

AD-A102 294

MITRE CORP BEDFORD MA
MICROCOMPUTER POLLING IMPROVEMENTS FOR AFSATCOM.(U)
JUN 81 J HANDWERKER

F/G 17/2.1

UNCLASSIFIED

MTR-8239

ESD-TR-81-118

F19628-81-C-0001
NL

1-4
AD
4-102294



ESD-TR-81-118

LEVEL II

12

MTR-8239

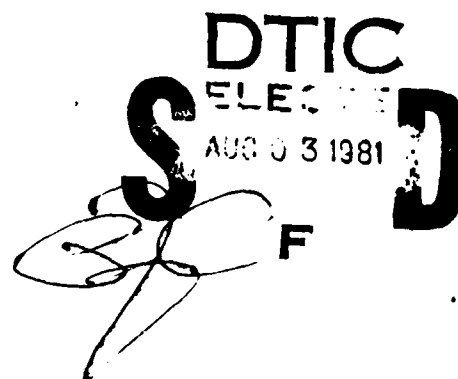
**MICROCOMPUTER POLLING IMPROVEMENTS
FOR AFSATCOM**

JACOB HANDWERKER

JUNE 1981

Prepared for

**DEPUTY FOR COMMUNICATIONS AND INFORMATION SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Massachusetts**



Approved for public release;
distribution unlimited.

Project No. 6340

Prepared by

**THE MITRE CORPORATION
Bedford, Massachusetts**

Contract No. F19628-81-C-0001

81 7 31 074

AD A102294

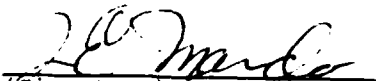
DTIC FILE COPY

When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

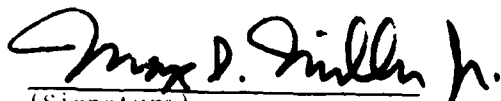
REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.


(Signature)

Joseph Mardo/GS-13
Project Engineer

FOR THE COMMANDER


(Signature)

MAX I. MILLER, Jr., Colonel, USAF
System Program Director
Satellite Communications Terminal SPO
Deputy for Communications and
Information Systems

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-81-118	2. GOVT ACCESSION NO. AD-A102 294	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MICROCOMPUTER POLLING IMPROVEMENTS FOR AFSATCOM		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Jacob Handwerker		8. PERFORMING ORG. REPORT NUMBER MTR-8239
		9. CONTRACT OR GRANT NUMBER(s) F19628-81-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation P. O. Box 208 Bedford, MA 01730		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 6340
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Communications and Information Systems Electronic Systems Division, AFSC Hanscom AFB, MA 01731		12. REPORT DATE June 1981
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES 122
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) AFSATCOM MICROCOMPUTER POLLING AFSATCOM POLLING MICROCOMPUTER MICROPROCESSOR		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report documents a cost effective approach for an on-line, remotable demonstra- tion of AFSATCOM polling improvements made possible through currently available, gen- eral purpose microcomputer technology. Descriptions of present and enhanced AFSAT- COM terminal capabilities, and a comprehensive overview of the resultant microprocessor- based hardware and software, and the support facility are presented. The conclusion is that message processor unit software-only changes or stand-alone microcomputer hardware additions to existing AFSATCOM terminals can significantly enhance (over)		

DD FORM 1473 1 JAN 73

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (Continued)

netting capabilities for the satellite-user community

X

UNCLASSIFIED

ABSTRACT

This report documents a cost effective approach for an on-line, remutable demonstration of AFSATCOM polling improvements made possible through currently available, general purpose microcomputer technology. Descriptions of present and enhanced AFSATCOM terminal capabilities, and a comprehensive overview of the resultant microprocessor-based hardware and software, and the support facility, are presented. The conclusion is that message processor unit software-only changes or stand-alone microcomputer hardware additions to existing AFSATCOM terminals can significantly enhance netting capabilities for the satellite-user community.

Accession For		
NTIS GRA&I	☒	
DTIC TAB	☐	
Unannounced	☐	
Justification_____		
By_____		
Distribution/		
Availability Codes		
Dist	Avail and/or	
A	Special	

ACKNOWLEDGMENTS

Grateful acknowledgment is made to Robert K. Waite whose software and hardware expertise and general participation in the microcomputer project described herein were instrumental to its success. Also, recognition is given to Anthony J. Lauletta for his mechanical and electronic skills in assembly of the system, and to Fay S. Durk for providing needed support and understanding of MITRE's Time Sharing Option (TSO) system. Finally, acknowledgment is made to William T. Higgins for his unwavering managerial support which ensured the success of this project.

This report has been prepared by The MITRE Corporation under Project 6340. The contract is sponsored by the Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Massachusetts.

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
LIST OF ILLUSTRATIONS	viii
LIST OF TABLES	x
1 INTRODUCTION	1
2 AFSATCOM SYSTEM OVERVIEW	3
2.1 SPACE SEGMENT	3
2.2 TERMINAL SEGMENT	5
2.3 AFSATCOM TYPE 12 COMMAND POST TERMINAL	6
2.4 CURRENT AFSATCOM POLLING CAPABILITIES	9
2.5 POLLING LIMITATIONS	9
3 AFSATCOM POLLING IMPROVEMENTS PROJECT	13
3.1 PROJECT DESCRIPTION	13
3.2 LIMITED TEST RESULTS	15
4 MICROCOMPUTER-BASED POLLING ENHANCEMENTS	16
4.1 MAJOR POLLING IMPROVEMENTS	16
4.1.1 Group/Non-Group Polling and Prioritization	16
4.1.2 Remote Entry Automatic Check-in	18
4.1.3 TDM Auto Polling	19
4.1.4 Automatic Slot Assignment	19
4.1.5 Encrypted Non-TDM Polling	20
4.1.6 Error Detection and Correction (EDAC)	20
4.1.7 Variable Non-Reportback Time-out	21

TABLE OF CONTENTS (continued)

<u>Section</u>	<u>Page</u>
4.2 NEW COMMANDS AND MESSAGES FOR POLLING	21
4.2.1 COMSUP Commands	21
4.2.2 Error Messages	26
4.2.3 External Message Formats	26
5 MICROCOMPUTER HARDWARE	28
5.1 INTEL SBC 80/20 SINGLE BOARD COMPUTER	28
5.2 NATIONAL SEMICONDUCTOR BLC 416 16K PROM BOARD	32
5.3 INTEL 450 16K RAM BOARD	32
5.4 I/O AND DISPLAY CIRCUITS	32
5.4.1 Master Clock Generator	32
5.4.2 External I/O Interface	35
5.4.3 RX Character Clock Generator	35
5.4.4 Memory I/O Address Decoder/Acknowledge	35
5.4.5 TX Character Deletion	35
5.4.6 Modem/MPU-to-SBC Serial Data Adapter	40
5.4.7 SBC 80/20 Interfaces	40
5.4.8 Slot Counter Display	40
5.4.9 RS-232 Serial Interfaces	44
5.5 MODULAR RACK HARDWARE FEATURES	44
6 MICROCOMPUTER SOFTWARE	46
6.1 FLOW CHARTS	46

