        WRITE FILE(SIMSYS) FROM(TMEMREC);
        END;
/******************************************************/
/*                                                      */
/*  WRITE PERIPHERAL CONFIGURATION ON FILE SIMSYS.      */
/*                                                      */
/******************************************************/
   IF NPER>0 THEN DO;
   TMEM_KEY='***';
   TLMPU=CEIL(NPER/3)+1;
   WRITE FILE(SIMSYS) FROM(TMEMREC);
   K=1;
   DO I=1 TO TLMPU;
        DO J=1 TO 3;
             RMPU(J)=PMPU(K);
             RADDR(J)=PADDR(K);
             RASSOC(J)=PASSOC(K);
             RASADD(J)=PASADD(K);
             RCODE(J)=PCODE(K);
             RCTYPE(J)=CTYPE(K);
             RC2(J)=C2(K);
             RC1(J)=C1(K);
             RDDR(J)=DDR(K);
             RPDR(J)=PDR(K);
             IF K=NPER THEN J=4;
             ELSE K=K+1;
             END;
        WRITE FILE(SIMSYS) FROM(PERREC);
        END;
   END;
   CLOSE FILE(SIMSYS);
   RETURN;
   END ISTORE;
ISIM: PROCEDURE;
```

```
   STR1='X';
  CARD='SIM';
   GET LIST(STR1);
   IF VERIFY(STR1,DCML)\=0 THEN DO;
        CALL IERR(14);
        GET LIST(TEMP);
        END;
/*********************************************************/
/*                                                       */
/*  ERROR(14)---- ILLEGAL INPUT.                         */
/*                                                       */
/*********************************************************/
   ELSE DO;
        MPU=BIN(STR1,31,0);
        IF MPU<0 ^ MPU>=NMPU THEN DO;
             CALL IERR(4);
             GET LIST(TEMP);
             END;
/*********************************************************/
/*                                                       */
/*  ERROR(04)---- ILLEGAL MPU NUMBER.                    */
/*                                                       */
/*********************************************************/
        ELSE GET LIST(SYSTBL(0,MPU));
        END;
   GET LIST(STR1);
   IF STR1\='*' THEN CALL IERR(2);
/*********************************************************/
/*                                                       */
/* ERROR(02)---- '*' NOT FOUND WHEN EXPECTED.            */
/*                                                       */
/*********************************************************/
   RETURN;
   END ISIM;
IVAL: PROCEDURE;
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE TRANSFERS THE PERIPHERALS         */
/*  READ IN FROM THE FILE 'SIMSYS' INTO THE PERTBL.  */
/*  ONLY PERIPHERALS THAT STILL REMAIN IN THE        */
/*  MEMORY ARE TRANSFERRED.                          */
/*                                                   */
/*****************************************************/
   IF NTPER>0 THEN DO MPU=0 TO NMPU-1;
        DO I=1 TO NTPER;
             IF TMPU(I)=MPU THEN DO;
                  IF TCODE(I)\=0 THEN DO;
                       J=MFIND(TADDR(I),1);
```

```
                    IF J>=0 THEN
                         IF SUBSTR(LOC(J),1,3)='010' THEN DO;
                         FLAG='0'B;
                         DO K=0 TO NPER WHILE(FLAG='0'B);
                              IF TADDR(I)=PADDR(K) THEN
                                   IF TMPU(I)=PMPU(K) THEN
                                        FLAG='1'B;
                              END;
                         IF FLAG='0' THEN DO;
                              IF PRCT>NPER THEN CALL IERR(27);
/*****************************************************/
/*                                                   */
/*  ERROR(27)---- ATTEMPT TO INITIALIZE MORE         */
/*                 PERIPHERALS THAN SPECIFIED.       */
/*                                                   */
/*****************************************************/
                              ELSE DO;
                                   PMPU(PRCT)=TMPU(I);
                                   PADDR(PRCT)=TADDR(I);
                                   PASSOC(PRCT)=TASSOC(I);
                                   PASADD(PRCT)=TASADD(I);
                                   PCODE(PRCT)=TCODE(I);
                                   CTYPE(PRCT)=TCTYPE(I);
                                   C2(PRCT)=TC2(I);
                                   C1(PRCT)=TC1(I);
                                   DDR(PRCT)=TDDR(I);
                                   PDR(PRCT)=TPDR(I);
                                   SUBSTR(LOC(J),5,8)=
                                        SUBSTR(PRCT,24,8);
                                   PRCT=PRCT+1;
                                   END;
                              END;
                         END;
                    END;
               TMPU(I)=-1;
               END;
          END;
     END;
  END IVAL;
IERR:  PROCEDURE(CODE);
  DCL CODE FIXED BIN(31,0);
  PUT SKIP(2) EDIT('INITIALIZATION ERROR: ',CODE,STR1,'IN CARD: '^^
      CARD)(COL(10),A,F(3,0),X(3),A,A);
  IF CODE=9 ^ CODE=12 ^ CODE=13 ^ CODE=23 ^ CODE=26 ^ CODE=29 THEN
       WARNING=WARNING +1;
  ELSE FATAL=FATAL+1;
  END IERR;
  END INTLC;
```

```
/*   IOEVNT   VERSION-1              6/16/78       */
IOEVNT:  PROCEDURE;
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE TAKES CARE OF THE I/O EVENTS      */
/*  THAT OCCUR BETWEEN PERIPHERAL DEVICES.           */
/*                                                   */
/*****************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL 1 PERTBL(*) CTL EXT,
        2 PMPU FIXED BIN(31,0),
        2 PADDR FIXED BIN(31,0),
        2 PASSOC FIXED BIN(31,0),
        2 PASADD FIXED BIN(31,0),
        2 PCODE FIXED BIN(31,0),
        2 CTYPE BIT(1),
        2 UNUSED BIT(5),
        2 C2 BIT(1),
        2 C1 BIT(1),
        2 DDR BIT(8),
        2 PDR BIT(8);
   DCL 1 MEMREC EXT,
        2 UNUSED BIT(16),
        2 MEM_KEY CHAR(10),
        2 LMPU FIXED BIN(31,0),
        2 LOC(0:31) BIT(16);
   DCL TNOW FLOAT BIN EXT;
   DCL (MPU,NNATR,NMPU) FIXED BIN(31,0) EXT;
   DCL MFIND ENTRY(FIXED BIN(31,0),FIXED BIN(31,0))
        RETURNS(FIXED BIN(31,0));
   DCL INSCHD ENTRY(FIXED BIN(31,0));
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL ODEV ENTRY;
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL SUCPTR ENTRY(PTR) RETURNS(PTR);
   DCL (ADDRESS,DATA,J,LINK,SRCE_MPU) FIXED BIN(31,0);
   DCL DATASTR BIT(31);
   DCL FLAG BIT(1);
   IF SUBSTR(SYSTBL(0,0),5,1)='1'B THEN PUT SKIP DATA(ATRIB);
   IF ATRIB(4)=1 THEN DO;
/*****************************************************/
/*                                                   */
/*  DATA TRANSFER EVENT.....                         */
/*                                                   */
/*****************************************************/
        IF ATRIB(3)<NMPU THEN DO;
/*****************************************************/
```

```
/*                                                    */
/*  MPU IS RECEIVING DEVICE....                       */
/*                                                    */
/******************************************************/
            MPU=ATRIB(3);
            ADDRESS=ATRIB(6);
            J=MFIND(ADDRESS,1);
            LINK=BIN(SUBSTR(LOC(J),4,9),31,0);
            IF PCODE(LINK)=3 THEN DO;
/******************************************************/
/*                                                    */
/*  ACIA IS THE RECEIVING PERIPHERAL DEVICE...        */
/*                                                    */
/******************************************************/
                IF SUBSTR(DDR(LINK),8,1)='1'B THEN DO;
/******************************************************/
/*                                                    */
/*  RDRF='1'... OVERRUN HAS OCCURRED.  IF CTYPE='1'*/
/*  THEN THIS IS THE SECOND OVERRUN, AND THE DATA  */
/*  IS LOST.  IF NOT, CTYPE IS SET TO '1' AND THE  */
/*  DATA TRANSFER IS RESCHEDULED.                     */
/*                                                    */
/******************************************************/
                    IF CTYPE(LINK)='1'B THEN RETURN;
                    CTYPE(LINK)='1'B;
                    ATRIB(1)=TNOW+0.000001*4;
                    CALL FILEM(1);
                    END;
                ELSE DO;
/******************************************************/
/*                                                    */
/*  RDRF='0'... PDR IS EMPTY.  TRANSFER THE DATA   */
/*  TO THE RDR AND CHECK FOR INTERRUPTS.              */
/*                                                    */
/******************************************************/
                    DATA=ATRIB(5);
                    DATASTR=BIT(DATA,31);
                    SUBSTR(LOC(J),13,4)=SUBSTR(DATASTR,24,4);
                    J=MFIND(ADDRESS+1,1);
                    SUBSTR(LOC(J),13,4)=SUBSTR(DATASTR,28,4);
                    SUBSTR(DDR(LINK),8,1)='1'B;
                    IF SUBSTR(PDR(LINK),1,1)='1'B THEN DO;
/******************************************************/
/*                                                    */
/*  RIE='1'... SCHEDULE AN INTERRUPT FOR THIS MPU  */
/*  AND SET THE IRQ BIT TO '1'.                       */
/*                                                    */
/******************************************************/
```

```
                        CALL INSCHD(1000+MPU);
                        SUBSTR(DDR(LINK),1,1)='1'B;
                        END;
                    IF CTYPE(LINK)='1'B THEN DO;
/*****************************************************/
/*                                                   */
/*  OVERRUN OCCURRED ON THE PREVIOUS DATA TRANSFER   */
/*  AND THE OVRN BIT MUST BE SET TO '1'.             */
/*                                                   */
/*****************************************************/
                        CTYPE(LINK)='0'B;
                        SUBSTR(DDR(LINK),1,1)='1'B;
                        END;
                    END;
                END;
            ELSE IF PCODE(LINK)=1 ^ PCODE(LINK)=2 THEN DO;
/*****************************************************/
/*                                                   */
/*  THE RECEIVING DEVICE IS A PIA.                   */
/*                                                   */
/*****************************************************/
                DATA=ATRIB(5);
                DATASTR=BIT(DATA,31);
                PDR(LINK)=SUBSTR(DATASTR,24,8);
                END;
            END;
        ELSE CALL ODEV;
/*****************************************************/
/*                                                   */
/*  NON-MPU DEVICE IS ACCESSED... CALL ODEV TO       */
/*  PERFORM THE EVENT.                               */
/*                                                   */
/*****************************************************/
        RETURN;
        END;
   IF ATRIB(4)=2 ^ ATRIB(4)=3 THEN DO;
/*****************************************************/
/*                                                   */
/*  C1 SET/RESET EVENT.                              */
/*                                                   */
/*****************************************************/
        IF ATRIB(3)<NMPU THEN DO;
/*****************************************************/
/*                                                   */
/*  MPU DEVICE IS SPECIFIED.                         */
/*                                                   */
/*****************************************************/
            MPU=ATRIB(3);
```

```
            ADDRESS=ATRIB(6);
            J=MFIND(ADDRESS,1);
            LINK=BIN(SUBSTR(LOC(J),4,9),31,0);
            IF PCODE(LINK)=1 ^ PCODE(LINK)=2 THEN DO;
/*****************************************************/
/*                                                   */
/*  PIA ACCESSED.... THIS IS THE ONLY VALID MPU      */
/*  DEVICE FOR THIS EVENT, ALL OTHERS ARE IGNORED.   */
/*                                                   */
/*****************************************************/
                J=MFIND(ADDRESS+1,1);
                FLAG='0'B;
                IF C1(LINK)='0'B & ATRIB(4)=2 THEN DO;
                     C1(LINK)='1'B;
                     IF SUBSTR(LOC(J),11,1)='1'B THEN DO;
                          SUBSTR(LOC(J),5,1)='1'B;
                          IF SUBSTR(LOC(J),12,1)='1'B THEN
                               CALL INSCHD(1000+MPU);
                          FLAG='1'B;
                          END;
                     END;
                ELSE IF C1(LINK)='1'B & ATRIB(4)=3 THEN DO;
                     C1(LINK)='0'B;
                     IF SUBSTR(LOC(J),11,1)='0'B THEN DO;
                          SUBSTR(LOC(J),5,1)='1'B;
                          IF SUBSTR(LOC(J),12,1)='1'B THEN
                               CALL INSCHD(1000+MPU);
                          FLAG='1'B;
                          END;
                     END;
                IF FLAG='1'B &
                     SUBSTR(LOC(J),7,3)='100'B THEN DO;
                     C2(LINK)='1'B;
/*****************************************************/
/*                                                   */
/*  SCHEDULE SET ON ASSOCIATE LINE TO OCCUR NOW.     */
/*                                                   */
/*****************************************************/
                     ATRIB(1)=TNOW;
                     ATRIB(2)=300;
                     ATRIB(3)=PASSOC(LINK);
                     IF CTYPE(LINK)='1'B THEN ATRIB(4)=2;
                     ELSE ATRIB(4)=4;
                     ATRIB(6)=PASADD(LINK);
                     CALL FILEM(1);
                     END;
                END;
            END;
```

