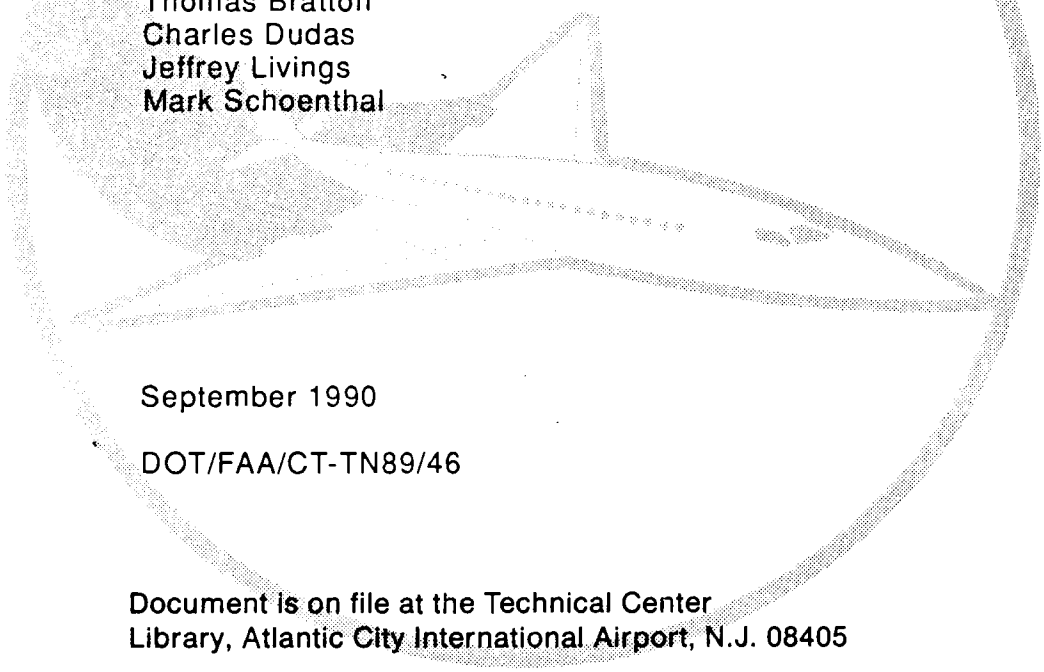
Communications Interface Driver (CID) Software Principles of Operation

Thomas Bratton
Charles Dudas
Jeffrey Livings
Mark Schoenthal



September 1990

DOT/FAA/CT-TN89/46

U.S. Department of Transportation
Federal Aviation Administration

Technical Center
Atlantic City International Airport, N.J. 08405

technical note technical note

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/CT-TN89/46		

4. Title and Subtitle	5. Report Date
COMMUNICATIONS INTERFACE DRIVER (CID) SOFTWARE PRINCIPLES OF OPERATION	September 1990
	6. Performing Organization Code
7. Author(s) Thomas Bratton, Charles Dudas, Jeffrey Livings, Mark Schoenthal	8. Performing Organization Report No. DOT/FAA/CT-TN89/46
9. Performing Organization Name and Address Federal Aviation Administration Technical Center Atlantic City International Airport, New Jersey 08405	10. Work Unit No. (TRAIS)
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Technical Center Atlantic City International Airport, New Jersey 08405	13. Type of Report and Period Covered Technical Note
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The Communications Interface Driver (CID) makes possible the performance assessment of the Mode Select (Mode S) sensor under it's specified maximum communication load. To accomplish this, CID uses prepared communication scenarios structured on air traffic models to generate the simulated communication messages for Mode S. The physical connection to the Mode S sensor is through X.25 Link Access Procedure Balanced (LAPB) communication lines.

This document, the CID Software Principles of Operation, describes the CID software and theory of operations. Each software module is discussed. Data flow diagrams, program flow charts, and timing charts are included where appropriate.

17. Key Words	18. Distribution Statement
CID Mode S Software, CID	Document is on file at the Technical Center Library, Atlantic City International Airport, NJ 08405

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	138	

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

EXECUTIVE SUMMARY

The Communications Interference Driver (CID) is a capacity test device for the production Mode Select (Mode S) beacon system. The design and development of the CID was the responsibility of ACN-220 Federal Aviation Administration (FAA) Technical Center personnel. The FAA will make this equipment available as government-furnished equipment to the Mode S contractor.

This document, along with the actual commented code, will be used to maintain both the real-time and off-line software of the CID. This document contains sections on both the real-time CID software and the CID initialization (CIDINIT) programs. For the real-time CID software, this document provides an overall description of the software's functions and subroutines and an explanation on how this software is integrated into the host computer's operating system, along with separate descriptions and flowcharts for each subroutine within the real-time software. This document provides a description and flowchart set for each of the CID's programs.

1. INTRODUCTION.

The Communications Interference Driver (CID) system is designed to simulate the X.25 communications environment for up to 700 aircraft for the purpose of testing the operation of Mode Select (Mode S) beacon interrogators under heavy communication load. The CID equipment consists of time-of-year (TOY) clock circuitry, twelve X.25 communication interfaces, a computer system, and associated computer peripheral equipment. Specific circuitry, as described above, was designed and developed by Secondary Surveillance Systems Branch, ACN-220, personnel. The computer system implemented is a Concurrent Computer Corporation model 3230XP with a 1600-baud tape device, 300-megabyte disc drive, and line printer. The X.25 interfaces were built by Macrolink and utilized the Western Digital WD-2511 X.25 protocol control chip as the communication line interface. The Macrolink interface also contains logic for the COMMUX and DMA interface to the Concurrent Computer System. Software, as described in section 6 of this report, was designed and developed by ACN-220 personnel. Figure 1-1 depicts a functional block diagram of the CID hardware.

The CID system was designed to work in conjunction with the Aircraft Reply and Interference Environmental Simulator (ARIES). These two simulation test systems will provide an environment of 700 aircraft with communication for the testing of the Mode S sensor. Figure 1-2 depicts the simulation block diagram for the CID, ARIES, and Mode S systems. The CID may be utilized in the live world environment for the testing of the Mode S sensor. Figure 1-3 depicts the live world block diagram for the CID and Mode S sensor. CID real-time software was designed in two phases. Phase 2 software includes the capability to utilize the CID with two Mode S sensors for the purpose of multisite testing. Figure 1-4 depicts the live world block diagram for the CID and two Mode S sensors.

2. OBJECTIVE.

The objective of this report is to describe the CID software. These include functional descriptions of each program component of the CID system. The functional descriptions are composed of discussions of all modules used to develop each program component of CID. Program flow charts and data structures have been included as figures in this report. This report will provide the top two levels of documentation for CID software. The third level documentation is provided by detailed comments that are included in the actual CID software.

3. BACKGROUND.

For the past decade, the Federal Aviation Administration (FAA) has been committed to the development of a new sophisticated aircraft surveillance system for air traffic control (ATC). The new system, Mode S, provides increased capacity, better azimuth measurement, and reduced interference. In addition, aircraft equipped with Mode S transponders provide ground-to-air and air-to-ground data link capabilities.

Due to the large target capacity of a Mode S sensor, it is not possible to find a current air traffic environment that is dense enough to fully test the sensor under heavy load conditions. It is also desirable to be able to repeat an identical test situation several times. For these reasons, a communications simulator has been built to provide a repeatable dense communication environment typical of what may be encountered in the future. CID is intended to provide the communications environment for the purpose of factory and field acceptance testing in the capacity situation and to provide a capacity load for the Mode S sensor. CID is not designed to be able to test all the system features of a Mode S sensor. Capabilities which are costly to implement and have little effect on sensor's behavior under load have not been implemented.

4. RELATED DOCUMENTS.

FAA-ER-2716, Specification for a Mode Select Beacon System (Mode S) sensor, March 24, 1983.

FAA Order 6365.1A, United States National Standard for the Mode Select Beacon System, January 3, 1983.

FAA-RD-80-14A, The Mode Select (Mode S) Surveillance and Communications, ATC and Non-ATC Link Protocols, and Message Formats, November 26, 1985.

5. CID SOFTWARE DESCRIPTION.

The CID consists of various software functions to perform the generation and simulation of X.25 communication functions for the Mode S system. The CID is designed to simulate the communication loading for up to 700 transponder-equipped aircraft. The software functions include:

a. Scenario Generation
b. CID Initialization
c. CID Simulation

Scenario Generation is the process in which communication messages are prepared off-line for the CID simulation program. Communication scenarios are prepared by a program function that generates scenario files that may be stored on disc or tape for future simulation. The CID initialization (CIDINIT) program creates and maintains files that contain the configuration data for the real-time simulation program. The CID program performs the real-time simulation of the X.25 communication messages. The interaction of these software modules is depicted in figure 5-1.

The process of creating a communication scenario begins off-line with the creation of the definition file. This definition is a part of the target definition for the ARIES system. Targets are defined by time, position, and transponder events. These definitions resemble flight plans but also include transponder events. Communication functions are added to the target definitions. These data are stored in a waypoint definition file. A configuration contains the definitions for operating parameters of the scenario generator. These two file sets are used as

input for the target scenario program. The output is a time ordered set of communication messages in raw binary format that the CID program can use as input during the real-time simulation. This program is described in the Scenario Generator User's Manual for the Aircraft Reply and Interference Simulator and the Communication Interface Driver System, DOT/FAA/CT-TN88/43.

5.1 CID REAL-TIME TARGET SIMULATION.

The CID software is designed around a highly efficient real-time simulation program. Certain functions, such as the communications scenario generation and CIDINIT, are performed prior to the actual simulation. Once a scenario is produced in these steps prior to the actual simulation, it may be used as many times as desired in an actual real-time simulation. The CID program has been designed to maximize processing efficiency and peripheral throughput. Each module was time and load tested to insure the proper operation of the real-time simulation system.

The CID real-time program is designed to operate X.25 communication lines. Messages generated by the Mode S Sensor are received by the CID and extracted on magnetic tape. The scenario contains a time ordered sequence of messages. These messages are transmitted on the X.25 lines at the specified time. Each X.25 line in the CID may be configured to operate at either 9600 baud or 56k baud. The definition for operation of each port is contained in the CID initialization file. This file also contains a set of default messages used in the real-time simulation.

The design of a scheduling function to permit efficient task switching time was also necessary. The scheduling function allows not only periodic scheduling of a module, but also the intermittent scheduling of a module. System input and output (I/O) had to be distributed over time in order to prevent the lockout of other processing modules. The I/O Manager was designed to distribute system input and output according to a priority scheme. The extraction function was designed to permit extraction of data from any processing module including the interrogation processing routines. It is semaphore protected and minimizes the amount of data relocation in order to maximize the extraction processing efficiency.

The CID program consists of the following major software modules:

5.1.2 Initialization (Init)
5.1.3 Task Scheduling (Scheduler)
5.1.4 Input Message Processing (INMESS)
5.1.5 Output Message Processing (OUTMESS)
5.1.6 Second processing (Sec)
5.1.7 Operation Communication (Op-Comm and Op-Comin)
5.1.8 I/O Management (I/O)
5.1.9 Data Extraction (Extraction)
5.1.10 Interupt Service routines (ISR)

These modules interact through the global common data structures as shown in figure 5.1-1. The operation of each module is summarized here and explained in detail in the aforementioned chapters of this report. The use of each common data structure is also summarized here. Note that Init and Extraction interact with all processing modules and data commons.

Init (5.1.2) sets up the logical devices, loads the global common data structures, positions the model input files, and resets the CID hardware for the simulation. It can also be used to initiate the data extraction process. Execution of this module occurs only once during each real-time simulation.

Scheduler (5.1.3) controls the order in which the processing modules in the CID task are executed. All processing modules are called by the Scheduler except Init and Extraction. Scheduler is accomplished by this module because of the severe overhead incurred in the use of operating system resources for this function.

All incoming X.25 messages are processed by the Input Message Processing routine (5.1.4). This routine interfaces with the extraction routine for data extraction of messages, the operator's communications (Op-Comm) routine for the display of selected target messages, and the network management processing subroutines for the interactive response of network management messages.

Message data is obtained from the model files stored on the disc. These data are processed by the Output Message processing modules (5.1.5). The message model file is time ordered. Messages created through the Op-Comm terminal are also processed by the Output Message Processing module.

Second processing (5.1.6) performs various regular functions required in the CID real-time system.

Op-Comm (5.1.7) provides the system operator with the capability to control the operation of the simulation. These functions include checking system status, checking system errors, creating or displaying X.25 messages, operating the data extraction module, and terminating the simulation.

I/O Management (5.1.8) provides an orderly control of the I/O for the model input files, extraction device, and the Op-Comm control terminals. I/O requests are processed using a priority algorithm that attempts to maintain a queued I/O for each file.

Extraction (5.1.9) processes all requests for data extraction. Extraction packs messages in output buffers and sets the output request flags for I/O. Extraction operates as a subroutine that can be called by any processing task. A second extraction subroutine with protecting semaphores is implemented for interrogation processing.

X.25 Communication Routines (5.1.10) performs all the functions associated with the processing of interrupts and data from the X.25 devices and the TOY device. Interrupts are generated by the X.25 devices for all incoming messages, outgoing messages, and error conditions. Interrupts are generated by the TOY device every 0.001024 seconds. The TOY interrupt provides the basic timing element for the CID real-time software.

The X.25 message scenario file is stored on the magnetic disc device. These files contain the X.25 message descriptions in a raw binary format to allow the most efficient real-time processing. These files are each built off-line by the scenario generation process. Many scenario files may exist on the disc. The operator selects the message scenario file during initialization for the real-time CID simulation. All operator functions and statistics are controlled and presented through the Op-Comm terminals.

The initialization parameters for an execution of the CID real-time software are contained within a file stored on the magnetic disc device. These files are built off-line by the CIDINIT program. Many initialization files may exist on the disc. The operator selects which initialization file is required at the start of the CID real-time program.

For each external file, there exists a buffered common area for the I/O data. The X.25 message scenario file data is buffered in the message input buffer. The Op-Comm terminal data, both input and output, are buffered in the Op-Comm I/O buffers. These buffers are maintained and processed by I/O.

5.1.1 OS/32 Integration.

The CID program operates on a Perkin-Elmer (PE) 3230 processor with the OS/32 operating system. The OS/32 is a versatile, multitasking, real-time operating system operates on the PE 3200 computer system family. It provides a responsive environment for the CID application. The operating environment is described in the following paragraphs.

The CID system requires initialization of OS/32 system functions before the ARIES program can execute. These functions are built into a command substitution system (CSS) that perform operations executed by a single command. By typing a single command, "CID," the user can execute all the steps required for OS/32 initialization and execute the actual CID program. The following steps are required for OS/32 initialization:

a. Terminate all OS/32 tasks.
b. Initialize the task common areas.
c. Load the writable control store with the Micro-Code Program.
d. Load the interrupt service routines in a task common area.
e. Install pointers in the Interrupt Service Pointer Table of OS/32.
f. Install interrupt branches in OS/32.
g. Assign I/O devices.
h. Load and start the CID task.

The termination of OS/32 task is accomplished by a program called Killtask. Killtask is an OS/32 executive task that removes the task control blocks for all tasks running under OS/32. This will remove all tasks from the system regardless of their current states. The initialization of the task common areas is performed by a program called Inittcom. It is an OS/32 executive task that stores a predetermined pattern in the fixed task common area.

The Writable Control Store (WCS) is loaded by an OS/32 utility. The microcode loaded in the WCS contains the function of TOY Interrupt Processing. The Interrupt Service Pointer table is modified to handle interrupts from the TOY hardware and the X.25 hardware. The entries in the Interrupt Service Pointer table are addresses of the Interrupt Service Routines (ISR's) for each Interrupt Process. Since the table entry is 16 bits, the address of each ISR must be in the bottom 64 kilobytes of memory. The major ISR's are located either in WCS or the Fixed Task Common. Therefore, the Interrupt Service Pointer table will point to a set of branch instruction that transfer the processor to either the WCS ISR or the Fixed Task Common ISR.

The final function of the CID CSS is to load the CID program, initialize devices, and start the execution of the CID Task. Once the CID Task is in progress, all commands are performed by Op-Comm. Should an OS/32 function be requested through the system console, a fault will be generated and undesirable results will be generated. Once the CID has completed its function, the user may stop the simulation process through an Op-Comm function provided to terminate the CID Task. Control of the system now reverts to the OS/32 System Console.

Two versions of the CID software were designed. The first phase is the generic CID real-time software as described. The second phase consists of all functions incorporated in phase I and additional software to provide for the connection of two Mode S sensors to a single CID system. The software for the second phase has been modified to incorporate network management functions to permit messages entering the CID from the first sensor to be directed to the second sensor. Selected messages with the sending sensor identification fields set for the second Mode S sensor will be retransmitted by the CID to the second Mode S sensor. Phase II software may be executed by typing a single command "CID2." The user can execute all the steps required for OS/32 initialization and execute the actual CID program with network management functions.

5.1.2 Init.

Init is the task that insures that all the required parts of CID are ready before beginning the real-time processes. A flowchart of Init is shown in figure 5.1.2-1. The following reflects the task operated by Init:

a. First the required disk files are obtained by Logical Unit Init.

b. The data extraction buffers are set up in Extraction Init.

c. The Scheduler is set up in Scheduler Init.

d. The CID system commons are initialized in Task Common Init.

e. The scenario buffers are initially filled in Scenario Buffer Init.

f. The system status display is then cleared followed by Op-Comm Init which initializes the Op-Comm functions.

The configuration for operating the CID program is generated by the CIDINIT program as described in section 5.2 of this report. The configuration file generated by CIDINIT contains all parameters for the operation of the CID real-time program. After the operator supplies the configuration definition filename, the operator will be told to switch the system console over to the control terminal. Init will wait until the switch has been turned, clears the control terminal's screen, and then set up the fault handlers in Task Status Word Initialization. Hardware Init will initialize the special purpose CID hardware. After all the required software and hardware has been initialized, Init will call the Scheduler to begin the real-time portion of the CID program.

5.1.2.1 Logical Unit Init.

Logical Unit Init is responsible for assigning the logical units, as required by the CID program. The logical unit assignments for the hardware devices required by the CID system is accomplished by the CID startup CSS. The following logical unit assignments are made:

a. X.25 Message Scenario File
b. Magnetic Tape Drive (Data Extraction)
c. Statistics Terminal
d. System Console (Default)
e. Menu And Control TerminaL

The files that the CID program require is the X.25 message scenario. A message is sent to the default unit (5-System Console) requesting the specific filename. After the filename is read from the console, the corresponding extension is appended to the filename (.CID for the X.25 message scenario). Finally, account number 2 is appended to complete the filename. The resultant file is then tested to insure that the file exists. The scenario files are also tested to insure that the file is of a contiguous type. After the file is tested, the file is opened to the correct logical unit number. If an error is encountered anywhere through the testing or open process, an error message will be sent to the system console and the CID program stopped.

5.1.2.2 Extraction Init.

Extraction Init will prompt the user for a YES/NO answer to the query ENABLE EXTRACTION. When the user answers with a Y (or YES), Extraction Init will send a rewind command to the magnetic tape unit (logical unit 2). The status returned from the rewind command will be tested. A nonzero status may indicate a tape drive error, but probably indicates that the tape is either off-line or not loaded. Thus, when a nonzero status is encountered, Extraction Init is restarted allowing the user to correct the problem and reanswer the enable extraction query. After a successful rewind is completed, the extraction buffer pointers will be initialized and the system flag ST EXT FLAG will be set indicating that extraction is enabled.

5.1.2.3 Scheduler Init.

The Scheduler is made up of eight separate subroutines: I/O, Op-Comm, Op-Comin, INMESS, OUTMESS, SECOND, and two dummy subroutines to leave room for expansion. SECOND is initialized to first run at 1 second into the simulation and each second thereafter. Op-Comm and Op-Comin are popup tasks where Op-Comin is popped up by I/O and Op-Comm is popped up by Op-Comin. The remaining tasks are initialized to values determined by testing and, as such, are subject to change.

5.1.2.4 Task Common Init.

Task Common Init is responsible for initializing all the required task common areas prior to system operation. Common areas are utilized for storage of data between the main real-time program, task subroutines, and the interrupt service routines. Two types of task commons are utilized. Some of these commons use storage within the task block. These commons are referenced only by the main real-time program and task subroutines. Other commons are fixed in memory location. These commons

are utilized by all routines within the system including the interrupt service routines. The accessibility of all commons from the main real-time program and subroutines is through the memory address controller of the 3230XP computer system and transparent to the program.

5.1.2.5 Scenario Buffer Init.

This subroutine is responsible for loading the X.25 message scenario buffers with valid data, and, then if a start time other than zero is requested from the user, to update the input buffers, and their associated pointers, to the appropriate scenario data. A flowchart for scenario buffer Init is given in figure 5.1.2.5-1.

First, the buffer pointers will be set to point to the first update of the first buffer. After the pointers have been set, the subroutine will fill all the buffers with scenario data from the disk. The subroutine uses the routine I/O Init to fill the buffers.

Once the buffers have been filled the operator will be given the opportunity to alter the simulation start time from the default of zero. The time is input into the variables START MIN and START SEC. If either variable is found to be a positive number, then a nondefault start time will be processed; otherwise, the start time will be zero and the subroutine will be exited.

When a nondefault start time has been read from the operator it will be received in minutes and seconds. First, the time will be converted to seconds and the 1's digit will be made 0, leaving the start time in tens of seconds. This is done because the interval between fruit scenario updates is 10 seconds. Since the fruit updates are not time tagged, the easiest way to insure that the target and fruit scenario remain synced in time is to start right at a 10-second boundary. After the time has been put into tens of seconds, the time will be converted into system timing units (STU) where 1STU = 1.024 seconds.

The X.25 message buffers will be updated with the scenario data corresponding to the new start time. The subroutine will read the time tag of the first update in each buffer (NEXT BUFFER) until a buffer is found with a time tag greater than the new start time. When a buffer has been found not to have a time tag greater than the new start time, the preceding buffer will be filled with new scenario data from the disk and the NEXT BUFFER pointer and the buffer number will be incremented and the next buffer tested. When a buffer has been found to contain a time tag greater than the new start time, then the preceding buffer will be read to find the initial update to be used in the simulation. The preceding buffer will be read until an update is found whose time tag is once again greater than the new start time. If such an update is found, then that update will be considered the initial update and the appropriate pointers will be set. If the buffer is completely read and no update is found with a time tag greater than the new start time, then the first update will be considered the initial update and the appropriate pointers will be set to point to it.

When an update is found whose time tag is less than zero (which indicates the end of the scenario), then the previous buffer will be searched. If no update is found with a time tag greater than the new start time, then the new start time is greater than the overall length of the scenario, and an error message will be output to the operator and the simulation cancelled.

After the scenarios have been positioned, the Scheduler will need to be updated to reflect the new start time. The Scheduler's time to execute fields must be reset in accordance with the new start time in STU's.

5.1.2.6 Op-Comm Init.

Op-Comm Init initializes the variables used to calculate the percent data extraction and Central Processing Unit (CPU). These variables are initialized here since it is an Op-Comm function to update these percentages. Next, Op-Comm is placed into the Scheduler where the first time to run is set to 1 second into the simulation and the frequency of operation is every 3 seconds. Op-Comm then sets the function flag to function 1 and queues the appropriate buffers to output the function menu on terminal 1 and the statistics display on terminal 2.

5.1.2.7 Hardware Init.

Hardware Init is responsible for insuring that all the special purpose CID hardware is prepared for the start of the simulation.

After the special purpose ARIES hardware devices are initialized, Initialize Hardware will disarm the system devices not needed by the ARIES program. The system devices that are disarmed are the Program Interval Clock (PIC), and the Line Frequency Clock (LFC).

5.1.2.8 X.25 Hardware Init.

X.25 Hardware Init is responsible for insuring that all the CID X.25 hardware is prepared for the start of the simulation. Each X.25 device required in the simulation will be reset, the DMA interface for the X.25 controller will be initiated, and the registers in the Western Digital WD-2511 X.25 controller will be loaded for normal operation.

5.1.2.9 Task Status Word Init.

Task Status Word (TSW) Init will initialize the fault handlers used during the CID program. During operation of the CID program OS/32 will not be handling the normal processor faults. To keep the CID system from crashing when a processor fault is encountered, the following fault handlers must be added to the CID operational program: Memory Access fault, Illegal Instruction fault, and Data Format fault. When the CID program encounters one of the above faults, a message detailing the fault, along with the address of the fault, will be displayed on the console. After the message is displayed, the ARIES hardware will be disarmed and the system hardware will be rearmed. Data Extraction will extract any remaining buffers of data onto the magnetic tape and then the CID program will be exited.

5.1.3 Scheduler.

The Scheduler is an Assembly Language Program that arbitrates the scheduling request of the CID system. This process contains three tables with one entry per task to be scheduled. These tables contain the location of the first executable instruction of the task, the next time to execute the task, and the time between executions of the task. The scheduling function uses a round robin scheduling algorithm that determines the order in which tasks are examined for their

processing needs. Tasks may be scheduled periodically by storing a positive value in the increment table for that task. Also, tasks may be scheduled once by storing the start execution time in the next time to execute table and a negative value in the time increment table for that task. The Scheduler was designed to accommodate eight potential tasks: I/O, Op-Commin, Op-Comm, INMESS, OUTMESS, SECOND, and two spare tasks.

The Scheduler will traverse the time to execute the table until it detects a time less than the current system time. Once this condition is met, the Scheduler will determine the next time for that task to request processing and store that time value in the time to execute the table. If the current request for task processing is a single shot popup request, the value stored in the time to execute the table is the maximum time value. Before the Scheduler calls the task, the PIC is started to measure the processing time. The task is then allowed to proceed. Once the task has completed, the task Scheduler will strobe the PIC to determine the amount of time used by the task. This time is accumulated and used by the Op-Comm for the presentation of processor loading. The Scheduler will then look at the time to execute the table, starting with the next entry in the table.

The Scheduler was designed in the above manner to enhance the performance of the system. The scheduling of separate tasks under OS/32 requires over 1 millisecond (ms) to switch a task. Performance was greatly enhanced by the table scheduler permitting task switch times of less than 100 ms. Program and data flow charts are presented in figures 5.1.3-1 and 5.1.3-2.

5.1.4 Input Message Processing.

The Input Message Processing (INMESS) task is responsible for inputting the messages as they arrive on the X.25 links and the extraction of these data packets. INMESS was designed to operate at the capacity 700 target communication load as described in the Mode S engineering requirement. The INMESS task was developed in two phases. Phase II (CID2) also includes functions for the display of incoming X.25 messages and network management functions. The phase II functions were designed to be executed in situations less than the capacity loading.

5.1.4.1 INMESS (Overview).

INMESS is written in modular form. The main module is named INMESS and is composed of INMESS.TOP, INMESSX.FTN, and INMESS.END. These groups of file are concatenated to form the module INMESS.FTN. A command substitution system (CSS) file named EXPANDIN.CSS was designed to facilitate the concatenation process. INMESS.TOP includes the common declaration and parameter definition. Next, 12 copies of the INMESSX.FTN file are concatenated. A separate copy INMESSX.FTN for each line must be used because of access to the the memory of the common data structures. These structures must be accessible from both the INMESS task but also from the ISR as described in section 5.1.10. Finally, the INMESS.END module is concatenated. This module contains the terminating code for the INMESS task.

The processing of X.25 lines is accomplished in a linear order. The algorithms for processing each X.25 line are identical for all X.25 lines. Figure 5.1.4.1-1 depicts the flowchart for the INMESS processing. INMESS will first check the next expected message pointer for the X.25 physical device. Then a determination is made as to whether a X.25 message is available. If a message is available, the length must be computed from data available from the X.25 device. Processing

statistics are updated and the message is extracted. At this point in processing, an algorithm for displaying messages in Op-Comm function seven is included in phase II software. Also, at this point an algorithm for network messages in included in phase II software. These algorithms are described in sections 5.1.4.2 and 5.1.4.3. The X.25 buffer is reset so that it may be used for another message, in consecutive order. Eight X.25 input message buffers are included. These buffers are used in rotation as the X.25 messages arrive. If another message is buffered, the entire process is repeated. Otherwise, statistics for the Op-Comm display function are updated. The X.25 frame reject flag is checked. If set, the buffer pointers, control variables, and the X.25 flags are reset. A flowchart of this process is depicted in figure 5.1.4.1-2. The entire process is repeated for each of the twelve X.25 lines in the CID system.

5.1.4.2 Message Display Function.

Phase II software permits the user to display of incoming X.25 messages on the statistics display. The user interface is provided by Op-Comm function 7. A complete explanation of Op-Comm function 7 is contained in section 5.1.7.1.9. If the Op-Comm function 7 active flag is set, INMESS must filter each message for the display function. The identification field of the message is checked to determine if it is one of the following:

a. 31 - Message Rejection/Delay Notice
b. 32 - Uplink Delivery Notice
c. 41 - Standard Downlink
d. 42 - ELM Downlink
e. 44 - Data Link Capacity
f. 45 - ATCRBS ID Code

If the message type is one of the above, the Mode S identification field (ID) will be checked against a list of five potential Mode S IDs. The five Mode S IDs are provided by the user through Op-Comm function 7. If the message is for one of the five selected Mode S IDs, the message is copied to an output buffer, the Mode S ID is stored, the byte count is stored, the type code is stored, the port number is stored, and the storage pointers are updated. The values stored will be used by Op-Comm function 7 for the subsequent display of the selected messages. Section 5.1.7.1.10 describes the operation of the Op-Comm display function. A flowchart for the operation of message filtering for the Op-Comm display function is depicted in figure 5.1.4.2-1.

5.1.4.3 Network Management Function.

Phase II (CID2) permits the retransmission of X.25 messages to support the network management functions of the Mode S system. When executing this program, X.25 messages arriving at the CID system are scanned. Network management messages will be transmitted to the Mode S sensor as defined in the receiving sensor identification (RSID) field. The port number of the CID system must comply with the RSID field for this to occur.

This process begins with a determination if the network management function is enabled. Then the identification field of the message is checked to determine if it is one of the following:

a. 91 - Data Start
b. 92 - Data Stop
c. 93 - Data Request
d. 94 - Track Data
e. 95 - Cancel Request
f. 9D - Primary Coordination
g. 9E - Track Alert
h. D1 - ATCRBS Data Start
i. D2 - ATCRBS Data Stop
j. D3 - ATCRBS Data Request
k. D4 - ATCRBS Track Data
l. D5 - ATCRBS Cancel Request
m. 71 - Status Message
n. 72 - Adjacent Sensor Status Request
o. 73 - Adjacent Sensor Status Response

If the message type is one of the above, the RSID field is determined. The message is composed and placed in the buffer for Output Message (OUTMESS) Processing. A message pointer is placed on a list of pending messages for transmission. These messages will be transmitted by OUTMESS to the sensor as defined by the RSID field. In the event space is not available for the message pointer to be placed on the list, an error is generated. This error will be accounted along with all other CID system errors. A flowchart for the operation of the network management function of INMESS is depicted in figure 5.1.4.3-1.

5.1.5 Output Message Processing.

The OUTMESS Processing task is responsible for handing scenario based messages to the TRANSMIT routine. TRANSMIT will load the messages into the hardware for final delivery. OUTMESS is a single subroutine made up of four blocks; OUTMESS main, GETLINE, GETBUF, and SEND MESSAGE. OUTMESS uses the $INCLUDE statement to include these blocks directly into OUTMESS main. Note that the block SEND MSG is also used by the create a message and message routing functions to send messages.

5.1.5.1 OUTMESS (Main).

OUTMESS starts by setting the extraction buffer index and the number of messages processed counters. OUTMESS builds an extraction buffer as it processes messages. OUTMESS then extracts this buffer just before exiting. A flowchart of OUTMESS main is given in figure 5.1.5.1-1.

The first messages processed by OUTMESS are the OLD messages. These are messages that OUTMESS had previously processed but received a nonfatal error return code from the hardware. OUTMESS then saved the messages as OLD messages. OUTMESS will only process 10 OLD messages for each call. The pointers to these messages are stored in a Fortran circular list. The messages themselves are still stored in the scenario message buffers. OUTMESS retrieves the message pointer by using the Fortran call RTL (read top of list). OUTMESS then places the pointer into the variable MSG POINTER. OUTMESS will check for errors and then set the appropriate

flags. Next OUTMESS will need to insure that the scenario buffer holding the OLD message is still ready and has not been overwritten. This is done by checking the scenario buffer's ready flag. When this flag is found to have been reset, then the buffer has been refilled. The OLD message has then been overwritten and is lost. OUTMESS will increment and extract the error condition.

OUTMESS uses the variable SCEN POINTER to update MSG POINTER when processing scenario messages and not OLD messages.

Now MSG POINTER is pointing to the message that OUTMESS will process. OUTMESS next processes the time field of the current message. A positive time field indicates the time at which the message should be sent. A zero time field indicates the end of the scenario buffer, while a negative time field indicates the end of the scenario.

A time field that is greater than the current system time indicates there are no more scenario messages scheduled to be sent now. OUTMESS will then extract its extraction buffer and return to the CID task Scheduler.

A zero time field will cause OUTMESS to process the block GETBUF. GETBUF will get the pointers to the next scenario buffer. After GETBUF is finished, OUTMESS will extract its extraction buffer and return.

A negative time field will cause OUTMESS to deschedule itself from the CID's task scheduler and extract that event. OUTMESS will also extract its extraction buffer and then return.

A positive time field that is less than or equal to the current system time will cause OUTMESS to process the message. OUTMESS will read the header, port number and byte count of the message, and then enter the block GETLINE. GETLINE will return with the variable LINE set on and the multi-line variable sent accordingly.

Next, OUTMESS will check to see if the message has aged too much to process. This would usually be an OLD message. When the time difference between the message's time field and the current system time is greater than a preset variable, the message is too old. OUTMESS will enter the message pointers and a failure code into the extraction buffer and return to the top of its message processing loop.

OUTMESS will then enter the block SEND MESSAGE. This block will try to send the message on the line number held in the variable LINE. SEND MESSAGE will check return codes from the hardware and on failures to try to resend messages from multi-line ports. When there are no available lines, SEND MESSAGE will try to save the message on the old list.

OUTMESS will next increment the scenario index. OLD messages do not apply here. OUTMESS will increment the scenario index by five words for a scenario message that was a default message. For other scenario messages, the scenario index will be incremented depending on the length of the message. The header field holds the length of the message in bytes. OUTMESS will then return to the top of its message processing loop (label 10). OUTMESS will continue to process messages until it finds a time field that is either greater than the current system time, zero, or negative.

5.1.5.2 GETLINE.

GETLINE is a block of code that is included into the OUTMESS subroutine as in-line code. GETLINE will take the port number of the message and determine its port and line status. A flowchart of GETLINE is given in figure 5.1.5.2-1.

GETLINE first checks that the port number passed to it is a legal port number. This step should not be necessary, but since an illegal port number will cause a system crash, it is done anyway.

Next GETLINE checks for a single-line or multi-line port. Single line ports have their port number converted directly to a line number. Multi-line ports are more complicated.

First, with a multi-line port GETLINE will check to see if the port has been configured. For a port that is not configured, the appropriate error message and counters are incremented and extracted. Program control will return to the top of the process loop in OUTMESS main. Next, GETLINE sets the MULTI flag and the variable ATTEMPTS. OUTMESS uses the variable ATTEMPTS to increment through the lines in a multi-line port when an error is received from the TRANSMIT routine. GETLINE then uses the variable PT PTR, along with the PORT number, to index in the array MLINE to get the line number.

5.1.5.3 GETBUF.

GETBUF is a block of code that is included into the OUTMESS subroutine as in-line code. GETBUF will get the pointers to the next scenario buffer and check for error conditions. A flowchart of GETBUF is given in figure 5.1.5.3-1.

The number of scenario buffers is set by the variable SCEN BUFS. I/O Processing fills these buffers by following the variable OBUF. OUTMESS uses the scenario ready and full flags to determine the scenario buffer status. When incrementing the scenario buffer index, GETBUF initially uses the variable NEWBUF, while OUTMESS uses the variable SBUF.

First, GETBUF will set its extraction buffer index to 0 and set the scenario index to 0, also. This means that the scenario index will point to the top of the new (or old) buffer. GETBUF will then increment the scenario buffer index, storing the result in the variable NEWBUF, thus preserving the variable SBUF.

When GETBUF finds the new buffer to be "not full," the appropriate error counters are incremented and entered into GETBUF's extraction buffer. Note that GETBUF used the variable NEWBUF to point to the next buffer. The variable SBUF still holds the index for the last processed buffer. GETBUF next sets the time field of the first message in SBUF to 0. This indicates that the buffer is empty and insuring that the OUTMESS will not process the messages in that buffer a second time. GETBUF is then exited and after an extraction call OUTMESS will also be exited. The next time that OUTMESS is processed, it will encounter the zero time field on SBUF. This will allow GETBUF to try to get the next buffer again. Hopefully, the additional time will have allowed I/O Processing to fill the new buffer.

When GETBUF finds that the new buffer is full, then GETBUF will reset the ready and full flags of the buffer pointed to by OBUF. GETBUF will then increment OBUF. This will allow I/O Processing to fill this buffer. Note that OBUF is kept a few (determined in INITIAL) buffers behind the buffers being processed by OUTMESS. This allows messages saved on the OLD list to still be available for delayed processing. GETBUF then set SBUF, the index used by OUTMESS, to the NEWBUF index. GETBUF also sets the appropriate counters and extraction fields. GETBUF then checks the buffer for a read error. For buffers that do not have a read error, GETBUF sets the ready flag, extracts GETBUF's extraction buffer, and exits. For buffers that have a read error GETBUF, sets an extraction word and control will then revert to the top of GETBUF to get the next buffer. GETBUF keeps a count of the number of bad (read error) buffers found in consecutive order. When this number reaches 8, GETBUF deschedules OUTMESS and extracts an error. This is done so this condition does not set up an endless loop.

5.1.5.4 SEND MESSAGE.

SEND MESSAGE is a block of code that is included into the OUTMESS subroutine as in-line code. SEND MESSAGE will try to send the message pointed to by the variables MINDEX and MBUF by calling the subroutine TRANSMIT. A flowchart of SEND MESSAGE is given in figure 5.1.5.4-1.

SEND MESSAGE first checks to see if the message is a default message. When SEND MESSAGE finds a default message, then SEND MESSAGE will copy the first two words of the message into the first two words of the default buffer. SEND MESSAGE then calls the subroutine TRANSMIT passing it to the location of the message. Default messages are transmitted from the default message buffers while other messages are transmitted from directly the scenario buffers.

SEND MESSAGE will next check the return code from TRANSMIT for errors. A return code that equals 0 indicates that the message was successfully loaded into the hardware to be transmitted. SEND MESSAGE will then increment the messages sent statistics and exit. A nonzero return code indicates there was an error and that TRANSMIT did not load the message into the hardware.

When SEND MESSAGE finds a nonzero return code, it will check to see if the messages's port is a multi-line port. When the port is a multi-line port, SEND MESSAGE will increment through the lines in the port calling TRANSMIT for each line. SEND MESSAGE will continue to try to send the message until either it is successful or until it has tried all the lines in the port.

When there are no other lines available to TRANSMIT the message on, SEND MESSAGE will try to save it. SEND MESSAGE will first check the return to see if the error was a fatal error. For a fatal error SEND MESSAGE will increment the appropriate counters and set the appropriate fields in the extraction buffer. For a nonfatal error SEND MESSAGE will try to add the message to the OLD list. SEND MESSAGE will check the return code from the Fortran call ABL (add to the bottom of the list). When the ABL call returns an error SEND MESSAGE will increment the appropriate counters and set the appropriate fields in the extraction buffer.

The SEND MESSAGE block is then complete.

5.1.6 Second Processing.

The second processor is a Fortran program that is executed by the Scheduler every second. The purpose of this program is to process periodic events within the CID real-time system. Two calls are made to the extraction routine in second processing. The extractor is described in section 5.1.9 of this report. The first call will extract the software error counters. The second call will extract the X25 device errors. These error counters consist of all the errors that are detected by the real-time CID program. A flow chart of the second processing routine is given in figure 5.1.6-1. A list of software errors is shown in figure 5.1.6-2 and the list of X25 errors is shown in figure 5.1.6-3.

5.1.7 Operator Communications (Op-Comm).

The Op-Comm package provides the user interface to the real-time operations of the CID. The package consists of two processing subroutines; the routine Op-Comm which performs the statistical processing, and the routine Op-Comin which performs the input processing. A third file, Opblkdta is used to provide the database of display screens which appear on the two CID terminals. Together, the three processes provide the user with the capability of displaying system status and errors, displaying X.25 device status and errors, and the necessary functions required to control the CID simulation.

The Op-Comm package is executed by the Scheduler. Op-Comin processes terminal input buffers (a string of characters, typed by the operator, terminated by a carriage return) from the control terminal. Scheduling of Op-Comin for execution is performed by I/O on receipt of each completed terminal input buffer. A terminal input buffer allows the CID operator a means of sending data to the CID task. Op-Comin schedules the Op-Comm routine when the user's input requires statistics processing. The two routines are synchronized by parameters contained in the CID Fortran common blocks.