TABLE OF CONTENTS (concluded)

<u>Section</u>	<u>Page</u>
6.2 PL/M-80 Source Code and Memory Allocation	49
6.3 MITRE TSO SYSTEM PL/M-80 SUPPORT	49
6.3.1 PL/M-80 High Level Language	49
6.3.2 PL/M-80 2-Pass Cross-Compiler	49
6.3.3 INTERP/80 for the 8080 Microprocessor	63
6.4 TEKTRONIX 8002A SOFTWARE DEVELOPMENT SYSTEM	63
7 CONCLUSIONS/RECOMMENDATIONS	69
LIST OF REFERENCES	72
APPENDIX A BASELINE DESCRIPTION OF THE AFSATCOM ROLL CALL POLLING MODE	73
APPENDIX B PL/M-80 SOURCE CODE	81
GLOSSARY	111

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Page</u>
2-1 AFSATCOM System Concept	4
2-2 Type 12 Ground Command Post Terminal Functional Block Diagram	7
2-3 Polling Delay Performance	12
3-1 Polling Hardware/Software Design Cycle	14
5-1 Microcomputer Functional Block Diagram	29
5-2 Microcomputer System Assembly	30
5-3 AFSATCOM T12-to-Microcomputer System Interface Diagram	33
5-4 Master Clock Generator (Sheet 1)	34
5-5 External I/O Interface (Sheet 2)	36
5-6 RX Character Clock Generator (Sheet 3)	37
5-7 Memory I/O Address Decoder/Acknowledge (Sheet 4)	38
5-8 TX Character Deletion (Sheet 5)	39
5-9 Modem/MPU-to-SBC Serial Data Adapter (Sheet 6)	41
5-10 SBC 80/20 Single Board Computer Interfaces (Sheet 7)	42
5-11 Slot Counter/Display (Sheet 8)	43
5-12 RS-232 Serial Interfaces	45
6-1 Microcomputer Development Facility	47
6-2 Polling Software Program Flowchart	48
6-3 Polling Algorithm Flowchart	54

LIST OF ILLUSTRATIONS (concluded)

<u>Figure</u>		<u>Page</u>
6-4	File Structure and Flow of Program Execution on 8080 PL/M Cross-Compiler	61
6-5	Run-Time Storage Organization of Memory Storage Allocation	62
6-6	Tektronix 8002A Microprocessor Laboratory Work Station	65

LIST OF TABLES

<u>Table</u>		<u>Page</u>
4-1	Improved Polling Performance Comparison	17
4-2	COMSUP Commands for Polling Enhancements	22
4-3	Error Message Formats	27
4-4	External Message Formats	27
5-1	EPROM (2708) Location Summary	31
6-1	Program Software Module Description	50
6-2	Symbol Table and Memory Map	55
6-3	INTERP/80 Commands	64

SECTION 1

INTRODUCTION

This report describes a breadboard microcomputer system based on an Intel SBC 80/20 Single Board Computer (SBC) which fulfills the project demonstration requirements for an improved polling capability for various satellite terminal users. The system underscores the flexibility and power of microprocessor-based hardware and software applications and provides extensive simulation capabilities related to terminal performance improvements for Air Force Satellite Communications (AFSATCOM) ultra high frequency (UHF) satellite communications in a real-time environment.

A preliminary study to investigate the attributes and growth potential of the AFSATCOM narrowband (NB) automatic Roll-Call Polling mode was undertaken by The MITRE Corporation for the Electronic Systems Division/Air Force Systems Command (ESD/AFSC). The purpose of this study was to lay the groundwork for eventual implementation of polling mode enhancements through software-only changes to the AFSATCOM message processor unit (MPU) subsystem as one means of improving overall netting capabilities. Subsequent polling-related studies involving advanced concepts have identified growth possibilities in the MPU subsystem which need software-only changes.

As a result of these studies, it was concluded that many of these new polling capabilities could be demonstrated easily and economically through use of currently available microcomputer techniques without the need to modify any existing AFSATCOM hardware or software. New functional capabilities could be added by a serially transparent microcomputer using existing interfaces presently employed by the AFSATCOM MPU subsystem. A polling improvements project was therefore undertaken which resulted in the development and testing of the microcomputer system detailed in this document.

To place the polling improvements in perspective, an overview of the AFSATCOM space and terminal segments is presented in section 2. Section 3 briefly describes the polling improvements project and its initial test results. Section 4 presents details of the polling improvements actually implemented by using a microcomputer add-on enhancement with an AFSATCOM terminal. Sections 5 and 6 provide details of the microcomputer hardware and software, respectively, for the demonstration system. Section 7 presents the overall conclusions and recommendations with regard to AFSATCOM polling enhancements and the follow-on objectives and planning for

subsequent demonstration testing. The AFSATCOM Roll-Call Polling mode is described in appendix A and a full source code listing is provided in appendix B.

SECTION 2

AFSATCOM SYSTEM OVERVIEW

The AFSATCOM System is designed to provide command, control, and communications capability on a worldwide basis to Single Integrated Operations Plan (SIOP) and other designated high priority users for emergency action message (EAM) dissemination, force direction, force reportback, and Commander-in-Chief (CINC) internetting. Communications are also provided for a limited number of non-SIOP normal force elements. The AFSATCOM System operates in the 225 to 400 MHz UHF spectrum and uses the 75 bits-per-second (b/s) teletype (TTY) service with frequency shift keying (FSK) modulation.

The AFSATCOM System, depicted in figure 2-1, consists of a space segment and a terminal segment. The space segment includes a communications capability designed into satellites such as the Navy Fleet Satellite Communications (FLTSATCOM) satellites as well as Air Force transponders carried "piggy-back" on other vehicles. The terminal segment is composed of various configurations operationally characterized as SIOP, normal force (non-SIOP), and command. SIOP and normal force terminals utilize only 5-kHz NB FSK channels, while command terminals also have access to an M-ary FSK 500-kHz bandwidth wideband spectrum. To ensure orderly management of day-to-day operation and control, each terminal user is part of an overall AFSATCOM System control structure.

2.1 SPACE SEGMENT

The AFSATCOM space segment consists of several types of transponder carried on different host satellites. AFSATCOM equipment is designed into each of the Navy FLTSATCOM satellites in geosynchronous equatorial orbit to provide overlapping earth coverage in all areas except the polar regions. Polar coverage is provided by Satellite Data System (SDS) satellites placed in highly inclined elliptical orbits.