```
        ELSE CALL ODEV;
/*****************************************************/
/*                                                   */
/*  NON-MPU DEVICE.... CALL ODEV TO PROCESS THE      */
/*                                                   */
/*****************************************************/
        RETURN;
        END;
   IF ATRIB(4)=4 ^ ATRIB(4)=5 THEN DO;
/*****************************************************/
/*                                                   */
/*  C2 SET/RESET EVENT.....                          */
/*                                                   */
/*****************************************************/
        IF ATRIB(3)<NMPU THEN DO;
/*****************************************************/
/*                                                   */
/*  MPU DEVICE.....                                  */
/*                                                   */
/*****************************************************/
             MPU=ATRIB(3);
             ADDRESS=ATRIB(6);
             J=MFIND(ADDRESS,1);
             LINK=BIN(SUBSTR(LOC(J),4,9),31,0);
             IF PCODE(LINK)=1 ^ PCODE(LINK)=2 THEN DO;
/*****************************************************/
/*                                                   */
/*  PIA IS ACCESSED...CHECK FOR INTERRUPTS.          */
/*                                                   */
/*****************************************************/
                  J=MFIND(ADDRESS+1,1);
                  IF SUBSTR(LOC(J),7,1)='0'B THEN DO;
/*****************************************************/
/*                                                   */
/*  C2 USED AS AN INPUT...INTERRUPTS ARE POSSIBLE    */
/* IN THIS MODE.  IN THE OUTPUT MODE ALL C2 LINE     */
/*  CHANGES ARE IGNORED.                             */
/*                                                   */
/*****************************************************/
                       IF C2(LINK)='1'B & ATRIB(4)=5 THEN DO;
/*****************************************************/
/*                                                   */
/*  NEGATIVE TRANSITION ON C2.                       */
/*                                                   */
/*****************************************************/
                            IF SUBSTR(LOC(J),8,1)='0'B THEN DO;
                                 SUBSTR(LOC(J),6,1)='1'B;
                                 IF SUBSTR(LOC(J),9,1)='1'B THEN
```

```
                                        CALL INSCHD(1000+MPU);
                                    END;
                               END;
                          ELSE IF C2(LINK)='0'B & ATRIB(4)=4 THEN DO;
/*********************************************************/
/*                                                       */
/*  POSITIVE TRANSITION ON LINE C2.                      */
/*                                                       */
/*********************************************************/
                               IF SUBSTR(LOC(J),8,1)='1'B THEN DO;
                                    SUBSTR(LOC(J),6,1)='1'B;
                                    IF SUBSTR(LOC(J),9,1)='1'B THEN
                                         CALL INSCHD(1000+MPU);
                                    END;
                               END;
                          END;
                     IF ATRIB(4)=4 THEN C2(LINK)='1'B;
                     ELSE C2(LINK)='0'B;
                     END;
               END;
          ELSE CALL ODEV;
/*********************************************************/
/*                                                       */
/*  NON-MPU DEVICE..... CALL ODEV TO PROCESS THE         */
/*  EVENT.                                               */
/*                                                       */
/*********************************************************/
          RETURN;
          END;
     IF ATRIB(4)>5 THEN CALL ODEV;
/*********************************************************/
/*                                                       */
/*  ANY I/O EVNT NUMBERS GREATER THAN 5 ARE              */
/*  NON-MPU DEVICES......  ODEV IS USED TO PROCESS       */
/*  EVENTS.                                              */
/*                                                       */
/*********************************************************/
     RETURN;
     END IOEVNT;
```

```
/*  MAIN   VERSION-1                                          */
MAIN:  PROCEDURE OPTIONS(MAIN);
    DCL GASP ENTRY;
   DCL PL FIXED BIN(31,0) EXT;
   DCL (TIME,DATE) BUILTIN;
   DCL TTIME CHAR(9);
   DCL DDATE CHAR(8);
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
 /*************************************************************/
 /****                                                     ****/
 /****     CALL GASP EXECUTIVE TO START SIMULATION         ****/
 /****                                                     ****/
 /*************************************************************/
   PL=85;
   OPEN FILE(SYSPRINT) PAGESIZE(PL) LINESIZE(120);
   PUT EDIT('-')(LINE(1),A);
   CALL GASP;
   TTIME=TIME;
   DDATE=DATE;
   PUT SKIP(5) EDIT('RUN TERMINATED NORMALLY AT '^^
        SUBSTR(TTIME,1,2)^^':'^^SUBSTR(TTIME,3,2)^^':'^^
        SUBSTR(TTIME,5,2),SUBSTR(DDATE,3,2)^^'/'^^
        SUBSTR(DDATE,5,2)^^'/'^^SUBSTR(DDATE,1,2))
        (COL(10),A,X(3),A);
   PUT EDIT(' ')(LINE(1),A);
   CLOSE FILE(MEM1);
   END MAIN;
```

```
/*  MDUMP  VERSION-2                3/25/78                */
MDUMP: PROCEDURE;
/*******************************************************/
/*                                                     */
/*  THIS PROCEDURE PRINTS MEMORY DUMPS....             */
/*                                                     */
/*******************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (I,J,K) FIXED BIN(31,0);
   DCL (MPU,PL) FIXED BIN(31,0) EXT;
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL MFIND ENTRY(FIXED BIN(31,0),FIXED BIN(31,0))
        RETURNS(FIXED BIN(31,0));
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL 1 MEMREC EXT,
        2 UNUSED BIT(16),
        2 MEM_KEY CHAR(10),
        2 LMPU FIXED BIN(31,0),
        2 LOC(0:31) BIT(16);
   DCL 1 LINE,
        2 ADDRESS CHAR(4),
        2 DASH CHAR(2) INIT('--'),
        2 CHEX(0:31) CHAR(2);
/*******************************************************/
/*                                                     */
/*  PRINT HEADINGS...                                  */
/*                                                     */
/*******************************************************/
   IF ALLOCATION(SYSTBL)='1'B THEN DO;
        PUT EDIT('MEMORY DUMP FOR MPU',MPU)(LINE(1),SKIP(4),A,F(3,0));
        PUT SKIP(2) EDIT(' ') (A);
        DO I=0 TO SYSTBL(1,MPU)-8 BY 32;
             J=MFIND(I,1);
             ADDRESS=SUBSTR(HHEX(I),2,4);
             CHEX=' ';
             DO J=0 TO 31;
                  IF J+I<=SYSTBL(1,MPU)-8 THEN
                       IF SUBSTR(LOC(J),2,1)='1'B THEN CHEX(J)='PP';
                       ELSE CHEX(J)=
                            SUBSTR(HHEX(SUBSTR(LOC(J),9,8)),4,2);
                  ELSE J=32;
                  END;
             PUT SKIP EDIT(LINE)(A);
             IF LINENO(SYSPRINT)>PL-15 THEN
                  PUT EDIT('MEMORY DUMP FOR MPU',MPU,' ')
                  (LINE(1),SKIP(4),A,F(3,0),SKIP(2),A);
             END;
        ADDRESS='FFF8';
```

```
     CHEX=' ';
     K=0;
     DO I=65528 TO 65535;
          J= MFIND(I,1);
          CHEX(K)=SUBSTR(HHEX(SUBSTR(LOC(J),9,8)),4,2);
          K=K+1;
          END;
     PUT SKIP EDIT(LINE)(A);
     END;
ELSE PUT SKIP EDIT('UNABLE TO PERFORM MEMORY DUMP')(COL(10),A);
RETURN;
END MDUMP;
```

```
/*    MFIND      VERSION-1       3/6/78                              */
MFIND:  PROCEDURE(ADDRESS,TCODE) RETURNS(FIXED BIN(31,0));
/*****************************************************************/
/*                                                               */
/*  THIS PROCEDURE LOCATES A MEMORY LOCATION AND STORES IT       */
/*  IN THE ARRAY LOC(0:31).                                      */
/*                                                               */
/*****************************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (MPU,NMPU,OLDADD)  FIXED BIN(31,0) EXT;
   DCL 1 CMEM(*) CTL EXT,
        2 CMPU FIXED BIN(31,0),
        2 CLOC(0:31) BIT(16);
   DCL MEMTYPE CHAR(6) EXT;
   DCL 1 MEMREC EXT,
        2 UNUSED BIT(16),
        2 MEM_KEY CHAR(10),
        2 LMPU FIXED BIN(31,0),
        2 LOC(0:31) BIT(16);
   DCL (ADDRESS,JJ,TCODE,TEMP,TADDRESS) FIXED BIN(31,0);
   DCL TEMP_KEY CHAR(10);
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL (ONKEY,ONCODE) BUILTIN;
/*****************************************************************/
/*                                                               */
/*    ON UNITS.....................                              */
/*                                                               */
/*****************************************************************/
   ON ENDFILE(MEM1) BEGIN;
        PUT SKIP(2) EDIT('******** END-OF-FILE  FOUND ON '^^
             'FILE(MEM1).  (FATAL)')(A);
        JJ=-1;
        GO TO EOP;
        END;
   ON KEY(MEM1) BEGIN;
        PUT SKIP(2) EDIT('******** KEY ERROR IN FILE(MEM1):'^^
             '  CODE='^^ONCODE^^';  KEY='^^ONKEY^^'.')(A);
        JJ=-1;
        GO TO EOP;
        END;
/*****************************************************************/
/*                                                               */
/*  FIND MEMORY LOCATION, READ FROM MEMORY, AND PLACE IN         */
/*  THE ARRAY: LOC(0:31).  THE VALUE RETURNED IS THE VALUE OF    */
/*  THE SUBSCRIPT OF THE LOCATION WHICH CONTAINS THE MEMORY      */
/*  LOCATION.   A -1 IS RETURNED FOR INVALID ADDRESSES.          */
/*                                                               */
/*****************************************************************/
```

```
   JJ=0;
   IF MPU>=0 & MPU<NMPU THEN
     IF (ADDRESS>=0 & ADDRESS<=SYSTBL(1,MPU)-8 ) ^
        (ADDRESS>65527 & ADDRESS<65536) THEN DO;
        IF TCODE\=0 THEN DO;
             IF ADDRESS>65527 & ADDRESS<65536 THEN
                  TEMP=SYSTBL(1,MPU)-65535+ADDRESS;
             ELSE TEMP=ADDRESS;
             TADDRESS=SYSTBL(2,MPU)+FLOOR(TEMP/32);
             TEMP_KEY=DEC(TADDRESS,7,0);
             IF TEMP_KEY\=MEM_KEY THEN DO;
                  IF MEMTYPE='DISK' THEN DO;
                       REWRITE FILE(MEM1) FROM(MEMREC) KEY(MEM_KEY);
                       READ FILE(MEM1) INTO(MEMREC) KEY(TEMP_KEY);
                       END;
                  ELSE DO;
                       DO I=0 TO 31;
                            CLOC(OLDADD,I)=LOC(I);
                            END;
                       DO I=0 TO 31;
                            LOC(I)=CLOC(TADDRESS,I);
                            END;
                       OLDADD=TADDRESS;
                       MEM_KEY=TEMP_KEY;
                       END;
                  END;
             JJ=MOD(TEMP,32);
             END;
        END;
     ELSE JJ=-2;
   ELSE JJ=-3;
EOP:  RETURN(JJ);
   END MFIND;
```

```
/*    MPHLT       VERSION-1              2/19/78          */
MPHLT: PROCEDURE(I);
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE IS USED TO HALT THE MPU'S         */
/*                                                   */
/*****************************************************/
   DCL I FIXED BIN(31,0);
   DCL (NNQ(*),SYSTBL(*,*)) FIXED BIN(31,0) CTL EXT;
   DCL MSTOP CHAR(5) EXT;
   DCL (MPU,NMPU) FIXED BIN(31,0) EXT;
   DCL 1 STAT(*) CTL EXT,
        2 INST FIXED BIN(31,0),
        2 INPT FIXED BIN(31,0),
        2 OTPT FIXED BIN(31,0),
        2 ERRS FIXED BIN(31,0),
        2 INTS FIXED BIN(31,0),
        2 WARN FIXED BIN(31,0),
        2 BRCH FIXED BIN(31,0),
        2 RTRN FIXED BIN(31,0),
        2 RSTS FIXED BIN(31,0),
        2 HALT FLOAT BIN(31),
        2 RUN FLOAT BIN(31),
        2 RESTRT FLOAT BIN(31),
        2 LSTHLT FLOAT BIN(31);
   DCL UERR ENTRY(FIXED BIN(31,0));
   DCL STR7 BIT(31);
   DCL ERR ENTRY;
   DCL TNOW FLOAT BIN EXT;
   MPU=I-100;
   STR7=BIT(SYSTBL(7,MPU),31);
   IF SUBSTR(STR7,1,4)='0000'B THEN DO;
        MSTOP='STOP';
        SUBSTR(STR7,4,1)='1'B;
        CALL ERR;
        END;
   IF SUBSTR(STR7,4,1)='1'B THEN DO;
        RUN(MPU)=RUN(MPU)+TNOW-RESTRT(MPU);
        LSTHLT(MPU)=TNOW;
        SYSTBL(4,MPU)=0;
        END;
   SYSTBL(7,MPU)=BIN(STR7,31,0);
   DO MPU=0 TO NMPU-1;
        IF SUBSTR(SYSTBL(7,MPU),4,1)='0'B THEN MSTOP='USER';
        ELSE IF NNQ(1)\=0 THEN MSTOP='USER';
        END;
   RETURN;
   END MPHLT;
```

```
/*   ODEV     VERSION-1              6/16/78       */
ODEV:  PROCEDURE;
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE TAKES CARE OF THE I/O EVENTS      */
/*  THAT OCCUR FOR THE NON-MPU DEVICES.              */
/*                                                   */
/*****************************************************/
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL 1 BUSTBL(0:2) EXT,
        2 ACTIVE BIT(1),
        2 CONTR FIXED BIN(31,0);
   DCL (BASE,BUSS,DEVICE,START_ADDRESS) FIXED BIN(31,0);
   DCL NO_BYTES FIXED BIN(31,0);
   DCL 1 BIMTBL(0:3) EXT,
        2 BDATA FIXED BIN(31,0),
        2 STATUS BIT(8),
        2 RBFST FIXED BIN(16,0);
   DCL TNOW FLOAT BIN EXT;
   DCL (MPU,NMPU) FIXED BIN(31,0) EXT;
   DCL UNFRM ENTRY(FLOAT BIN,FLOAT BIN, FIXED BIN(31,0))
        RETURNS(BIN);
   DCL COPY ENTRY(PTR);
   DCL READ ENTRY(FIXED BIN(16,0)) RETURNS(FIXED BIN(8,0));
   DCL RMOVE ENTRY(PTR,FIXED BIN(31,0));
   DCL FIND ENTRY(FLOAT BIN,FIXED BIN(31,0),PTR,FIXED BIN(31,0),
        FLOAT BIN) RETURNS(PTR);
   DCL COLCT ENTRY(FLOAT BIN,FIXED BIN(31,0));
   DCL MFE(*) PTR CTL EXT;
   DCL SNTRY PTR;
   DCL NULL BUILTIN;
   DCL WRITE ENTRY(FIXED BIN(8,0),FIXED BIN(16,0));
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL UERR ENTRY(FIXED BIN(31,0));
   DCL SUCPTR ENTRY(PTR) RETURNS(PTR);
   DCL (ADDRESS,DATA,I,J,SRCE_MPU) FIXED BIN(31,0);
   DCL (SET_UP,TRANS_TIME) FLOAT BIN(31);
   DCL DBUF(260) FIXED BIN(8,0) EXT;
   DCL FLAG BIT(1);
   IF ATRIB(4)=1 THEN DO;
/*****************************************************/
/*                                                   */
/*  DATA TRANSFER EVENT.....                         */
/*                                                   */
/*****************************************************/
        IF ATRIB(3)>99 & ATRIB(3)<200 THEN DO;
/*****************************************************/
/*                                                   */
```