The Opblkdta routine is a Fortran block data subprogram which defines the various display screens. This database holds both the ASCII contents of each display screen and the cursor positioning information of each string on a particular screen. The individual display screens, or buffers, are selected for display by the Op-Comm package; the actual display of the screens is performed by I/O.

5.1.7.1 Operator Communications Input (Op-Comin).

Op-Comin is responsible for processing input from the operator, handling input errors, setting up for the display of statistical screen templates, and setting up pointers for the next function to be processed in both Op-Comin and Op-Comm. Op-Comin is also responsible for scheduling Op-Comm, but only when the operator's request requires statistical processing.

The Op-Comin routine consists of ten functions. These functions correspond to the individual menu items of figure 5.1.7.1-1. An optimized version of a subroutine call is used to vector to one of these functions depending on the user's response to the CID menu. This subroutine call is based on an Assembly Language implementation of a Fortran "computed GOTO" statement. It uses a location table to look up a subroutine address based on the state condition (held in OPINFUNC) passed to it. The key to this algorithm is that there is no parameter list associated

with the subroutines. Instead, common blocks are used to pass information to the subroutine. This allows the standard subroutine call overhead to be bypassed, hence speeding up the vectoring process. Figure 5.1.7.1-2 shows a flowchart of the computed GOTO algorithm.

On entering the Op-Comin main routine, the first thing done is to check OPINSTATE. This variable, set during Initialization and each subsequent state of Op-Comin, determines whether a previous function has been completed. When OPINSTATE does not equal 0, program control is transferred to the appropriate function, using OPINFUNC as the vector, by utilizing the computed GOTO algorithm.

When all previous states of a function have been completed, Op-Comin is ready to process the next function request from the user. This function request comes in the form of a filled input buffer.

The input buffer processed in response to the CID menu must be converted into appropriate form. In response to the CID menu, this requires the conversion of the input buffer to integer representation. It is known that only one character need be processed since the response at this point must be a value "1" through "9." When any other value is typed in by the user, an error message is displayed and the user is asked to try again. Figure 5.1.7.1-3 shows a flowchart of the Op-Comin routine.

5.1.7.1.1 Input Processing.

Due to time constraints placed on the Op-Comin processing routine by the CID design, standard Fortran I/O processing (i.e., reads and writes) is inadequate for use in communicating with the CID program. A series of algorithms have been developed for each type of input expected by Op-Comin. The user is able to communicate directly with the CID program only through certain windows provided by the Op-Comin routines. Input to the simulation occurs when:

a. Responding to the CID menu.
b. Selecting an X.25 device to activate or deactivate.
c. Displaying a message.
d. Creating a message.

Figure 5.1.7.1.1-1 lists each type of input expected and the processing routines associated with it. Any input by the user must be in response to a menu or question displayed on the CID control terminal. An input comes in the form of a filled input buffer. The input buffers processed throughout the Op-Comin routine are ASCII representations of the input. A filled input buffer is defined as a buffer of 80 characters (bytes) in length, or a buffer less than 80 characters terminated by a hexadecimal (hex) D, and the ASCII representation of a carriage return <cr>. The user has the choice of filling an input buffer with a character string within the range defined in the query or taking the default value by simply hitting the carriage return. In either case, the range of the input allowed and the default value taken are given in the CID user's manual.

In order to process input into usable data, the information must be converted into appropriate format for the intended operation. An input buffer is requested to be opened by Op-Comin, prepared by I/O, and ultimately filled by the user in response to the displayed CID menu or query. The input buffer is guaranteed to be ready for processing by Op-Comin because I/O does not schedule the Op-Comin routine until the input buffer has been filled by the user.

Single character input is the simplest input to process. It is known that only one character need be processed, and that the response from the user be one of two possible values. Figure 5.1.7.1.1-2 shows a flowchart of Single Character Input Processing. The algorithm to perform this process using "Y" and "N" (for YES and NO) as the expected inputs is listed below:

a. Load the high order byte of the input buffer, IVAR(1), into INTBYTE.

b. If the value of INTBYTE is "Y" (hex 59), perform the YES branch of the necessary logic.

c. Otherwise, perform the NO branch of the necessary logic.

Single digit decimal input requires slightly different processing. In this case, the conversion of the input buffer is to integer. It is known that only one character need be processed since the response at this point should be a 0 through 9. Figure 5.1.7.1.1-3 shows the flowchart of Single Digit Decimal Input Processing. The algorithm to perform this process is listed below:

a. Load the high order byte of the input buffer, IVAR(1), into INTBYTE.

b. Compare INTBYTE with <cr>. A <cr> signifies the end of the input buffer.

c. When INTBYTE is not <cr>, insure that INTBYTE is between hex 30 and 39, subtract hex 30 from INTBYTE, and save this value for later use.

d. When INTBYTE is <cr>, the user selected the default. This default value depends on the individual function and state. The CID menu does not use a default, a value from "0" to "9" is required.

e. Otherwise, if INTBYTE was not between hex 30 and hex 39 and not a <cr>, an error message will be queued to the control screen by setting COMBUF equal to 8. I/O will inform that an input buffer needs to be opened by setting IOBUFILL equal to TRUE, and return to the Scheduler while keeping OPINSTATE unmodified so that the next input buffer is processed for the same function and state.

Several digit decimal inputs handle responses from the user from "0" to "9999." The user's response is terminated by a <cr>. In this case, the conversion of the input buffer is to integer. Figure 5.1.7.1.1-4 shows a flowchart of Several Digit Decimal Input Processing. The algorithm to perform this process is listed below:

a. Set I to 1. Set DECI VAL to 0. Load the high order byte of the input buffer, IVAR(1), into INTBYTE(I).

b. Compare INTBYTE(I) with <cr>. A <cr> signifies the end of the input buffer. When I is 1 and a <cr> is entered, i.e., the user did not enter a value but only the <cr>, the user has selected the default value to be used, if it is applicable for the function and state process in effect.

c. When INTBYTE(I) is not <cr>, insure that INTBYTE is between hex 30 and 39 and set DECI VAL equal to (DECI VAL *10) + (INTBYTE(I) - X'30'). Increment I by 1, and check to see if the maximum of four digits has been processed. If so, jump out of this loop to step 5. If the maximum of four digits has not been processed, load the next byte of the input buffer IVAR into the next halfword of INTBYTE and jump back to step 2.

d. When INTBYTE(I) is not <cr>, and INTBYTE(I) is not between hex 30 and 39, then an illegal digit has been entered and the user is informed to try again. COMBUF is set to 7 to queue the message, an input buffer is opened by setting OIBUFILL to TRUE, and Opcomm is descheduled.

e. When acceptable input terminated by a <cr> has been processed at this point, a check is performed to verify that DECI VAL is within the acceptable limits for this function and state. Again, if DECI VAL is not acceptable, COMBUF is set to 7, an input buffer is opened, and Op-Comm is descheduled.

Single digit hex input requires only one digit input processing. In this case, the conversion of the input buffer is to integer. It is known that only one character need be processed since the response at this point should be a hex 0 through F. Figure 5.1.7.1.1-5 shows the flowchart of Single Digit Hexadecimal Input Processing. The algorithm to perform this process is listed below:

a. Load the high order byte of the input buffer, IVAR(1), into INTBYTE.

b. Compare INTBYTE with <esc>. An <esc> signifies the input buffer contains an escape character.

c. When INTBYTE is not <esc>, insure that INTBYTE is between hex 30 and 46. Check if INTBYTE is greater than hex 39. If it is, subtract hex 37 from INTBYTE, or else subtract hex 30 from INTBYTE and save this value for later use.

d. When INTBYTE is <esc>, the user selected the default.

e. Otherwise, if INTBYTE was not between hex 30 and hex 46, queue an error message to the control screen by setting COMBUF equal to 15. Inform I/O that an input buffer needs to be opened by setting IOBUFILL equal to TRUE, and return to the Scheduler by setting OPINSTATE equal to 7 in order that the next input buffer can be processed for the same function and state.

A two-digit hex input requires up to two-digit input processing. In this case, the conversion of the input buffer is to integer. It is known that two characters need to be processed since the response at this point should be a hex 0 through FF. Figure 5.1.7.1.1-6 shows the flowchart of Two-Digit Hexadecimal Input Processing. The algorithm to perform this process is listed below:

a. Load the high order byte of the input buffer, IVAR(1), into INTBYTE.

b. Compare INTBYTE with <esc>. An <esc> signifies the input buffer contains an escape character.

c. When INTBYTE is not <esc>, insure that INTBYTE is between hex 30 and 46. Check if INTBYTE is greater than hex 39. If it is, subtract hex 37 from INTBYTE, or else subtract hex 30 from INTBYTE and save this value for later use. Increment I by 1, and check if the maximum of two digits has been processed. If two digits have been processed, save this value for later use and perform the next state processing. If two digits have not been processed, load the next byte of the input buffer IVAR into the next halfword of INTBYTE.

d. Compare INTBYTE with <cr>. If it is a <cr>, perform the next state processing. If it is not a <cr>, repeat step c.

e. When INTBYTE is <esc>, the user selected the default.

f. Otherwise, if INTBYTE was not between hex 30 and hex 46, queue an error message to the control screen by setting COMBUF equal to 15. Inform I/O that an input buffer needs to be opened by setting IOBUFILL equal to TRUE, and return to the Scheduler by setting OPINSTATE equal to the current state in order that the next input buffer can be processed for the same function and state.

A six-digit hex input requires up to six-digit input processing. In this case, the conversion of the input buffer is to integer. It is known that six characters need to be processed since the response at this point should be a hex 0 through FFFFFF. Figure 5.1.7.1.1-7 shows the flowchart of Six-Digit Hexadecimal Input Processing. The algorithm to perform this process is listed below:

a. Load the high order byte of the input buffer, IVAR(1), into INTBYTE.

b. Compare INTBYTE with <esc>. An <esc> signifies the input buffer contains an escape character.

c. When INTBYTE is not <esc>, insure that INTBYTE is between hex 30 and 46. Check if INTBYTE is greater than hex 39. If it is, subtract hex 37 from INTBYTE, or else subtract hex 30 from INTBYTE and save this value for later use. Increment I by 1, and check if the maximum of six digits has been processed. If six digits have been processed, save this value for later use and perform the next state processing. If six digits have not been processed, load the next byte of the input buffer IVAR into the next halfword of INTBYTE.

d. Compare INTBYTE with <cr>. If it is a <cr>, perform the next state processing. If it is not a <cr>, repeat step c.

e. When INTBYTE is <esc>, the user selected the default.

f. Otherwise, if INTBYTE was not between hex 30 and hex 46, queue an error message to the control screen by setting COMBUF equal to 15. Inform I/O that an input buffer needs to be opened by setting IOBUFILL equal to TRUE, and return to the Scheduler by setting OPINSTATE equal to the current state in order that the next input buffer can be processed for the same function and state.

A 14-digit hex input requires a somewhat different approach to decoding. All other input until now required only one fullword of storage space to hold the result. In this case, the result requires two fullwords of storage space. Also, since hex utilizes the characters A through F to represent the decimal values 10 through 15, the check for a legal digit in the input buffer becomes more complicated. Figure 5.1.7.1.1-8 shows a flowchart of 14-Digit Hex Input Processing. The algorithm below describes the process:

a. I is initialized to 1. The first byte of the input buffer, IVAR(1), is loaded into the processing buffer INTBYTE(1) which is a halfword-oriented array.

b. When the user simply entered a <cr>, INPT2 CRSR POS is incremented by 1, and the cursor position buffer is queued for output by setting COMBUF to 5. An input buffer is opened by setting OIBUFILL to TRUE, and a check is made to see if

F5 UPDATE FLAG is set to 1. When the flag is set to 1, it means that this is an update to track already on the CTL. In this case, the parameter cannot be updated. When F5 UPDATE FLAG is not 1 and the user typed only a <cr>, the default values for the two variables generated are set to the default values.

c. When the first character in the input buffer was not a <cr>, 14 hex characters are expected. When INTBYTE(I) is between hex 30 and 39, the value is normalized to hex by subtracting hex 30 from it. When INTBYTE(I) is between hex 41 and 46 (A and F), the value is normalized by subtracting hex 37 from it. Otherwise, the INTBYTE(I) is considered illegal and error processing is invoked by setting COMBUF equal to 7, OIBUFILL to TRUE, and descheduling Op-Comm.

d. When INTBYTE(I) holds a valid hex value, the pointer into the input buffer, IVAR, is set to increment by one halfword, and the byte pointer into the proper halfword element of IVAR is selected. I the index into the INTBYTE array is also incremented by 1, and using the pointers just computed, the next byte of the input buffer is loaded into the next halfword of the INTBYTE array.

e. Now that the index I has been incremented to the next value, it can be compared against the value of 15. When I is less than 15, the control passes back to step 2. When I is equal to 15, the final format of the data is computed. This is done by shifting the INTBYTE array elements by magnitudes of 10 according to the individual function state requirements which yields two fullwords of computed values.

A 20-digit hex input requires up to 20 digit input processing. All other input until now required only one fullword of storage space to hold the result. In this case, the result requires three fullwords of storage space. Figure 5.1.7.1.1-9 shows a flowchart of 20-Digit Hexadecimal Input Processing. The algorithm below describes the process.

a. I is initialized to 1. The first byte of the input buffer, IVAR(1), is loaded into the processing buffer INTBYTE(1) which is a halfword-oriented array.

b. Compare INTBYTE with <esc>. An <esc> signifies the input buffer contains an escape character.

c. When INTBYTE is not <esc>, insure that INTBYTE is between hex 30 and 46. Check if INTBYTE is greater than hex 39. If it is, subtract hex 37 from INTBYTE, or else subtract hex 30 from INTBYTE and save this value for later use. Increment I by 1, and check if the maximum of 20 digits has been processed. If 20 digits have been processed, save this value to process three fullword input. The fullword processing is done by mutiplying the appropriate hex bit field to each input byte and storing it in a buffer. If 20 digits have not been processed, load the next byte of the input buffer IVAR into the next halfword of INTBYTE.

d. Compare INTBYTE with <cr>. If it is a <cr>, perform the next state processing. If it is not a <cr>, repeat step c.

e. When INTBYTE is <esc>, the user selected the default.

f. Otherwise, if INTBYTE was not between hex 30 and hex 46, queue an error message to the control screen by setting COMBUF equal to 15. Inform I/O that an input buffer needs to be opened by setting IOBUFILL equal to TRUE, and return to the Scheduler by setting OPINSTATE equal to the current state in order that the next input buffer can be processed for the same function and state.

5.1.7.1.2 Function 0 - Output Menu.

Function 0, the output menu subroutine, is not directly accessible by the operator. This function is invoked whenever the simulation is initialized or any function is concluded. The purpose of this function is to queue the CID menu buffer for display by setting COMBUF to 1 and permit input to be accepted by I/O's input buffer by setting OIBUFILL to TRUE. In addition, OPINSTATE is set to 0, informing the main routine of Op-Comin to process the next input buffer for a new function value. Figure 5.1.7.1.2-1 shows a flowchart of Op-Comin function 0.

5.1.7.1.3 Function 1 - Display Statistics.

Function 1, the display statistics subroutine, will toggle the display of the CID system statistics. When the statistics function is enabled (OPINFUNC is equal to 1) on entering this function, it becomes disabled. Op-Comm is descheduled by setting the next time to process Op-Comm, ITIME(3), to the largest number possible (hex 7FFFFFFF). Also, the frequency of scheduling Op-Comm, INCTIME(3), is set to a negative number (hex 80000000). OPINFUNC, which contains the next function to process, is set to 0. The CID menu buffer is queued by setting COMBUF equal to 1 and an input buffer is opened allowing the user to type in his next response to the CID menu by setting OIBUFILL to TRUE. OPINSTATE is set to 0 for the next time through the Op-Comin main routine.

When the Statistic Terminal is disabled (OPINFUNC is not equal to 1) it now becomes enabled by setting OPINFUNC to 1. This informs Op-Comm of the function to process. Op-Comm is scheduled to run in 1 second from now and every 4 seconds, thereafter, by setting ITIME(3) to the current system time, plus 1000 STUs and setting INCTIME(3) to the value of the schedule frequency variable, SCH UPD FREQ, which is 4 seconds. COOBUF is set to 1 to queue the template buffer of the display statistics function. COMBUF is set to 1 to queue the menu buffer for display. An input buffer is opened allowing a response to the menu, and finally, OPINSTATE is set to 0, signifying the completion of Op-Comin's processing of this function. Figure 5.1.7.1.3-1 shows a flowchart of Op-Comin function 1.

5.1.7.1.4 Function 2 - Display System Errors.

Function 2, the display system errors subroutine, will toggle the display of the CID system errors. When the error function is currently enabled (OPINFUNC is equal to 2), it becomes disabled. Op-Comm is descheduled by setting the next time to process Op-Comm, ITIME(3), to the largest number possible (hex 7FFFFFFF). Also, the frequency of scheduling Op-Comm, INCTIME(3), is set to a negative number (hex 80000000). OPINFUNC, which contains the next function to process, is set to 0. The CID menu buffer is queued by setting COMBUF equal to 1, and a buffer is opened allowing the user to type in his next response to the CID menu by setting OIBUFILL to TRUE. OPINSTATE is set to 0 for the next time through the Op-Comin main routine.

When the system error function is disabled (OPINFUNC is not equal to 2), it now becomes enabled by setting OPINFUNC to 2. Op-Comm is scheduled to run in 1 second from the current system time and every 4 seconds, thereafter, by setting ITIME(3) to the current system time, plus 1000 STUs and setting INCTIME(3) to the value of the schedule frequency variable, SCH UPD FREQ, which is 4 seconds. COOBUF is set to 1 to queue the template buffer of the system error function. COMBUF is set to 1 to queue the menu buffer for display. An input buffer is opened allowing a response to the menu, and finally, OPINSTATE is set to 0, signifying the completion of Op-Comin's processing of this function. Figure 5.1.7.1.4-1 shows a flowchart of Op-Comin function 2.

5.1.7.1.5 Function 3 - Display X.25 Statistics.

Function 3, the display X.25 statistics subroutine, toggles the display of the CID system errors. When the error function is currently enabled (OPINFUNC is equal to 3), it becomes disabled. Op-Comm is descheduled by setting the next time to process Op-Comm, ITIME(3), to the largest number possible (hex 7FFFFFFF). Also, the frequency of scheduling Op-Comm, INCTIME(3), is set to a negative number (hex 80000000). OPINFUNC, which contains the next function to process, is set to 0. The CID menu buffer is queued by setting COMBUF equal to 1 and a buffer is opened allowing the user to type in his next response to the CID menu by setting OIBUFILL to TRUE. OPINSTATE is set to 0 for the next time through the Op-Comin main routine.

When the X.25 statistic function is disabled (OPINFUNC is not equal to 3), it becomes enabled by setting OPINFUNC to 3. Op-Comm is scheduled to run in 1 second from the current system time and every 4 seconds, thereafter, by setting ITIME(3) to the current system time, plus 1000 STUs and setting INCTIME(3) to the value of the schedule frequency variable, SCH UPD FREQ, which is 4 seconds. COOBUF is set to 5 to queue the template buffer of the X.25 statistics function. COMBUF is set to 1 to queue the menu buffer for display. An input buffer is opened allowing a response to the menu, and finally, OPINSTATE is set to 0, signifying the completion of Op-Comin's processing of this function. Figure 5.1.7.1.5-1 shows a flowchart of Op-Comin function 3.

5.1.7.1.6 Function 4 - Display X.25 Errors.

Function 4, the display X.25 errors subroutine, toggles the display of the CID X.25 errors. When the error function is currently enabled (OPINFUNC is equal to 4), it becomes disabled. Op-Comm is descheduled by setting the next time to process Op-Comm, ITIME(3), to the largest number possible (hex 7FFFFFFF). Also, the frequency of scheduling Op-Comm, INCTIME(3), is set to a negative number (hex 80000000). OPINFUNC, which contains the next function to process, is set to 0. The CID menu buffer is queued by setting COMBUF equal to 1 and a buffer is opened allowing the user to type in his next response to the CID menu by setting OIBUFILL to TRUE. OPINSTATE is set to 0 for the next time through the Op-Comin main routine.

When the X.25 error function is disabled (OPINFUNC is not equal to 4), it becomes enabled by setting OPINFUNC to 4. Op-Comm is scheduled to run in 1 second from the current system time and every 4 seconds, thereafter, by setting ITIME(3) to the current system time, plus 1000 STUs and setting INCTIME(3) to the value of the schedule frequency variable, SCH UPD FREQ, which is 4 seconds. COOBUF is set to 7 to queue the template buffer of the X.25 error function. COMBUF is set to 1 to queue the menu buffer for display. An input buffer is opened allowing a response to the menu, and finally, OPINSTATE is set to 0, signifying the completion of Op-Comin's processing of this function. Figure 5.1.7.1.6-1 shows a flowchart of Op-Comin function 4.

5.1.7.1.7 Function 5 - Modify the X.25 Device Status.

Function 5, the modify X.25 device status subroutine, toggles the status of a selected X.25 device, when requested.

The function has three states. In state 1, the modify X.25 query buffer is queued for display by setting COMBUF equal to 3. An input buffer is opened for the user's response by setting OIBUFILL to TRUE. OPINSTATE is set to 2 for the next time through this function (after the user responds to the modify query). OPER FUNC is set to 3 which selects the X.25 statistics function for execution. OPER STATE is set to 0. ITIME(3) and INCTIME(3) are set to run the X.25 statistics function in 1 second and every 4 seconds, thereafter. Finally, COOBUF is set to 5 which queues the X.25 statistics template screen for display.

After the user responds to the modify query, Op-Comin uses the flags set in state 1 to execute state 2 of function 5. In this state, the input buffer is processed. When a valid input is decoded (0 through 11, or an <esc>), the error processing routine is skipped. The error processing simply reselects function 5 state 2 for reexecution after clearing the user's errant input from the control screen.

When the routine decodes an <esc>, function 5 is terminated after the CID function menu is queued for display by setting COMBUF equal to 1.

When the routine decodes a device from 0 through 11, a check is performed to verify that the device was configured during CID Initializaion. When the check finds that the device is not configured (ST X25 CONFIGURED(DEV NUMBER) is not greater than 0), COMBUF is set to 5 which queues the "not configured" error buffer and function 5 state 2 is selected for execution once again. When the configuration test declares that the decoded device number is configured (ST X25 CONFIGURED(DEV NUMBER) is greater than 0), the selected device number is checked to determine whether the device is currently active. When the device is active (CDX ACTIVE FLAG is TRUE), DEVICE ACTIVE is set TRUE and COMBUF is set to 6 which queues the "OK to de-activate" buffer. When the device is inactive (CDX ACTIVE FLAG is FALSE), DEVICE ACTIVE is set FALSE and COMBUF is set to 7 which queues the "OK to activate" buffer. Whether the device is active or inactive OPIN STATE is set to 3, selecting function 5 state 3 for execution after the user responds to the displayed buffer.

In state 3, when the user responds with a "N" to the displayed buffer selected in state 2, state 2 is reselected for execution by setting COMBUF to 3 and OPIN STATE to 2. Otherwise, DEVICE NUMBER is checked and the appropriate CDX ACTIVE FLAG is toggled by performing a logical NOT on the value. Next, COMBUF is set to 3 and OPIN STATE is set to 2, which selects state 2 of function 5 for execution after displaying the associated buffer. In any case, an input buffer is opened before terminating function 5 state 3 by setting OIBUFILL to TRUE. Figure 5.1.7.1.7-1 shows a flowchart of function 5.

5.1.7.1.8 Function 6 - Create A Message.

The Create A Message subroutine, function 6 of the CID main menu, will allow the user to create various messages to be transmitted on the designated port. Figure 5.1.7.1.8-1 shows a flowchart of Op-Comin function 6.

The function consist of various states. The first state will queue the Create A Message option screen to terminal 1 by setting COMBUF equal to 24. An input buffer is opened for a user response by setting OIBUFILL equal to TRUE. OPINSTATE is set to 2 for the next state to process. OPERFUNC is set to 6 which allows the Create A Message function to be executed. ITIME(3) and INCTIME(3) are set to run the Create A Message function in 1 second at every 4-second intervals, respectively. Terminal 2 is cleared by setting COOBUF equal to 10. Figure 5.1.7.1.8-2 shows a flowchart of Op-Comin function 6 - State 1.

In the second state, the user selects a message to create. Once the user selects a message to be created, Op-Comin recognizes the flags which were set in state 1 to execute state 2 of function 6. If the user does not type a valid entry from 1 to 7, then Create A Message screen is queued again. This is done by setting COMBUF equal to 24, OIBUFILL equal to TRUE, and OPINSTATE equal to 2. If a valid input is entered, then OPINSTATE is set to 3 to allow state 3 to be processed. Again, OIBUFILL is set to TRUE to allow user response. COMBUF is set to 25 to queue the Assigned Message Number buffer. If the user enters a 7, then OPINSTATE is set to 0 to inform Op-Comin to ignore all states in function 6. COMBUF is set to 1 to allow the main CID menu buffer to be queued and OIBUFILL is set to TRUE. Figure 5.1.7.1.8-3 shows a flowchart of Op-Comin function 6 - State 2.

In the third state, the user enters the assigned message number. If the user enters a valid input (0-F), then COMBUF is set equal to 26, OPINSTATE equal to 4, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 4 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF25(93:94) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 32-bit integer value and stored in MSGNO to be added to the transmission buffer (BUFFER). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "-" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 3. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 3. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 4 and the X.25 Port No. buffer will be queued. This is done by setting OPINSTATE equal to 4, COMBUF equal to 26, and OIBUFILL equal to TRUE. Figure 5.1.7.1.8-4 shows a flowchart of Op-Comin function 6 - States 3 to 17.

In the fourth state, the user enters the X.25 port number. If the user enters a valid input (1-6), then COMBUF is set equal to 27, OPINSTATE equal to 5, and OIBUFILL equal to TRUE, for Create A Message selection (1-5). This will allow Op-Comin to execute state 5 in function 6 for the next time around this subroutine. For a Create A Message selection of (6), COMBUF is set equal to 38, OPINSTATE equal to 6, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 6 in function 6 for the next time around this subroutine.

The input is converted to ASCII and stored in OMBUF26(93:93) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 32-bit integer value and stored in the transmission buffer (PORT). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "-" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 4. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 4. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 5 and the Mode S ID buffer will be queued for Create A Message selection (1-5). This is done by setting OPINSTATE equal to 5, COMBUF equal to 27, and OIBUFILL equal to TRUE. If the FILL flag was set high when the <esc> key was entered, then state 6 and the Message Length buffer will be queued, for Create A Message selection (6). This is done by setting OPINSTATE equal to 6, COMBUF equal to 38, and OIBUFILL equal to TRUE.

In the fifth state, the user enters the Mode S ID. If the user enters a valid input (0-F) for each of the six-hex digits, then COMBUF is set equal to 28, OPINSTATE equal to 7, and OIBUFILL equal to TRUE, for Create A Message selection (1-3). This will allow Op-Comin to execute state 7 in function 6 for the next time around this subroutine. For Create A Message selection (4), COMBUF is set equal to 40, OPINSTATE equal to 18, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 18 in function 6 for the next time around this subroutine. For a Create A Message selection (5), COMBUF is set equal to 36, OPINSTATE equal to 8, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 8 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF27(93:98) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 24-bit integer value and stored in MODESIN to be added to the transmission buffer (BUFFER). The BYTE COUNT is set to 5 bytes, if Create A Message selection was 4. If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "/" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 5. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 5. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 7 and the Priority buffer will be queued, for Create A Message selection (1-3). This is done by setting OPINSTATE equal to 7, COMBUF equal to 28, and OIBUFILL equal to TRUE. If the FILL flag was set high when the <esc> key was entered, then state 18 and the OK To Send Message

buffer will be queued, for Create A Message selection (4). This is done by setting OPINSTATE equal to 18, COMBUF equal to 40, and OIBUFILL equal to TRUE. If the FILL flag was set high when the <esc> key was entered, then state 8 and the Reference Message Number buffer will be queued, for Create A Message selection (5). This is done by setting OPINSTATE equal to 8, COMBUF equal to 36, and OIBUFILL equal to TRUE.

In the sixth state, the user enters the message length. If the user enters a valid input (0-F) for each of the two-hex digits, then COMBUF is set equal to 39, OPINSTATE equal to 9, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 9 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF38(93:94) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to an 8-bit integer value and stored in MSGLN to be added to the transmission buffer (BUFFER). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "/" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 6. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 6. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 9 and the Message Bit Stream buffer will be queued. This is done by setting OPINSTATE equal to 9, COMBUF equal to 39, and OIBUFILL equal to TRUE.

In the seventh state, the user enters the priority field. If the user enters a valid input (0-F), then COMBUF is set equal to 29, OPINSTATE equal to 10, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 10 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF28(93:93) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 24-bit integer value and stored in PRIORITY to be added to the transmission buffer (BUFFER). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "0" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 7. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 7. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 10 and the Expiration buffer will be queued. This is done by setting OPINSTATE equal to 10, COMBUF equal to 29, and OIBUFILL equal to TRUE.

In the eighth state, the user enters the reference message number. If the user enters a valid input (0-F) for the two-hex integers, then COMBUF is set equal to 37, OPINSTATE equal to 11, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 11 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF36(93:94) to allow the user to view what was typed in when a response is made not to send the message. The input is also

converted to a 20-bit integer value and stored in RMSGNO to be added to the transmission buffer (BUFFER). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "0" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 8. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 8.

If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 11 and the Reference Type Code buffer will be queued. This is done by setting OPINSTATE equal to 11, COMBUF equal to 37, and OIBUFILL equal to TRUE.

In the ninth state, the user enters the message bit stream. If the user enters a valid input (0-F) for the 20-hex integers, then COMBUF is set equal to 40, OPINSTATE equal to 18, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 18 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF39(93:112) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a series of 32-bit integer values and stored in STREAM to be added to the transmission buffer (BUFFER). The BYTE COUNT is computed by taking MSGLN, add 3 to it, and divide by 2 to get the number of characters to transmit. If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "0" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 9. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 9. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 18 and the OK To Send Message buffer will be queued. This is done by setting OPINSTATE equal to 18, COMBUF equal to 40, and OIBUFILL equal to TRUE.

In the tenth state, the user enters the expiration field. If the user enters a valid input (0-7), then COMBUF is set equal to 30, OPINSTATE equal to 12, and OIBUFILL equal to TRUE, for Create A Message selection (1). This will allow Op-Comin to execute state 12 in function 6 for the next time around this subroutine. For Create A Message selection (2), COMBUF is set equal to 33, OPINSTATE equal to 13, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 13 in function 6 for the next time around this subroutine. For Create A Message selection (3), COMBUF is set equal to 35, OPINSTATE equal to 14, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 14 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF29(93:93) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 20-bit integer value and stored in EXPIRE to be added to the transmission buffer (BUFFER). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "1" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 10. This will display that the user typed in an invalid

input and to notify to reenter another input, reposition the cursor, and reexecute state 10. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry.

If the FILL flag was set high when the <esc> key was entered, then state 12 and the Acknowledgement buffer will be queued, for Create A Message selection (1). This is done by setting OPINSTATE equal to 12, COMBUF equal to 30, and OIBUFILL equal to TRUE. If the FILL flag was set high when the <esc> key was entered, then state 13 and the No. of Segments To Send buffer will be queued, for Create A Message selection (2). This is done by setting OPINSTATE equal to 13, COMBUF equal to 33, and OIBUFILL equal to TRUE. If the FILL flag was set high when the <esc> key was entered, then state 14 and the BDS Field buffer will be queued, for Create A Message selection (3). This is done by setting OPINSTATE equal to 14, COMBUF equal to 35, and OIBUFILL equal to TRUE.

In the eleventh state, the user enters the reference type code. If the user enters a valid input (0-F) for the two hex integers, then COMBUF is set equal to 40, OPINSTATE equal to 18, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 18 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF37(93:94) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 16-bit integer value and stored in REFTYPE to be added to the transmission buffer (BUFFER). The BYTE COUNT is set to 7 bytes. If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "1" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 11. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 11. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 18 and the OK To Send Message buffer will be queued. This is done by setting OPINSTATE equal to 18, COMBUF equal to 40, and OIBUFILL equal to TRUE.

In the twelfth state, the user enters the acknowledgement bit. If the user enters a valid input (0-1) for the bit integer, then COMBUF is set equal to 31, OPINSTATE equal to 15, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 15 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF30(93:93) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 20-bit integer value and stored in ACK to be added to the transmission buffer (BUFFER). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "2" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 12. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 12. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 15 and the Segment Count buffer will be queued. This is done by setting OPINSTATE equal to 15, COMBUF equal to 31, and OIBUFILL equal to TRUE.

In the thirteenth state, the user enters the number of segments to send. If the user enters a valid input (0-4) for the hex integer, then COMBUF is set equal to 34, OPINSTATE equal to 16, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 16 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF33(93:93) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 12-bit integer value and stored in SEGMENTS to be added to the transmission buffer (BUFFER). If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "2" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 12. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 13. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 16 and the ELM Default Text buffer will be queued. This is done by setting OPINSTATE equal to 15, COMBUF equal to 34, and OIBUFILL equal to TRUE.

In the fourteenth state, the user enters the B-Definition Subfield (BDS) field. If the user enters a valid input (0-F) for the two-hex integers, then COMBUF is set equal to 40, OPINSTATE equal to 18, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 18 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF35(93:94) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 16-bit integer value and stored in BDS to be added to the transmission buffer (BUFFER). The BYTE COUNT is set to 7 bytes. If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "2" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 14. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 14. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 18 and the OK To Send Message buffer will be queued. This is done by setting OPINSTATE equal to 18, COMBUF equal to 40, and OIBUFILL equal to TRUE.

In the fifteenth state, the user enters the segment count. If the user enters a valid input (0-3) for the hex integer, then COMBUF is set equal to 32, OPINSTATE equal to 17, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 17 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF31(93:93) to allow the user to view what was typed in when a response is made not to send the message. The input is also converted to a 16-bit integer value and stored in SEGCOUNT to be added to the transmission buffer (BUFFER). OMBUF32(114:157) is cleared for the next state. If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "3" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 15.

This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 15. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 17 and the Comm-A Message buffer will be queued. This is done by setting OPINSTATE equal to 17, COMBUF equal to 32, and OIBUFILL equal to TRUE.

In the sixteenth state, the user enters the elm default text to transmit. If the user enters a valid input (0-5) for the hex integer, then COMBUF is set equal to 40, OPINSTATE equal to 18, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 18 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF34(93:93) to allow the user to view what was typed in when a response is made not to send the message. The input is stored in ELMDEF to be determined later for the desired default text to transmit. The BYTE COUNT is set to 17 bytes, if the number of segments to send was a 0, 27 bytes, if the number of segments to send was a 1, 37 bytes, if the segments to send was a 2, 47 bytes, if the number of segments to send was a 3, and 57 bytes, if the number of segments to send was a 4. If the user set the ELMDEF to 1, then the ELMTXT(1-14) buffer will be left as a series of 32 bits of hex As, Bs, Cs, Ds, and Es. If the user set the ELMDEF to 2, then the ELMTXT(1-14) buffer will be set as a series of 32 bits of hex Bs, Cs, Ds, Es, and As, and so forth. If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "3" and "F," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 16. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 16. If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 18 and the OK To Send Message buffer will be queued. This is done by setting OPINSTATE equal to 18, COMBUF equal to 40, and OIBUFILL equal to TRUE.

In the seventeenth state, the user enters the Comm-A message to transmit. If the user enters a valid input (0-F) for the 56 hex integers, then COMBUF is set equal to 40, OPINSTATE equal to 18, and OIBUFILL equal to TRUE. This will allow Op-Comin to execute state 18 in function 6 for the next time around this subroutine. The input is converted to ASCII and stored in OMBUF32(99:157) to allow the user to view what was typed in when a response is made not to send the message. The input is converted to 32-bit integers and stored in a series of COMMA buffers to be stored in the transmission buffer (BUFFER). The BYTE COUNT is set to 14 bytes, if the segment count was a 0, 21 bytes, if the segment count was a 1, 28 bytes, if the segment count was a 2, and 35 bytes, if the segment count was a 3. If the user types an invalid input, then OMBUF15(52:52) and OMBUF15(55:55) are set to "5" and ""," respectively, prior to when COMBUF is set to 15 and OPINSTATE is set to 17. This will display that the user typed in an invalid input and to notify to reenter another input, reposition the cursor, and reexecute state 17.

If the user types in an <esc> key and the FILL flag is not set high, then the user will be requested to reenter the input, and the error message will be displayed similar to the invalid input prompt. The FILL flag allows the user to skip over this state, had the user already typed a previous value for this entry. If the FILL flag was set high when the <esc> key was entered, then state 18 and the OK To Send Message buffer will be queued. This is done by setting OPINSTATE equal to 18, COMBUF equal to 40, and OIBUFILL equal to TRUE.

In the final state, state 18, the user responds to a request to send the message that was just created. If the user enters a 'Y' or a <cr>, then the word length (WLENGTH), is calculated for transmission of the message. The WLENGTH is calculated by adding 3 to the BYTE COUNT and divide its result by 4. The BUFFER which contains the message created is added to the bottom of the message-scenario list. The FILL flag is reset to 0 to ignore <esc> keys. After the message gets transmitted, then the Create A Message menu is returned for selection. This is done by setting COMBUF to 24, OPINSTATE to 2, and OIBUFILL to TRUE. If the user enters a "N" to the response, then the FILL flag is set to 1 to allow the user to enter an <esc> key. The user, at this point is returned to state 3 to reenter the message created. This is done by setting COMBUF to 25, OPINSTATE to 3, and OIBUFILL to TRUE. The user is returned to state 18 for an invalid input response. This is done by setting COMBUF back to 40, OPINSTATE back to 18, and OIBUFILL to TRUE. Figure 5.1.7.1.8-5 shows a flowchart of Op-Comin Function 6 - State 18.

5.1.7.1.9 Function 7 - Display A Message.

The Display A Message subroutine, function 7 of the CID main menu, will allow the user to display various messages on the status terminal. Figure 5.1.7.1.9-1 shows a flowchart of Op-Comin Function 7 - main subroutine.

The function consist of various states. The first state will queue the Display A Message option screen to terminal 1 by setting COMBUF equal to 9. An input buffer is opened for a user response by setting OIBUFILL equal to TRUE. OPINSTATE is set to 2 for the next state to process. OPERFUNC is set to 7 which allows for the Display A Message function to be executed. ITIME(3) and INCTIME(3) are set to run the Display A Message function in 1 second at every 4-second intervals, respectively. Terminal 2 is queued with a Display Message List template by setting COOBUF equal to 8. This screen is updated at every 4-second intervals. Figure 5.1.7.1.9-2 shows a flowchart of Op-Comin Function 7 - State 1.

In the second state, the user selects a message to view, add a Mode S ID to the search list, or delete a Mode S ID from the search list. Once the user selects a message display option, Op-Comin recognizes the flags which were set in state 1 to execute state 2 of function 7. The 07 FUN ACTIVE flag is set to TRUE if the 07 SEARCH LIST contains any Mode S IDs. If a valid input of 1 is entered, then OPINSTATE is set to 5 to allow state 3 to be processed. Again, OIBUFILL is set to TRUE to allow user response. COMBUF is set to 10 to queue the View A Message buffer. If a valid input of 2 is entered, then OPINSTATE is set to 4 to allow state 4 to be processed, OIBUFILL is set to TRUE, and COMBUF is set to 11 to queue the Add Mode S ID To Search List buffer. If the search list was already full (maximum of 5 IDs), then OPINSTATE is set to 2, OIBUFILL is set to TRUE, and COMBUF is set to 13. This will queue to the control terminal the prompt List Is Full buffer and return to state 2. If a valid input of 3 is entered, then OPINSTATE is set to 6 to allow state 5 to be processed, OIBUFILL is set to TRUE,

and COMBUF is set to 12 to queue the Delete Mode S ID From Search List buffer. If the search list was already empty, then OPINSTATE is set to 2, OIBUFILL is set to TRUE, and COMBUF is set to 17. This will queue to the control terminal the prompt List Is Empty buffer and return to state 2. If an invalid input was entered for the menu response, then OPINSTATE is set to 2, OIBUFILL is set to TRUE, and COMBUF is set to 19. This will queue to the control terminal the prompt Input Not Between 1 and 4 buffer and return to state 2. Figure 5.1.7.1.9-3 shows a flowchart of Op-Comin Function 7 - State 2.

In the third state, the user selects a message to view. The message buffer OMBUF41(94:321) is cleared with blank strings prior to it being filled. If the user enters an <esc> key to escape this function, then COMBUF is set to 9, OPINSTATE is set to 2, and OIBUFILL is set to TRUE. This will return the user to the Display A Message menu in state 2. The user input is checked to see if it is a valid decimal number. If the input is not a decimal number, then COMBUF is set to 20, OPINSTATE is set to 5, and OIBUFILL is set to TRUE. This will queue the Input Not A Decimal Number buffer and return to state 3. The input is then checked to see if it is within decimal values of 1 to 20. If it is not, then COMBUF is set to 21, OPINSTATE is set to 5, and OIBUFILL is set to TRUE. This will queue the Message Number Out Of Boundaries buffer and return to state 3. The input value is used for an array 07 MESS NUM for a pointer to the message being displayed on the status terminal.