Each satellite has 12 NB channels for specific user functions. The first seven NB channels are used to disseminate the EAM, force direction, and force reportback for SIOP/nuclear-capable forces. The next four NB channels are assigned to support essential operations of non-SIOP high priority Air Force users. The twelfth NB channel is used for system control and is designed to function as the orderwire (OW). Each of the FLTSATCOM satellites also has a

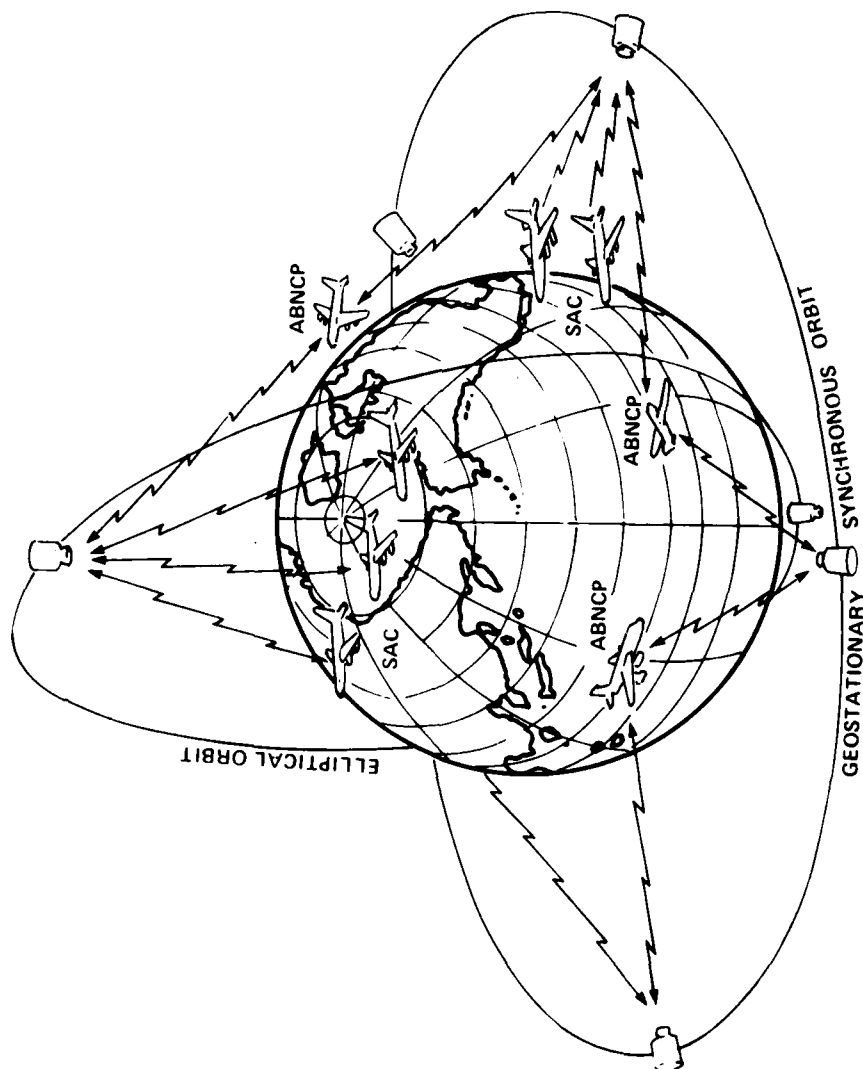


Figure 2-1 AFSATCOM System Concept

wideband (WB) channel. Accesses on the WB channel have been assigned to the National Command Authority (NCA), the CINCs, and members of the Worldwide Airborne Command Post (WWABNCP) fleet for the CINC Internet. SIOP and force terminals do not have access to the WB channel. Command post and net control terminal elements have the capability to command the various satellite modes using their OW or NB channel equipment.

2.2 TERMINAL SEGMENT

The AFSATCOM terminal segment consists of various equipment configurations to meet SIOP, normal force, and command operational requirements. The terminals are airborne or ground-based depending on the user application.

SIOP terminals provide 75 b/s half-duplex TTY operation via the 5-kHz NB satellite channels and can be operated in time division multiplex (TDM) modes under control of a command post. The input/output (I/O) devices are a keyboard, teleprinter, and appropriate control/monitor panels. Hard copy is available from terminals using the teleprinter. A SIOP synchronizer provides proper synchronization during TDM operation.

Normal force terminals (airborne and ground) provide 75 b/s half-duplex TTY operation via a 5-kHz, fixed frequency satellite channel. The absence of a SIOP synchronizer allows the normal force terminals to operate only in non-TDM modes.

Command post (CP) terminals can send, receive, and monitor messages; establish SIOP timing synchronization for TDM-1 (normal) and TDM-2 (stressed) modes; provide satellite UHF commands; initiate EAMs; and operate in a network control system using an OW channel. They can also monitor and record all reportback data to allow command transition. These terminals use full-duplex 75 b/s TTY operation on the 500-kHz WB channel, along with half-duplex and full-duplex 75 b/s TTY on the 5-kHz NB channels. Each CP terminal can operate as either a CP terminal or SIOP force terminal when in a slave mode. In the slave mode, SIOP timing synchronization is provided by another CP terminal. The AFSATCOM Type 12 CP terminal is described below since it was used for the polling improvement demonstration capability discussed later.

2.3 AFSATCOM TYPE 12 COMMAND POST TERMINAL

The AFSATCOM Type 12 (AN/TSC-88) CP terminal provides full-duplex record communications using FSK modulation at 75 b/s and is housed in an S-280 shelter that can be transported for rapid global deployment. The terminal consists of equipment installed in four electrical equipment cabinets and a two-position operator console, all within the shelter. Steerable and fixed antennas are set up outside the shelter when it is operational and are stowed inside during transport. Figure 2-2 is a functional block diagram of the Type 12 CP terminal.

Five AN/ARC-171 transceivers (R/Ts) and their associated controls, in conjunction with two half-duplex and one full-duplex narrowband modems, and two full-duplex wideband modems, provide five transmit (TX) and 12 receive (RX) communications channels. These channels are identified as follows:

- NB-1: a one transmit/one receive (1x1), half-duplex, NB (5-kHz) channel.
- NB-2: a one transmit/eight receive (1x8), full-duplex, NB (5-kHz) channel.
- OW: a one transmit/one receive (1x1), half-duplex, NB (5-kHz) channel.
- WB-1: a one transmit/one receive, full-duplex, WB (500-kHz) channel.
- WB-2: a one transmit/one receive, full-duplex, WB (500-kHz) channel.

Several narrowband operating modes are available in the AFSATCOM System. During normal operation, messages on NB channels one through seven are received by FLTSATCOM and SDS satellite packages on fixed uplink frequencies and retransmitted on fixed downlink frequencies. Methods of transmission in the normal mode include random and polled access. Random operation allows the terminal to enter the net on a random access basis using an open (not busy) select channel. Polled operation allows the Type 12 CP terminal to query each terminal in the net in sequence, with each terminal responding automatically with a precomposed message when it detects its own unique code sequence.