```
/*  BIM ACCESSED.  STORE DATA IN BDATA REGISTER.   */
/*                                                 */
/***************************************************/
             DEVICE=ATRIB(3)-100;
             BDATA(DEVICE)=ATRIB(5);
             END;
        ELSE;
/***************************************************/
/*                                                 */
/*  NON-BIM DEVICE ACESSED......                   */
/*                                                 */
/***************************************************/
        RETURN;
        END;
   IF ATRIB(4)=2 ^ ATRIB(4)=3 THEN DO;
/***************************************************/
/*                                                 */
/*  C1 SET/RESET EVENT.                            */
/*                                                 */
/***************************************************/
        IF ATRIB(3)>99 & ATRIB(3)<200 THEN
             IF ATRIB(4)=3 & ATRIB(6)=0 THEN DO;
/***************************************************/
/*                                                 */
/*  BIM DEVICE... READ AND DECODE THE DATA         */
/*  PRESENT IN THE BDATA REGISTER...               */
/*                                                 */
/***************************************************/
             DEVICE=ATRIB(3)-100;
             ATRIB(1)=TNOW+UNFRM(2.5E-05,2.0E-04,1);
             ATRIB(8)=TNOW;
             IF BDATA(DEVICE)=1 THEN DO;
/***************************************************/
/*                                                 */
/*  'SND' COMMAND...SCHEDULE SEND DATA EVENT.      */
/*                                                 */
/***************************************************/
                  ATRIB(4)=6;
                  CALL FILEM(1);
                  END;
             IF BDATA(DEVICE)=2 THEN DO;
/***************************************************/
/*                                                 */
/*  'STB' COMMAND.... SCHEDULE SEND TOS TABLES     */
/*  EVENT.                                         */
/*                                                 */
/***************************************************/
                  ATRIB(4)=8;
```

```
                CALL FILEM(1);
                END;
            IF BDATA(DEVICE)=3 THEN DO;
/*****************************************************/
/*                                                   */
/*  'RST' COMMAND.. SCHEDULE DATA TRANSFER EVENT     */
/*  STATUS WORD TO THE CPS IN 23 MICRO SECONDS.      */
/*  THIS IS INCLUDES THE C1 RESET.                   */
/*                                                   */
/*****************************************************/
                ATRIB(1)=TNOW+0.000023;
                ATRIB(3)=DEVICE;
                ATRIB(4)=1;
                ATRIB(5)=BIN(STATUS(DEVICE),31,0);
                ATRIB(6)=2048;
                CALL FILEM(1);
                ATRIB(1)=ATRIB(1)+0.000001;
                ATRIB(4)=3;
                CALL FILEM (1);
                ATRIB(1)=ATRIB(1)+0.000001;
                ATRIB(4)=2;
                CALL FILEM(1);
                END;
            IF BDATA(DEVICE)=4 THEN DO;
/*****************************************************/
/*                                                   */
/*  'GTC' COMMAND...SCHEDULE EXECUTION OF THE 'GTC'  */
/*  COMMAND.                                         */
/*                                                   */
/*****************************************************/
                ATRIB(4)=11;
                CALL FILEM(1);
                END;
            IF BDATA(DEVICE)=5 THEN DO;
/*****************************************************/
/*                                                   */
/*  'BCR'  COMMAND.... EXECUTE BIM CLEAR.            */
/*                                                   */
/*****************************************************/
                STATUS(DEVICE)='00000000'B;
                MPU=DEVICE;
                RBFST(MPU)=256*READ(1794)+READ(1795);
                SNTRY=MFE(1);
LOOP:           SNTRY=FIND(100+DEVICE,5,SNTRY,3,0);
                IF SNTRY\=NULL THEN DO;
                     CALL COPY(SNTRY);
                     IF ATRIB(4)>5 THEN DO;
                          CALL RMOVE(SNTRY,1);
```

```
                         IF ATRIB(4)=9 THEN DO MPU=0 TO 3;
                              SUBSTR(STATUS(MPU),2,1)='0'B;
                              CALL WRITE(STATUS(MPU),1796);
                              END;
                         END;
                    SNTRY=SUCPTR(SNTRY);
                    GO TO LOOP;
                    END;
               DO I=0 TO 2;
                    IF CONTR(I)=DEVICE+10 THEN ACTIVE(I)='0'B;
                    END;
               END;
          ATRIB(1)=TNOW+0.000014;
          ATRIB(2)=300;
          ATRIB(3)=DEVICE;
          ATRIB(4)=3;
          ATRIB(6)=2050;
          CALL FILEM(1);
          ATRIB(1)=ATRIB(1)+0.000001;
          ATRIB(4)=2;
          CALL FILEM(1);
          IF BDATA(DEVICE)=5 THEN DO;
               ATRIB(6)=2048;
               CALL FILEM(1);
               END;
          END;
     RETURN;
     END;
  IF ATRIB(4)=4 ^ ATRIB(4)=5 THEN DO;
/*****************************************************/
/*                                                   */
/*  C2 SET/RESET EVENT.....                          */
/*  THE BIMS ARE INITIALIZED AS PULSE OUTPUT MODE.   */
/*  THEY DO NOT USE C2 AS AN INPUT...AND THUS THIS   */
/*  IS AN ILLEGAL EVENT.                             */
/*                                                   */
/*****************************************************/
    CALL UERR(20);
     RETURN;
     END;
/*****************************************************/
/*                                                   */
/*  THE FOLLOWING EVENTS ARE DEFINED FOR BIMS        */
/*  ONLY.....                                        */
/*                                                   */
/*****************************************************/
  IF ATRIB(4)=6 THEN DO;
/*****************************************************/
```

```
/*                                                    */
/*  'SND' EVENT.... CHECK TO SEE IF BIM IS ACTIVE     */
/*                                                    */
/****************************************************/
        MPU=ATRIB(3)-100;
        IF SUBSTR(STATUS(MPU),6,1)='0'B &
             SUBSTR(STATUS(MPU),1,1)='1'B THEN DO;
/****************************************************/
/*                                                    */
/*  BIM IS ACTIVE... RESCHEDULE 'SND IN 30 MICRO-     */
/*  SECONDS......                                     */
/*                                                    */
/****************************************************/
             ATRIB(1)=TNOW+0.00003;
             CALL FILEM(1);
             END;
        ELSE DO;
/****************************************************/
/*                                                    */
/*  BIM IS INACTIVE...PERFORM EVENT.                  */
/*                                                    */
/****************************************************/
             SUBSTR(STATUS(MPU),1,1)='1'B;
             BUSS=-1;
             DO I=0 TO 2;
                  IF CONTR(I)=MPU+10 THEN BUSS=I;
                  END;
             IF BUSS=-1 THEN DO I=0 TO 2;
                  IF ACTIVE(I)='0'B THEN BUSS=I;
                  END;
             IF BUSS=-1 THEN DO;
/****************************************************/
/*                                                    */
/*  ALL BUSSES ARE ACTIVE..RECHECK IN 50 MICRO-       */
/*  SECONDS.                                          */
/*                                                    */
/****************************************************/
                  ATRIB(1)=TNOW+0.00005;
                  CALL FILEM(1);
                  SUBSTR(STATUS(MPU),6,1)='1'B;
                  END;
             ELSE DO;
/****************************************************/
/*                                                    */
/*  INACTIVE BUSS FOUND... GET CONTROL, IF NOT        */
/*  ALREADY CONTROLLER AND START TRANSFER             */
/*                                                    */
/****************************************************/
```

```
                CALL COLCT(TNOW-ATRIB(8),MPU+NMPU+1);
                CALL COLCT(TNOW-ATRIB(8),13);
                ADDRESS=256*READ(1792)+READ(1793);
                NO_BYTES=READ(ADDRESS);
                STATUS(MPU)='10011000'B;
                ACTIVE(BUSS)='1'B;
                IF CONTR(BUSS)=MPU+10 THEN SET_UP=0.000030;
                ELSE SET_UP=0.00005;
                TRANS_TIME=(NO_BYTES+2)*0.00003;
                CONTR(BUSS)=MPU+10;
                ATRIB(1)=TNOW+SET_UP+TRANS_TIME;
                ATRIB(2)=300;
                ATRIB(4)=7;
                ATRIB(6)=NO_BYTES;
                ATRIB(7)=ADDRESS;
                CALL FILEM(1);
/*************************************************/
/*                                               */
/*  SEND 'TRS' TO CPS.....                       */
/*                                               */
/*************************************************/
                ATRIB(1)=TNOW+0.000025;
                ATRIB(2)=300;
                ATRIB(3)=MPU;
                ATRIB(4)=1;
                ATRIB(5)=1;
                ATRIB(6)=2048;
                CALL FILEM(1);
                ATRIB(1)=ATRIB(1)+0.000001;
                ATRIB(4)=3;
                CALL FILEM(1);
                ATRIB(1)=ATRIB(1)+0.000001;
                ATRIB(4)=2;
                CALL FILEM(1);
                END;
            END;
        RETURN;
        END;
   IF ATRIB(4)=7 THEN DO;
/*************************************************/
/*                                               */
/*  'SDC' EVENT...COPY DATA FROM OUTPUT BUFFER TO */
/*  INPUT BUFFERS OF RECEIVING DEVICES.          */
/*                                               */
/*************************************************/
        SRCE_MPU,MPU=ATRIB(3)-100;
        ADDRESS=ATRIB(7);
        NO_BYTES=ATRIB(6);
```

```
        FLAG='0'B;
        I=0;
        START_ADDRESS=ADDRESS+1;
        SNTRY=FIND(10+MPU,5,MFE(2),1,0);
        IF SNTRY\=NULL THEN DO;
             CALL RMOVE(SNTRY,2);
             ATRIB(2)=ATRIB(2)+NO_BYTES;
             CALL FILEM(2);
             END;
        DO WHILE(FLAG='0'B);
             DATA=READ(START_ADDRESS);
             IF DATA=255 THEN I=I+1;
             IF I=2 THEN FLAG='1'B;
             START_ADDRESS=START_ADDRESS+1;
             IF START_ADDRESS>ADDRESS+NO_BYTES THEN DO;
/*****************************************************/
/*                                                   */
/*  BUFFER ERROR...SEND CPS ERROR MESSAGE AND        */
/*  DEACTIVATE BIM.                                  */
/*                                                   */
/*****************************************************/
                  ATRIB(1)=TNOW+0.0002;
                  ATRIB(2)=300;
                  ATRIB(3)=MPU;
                  ATRIB(4)=1;
                  ATRIB(5)=7;
                  ATRIB(6)=2048;
                  CALL FILEM(1);
                  ATRIB(4)=3;
                  CALL FILEM(1);
                  ATRIB(1)=ATRIB(1)+0.000001;
                  ATRIB(4)=2;
                  CALL FILEM(1);
                  STATUS(MPU)='00000000'B;
                  CALL WRITE(STATUS(MPU),1796);
                  DO I=0 TO 2;
                       IF CONTR(I)=MPU+10 THEN ACTIVE(I)='0'B;
                       END;
                  RETURN;
                  END;
             END;
        DBUF(1)=MPU;
        BASE=START_ADDRESS;
        DO I=2 TO NO_BYTES;
             DBUF(I)=READ(BASE);
             BASE=BASE+1;
             IF BASE=1023 THEN BASE=128-I;
             END;
```

```
        DBUF(I)=0;
        I=I+1;
        DBUF(I)=0;
        DO J=ADDRESS+1 TO START_ADDRESS-3;
             MPU=SRCE_MPU;
             DEVICE=READ(J);
             IF DEVICE<NMPU THEN DO;
/*****************************************************/
/*                                                   */
/*  CPS IS THE RECEIVER... MUST GET THE RECEIVER     */
/*  BUFFER START ADDRESS AND STORE IN BIM FOR THIS   */
/*  RECEIVER.                                        */
/*                                                   */
/*****************************************************/
                  MPU=DEVICE;
                  SNTRY=FIND(DEVICE+10,5,MFE(2),1,0);
                  IF SNTRY\=NULL THEN DO;
                       CALL RMOVE(SNTRY,2);
                       ATRIB(3)=ATRIB(3)+NO_BYTES;
                       CALL FILEM(2);
                       END;
                  CALL WRITE(SUBSTR(RBFST(MPU),1,8),1794);
                  CALL WRITE(SUBSTR(RBFST(MPU),9,8),1795);
/*****************************************************/
/*                                                   */
/*  SCHEDULE 'DIN' MESSAGE TO BE SENT TO RECEIVING   */
/*  MPU.                                             */
/*                                                   */
/*****************************************************/
                  ATRIB(1)=TNOW;
                  ATRIB(2)=300;
                  ATRIB(3)=DEVICE;
                  ATRIB(4)=1;
                  ATRIB(5)=8;
                  ATRIB(6)=2048;
                  CALL FILEM(1);
                  ATRIB(4)=3;
                  CALL FILEM(1);
                  ATRIB(1)=TNOW+0.000001;
                  ATRIB(4)=2;
                  CALL FILEM(1);
                  DO I=1 TO NO_BYTES;
                       CALL WRITE(DBUF(1),RBFST(DEVICE));
                       IF RBFST(DEVICE)=1791 THEN
                            RBFST(DEVICE)=1023;
                       RBFST(DEVICE)=RBFST(DEVICE)+1;
                       END;
                  END;
```