If any values within the array 07 MESS NUM is equal to 0, then COMBUF is set to 22, OPINSTATE is set to 5, and OIBUFILL is set to TRUE. This will queue the No Data At This Message Number buffer and return to state 3. If a value within the array 07 MESS NUM is equal to 1, then the requested message location to display (buffer which contains the message displayed on the status terminal) will be computed, the computed message location will be stored in the control terminal display buffer, and the incoming messages are converted to ASCII, hex format. To compute the requested message location (PTR) which contains the data, the input value is subtracted by 1 and multiplied by 73 to obtain the location value of OOBUF09. OOBUF09 is the buffer which contains the message data used for displaying it on the status terminal. A second message location (PTR2) is computed to obtain the message-number string from the Display A Message template buffer. The computation for PTR2 is done by subtracting 1 from the input value, multiplying it by 8, and then adding 196 to the result to obtain the location of the template buffer OOBUF08. OOBUF08 is the template buffer which holds all the headers and empty data string buffers for the Display A Message buffer. The computed location data are then stored in the status terminal buffer OMBUF41 to view the message. The length of the entire message (WLENGTH) is then computed. WLENGTH is computed by obtaining the byte count (07 BYTE), adding 3 to it, and dividing by 4 to obtain the word length. A third message location (MSPTR) is computed to determine where to store the entire message within the OMBUF41 buffer. MSPTR is computed by subtracting 1 from the value of K, if K is less than 9, multiplying that by 8, and adding 94 to the result; where K is the value of WLENGTH. If K is less than 18, then 10 is subtracted from K, multiplied by 8, and 172 is added to the result. If K is less than 27, then 19 is subtracted from K, multiplied by 8, and 250 is added to the result. Prior to the entire message being stored to OMBUF41, the hex integers are all converted to ASCII-hex values. The individual byte is obtained from each 32-bit message data. Each nibble of each byte is added to a hex 30. This will allow to obtain the appropriate ASCII value. If its result is greater than hex 39, then it is added to it 7. This will allow to obtain the appropriate hex values

greater than hex 40. The result is then multiplied by the appropriate byte field within a 32-bit ASCII word. The final result of a 32-bit hex integer is two 32-bit ASCII-hex words. Now that the message has been converted, it is then stored in the control terminal buffer OMBUF41 and queued for display. This is done by setting COMBUF to 41, OPINSTATE to 5, and OIBUFILL to TRUE. Figure 5.1.7.1.9-4 shows a flowchart of Op-Comin Function 7 - State 3.

In the fourth state, the user selects to add a Mode S ID to the search list. If the user enters an <esc> key to escape this function, then COMBUF is set to 9, OPINSTATE is set to 2, and OIBUFILL is set to TRUE. This will return the user to the Display A Message menu in state 2. The user input is checked to see if it is a valid, 6-digit hex number between 0 and F. If the input is not a hex number, then COMBUF is set to 23, OPINSTATE is set to 4, and OIBUFILL is set to TRUE. This will queue the ID Not A Hexadecimal Number buffer and return to state 4. If the input was a valid hex input, then it is converted to a 24-bit integer. The input is converted to an integer by taking each ASCII character of the 6-digit input and multiplying it by the appropriate and corresponding nibble of a 32-bit integer. The 24-bit integer is stored in MODESIN. The value in MODESIN is then compared against the search list (O7 ID) to see if there were any current IDs with the same Mode S ID.

If there were any matching IDs with the existing IDs in O7 ID, then COMBUF is set to 14, OPINSTATE is set to 4, and OIBUFILL is set to TRUE. This will queue ID Already On Search List buffer and return to state 4. The input is also saved in its ASCII format to be stored in MSID1 and MSID2, which is used to display it on the Display A Message screen on the status terminal. MSID1 contains the first 32 bits of the ASCII Mode S ID and MSID2 contains the remainder of the 16 bits. The 24-bit integer ID is then stored in the Search List (O7 ID). The two-part 32-bit ASCII ID is stored in both OMBUF11 and OMBUF12 to display it to the control terminal. After the ID is stored in the O7 ID, the search list counter is incremented by 1 to keep track of where to store the next ID. The user is then returned to the Display A Message menu. Figure 5.1.7.1.9-5 shows a flowchart of Op-Comin Function 7 - State 4.

In the fifth state, the user selects to delete a Mode S ID from the search list. If the user enters an <esc> key to escape this function, then COMBUF is set to 9, OPINSTATE is set to 2, and OIBUFILL is set to TRUE. This will return the user to the Display A Message menu in state 2. The user input is checked to see if it is a valid, 6-digit hex number between 0 and F. If the input is not a hexadecimal number, then COMBUF is set to 23, OPINSTATE is set to 6, and OIBUFILL is set to TRUE. This will queue the ID Not A Hexadecimal Number buffer and return to state 5. If the input was a valid hexadecimal input, then it is converted to a 24-bit integer. The input is converted to an integer by taking each ASCII character of the 6-digit input and multiplying it by the appropriate and corresponding nibble of a 32-bit integer. The 24-bit integer is stored in MODESIN. The value in MODESIN is then compared against the search list (O7 ID) to see if there were any current IDs with the same Mode S ID. If there were no matching IDs with the existing IDs in O7 ID, then COMBUF is set to 18, OPINSTATE is set to 6, and OIBUFILL is set to TRUE. This will queue ID Not On Search List buffer and return to state 5. If the value in MODESIN did match an ID in O7 ID, then that value is removed from O7 ID, OMBUF11, OMBUF12, MSID1, and MSID2. The deletion of the Mode S IDs are done in a sorting manner. If there were five Mode S IDs in the search list and the user ID matched the third ID in the O7 ID, then the fourth ID

will overwrite the third ID, the fifth ID will overwrite the fourth, and the fifth ID gets blanked out. After the ID is deleted from all of the ID parameters, the search list counter is decremented by 1 to keep track of where to store the next ID. The user is then returned to the Display A Message menu. Figure 5.1.7.1.9-6 shows a flowchart of Op-Comin Function 7 - State 5.

5.1.7.1.10 Function 8 - Toggle Data Extraction.

Function 8, toggle data extraction subroutine, will toggle the extraction of data during the CID simulation. The current data extraction setting is displayed on the display screen of the system statistics function. To reinforce the toggling of extraction, the System status function is invoked to display the data extraction status flag.

When extraction is currently enabled (ST EXT FLAG is greater than 0), it becomes disabled by setting ST EXT FLAG to -1. An extractor termination message is included by writing the TOY on the end of the extractor tape. Also, the last extraction buffer is queued by calling extract.

When the system status function is disabled (ST EXT FLAG is less than 0), it becomes enabled by setting ST EXT FLAG to 1 when the tape drive is found to be loaded and on-line by the tape drive status routine. When the tape drive is found off-line, the request is ignored. Op-Comm is scheduled to run 1 second from the current system time and every 4 seconds, thereafter, by setting ITIME(3) to the current system time, plus 1000 STUs and setting INCTIME(3) to the value of the schedule update frequency variable which is 4 seconds. COOBUF is set to 4 to queue the label buffer of the display system errors function, COMBUF is set to 1 to queue the menu buffer for display, an input buffer is opened allowing a response to the menu, and finally OPINSTATE is set to 0 signifying the completion of the Op-Comin processing of this function. Figure 5.1.7.1.10-1 shows a flowchart of Op-Comin Function 8.

5.1.7.1.11 Function 9 - Terminate the Scenario.

Function 9, the terminate scenario subroutine, brings the CID simulation to a clean stop when the user responds with a "Y" to the "OK to terminate (Y/N) ??" query displayed on the control terminal. Hardware devices are reset by calling the Termhard subroutine, normal Fortran I/O is enabled by calling the Fortran subroutine Carcon, and a Fortran write is used to display a message on the control terminal to return the switch back to the system console position. Figure 5.1.7.1.11-1 shows a flowchart of Op-Comin Function 9.

5.1.7.2 Op-Comm.

Op-Comm's primary responsibility is to manipulate the CID display screen contents. Other functions of Op-Comm are to perform statistical calculations, update data bases, set up for the display of statistical screen data fields, and set up pointers for the next routine to be processed in the Op-Comin routine.

When manipulating data, it is necessary to access and modify values shown on the display screens. A discussion of the display screen databases and the manipulation of these databases is contained in section 5.1.7.3 (Opblkdta). All computations on the statistics are performed in integer mode. The necessary ASCII conversions to display statistical results on the display terminal are explained in detail in section 5.1.7.3.1.

Op-Comm is broken down into 10 functions. These functions correspond to the functions shown in the alogorithm's flowchart in figure 5.1.7.2-1. An optimized subroutine call is used to vector to one of these functions depending on the value passed to it from the Op-Comin routine. The optimized subroutine call is identical to the algorithm used for Op-Comin as detailed in section 5.1.7.1.

On entering Op-Comm, a value of 0 through 9 is expected in the variable OPERFUNC. One of the ten OPCOMM function routines will be performed based on the value contained in this variable.

5.1.7.2.1 Output Processing.

Due to time constraints placed on Op-Comm by the CID system design, standard I/O processing (i.e., Fortran reads and writes) are inadequate for use in communicating with the CID real-time system. In preparing output for display, Op-Comm utilizes specific algorithms to convert the internal variables computed by Op-Comm to the ASCII format necessary for output on the display screen of the CID system. Section 5.1.7.3 contains a discussion of the makeup of the display screen, the associated buffers, and how the individual variables are positioned within a buffer created for display.

The normal output processing routine used in Op-Comm converts either a halfword or fullword integer value and converts it to an 8-byte (four character) ASCII. A common piece of Fortran code, called TOASCII, is included with Op-Comm through the use of the Fortran compiler's $INCLUDE directive to perform this conversion where required. The algorithm to perform this conversion follows:

a. When INT NUMBER, the value passed to the algorithm, is less than 10, AR NUMBER is computed to be,

```
AR NUMBER = INT NUMBER + Y'20202030'.
```

The above conversion merely adds a hex 30 to the single digit value held in INT NUMBER. The remaining three characters to the left of the converted value are ensured to be cleared of any prior information by inserting the ASCII code for a blank at those locations.

b. Otherwise, when INT NUMBER is less than 100, AR NUMBER is computed as:

```
AR NUMBER = (INT NUMBER/10) * X'100' +
            (INT NUMBER - INT NUMBER / 10 * 10) +
            Y'20203030'.
```

The above conversion spreads the integer of INT NUMBER across 4 storage bytes and adds a hex 20203030 to the resultant value to convert the 4 right-hand bytes to ASCII and ensure that the left-hand bytes are cleared of any previous information.

c. Otherwise, when INT NUMBER is less than 1000, AR NUMBER is computed as:

```
AR NUMBER = (INT NUMBER/100) * X'10000' +
            (INT NUMBER - INT NUMBER/100*100)/10*X'100'
            (INT NUMBER - INT NUMBER / 10 * 10) +
            Y'20303030'.
```

The above conversion spreads the integer of INT NUMBER across 6 storage bytes and adds a hex 20303030 to the resultant value to convert the 6 right-hand bytes to ASCII and ensure that the left-hand byte is cleared of any previous information.

d. Otherwise, when INT NUMBER is less than 10000, AR NUMBER is computed as:

```
AR NUMBER = (INT NUMBER/1000) * X'1000000' +
(INT NUMBER - INT NUMBER/1000*1000)/100*X'10000'+
(INT NUMBER - INT NUMBER/100*100)/10*X'100'
(INT NUMBER - INT NUMBER / 10 * 10) +
Y'30303030'.
```

The above conversion spreads the integer of INT NUMBER across 8 storage bytes and adds a hex 30303030 to the resultant value to convert the value to ASCII.

e. Otherwise, when INT NUMBER is greater than or equal to 10000, AR NUMBER is computed as:

```
AR NUMBER = Y'2A2A2A2A'.
```

This places the string **** in the storage location used for the particular variable to signify that an overflow has occurred and the algorithm is unable to convert this value to ASCII in the allotted storage space. Figure 5.1.7.2.1-1 is a flowchart of the ASCII algorithm.

A similar output conversion algorithm is used to convert the IPCNT (the percent CPU utilization) variable to ASCII in Function 1 of OPCOMM. IPCNT is computed to the tenths digit. This requires the placement of a decimal point in the ASCII result. The algorithm below explains this process:

a. When IPCNT is less than 100, AR IPCNT is computed as:

```
AR IPCNT = IPCNT/10*Y'10000'+
           (IPCNT-IPCNT/10*10)+
           Y'20302E30'
```

The above conversion spreads the decimal of IPCNT across 6 bytes while placing a "." represented by hex 2E, between the tens and tenths digits.

b. Otherwise, when IPCNT is less than 1000, AR IPCNT is computed as:

```
AR IPCNT = IPCNT/100*Y'1000000'+
           (IPCNT-IPCNT/100*100)+
           (IPCNT-IPCNT/10*10)+
           Y'30302E30'
```

The above conversion spreads the decimal of IPCNT across 8 bytes while placing a "." represented by hex 2E, between the tens and tenths digits.

c. Since the percent CPU utilization cannot exceed 99.9 percent, processing is not performed on IPCNT when it is greater than or equal to 1000. The overflow case algorithm is computed to be,

AR NUMBER = Y'2A2A2A2A'.

This places the string '****' in the storage location used for the AR IPCNT to signify that an overflow has occurred and the algorithm is unable to convert this value to ASCII in the allotted storage space. Figure 5.1.7.2.1-2 is a flowchart of the decimal to ASCII algorithm.

To convert the Time Into the Simulation (TIS) parameter into display format, a special algorithm is used to break down the ST SYSTEM CLOCK variable into minute (TIS MIN) and second (TIS SEC) units. The following algorithm is used:

a. STUs are converted to milliseconds by the equation,

MIN SEC = ST SYSTEM CLOCK/976,

b. When MIN SEC is less than 600, i.e., less than 10 minutes, the following equation is used to strip the "minute" value off of the MIN SEC variable and assigns the result to TIS MIN value, which is part of the display buffer being prepared for display.

TIS MIN = MIN SEC/60 + X'2030',

c. Otherwise, MIN SEC is assumed to be less than the value 9959, which is the largest value allowed for this equation. In this case, the following equation is used to determine the "minute" value.

TIS MIN = MIN SEC/600*X'100'+ (MIN SEC/60-MIN SEC/600*10) +
X'3030',

d. In any case, the seconds value, TIS SEC, is computed by the following equation. In the output buffer where this ASCII converted information is stored, a colon has been prepositioned between the variables TIS SEC and TIS MIN to generate the time into simulation running time display. Figure 5.1.7.2.1-3 is a flowchart of the algorithm.

TIS SEC = (MIN SEC - MIN SEC/60*60)/10 *X'100' +
(MIN SEC - MIN SEC/10*10) + X'3030'.

5.1.7.2.2 Function 0 - Output Menu.

Function 0 of Op-Comm is not utilized by the Op-Comm package.

5.1.7.2.3 Function 1 - Display Statistics.

Function 1, the system statistics subroutine, is responsible for computing the current system statistics and converting them to ASCII for output. The system statistics processed in this subroutine can be found in the CID user's manual.

The system statistics subroutine is scheduled to be run in 4-second increments by Op-Comin. It is run repetitively until another function is selected, or is disabled by reselecting Function 1.

On entering Op-Comm Function 1 processing, the percent CPU used is computed. IPCNT, the percent CPU used, is computed as 10 times its actual value. This is done to obtain IPCNT in one-tenths units. The formula for computing IPCNT in tenths of a millisecond is:

IPCNT = (ST PROCESS TIME) / (ST SYSTEM CLOCK - ST OLD SYSTEM CLOCK),

where ST PROCESS TIME is the CPU time used since the last update and (ST SYSTEM CLOCK - ST OLD SYSTEM CLOCK) is the total time since the last update. To convert this value to tenths of STUs:

IPCNT = IPCNT * 125 / 128,

where the factor (125/128) yields the conversion from milliseconds to seconds without overflowing the fullword integer storage location used for the IPCNT variable. The results are then converted from fullword integer to ASCII. The final step before moving on to the next statistic is to reset the computation parameters for the next time through the routine. This requires that ST PROCESS TIME be set to 0, and that ST OLD SYSTEM CLOCK is set to ST SYSTEM CLOCK, effectively resetting the stopwatch.

The percentage data extraction utilized, PCNT DEX, is computed next. Since the data extraction buffers work on a circular queue principle (section 5.1.10), a check must be made to verify when the buffer being used has wrapped past the maximum buffer value. This is done by checking if OLD DEX BUF, the value in which the buffer used the last time through this routine was saved, is less than CEXTBUFF, the current extraction buffer. When this is true, PCNT DEX is computed to be:

PCNT DEX = CEXTBUFF - OLD DEX BUF

When the check is not true, PCNT DEX is computed to be:

PCNT DEX = 100 - (OLD DEX BUF - CEXTBUFF)

OLD DEX BUF is then assigned the current value of CEXTBUFF for the next time through the routine. As a final step before conversion to ASCII, PCNT DEX is multiplied by 2 to yield the actual percentage extraction utilized value. The results are then converted from fullword integer to ASCII.

To tally the total number of system errors, TOTAL ERRORS is computed by summing the array ST ERROR from ST ERROR(1) to ST ERROR(30). The result is then converted from fullword integer to ASCII.

To tally the total number of X.25 errors, TOTAL X25 ERRORS is computed by summing the five error fields in each of the Task Common areas, CD0XXXXX to CDBXXXXX. The result is then converted from fullword integer to ASCII.

The next value processed is the number of scenario buffers processed. This value, ST SCEN NUM, is directly converted to ASCII by using the TOASCII in-line routine.

The next status value processed is that of the current data extraction processing status. This is found by checking the flag ST EXT FLAG. When ST EXT FLAG is 0, data extraction processing is currently disabled and the statistics display screen is directly modified by setting the 4 bytes starting at byte 69 of the statistics display screen, OOBUF02(69:72), to the character string "DISA." When ST EXT FLAG is not set to 0, data extraction processing is enabled and the statistics display screen is modified to reflect this by setting the 4 bytes starting at byte 69 of the statistics display screen, OOBUF02(69:72), to the character string "ENA." A complete explanation of how the statistics display screen is organized and manipulated is included in section 5.1.7.3.1. (Opblkdta)

The next 17 values processed are ST COUNT1 through ST COUNTG. They are in fullword integer format and need only be converted to ASCII.

The time into simulation is computed by simply dividing ST SYSTEM CLOCK by 976. The value 976 is used to convert from STUs to seconds. This result, stored in MINSEC is the time into the simulation in seconds. The conversion of this value into the minute and second format is based on two considerations. First, if the value of MINSEC is less than 600, there is no need to handle the tens digit of the result. Second, if the MINSEC value is greater than 600, the extra digit in the minutes field is computed and eventually displayed.

The final step of this function is to queue the statistics buffer for display by setting COMBUF to 2.

5.1.7.2.4 Function 2 - Display System Errors.

On entering function 2, the TOTAL ERRORS counter is set to 0.

The following errors are converted to ASCII in function 2:

a. Data extraction I/O (ST ERROR(1)).
b. Message Scenario I/O (ST ERROR(2)).
c. Op-Comm I/O (ST ERROR(4)).
d. Data extraction data lost (ST ERROR(5)).
e. Scenario buffer overwrite (ST ERROR(8)).
f. Scenario buffer empty (ST ERROR(9)).
g. Bad port number (ST ERROR(10)).
h. Savelist empty (ST ERROR(12)).
i. Savelist full (ST ERROR(13)).
j. Illegal X.25 device (ST ERROR(11)).
k. X.25 packets (ST ERROR(16)).
l. Link not active (ST ERROR(17)).

For each of the above errors, the error value is accumulated in TOTAL ERRORS to tally the total number of system errors. TOTAL ERRORS is converted to ASCII after the 12 individual errors are processed. The total number of X.25 errors is tallied and converted by summing the eight individual errors for each of the 12 task common areas, CD0 SOFT ERRORS through CDB SOFT ERRORS.

Both system time and time into simulation are converted to ASCII as described in section 5.1.7.2.3

The final step of this function is to queue the system error buffer for display by setting COMBUF to 4.

5.1.7.2.5 Function 3 - Display X.25 Status.

For each of the 12 (hex 0 through hex B, denoted by "x" in the following variable names) X.25 devices included in the CID, it is necessary to process the link status and device status flags when a X.25 device is configured.

When ST X25 CONFIGURED(x) is greater than 0, the device is configured. When configured, CDx ACTIVE FLAG is checked. When the active flag is TRUE, the appropriate field in the output buffer (OOBUF06) is set to "A." When the active flag is FALSE, the OOBUF06 field is set to "I." Next, CDx LINK CONNECT is checked to determine when the link status flag is TRUE. When the flag is TRUE, the link status field for the device being processed in OOBUF06 is set to "C." When the link connect flag is FALSE, the link status field for the device is set to "D."

When ST X25 CONFIGURED(x) is less than or equal to 0, the device is not configured and "N" is placed in the device status field of OOBUF06.

After the status fields of each device is modified, the X.25 statistics must be updated. The statistics included in this function are input bytes per second, input messages per second, output bytes per second, and output messages per second. Statistics are computed only for configured devices.

To compute input bytes per second, a measure of the number of input bytes, ST X25 STAT(x,1), is divided by the number of seconds since the last update (ST SYSTEM CLOCK - ST OLD SYSTEM CLOCK all divided by 977 to change from STUs to seconds).

The computed value is then converted to ASCII, but only if the result of the input bytes per second computation is not 0. When the result is 0, the string "< 1" is placed in the input bytes per second field of OOBUF06. A "0" is placed in OOBUF06 if the original ST X25 STAT(x,1) was 0 on entering this function.

Input messages per second, output bytes per second, and output messages per second are each computed using the identical algorithm, but different variables to compute and convert their results.

Once the X.25 statistics are computed, ST SYSTEM CLOCK is stored in ST OLD SYSTEM CLOCK to reset the stop watch so that the interval between executions of function 3 can be properly timed.

Both system time and time into simulation are converted to ASCII as described in section 5.1.7.2.3.

The final step of this function is to queue the X.25 statistics buffer for display by setting COMBUF to 6.

5.1.7.2.6 Function 4 - Display X.25 Errors.

The processing routine of the X.25 errors is identical to that of section 5.1.7.2.5, except that the four X.25 statistic columns (input bytes per second, etc.) are replaced by six columns of X.25 errors.

The number of messages lost (ST MSG LOST(x)) is an array held in the system table. These values are stored in the appropriate task common area variable (CDx SOFT ERRORS(4)) at the beginning of the X.25 error routine to simplify the following conversion routine.

For each of the 12 (hex 0 through hex B, denoted by "x" in the following variable names) X.25 devices included in the CID, it is necessary to process the X.25 device errors only when the device is configured.

When a device is configured, errors CDx SOFT ERRORS(1) through CDx SOFT ERRORS(5) are computed, converted to ASCII, and finally stored in the proper field of OOBUF07. The sum of errors CDx SOFT ERRORS(2) through CDx SOFT ERRORS(5) is taken as a step in computing the total X.25 errors. A sixth error is computed by summing CDx SOFT ERRORS(6) through CDx SOFT ERRORS(10). This sum is converted to ASCII, stored in the proper field of OOBUF07, and also added to the total X25 error count.

Once each of the configured X.25 devices are processed as stated above, the running total of X.25 errors is converted to ASCII and stored in the proper field of OOBUF07.

Both system time and time into simulation are converted to ASCII as described in section 5.1.7.2.3.

The final step of this function is to queue the X.25 error buffer for display by setting COMBUF to 7.

5.1.7.2.7 Function 5 - Modify X.25 Status.

Function 5 of Op-Comm is not utilized by the Op-Comm package.

5.1.7.2.8 Function 6 - Creat a Message.

Function 6 of Op-Comm is not utilized by the Op-Comm package.

5.1.7.2.9 Function 7 - Display a Message.

The Display A Message Periodic subroutine, function 7 of the CID main menu, will handle all input messages and convert them to ASCII prior to updating the individual messages to the Display A Message List screen. This routine will process at every 4-second intervals, which will update the screen during this time. Figure 5.1.7.2.9-1 shows a flowchart of Op-Comm Function 7 - subroutine.

A check is made to determine if the input and output message pointer (07 IN PTR and 07 OUT PTR), of the common storage area, have a different value. 07 IN PTR is the pointer to keep track of all input messages. 07 OUT PTR is the pointer to keep track of all messages which were output to the Display A Message List screen. If the two pointers are of equal value, the screen is only updated with the current messages stored in COOBUF9 buffer. COOBUF9 stores the Display A Message List data area. If the two message pointers did not have the same value, then there would exist another message to be processed before displaying it to the screen. Prior to processing the message, various parameters have to be obtained from the common storage area. The message (07 MESSAGE), type code (07 TYPE), port number (07 PORT), and Mode S ID (07 MSID) all have to be obtained from the common storage area.

The type code is checked against a series of message types (hex 41, 42, 44, 45, 31, and 32). If the type code matches with hex 41, the type code (CTYPE) is set to an ASCII value of "434F4D42" to obtain the string "COMB." The message number (REF) is obtained from the first byte of the second 32-bit word in O7 MESSAGE, which is the actual message. It is then converted to ASCII by use of the in-line Fortran TOASCII code. The status field (MISC) is obtained from the second byte of the second 32-bit word in O7 MESSAGE. It is converted to ASCII by use of the in-line Fortran TOASCII code. If the type code matches with hex 42, CTYPE is set to an ASCII value of "20454C4D" to obtain the string "ELM." REF is obtained from the first byte of the second 32-bit word in O7 MESSAGE, which is the actual message. It is then converted to ASCII by use of the in-line Fortran TOASCII code. The status field (MISC) is obtained from the second byte of the second 32-bit word in O7 MESSAGE. It is converted to ASCII by use of the in-line Fortran TOASCII code. If the type code matches with hex 44, CTYPE is set to an ASCII value of "20434150" to obtain the string "CAP." REF is obtained from the first byte of the second 32-bit word in O7 MESSAGE, which is the actual message. It is then converted to ASCII by use of the in-line Fortran TOASCII code. The status field (MISC) is obtained from the second byte of the second 32-bit word in O7 MESSAGE. It is converted to ascii by use of the in-line Fortran TOASCII code. If the type code matches with hex 45, CTYPE is set to an ASCII value of "4303936" to obtain the string "CAP." REF is obtained from the first byte of the second 32-bit word in O7 MESSAGE, which is the actual message. It is then converted to ASCII by use of the in-line Fortran TOASCII code. The status field (MISC) is obtained from the second nibble, of the second byte, of the second 32-bit word, and the third byte of the 32-bit word in O7 MESSAGE. It is converted to ASCII by use of the in-line Fortran TOASCII code. If the type code matches with hex 31, CTYPE is set to an ASCII value of "2052454A" to obtain the string "REJ." REF is set to an ASCII value of "20202020," since there is no message number in this type of message.

The status field (MISC) is obtained from the second byte, of the second 32-bit word, in O7 MESSAGE. MISC is then checked for various status values. If MISC has a value of hex 80, AMISC is set to the ASCII string "NO ELM." If MISC has a value of hex 60, AMISC is set to the ASCII string "NOT PR." If MISC has a value of hex 20, AMISC is set to the ASCII string "NOT RC." If MISC has a value of 0, AMISC is set to the ASCII string "NO TGT." If the type code matches with hex 32, CTYPE is set to an ASCII value of "2044454C" to obtain the string "DEL." REF is set to an ASCII value of "20202020," since there is no message number in this type of message. The status field (MISC) is obtained from the second byte, of the second 32-bit word, in O7 MESSAGE. MISC is then checked for various status values. If MISC has a value of hex 80, AMISC is set to the ASCII string "EXP." If MISC has a value of 0, AMISC is set to the ASCII string "DEL."

Once the port number (O7 PORT) is obtained, depending on its value, the ASCII equivalence is stored in CPORT for the display buffer. If the port number was a 1, then the ASCII eqivalence would be "31."

The Mode S ID in O7 MSID is compared with the user ID from the Search List (O7 ID). Once the ID is matched, its ASCII equivalence in AMSID1 and AMSID2 is assigned to AMODES1 and AMODES2 for the display buffer.

The TOY is obtained by masking out the required bits for the minutes and seconds from ST TOY1. The value is then stored in TOYS MIN and TOYS SEC.

Two message pointers (PTR and PTR2) are computed for the current and previous message buffers, respectively. The current message pointer (PTR) is computed by subtracting 1 from the output message pointer (07 OUT PTR) and multiplying its result with 73, to store the message in the appropriate buffer location OOBUF09. The previous message pointer (PTR2) is computed by subtracting 2 from the output message pointer (07 OUT PTR) and multiplying its result with 73, to clear the message pointer in the appropriate buffer location OOBUF09.

When the message is stored in the OOBUF09 buffer, a flag is set to tell OPINFUN7 that a message is present at that location for View A Message option. This is done by setting 07 MESS NUM to 1. The ouput message pointer is also incremented by 1 to keep step with the input message pointer 07 IN PTR.

The screen is only supposed to display 20 lines of individual messages, therefore, when 07 OUT PTR reaches a value greater than 20, it is reset back to 1.

Finally, the Display A Message List screen is updated with the new message line by setting COOBUF equal to 9.

5.1.7.2.10 Function 8 - Data Extraction Toggle.

Function 8 of Op-Comm is not utilized by the Op-Comm package.

5.1.7.2.11 Function 9 - Stop The CID Simulation.

Function 9 of Op-Comm is not utilized by the Op-Comm package.

5.1.7.3 Opblkdta Routine.

A separate program unit, called a block data, is allowed in Fortran to initialize data values of arrays and variables appearing in common blocks. The Op-Comm package utilizes this program unit to create a database of display screens, menu screens, prompts and messages to communicate with the CID user. This block data, called OPBLKDTA, contains all the necessary information to provide the user with full communication ability within the CID.

The Op-Comm package is designed to minimize the amount of CPU utilization. The CID software design aids in this effort by removing much of the time consumptive work which must be performed. The three cases where this is most apparent is in task scheduling, data extraction, and I/O. The I/O function, which this section covers requires that any I/O that is necessary during the simulation must be performed by the I/O Manager. Required information about a buffer to be output must come in the form of an output request by the routine requiring the transaction. For Op-Comm and Op-Comin, an output request is queued for execution by setting the COOBUF or COMBUF common variable to the index of the buffer to be output. This queues the output request for eventual execution by I/O. To the CID operator, the eventual execution takes place seemingly without delay.

COOBUF serves as a pointer to a series of buffers which are eligible for display on the CID Display Terminal. COOBUF can take on the values 1 through 8 and serves as an index to the buffers OOBUF01 through OOBUF08. These buffers inform the user of the status of the system in response to a menu choice.

COMBUF serves as a pointer to a series of buffers which are eligible for display on the CID Menu Terminal. COMBUF can take on the values from 1 to 14 and serves as an index to the buffers OMBUF01 through OMBUF14. These buffers consist of information to formulate the necessary menu screens, error messages, and prompts which the user may request or require.

5.1.7.3.1 CID Display Screen Buffers.

Figure 5.1.7.3.1-1 lists each of the Display Screen buffers defined in the OPBLKDTA subprogram. OOBUF01 through OOBUF07 are the buffers which form the displays of statistics required for several of the CID menu functions. The display process is broken into two parts to save processing time. A template buffer is used for the display screens to provide a base of information on the screen which is overlayed with the computed and converted (to ASCII) data buffer. The template buffer is displayed once per function request. The data buffer is updated repetitively on the screen at defined time intervals until the function is reselected or another function is requested. By doing this, processing time is saved because less time is spent displaying unnecessary information. Another reason this is done, is to provide a time efficient method of processing data. Each of the status data buffers use Fortran equivalence statements to allow an array to have more than one variable name, and variable type, associated with a particular storage location. This allows arithmetic calculations to be performed on data and allows the computed result to be converted immediately to ASCII format. This precludes the use of the standard Fortran write statement to display information on the status screen, which has proven to be too slow for CID's requirements.

A template buffer is comprised of a series of FORTRAN DATA statements which include headings, text and cursor positioning information to properly position the information on the display screen. The cursor positioning information is a 6-byte hex field which controls positioning in the horizontal and vertical directions. The ASCII for the sequence is, <esc> X ??? <esc> Y ???, where the question marks determine the line and column number which enables the cursor to be positioned anywhere on the screen.

The headings and text which are contained in a template buffer are the actual character strings to be displayed. By merging cursor positioning information with each of the character strings to be displayed, a template buffer is formed. When defining the buffer, the data statement is used to initialize the headings, text, and cursor positioning information. Each of the template buffers is defined as a single array of a length long enough to contain all of the information necessary. Fortran requires that a particular DATA statement cannot be longer than 135 characters. This requires that the template buffers, along with all other types of buffers, be broken into a series of manageable data statements. The management of a template buffer thus becomes a series of DATA statements, which define an exact number of characters in the character string, with each DATA statement containing either cursor positioning information, header definitions or text. Figure 5.1.7.3.1-2 taken from OOBUF01, the System Status Template, is a brief example of a template buffer. There are several things to notice about the template. In line 1, the 2 numerical values separated by the colon represent the array within the OOBUF01 character array which the value within the slashes (/) represent. The character Z which precedes the string of numbers 1B48000000000000, informs the FORTRAN compiler that the following string of numbers is in hex. The

number string itself represents the 8-byte cursor positioning command to clear the display screen of all information and return the cursor to the home position (which is the upper left-hand corner of the screen). The 12 zeros which follow the command "1B48" represent a delay necessary to allow the display screen to complete the intended operation.

In line 2, the array substring from byte 9 through 14 contains the cursor positioning information to position the cursor at line 4 and column 23 of the display screen. In line 3, the array substring from byte 15 through 51 is defined as the header label for the system status function of the CID simulation.

The remaining lines of the example above follow the format of lines 1 through 3. For further information on cursor positioning, refer to the PE Model 550 Maintenance Manual.

The primary functions of the status data buffers are to provide storage locations for the converted status information computed in Op-Comm and provide cursor positioning information for the proper display of this information on the display screen. The format of the status data buffers is the same as for the template buffers described above. Care has been taken in designing the status data buffers to keep the storage locations for the data on halfword or fullword boundaries and to space the data evenly for simple two-dimensional array access.

The following text describes each Display Screen buffer.

OOBUF01, the System Status template buffer, is queued for display in Op-Comin Function 1. The buffer is 1076 bytes long.

OOBUF02, the System Status data buffer, is queued for display in Op-Comm Function 1. The buffer is 341 bytes long.

OOBUF03, the System Error template buffer, is queued for display in Op-Comin Function 2. The buffer is 805 bytes long.

OOBUF04, the System Error data buffer, is queued for display in Op-Comm Function 2. The buffer is 247 bytes long.

OOBUF05, the X.25 Statistics and Error template buffer, is queued for display in Op-Comin function 3 and Op-Comin function 4. The buffer is 565 bytes long. The buffer contains information which is common to both display screens.

OOBUF06, the X.25 Statistics data buffer, is queued for display in the Op-Comm Function 3. The buffer is 1128 bytes long.

OOBUF07, the X.25 Error data buffer, is queued for display in Op-Comm Function 4. The buffer is 1495 bytes long.

OOBUF08, the CID Display Message List buffer, is queued for display in Op-Comin Function 7. The buffer is 1536 bytes long. It contains the CID message list template for 20 lines of display.

OOBUF09, the CID Display Message Data buffer, is queued for display in Op-Comm Function 7. The buffer is 1536 bytes long. It contains the buffer area to store the message pointer, time message was input to CID, message type, message number, port number, and the message status.

5.1.7.3.2 CID Menu Screen Buffers.

Figure 5.1.7.3.2-1 lists each of the CID menu screen buffers as found in the OPBLKDTA subprogram. OMBUF01 through OMBUF08 are buffers which form the displays of menus, prompts, and queries utilized by the Op-Comm package and displayed on the menu screen. Since the display of these buffers is on a one-to-one relationship with an event (i.e., a function request on the CID menu, or an illegal input in response to a menu or a query), the display of the Menu Screen buffers simply requires that the buffer be queued for output by setting the COMBUF variable to the appropriate buffer index. I/O handles the output procedure from that point.

The following text describes each Menu Screen buffer:

OMBUF01, the CID Menu Screen buffer, is queued for display upon both normal and abnormal termination of a user requested function throughout the CID simulation. Basically, the CID menu screen provides the user with all the options for controlling the simulation. The buffer is 520 bytes long.

OMBUF02, the CID termination verification query, is queued for display in Op-Comin Function 9. The buffer is 46 bytes long.

OMBUF03, the Modify X.25 Device main menu, is queued for display in Op-Comin Function 5 State 1. The buffer is 211 bytes long.

OMBUF04, the Modify X.25 Device error query, is queued for display in Op-Comin Function 5 State 2 when the user enters a device number less than 0 or greater than 11. The buffer is 66 bytes long.

OMBUF05, the Modify X.25 Device not configured error buffer, is queued for display in Op-Comin Function 5 State 2. The buffer is displayed when the user types a device number which is not configured in this simulation. The buffer is 66 bytes long.

OMBUF06, the Modify X.25 Device deactivate buffer, is queued for display in Op-Comin Function 5 State 2. The buffer is displayed when the user types a device number which is currently active. The buffer is 63 bytes long.

OMBUF07, the Modify X.25 Device activate buffer, is queued for display in Op-Comin Function 5 State 2. The buffer is displayed when the user types a device number which is currently inactive. The buffer is 63 bytes long.

OMBUF08, the Input Error Reprompt buffer, is queued for display in the Op-Comin main routine. The buffer is displayed when the user types a value other than "0" through "9" in response to the CID main menu. The buffer is 42 bytes long.

OMBUF09, CID Display Message List Menu buffer, is queued for display in Op-Comin Function 7. The buffer is 512 bytes long. It contains the options for controlling the display of various input messages.

OMBUF10, View A Message option buffer, is queued for display in Op-Comin Function 7. The buffer is 256 bytes long. It contains the option to enter the desired message number to view.

OMBUF11, IDs In Search List and Add ID To Search List buffer, is queued for display in Op-Comin Function 7. The buffer is 256 bytes long. It contains the option to add an ID to the search list.

OMBUF12, IDs In Search List and Delete ID From Search List buffer, is queued for display in Op-Comin Function 7. The buffer is 256 bytes long. It contains the option to delete an ID from the search list.

OMBUF13, List Is Full buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies the user when the search list is full.

OMBUF14, ID Already Exist buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies the user that the ID entered already existed.

OMBUF15, Invalid Input buffer, is queued for display in Op-Comin Function 6. It is 128 bytes long. It notifies that the input was not valid.

OMBUF17, Search List Is Empty buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies that there are no more IDs in the search list.

OMBUF18, Nonexising ID In Search List buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies that the ID entered does not exist on the search list.

OMBUF19, Input Must Be 1 and 4 buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies that the menu selection was not between 1 and 4.

OMBUF20, Input Must Be Decimal buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies that the input must be a decimal value.

OMBUF21, Input Must Be 1 - 20 buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies that the input must be between 1 and 20.

OMBUF22, No Message To View At This Message Number buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies that there is no message to view at the desired message number.

OMBUF23, ID Must Be In Hex buffer, is queued for display in Op-Comin Function 7. It is 128 bytes long. It notifies that the ID is not in hex when adding to or deleting from the search list.

OMBUF24, CID Create A Message buffer, is queued for display in Op-Comin Function 6. It is 512 bytes long. It gives the option to create various types of messages.

OMBUF25, Assigned Message Number buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to assign a message number to the message being created.

OMBUF26, X.25 Port Number buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to assign a port number to the message being created.

OMBUF27, Mode S Address buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to assign a Mode S ID to the message being created.

OMBUF28, Priority buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to assign a priority field to the message being created.

OMBUF29, Expiration buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to assign an expiration field to the message being created.

OMBUF30, Acknowledgement buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to assign an acknowledgement field to the message being created.

OMBUF31, Segment Count buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to assign a segment count to the message being created.

OMBUF32, COMM-A Message Text buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to enter a COMM-A text field to the message being created.

OMBUF33, No Of Segments To Send buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to select the number of segments of the text field to send for the message being created.

OMBUF34, ELM Default Text Block buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to select the elm default text block to send for the message being created.

OMBUF34, ELM Default Text Block buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to select the elm default text block to send for the message being created.

OMBUF35, BDS1 and BDS2 buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to enter the BDS field for the message being created.

OMBUF36, Reference Message Number buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to enter the reference message number for the message being created.

OMBUF37, Reference Type Code buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to enter the reference type code for the message being created.

OMBUF38, Message Length In Bytes buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to enter the message length for the message being created.