All of the NB channels, including the OW channel, can be used for automatic polling. In the polling mode, the Type 12 CP terminal sequentially addresses poll inquiry messages to terminals assigned

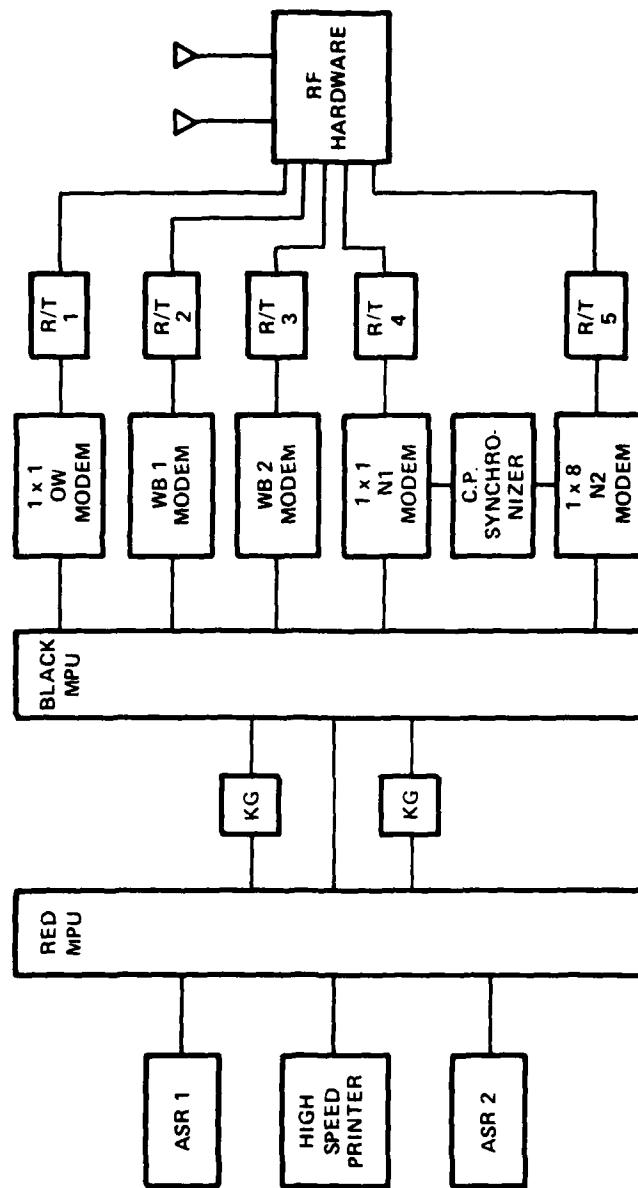


Figure 2-2 Type 12 Ground Command Post Functional Block Diagram

to various poll groups. Poll groups and polling operations are under the control of software programs in the terminal computer, the MPU. Four separate poll groups and up to 16 members apiece are allowed by the software. Each terminal to be polled will precompose a poll response message. When the unique address of that terminal is recognized in a poll message, the precomposed message of up to 30 seconds in length is automatically transmitted.

During periods of high density traffic, the Type 12 terminal can initiate synchronized operation for SIOP users by implementing a TDM mode. In this mode, the CP synchronizer periodically transmits a sync message which automatically establishes a time reference within each SIOP terminal in the net. In the TDM-1 mode, each SIOP terminal is constrained to transmit a precomposed message within its assigned time slot. Sixty consecutive time slots constitute a frame after which the sequence is repeated, with the CP synchronizer transmitting a new sync message in the last time slot of each frame.

During TDM-1, only the transmit time of each SIOP terminal is controlled by the CP synchronization message. During TDM-2, both the transmit time and transmit frequency of each SIOP terminal are synchronized by the CP synchronizer. In the TDM-2 mode, the satellite transponder is commanded into a stressed mode, which causes the first seven NB receive channels in the satellite to frequency hop on a pseudorandom basis. Also in the TDM-2 mode, each SIOP terminal is constrained to transmit in its assigned time slot and the transmit frequency changes with each time slot. The transmit frequency is determined by the code-of-the-day which is programmed into each synchronizer by the operator. Before the net can enter the TDM-2 mode, the satellite must be commanded into the stressed mode.

Three modes of operation are also provided on the WB M'ary FSK channels. These are: (1) random access precomposed message; (2) random access manual message composition; and (3) full-duplex relay, which is also available in NB random access mode.

During normal CP terminal operations, communications channels OW and WB-2 are assigned to operate with a fixed antenna. The WB-1, NB-1, and NB-2 channels are assigned to operate with a tracking antenna. Antenna switching is provided to allow the OW and WB-1 channels to operate with either antenna.

Additional details pertaining to the AFSATCOM Type 12 terminal can be found in another document.

2.4 CURRENT AFSATCOM POLLING CAPABILITIES

The present AFSATCOM polling capability is primarily structured for automatic operation in the non-TDM unencrypted mode over a single 5-kHz NB channel. Polling operations for AFSATCOM System control are conducted from MPU-equipped terminals by utilizing a polling algorithm contained in the MPU software. Communications supervisory (COMSUP) commands input by the terminal operator are used either to enter or delete poll addresses as well as to start polling operations. Up to 64 pollees may be entered into a poll net consisting of four groups having 16 members, maximum, per group. Addresses for each group member are related to others in the same group, with poll address recognition actually taking place within the AFSATCOM automatic send-receive (ASR) I/O hardware. During MPU automatic polling operations, a non-reportback of a polled address will cause a 30-second fixed time-out delay within the MPU before generation of the next poll message. Appendix A gives a more complete description of current AFSATCOM polling operation, including polling message formats, address structure, and poll-related COMSUP commands for the MPU.

2.5 POLLING LIMITATIONS

The following limitations of the AFSATCOM polling mode constrain its utility in an operational environment.

1. Small Network Size

A maximum of only 64 pollees can be accommodated in the existing MPU polling software. Present AFSATCOM requirements foresee nets needing to accommodate between two and three times this number.

2. Inflexible Addressing Structure

The group organization required in present auto polling dictates that all members of a poll group have the last two of their three address characters in common. Polling always commences with the lowest address of each group of 16 and proceeds in sequence toward the highest address before attempting to poll the next group. No provision is made for randomized poll addressees who are not group-related members.

3. No Prioritization/Interruption Features

No provision is made for real-time prioritization of pollees based on operational needs for early or more frequent report-back requirements. Also, no provision is made for momentary or short-term interruptions of polling operations to accommodate higher priority traffic without complete repolling of all users.

4. Inefficient Bookkeeping Scheme

Current polling operations dictate that all poll address additions or deletions be entered by COMSUP command. This can be a time consuming process, especially since no provision for a truly automated MPU poll table generation/display is available to the operator. Also, prior coordination of poll address assignments is essential but is unwieldy when attempted in real-time. Manual bookkeeping procedures are thus mandated to keep abreast of current poll net membership.

5. No Automated Handover Procedures

No provision is currently made for an automated handover procedure to an alternate net control station (NCS) for polling operations. All poll address additions/deletions must be accommodated by the alternate NCS on a manual basis.

6. No TDM Polling Mode

Automatic polling is currently restricted to the non-TDM mode. Although TDM poll response is possible from a polled I/O, the existing MPU polling software is not presently configured to cope with the structural differences between AFSATCOM non-TDM and TDM operation since automated TDM polling was not originally conceived as a basic AFSATCOM netting requirement.