```
            ELSE DO;
/*****************************************************/
/*                                                   */
/*  NON-CPS DEVICE IS THE RECEIVER....               */
/*                                                   */
/*****************************************************/
                ATRIB(2)=300;
                ATRIB(3)=SRCE_MPU;
                ATRIB(4)=20;
                ATRIB(5)=READ(ADDRESS);
                ATRIB(6)=DEVICE;
                ATRIB(1)=TNOW+UNFRM(5.0E-5,5.0E-4,1)+ATRIB(5)*3.0E-5;
                CALL FILEM(1);
                DO I=0 TO 2;
                     IF CONTR(I)=MPU+10 THEN ACTIVE(I)='1'B;
                     END;
                STATUS(MPU)='10000000'B;
                CALL WRITE(STATUS(MPU),1796);
                END;
           END;
        RETURN;
        END;
   IF ATRIB(4)=8  THEN DO;
/*****************************************************/
/*                                                   */
/*  'STB' EVENT... SCHEDULE TRANSFER START IF 8      */
/*  OR TRANSFER COMPLETE IF 10.  CHECK FOR BIM       */
/*  ACTIVE,  SCHEDULE A RECHECK IF IT IS           */
/*                                                   */
/*****************************************************/
        SRCE_MPU=ATRIB(3)-100;
        IF SUBSTR(STATUS(SRCE_MPU),1,2)='10'B THEN DO;
             ATRIB(1)=TNOW+0.00005;
             CALL FILEM(1);
             END;
        ELSE DO;
             MPU=SRCE_MPU;
             I=3+READ(2)*4;
             DO ADDRESS=0 TO I;
                  MPU=SRCE_MPU;
                  DATA=READ(ADDRESS);
                  DO MPU=0 TO NMPU-1;
                       IF MPU\=SRCE_MPU THEN
                            CALL WRITE(DATA,ADDRESS);
                       END;
                  END;
                  DO MPU=0 TO NMPU-1;
                       SNTRY=FIND(MPU+10,5,MFE(2),1,0);
```

```
                    IF MPU\=SRCE_MPU & SNTRY\=NULL THEN DO;
                         CALL RMOVE(SNTRY,2);
                         ATRIB(3)=ATRIB(3)+I;
                         CALL FILEM(2);
                         END;
                    END;
/*******************************************************/
/*                                                     */
/*  SCHEDULE 'STC' EVENT.....                          */
/*                                                     */
/*******************************************************/
          MPU=SRCE_MPU;
          ATRIB(8)=READ(2);
          ATRIB(1)=TNOW+ATRIB(8)*0.000013;
          ATRIB(2)=300;
          ATRIB(3)=SRCE_MPU+100;
          ATRIB(4)=9;
          CALL FILEM(1);
          SNTRY=FIND(MPU+10,5,MFE(2),1,0);
          IF SNTRY\=NULL THEN DO;
               CALL RMOVE(SNTRY,2);
               ATRIB(2)=ATRIB(2)+I;
               CALL FILEM(2);
               END;
          END;
     MPU=SRCE_MPU;
     CALL WRITE(STATUS(MPU),1796);
     RETURN;
     END;
  IF ATRIB(4)=9 THEN DO;
/*******************************************************/
/*                                                     */
/*  'STC' EVENT...SYSTEM TABLE TRANSFER COMPLETE       */
/*                                                     */
/*******************************************************/
     SRCE_MPU,MPU=ATRIB(3)-100;
     SUBSTR(STATUS(MPU),1,1)='0'B;
     DO MPU=0 TO NMPU-1;
          SUBSTR(STATUS(MPU),2,1)='0'B;
          CALL WRITE(STATUS(MPU),1796);
          END;
     DO I=0 TO 2;
          IF CONTR(I)=SRCE_MPU+10 THEN ACTIVE(I)='0'B;
          END;
     ATRIB(1)=TNOW+0.000001;
     ATRIB(2)=300;
     ATRIB(3)=SRCE_MPU;
     ATRIB(4)=1;
```

```
        ATRIB(5)=6;
        ATRIB(6)=2048;
        CALL FILEM(1);
        ATRIB(1)=ATRIB(1)+0.000001;
        ATRIB(4)=3;
        CALL FILEM(1);
        ATRIB(1)=ATRIB(1)+0.000001;
        ATRIB(4)=2;
        CALL FILEM(1);
        RETURN;
        END;
   IF ATRIB(4)=11 THEN DO;
/*****************************************************/
/*                                                   */
/*  'GTC' COMMAND.... CHECK TO SEE IF BIM IS         */
/*  ACTIVE.  IF NOT, SCHEDULE RECHECK AT TNOW +      */
/*  30 US.                                           */
/*                                                   */
/*****************************************************/
        MPU=ATRIB(3)-100;
        IF SUBSTR(STATUS(MPU),1,1)='1'B THEN DO;
/*****************************************************/
/*                                                   */
/*  BIM IS ACTIVE.......                             */
/*                                                   */
/*****************************************************/
             ATRIB(1)=TNOW+0.00003;
             CALL FILEM(1);
             END;
        ELSE DO;
/*****************************************************/
/*                                                   */
/*  BIM IS INACTIVE...CHECK TO SEE IF THERE IS AN    */
/*  INACTIVE BUSS AVAILABLE.                         */
/*                                                   */
/*****************************************************/
             BUSS=-1;
             SUBSTR(STATUS(MPU),1,1)='1'B;
             CALL WRITE(STATUS(MPU),1796);
             DO I=0 TO 2;
                  IF CONTR(I)=MPU+10 THEN BUSS=I;
                  END;
             IF BUSS=-1 THEN DO I=0 TO 2;
                  IF ACTIVE(I)='0'B THEN BUSS=I;
                  END;
             IF BUSS=-1 THEN DO;
/*****************************************************/
/*                                                   */
```

```
/*  NO INACTIVE BUSSES ARE AVAILABLE...                    */
/*  RESCHEDULE A RECHECK IN ANOTHER 30 US.                 */
/*                                                         */
/*********************************************************/
                  ATRIB(1)=TNOW+0.000030;
                  CALL FILEM(1);
                  SUBSTR(STATUS(MPU),6,1)='1'B;
                  SUBSTR(STATUS(MPU),1,1)='0'B;
                  CALL WRITE(STATUS(MPU),1796);
                  END;
               ELSE DO;
/*********************************************************/
/*                                                       */
/*  INACTIVE BUSS FOUND... CHECK TO SEE IF TOS           */
/*  TABLES ARE CURRENTLY BEING UPDATED.                  */
/*                                                       */
/*********************************************************/
                  ATRIB(7)=0;
                  IF SUBSTR(STATUS(MPU),2,1)='1'B THEN DO;
                       ATRIB(1)=TNOW+0.000030;
                       CALL FILEM(1);
                       SUBSTR(STATUS(MPU),1,1)='0'B;
                       END;
                  ELSE DO;
/*********************************************************/
/*                                                       */
/*  TOS TABLES ARE NOT BEING UPDATED... GET CONTROL      */
/*  OF BUSS AND START TOS TABLE UPDATE.                  */
/*                                                       */
/*********************************************************/
                       CALL COLCT(TNOW-ATRIB(8),MPU+NMPU+1);
                       CALL COLCT(TNOW-ATRIB(8),13);
                       ACTIVE(BUSS)='1'B;;
                       CONTR(BUSS)=MPU+10;
                       ATRIB(1)=TNOW+UNFRM(2.5E-05,2.0E-04,1);
                       ATRIB(2)=300;
                       CALL COLCT(ATRIB(1)-ATRIB(8),MPU+NMPU+5);
                       CALL COLCT(ATRIB(1)-ATRIB(8),14);
                       ATRIB(3)=MPU;
                       ATRIB(4)=1;
                       ATRIB(5)=9;
                       ATRIB(6)=2048;
                       CALL FILEM(1);
                       ATRIB(1)=ATRIB(1)+0.000001;
                       ATRIB(4)=3;
                       CALL FILEM(1);
                       ATRIB(1)=ATRIB(1)+0.000001;
                       ATRIB(4)=2;
```

```
                    CALL FILEM(1);
                    DO MPU=0 TO NMPU-1;
                         SUBSTR(STATUS(MPU),2,1)='1'B;
                         CALL WRITE(STATUS(MPU),1796);
                         END;
                    END;
               END;
          END;
       RETURN;
       END;
  IF ATRIB(4)=20 THEN DO;
/*****************************************************/
/*                                                   */
/*  RETURN DATA EVENT.....                           */
/*                                                   */
/*****************************************************/
       MPU=ATRIB(3);
       NO_BYTES=ATRIB(5);
       DEVICE=ATRIB(6);
       IF NO_BYTES>0 THEN DO;
            SNTRY=FIND(MPU+10,5,MFE(2),1,0);
            IF SNTRY\=NULL THEN DO;
                 CALL RMOVE(SNTRY,2);
                 ATRIB(3)=ATRIB(3)+NO_BYTES;
                 CALL FILEM(2);
                 END;
            CALL WRITE(SUBSTR(RBFST(MPU),1,8),1794);
            CALL WRITE(SUBSTR(RBFST(MPU),9,8),1795);
/*****************************************************/
/*                                                   */
/*  SCHEDULE 'DIN' MESSAGE TO BE SENT TO RECEIVING   */
/*  MPU.                                             */
/*                                                   */
/*****************************************************/
            ATRIB(1)=TNOW;
            ATRIB(2)=300;
            ATRIB(3)=MPU;
            ATRIB(4)=1;
            ATRIB(5)=8;
            ATRIB(6)=2043;
            CALL FILEM(1);
            ATRIB(4)=3;
            CALL FILEM(1);
            ATRIB(1)=TNOW+0.000001;
            ATRIB(4)=2;
            CALL FILEM(1);
            CALL WRITE(NO_BYTES,RBFST(MPU));
            CALL WRITE(DEVICE,RBFST(MPU)+1);
```

```
          RBFST(MPU)=RBFST(MPU)+NO_BYTES;
          IF RBFST(MPU)>1791 THEN
               RBFST(MPU)=1023 +RBFST(MPU)-1791;
          CALL WRITE(0,RBFST(MPU)-1);
          DO I=0 TO 2;
               IF CONTR(I)=MPU+10 THEN ACTIVE(I)='0'B;
               END;
          END;
     STATUS(MPU)='00000000'B;
     CALL WRITE(STATUS(MPU),1796);
     RETURN;
     END;
END ODEV;
```

```
/*   OTPUT   VERSION-2                2/21/78                    */
OTPUT: PROCEDURE;
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE PRINTS THE FINAL REPORTS ON THE   */
/*  MPUS IN THE SYSTEM.                              */
/*                                                   */
/*****************************************************/
   DCL (SYSTBL(*,*),NNQ(*)) FIXED BIN(31,0) CTL EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL TNOW FLOAT BIN EXT;
   DCL I FIXED BIN(31,0);
   DCL (MPU,NMPU,NPER,PL) FIXED BIN(31,0) EXT;
   DCL 1 STAT(*) CTL EXT,
        2 INST FIXED BIN(31,0),
        2 INPT FIXED BIN(31,0),
        2 OTPT FIXED BIN(31,0),
        2 ERRS FIXED BIN(31,0),
        2 INTS FIXED BIN(31,0),
        2 WARN FIXED BIN(31,0),
        2 BRCH FIXED BIN(31,0),
        2 RTRN FIXED BIN(31,0),
        2 RSTS FIXED BIN(31,0),
        2 HALT FLOAT BIN(31),
        2 RUN FLOAT BIN(31),
        2 RESTRT FLOAT BIN(31),
        2 LSTHLT FLOAT BIN(31);
   DCL (MDUMP,PDUMP,TRCPRT,USUM) ENTRY;
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL (PC(*),SP(*),INR(*)) FIXED BIN(16,0) CTL EXT;
   DCL (ACCA(*),ACCB(*)) FIXED BIN(8,0) CTL EXT;
   DCL CCR(*) BIT(8) CTL EXT;
/*****************************************************/
/*                                                   */
/*  PRINT ANY TRACE ENTRIES STILL IN THE TRACE       */
/*  FILES...                                         */
/*                                                   */
/*****************************************************/
   DO MPU=0 TO NMPU-1;
        DO WHILE(NNQ(MPU+3)\=0);
             CALL TRCPRT;
             END;
        END;
   PUT EDIT(' ')(LINE(PL-1),A);
   PUT SKIP(3) EDIT('****  SIMULATION RESULTS  ****')
        (COL(16),A);
   PUT SKIP(3);
   DO MPU=0 TO NMPU-1;
```

```
        IF LINENO(SYSPRINT)>PL-10 THEN PUT EDIT(' ')(LINE(PL-1),A);
        PUT SKIP(2) EDIT('FINAL STATUS REPORT---- MPU NUMBER',
             MPU,':')(A,F(3,0),A);
        PUT SKIP(2) EDIT('PC='^^HHEX(PC(MPU))^^' SP='^^HHEX(SP(MPU))
             ^^' INR='^^HHEX(INR(MPU))^^' ACCA='^^HHEX(ACCA(MPU))
             ^^' ACCB='^^HHEX(ACCB(MPU))^^' CCR='^^CCR(MPU))(A);
        END;
   CALL PDUMP;
   DO MPU=0 TO NMPU-1;
        IF SUBSTR(SYSTBL(0,MPU),2,1)='1'B THEN CALL MDUMP;
        END;
/*****************************************************/
/*                                                   */
/*  PRINT SIMULATION STATISTICS...                   */
/*                                                   */
/*****************************************************/
   IF LINENO(SYSPRINT)>PL-15-2*NMPU THEN PUT EDIT(' ')(LINE(PL),A);
   PUT SKIP(3) EDIT('SIMULATION STATISTICS')(COL(19),A);
   PUT SKIP(2) EDIT('MPU','INST','INPT','OTPT','ERRS','WARN',
        'BRCH','INTS','RTRN')(COL(2),A,(8)(X(2),A(4)));
   PUT SKIP EDIT((60)'=')(A);
   DO MPU=0 TO NMPU-1;
        PUT SKIP EDIT(MPU,INST(MPU),INPT(MPU),OTPT(MPU),ERRS(MPU),
             WARN(MPU),BRCH(MPU),INTS(MPU),RTRN(MPU))
             (F(3,0),(8)(F(6,0)));
        END;
   PUT SKIP EDIT((60)'=')(A);
   PUT SKIP(2) EDIT('MPU','RESET','RUNNING','HALTED','LAST RESTART',
        'LAST HALT')(COL(2),A,COL(6),A,COL(13),A,COL(26),A,COL(36),A,
        COL(50),A);
   PUT SKIP EDIT((60)'=')(A);
   DO MPU=0 TO NMPU-1;
        IF SUBSTR(SYSTBL(7,MPU),4,1)='0'B THEN DO;
             RUN(MPU)=TNOW+RUN(MPU)-RESTRT(MPU);
             LSTHLT(MPU)=TNOW;
             END;
        ELSE DO;
             HALT(MPU)=TNOW+HALT(MPU)-LSTHLT(MPU);
             END;
        PUT SKIP EDIT(MPU,RSTS(MPU),RUN(MPU),HALT(MPU),RESTRT(MPU),
             LSTHLT(MPU))(F(3,0),X(1),F(3,0),(4)(X(1),E(12,5)));
        END;
   PUT SKIP EDIT((60)'=')(A);
   CALL USUM;
   PUT EDIT(' ') (LINE(1),A);
   RETURN;
   END OTPUT;
```