OMBUF39, Message Bit Stream buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to enter the message bit stream for the message being created.

OMBUF40, OK To Send Message buffer, is queued for display in Op-Comin Function 6. It is 256 bytes long. It allows the user to enter select if it is OK to send the message for the message being created.

OMBUF41, View A Message buffer, is queued for display in Op-Comin Function 7. It is 512 bytes long. It contains the entire context of the message in which the user selected to view.

5.1.8 I/O Management.

I/O is an Assembly Language Program that processes the I/O requests from other modules in the system. A flowchart for I/O is given in figure 5.1.8.1-1. Requests are performed for the input of fruit and target scenario records, I/O of Op-Comm records, and output of extraction records. These I/O records are buffered in common areas along with their control buffer variables. The output buffers are filled by their respective tasks and the control variables are set to request I/O to send output. The input buffers are filled by an I/O request and the control variables are set to notify the pertinent task of completed input. The following itemizes the unit assignment and primary task user:

Logical Unit	Function	Task
1	Message Scenario Input	OUTMESS
2	Data Extraction	Extraction
3	Statistics Terminal	Op-Comm
5	System Console I/O	Initial
6	CONTROL Terminal	Op-Comin

The Message Scenario and Data Extraction buffers are configured in a set of circular used buffers. I/O attempts to maintain a full set of input buffers and an empty set of output buffers in each of these situations. The terminal buffers are individual buffers and are controlled by a pointer containing the buffer to be used for I/O. In every case only one I/O is executed in each call to I/O. The reason for the single I/O per call is that each I/O requires about 1 ms to process. Multiple calls for I/O would monopolize the processor for long periods of time not allowing other processes access time. The selection of the buffer to be processed is through the following prioritized system.

a. Extraction Output
b. Message Scenario Input
c. Operator Control Terminal Output
d. Operator Control Terminal Input
e. Operator Statistics Terminal Output

First, I/O will process an extraction buffer for output if one is requested and there are less than two in the current output queue. Each extraction buffer requires about 100 ms between the initial I/O and the final interrupt from the drive. Therefore, it is not necessary to queue more than two records before other I/O's can be accomplished. There exist 100 extraction buffers in order to permit the extraction of data without overloading the internal buffer structures.

An I/O request for message scenario input will be processed next. Ten input buffers have been assigned for the message scenario. Each of the scenario input buffers a large number of messages. In the capacity situation, the internal buffers are designed to hold more than one complete scan of messages.

Operator terminal I/O is then performed. The output buffer is processed on the first pass of I/O and the input buffer is processed. Statistics terminal output is performed last.

Once I/O has performed an I/O request, it processes the I/O complete flags for all possible pending I/O functions. This is accomplished by surveying the supervisor call for each pending I/O. A special device code (01) is inserted in the processor control block for each I/O operation. A completed I/O will contain a device code zero if no error occurred or a device code other than (01) in the case of a failure. If an I/O function for any device is complete, the buffer is released. If an error occurs, the appropriate error counter is incremented and the buffer is flagged with the error.

I/O is a scheduled task under the Scheduler. This task is allowed periodic access to the processor every 10 ms. Since only a single I/O request is processed and all pending I/O flags are processed, the maximum execution time is about 1 ms. The number of I/O requests per scan in a 700-target capacity situation is:

Device	I/O Per Scan	Time For 1 I/O
Extraction	50	90 ms
Message Scenario	10	400 ms
Control Terminal	2	4 sec
Statistics	1	4 sec

Since the I/O priority algorithm permits extraction to queue only two requests before processing other device requests, the possibility of I/O lockout to other devices is avoided. Since I/O can execute every 10 ms and the extraction device has processing priority, the extraction device can expect to be continuously queued in a capacity situation.

5.1.9 Extraction.

The purpose of extraction is to process requests for data extraction made by all other modules in the system. A flow chart for extraction is given in figure 5.1.9-1. Each extraction request is independently processed. If the data extraction flag is not set, an immediate return is executed. The space in the current extraction buffer is compared against the space necessary to process this request. If sufficient space is available, the extraction message along with the extraction header is moved into the output buffer and the current buffer pointer and word counter are adjusted. Otherwise, the current buffer is released for

output to I/O and a check is made to determine if the next buffer is busy. The actual I/O request for extraction data to be sent to the tape device is performed by I/O. Coordination between the two tasks is provided by an array of logical flags that define the state of each extraction buffer. If the buffer is not available, then the message is lost and a system error counter (#5) is incremented. Otherwise, the current buffer pointers and counters are updated for the next available buffer and the logical message is moved there with its header.

Extraction processes all extraction requests from any task in the system. This module has been structured so that any process that operates through the scheduling task may request extraction.

Each request for extraction must define to the extraction process the format number, the number of 32-bit words to be extracted, and the position of the first word to be extracted. This information composes a logical header which consist of two 16-bit fields and a 32-bit field, as depicted in figure 5.1.9-2. Logical records are combined into a physical record until a request for extraction exceeds the space available in the physical record. At this point a logical termination record is extracted. All data beyond the terminal record to the end of the physical record should be ignored.

5.1.10 X.25 Processing.

The X.25 processing routines consist of subroutines for the initialization of the X.25 links, subroutines for the transmission of packets, subroutines for the control of the X.25 links, and interrupt service routines. These routines interface all the X.25 functions with the CID real-time process except for the receiving of the data. The X.25 data receiving function is accomplished by the INMESS processing routine described in section 5.1.4 of this report. The buffering of all X.25 data and control variables is contained within the CDx common data areas. Figure 5.1.10-1 depicts the memory structure for the CID process.

The CDx common data areas are implemented as fixed task common storage elements. This will cause the address of these data areas to be predefined in order to allow access from not only the CID task routines but also the X.25 interrupt service routines and the Macrolink X.25 protocol control hardware. The interrupt service routine executes in interrupt state of the processor and must have access to the common area in memory directly without the memory address controller used to modify the address. The Macrolink X.25 protocol control hardware must have access to the common area through the direct memory access mode for interaction with the input buffers, output buffers, and control variables. Routines execution within the CID real-time simulation software will access the common areas through the normal common/task interface using the memory address control mechanism of the 3230XP processor.

A separate common area exist for each X.25 device in the CID system. The common area contains eight input buffers and eight output buffers for each of the Macrolink X.25 devices included within the CID. These buffers are 2048 bytes long for a total length of 32k bytes. The X.25 variable area is positioned after the buffer area. This variables consist of control parameters and error counters for the X.25 interface. The control variables and part of the error counters are directly accessible by the Macrolink X.25 protocol controller through the direct

memory access mode of the computer. Other error conditions are detected by software routine and the interrupt service routine. These variables are accessed directly by the appropriate routine. Figure 5.1.10-2 depicts the structure for the X.25 CID common area.

Various subroutines were written for the control of the Macrolink X.25 protocol controller board. These routines included: MDISC X25, RESET X25, INIT X25, LINK STATUS, ACTIVE X25, and DEACTIVE X25. These routines are utilized by the Init routine and the Op-Comm routines.

MDISC X25 will execute a master disconnect of the X.25 protocol controller and will set the link connect flag in the common to a condition of false. Execution of the routine will cause all current X.25 operations to cease and the protocol controller to be put in a quiescent state. This routine is executed by Init at the beginning of simulation and by Op-Comm function nine at the end of simulation. Figure 5.1.10-3 depicts a flowchart of the MDISC X25 subroutine.

RESET X25 will clear the direct memory access and COMMUX ports of the X.25 protocol controller. The direct memory access and COMMUX ports are the access paths from the current 3230XP computer. The COMMUX path provides the control path for the Macrolink X.25 protocol controller. The direct memory access path provides the path for data, incoming and outgoing X.25 packets, between the Macrolink protocol controller and the current 3230XP computer. Figure 5.1.10-4 depicts a flowchart of the RESET X25 subroutine.

INIT X25 will initialize the direct memory access function and the X.25 protocol hardware of the Macrolink protocol controller. This routine is utilized by the INIT routine at the beginning of the real-time simulation. Figure 5.1.10-5 depicts a flowchart of the INIT X25 subroutine.

LINK STATUS is a logical function that will determine the current status of the X.25 link. The inquiry is made to determine if the X.25 link is active. The logical function will return the current state of link activity. This is accomplished by accessing registers on the Macrolink protocol controller for the current status of the X.25 link. Figure 5.1.10-6 depicts a flowchart of the LINK STATUS function.

ACTIVE X25 will set the X.25 active flag in the common area. DEACTIVE X25 will reset the X.25 active flag in the common area. Figure 5.1.10-7 depicts a flowchart of the ACTIVE X25 and DEACTIVE X25 subroutines.

TXMITn is a subroutine that provides for the transmission of X.25 packets. A copy of the routine with appropriate common memory access exist for each of the Macrolink X.25 protocol controllers in the CID system. This routine requires a buffer with the data to be transmitted in the packet and the size of the packet. TXMIT will return a status for the transmission request. The routine will first check for the availability of a buffer and report an error if no buffer is available. A check of the packet size is made to determine that the request is within the bounds for transmission. An error is reported if the packet size is out of bounds. The status of the link is checked to determine if the message may be accepted for transmission. An error is reported if the message is not accepted. The data is moved to the transmit buffer in the task common and the Macrolink protocol controller is instructed to proceed with the transmission of the packet. Pointers are updated before the routine returns to the call process. Figure 5.1.10-8 depicts a flowchart of the TXMITx subroutine.

The interrupt service routine (ISR) is an assembly language program for processing all interrupts from the Macrolink protocol controller. Interrupts are generated whenever a packet is transmitted, a packet is received, or an error is generated. A separate copy of the ISR is maintained for each X.25 device in the CID system. The ISRs are located in memory just above each fixed task common containing the common data interface to the real-time task. This was done to accommodate the ease of access of the common areas for both the ISR and real-time task.

When an interrupt occurs, the ISR first will check the status byte for the interrupt type. Only interrupts from the X.25 network device require processing. All interrupt conditions are saved for possible later analysis. Any combination of the three types of X.25 network interrupt conditions may occur during an interrupt. Each condition must be checked and processed independently. First, the receiver packet bit is examined in the status byte. If a packet was received, the packet received bit is set for that device in fixed task common area. The INMESS routine will see this bit and process the incoming data packet. Next, the packet transmit acknowledge bit is examined in the status byte. And, finally, the error bit is examined in the status byte.

When the error bit is set in the status byte, a single error has occurred. This error will be determined by examining the error register in the X.25 network device. Errors may occur because of the following conditions:

a. Receiver Overrun
b. Transmitter Underrun
c. Receiver not ready for next packet (buffer not available)
d. Link is up (link was down)
e. Disconnect sent
f. Received a disconnect
g. Link reset received
h. Supervisory command timeout
i. Frame reject received
j. Frame reject sent

Each error has an associated error counter that will be extracted regularly. The error counters are also displayed on the statistics screen by Op-Comm function 4. The ISR will process the link up interrupt, the link reset interrupt, and the frame reset interrupt with and additional algorithm to reset the buffers and hardware of the X.25 network device. A flowchart of the ISR is depicted in figure 5.1.10-9.

5.2 CID INITIALIZATION PROGRAM.

The CID Initialization (CIDINIT) Database program builds a file that contains the configuration data required to run the CID real-time program. This data consists of physical line to logical port assignments, X.25 set up parameters, and default text fields for standard uplink and ELM uplink messages. This program provides the method for bypassing the CIDINIT routine's interactive session. All of the values normally requested during the CID interactive session can be defined and saved into the CIDINIT data file before a CID simulation is run. This offers the user a method of ensuring that two separate CID simulation runs use the same initialization parameters. CIDINIT is flexible enough to allow the selective definition of the initialization parameters, i.e., the CID interactive session bypasses those queries which are defined within the CIDINIT Database file, but still requests those parameters which are not defined within the file.

CIDINIT is written in "C" under the OS/32 operating system. The program uses a main routine to process the CIDINIT main menu and function calls to process each of the four sub-menus which may be requested by the user. The sub-menus include: (1) X25 device initialization screen, (2) simulation initialization screen, (3) standard uplink default message screen, and (4) the ELM uplink default message screen.

5.2.1 CIDINIT Main Routine.

The source file for the main routine is CIDINIT.C.

The main routine first prints the main menu on the user terminal. An answer from the user is then processed. When the answer decoded is a "P," the print request filename query is displayed. The answer from the user is then processed. When the answer received is a valid filename (meets OS/32 naming conventions and it exists on disk), the file is opened, read, spooled to the line printer, and then closed.

When the response to the main menu is a "M," the modify request filename query is displayed. The answer from the user is then processed. When the answer processed is a valid filename (meets OS/32 naming conventions and it exists on disk), the file is opened, read, processed by each of the sub-menu routines, written back to the named file, and then closed.

When the response to the main menu is a "C," the create request filename query is displayed. The answer from the user is then processed. When the answer processed is a valid filename (meets OS/32 naming conventions and it does not exist on disk), the file is created, initialized, processed by each of the sub-menu routines, written back to the named file, and then closed.

When the response to the main menu is an <esc><cr>, the program is terminated.

Figure 5.2.1-1 is a flowchart of the main routine.

5.2.2 Initialization and I/O Routines.

Routines to initialize data structures and process I/O are stored in the CONFINIT.C source file.

The creat_init routine is used to initialize standard uplink arrays, ELM uplink arrays, port values and baud-rate values on creation of a new file. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.2-1 is a flowchart of this routine.

The in_c_file routine is used to read data from the named file when either a print request or modify request is chosen by the user. This routine is a series of read statements to read each of the values stored in the file in the past. After the read portion of the routine is complete, the ASCII values read from the file are converted to a usable form for later processing needs. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.2-2 is a flowchart of this routine.

The out_c_file routine is used to write data to the named file after either a create request or modify request is chosen by the user. The routine is called by the main routine. It neither requires or returns any parameters. This routine is a series of write statements to write each of the values to be contained in the file. Figure 5.2.2-3 is a flowchart of this routine.

5.2.3 X.25 Device Routines.

Routines to process the X.25 device routines are stored in x25dev.c.

The x25_dev_sc routine is used to display the X.25 device configuration screen. The routine is a series of cursor positioning and print statements to locate and print the menu's text strings. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.3-1 is a flowchart of this routine.

The mod_x25_dev routine is used to process the the user's choices after displaying a message. The message asks the user to select a X.25 device to modify. Once a legitimate device number is selected, two routines; port and baud, are called to process the port number and baud rate columns of the x.25 device menu. This routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.3-2 is a flowchart of this routine.

The port routine contains logic to process the user's port number selection. The routine calls valid_port to make sure that the value chosen by the user is possible based on the values already contained in the port number and baud rate buffers. Port is called by the mod_x25_dev routine. It neither requires or returns any parameters. Figure 5.2.3-3 is a flowchart of this routine.

The baud routine contains logic to process the user's baud rate selection. The routine calls valid_baud to make sure that the value chosen by the user is possible based on the values already contained in the port number and baud rate buffers. Baud is called by the mod_x25_dev routine. It neither requires or returns any parameters. Figure 5.2.3-4 is a flowchart of this routine.

5.2.4 Simulation Initialization Routines.

Routines to process the simulation initialization routines are stored in SIMINIT.C.

The sim_init routine is used to display the simulation initialization screen. The routine is a series of cursor positioning and print statements to locate and print the menu's text strings. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.4-1 is a flowchart of this routine.

The mod_sim_init routine is used to process the the user's choice after displaying a message. The message asks the user whether to continue with this screen or not. When a "M" is selected, mod_sim_init processing continues. Responses to each of the five questions on this screen are then processed. This routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.4-2 is a flowchart of this routine.

5.2.5 Standard Uplink and ELM Default Message Routines.

Routines to process the default message menus are stored in MESSDFLT.C.

The comma_sc routine is used to display the standard uplink (comm A) default message menu. The routine is a series of cursor positioning and print statements to locate and print the menu's text strings. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.5-1 is a flowchart of this routine.

The elm_sc routine is used to display the ELM default message menu. The routine is a series of cursor positioning and print statements to locate and print the menu's text strings. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.5-2 is a flowchart of this routine.

The mod_comma routine is used to process the user's choice after displaying a message. The message asks for the standard uplink default message number to modify. Once a legitimate default message is selected, the user's input default message is read. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.5-3 is a flowchart of this routine.

The mod_elm routine is used to process the user's choice after displaying a message. The message asks for the elm default message number to modify. Once a legitimate default message is selected, the user's input default message is read. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.5-4 is a flowchart of this routine.

5.2.6 X.25 Device Configuration Display Routine.

PRINTCID.C is the source file for this routine.

The routine simply spools each of the 4 screens to the printer by using 4 sets of print statements to format the desired output. The routine is called by the main routine. It neither requires or returns any parameters. Figure 5.2.6-1 is a flowchart of this routine.

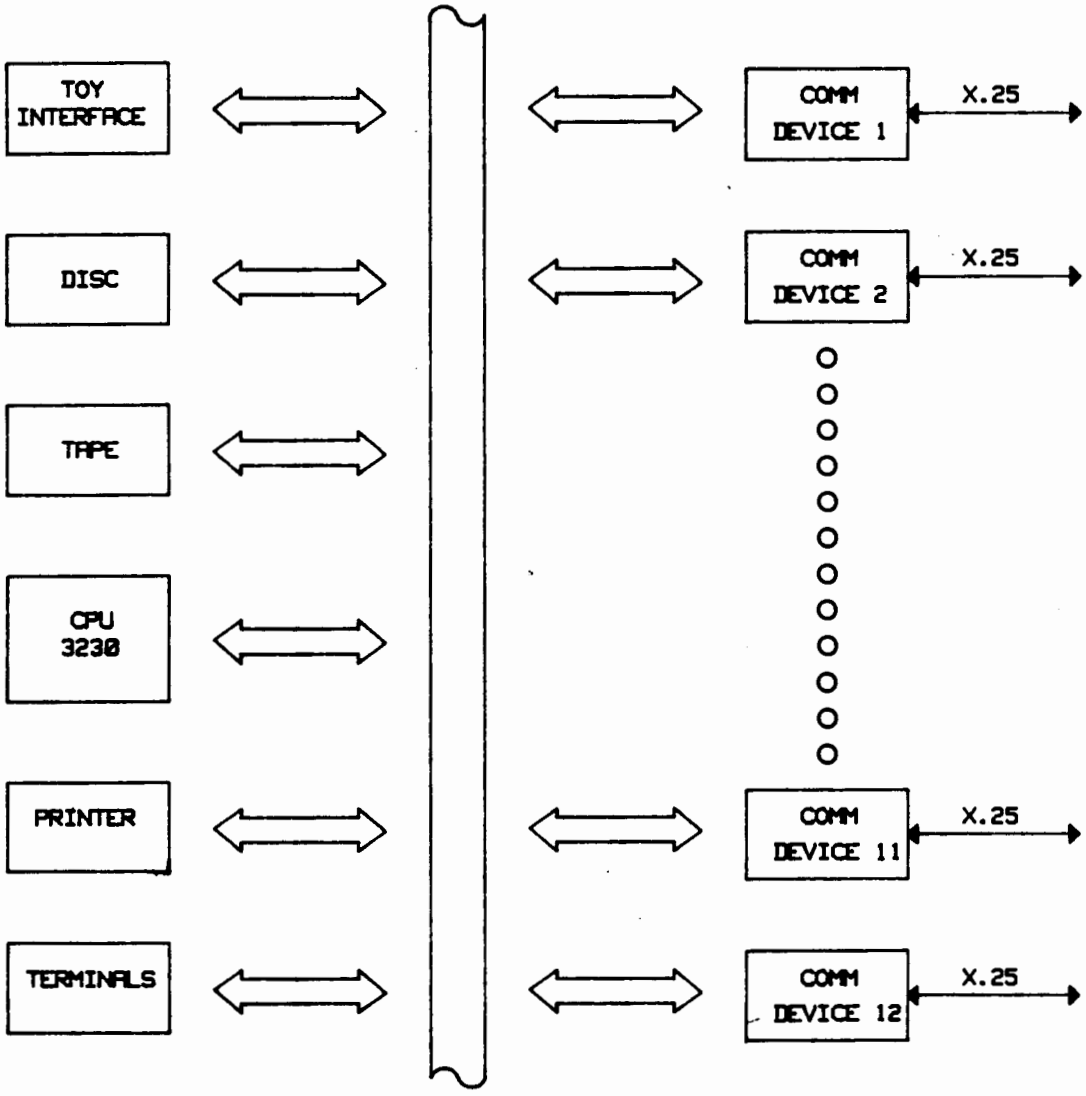


FIGURE 1-1. CID FUNCTIONAL BLOCK DIAGRAM

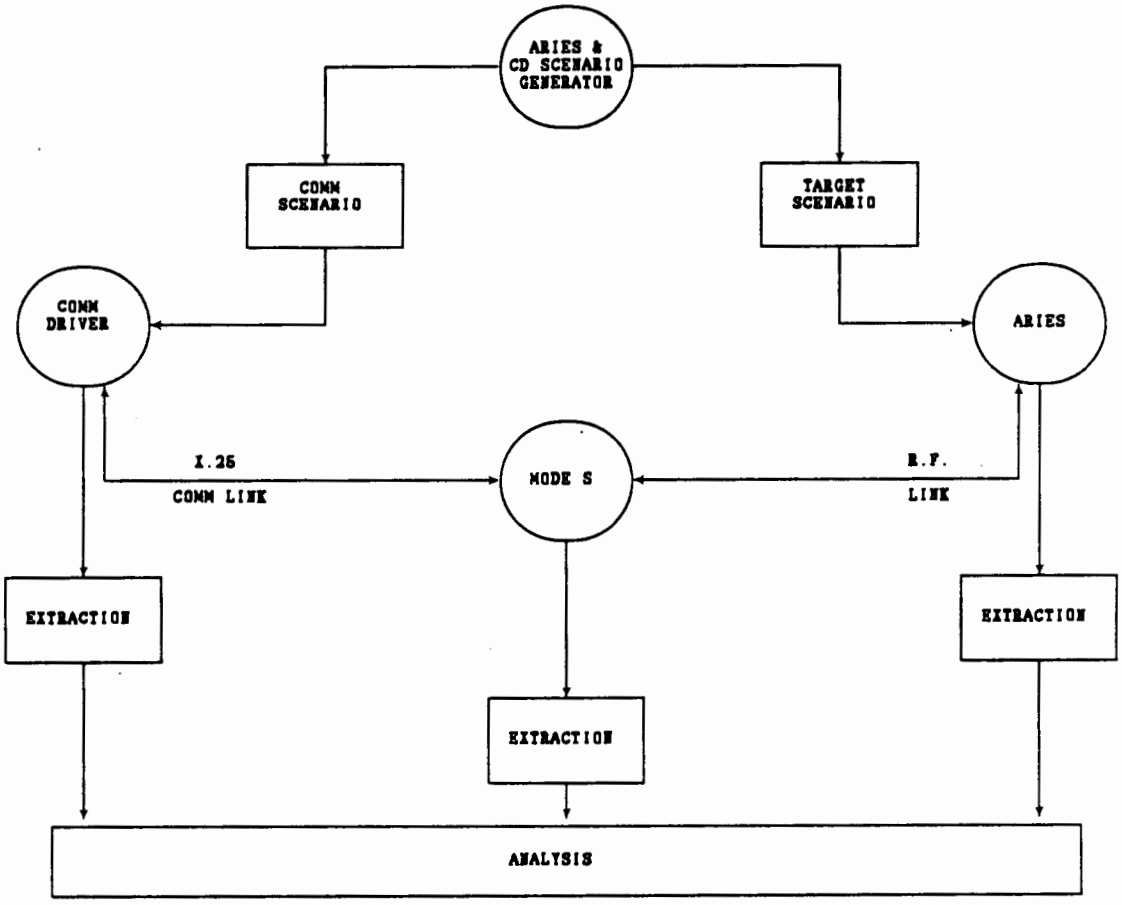


FIGURE 1-2. ARIES/CID SIMULATION BLOCK DIAGRAM (PHASE I)

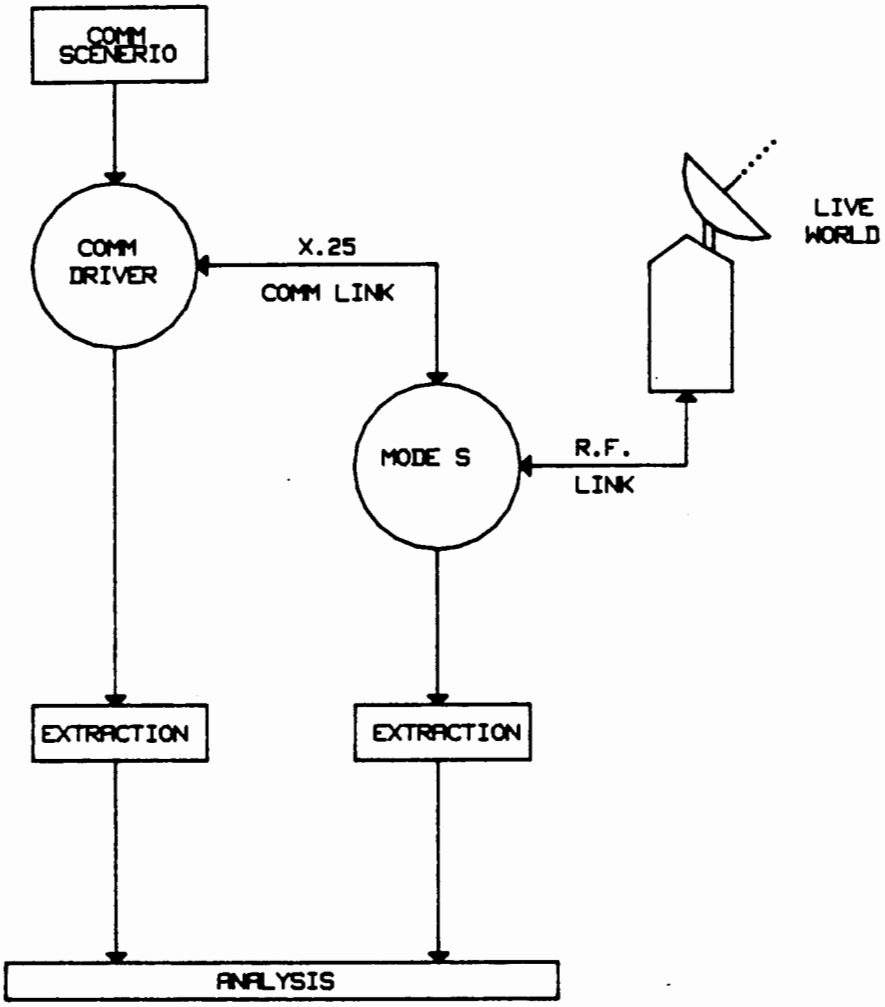


FIGURE 1-3. CID LIVE WORLD BLOCK DIAGRAM (PHASE I)

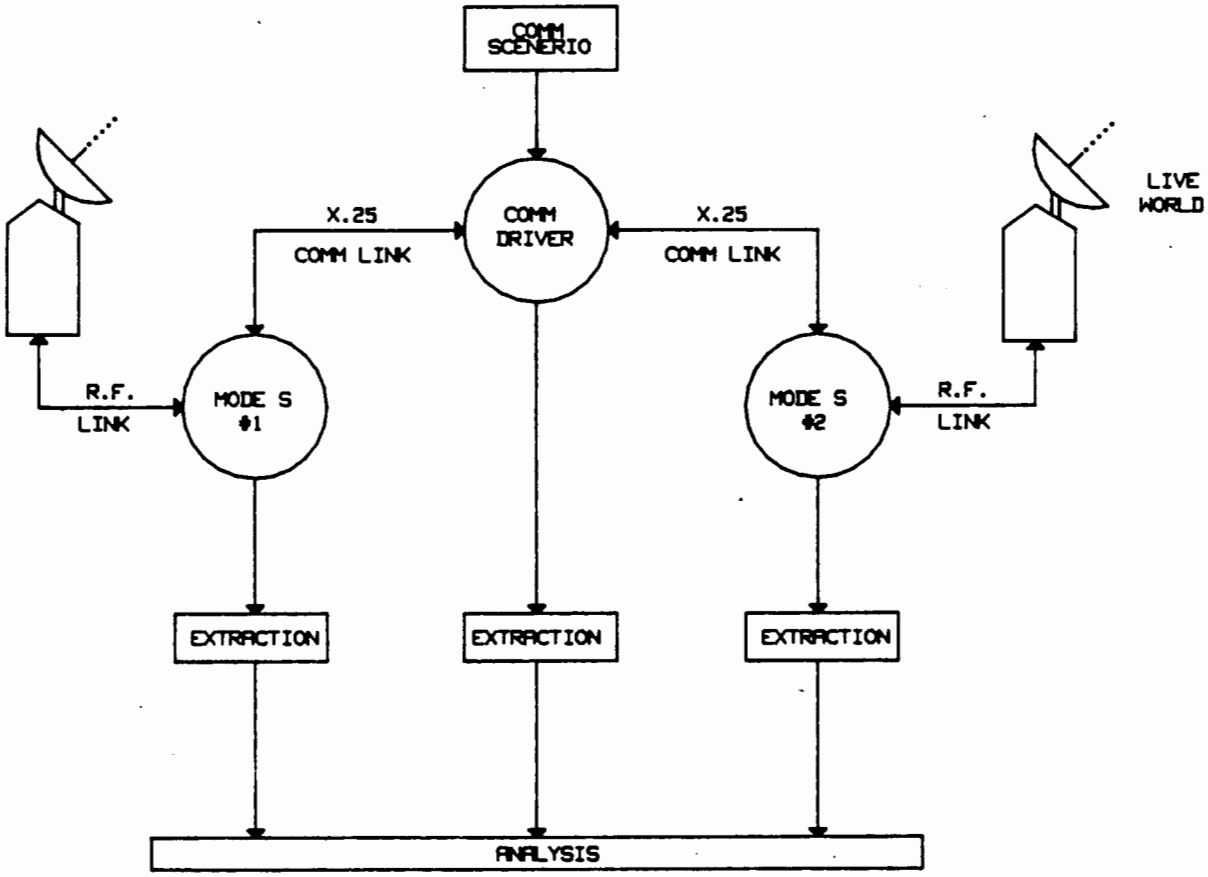


FIGURE 1-4. CID LIVE WORLD BLOCK DIAGRAM (PHASE II)