7. Single Channel Polling Restriction

Polling inquiry messages sent by the MPU during polling operations are limited to a single channel. Simultaneous, independent polling on more than one channel is not currently possible in a terminal equipped with a single MPU.

8. Unencrypted Reportback Limitation

In general, the 30-second time-out constraint during automatic polling limits the length of reportback possible. In addition,

MPU-equipped terminals incur internal MPU delays such that their reportback is limited to 130 characters, maximum. This restricts the use of the Auto Polling mode to unencrypted operation with a fixed upper limit on the size of a reportback message; the upper limit depends on the type of terminal responding.

9. No-show Delay Penalty

The present fixed 30-second time-out for auto-polling also incurs a severe delay penalty when a no-show occurs. A no-show, or non-reportback, may occur for a variety of reasons such as failure to properly receive the polling inquiry message, failure to be in the I/O poll mode, etc. Figure 2-3 shows the polling delay performance of the present non-TDM roll-call polling mode as a function of the number of non-reportbacks (assuming a 40-character reportback when one does occur) for the maximum net size of 64 polled non-MPU terminals. The vertical axis indicates the time to completion for polling the entire net. As fewer net members report back, the time to completion for polling increases dramatically. This is counter to the concept of maintaining a predictable level of AFSATCOM network performance in situations supposedly under the control of a CP terminal.

10. Error Detection and Correction

No automated procedure currently exists for repolling a net member upon detection of a character error while in the polling mode. The existing AFSATCOM unencrypted message structure provides for odd-parity character transmission, but this capability has yet to be fully exploited.