```
/*  PDUMP  VERSION-2                    3/25/78                 */
PDUMP: PROCEDURE;
/*******************************************************/
/*                                                     */
/*  THIS PROCEDURE PRINTS THE PERIPHERAL               */
/*  CONFIGURATION OF THE SYSTEM.                       */
/*                                                     */
/*******************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (I,J) FIXED BIN(31,0);
   DCL (MPU,NMPU,NPER,PL) FIXED BIN(31,0) EXT;
   DCL TYPE CHAR(1);
   DCL CODE BIT(3);
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL MFIND ENTRY(FIXED BIN(31,0),FIXED BIN(31,0))
        RETURNS(FIXED BIN(31,0));
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL 1 MEMREC EXT,
        2 UNUSED BIT(16),
        2 MEM_KEY CHAR(10),
        2 LMPU FIXED BIN(31,0),
        2 LOC(0:31) BIT(16);
   DCL ADDRESS FIXED BIN(31,0);
   DCL TEMP BIT(8);
   DCL 1 PERTBL(*) CTL EXT,
        2 PMPU FIXED BIN(31,0),
        2 PADDR FIXED BIN(31,0),
        2 PASSOC FIXED BIN(31,0),
        2 PASADD FIXED BIN(31,0),
        2 PCODE FIXED BIN(31,0),
        2 CTYPE BIT(1),
        2 UNUSED BIT(5),
        2 C2 BIT(1),
        2 C1 BIT(1),
        2 DDR BIT(8),
        2 PDR BIT(8);
/*******************************************************/
/*                                                     */
/*  PRINT HEADINGS....                                 */
/*                                                     */
/*******************************************************/
   IF LINENO(SYSPRINT)>PL-10-NPER THEN PUT EDIT(' ')(LINE(1),A);
   PUT SKIP(2) EDIT('PERIPHERAL CONFIGURATION',(24)'*')
      (SKIP(4),COL(18),A,SKIP,COL(18),A);
/*******************************************************/
/*                                                     */
/*  PRINT PIA INFORMATION....                          */
/*                                                     */
```

```
/*****************************************************/
        PUT SKIP(3) EDIT('PIA UNITS')(COL(25),A);
        PUT SKIP EDIT('ADDR','ASSOC','ASADD','SIDE',
             'CR','DDR','PDR','C1','C2','CTYPE','MPU')
             (COL(2),A,COL(7),A,COL(13),A,COL(21),A,
             COL(28),A,COL(36),A,COL(45),A,COL(52),A,COL(55),
             A,X(2),A,X(2),A);
        PUT SKIP EDIT((70)'=')(A);
        ADDRESS=-1;
   DO MPU=0 TO NMPU-1;
        J=32;
        IF SYSTBL(6,MPU)\=0 THEN
             DO I= 0 TO NPER;
             IF PMPU(I)=MPU THEN
                  IF PCODE(I)=2 ^ PCODE(I)=1 THEN DO;
                  IF PCODE(I)=1 THEN TYPE='A';
                  ELSE TYPE='B';
                  PUT SKIP EDIT(HHEX(PADDR(I)),
                       PASSOC(I),HHEX(PASADD(I)),
                            TYPE)(COL(1),A,COL(8),F(3,0),COL(12),
                            A,COL(22),A);
                  ADDRESS=PADDR(I)+1;
                  J=MFIND(ADDRESS,1);
                  IF J>=0 THEN
                       PUT EDIT(SUBSTR(LOC(J),5,8),DDR(I),
                            PDR(I),C1(I),C2(I),CTYPE(I),MPU)
                            (COL(25),B(8),COL(34),B(8),
                       COL(43),B(8),COL(53),B,COL(56),B,X(5),B(1),
                       X(2),F(3,0));
                  END;
             END;
        END;
   IF ADDRESS<0 THEN PUT SKIP EDIT('NONE')(COL(30),A);
   PUT SKIP EDIT((70)'=')(A);
/*****************************************************/
/*                                                   */
/*  PRINT ACIA INFORMATION.....                      */
/*                                                   */
/*****************************************************/
        PUT SKIP(3) EDIT('ACIA UNITS')(COL(25),A);
        PUT SKIP EDIT('ADDR','ASSOC','ASADD','CR','SR',
             'TDR','RDR','MPU')
             (COL(2),A,COL(7),A,COL(13),A,COL(22),A,COL(31),A,
             COL(40),A,COL(49),A,COL(56),A);
        PUT SKIP EDIT((60)'=')(A);
   ADDRESS=-1;
   DO MPU=0 TO NMPU-1;
        J=32;
```

```
     IF SYSTBL(6,MPU)\=0 THEN
          DO I=0 TO NPER;
          IF PMPU(I)=MPU THEN
               IF PCODE(I)=3 THEN DO;
               PUT SKIP EDIT(HHEX(PADDR(I)),
                    PASSOC(I),HHEX(PASADD(I)),
                    PDR(I),DDR(I))(COL(1),A,COL(8),F(3,0),COL(12),A,
                    COL(19),B,COL(28),B);
               ADDRESS=PADDR(I);
               J=MFIND(ADDRESS,1);
               IF J>=0 THEN
                    SUBSTR(TEMP,5,4)=SUBSTR(LOC(J),13,4);
               J=MFIND(ADDRESS,1);
               IF J>=0 THEN
                    SUBSTR(TEMP,1,4)=SUBSTR(LOC(J),13,4);
               PUT EDIT(TEMP,SUBSTR(LOC(J),5,8),MPU)
                    (COL(37),B(8),COL(46),B(8),COL(55),F(3,0));
               END;
          END;
     END;
IF ADDRESS<0 THEN PUT SKIP EDIT('NONE')(COL(30),A);
     PUT SKIP  EDIT((60)'=')(A);
RETURN;
END PDUMP;
```

```
/*  READ VERSION-2                              4/7/78          */
READ:  PROCEDURE(ADDR) RETURNS(FIXED BIN(8,0));
/*****************************************************/
/*                                                   */
/*  THIS SUBPROGRAM READS THE MEMORY LOCATION AND    */
/*  DETERMINES WHAT TYPE OF LOCATION IT IS AND       */
/*  TAKES CARE OF ALL THE PERIPHERAL ACCESSES.       */
/*  ERRORS ARE FLAGGED IN SYSTBL(8,MPU).             */
/*                                                   */
/*****************************************************/
   DCL 1 MEMREC EXT,
        2 UNUSED BIT(16),
        2 MEM_KEY CHAR(10),
        2 LMPU FIXED BIN(31,0),
        2 LOC(0:31) BIT (16);
   DCL (I,J) FIXED BIN(31,0);
   DCL TNOW FLOAT BIN EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL MPU FIXED BIN(31,0) EXT;
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL 1 PERTBL(*) CTL EXT,
        2 PMPU FIXED BIN(31,0),
        2 PADDR FIXED BIN(31,0),
        2 PASSOC FIXED BIN(31,0),
        2 PASADD FIXED BIN(31,0),
        2 PCODE FIXED BIN(31,0),
        2 CTYPE BIT(1),
        2 UNUSED BIT(5),
        2 C2 BIT(1),
        2 C1 BIT(1),
        2 DDR BIT(8),
        2 PDR BIT(8);
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL (STR7,STR8) BIT(31);
   DCL ADDR FIXED BIN(16,0);
   DCL (DATA,ASMPU) FIXED BIN(8,0);
   DCL TEMP BIT(8);
   DCL CODE BIT(3);
   DCL ASADD FIXED BIN(16,0);
   DCL LINK FIXED BIN(31,0);
   DCL HHEX ENTRY(FIXED BIN(31,0)) RETURNS(CHAR(6));
   DCL MFIND ENTRY(FIXED BIN(31,0),
        FIXED BIN(31,0)) RETURNS(FIXED BIN(31,0));
   DATA=0;
   STR7=BIT(SYSTBL(7,MPU),31);
   STR8=BIT(SYSTBL(8,MPU),31);
   I=MFIND(ADDR,1);
```

```
   IF I<0 THEN SUBSTR(STR8,2,1)='1'B;
/*****************************************************/
/*                                                   */
/*  ERROR(02)=---- ATTEMPT TO ACCESS A NONEXISTENT   */
/*                 MEMORY LOCATION.                  */
/*                                                   */
/*****************************************************/
   ELSE DO;
        CODE=SUESTR(LOC(I),1,3);
        IF SUBSTR(CODE,2,1)='1'B THEN DO;
/*****************************************************/
/*                                                   */
/*  PERIPHERAL CHIP ACCESSED.                        */
/*                                                   */
/*****************************************************/
             IF SUBSTR(LOC(I),3,1)='1'B THEN DO;
/*****************************************************/
/*                                                   */
/*  SECONDARY PERIPHERAL LOCATION.                   */
/*                                                   */
/*****************************************************/
                  ADDR=ADDR-1;
                  I=MFIND(ADDR,1);
                  LINK=BIN(SUBSTR(LOC(I),4,9),31,0);
                  IF PCODE(LINK)=3 THEN DO;
/*****************************************************/
/*                                                   */
/*  ACIA ACCESSED.                                   */
/*                                                   */
/*****************************************************/
                       DATA=BIN(SUBSTR(LOC(I),13,4),8,0);
                       ADDR=ADDR+1;
                       I=MFIND(ADDR,1);
                       DATA=DATA+BIN(SUBSTR(LOC(I),13,4),8,0)*16;
                       SUBSTR(DDR(LINK),1,1)='0'B;
                       SUBSTR(STR7,11,1)='1'B;
                       END;
                  ELSE DO;
                       ADDR=ADDR+1;
                       I=MFIND(ADDR,1);
                       IF PCODE(LINK)=1 ^ PCODE(LINK)=2 THEN
                            DATA=BIN(SUBSTR(LOC(I),5,8),8,0);
/*****************************************************/
/*                                                   */
/*  P1A ACCESSED.                                    */
/*                                                   */
/*****************************************************/
                       ELSE SUBSTR(STR8,6,1)='1'B;
```

```
/*****************************************************/
/*                                                   */
/*  BAD PERIPHERAL ACCESSED.                         */
/*                                                   */
/*****************************************************/
                    END;
                END;
            ELSE DO;
/*****************************************************/
/*                                                   */
/*  FIRST LOCATION PERIPHERAL CHIP.                  */
/*                                                   */
/*****************************************************/
                LINK=BIN(SUBSTR(LOC(I),4,9),31,0);
                IF PCODE(LINK)=3 THEN DATA=BIN(PDR(LINK),8,0);
/*****************************************************/
/*                                                   */
/*  ACIA ACCESSED.                                   */
/*                                                   */
/*****************************************************/
                ELSE DO;
/*****************************************************/
/*                                                   */
/*  PIA ACCESSED.                                    */
/*                                                   */
/*****************************************************/
                    IF PCODE(LINK)=1 ^ PCODE(LINK)=2 THEN DO;
                        ADDR=ADDR+1;
                        I=MFIND(ADDR,1);
                        IF SUBSTR(LOC(I),10,1)='1'B THEN DO;
                            TEMP=BIN(PDR(LINK),8,0);
                            SUBSTR(LOC(I),5,2)='00'B;
                            SUBSTR(STR7,11,1)='1'B;
                            IF PCODE(LINK)=2 THEN DO I=1 TO 8;
                                IF SUBSTR(DDR(LINK),I,1)='1'B
                                    THEN SUBSTR(TEMP,I,1)='1'B;
                                END;
                            ELSE DO;
                                IF SUBSTR(LOC(I),7,3)='100'B
                                    THEN DO;
/*****************************************************/
/*                                                   */
/*  SCHEDULE C1 RESET                                */
/*                                                   */
/*****************************************************/
                                        IF CYCLES>0 THEN ATRIB(1)=
                                            TNOW+(CYCLES+1)
                                            *0.000001;
```

```
                                        ELSE ATRIB(1)=TNOW +
                                             0.000001*4;
                                        ATRIB(2)=300;
                                        ATRIB(3)=PASSOC(LINK);
                                        IF CTYPE(LINK)='1'B THEN
                                             ATRIB(4)=3;
                                        ELSE ATRIB(4)=5;
                                        ATRIB(6)=PASADD(LINK);
                                        CALL FILEM(1);
                                        END;
                                   IF SUBSTR(LOC(I),7,3)='101'B
                                        THEN DO;
/*****************************************************/
/*                                                   */
/*  SCHEDULE PULSE START AND END                     */
/*                                                   */
/*****************************************************/
                                        IF CYCLES>0 THEN ATRIB(1)=
                                             TNOW+(CYCLES+1)
                                             *0.000001;
                                        ELSE ATRIB(1)=TNOW+
                                             4*0.000001;
                                        ATRIB(2)=300;
                                        ATRIB(3)=PASSOC(LINK);
                                        IF CTYPE(LINK)='1'B THEN
                                             ATRIB(4)=3;
                                        ELSE ATRIB(4)=5;
                                        ATRIB(6)=PASADD(LINK);
                                        CALL FILEM(1);
                                        ATRIB(1)=ATRIB(1)+
                                             0.000001;
                                        ATRIB(4)=ATRIB(4)-1;;
                                        CALL FILEM(1);
                                        ATRIB(3)=MPU;
                                        ATRIB(4)=4;
                                        ATRIB(6)=PADDR(LINK);
                                        CALL FILEM(1);
                                        END;
                                   C2(LINK)='0'B;
                                   END;
                              DATA=BIN(TEMP,8,0);
                              END;
                         ELSE DATA=BIN(DDR(LINK),8,0);
                         END;
                    ELSE SUBSTR(STR8,6,1)='1'B;
/*****************************************************/
/*                                                   */
/*  BAD PERIPHERAL CHIP ACCESSED.                    */
```