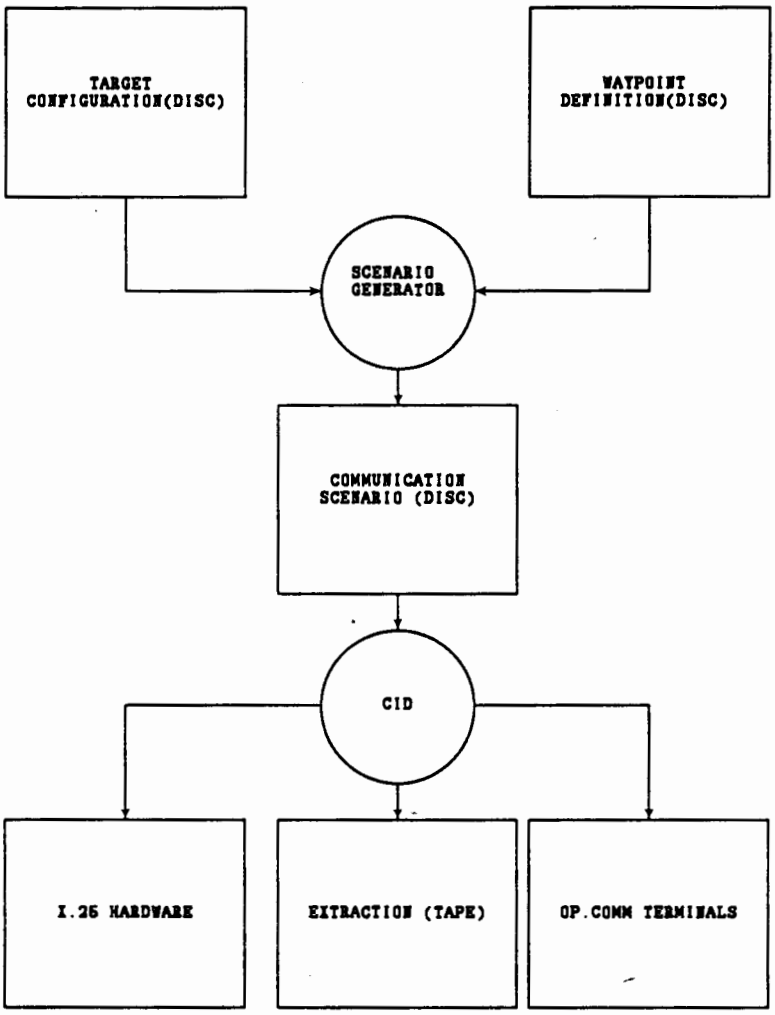


FIGURE 5-1. SOFTWARE OVERVIEW

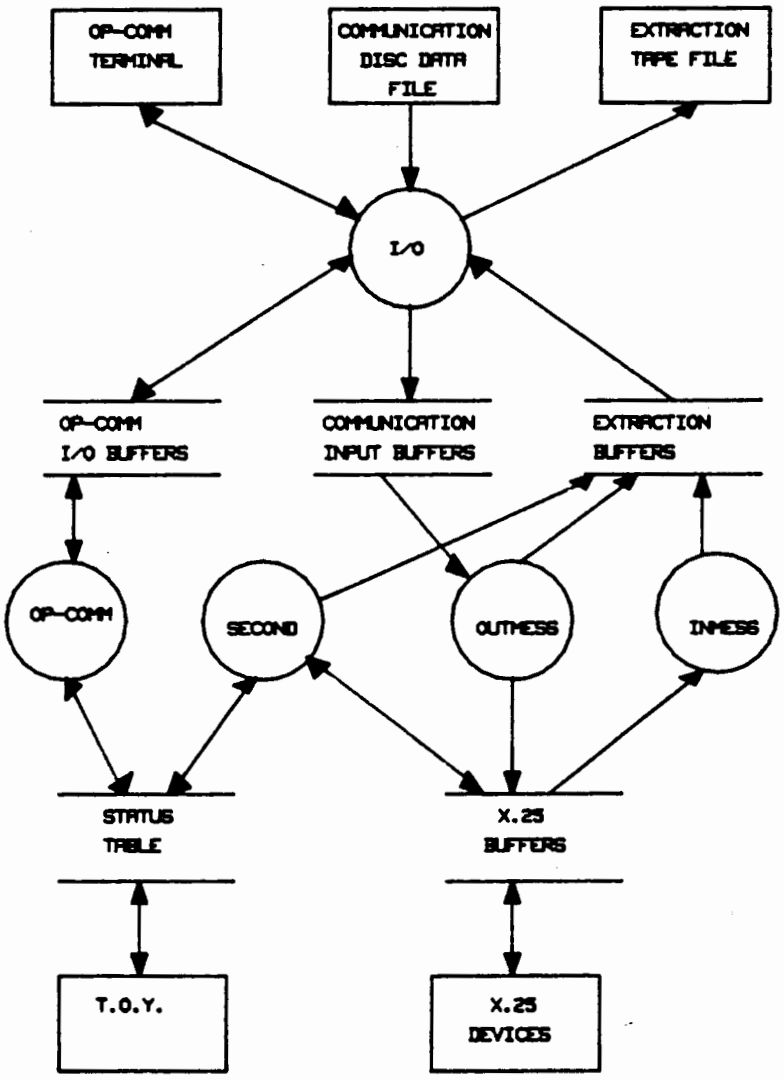


FIGURE 5.1-1. CID MODULE OVERVIEW

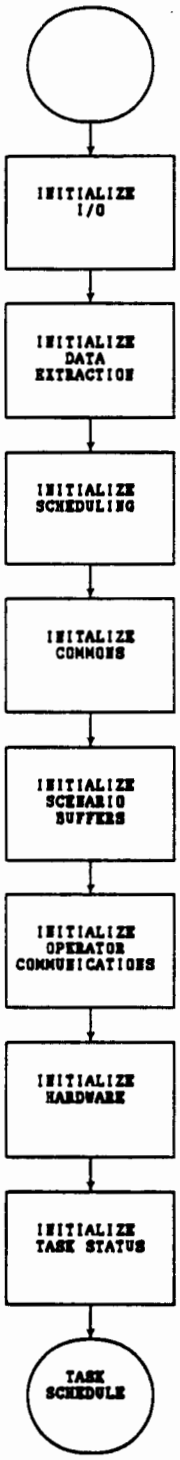


FIGURE 5.1.2-1. INITIAL PROGRAM FLOWCHART

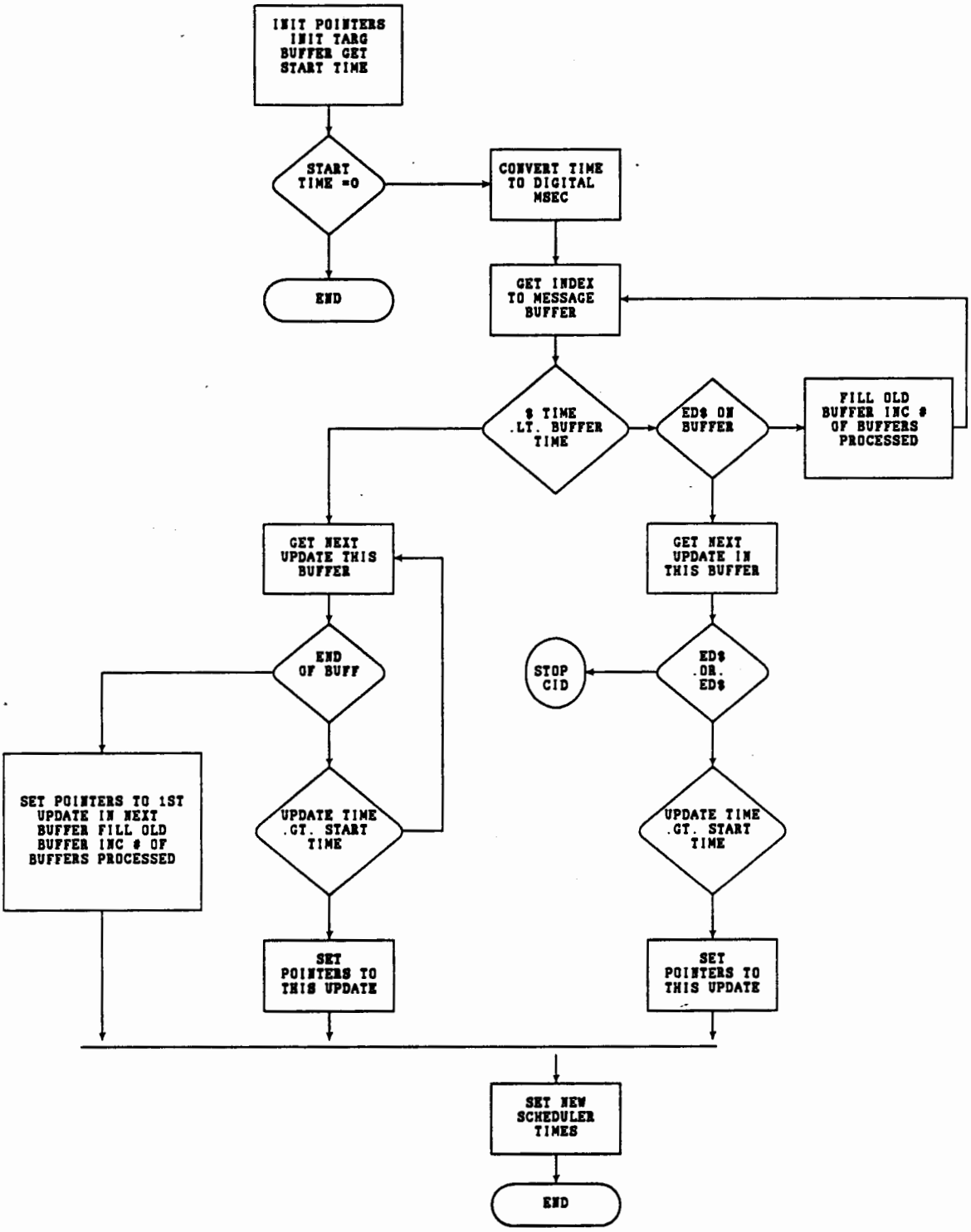


FIGURE 5.1.2.5-1. FLOWCHART OF BLOCK INIT SCEN BUFFERS

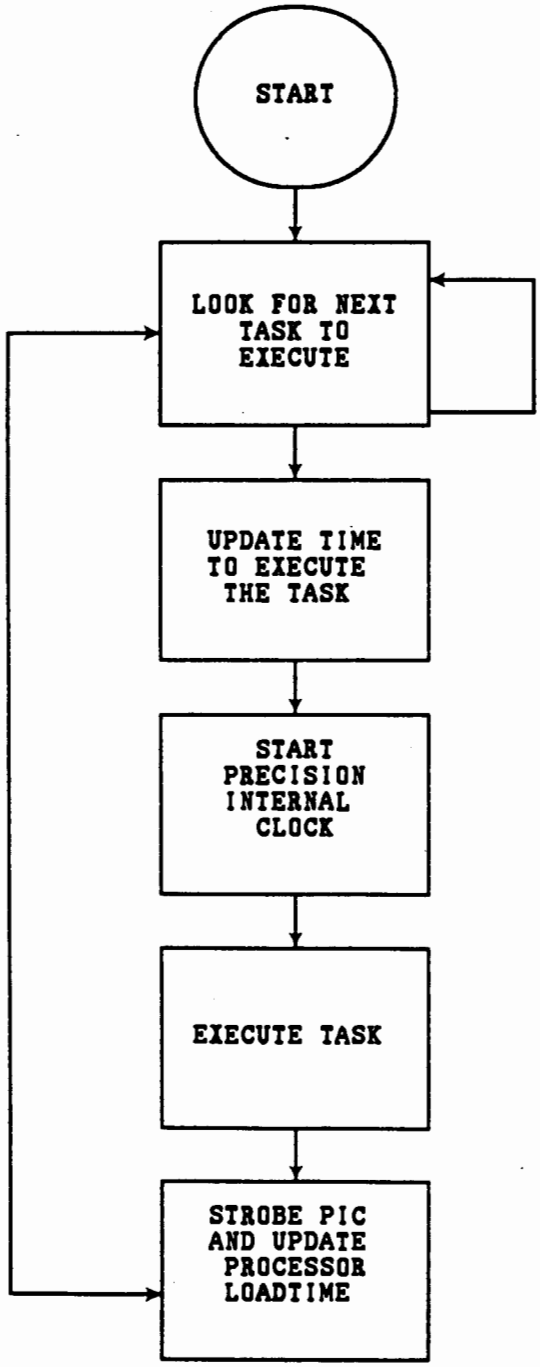


FIGURE 5.1.3-1. TASK SCHEDULING PROGRAM FLOWCHART

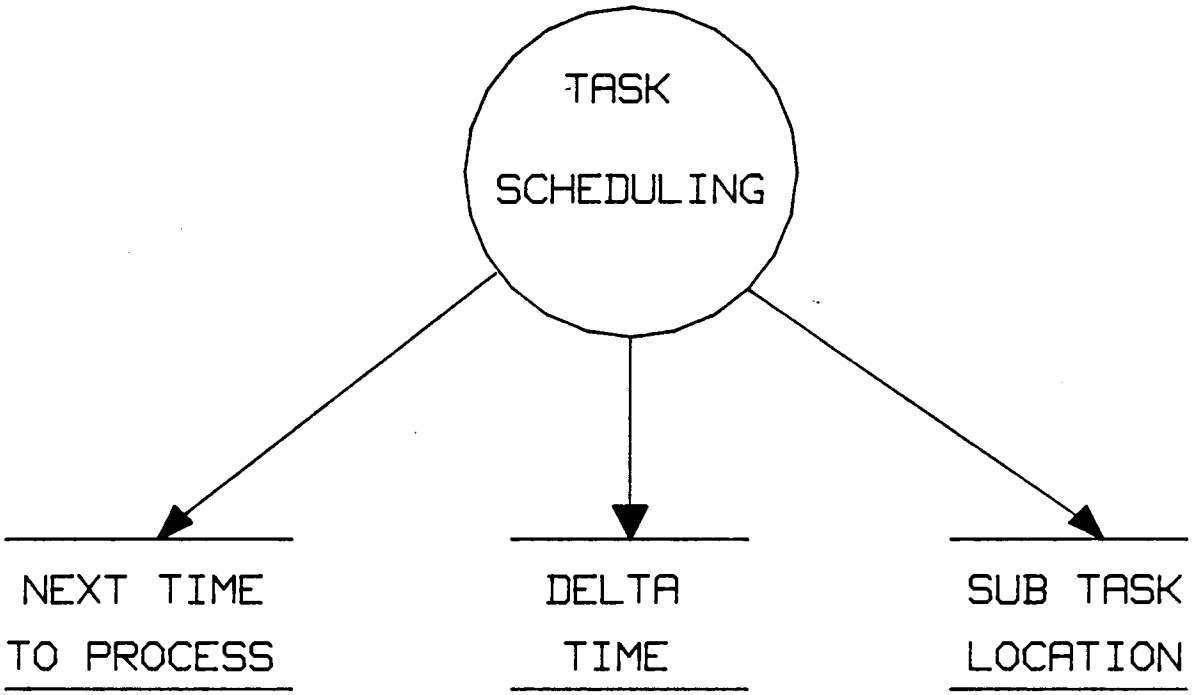


FIGURE 5.1.3-2. TASK SCHEDULING DATA FLOWCHART

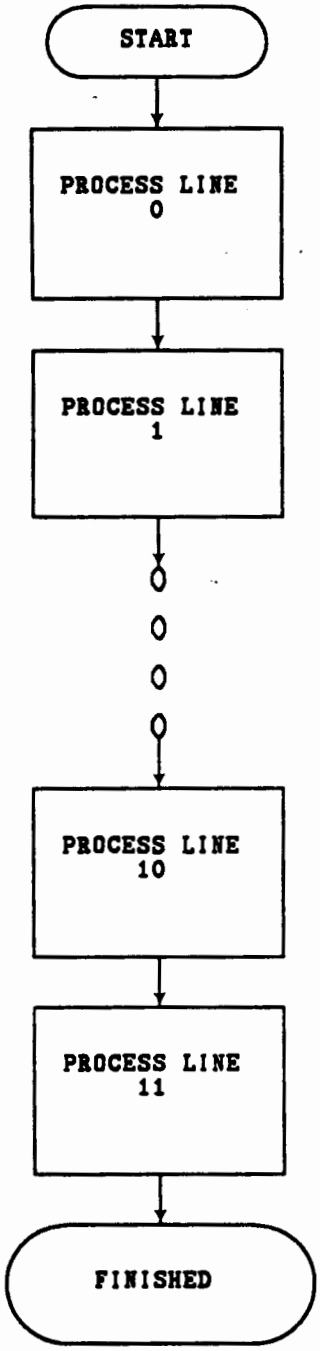


FIGURE 5.1.4.1-1. FLOWCHART OF INMESS

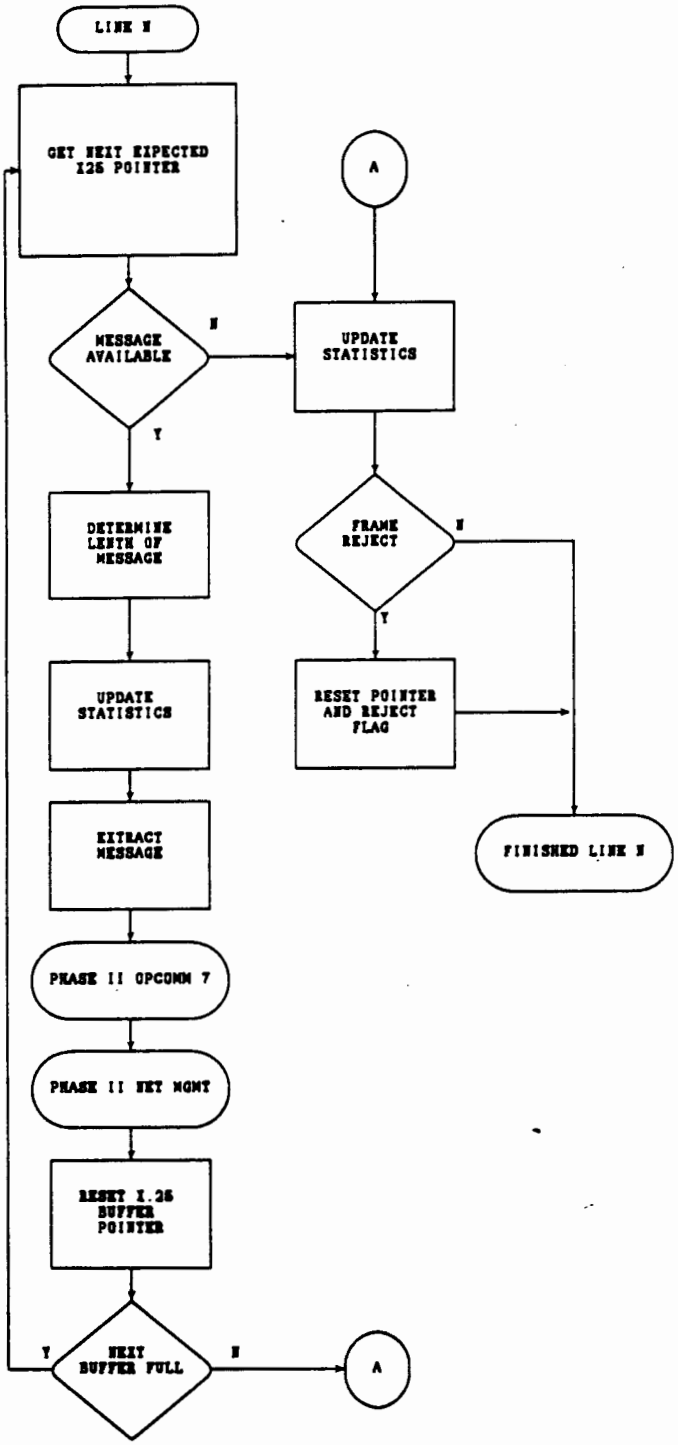


FIGURE 5.1.4.1-2. FLOWCHART OF X.25 LINE PROCESSING

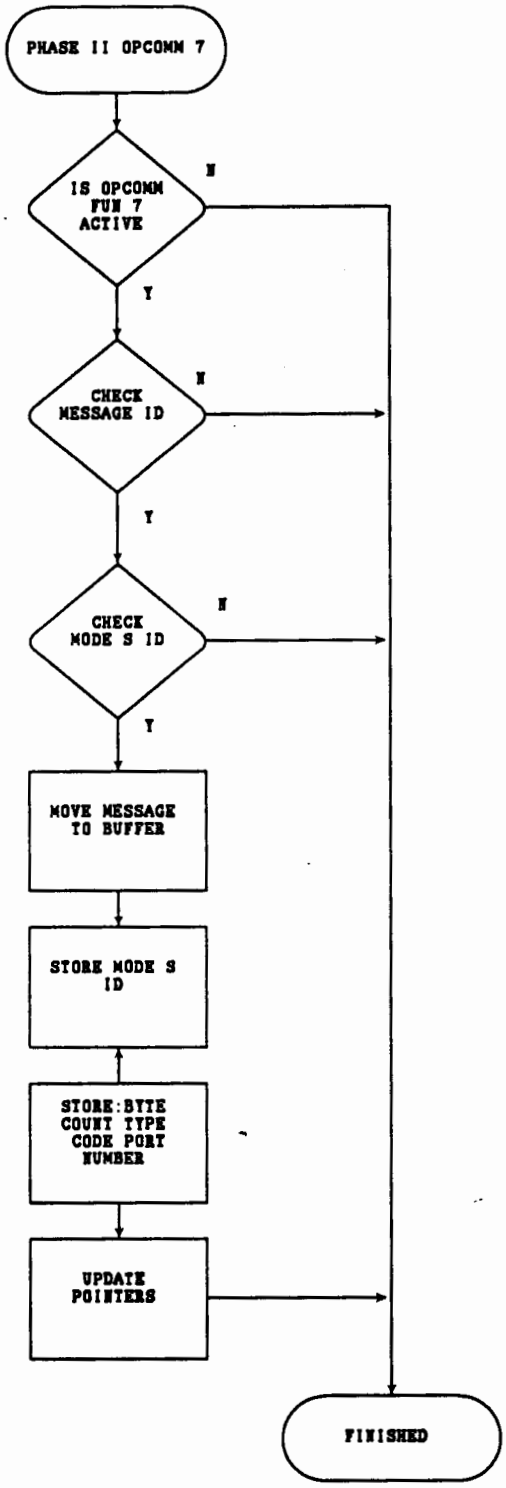


FIGURE 5.1.4.2-1. FLOWCHART OF OP-COMM MESSAGE INTERFACE (PHASE I)

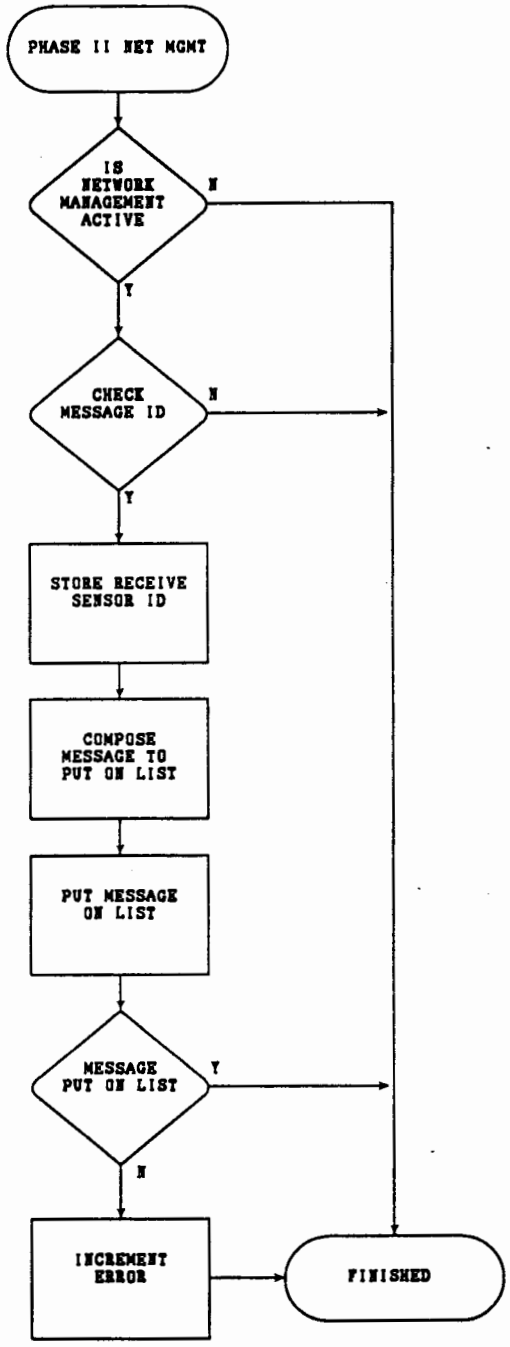


FIGURE 5.1.4.3-1. FLOWCHART OF NETWORK MANAGEMENT INTERFACE (PHASE II)

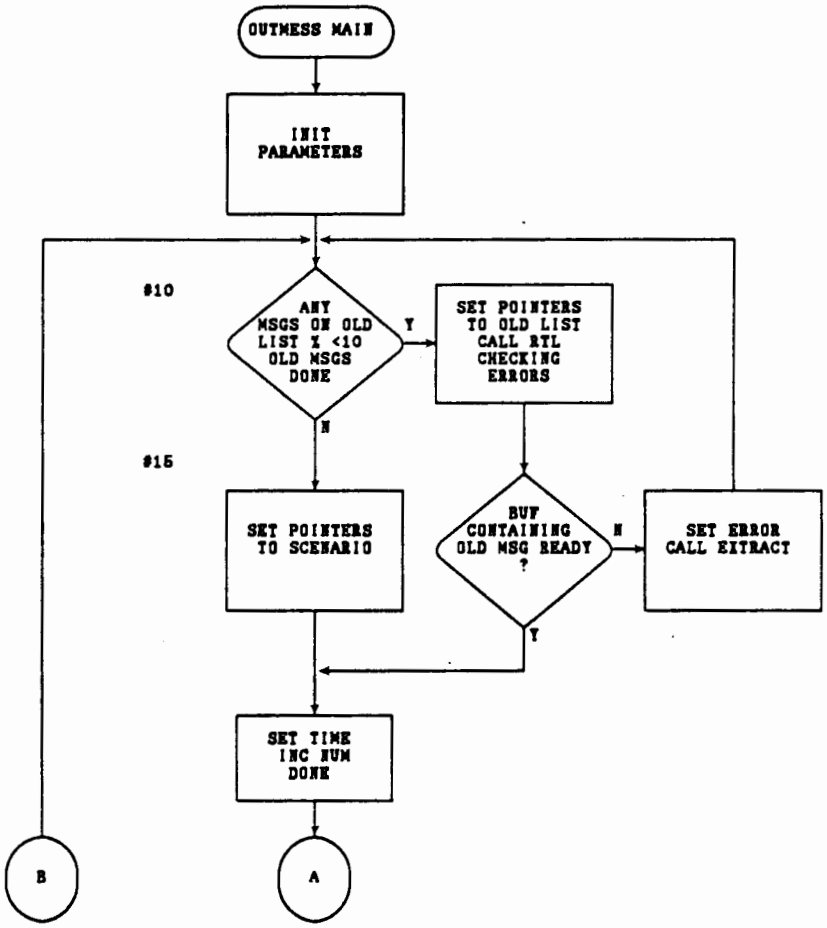


FIGURE 5.1.5-1. FLOWCHART OF OUTMESS (Page 1 of 2)

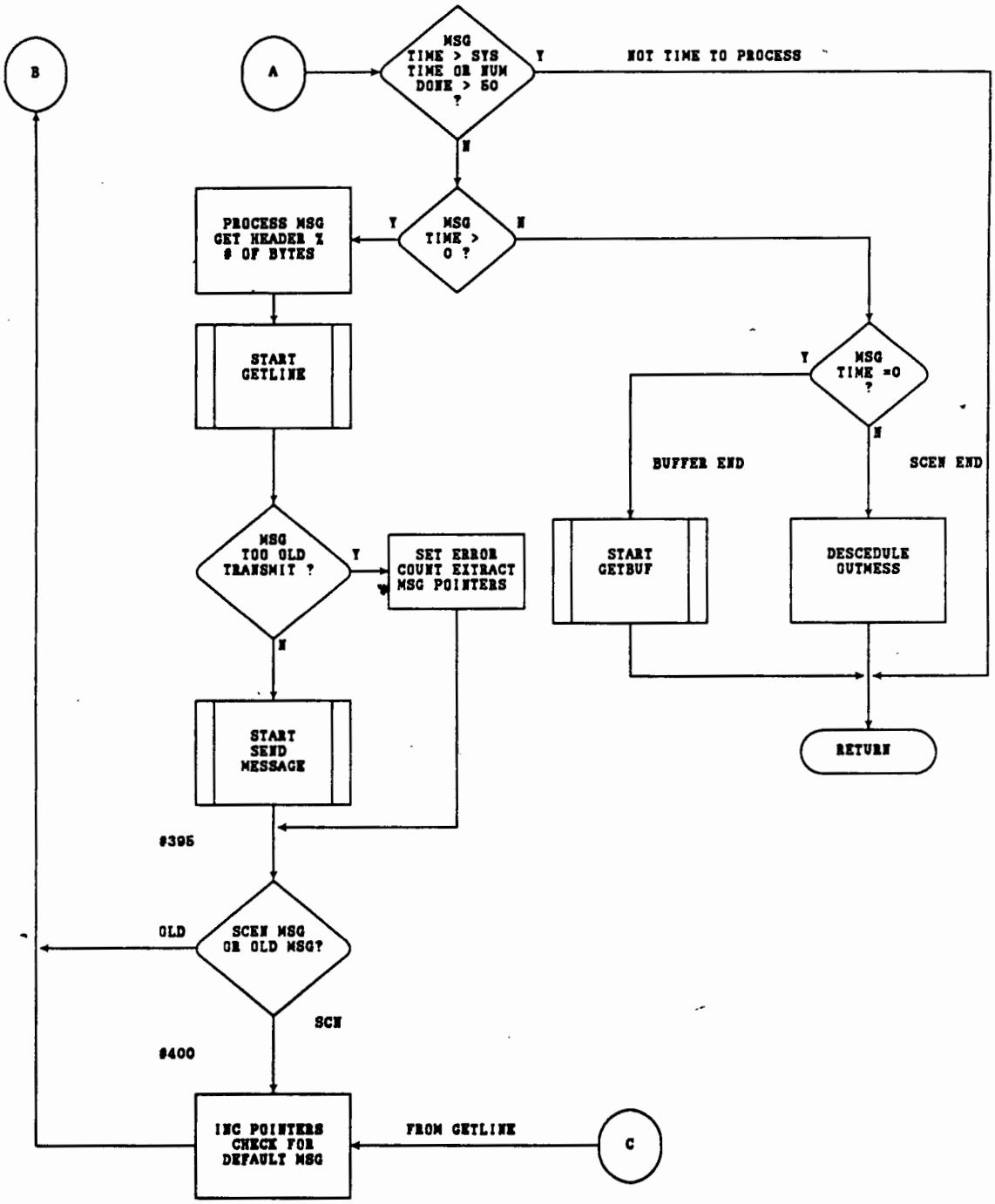


FIGURE 5.1.5-1. FLOWCHART OF OUTMESS (Page 2 of 2)

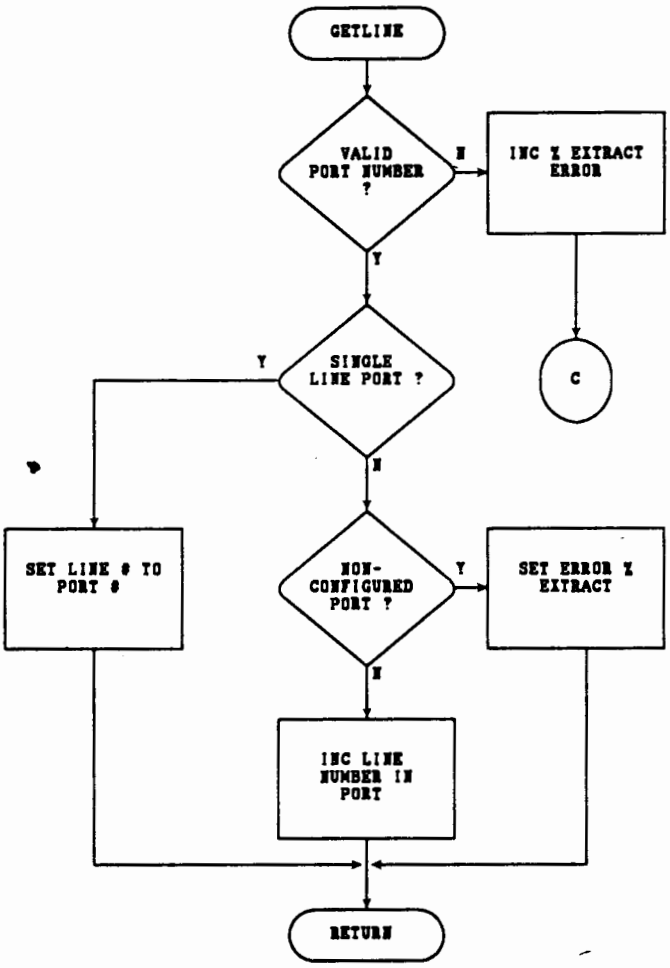


FIGURE 5.1.5-2. FLOWCHART OF GETLINE

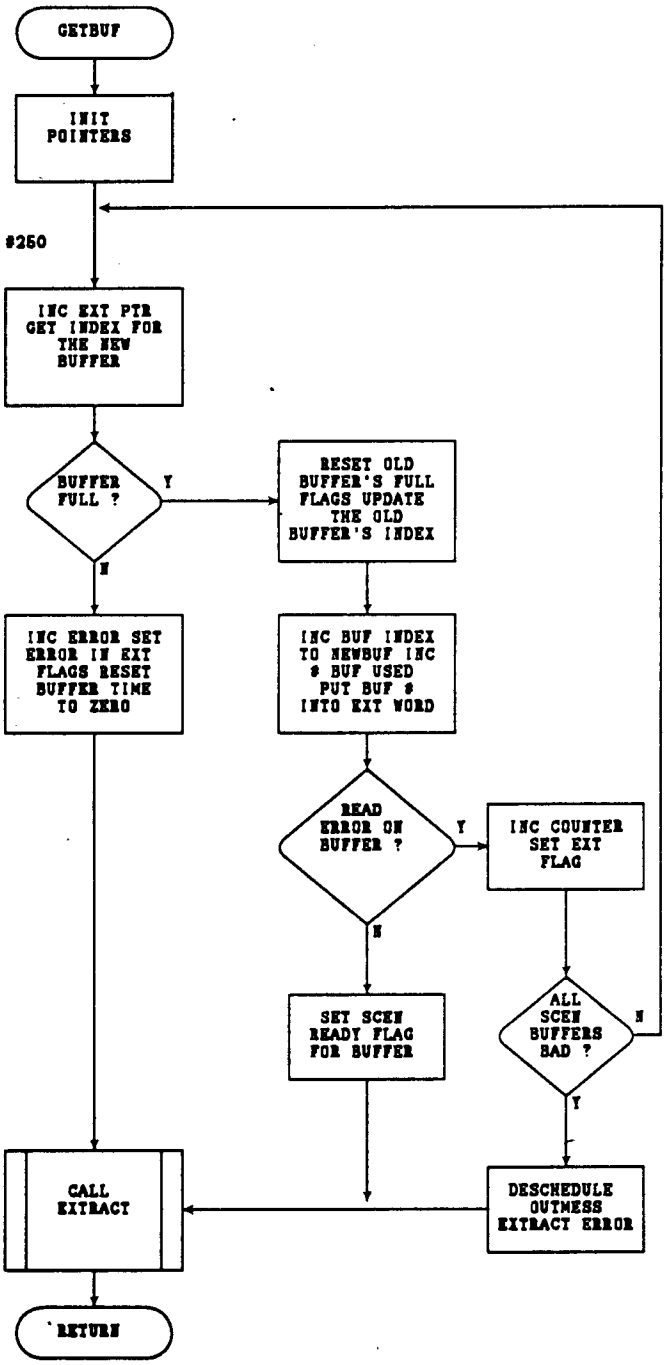


FIGURE 5.1.5-3. FLOWCHART OF GETBUF

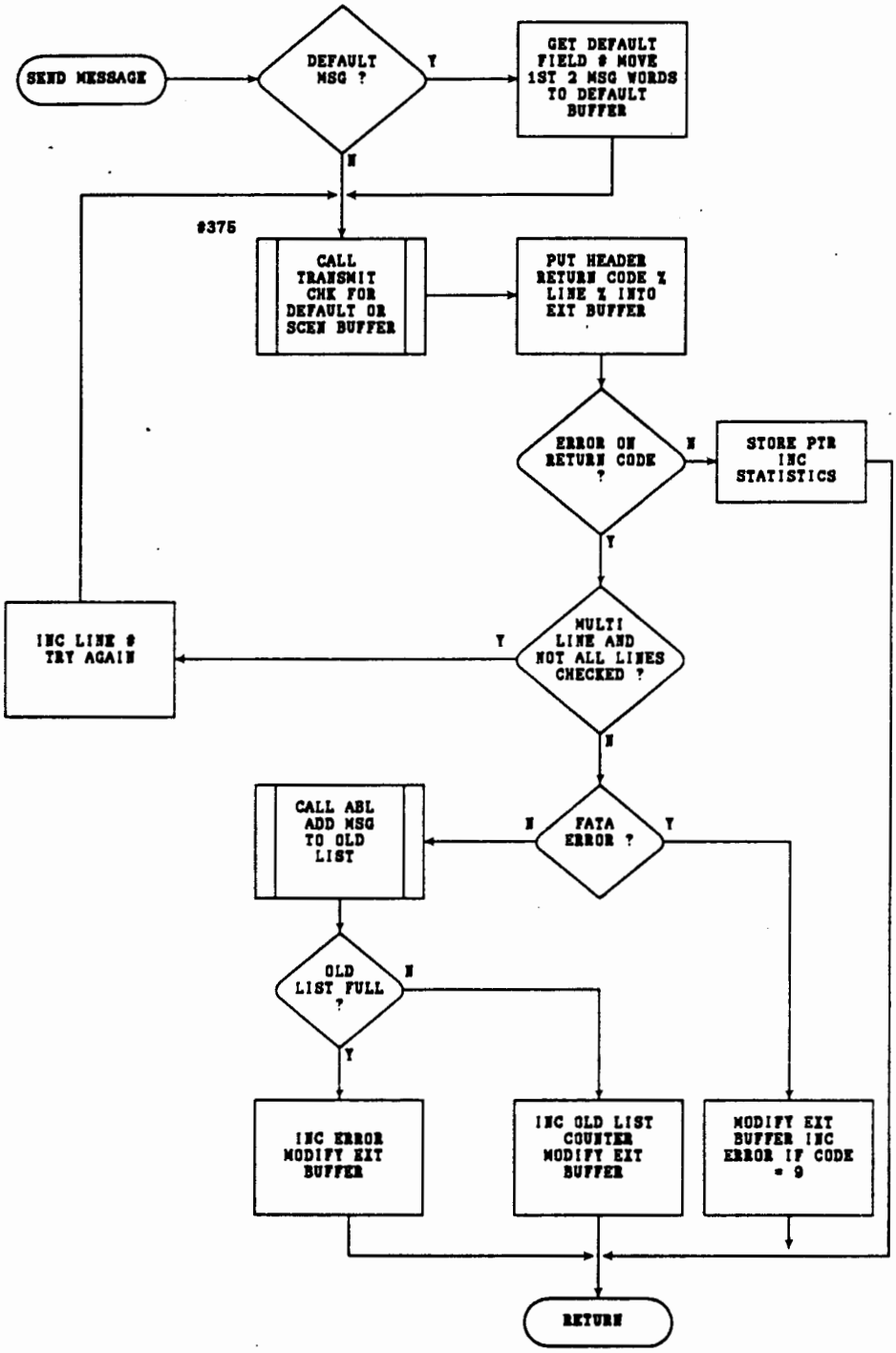


FIGURE 5.1.5-4. FLOWCHART OF SEND MESSAGE

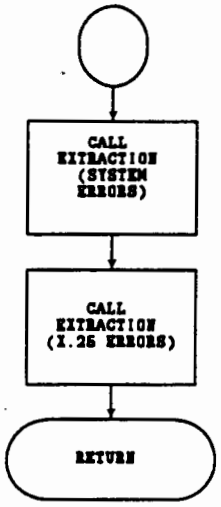


FIGURE 5.1.6-1. SECOND PROCESSING FLOWCHART

ERROR	DESCRIPTION
1	Extraction I/O
2	Message Scenario I/O
3	unused
4	OP COMM I/O
5	Extraction Messages Lost
6	unused
7	unused
8	Scenario Buffers Overwrite
9	Empty Scenario Buffers
10	Bad Port Number Found
11	Illegal X.25 Line
12	Savelist Found Empty
13	Savelist Full
14	Bad Line Number on Input
15	unused
16	X.25 Packet Error
17	Link Not Active
18	unused
19	unused
20	unused
21	unused
22	unused
23	unused
24	unused
25	unused
26	unused
27	unused
28	unused
29	unused
30	unused

FIGURE 5.1.6-2. SOFTWARE ERRORS

ERROR	DESCRIPTION
1	FCS ERROR
2	Short Frame
3	T1 Expiration
4	Rejection RX
5	Rejection TX
6	Link Initialization
7	Frame Reject on Transmit
8	Frame Reject on Receive
9	Transmit Buffers Full
10	Receive Buffers Full
11	S Command Time Out
12	Receiver Underrun
13	Transmitter Overrun
14	Link Reset
15	Other Errors

FIGURE 5.1.6-3. X.25 HARDWARE ERRORS

COMMUNICATIONS INTERFACE DRIVER FUNCTION MENU

1 - CID STATISTICS
2 - CID ERRORS
3 - X.25 DEVICE STATISTICS
4 - X.25 DEVICE ERRORS
5 - MODIFY X.25 DEVICE STATUS
6 - CREATE A MESSAGE
7 - DISPLAY A MESSAGE
8 - TOGGLE DATA EXTRACTION
9 - TERMINATE THE CID

PLEASE SELECT A FUNCTION

FIGURE 5.1.7.1-1. OP-COMM MENU

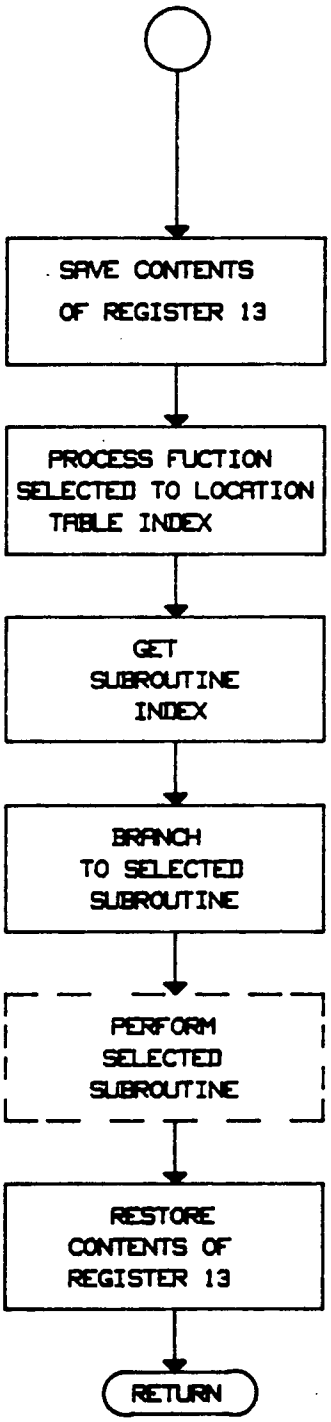


FIGURE 5.1.7.1-2. FLOWCHART OF COMPUTED GOTO ALGORITHM

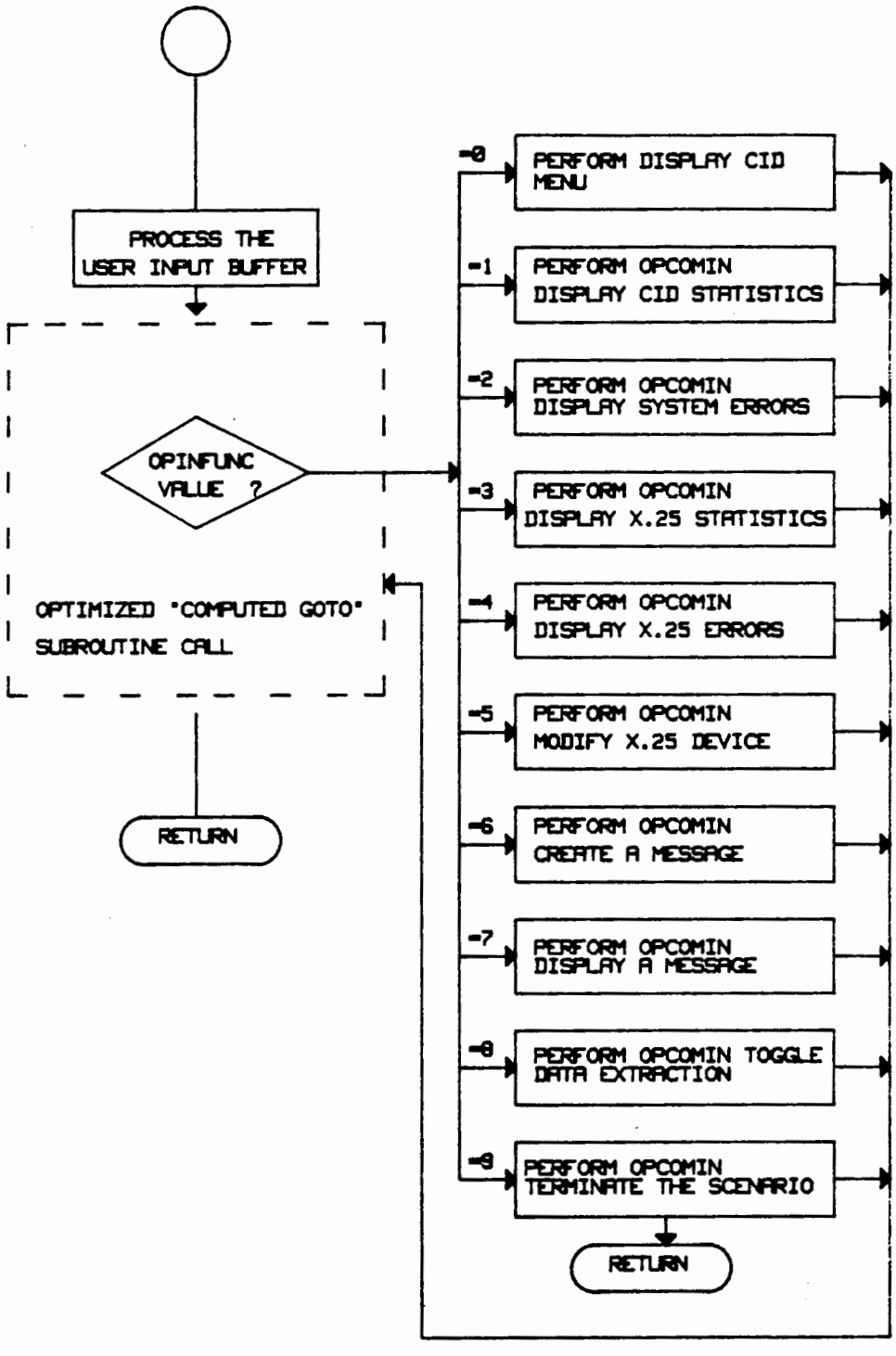


FIGURE 5.1.7.1-3. FLOWCHART OF THE OP-COMIN ROUTINE

Input Type	Function	State
Single Character	5	3
	6	18
Single Digit Decimal	Main Routine	
	6	2,4,10,12,13,15,16
	7	2
Several Digit Decimal	5	2
	7	3
Single Digit Hexadecimal	6	7
2 Digit Hexadecimal	6	3,6,8,11,14
6 Digit Hexadecimal	6	5
	7	4,5
14 Digit Hexadecimal	6	17
20 Digit Hexadecimal	6	9