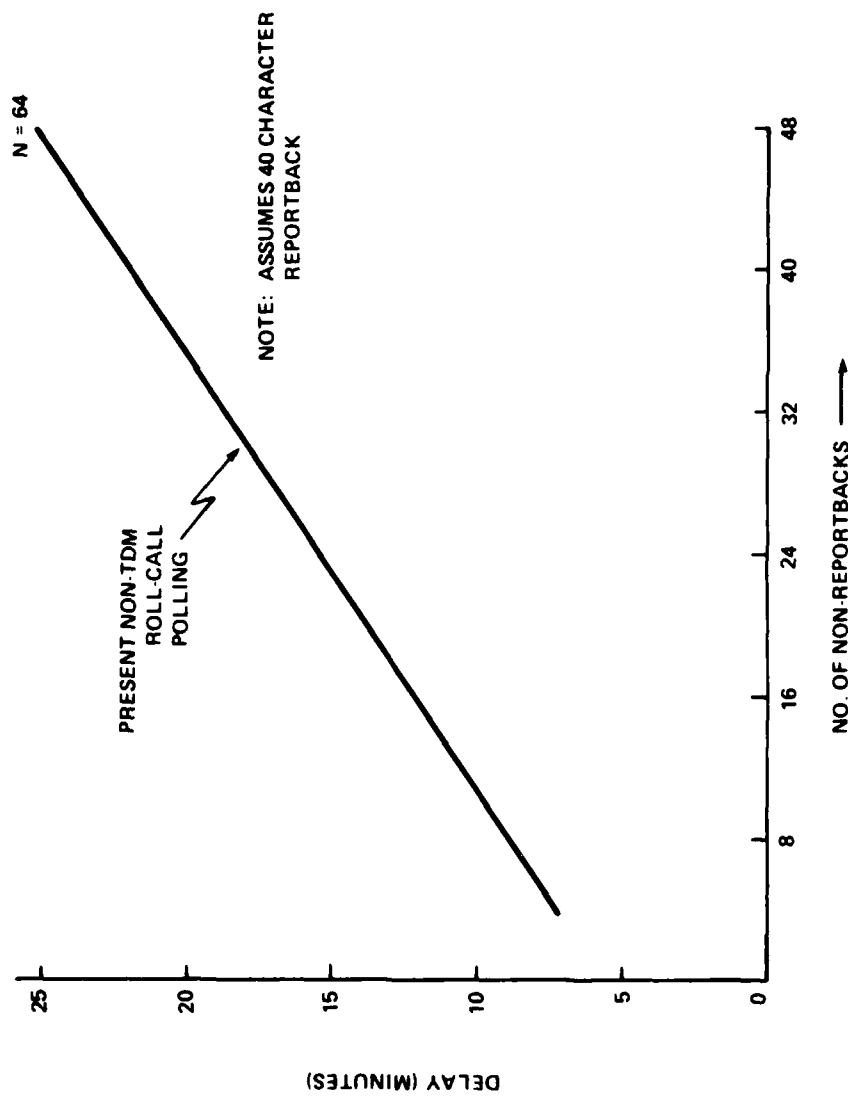


Figure 2-3 Polling Delay Performance

SECTION 3

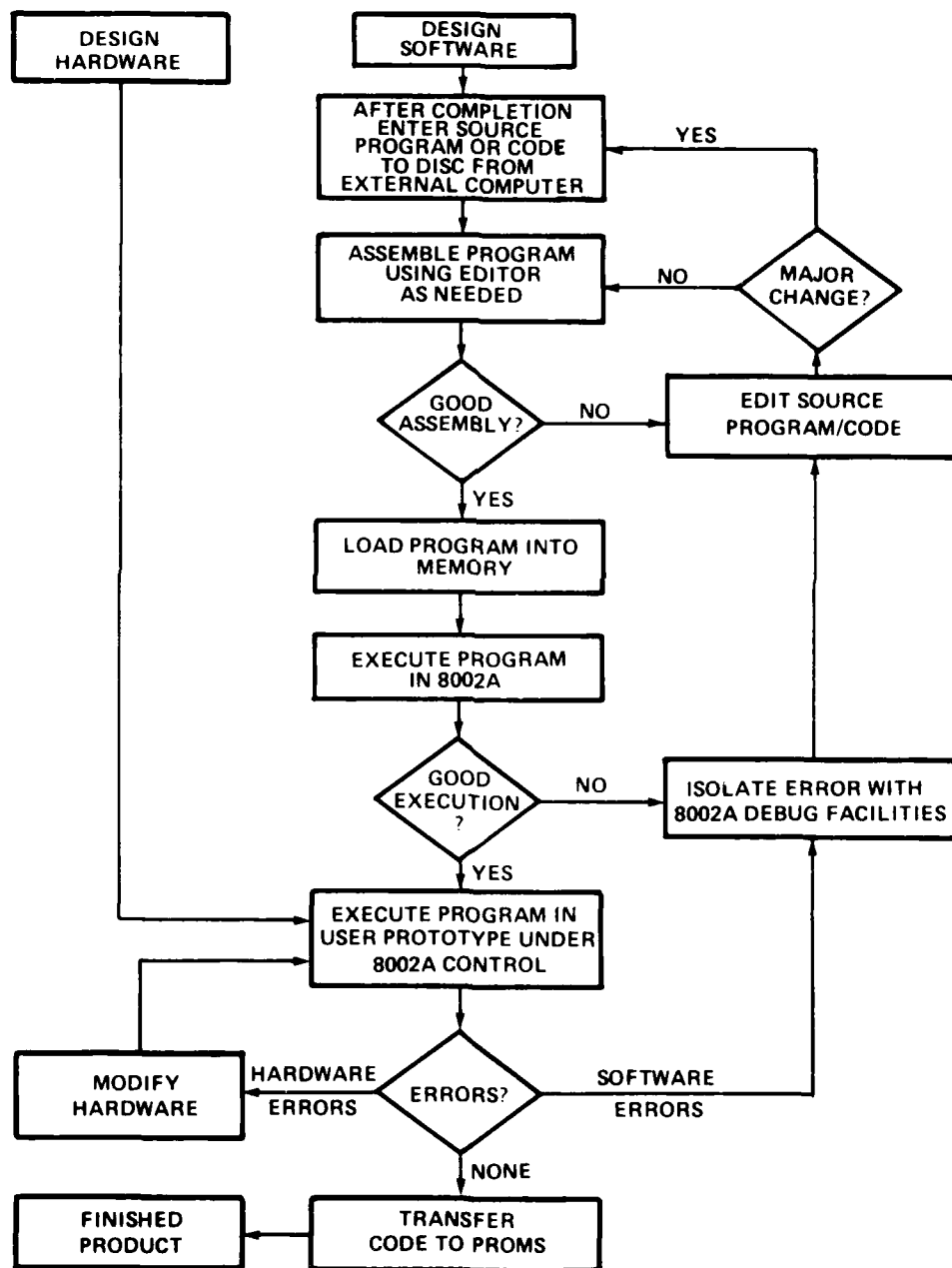
AFSATCOM POLLING IMPROVEMENTS PROJECT

Providing enhanced polling capabilities by using AFSATCOM-compatible techniques to meet the ever increasing demands of the AFSATCOM System is a primary concern because the Air Force has an extensive hardware investment in terminal assets. Significant changes to present hardware or software must be evaluated with regard to both operational and logistical effects. Polling improvements should also provide flexibility for a wide range of applications yet to be defined and at the same time address the reportback needs of specific AFSATCOM users with known requirements.

3.1 PROJECT DESCRIPTION

A microcomputer-based approach for demonstrating various improved AFSATCOM polling capabilities, involving no hardware or software modifications to existing AFSATCOM equipment, was conceived by MITRE to be the most cost-effective means of validating many of the proposed concepts. The microcomputer provides real-time, on-line simulation of new polling capabilities while functionally emulating changes to MPU software. Making this microcomputer serially transparent to existing system operation with no need to modify AFSATCOM assets is an inexpensive way to validate new polling concepts. Operational MPU software does not need to be modified until after these concepts are validated and user coordination is achieved on a final implementation approach. Physically, the microcomputer interfaces between the MPU and either NB-1 or OW modem in the AFSATCOM Type 12 terminal. A project was initiated to translate these improved polling concepts into actual breadboard hardware and software suitable for demonstration purposes. For this purpose, the microcomputer development facility based around a Tektronix 8002A microprocessor laboratory at MITRE-Bedford was used.

Figure 3-1 describes the overall implementation approach for the project. Because of the small staff for this undertaking (two part-time MITRE technical staff), relatively close coordination between hardware and software design efforts was possible, with the result that the project was successfully completed in nine calendar months.



1A 59 075

Figure 3-1 Polling Hardware/Software Design Cycle

3.2 LIMITED TEST RESULTS

Following completion of bench testing and terminal integration, the microcomputer system was made available for demonstration testing. Technical compatibility of the improved polling hardware/ software with the AFSATCOM Type 12 terminal was demonstrated during the initial tests. With the microcomputer installed in the terminal, both software and hardware bypasses provided AFSATCOM system transparency for normal AFSATCOM modes. After completion of various terminal performance tests, actual network testing was conducted using the FLTSATCOM satellites. All of the significant performance improvements for polling (described in detail in section 4) were also demonstrated successfully in conjunction with other cooperative AFSATCOM terminals operating through the satellite system. As a result, the microcomputer system is now considered available for network-wide tests on a much larger scale, with emphasis on operational testing to demonstrate improved netting capabilities to potential users.

SECTION 4

MICROCOMPUTER-BASED POLLING ENHANCEMENTS

Significant improvements to the present AFSATCOM polling mode operation have been achieved with the microcomputer-based addition to the AFSATCOM Type 12 CP terminal. These improvements have been demonstrated to participating terminal operators during actual on-line demonstrations.

4.1 MAJOR POLLING IMPROVEMENTS

The major polling improvements implemented in the AFSATCOM Type 12 CP terminal are described below. Table 4-1 compares existing AFSATCOM polling capabilities and the microcomputer-based improvements. These improvements are a real-time simulation of capabilities possible with software-only improvements to the AFSATCOM MPU.

4.1.1 Group/Non-Group Polling and Prioritization

The current AFSATCOM System can poll up to four groups of related net members where each group is defined as a set of 16 addresses and each address in the group has the same last two hexadecimal address characters. This group structure is retained in the improved polling software. In addition, a new set of net members is permitted which may contain up to 80 random addressees (equivalent to five 16-member groups) independent of any related group address structure. This provides flexibility to organize and poll up to a maximum of 144 group and non-group net members per channel.

During polling net organization with the microcomputer software, the non-group members are accorded a higher priority level with respect to group-related members. Four priority levels of polling net organization have been chosen for demonstration purposes: (1) priority check-ins, (2) routine check-ins, (3) no-traffic check-ins, and (4) group check-ins.

During polling operations, the non-group related check-ins of priority levels 1, 2, and 3 are polled before the group-related check-ins of level 4. The NCS terminal operator can enter any of these check-ins into the polling net by COMSUP command. A remote-entry automatic check-in for levels 1, 2, and 3 is also possible.

Table 4-1
Improved Polling Performance Comparison

AFSATCOM Functional Capability	Non-TDM Polling Mode		TDM Polling Mode	
	Present	With Microcomputer	Present	With Microcomputer
Auto Polling	Yes	Yes	No	Yes
Encryption	No	Yes	N/A	No
Limited EDAC	No	Yes*	N/A	No
Non-Group Polling/Prioritization	No	Yes	N/A	Yes
Auto Check-in	No	Yes	N/A	Yes
Automatic Slot Assignment	N/A	N/A	N/A	Yes
Variable Time-Out	No	Yes	N/A	N/A
Table Printout/Transfer (With Selective Routing)	No	Yes	N/A	Yes
Poll Interrupt/Resume	No	Yes	N/A	Yes
Number of Users	64**	144***	N/A	48/Frame

*Poll Messages Only

**Four Groups of 16

***Four Groups of 16 Plus 80 Non-Group Users

In addition to normal termination of polling, an automatic repoll capability has been provided to allow continuous monitoring of reportback status activity. Also, an interrupt-and-resume feature provides a standby mode for polling operations to accommodate higher priority communications without the need to re-initiate a repoll of net members who have already reported back.