```
/*                                                              */
/****************************************************************/
                        END;
                    END;
                END;
            ELSE DATA=BIN(SUBSTR(LOC(I),9,8),8,0);;
            END;
    SYSTBL(7,MPU)=BIN(STR7,31,0);
    SYSTBL(8,MPU)=BIN(STR8,31,0);
    RETURN(DATA);
    END READ;
```

```
/*    TRACE    VERSION-1              12/21/77              */
TRACE:  PROCEDURE;
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE PERFORMS PROGRAM TRACING DURING   */
/*  SIMULATION RUNS.   IT USES THE SYSTEM TABLE,     */
/*  'TRCE(*:*)' AND HAS THE FOLLOWING STRUCTURE,     */
/*                                                   */
/*       TRCE(1,MPU)  IS THE TRACE TYPE.             */
/*                                                   */
/*      TRCE(1,MPU)= 0    TRACE INHIBITED            */
/*      TRCE(1,MPU)= 1    ADDRESS START & STOP       */
/*      TRCE(1,MPU)= 2    TIME START & STOP          */
/*      TRCE(1,MPU)= 3    BRANCH TRACE ONLY          */
/*                                                   */
/*      TRCE(2,MPU)  IS THE START ADDRESS OR TIME.   */
/*                                                   */
/*      TRCE(3,MPU)  IS THE STOP ADDRESS OR TIME.    */
/*                                                   */
/*                                                   */
/*  TRCE(1,MPU) WILL ALSO BE USED AS A FLAG DURING   */
/*  RUNS TO INDICATE IF THE TRACE IS ON OR OFF.      */
/*                                                   */
/*****************************************************/
   DCL (NNQ(*),SYSTBL(*,*)) FIXED BIN(31,0) CTL EXT;
   DCL (SP(*),INR(*),PC(*)) FIXED BIN(16,0) CTL EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL OPC FIXED BIN(16,0) EXT;
   DCL (CYCLES,MPU,NMPU,NNTRY) FIXED BIN(31,0) EXT;
   DCL TRCPRT ENTRY;
   DCL 1 STAT(*) CTL EXT,
        2 INST FIXED BIN(31,0),
        2 INPT FIXED BIN(31,0),
        2 OTPT FIXED BIN(31,0),
        2 ERRS FIXED BIN(31,0),
        2 INTS FIXED BIN(31,0),
        2 WARN FIXED BIN(31,0),
        2 BRCH FIXED BIN(31,0),
        2 RTRN FIXED BIN(31,0),
        2 RSTS FIXED BIN(31,0),
        2 HALT FLOAT BIN(31),
        2 RUN FLOAT BIN(31),
        2 RESTRT FLOAT BIN(31),
        2 LSTHLT FLOAT BIN(31);
   DCL (ACCA(*),ACCB(*)) FIXED BIN(8,0) CTL EXT;
   DCL (CODE,OPRND1,OPRND2) FIXED BIN(8,0) EXT;
   DCL TRCE(*,*) FLOAT BIN(31) CTL EXT;
   DCL TEMP BIT(31);
```

```
   DCL (I,J) FIXED BIN(31,0);
   DCL TNOW BIN FLOAT EXT;
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL NNOP FIXED BIN(31,0);
   DCL CCR(*) BIT(8) CTL EXT;
   INST(MPU)=INST(MPU)+1;
/*****************************************************/
/*                                                   */
/*  CHECK TO SEE IF THE FILING ARRAY HAS ROOM FOR    */
/*  ANOTHER TRACE ENTRY.                             */
/*                                                   */
/*****************************************************/
   IF SUM(NNQ)>=NNTRY-20 THEN DO;
        J=MPU;
        DO I=0 TO NMPU-1;
             IF NNQ(I+3)>NNQ(MPU+3) THEN MPU=I;
             END;
        CALL TRCPRT;
        MPU=J;
        END;
/*****************************************************/
/*                                                   */
/*  FILE TRACE INFORMATION IN THE TRACE FILE.        */
/*                                                   */
/*****************************************************/
   I=0;
   IF TRCE(1,MPU)>0 ^ SYSTBL(8,MPU)\=0 THEN DO;
        IF TRCE(1,MPU)=1 THEN DO;
             IF TRCE(2,MPU)<= PC(MPU) & TRCE(3,MPU)>=PC(MPU) THEN
                  TRCE(1,MPU)=6;
             ELSE I=-1;
             END;
        IF TRCE(1,MPU)=2 THEN DO;
             IF TRCE(2,MPU)<=TNOW & TRCE(3,MPU)>=TNOW
                  THEN TRCE(1,MPU)=7;
             ELSE I=-1;
             END;
        IF TRCE(1,MPU)=3 THEN DO;
             TEMP=BIT(SYSTBL(7,MPU),31);
             IF BOOL(SUBSTR(TEMP,7,3),'111'B,'0001'B)='000'B
                  & SUBSTR(TEMP,13,1)='0'B THEN I=-1;
             END;
        IF SYSTBL(8,MPU)=0 & I<0 THEN RETURN;
        NNOP=2;
        IF OPRND1<0 THEN NNOP=0;
        ELSE IF OPRND2<0 THEN NNOP=1;
        IF NNOP=0 THEN OPRND1,OPRND2=0;
        IF NNOP=1 THEN OPRND2=0;
```

```
        ATRIB(1)=1;
        ATRIB(2)=TNOW+CYCLES*0.000001;
        ATRIB(3)=OPC;
        ATRIB(4)=NNOP;
        ATRIB(5)=OPRND1;
        ATRIB(6)=OPRND2;
        ATRIB(7)=SP(MPU);
        ATRIB(8)=PC(MPU);
        CALL FILEM(MPU+3);
        ATRIB(1)=2;
        ATRIB(2)=INR(MPU);
        ATRIB(3)=ACCA(MPU);
        ATRIB(4)=ACCB(MPU);
        ATRIB(5)=CCR(MPU);
        ATRIB(6)=CODE;
        ATRIB(7)=SYSTBL(7,MPU);
        ATRIB(8)=SYSTBL(8,MPU);
        CALL FILEM(MPU+3);
/*****************************************************/
/*                                                   */
/*  CHECK TO SEE IF THE SIMULATION TIME HAS PASSED   */
/*  THE STOP TIME FOR TRACING IF A TIME TRACE IS     */
/*  SPECIFIED.                                       */
/*                                                   */
/*****************************************************/
        IF TRCE(1,MPU)=7 THEN
             IF TRCE(3,MPU)<TNOW+0.000001*CYCLES
                  THEN TRCE(1,MPU)=2;
/*****************************************************/
/*                                                   */
/*  CHECK TO SEE IF THE PROGRAM COUNTER HAS PASSED   */
/*  THE STOP ADDRESS FOR AN ADDRESS TRACE, IF        */
/*  ENABLED.  STOP THE TRACE IF IT HAS.              */
/*                                                   */
/*****************************************************/
        IF TRCE(1,MPU)=6 THEN
             IF TRCE(3,MPU)<PC(MPU) ^ TRCE(2,MPU)>PC(MPU)
                  THEN TRCE(1,MPU)=1;
        END;
   RETURN;
   END TRACE;
```

```
/*   TRCPRT  VERSION-1                2/21/78                */
TRCPRT: PROCEDURE;
/*******************************************************/
/*                                                     */
/*  THIS PROCEDURE PRINTS TRACE INFORMATION FOR        */
/* THE MPUS DURING RUNS.                               */
/*                                                     */
/*******************************************************/
   DCL NNQ(*) FIXED BIN(31,0) CTL EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL (J,TEMP) FIXED BIN(31,0);
   DCL TNOW FLOAT BIN EXT;
   DCL (NFILE,MAX,I) FIXED BIN(31,0);
   DCL RMOVE ENTRY(PTR,FIXED BIN(31,0));
   DCL (MPU,NMPU,PL) FIXED BIN(31,0) EXT;
   DCL MFE(*) PTR CTL EXT;
   DCL 1 LINE,
        2 X1 CHAR(1) INIT(' '),
        2 OPC CHAR(6),
        2 X2 CHAR(2) INIT(' '),
        2 INS CHAR(2),
        2 X3 CHAR(3) INIT(' '),
        2 OP1 CHAR(2),
        2 X4 CHAR(3) INIT(' '),
        2 OP2 CHAR(2),
        2 X5  CHAR(1) INIT(' '),
        2 NPC CHAR(6),
        2 SP CHAR(6),
        2 INR CHAR(6),
        2 X8 CHAR(2) INIT(' '),
        2 ACCA CHAR(2),
        2 X9 CHAR(3) INIT(' '),
        2 ACCB CHAR(2),
        2 X6 CHAR(2) INIT(' '),
        2 CCR BIT(8);
   DCL UERR ENTRY(FIXED BIN(31,0));
   DCL CONPRT ENTRY(BIT(31));;
   DCL ERRPRT ENTRY(BIT(31));
   IF MPU<0 ^ MPU>=NMPU THEN CALL UERR(2);
   ELSE DO;
        NFILE=MPU+3;
        IF NNQ(NFILE)\=0 THEN DO;
/*******************************************************/
/*                                                     */
/*  PRINT TRACE PRINT HEADINGS.                        */
/*                                                     */
/*******************************************************/
```

```
          PUT EDIT(' ') (LINE(1),A);
          PUT SKIP(2) EDIT('MPU:  ',MPU,'; TRACE PRINT AT ',TNOW,
               'SEC.')(COL(10),A,F(2,0),A,F(12,6),X(1),A);
          PUT SKIP(3) EDIT('REGISTERS AFTER EXECUTION')
               (COL(41),A);
          PUT SKIP EDIT((37)'=')(COL(36),A);
          PUT SKIP EDIT('TIME','PC','INST','OP1','OP2','PC',
               'SP','IX','ACCA','ACCB','XXHINZVC')
               (COL(6),A,COL(16),A,COL(21),A,COL(27),A,COL(32),
               A,COL(37),A,COL(43),A,COL(49),A,COL(54),A,COL(59),A,
               COL(64),A);
          PUT SKIP EDIT((72)'=')(A);
/*****************************************************/
/*                                                   */
/*  PRINT TRACE INFORMATION.                         */
/*                                                   */
/*****************************************************/
          MAX=NNQ(NFILE)/2;
          IF MAX>PL-5 THEN MAX=PL-5;
          DO J=1 TO MAX;
               CALL RMOVE(MFE(NFILE),NFILE);
               PUT SKIP EDIT(ATRIB(2))(F(12,6));
               IF ATRIB(4)>0 THEN DO;
                    OP1=SUBSTR(HHEX(ATRIB(5)),4,2);
                    IF ATRIB(4)>1 THEN
                         OP2=SUBSTR(HHEX(ATRIB(6)),4,2);
                    ELSE OP2=' ';
                    END;
               ELSE OP1,OP2=' ';
               OPC=HHEX(ATRIB(3));
               SP=HHEX(ATRIB(7));
               NPC=HHEX(ATRIB(8));
               CALL RMOVE(MFE(NFILE),NFILE);
               INR=HHEX(ATRIB(2));
               ACCA=SUBSTR(HHEX(ATRIB(3)),4,2);
               ACCB=SUBSTR(HHEX(ATRIB(4)),4,2);
               INS=SUBSTR(HHEX(ATRIB(6)),4,2);
               I=ATRIB(5);
               CCR=SUBSTR(I,24,8);
               PUT EDIT(LINE)(A);
               TEMP=ATRIB(8);
               IF ATRIB(8)\=0 THEN CALL ERRPRT(BIT(TEMP,31));
               TEMP=ATRIB(7);
               IF ATRIB(7)\=0 THEN CALL CONPRT(BIT(TEMP,31));
               IF LINENO(SYSPRINT)>PL-10 THEN J=PL;
               END;
          PUT SKIP EDIT((72)'=')(A);
          PUT EDIT(' ')(LINE(PL-1),A);
```

```
          END;
     ELSE PUT SKIP(2) EDIT('TRACE FILE IS EMPTY.')(COL(10),A);
     END;
END TRCPRT;
```

```
/*   UERR    VERSION-1                2/19/78                */
UERR:  PROCEDURE(CODE);
   DCL CODE FIXED BIN(31,0);
   PUT SKIP(2) EDIT('*********  RUN ERROR NUMBER ',CODE)
        (A,F(3,0));
   END UERR;
```

```
/*     UMONT   VERSION-1                  2/19/78          */
UMONT: PROCEDURE(IX);
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE PRINTS EVENT MONITORIN            */
/*  INFORMATION.                                     */
/*                                                   */
/*****************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL (PC(*),SP(*),INR(*)) FIXED BIN(16,0) CTL EXT;
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL TNOW FLOAT BIN EXT;
   DCL (JEVNT,MPU,NNATR,NMPU) FIXED BIN(31,0) EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL (I,IX) FIXED BIN(31,0);
   PUT SKIP(2) EDIT('TNOW=',TNOW)(COL(10),A,E(12,4));
   IF JEVNT>9 &JEVNT<NMPU+10 THEN DO;
        PUT EDIT('MPU=',JEVNT-10,'STATUS=',SYSTBL(7,JEVNT-10),
           'INTERRUPTS PENDING:  NMI=',SYSTBL(10,JEVNT-10),
            'MI=',SYSTBL(9,JEVNT-10))
            (X(3),A,F(5,0),X(3),A,B(31),X(3),A,F(3,0),X(2),A,F(3,0));
        PUT SKIP EDIT('PC='^^HHEX(PC(JEVNT-10)),'SP='^^
             HHEX(SP(JEVNT-10)),'INR='^^HHEX(INR(JEVNT-10)),
             'SYSTBL(8,MPU)=',SYSTBL(8,JEVNT-10))
             (COL(10),A,X(1),A,X(1),A,X(1),A,B(31));
        END;
   ELSE PUT SKIP(2) EDIT('JEVNT=',JEVNT,('ATRIB(',I,')=',
        ATRIB(I)  DO I=1 TO NNATR))
        (X(7),A,F(5,0),X(3),(NNATR)(X(2),A,F(2,0),A,E(12,4)));
   IX=0;
   END UMONT;
```