FIGURE 5.1.7.1.1-1. TYPES OF EXPECTED INPUT

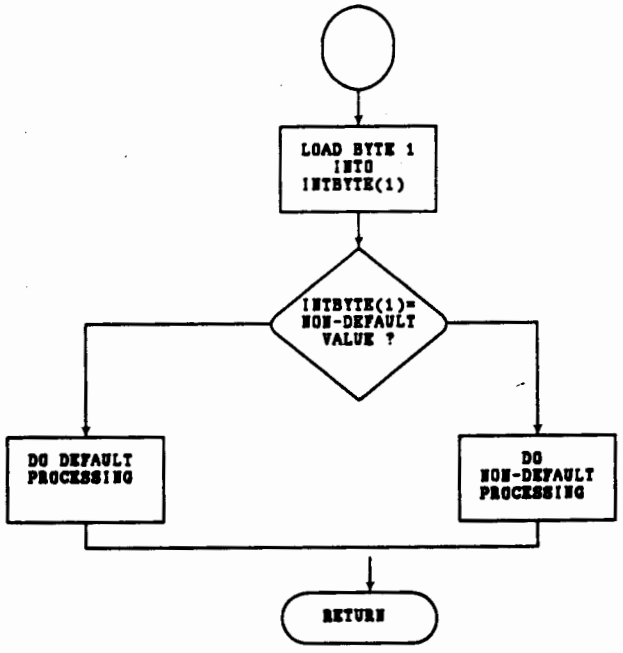


FIGURE 5.1.7.1.1-2. SINGLE CHARACTER INPUT FLOWCHART

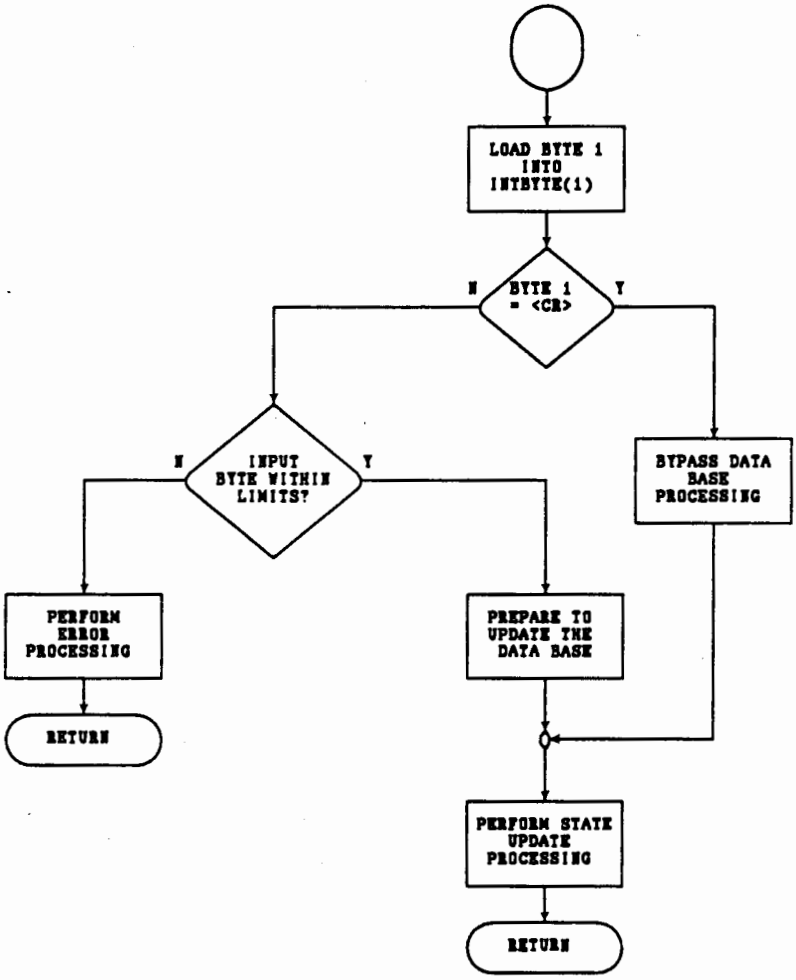


FIGURE 5.1.7.1.1-3. SINGLE DIGIT DECIMAL INPUT FLOWCHART

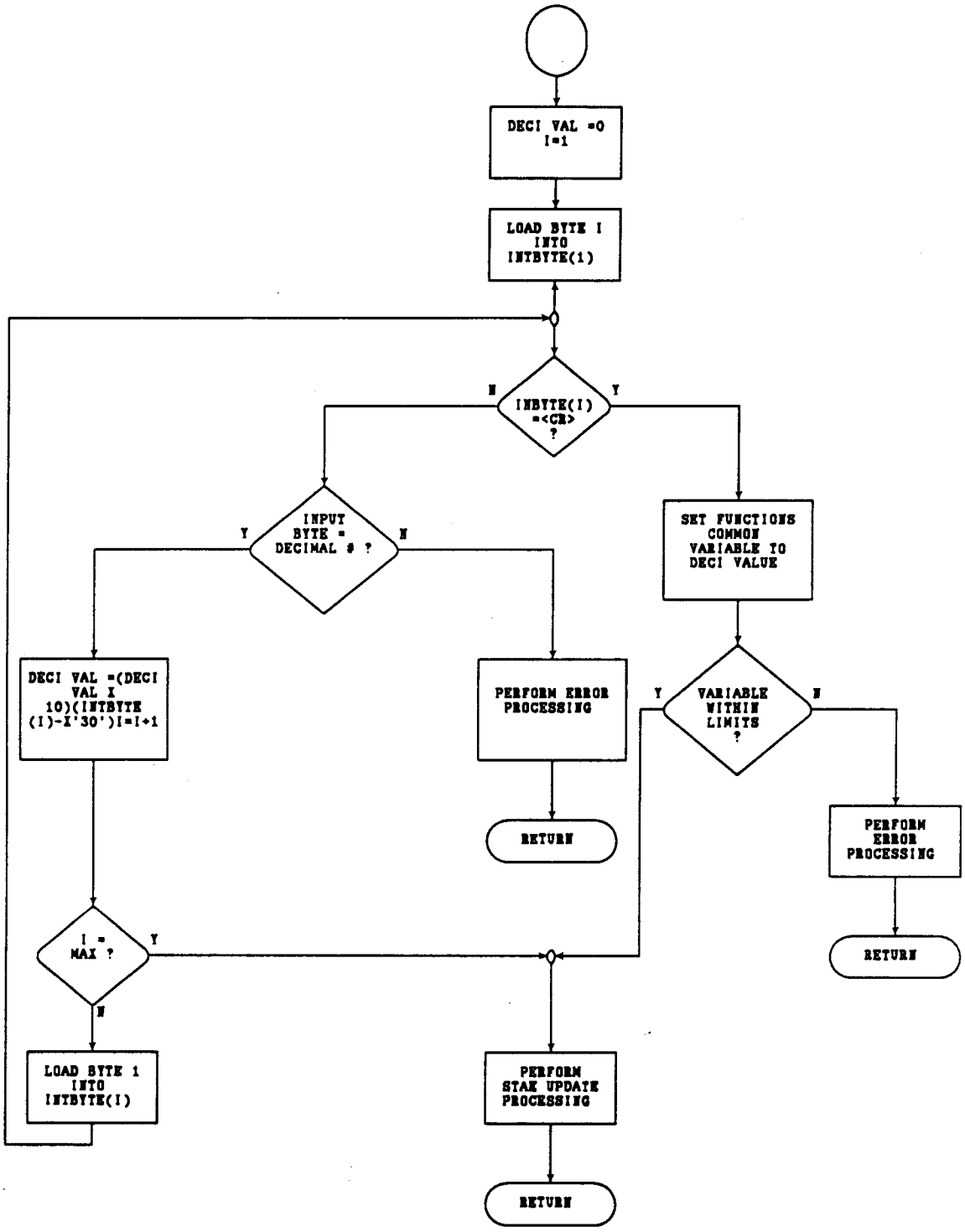


FIGURE 5.1.7.1.1-4. SEVERAL DIGIT DECIMAL INPUT FLOWCHART

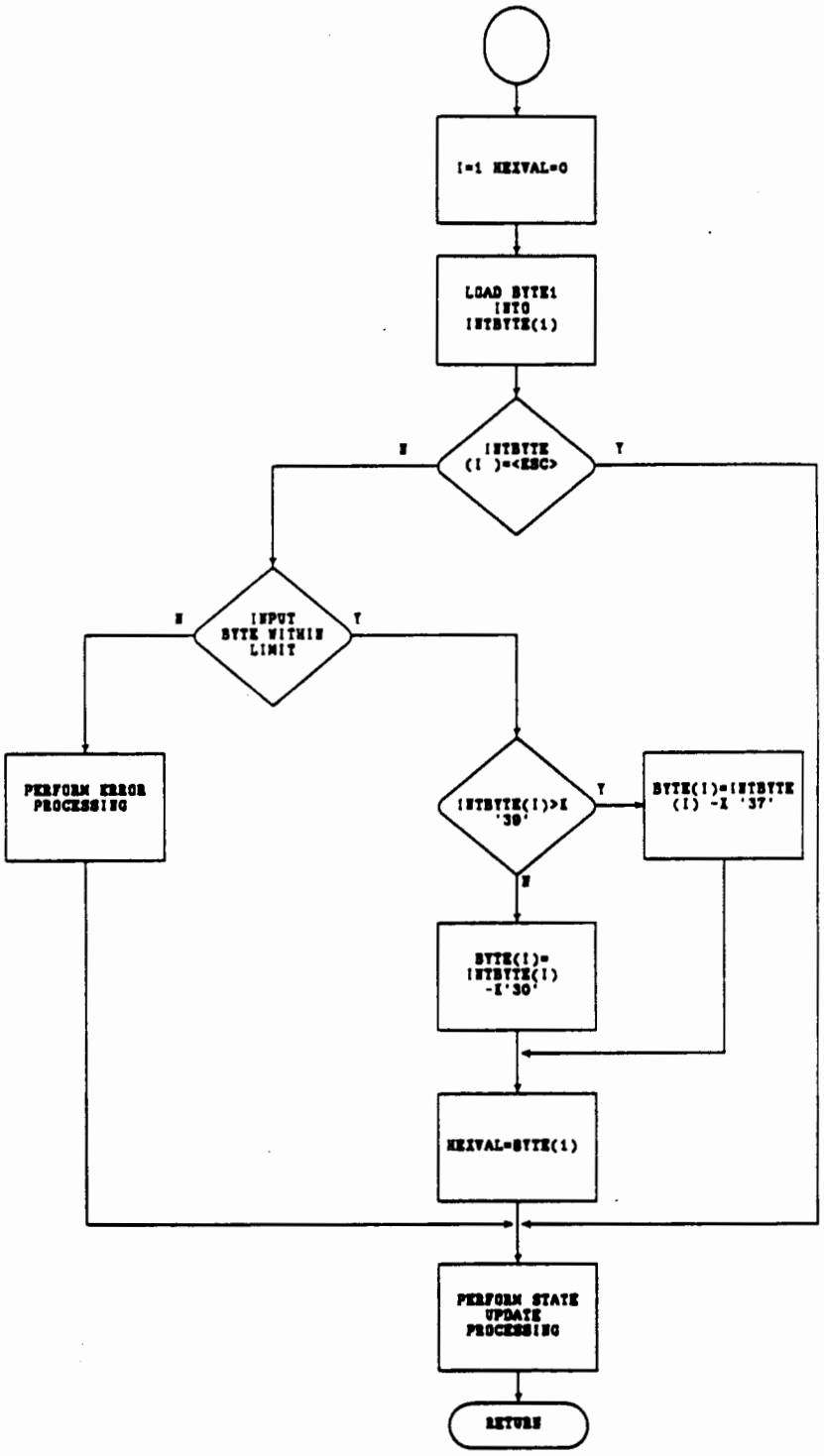


FIGURE 5.1.7.1.1-5. SINGLE DIGIT HEX INPUT FLOWCHART

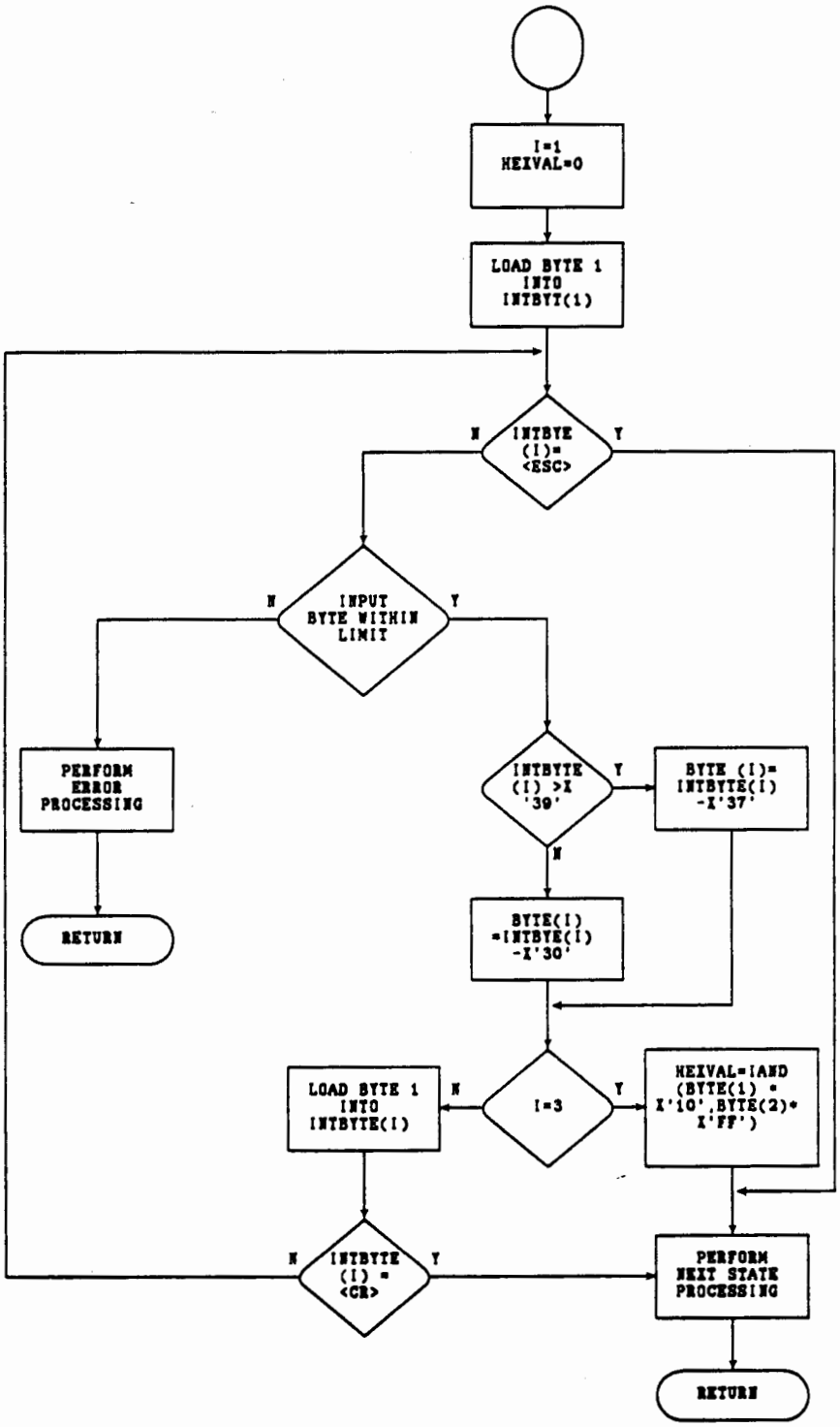


FIGURE 5.1.7.1.1-6. TWO DIGIT HEX INPUT FLOWCHART

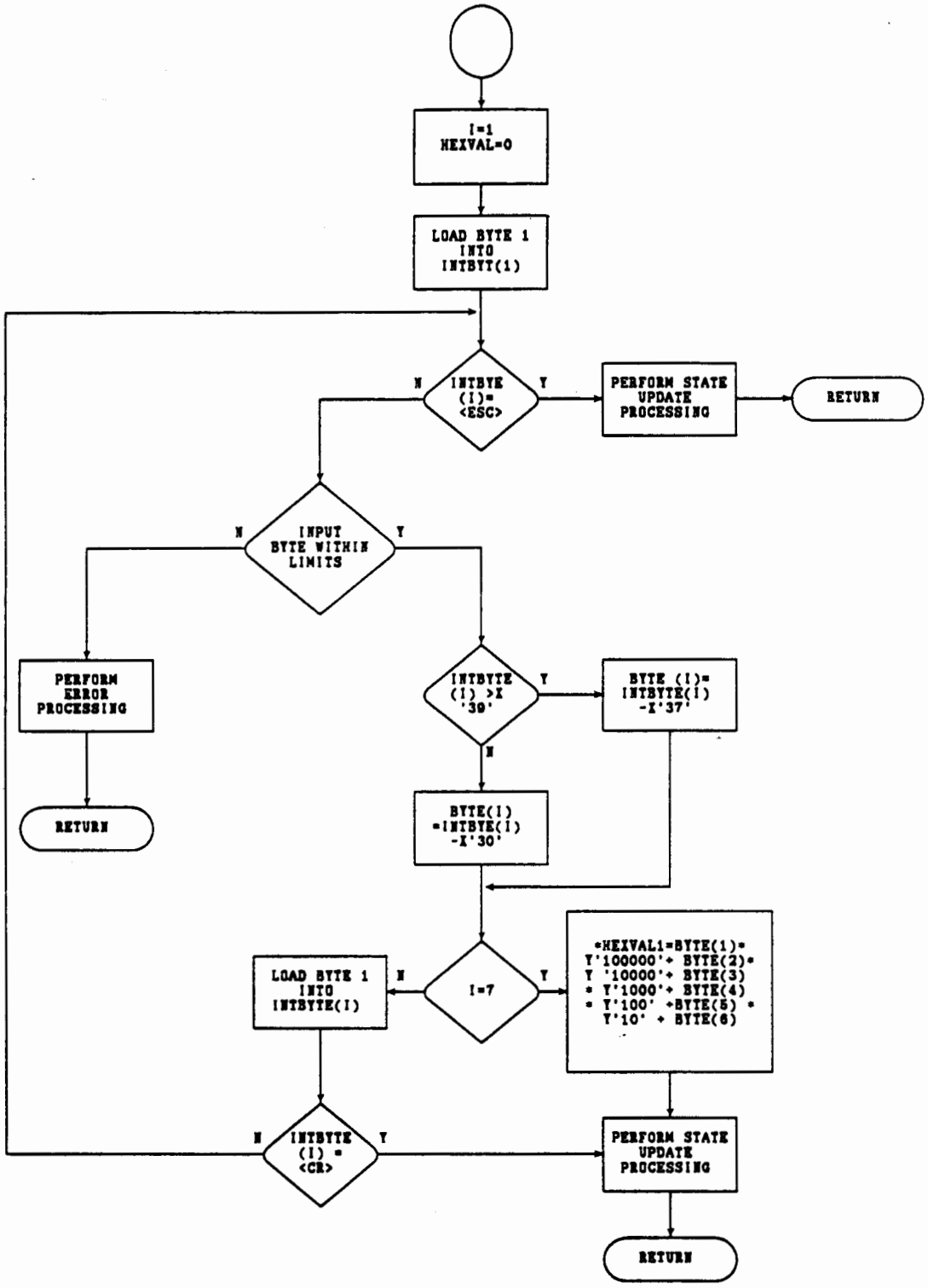


FIGURE 5.1.7.1.1-7. SIX DIGIT HEX INPUT FLOWCHART

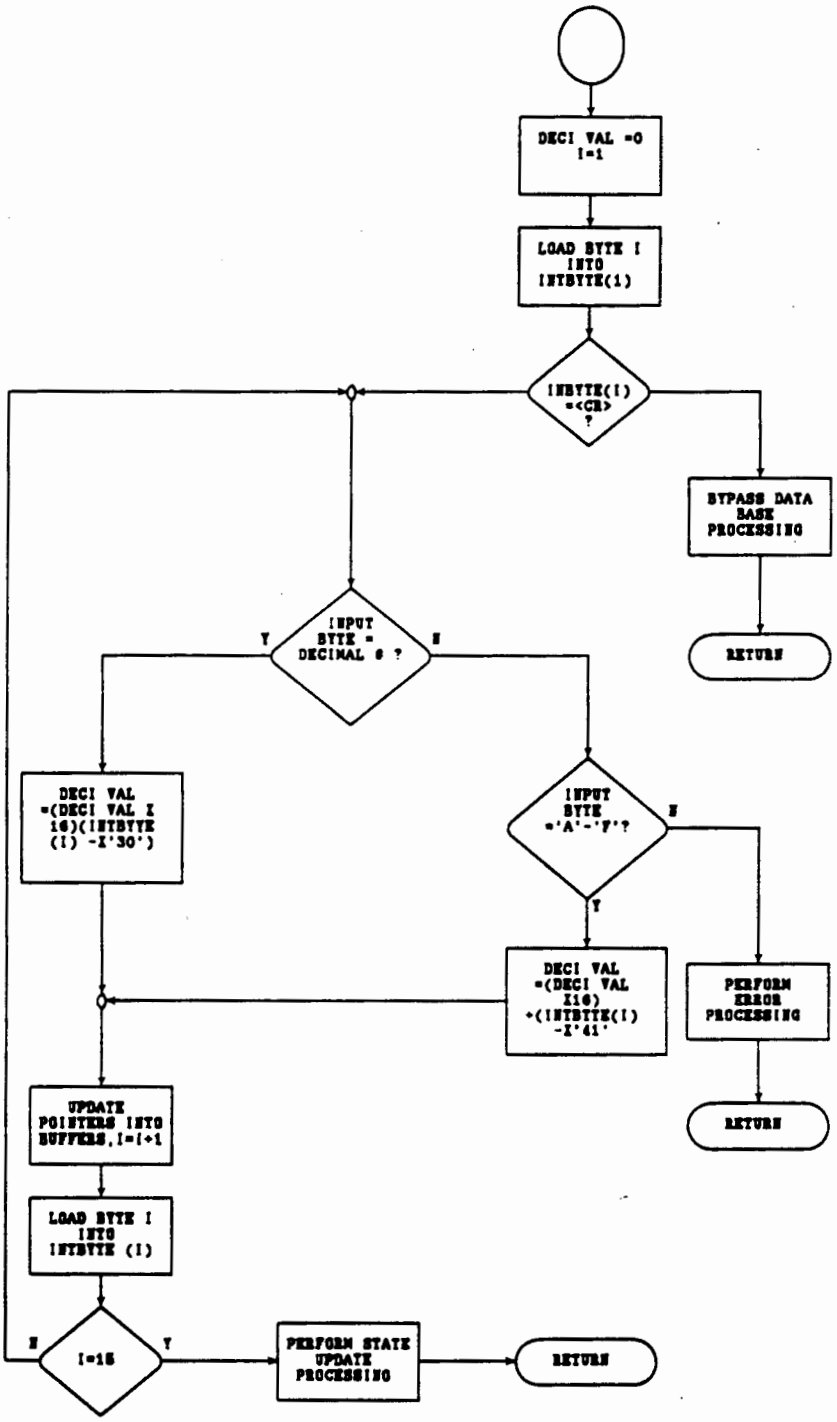


FIGURE 5.1.7.1.1-8. FOURTEEN DIGIT HEX INPUT FLOWCHART

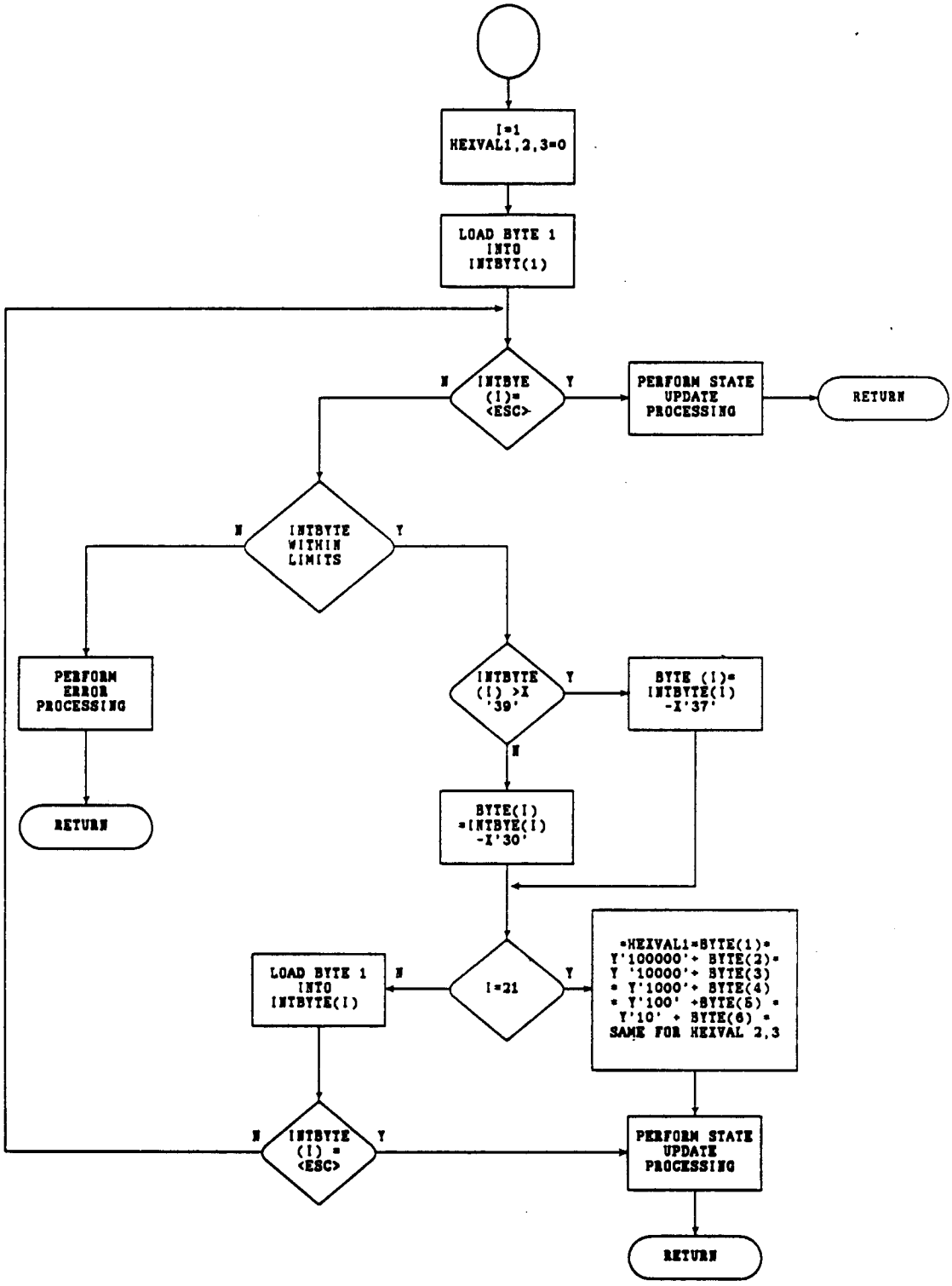


FIGURE 5.1.7.1.1-9. TWENTY DIGIT HEX INPUT FLOWCHART

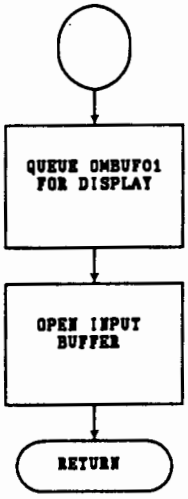


FIGURE 5.1.7.1.2-1. FLOWCHART OF OP-COMIN FUNCTION 0

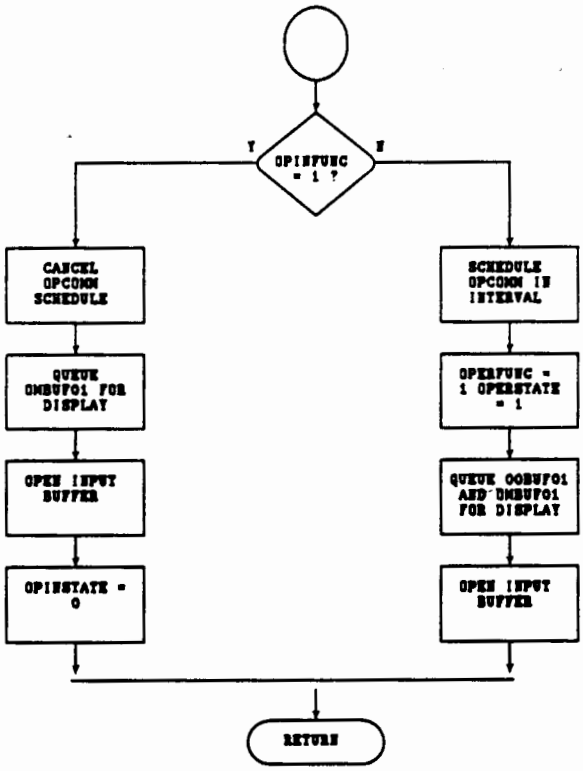


FIGURE 5.1.7.1.3-1. FLOWCHART OF OP-COMIN FUNCTION 1

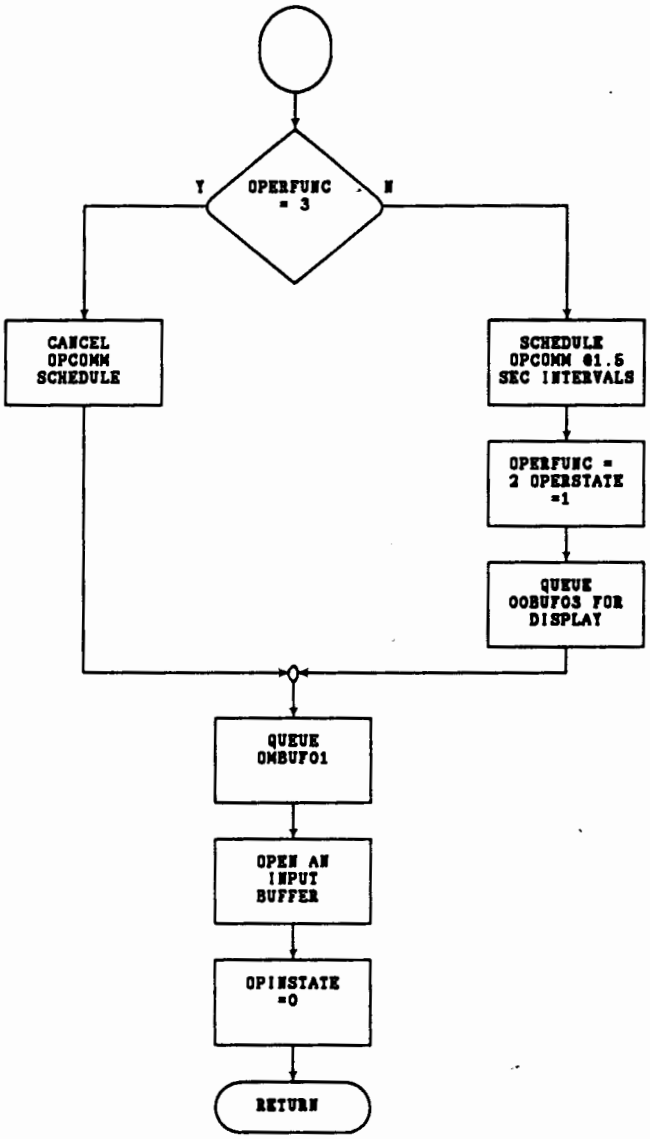


FIGURE 5.1.7.1.4-1. FLOWCHART OF OP-COMIN FUNCTION 2

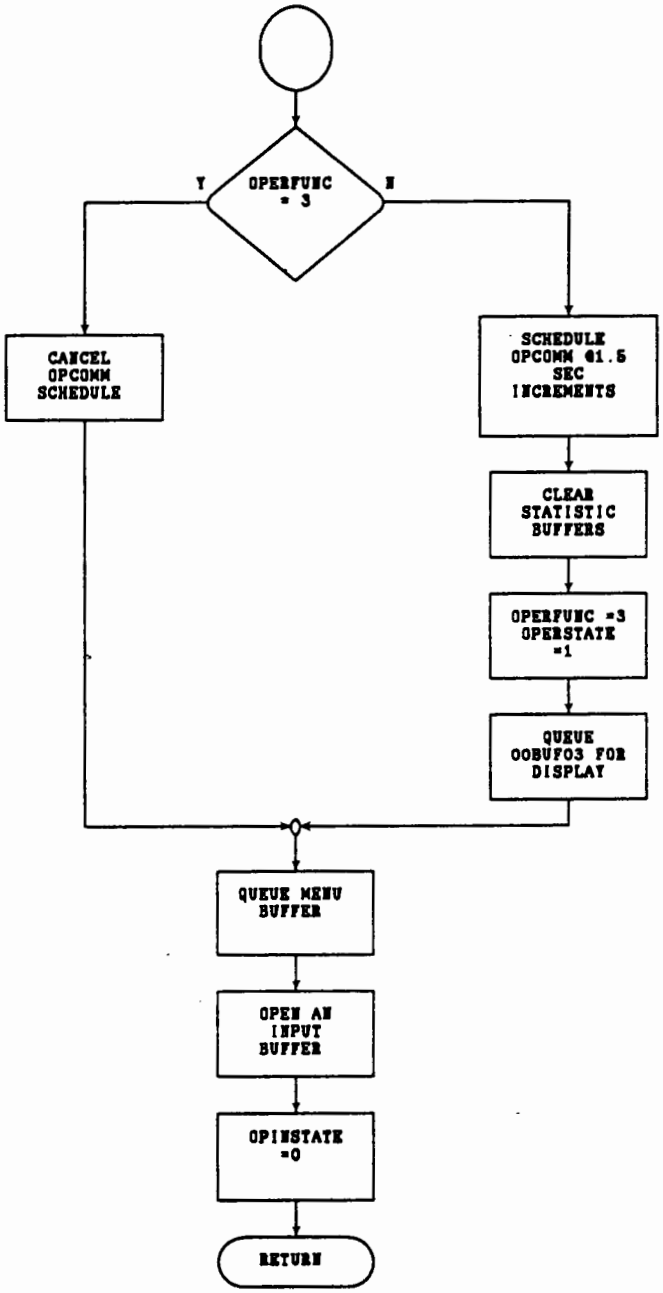


FIGURE 5.1.7.1.5-1. FLOWCHART OF OP-COMIN FUNCTION 3

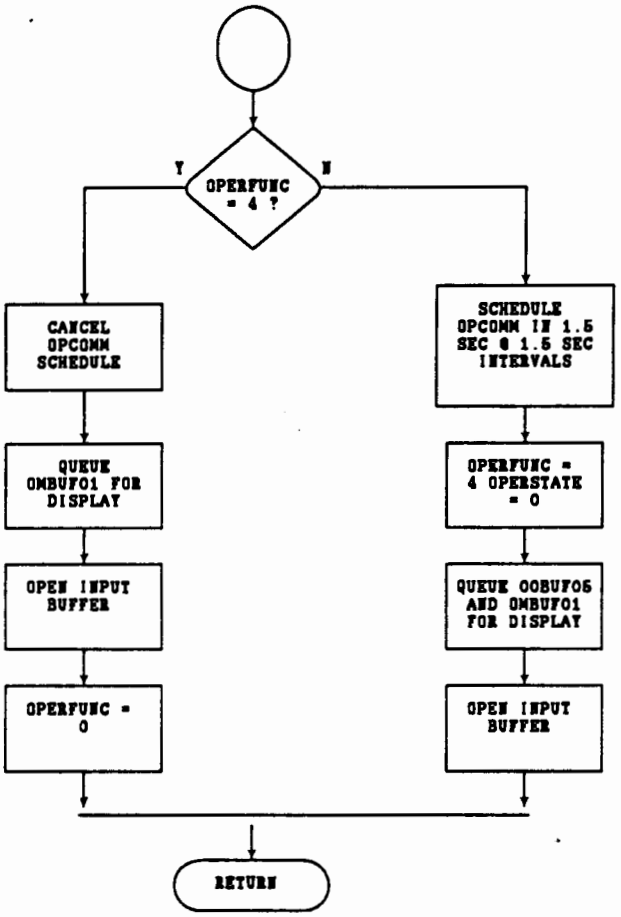


FIGURE 5.1.7.1.6-1. FLOWCHART OF OP-COMIN FUNCTION 4

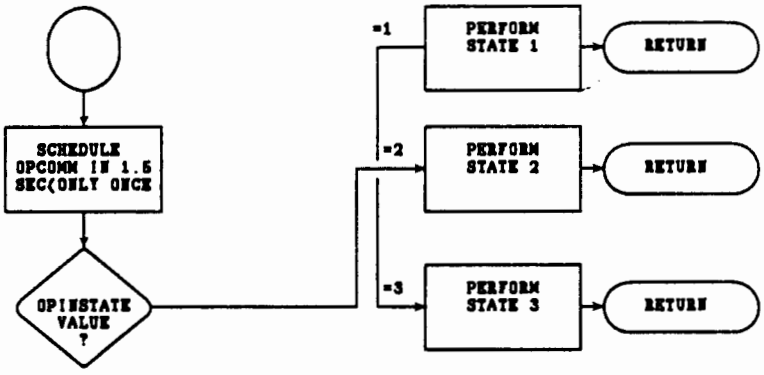


FIGURE 5.1.7.1.7-1. FLOWCHART OF OP-COMIN FUNCTION 5

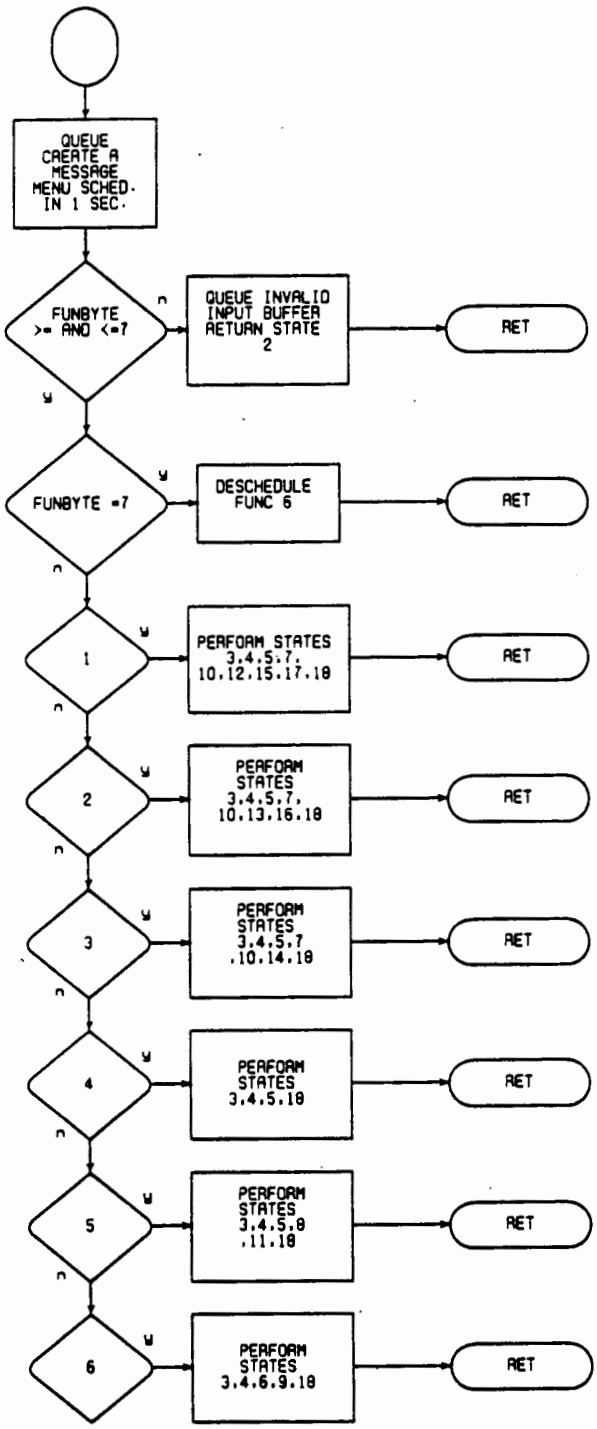


FIGURE 5.1.7.1.8-1. FLOWCHART OF OP-COMIN FUNCTION 6 (MAIN SUBROUTINE)

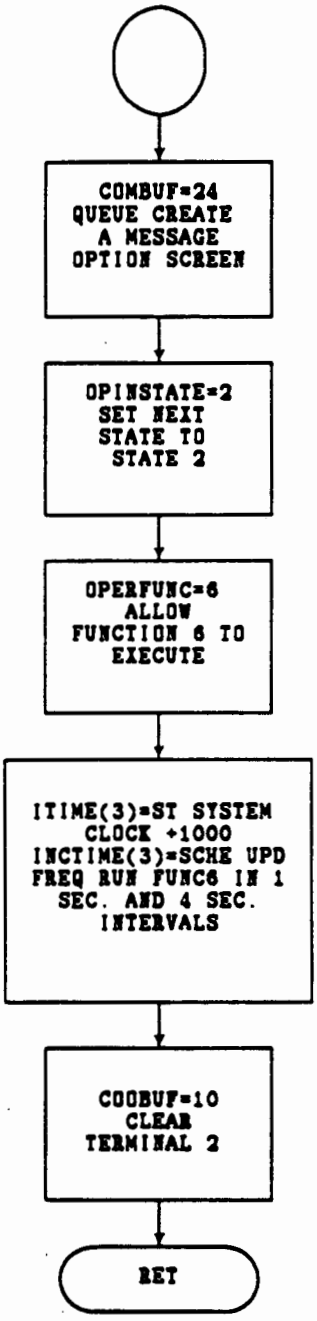


FIGURE 5.1.7.1.8-2. FLOWCHART OF OP-COMIN FUNCTION 6 (State 1)

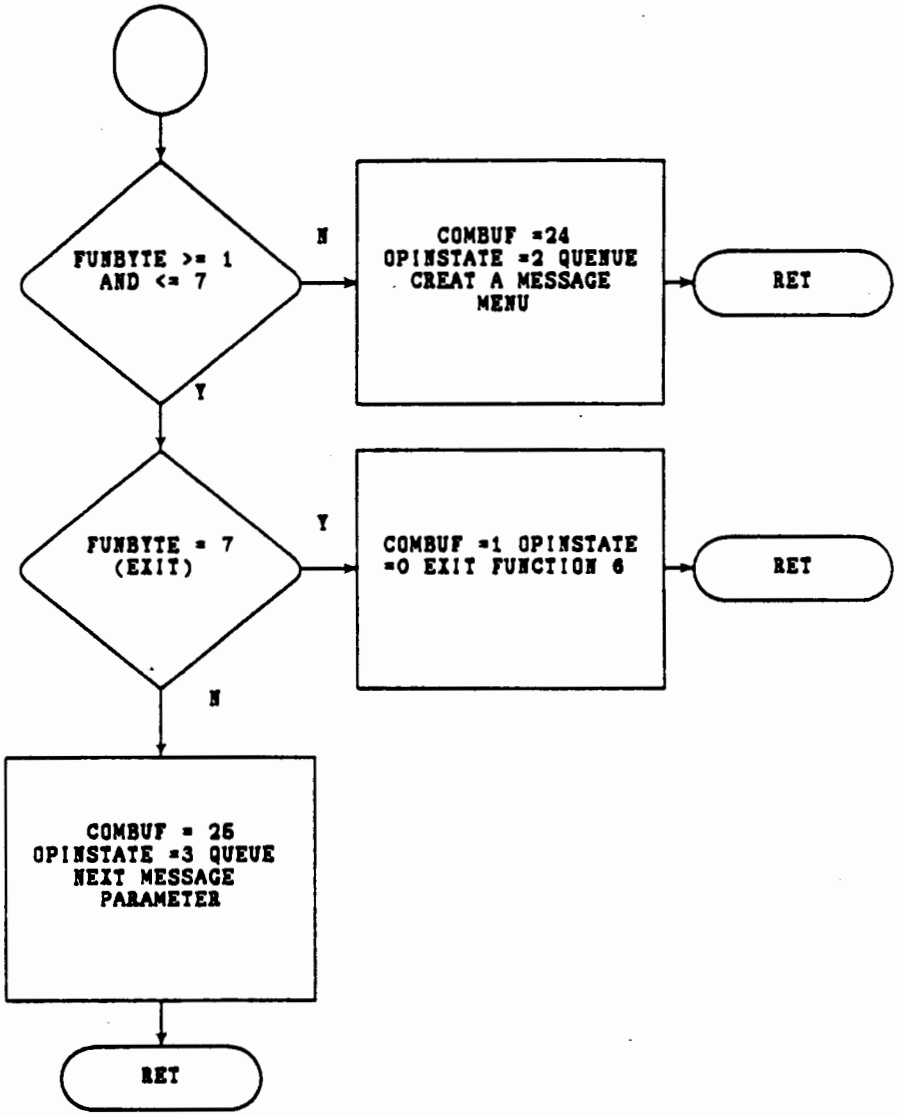


FIGURE 5.1.7.1.8-3. FLOWCHART OF OP-COMIN FUNCTION 6 (State 2)