4.1.2 Remote Entry Automatic Check-in

The microcomputer software provides an automatic remote self check-in before the start of actual polling operations. This allows potential users to become polling network members without manual intervention by the NCS terminal operator. A non-group polling list can be rapidly structured yet the work-load imposed on the operator by the large number of possible random addressees is minimized. This enhancement also allows check-in table entry while current polling operations are in progress.

The NCS terminal operator retains positive control of all check-ins, including remote check-ins with the following software features:

1. The remote check-in mode can be enabled or disabled separately by COMSUP command.
2. Check-ins can be verified using COMSUP commands to obtain hardcopy poll table printouts of listed poll check-ins and poll net members. (This includes both group and non-group check-ins.) Check-in addition/deletion can be accommodated by COMSUP command as needed.
3. Automatic slot assignment and reportback prioritization (order of reportback) of poll net members is done in the polling software before the start of actual polling operations; reassignment of slots is also possible with a COMSUP command.
4. An ALL CALL message initiated via COMSUP command by the NCS terminal operator informs all listed polling net members of their TDM slot assignments (to be dialed manually into their respective AFSATCOM synchronizers). Dial-up of respective slot assignments avoids subsequent contention problems during TDM polling operations should that mode become active.

4.1.3 TDM Auto Polling

This enhancement provides TDM auto polling in addition to the polling capabilities available in the non-TDM mode. Thus, AFSATCOM TDM-1 and TDM-2 system modes are made compatible with the new terminal polling capabilities. This implementation requires the AFSATCOM synchronizer in the CP terminal to act as a master (i.e., a net control element) during TDM polling operations.

To carry out auto polling in the TDM modes, the software utilizes a tabular listing containing the addresses of all terminals in the poll network and their respective assigned reportback time slots. The TDM mode provides 60 slots per frame. During actual polling operation in the TDM mode, eight contiguous slots containing up to six poll messages per slot at the beginning of each AFSATCOM frame are dedicated to sending polling messages from the CP terminal to the pollees. Three unassigned slots immediately following these eight may be used for break-in by unassigned or emergency users. The 48 slots following can then be used for terminal reportbacks. The final slot in the frame, however, is normally reserved for synchronization messages. Because each AFSATCOM terminal is assigned its own unique address, several pollees can be assigned to the same reportback slot. The microcomputer software presently provides for polling a maximum of 144 users in three successive frames (with 48 reportbacks per frame). This, however, is not an ultimate AFSATCOM System limitation. Up to three addressees may be assigned to the same slot under this polling scheme, but this number could be increased to accommodate additional users as necessary, if more frames are used.

The AFSATCOM TDM response of force terminals (or CP terminals in the slave mode) limits polling responses by a polled terminal to 40 message characters during a frame. Also, TDM polling does not have an encryption capability as presently implemented. However, various priority traffic indicators in the poll response message cause a printout alert message at the NCS terminal for subsequent operator action. These are described in more detail below.

4.1.4 Automatic Slot Assignment

In conjunction with TDM polling, the microcomputer software provides automatic slot assignment of listed poll net members for subsequent TDM reportback. Before actual start of polling, members of the network are informed of their slot assignments by means of an ALL CALL message generated by the software. This message also indicates which users have been accepted into the poll listing along with their priority order of reportback. The ALL CALL message, like

the slot assignment made during the course of compiling the poll table, is generated automatically by operator-initiated COMSUP command.

4.1.5 Encrypted Non-TDM Polling

The present AFSATCOM polling time-out limitation of 30 seconds precludes the use of long encrypted reportback sequences. The microcomputer system software, however, permits encrypted polling responses in the non-TDM mode of operation. The software is configured to recognize KG-35 encryption device message indicator (MI) sequences at the beginning of a message and automatically disables odd-parity checking in the AFSATCOM NB modem upon recognition of a valid MI header. While message traffic is being received during polling, no message time-out occurs in order to accommodate encrypted sequences of long lengths. Once received, the AFSATCOM MPU routes the incoming encrypted reportback to a KG-35 device for eventual decryption.

4.1.6 Error Detection and Correction (EDAC)

The present AFSATCOM polling capability does not provide for any EDAC during either polled or non-polled operations. The microcomputer software does provide limited EDAC during polling through satellite downlink monitoring of poll messages. Polling message retransmission then takes place upon detection of error. This technique, however, requires full-duplex operation (simultaneous transmit and receive) whereas the demonstration hardware is interfaced with either the AFSATCOM OW or the 1x1 NB modem, both of which are configured as half-duplex equipment.

With a full-duplex channel, after transmission of a polling message during non-TDM mode, the microcomputer compares the message sent with the one received on the satellite downlink and causes a retransmission of the same message upon detection of any mismatch. A maximum of three automatic poll message retransmissions is possible with the present microcomputer software. No EDAC is presently provided in the TDM polling mode nor is EDAC provided on reportback responses, although no inherent limitation precludes such additional capability.

4.1.7 Variable Non-Reportback Time-out

A variable non-reportback time-out feature using a COMSUP command with the microcomputer software provides the terminal NCS operator with an added dimension for control. By controlling the time-out delay from the possible non-reportback of polled terminals, the operator can minimize the time-to-completion of polling the overall network. The operator enters the COMSUP command with the desired time-out duration expressed in seconds. Note that the present AFSATCOM non-reportback poll time-out is a non-changeable 30 seconds, whereas the microcomputer software allows for a 1 to 99 second range. Choice of the optimum delay depends on actual satellite and AFSATCOM I/O device delays and the polling strategy to be employed. The ability to vary the time-out delay can have profound effects on the overall polling completion time.

4.2 NEW COMMANDS AND MESSAGES FOR POLLING

The following new polling commands and message formats are possible with the microcomputer software.

4.2.1 COMSUP Commands

Table 4-2 summarizes the new COMSUP commands available to the NCS terminal operator when using the microcomputer system polling enhancements. Note that all the commands listed contain the ZNR character field in the first three character positions in order to identify this properly as an unencrypted message in an AFSATCOM preproduction Type 12 terminal. (AFSATCOM production-type terminals would require a "UUU" sequence.) The fourth character position is an exclamation point ("!") which signifies a COMSUP message to the microcomputer's improved polling software. The fifth character position of a COMSUP message is always a mode character. Characters six through nine are microcomputer address characters. (The microcomputer address is also changeable via a COMSUP command.) COMSUP commands intended for the microcomputer, and any other unencrypted messages, are input to the AFSATCOM terminal and intercepted by the polling software. These COMSUP commands are never transmitted to the AFSATCOM modem.

The seven message categories defining the new COMSUP commands are shown in table 4-2. These functional categories satisfy the increased control and data entry requirements for the various polling mode enhancements.