```
USUM:  PROCEDURE;
/*****************************************************/
/*                                                   */
/*  THIS PROCEDURE PRINTS SUMMARY REPORTS FOR THE    */
/*  BIMS.                                            */
/*                                                   */
/*****************************************************/
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL FIND ENTRY(FLOAT BIN,FIXED BIN(31,0),PTR,FIXED BIN(31,0),
        FLOAT BIN) RETURNS(PTR);
   DCL 1 BUSTBL(0:2) EXT,
        2 ACTIVE BIT(1),
        2 CONTR FIXED BIN(31,0);
   DCL 1 BIMTBL(0:3) EXT,
        2 BDATA   FIXED BIN(31,0),
        2 STATUS BIT(8),
        2 RBFST FIXED BIN(16,0);
   DCL RMOVE ENTRY(PTR,FIXED BIN(31,0));
   DCL SNTRY POINTER;
   DCL NULL BUILTIN;
   DCL MFE(*) POINTER CTL EXT;
   DCL (MPU,NMPU,PL) FIXED BIN(31,0) EXT;
   DCL I FIXED BIN(31,0);
   IF LINENO(SYSPRINT)>PL-10-NMPU THEN PUT EDIT(' ') (LINE(PL-1),A);
   PUT SKIP(3) EDIT('BIM  I/O')(COL(8),A);
   PUT SKIP EDIT('BIM','OUTPUTS','INPUTS')(COL(4),A,
        COL(12),A,COL(23),A);
   PUT SKIP EDIT((30)'=')(A);
   DO I=0 TO NMPU-1;
        SNTRY=FIND(10+I,5,MFE(2),1,0);
        IF SNTRY\=NULL THEN DO;
             CALL RMOVE(SNTRY,2);
             END;
        ELSE ATRIB(1)=-1;
        PUT SKIP EDIT(I)(COL(3),F(3,0));
        IF ATRIB(1)>0 THEN PUT EDIT(ATRIB(2),ATRIB(3))
                  (COL(9),F(7,0),COL(19),F(7,0));
        ELSE PUT EDIT('NO VALUES RECORDED')(COL(9),A);
        END;
   PUT SKIP EDIT((30)'=')(A);
   END USUM;
```

```
/*   WRITE    VERSION-1              1/13/78                */
WRITE: PROCEDURE(DATA,ADDRESS);
/*******************************************************/
/*                                                     */
/*  THIS SUBPROGRAM WRITES THE DATA ONTO MEMORY,       */
/*  ACCESSES PERIPHERALS AND FLAGS ERRORS OR           */
/*  CONDITIONS IN THE SYSTEM TABLES.                   */
/*                                                     */
/*******************************************************/
   DCL SYSTBL(*,*) FIXED BIN(31,0) CTL EXT;
   DCL 1 PERTBL(*) CTL EXT,
        2 PMPU FIXED BIN(31,0),
        2 PADDR FIXED BIN(31,0),
        2 PASSOC FIXED BIN(31,0),
        2 PASADD FIXED BIN(31,0),
        2 PCODE FIXED BIN(31,0),
        2 CTYPE BIT(1),
        2 UNUSED BIT(5),
        2 C2 BIT(1),
        2 C1 BIT(1),
        2 DDR BIT(8),
        2 PDR BIT(8);
   DCL HHEX ENTRY(FIXED BIN(16,0)) RETURNS(CHAR(6));
   DCL FILEM ENTRY(FIXED BIN(31,0));
   DCL MFIND ENTRY(FIXED BIN(31,0),FIXED BIN(31,0))
        RETURNS(FIXED BIN(31,0));
   DCL MEM1 FILE RECORD KEYED ENV(REGIONAL(1) F(80));
   DCL TNOW FLOAT BIN EXT;
   DCL ATRIB(*) FLOAT BIN CTL EXT;
   DCL FLAG BIT(3);
   DCL DATA FIXED BIN(8,0);
   DCL ADDRESS FIXED BIN(16,0);
   DCL (CYCLES,MPU) FIXED BIN(31,0) EXT;
   DCL 1 MEMREC EXT,
        2 UNUSED BIT(16),
        2 MEM_KEY CHAR(10),
        2 LMPU FIXED BIN(31,0),
        2 LOC(0:31) BIT(16);
   DCL (I,J,LINK) FIXED BIN(31,0);
   DCL (STR7,STR8) BIT(31);
/*******************************************************/
/*                                                     */
/*  READ MEMORY LOCATION                               */
/*                                                     */
/*******************************************************/
   I=MFIND(ADDRESS,1);
   STR8=BIT(SYSTBL(8,MPU),8);
   STR7=BIT(SYSTBL(7,MPU),8);
```

```
   IF I<0 THEN SUBSTR(STR8,2,1)='1'B;
/*****************************************************/
/*                                                   */
/*  NON-EXISTENT MEMORY LOCATION ACCESSED.           */
/*                                                   */
/*****************************************************/
   ELSE DO;
        FLAG=SUBSTR(LOC(I),1,3);
        IF FLAG='000'B THEN SUBSTR(LOC(I),9,8)=BIT(DATA,8);
/*****************************************************/
/*                                                   */
/*  RAM ACCESSED.                                    */
/*                                                   */
/*****************************************************/
        IF FLAG='001'B THEN SUBSTR(STR8,5,1)='1'B;
/*****************************************************/
/*                                                   */
/*  ATTEMPT TO WRITE ON ROM...DATA NOT WRITTEN.      */
/*                                                   */
/*****************************************************/
        IF FLAG='010' ^ FLAG='011'B THEN DO;
/*****************************************************/
/*                                                   */
/*  PERIPHERAL CHIP ACCESSED.                        */
/*                                                   */
/*****************************************************/
             IF FLAG='011'B THEN DO;
/*****************************************************/
/*                                                   */
/*  SECOND LOCATION PERIPHERAL ACCESSED.             */
/*                                                   */
/*****************************************************/
                  I=MFIND(ADDRESS-1,1);
                  LINK=BIN(SUBSTR(LOC(I),4,9),31,0);
                  I=MFIND(ADDRESS,1);
                  IF PCODE(LINK)=1 ^ PCODE(LINK)=2 THEN DO;
/*****************************************************/
/*                                                   */
/*  PIA ACCESSED (SECOND LOCATION).                  */
/*  WRITE DATA INTO CONTROL REGISTER BITS 0 TO 5     */
/*                                                   */
/*****************************************************/
                       SUBSTR(LOC(I),7,6)=SUBSTR(DATA,3,6);
                       IF SUBSTR(LOC(I),7,1)='1'B THEN DO;
/*****************************************************/
/*                                                   */
/*  CHANGE C2 TO CORRECT VALUE FOR OUTPUT MODE.      */
/*                                                   */
```

```
/*****************************************************/
                          ATRIB(1)=TNOW;
                          ATRIB(2)=300;
                          ATRIB(3)=PASSOC(LINK);
                          IF CTYPE(LINK)='1'B THEN DO;
                               IF SUBSTR(LOC(I),9,1)='1'B THEN DO;
                                    ATRIB(4)=2;
                                    C2(LINK)='1'B;
                                    END;
                               ELSE DO;
                                    ATRIB(4)=3;
                                    C2(LINK)='0'B;
                                    END;
                               END;
                          ELSE DO;
                               IF SUBSTR(LOC(I),9,1)='1'B THEN DO;
                                    ATRIB(4)=4;
                                    C2(LINK)='1'B;
                                    END;
                               ELSE DO;
                                    ATRIB(4)=5;
                                    C2(LINK)='0'B;
                                    END;
                               END;
                          ATRIB(5)=0;
                          ATRIB(6)=PASADD(LINK);
                          CALL FILEM(1);
                          END;
                     END;
                ELSE DO;
                          IF PCODE(LINK)=3 THEN DO;
/*****************************************************/
/*                                                   */
/*  ACIA ACCESSED (SECOND LOCATION).                 */
/*  WRITE DATA INTO TRANSMIT DATA REGISTER AND       */
/*  SCHEDULE A DATA TRANSFER EVENT FOR ASSOCIATE.    */
/*                                                   */
/*****************************************************/
                          SUBSTR(LOC(I),5,8)=BIT(DATA,8);
                          SUBSTR(DDR(LINK),7,1)='1'B;
/*****************************************************/
/*                                                   */
/*  SCHEDULE DATA TRANSFER EVENT FOR ASSOCIATE.      */
/*                                                   */
/*****************************************************/
                          IF CYCLES>0
                               THEN ATRIB(1)=TNOW+CYCLES*0.000001;
                          ELSE ATRIB(1)=TNOW+0.000004;
```

```
                        ATRIB(2)=300;
                        ATRIB(3)=PASSOC(LINK);;
                        ATRIB(4)=1;
                        ATRIB(5)=DATA;
                        ATRIB(6)=PASADD(LINK);
                        CALL FILEM(1);
                        SUBSTR(STR7,10,1)='1'B;
                        END;
                    ELSE SUBSTR(STR8,6,1)='1'B;
/*****************************************************/
/*                                                   */
/*  BAD PERIPHERAL DEVICE ACCESSED.                  */
/*                                                   */
/*****************************************************/
                    END;
                END;
            ELSE DO;
/*****************************************************/
/*                                                   */
/*  FIRST LOCATION PERIPHERAL CHIP ACCESSED.         */
/*                                                   */
/*****************************************************/
                LINK=BIN(SUBSTR(LOC(I),4,9),31,0);
                IF PCODE(LINK)=1 ^ PCODE(LINK)=2 THEN DO;
/*****************************************************/
/*                                                   */
/*  PIA ACCESSED.                                    */
/*                                                   */
/*****************************************************/
                    I=MFIND(ADDRESS+1,1);
                    IF SUBSTR(LOC(I),10,1)='1'B THEN DO;
/*****************************************************/
/*                                                   */
/*   PERIPHER DATA REGISTER ACCESSED.                */
/*                                                   */
/*****************************************************/
                        DO J=1 TO 8;
                            IF SUBSTR(DDR(LINK),J,1)='1'B THEN
                                SUBSTR(PDR(LINK),J,1)=
                                SUBSTR(DATA,J,1);
                            END;
/*****************************************************/
/*                                                   */
/*  SCHEDULE DATA TRANSFER EVENT.                    */
/*                                                   */
/*****************************************************/
                        IF CYCLES>0 THEN
                            ATRIB(1)=TNOW+CYCLES*0.000001;
```

```
                        ELSE ATRIB(1)=TNOW+0.000004;
                        ATRIB(2)=300;
                        ATRIB(3)=PASSOC(LINK);
                        ATRIB(4)=1;
                        ATRIB(5)=BIN(PDR(LINK),31,0);
                        ATRIB(6)=PASADD(LINK);
                        CALL FILEM(1);
                        SUBSTR(STR7,10,1)='1'B;
                        IF PCODE(LINK)=2 THEN DO;
                             IF SUBSTR(LOC(I),7,3)='101'B THEN DO;
/*****************************************************/
/*                                                   */
/*  SCHEDULE C2 RESET AND SET  (PULSE MODE).         */
/*                                                   */
/*****************************************************/
                                  IF CYCLES>0 THEN ATRIB(1)=TNOW+
                                       0.000001*(CYCLES+1);
                                  ELSE ATRIB(1)=TNOW+0.000005;
                                  ATRIB(2)=300;
                                  ATRIB(3)=PASSOC(LINK);
                                  IF CTYPE(LINK)='1'B THEN
                                       ATRIB(4)=3;
                                  ELSE ATRIB(4)=5;
                                  ATRIB(6)=PASADD(LINK);
                                  CALL FILEM(1);
                                  ATRIB(1)=ATRIB(1)+0.000001;
                                  ATRIB(4)=ATRIB(4)-1;
                                  CALL FILEM(1);
                                  ATRIB(3)=MPU;
                                  ATRIB(4)=4;
                                  ATRIB(6)=ADDRESS;
                                  CALL FILEM(1);
                                  C2(LINK)='0'B;
                                  END;
                             IF SUBSTR(LOC(I),7,3)='100'B THEN DO;
/*****************************************************/
/*                                                   */
/*  HANDSHAKE MODE ON B-SIDE...SCHEDULE C2 RESET.    */
/*                                                   */
/*****************************************************/
                                  IF CYCLES>0 THEN ATRIB(1)=
                                       TNOW+0.000001*(CYCLES+1);
                                  ELSE ATRIB(1)=TNOW+0.000001*4;
                                  ATRIB(2)=300;
                                  ATRIB(3)=PASSOC(LINK);
                                  IF CTYPE(LINK)='1'B THEN
                                       ATRIB(4)=3;
                                  ELSE ATRIB(4)=5;
```