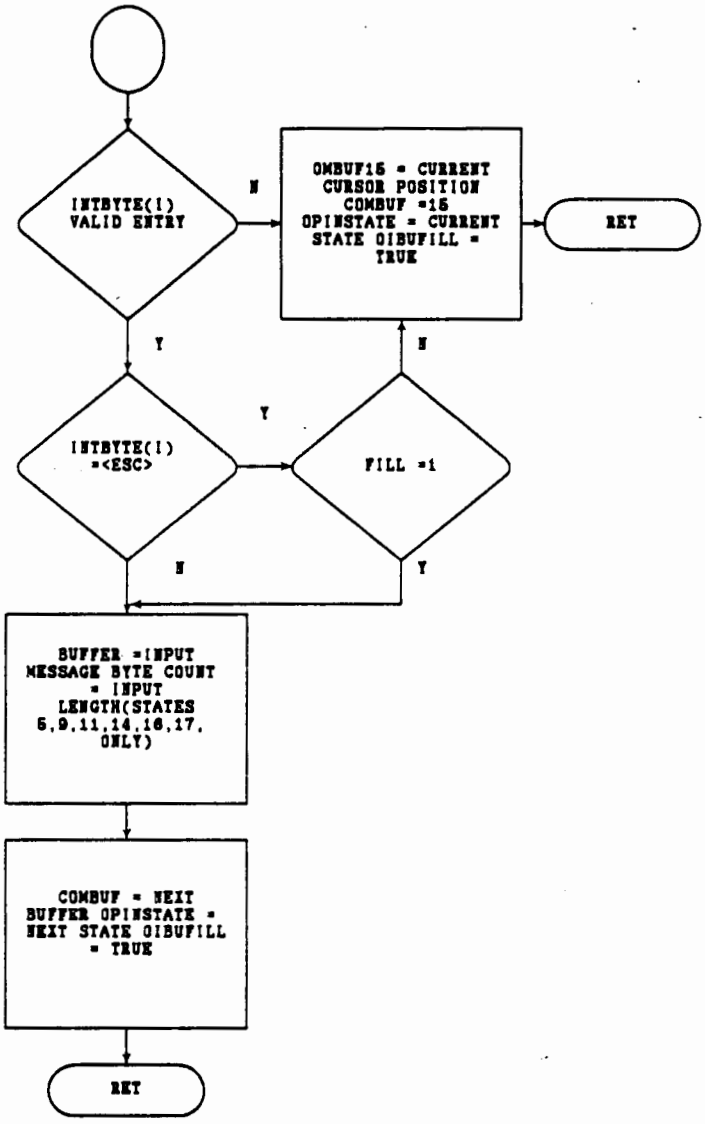


FIGURE 5.1.7.1.8-4. FLOWCHART OF OP-COMIN FUNCTION 6 (States 3-17)

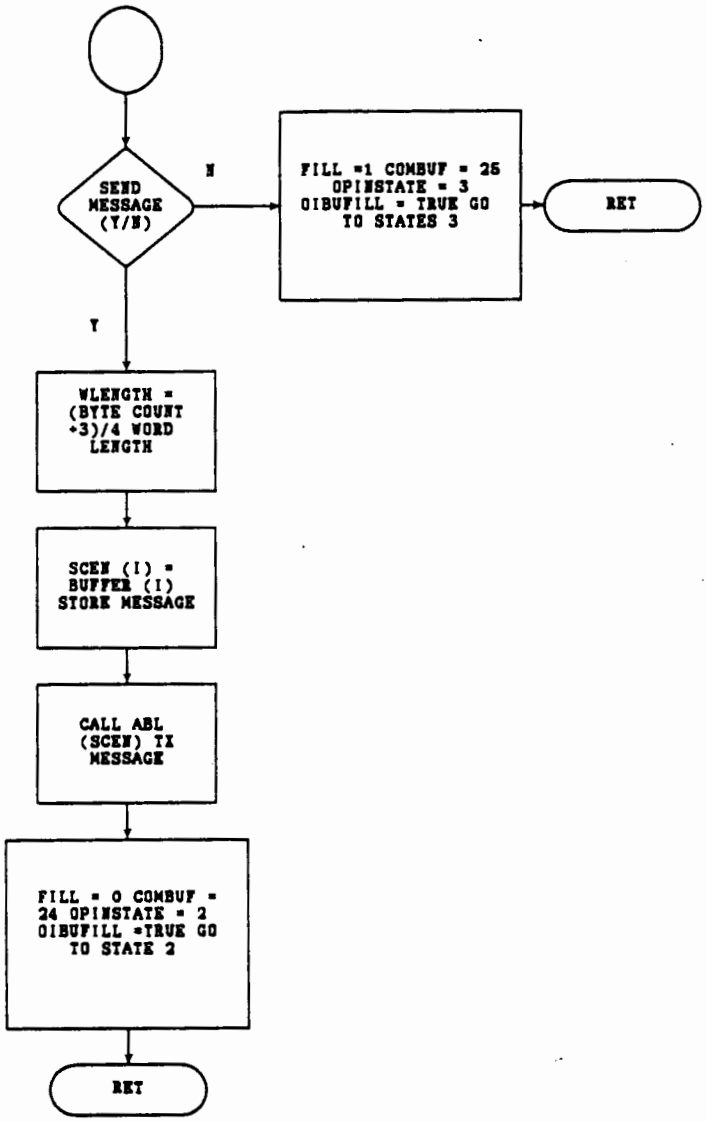


FIGURE 5.1.7.1.8-5. FLOWCHART OF OP-COMIN FUNCTION 6 (State 18)

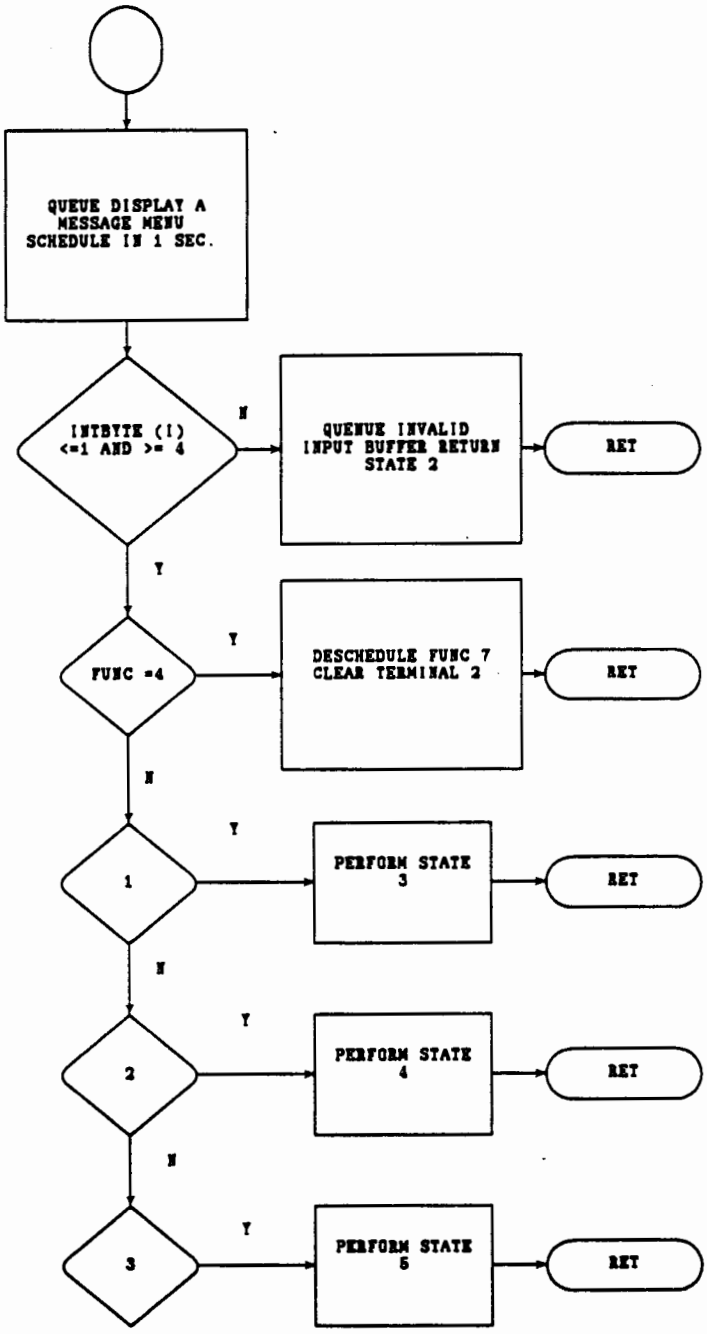


FIGURE 5.1.7.1.9-1. FLOWCHART OF OP-COMIN FUNCTION 7 (MAIN SUBROUTINE)

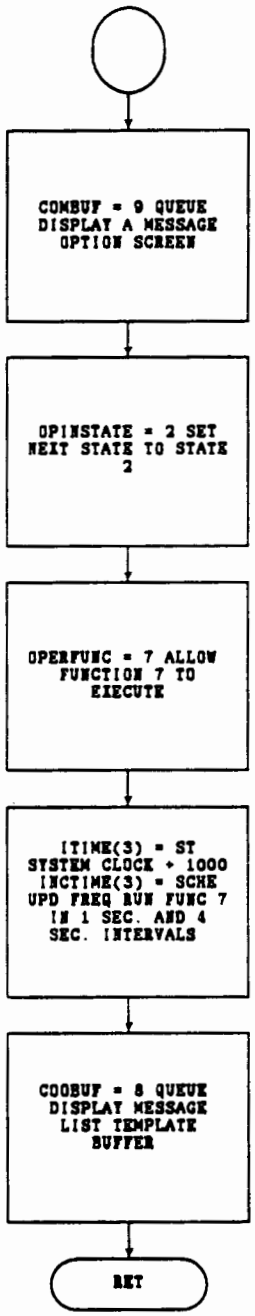


FIGURE 5.1.7.1.9-2. FLOWCHART OF OP-COMIN FUNCTION 7 (State 1)

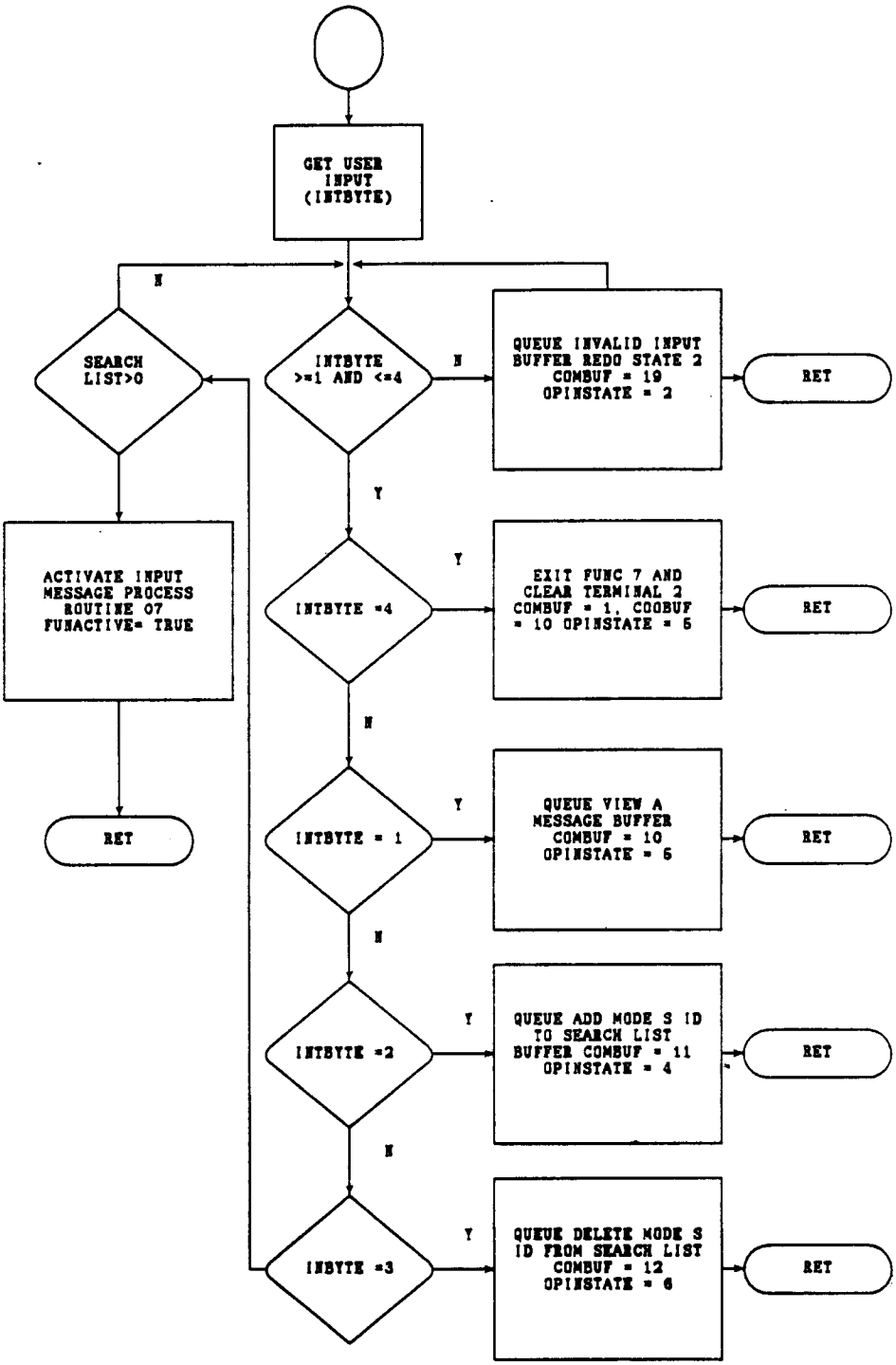


FIGURE 5.1.7.1.9-3. FLOWCHART OF OP-COMIN FUNCTION 7 (State 2)

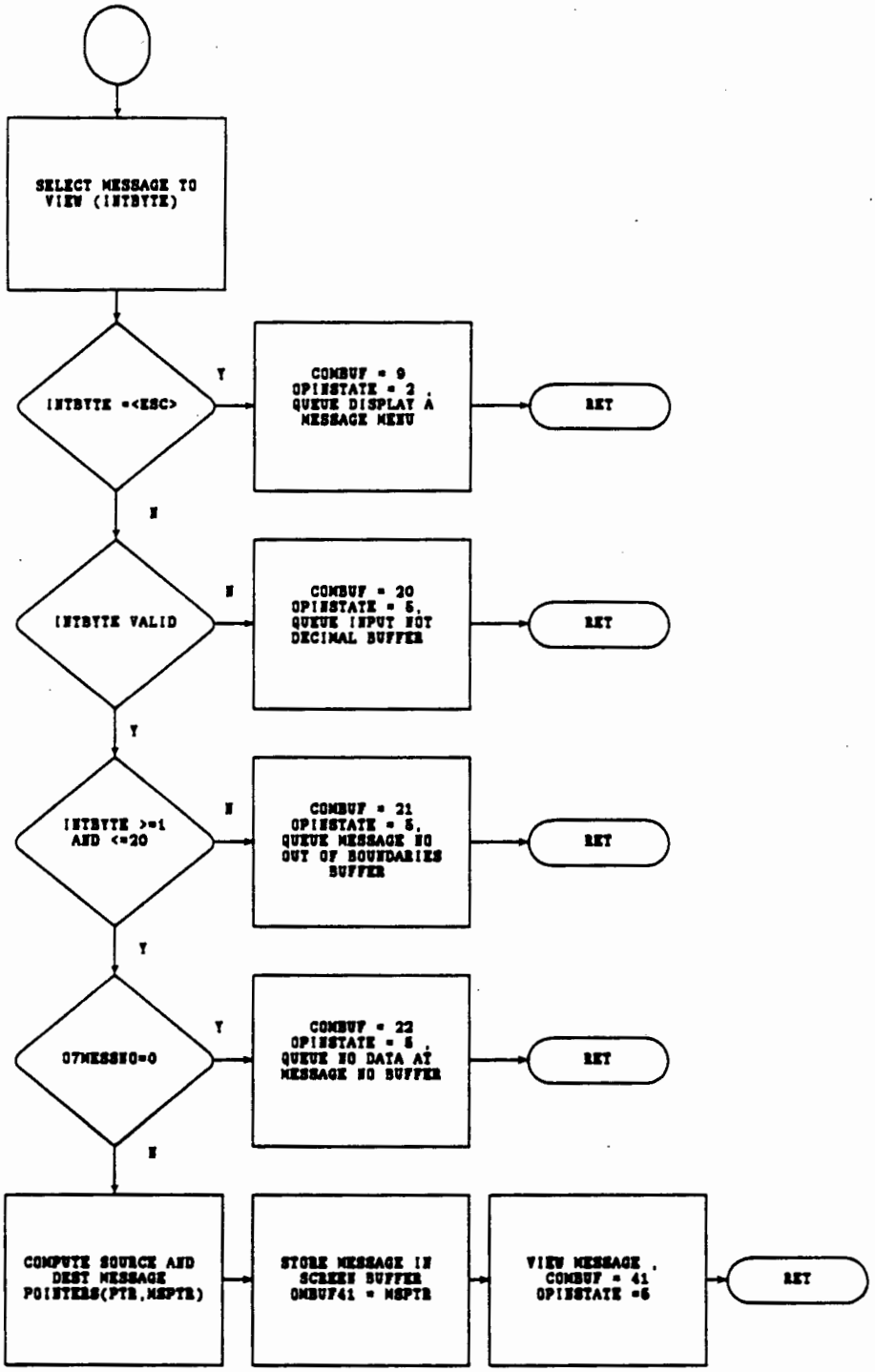


FIGURE 5.1.7.1.9-4. FLOWCHART OF OP-COMIN FUNCTION 7 (State 3)

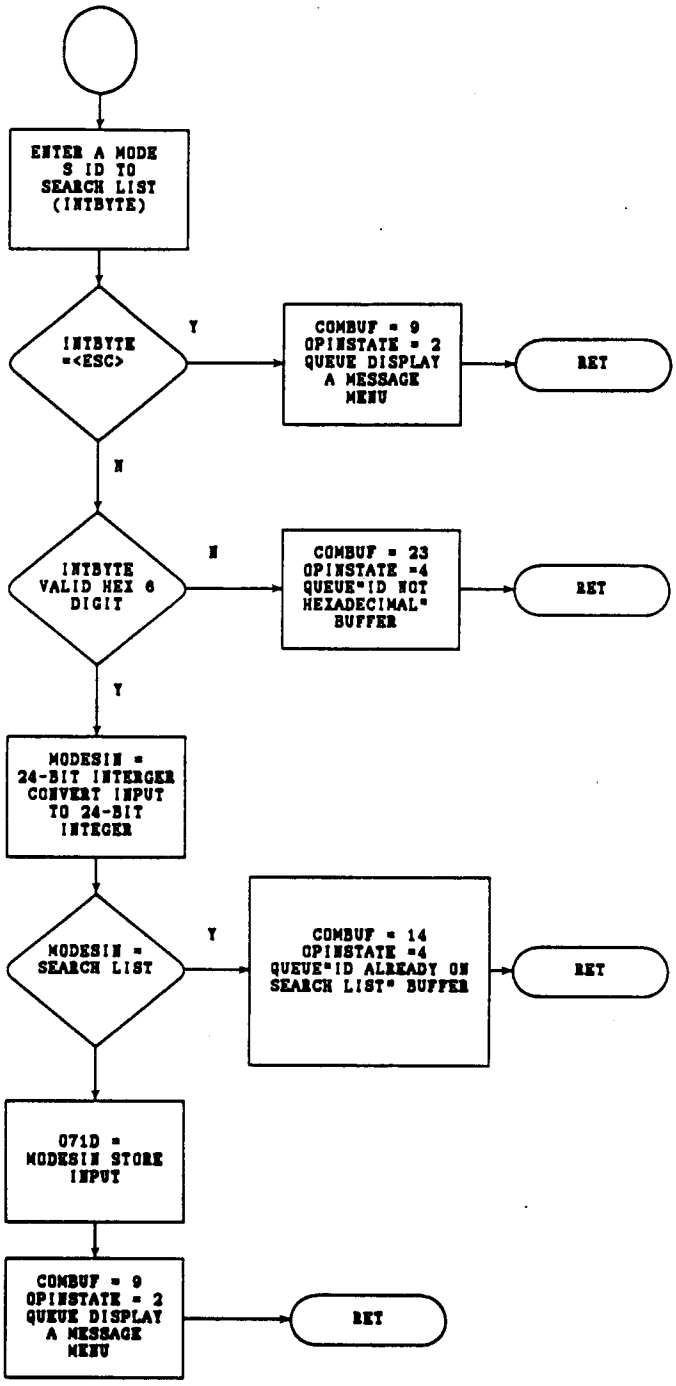


FIGURE 5.1.7.1.9-5. FLOWCHART OF OP-COMIN FUNCTION 7 (State 4)

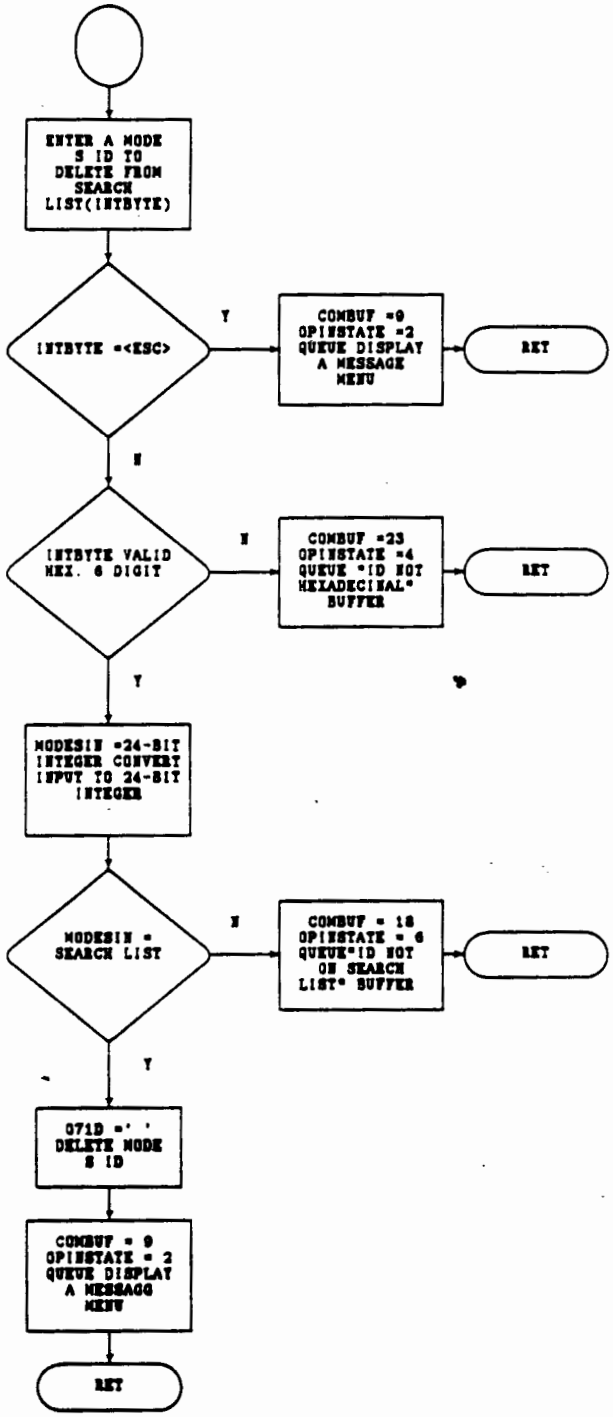


FIGURE 5.1.7.1.9-6. FLOWCHART OF OP-COMIN FUNCTION 7 (State 5)

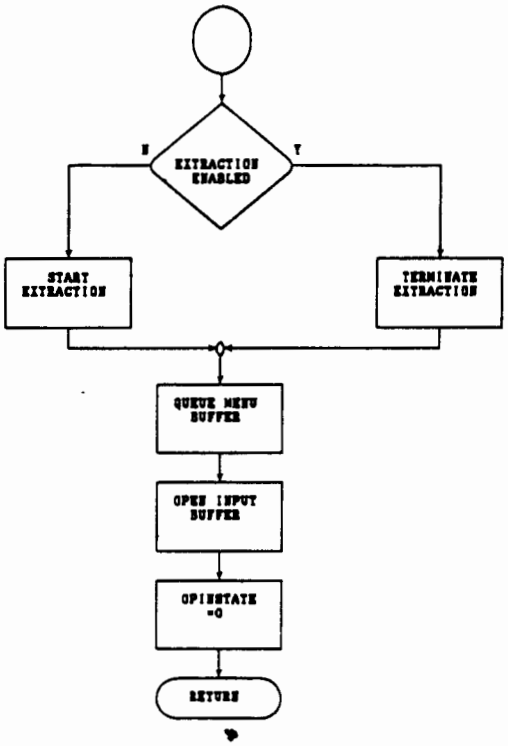


FIGURE 5.1.7.1.10-1. FLOWCHART OF OP-COMIN FUNCTION 8

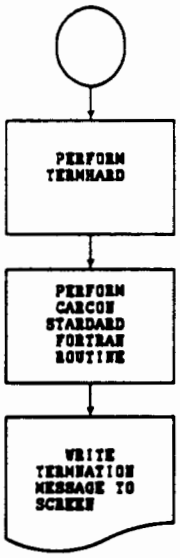


FIGURE 5.1.7.1.11-1. FLOWCHART OF OP-COMIN FUNCTION 9

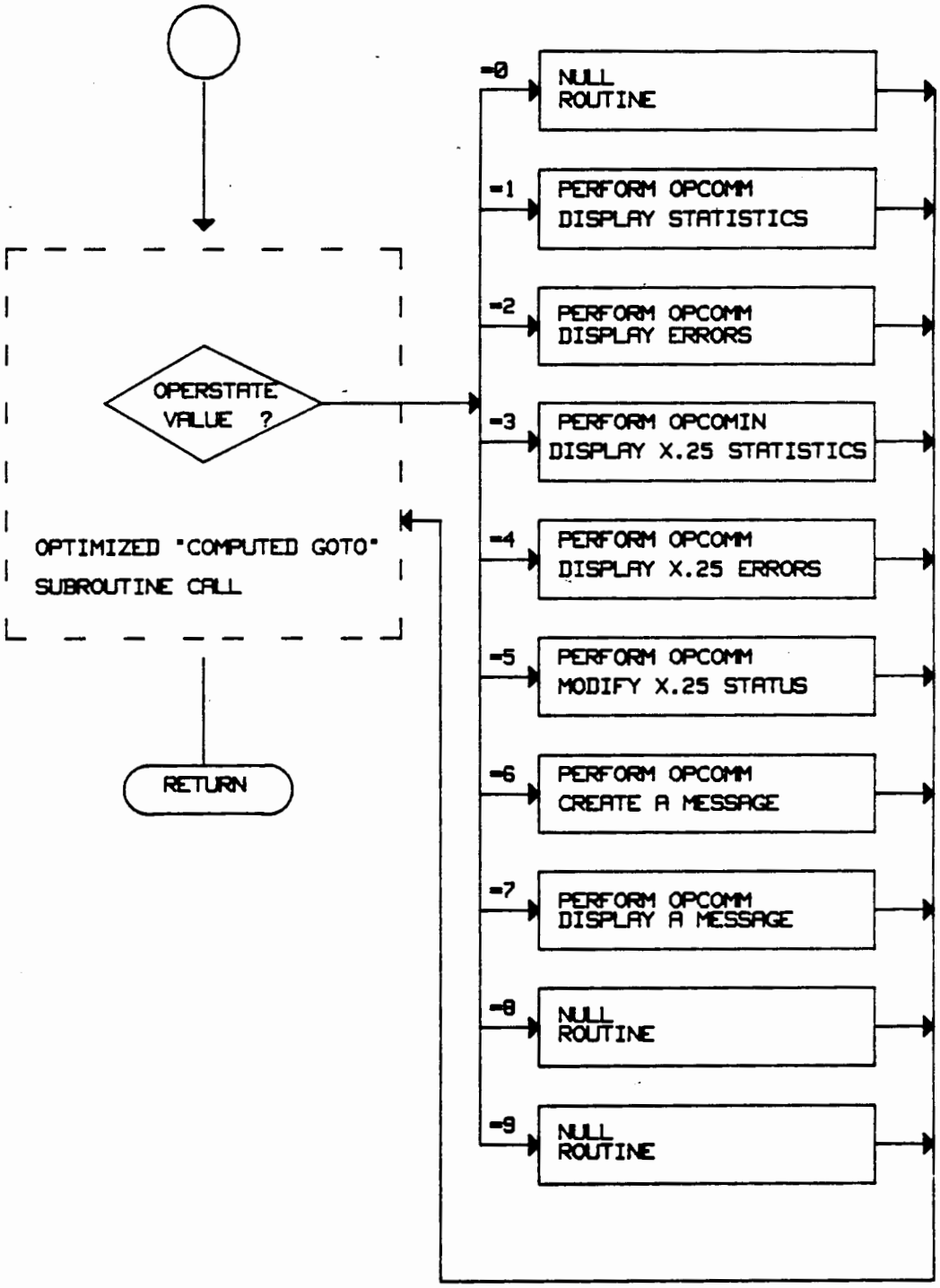


FIGURE 5.1.7.2-1. HIGH LEVEL FLOWCHART OF OP-COMM

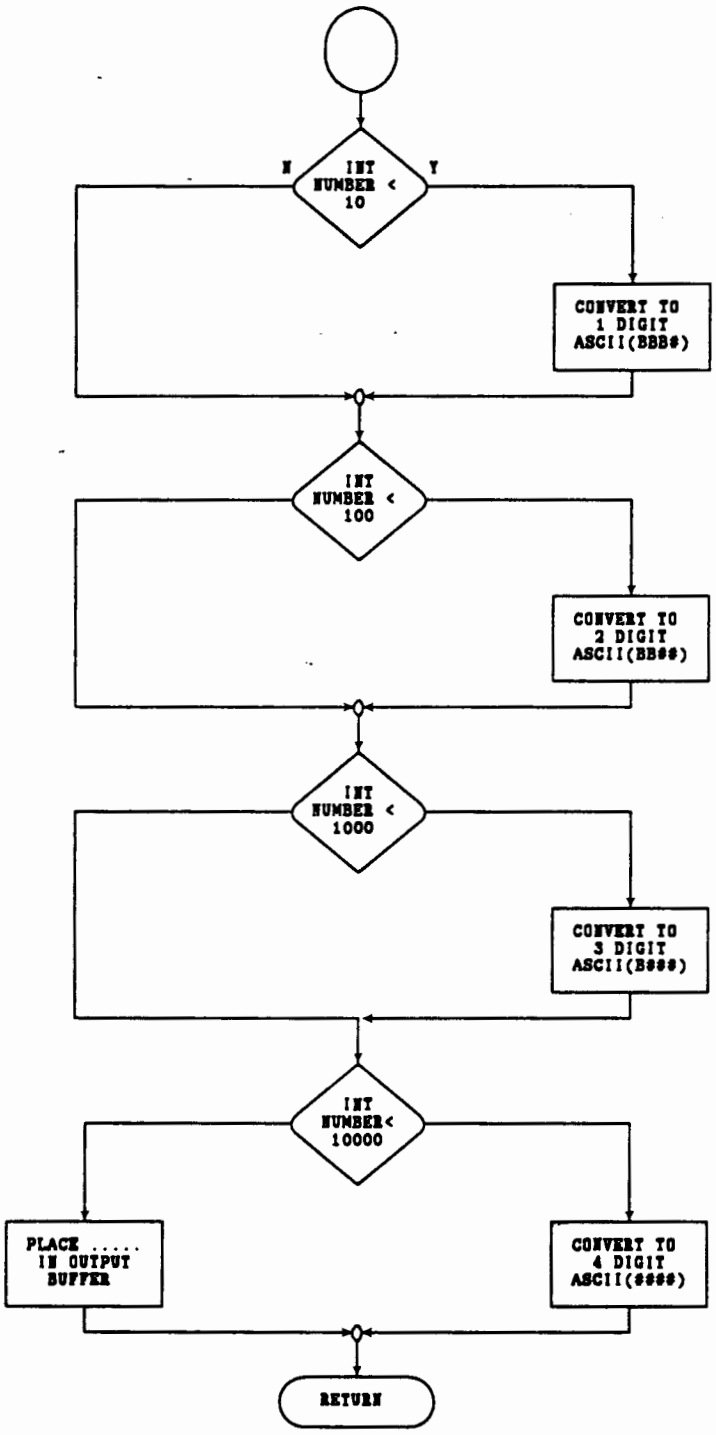


FIGURE 5.1.7.2.1-1. FLOWCHART OF OP-COMMS TOASCII ALGORITHM

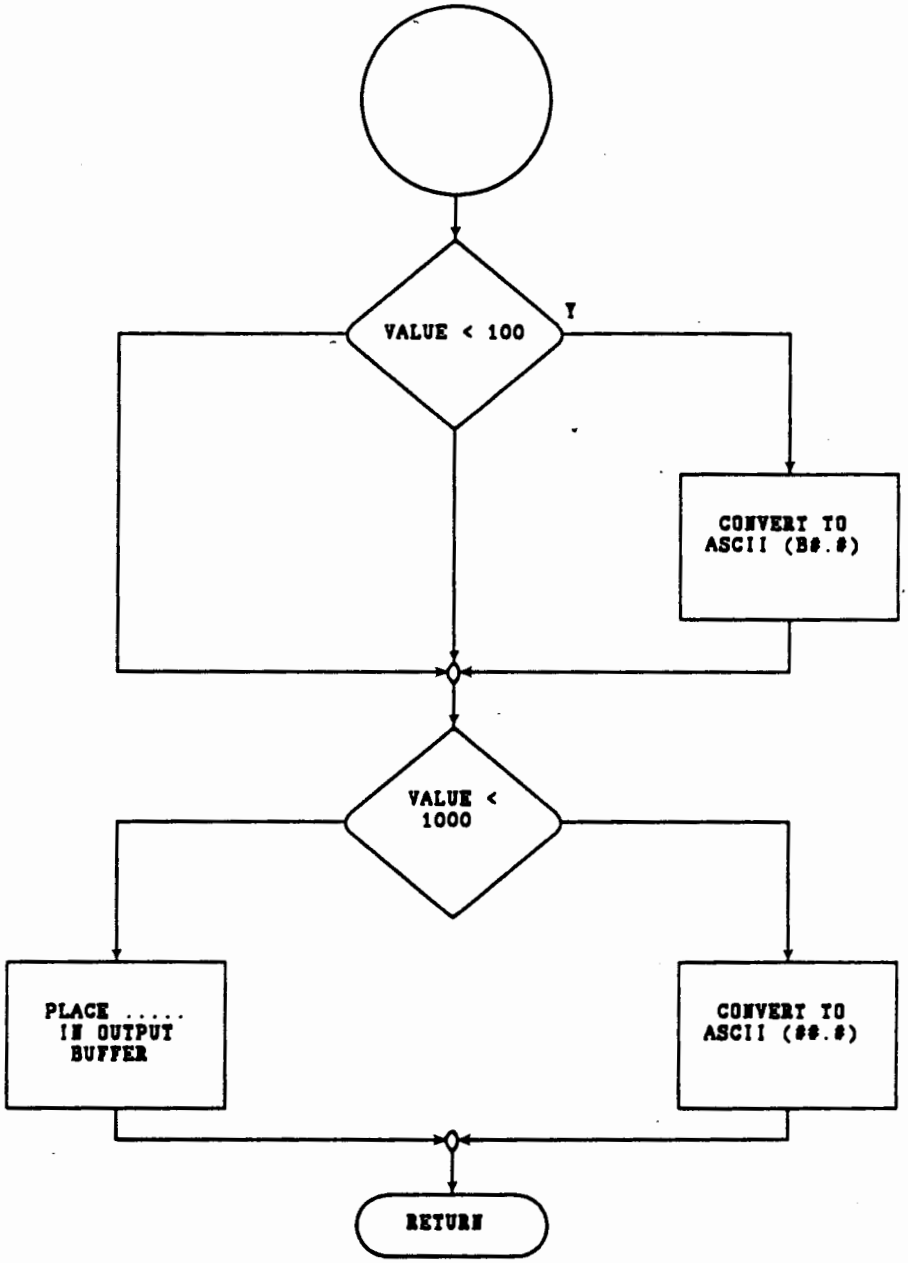


FIGURE 5.1.7.2.1-2. FLOWCHART OF OP-COMMS DECIMAL TOASCII ALGORITHM

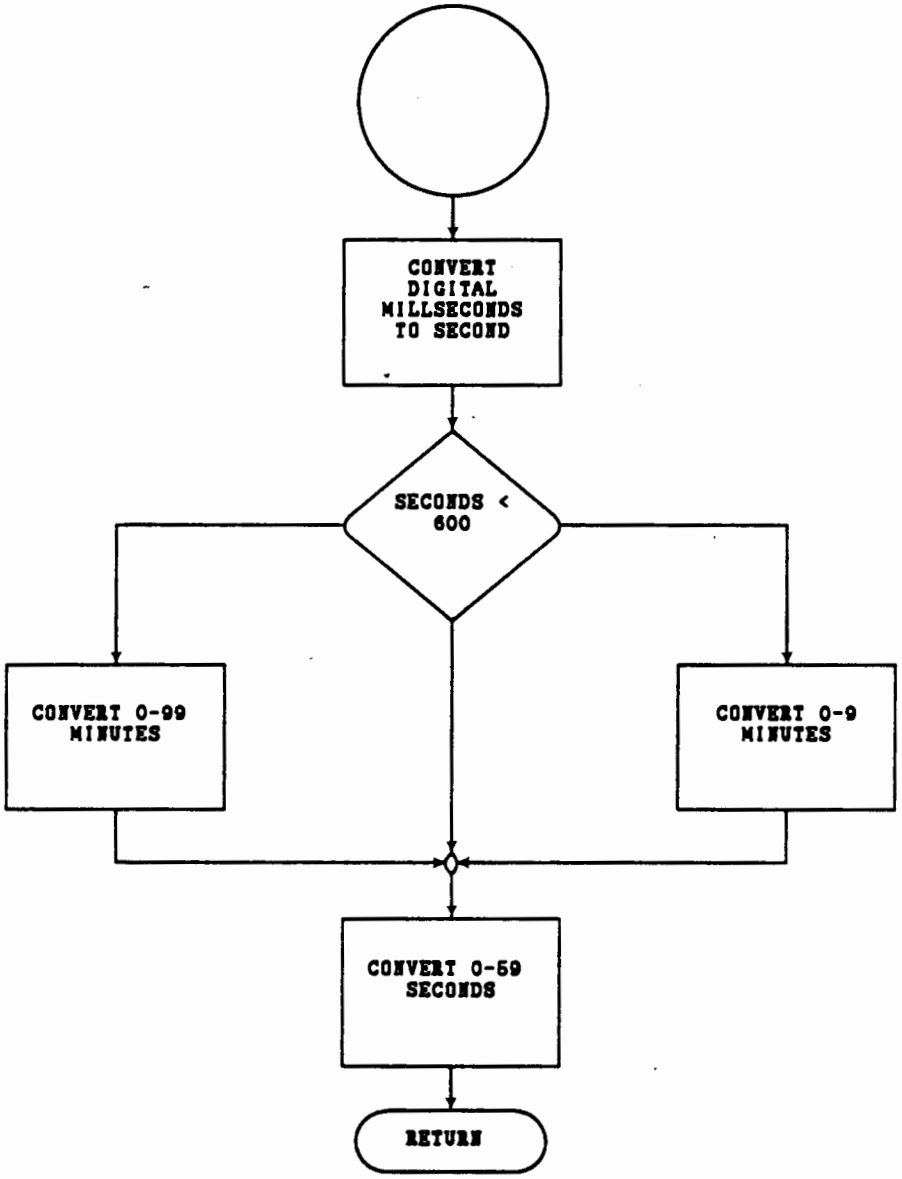


FIGURE 5.1.7.2.1-3. FLOWCHART OF OP-COMMS TIME OUTPUT ALGORITHM

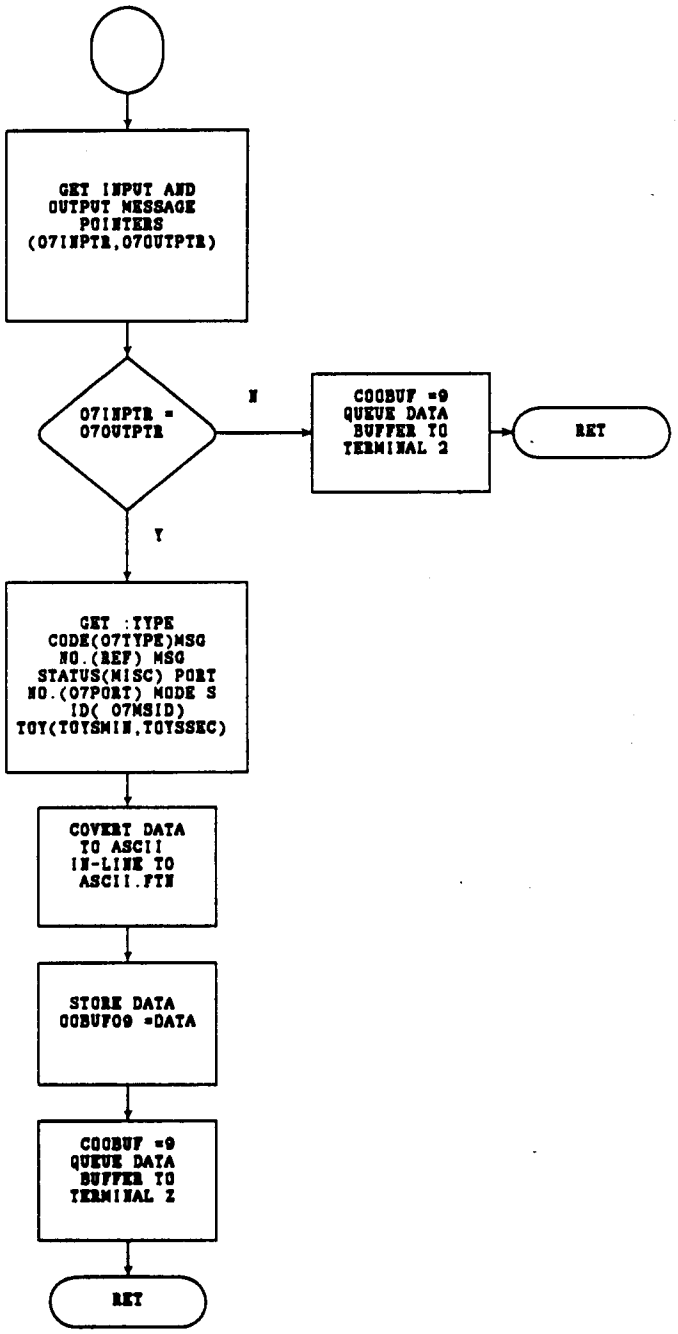


FIGURE 5.1.7.2.9-1. FLOWCHART OF OP-COMM FUNCTION 7 (SUBROUTINE)

BUFFER			DESCRIPTION
00BUF01	-	System Status Template	
00BUF02	-	System Status Data	
00BUF03	-	System Error Template	
00BUF04	-	System Error Data	
00BUF05	-	X.25 Status & Error Template	
00BUF06	-	X.25 Status Data	
00BUF07	-	X.25 Error Data	
00BUF08	-	Display Message Template	
00BUF09	-	Display Message Data	
00BUF10	-	Create a Message Screen Clear	

FIGURE 5.1.7.3.1-1. LIST OF OPBLKDTA DISPLAY SCREEN BUFFERS

```
(1)     DATA 00BUF01(   1:8   )/Z1B48000000000000/
(2)     DATA 00BUF01(   9:14  )/Z1B58211B5936/
(3)     DATA 00BUF01(  15:51  )/'     C I D  S Y S T E M  S T A T U S'/
(4)     DATA 00BUF01(  52:57  )/Z1B58211B5936/
(5)     DATA 00BUF01(  58:104 )/'                                        '/
(6)     DATA 00BUF01( 105:110 )/Z1B58241B5925/
(7)     DATA 00BUF01( 111:143 )/' % CPU UTILIZATION                      '/
(8)     DATA 00BUF01( 144:149 )/Z1B58251B5925/
(9)     DATA 00BUF01( 150:182 )/' % DATA EXTRACTION UTILIZATION          '/
          .     .       .   .                    .
          .     .       .   .                    .
          .     .       .   .                    .
(10)    DATA 00BUF01(1002:1007)/Z1B58361B5950/
(11)    DATA 00BUF01(1008:1023)/' - TIME OF YEAR                         '/
```

FIGURE 5.1.7.3.1-2. TEMPLATE BUFFER EXAMPLE

BUFFER	DESCRIPTION
0MBUF01	Main Menu Buffer
0MBUF02	Terminate Question
0MBUF03	Modify X.25 Main Query
0MBUF04	Modify X.25 Main Query Error Buffer
0MBUF05	Device is not Configured Prompt
0MBUF06	Deactivate Device Query
0MBUF07	Activate Device Query
0MBUF08	Main Menu Input Error Prompt
0MBUF09	Display Message List Option Menu
0MBUF10	View A Message Option Menu
0MBUF11	Add To Search List Screen
0MBUF12	Delete from Search List Screen
0MBUF13	List is Full Error Prompt
0MBUF14	Id Already Exists Error Prompt
0MBUF15	Create a Message Error Prompt
0MBUF17	Search List Empty Error Prompt
0MBUF18	ID does not Exist Error Prompt
0MBUF19	Display Message List Input Error Prompt
0MBUF20	View a Message Input Error
0MBUF21	View a Message Out of Bound Error Prompt
0MBUF22	View a Message No Data Error Message
0MBUF23	Add/Delete ID Error Message
0MBUF24	Create a Message Menu
0MBUF25	Create a Message Number Query
0MBUF26	Create a Message X.25 Port Number Query
0MBUF27	Create a Message Mode S Address Query
0MBUF28	Create a Message Priority Query
0MBUF29	Create a Message Expiration Query
0MBUF30	Create a Message Acknowledgement Query
0MBUF31	Create a Message Segment Count Query
0MBUF32	Create a Message Comm-A Text Query
0MBUF33	Create a Message Number of Segments Query
0MBUF34	Create a Message ELM Default Text Query
0MBUF35	Create a Message BDS Query
0MBUF36	Create a Message Reference Message # Query
0MBUF37	Create a Message Reference Type Code Query
0MBUF38	Create a Message Length Query
0MBUF39	Create a Message Bit Stream Query
0MBUF40	Create a Message Send Message Query
0MBUF41	Display Message List View Message Buffer

FIGURE 5.1.7.3.2-1. MENU SCREEN BUFFER LIST

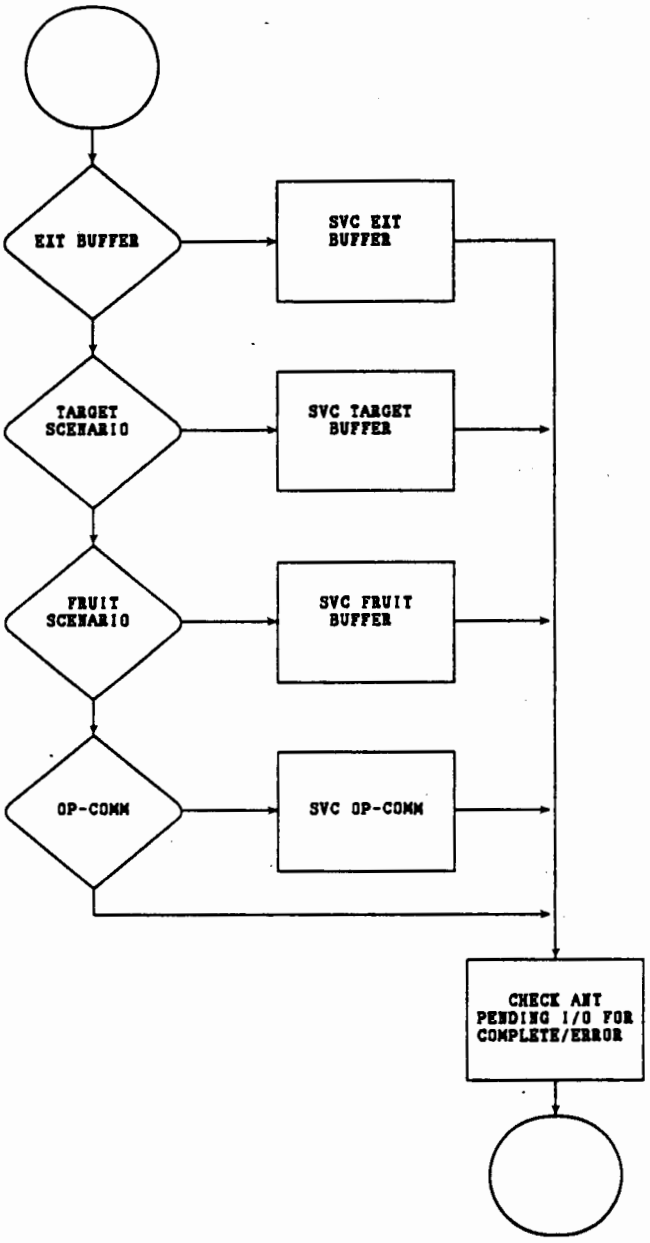


FIGURE 5.1.8-1. I/O PROGRAM FLOWCHART

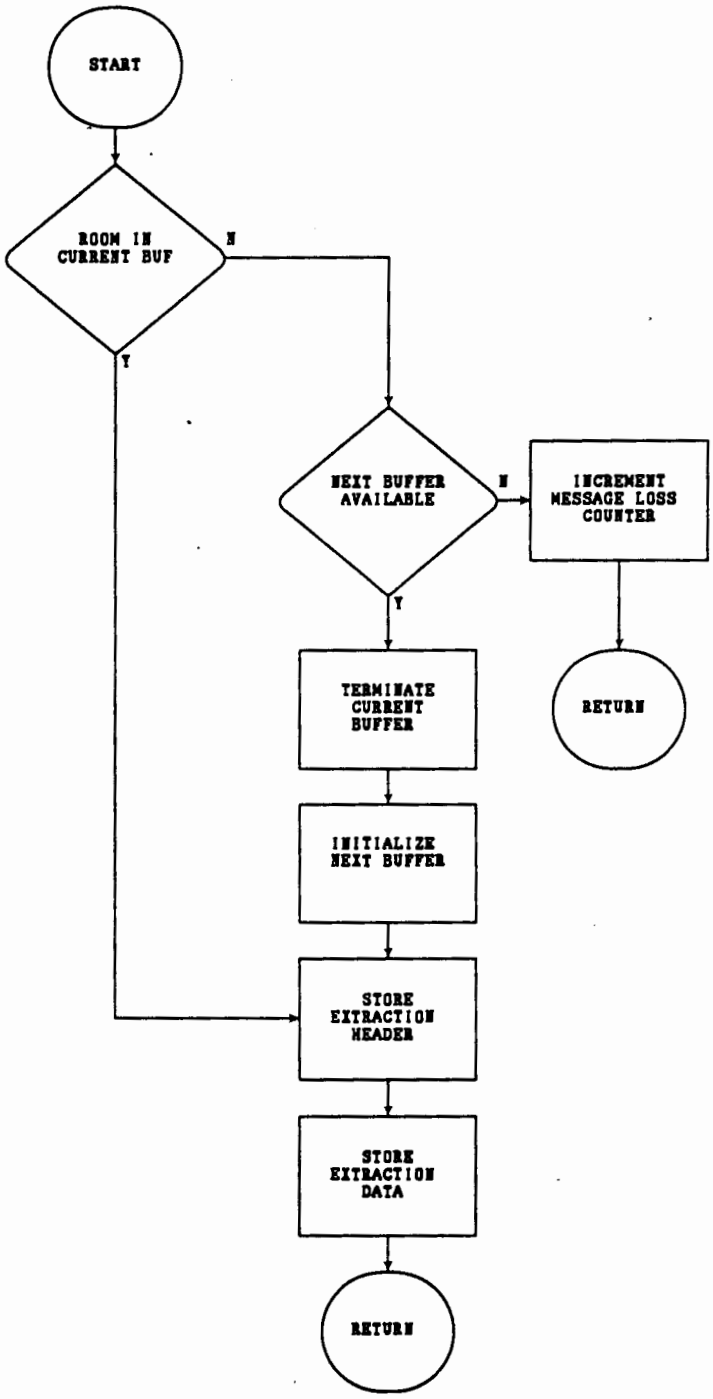


FIGURE 5.1.9-1. EXTRACTION PROGRAM FLOWCHART

LOGICAL RECORD STRUCTURE:

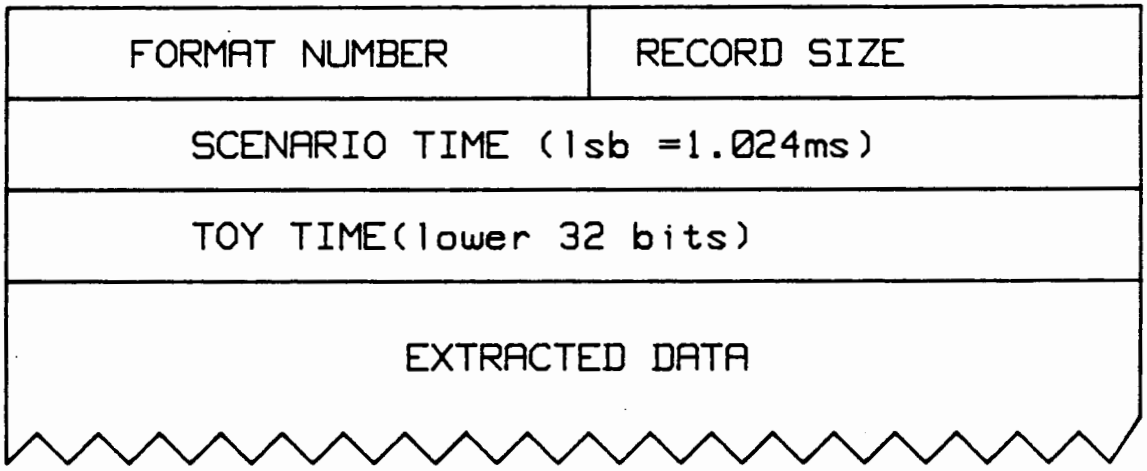


FIGURE 5.1.9-2. EXTRACTION HEADER RECORD DESCRIPTION

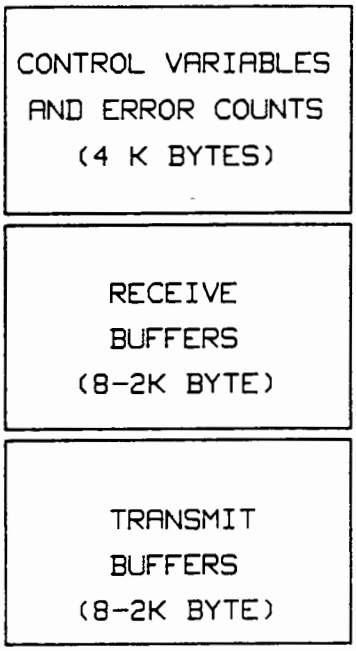


FIGURE 5.1.10-1. CID COMMON MEMORY STRUCTURE

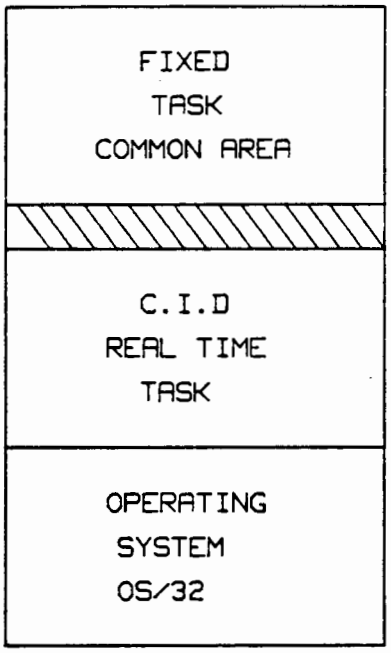


FIGURE 5.1.10-2. CID REAL-TIME MEMORY STRUCTURE

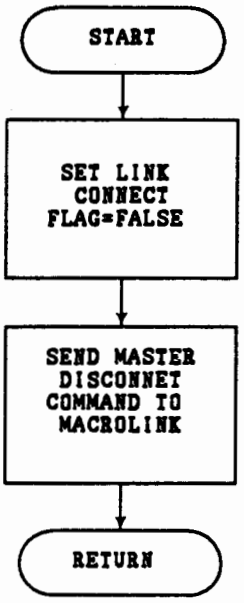


FIGURE 5.1.10-3. FLOWCHART FOR MDISC X25

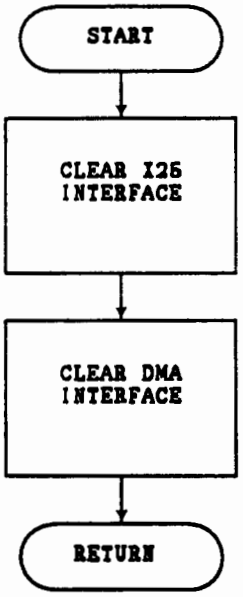


FIGURE 5.1.10-4. FLOWCHART FOR RESET X25

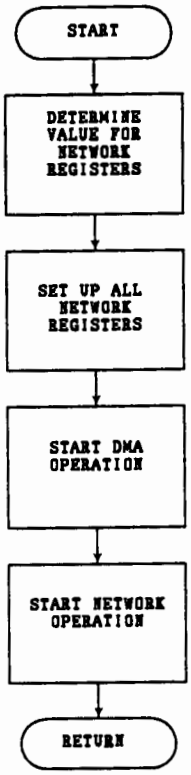


FIGURE 5.1.10-5. FLOWCHART FOR INIT X25

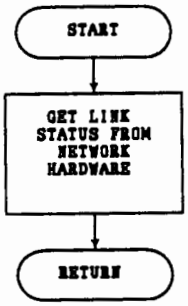


FIGURE 5.1.10-6. FLOWCHART FOR LINK STATUS

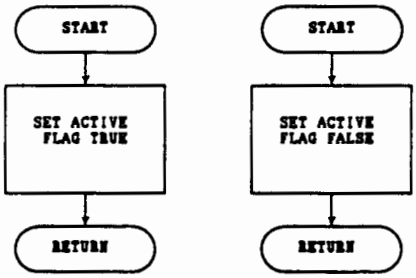


FIGURE 5.1.10-7. FLOWCHART FOR ACTIVE X25 AND DEACTIVE X25

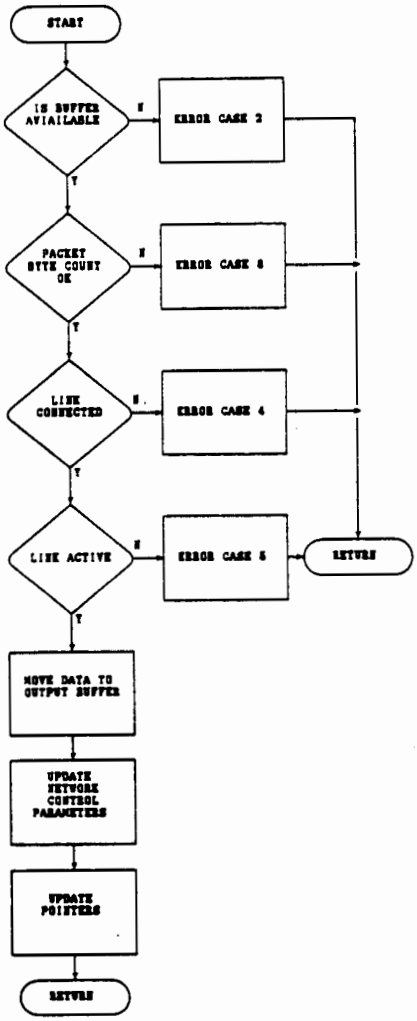


FIGURE 5.1.10-8. FLOWCHART FOR TXMITx

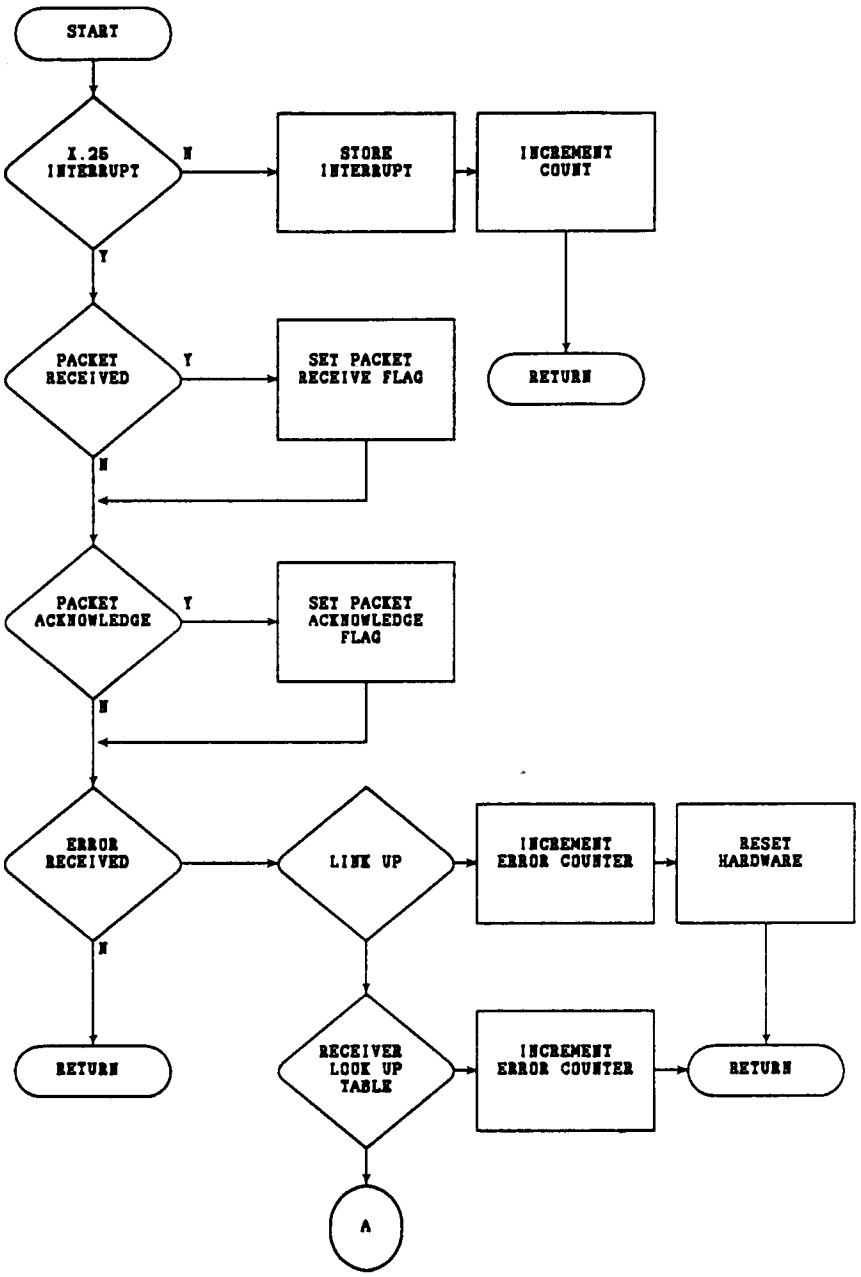


FIGURE 5.1.10-9. FLOWCHART FOR INTERRUPT SERVICE ROUTINE (Page 1 of 2)

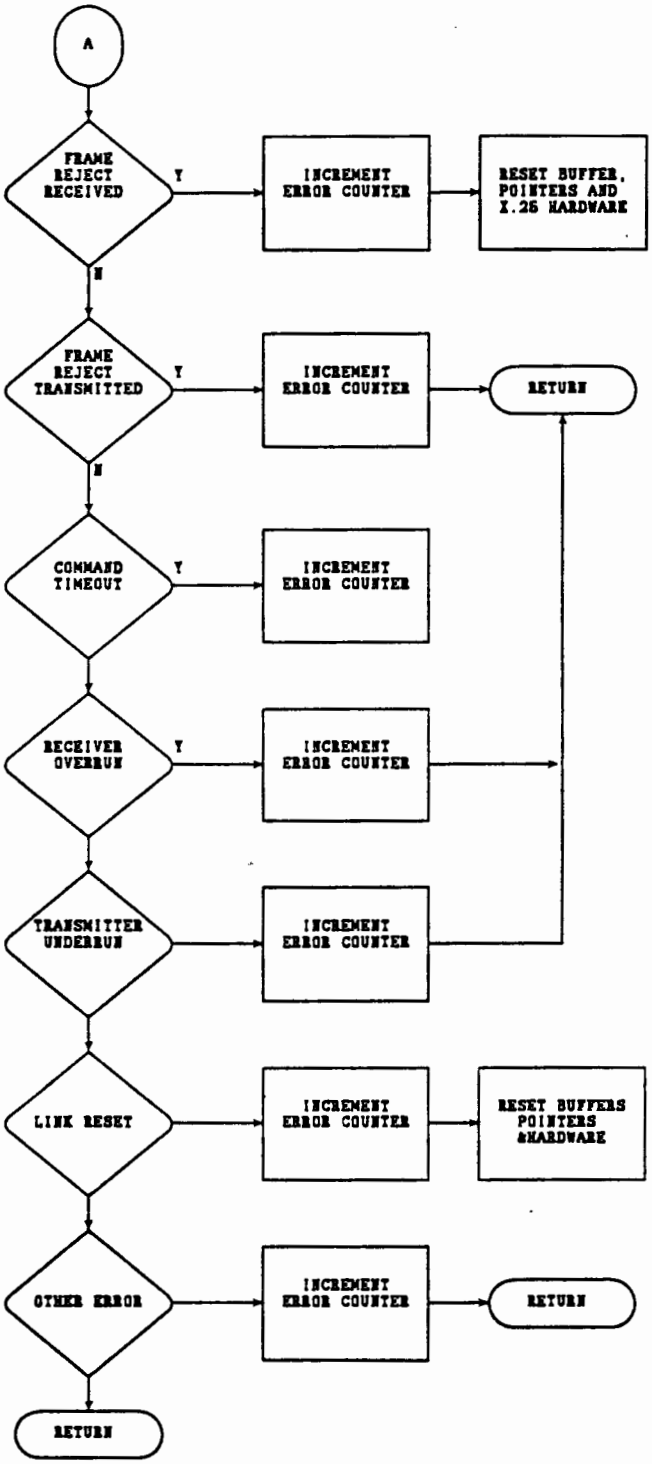


FIGURE 5.1.10-9. FLOWCHART FOR INTERRUPT SERVICE ROUTINE (Page 2 of 2)

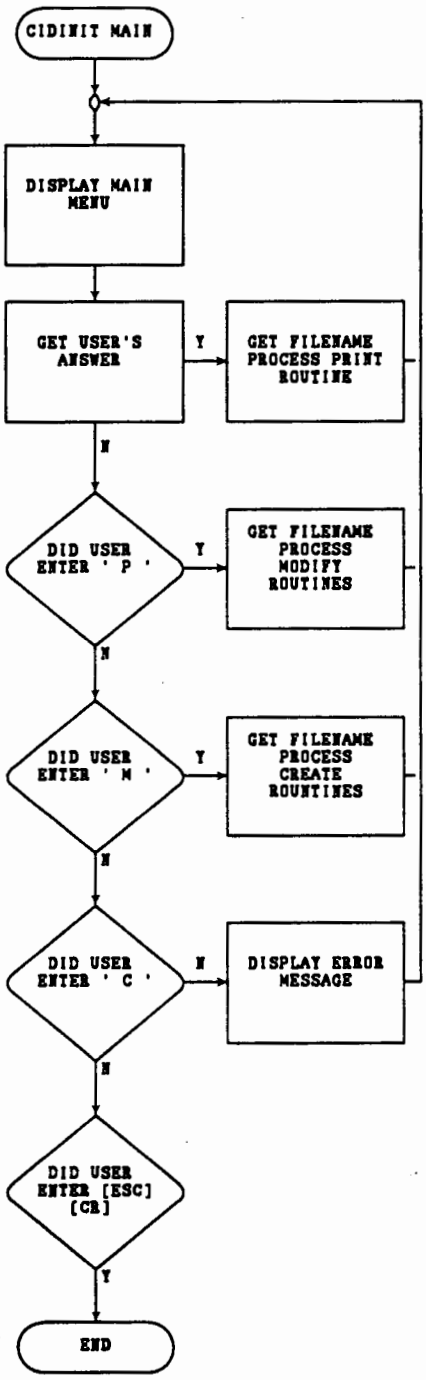


FIGURE 5.2.1-1. CIDINIT MAIN ROUTINE FLOWCHART

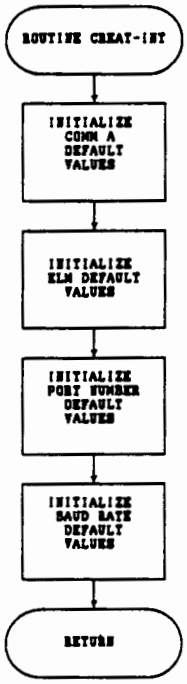


FIGURE 5.2.2-1. CIDINIT CREAT_INIT ROUTINE FLOWCHART

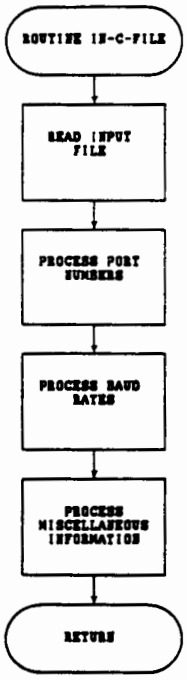


FIGURE 5.2.2-2. CIDINIT IN_C_ FILE ROUTINE FLOWCHART

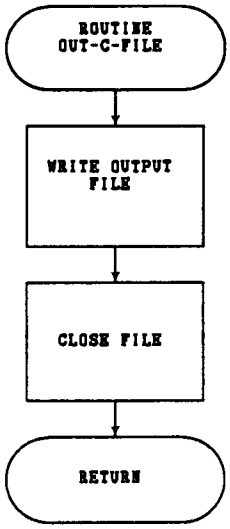


FIGURE 5.2.2-3. CIDINIT OUT_C_FILE ROUTINE FLOWCHART

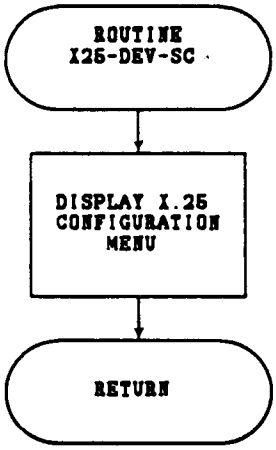


FIGURE 5.2.3-1. CIDINIT X25_DEV_SC ROUTINE FLOWCHART

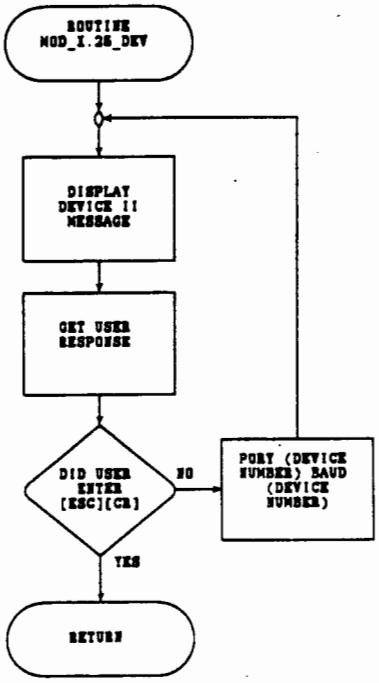


FIGURE 5.2.3-2. CIDINIT MOD_X.25_DEV ROUTINE FLOWCHART

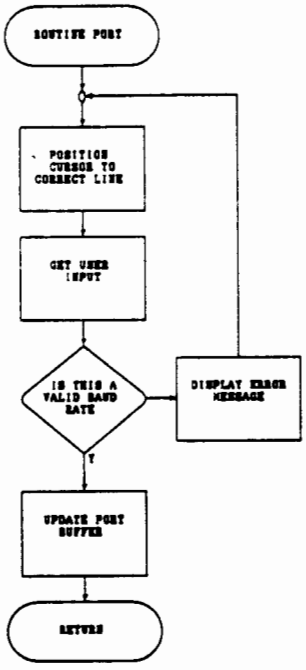


FIGURE 5.2.3-3. CIDINIT PORT ROUTINE FLOWCHART

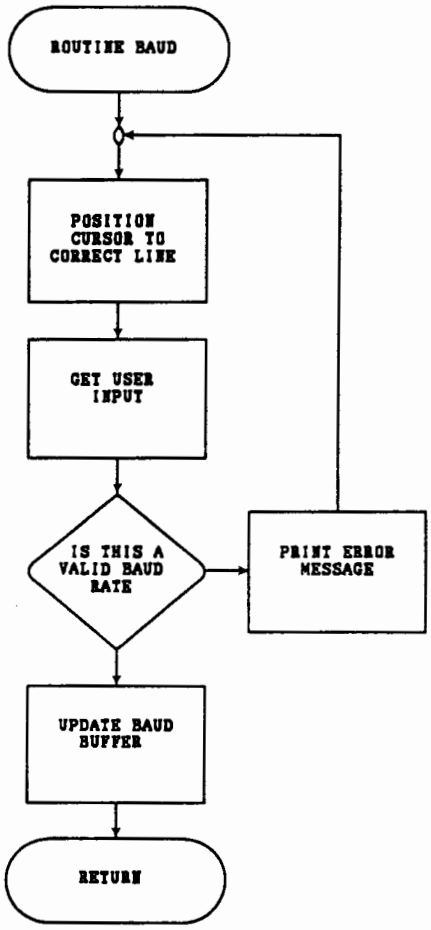


FIGURE 5.2.3-4. CIDINIT BAUD ROUTINE FLOWCHART

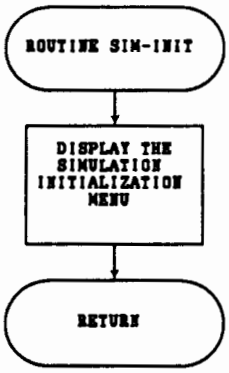


FIGURE 5.2.4-1. CIDINIT SIM_INIT ROUTINE FLOWCHART

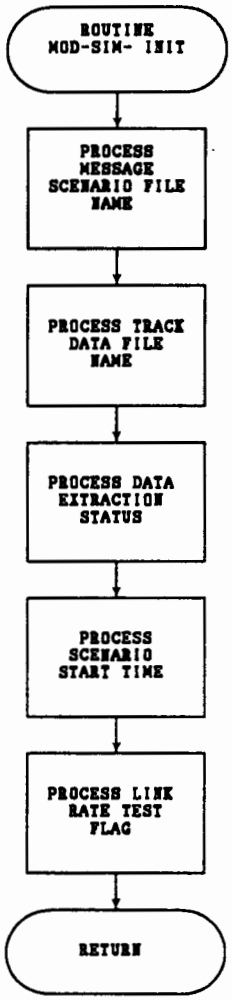


FIGURE 5.2.4-2. CIDINIT MOD_SIM_INIT ROUTINE FLOWCHART

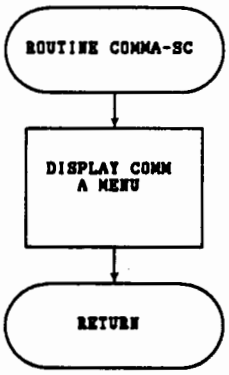


FIGURE 5.2.5-1. CIDINIT COMMA_SC ROUTINE FLOWCHART

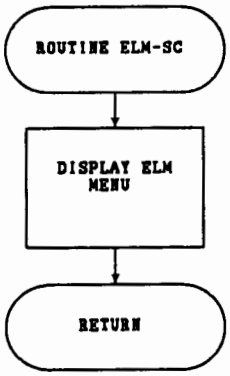


FIGURE 5.2.5-2. CIDINIT ELM_SC ROUTINE FLOWCHART

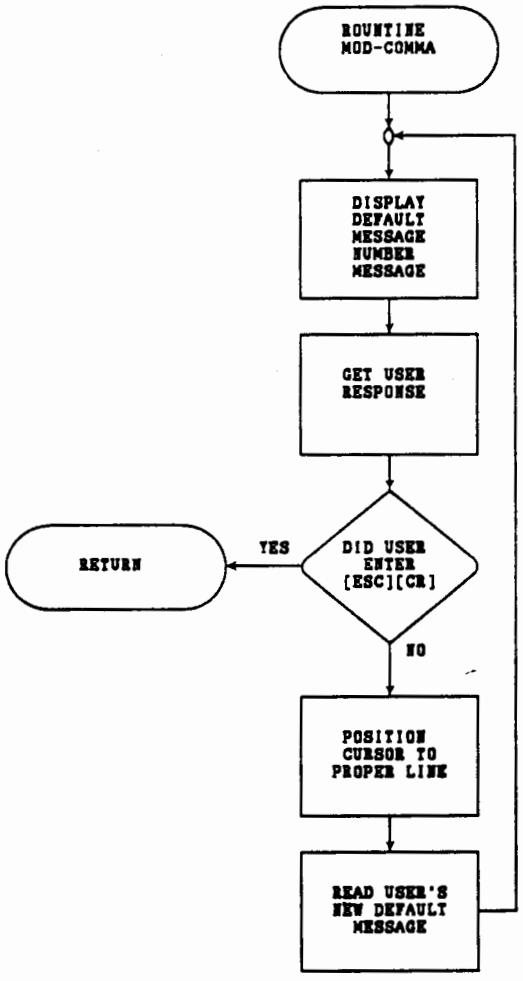


FIGURE 5.2.5-3. CIDINIT MOD_COMMA ROUTINE FLOWCHART

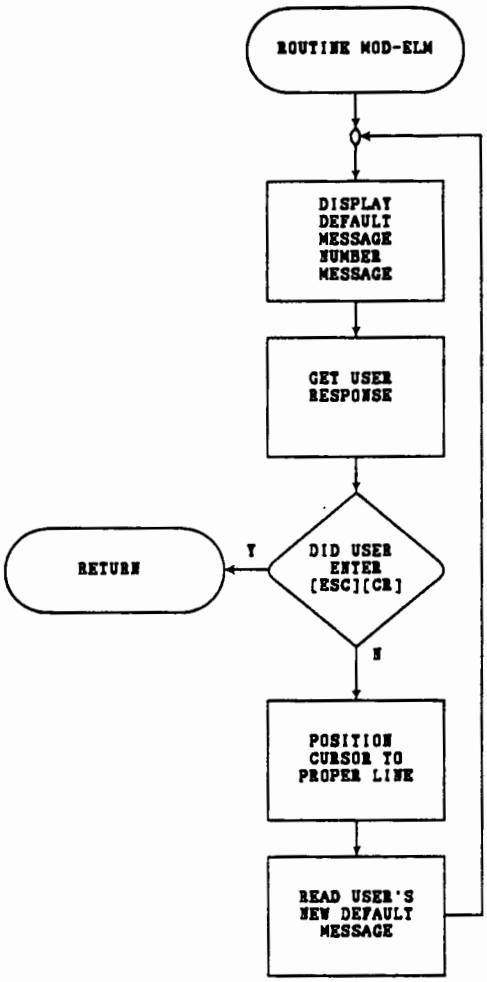


FIGURE 5.2.5-4. CIDINIT MOD_ELM ROUTINE FLOWCHART

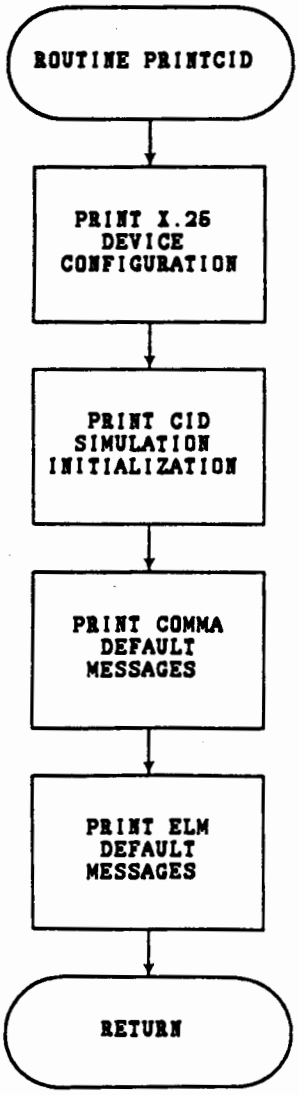


FIGURE 5.2.6-1. CIDINIT PRINTCID ROUTINE FLOWCHART