```
                                        ATRIB(6)=PASADD(LINK);
                                        CALL FILEM(1);
                                        ATRIB(3)=MPU;
                                        ATRIB(4)=5;
                                        ATRIB(6)=ADDRESS;
                                        CALL FILEM(1);
                                        END;
                                   END;
                              END;
                         ELSE DDR(LINK)=BIT(DATA,8);
/*******************************************************/
/*                                                     */
/*  DATA DIRECTION REGISTER ACESSED.                   */
/*                                                     */
/*******************************************************/
                         END;
                    ELSE DO;
/*******************************************************/
/*                                                     */
/*  ACIA FIRST LOCATION ACCESSED.                      */
/*                                                     */
/*******************************************************/
                         IF PCODE(LINK)\=3 THEN SUBSTR(STR8,6,1)='1'B;
/*******************************************************/
/*                                                     */
/*  BAD PERIPHERAL LOCATION ACCESSED.                  */
/*                                                     */
/*******************************************************/
                         ELSE DO;
                              PDR(LINK)=BIT(DATA,8);
                              IF SUBSTR(PDR(LINK),7,2)='00'B THEN DO;
/*******************************************************/
/*                                                     */
/*  MASTER RESET.                                      */
/*                                                     */
/*******************************************************/
                                   PDR(LINK)='0'B;
                                   DDR(LINK)='0'B;
                                   SUBSTR(LOC(I),13,4)='0000'B;
                                   I=MFIND(ADDRESS,1);
                                   SUBSTR(LOC(I),13,4)='0000'B;
                                   CTYPE(LINK)='0'B;
                                   END;
                              END;
                         END;
                    END;
               END;
          SYSTBL(8,MPU)=BIN(STR8,31,0);
```

```
     SYSTBL(7,MPU)=BIN(STR7,31,0);
     END;
RETURN;
END WRITE;
```

APPENDIX 4: OUTPUT EXAMPLES.

The following is an example of the printouts produced by the simulator either by default or by user specification. Not all of the outputs produced by the GASP-PL/I programs are included because all of these are reproduced in reference 12.

INITIAL SYSTEM CONFIGURATION
****

MPU NUMBER	0	1	2	3
MEMORY SIZE	4096	4096	4096	4096
PERIPHERALS	2	2	2	2
TRACE TYPE	ADDRESS	ADDRESS	ADDRESS	ADDRESS
TRACE START	0998	0998	0998	0998
TRACE STOP	099A	099A	099A	099A
TRACE FILE	3	4	5	6

SIMULATION CONTROL SPECIFICATIONS:

0	10000000000000000000000000000000
1	00000000000000000000000000000000
2	00000000000000000000000000000000
3	00000000000000000000000000000000

PERIPHERAL CONFIGURATION
****

PIA UNITS

ADDR	ASSOC	ASADD	SIDE	CR	DDR	PDR	C1	C2	CTYPE	MPU
0800	100	0001	A	11000000	00000000	00000000	0	0	1	0
0802	100	0000	B	11000000	00000000	00000000	0	0	1	0
0800	101	0001	A	11000000	00000000	00000000	0	0	1	1
0802	101	0000	B	11000000	00000000	00000000	0	0	1	1
0800	102	0001	A	11000000	00000000	00000000	0	0	1	2
0802	102	0000	B	11000000	00000000	00000000	0	0	1	2
0800	103	0001	A	11000000	00000000	00000000	0	0	1	3
0802	103	0000	B	11000000	00000000	00000000	0	0	1	3

ACIA UNITS

ADDR	ASSOC	ASADD	CR	SR	TDR	RDR	MPU
NONE							

Figure A4.1: Initialization Echo Check of System Configuration.

MEMORY DUMP FOR MPU 0

```
0800--PPPPPPPP0000010000000000000000008200000850000000000000010000000
0820--08000A00010B00020C00030D00040A60050B60060C60070D60080E00090F000A
0840--0E600B0F60000000000000000000000801010201030 1FF0107010401050 1FF
0860--01090 10B010A01FF0100000000000000000000000000000000000000000000
0880--00000000000000000000000000000000000000000000000000000000000000
08A0--00000000000000000000000000000000000000000000000000000000000000
08C0--00000000000000000000000000000000000000000000000000000000000000
08E0--00000000000000000000000000000000000000000000000000000000000000
0900--B60801858027 06B60800B7081B3B0000000000000000000000000000000000
0920--8E08FF86FFB708028625B70801862CB70803B60800B60802CE0900FFFFF88605
0940--B708020EB6081ABD09E38601A501274A7E09849600B7081ABD09E3FF08169601
0960--E600C1FF274FBD09D6E7017F00014FBD09D6A6019700FE08168601A50127D486
0980--02B708028620B70705B6081ABD09EFA6002C03BD2000EE01AD008690B70705B6
09A0--0704858026F98604B708023EB6081B850926A020F6BD09D6A600A7016C03A603
09C0--A10226A76F037C0001960291012E037F0001960 12099CE00034A2D0608080808
09E0--20F739FE0814084A2D04080820F939FE0810084A2D0508080820F83900000000
0A00--FE08064C0926FC3900000000000000000000000000000000000000000000000
0A20--00000000000000000000000000000000000000000000000000000000000000
0A40--00000000000000000000000000000000000000000000000000000000000000
0A60--FE08064C0926FC3900000000000000000000000000000000000000000000000
0A80--00000000000000000000000000000000000000000000000000000000000000
0AA0--00000000000000000000000000000000000000000000000000000000000000
0AC0--00000000000000000000000000000000000000000000000000000000000000
0AE0--00000000000000000000000000000000000000000000000000000000000000
0B00--FE08064C0926FC3900000000000000000000000000000000000000000000000
0B20--00000000000000000000000000000000000000000000000000000000000000
0B40--00000000000000000000000000000000000000000000000000000000000000
0B60--FE08064C0926FC3900000000000000000000000000000000000000000000000
0B80--00000000000000000000000000000000000000000000000000000000000000
0BA0--00000000000000000000000000000000000000000000000000000000000000
0BC0--00000000000000000000000000000000000000000000000000000000000000
0BE0--00000000000000000000000000000000000000000000000000000000000000
0C00--FE08064C0926FC3900000000000000000000000000000000D0000000000000
0C20--00000000000000000000000000000000000000000000000000000000000000
0C40--00000000000000000000000000000000000000000000000000000000000000
0C60--FE08064C0926FC3900000000000000000000000000000000000000000000000
0C80--00000000000000000000000000000000000000000000000000000000000000
0CA0--00000000000000000000000000000000000000000000000000000000000000
0CC0--00000000000000000000000000000000000000000000000000000000000000
0CE0--00000000000000000000000000000000000000000000000000000000000000
```

Figure A4.1: Mermory Dump Example.

MPU: 0; TRACE PRINT AT 0.000631 SEC.

TIME	PC	INST	OP1	OP2	REGISTERS AFTER EXECUTION PC	SP	IX	ACCA	ACCB	XXHINZVC
0.000007	0920	8E	08	FF	0923	08FF	0000	00	00	11010000
******* RESET										
0.000009	0923	86	FF		0925	08FF	0000	FF	00	11011000
0.000014	0925	B7	08	02	0928	08FF	0000	FF	00	11011000
0.000015	0928	86	25		092A	08FF	0000	25	00	11010000
0.000020	092A	B7	08	01	092D	08FF	0000	25	00	11010000
0.000022	092D	86	2C		092F	08FF	0000	2C	00	11010000
0.000027	092F	B7	08	03	0932	08FF	0000	2C	00	11010000
0.000031	0932	B6	08	00	0935	08FF	0000	00	00	11010100
******* DATA INPUT										
0.000035	0935	B6	08	02	0938	08FF	0000	FF	00	11011000
******* DATA INPUT										
0.000037	0938	CE	09	00	093B	08FF	0900	FF	00	11010000
0.000043	093B	FF	FF	F8	093E	08FF	0900	FF	00	11010000
0.000045	093E	86	05		0940	08FF	0900	05	00	11010000
0.000050	0940	B7	08	02	0943	08FF	0900	05	00	11010000
******* DATA OUTPUT										
0.000052	0943	0E			0944	08FF	0900	05	00	11000000
0.000055	0944	B6	08	1A	0947	08FF	0900	00	00	11000100
0.000064	0947	BD	09	E3	09E3	08FD	0900	00	00	11000100
0.000069	09E3	FE	08	14	09E6	08FD	0850	00	00	11000000
0.000073	09E6	08			09E7	08FD	0851	00	00	11000000
0.000075	09E7	4A			09E8	08FD	0851	FF	00	11001000
0.000078	09E8	2D	04		09EE	08FD	0851	FF	00	11001000
******* PROGRAM BRANCH										
0.000083	09EE	39			094A	08FF	0851	FF	00	11001000
******* RETURN-FROM-SUBROUTINE										
0.000085	094A	86	01		094C	08FF	0851	01	00	11000000
0.000090	094C	A5	01		094E	08FF	0851	01	00	11000000
0.000094	094E	27	4A		0950	08FF	0851	01	00	11000000
0.000097	0950	7E	09	84	0984	08FF	0851	01	00	11000000
0.000098	0984	86	20		0986	08FF	0851	20	00	11000000
0.000103	0986	B7	07	05	0989	08FF	0851	20	00	11000000
0.000107	0989	B6	08	1A	098C	08FF	0851	00	00	11000100
0.000116	098C	BD	09	EF	09EF	08FD	0851	00	00	11000100
0.000120	09EF	FE	08	10	09F2	08FD	0820	00	00	11000000
0.000124	09F2	08			09F3	08FD	0821	00	00	11000000
0.000126	09F3	4A			09F4	08FD	0821	FF	00	11001000
0.000130	09F4	2D	05		09FB	08FD	0821	FF	00	11001000
******* PROGRAM BRANCH										
0.000135	09FB	39			098F	08FF	0821	FF	00	11001000
******* RETURN-FROM-SUBROUTINE										

Figure A4.3: Trace Print Example.

**** SIMULATION RESULTS ****

```
FINAL STATUS REPORT---- MPU NUMBER  0:
PC= 0A03  SP= 08FD  INR= 00FB  ACCA= 0005  ACCB= 0001  CCR=11000000

FINAL STATUS REPORT---- MPU NUMBER  1:
PC= 0000  SP= 0000  INR= 0000  ACCA= 0000  ACCB= 0000  CCR=11000000

FINAL STATUS REPORT---- MPU NUMBER  2:
PC= 0000  SP= 0000  INR= 0000  ACCA= 0000  ACCB= 0000  CCR=11000000

FINAL STATUS REPORT---- MPU NUMBER  3:
PC= 0000  SP= 0000  INR= 0000  ACCA= 0000  ACCB= 0000  CCR=11000000
```

PERIPHERAL CONFIGURATION
****

PIA UNITS

ADDR	ASSOC	ASADD	SIDE	CR	DDR	PDR	C1	C2	CTYPE	MPU
0800	100	0001	A	00100101	00000000	00001001	1	0	1	0
0802	100	0000	B	10101100	11111111	00000010	1	1	1	0
0800	101	0001	A	11000000	00000000	00000000	0	0	1	1
0802	101	0000	B	11000000	00000000	00000000	0	0	1	1
0800	102	0001	A	11000000	00000000	00000000	0	0	1	2
0802	102	0000	B	11000000	00000000	00000000	0	0	1	2
0800	103	0001	A	11000000	00000000	00000000	0	0	1	3
0802	103	0000	B	11000000	00000000	00000000	0	0	1	3

ACIA UNITS

ADDR	ASSOC	ASADD	CR	SR	TDR	RDR	MPU
NONE							

Figure A4.4: Final Summary Prints.

SIMULATION STATISTICS

MPU	INST	INPT	OTPT	ERRS	WARN	BRCH	INTS	RTRN
0	888	3	3	0	1	269	1	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

MPU	RESET	RUNNING	HALTED	LAST RESTART	LAST HALT
0	1	2.93140E-03	6.86047E-05	2.70761E-03	3.00000E-03
1	0	0.00000E+00	3.00000E-03	0.00000E+00	0.00000E+00
2	0	0.00000E+00	3.00000E-03	0.00000E+00	0.00000E+00
3	0	0.00000E+00	3.00000E-03	0.00000E+00	0.00000E+00

BIM	BIM I/O OUTPUTS	INPUTS
0	15	0
1	0	15
2	0	15
3	0	15

Figure A4.4: Final Summary Prints (continued).

GASP SUMMARY REPORT

SIMULATION PROJECT NUMBER 40 BY G. A. BRENT
DATE: 09/07/78 11:46:58 RUN NUMBER 1 OF 1
CURRENT TIME = 3.0000E-03

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN	STD.DEV.	S.D. OF MEAN	COEF. VAR.	MIN.	MAX.	OBS.
ITIME-0	3.4561E+00	1.1526E+00	3.8677E-02	3.3349E-01	2.0000E+00	1.0000E+01	888
ITIME-1	NO VALUES RECORDED						
ITIME-2	NO VALUES RECORDED						
ITIME-3	NO VALUES RECORDED						
BUS-0	3.0047E-05	0.0000E+00	0.0000E+00	0.0000E+00	3.0047E-05	3.0047E-05	1
BUS-1	NO VALUES RECORDED						
BUS-2	NO VALUES RECORDED						
BUS-3	NO VALUES RECORDED						
TOSWT-0	7.5234E-05	0.0000E+00	0.0000E+00	0.0000E+00	7.5234E-05	7.5234E-05	1
TOSWT-1	NO VALUES RECORDED						
TOSWT-2	NO VALUES RECORDED						
TOSWT-3	NO VALUES RECORDED						
BUSWT-S	3.0047E-05	0.0000E+00	0.0000E+00	0.0000E+00	3.0047E-05	3.0047E-05	1
TOSWT-S	7.5234E-05	0.0000E+00	0.0000E+00	0.0000E+00	7.5234E-05	7.5234E-05	1

Figure A4.5: GASP-PL/I Final Summary Report.

Vita

Graduate School
Southern Illinois University

Name Glen Alan Brent Date of Birth [redacted]

Address [redacted]

[redacted]

Education

Bachelor of Science in Engineering, Department of Electrical Sciences and Systems Engineering, Southern Illinois University, September 1973 to May 1977.

Master of Science in Engineering, Department of Electrical Sciences and Systems Engineering, Southern Illinois University, June 1977 to September 1978.

Honors

Tau Beta Pi

Thesis

"GASP-PL/I Simulation of Integrated Avionic System Processor Architectures"
Advisor: Dr. Thomas M. McCalla, Jr.

Publications

Glen A. Brent and Thomas M. McCalla, "Exploring Team Avionics Systems by Simulation," Eleventh Annual Simulation Symposium Tampa, Florida, March 1978.