Table 4-2

COMSUP Commands For Polling Enhancements

<u>Command Type</u>	<u>Function(s)</u>	<u>Command Message Format</u>
Check-in Table Entry	Enable External Check-ins	ZNR!SABC1
	Disable External Check-ins	ZNR!SABCØ
	Single Priority Check-in	ZNR!CABCAAA1
	Single Routine Check-in	ZNR!DABCBBB1
	Single No-Traffic Check-in	ZNR!EABCCCCØ
	Group Check-in	ZNR!BABCDD1
	Priority Table Load	ZNR!1ABC ADR1 ADR2
	Routine Table Load	ZNR!2ABC ADR3 ADR4
	No Traffic Table Load	ZNR!3ABC ADR5 ADR6
Check-in Table Deletion	Priority Table Re-initialize	ZNR!FABC
	Routine Table Re-initialize	ZNR!GABC
	No-Traffic Table Re-initialize	ZNR!HABC
	Group Table Re-initialize	ZNR!AABC
	Single Priority Deletion	ZNR!NABCPØ1
	Single Routine Deletion	ZNR!NABCRØ1
	Single No-Traffic Deletion	ZNR!NABCNØ1
	Single Group Deletion	ZNR!NABCGØ1

Table 4-2 (Continued)

COMSUP Commands For Polling Enhancements

<u>Command Type</u>	<u>Function(s)</u>	<u>Command Message Format</u>
Check-in/Poll Table Printout	Priority Table Printout	ZNR!UABCPXYZ
	Routine Table Printout	ZNR!UABCRXYZ
	No-Traffic Table Printout	ZNR!UABCQXYZ
	Group Table Printout	ZNR!UABCOXYZ
	Build/Printout of Poll Table	ZNR!MABC
	Poll Table Printout (Local)	ZNR!UABCZXYZL
	Poll Table Print- out (Local and Remote All Call)	ZNR!UABCZXYZR
External Table Transfer	Priority Check-in Table Transfer	ZNR!TABCXXYZ
	Routine Check-in Table Transfer	ZNR!TABCYXYZ
	No-Traffic Check-in Table Transfer	ZNR!TABCZXYZ
	Poll Table Transfer	ZNR!TABCTXYZ

Table 4-2 (Continued)

COMSUP Commands For Polling Enhancements

<u>Command Type</u>	<u>Function(s)</u>	<u>Command Message Format</u>
Poll Mode Control	Single Poll Enable	ZNR!JABC1S
	Single Poll Disable	ZNR!JABC0S
	Multiple Poll Enable	ZNR!JABC1M
	Multiple Poll Disable	ZNR!JABC0M
	Non-TDM Poll Interrupt	ZNR!LABC10
	TDM Poll Interrupt	ZNR!LABC01
	TDM and Non-TDM Poll Interrupt	ZNR!LABC11
	Disable All Poll Interrupts	ZNR!LABC00
	FDAC Mode Enable	ZNR!PABC1
	EDAC Mode Disable	ZNR!PABC0
	Poll Message Time Delay	ZNR!KABC10
	Reportback/Poll Message Slot Reassignment	ZNR!OABC10182634425004

Table 4-2 (Continued)

COMSUP Commands For Polling Enhancements

<u>Command Type</u>	<u>Function(s)</u>	<u>Command Message Format</u>
Miscellaneous	Disable COMSUP Msg. Printout	ZNR!4ABCØ
	Enable COMSUP Msg. Printout	ZNR!4ABC1
	Disregard All Non-COMSUP Msgs.	ZNR!VABC2
	Disregard Non-COMSUP Msgs. when Polling	ZNR!VABC1
	Enable Recognition of All Types of Msgs.	ZNR!VABCØ
	RS-232 I/O Enable	ZNR!QABC1
	RS-232 I/O Disable	ZNR!QABCØ
	Microcomputer Address Change	ZNR!IABCCAB
	Program Re-initialize	ZNR!RABC
	Loop-Around Message	ZNR* } Transmitted Message

4.2.2 Error Messages

Table 4-3 summarizes the new error messages available to the terminal operator with the microcomputer software. These messages assist in either flagging anomalous COMSUP command inputs to the microcomputer or indicating to the operator when pre-determined limits have been reached for polling check-in messages.

4.2.3 External Message Formats

Table 4-4 summarizes the types of externally generated messages from the AFSATCOM modem which are used for polling operations with the microcomputer software. These messages are divided into three basic categories:

1. Single Check-in
2. Check-in Table Transfer
3. Reportback

The single check-in message is normally sent by a terminal desiring to become a member of a poll network. It is accepted by the microcomputer software after the check-in mode has been enabled by the NCS terminal operator.

Check-in table transfer message formats allow entry of entire sets of check-ins involving multiple addressees into appropriately prioritized check-in tables.

Reportback message formats are normally sent by a terminal responding to a poll inquiry message. When the microcomputer software recognizes these reportback formats during polling operation, appropriate prioritization header messages are appended to the incoming message before printout on the terminal I/O device.

Table 4-3
Error Message Formats

<u>Message Name</u>	<u>Function</u>	<u>Message Text</u>
ADRERRORHDR	Informs of incorrect address characters	Incorrect ADR Characters
ERRORHDR	Informs of incorrect mode character	Incorrect Mode Character
TFRERRORHDR	Informs of attempted check-in table transfer with no check-ins listed	Table Transfer Error: No Check-ins
OVERFLOWHDR	Informs of Check-in limit being reached	Check-ins at the Limit

Table 4-4
External Message Formats

<u>Message Type</u>	<u>Function</u>	<u>Format</u>
Single Check-in	Priority Check-in	SOH C ABC ADR1 Text
	Routine Check-in	SOH D ABC ADR2 Text
	No-Traffic Check-in	SOH E ABC ADR3 Text
Check-in Table Transfer	Priority Check-in Table Transfer	SOH X ABC ADR1ADR2....
	Routine Check-in Table Transfer	SOH Y ABC ADR3ADR4....
	No-Traffic Check-in Table Transfer	SOH Z ABC ADR5ADR6....
Reportback Message	Priority Report back	SOH P ABC Text
	Routine Report back	SOH R ABC Text
	No-Traffic Report back	SOH N ABC Text

SECTION 5

MICROCOMPUTER HARDWARE

The microcomputer developed for demonstrating AFSATCOM polling improvements is shown in the functional block diagram in figure 5-1. It consists of an Intel SBC 80/20 Single Board Computer, a National Semiconductor Model BLC 416 16K-byte programmable read-only memory (PROM) board, an Intel Model 450 16K-byte random access memory (RAM) board, a power supply subsystem, and interface/display logic circuitry. The SBC 80/20, RAM, and erasable PROM (EPROM) boards are housed in the Intel SBC 604 Modular Cardcage/Backplane which provides interconnection for up to four plug-in boards. The SBC 604 also allows interconnection of two or more cardcage backplane assemblies for expansion, in addition to power supply connectors and signal line termination circuits. A small table-top 19-inch modular rack mounting, shown in figure 5-2, contains the cardcage, along with the power supply subsystem, slot selection and display circuits, bypass switches, test points, and cable terminations for circuit connections to the AFSATCOM System.

5.1 INTEL SBC 80/20 SINGLE BOARD COMPUTER

The Intel SBC 80/20 Single Board Computer is the heart of the microcomputer implementation for polling improvements. It is one of Intel's line of computer products and takes full advantage of large scale integration (LSI) technology to provide a self-contained computer capability on a single 6.75 x 12.00 inch printed circuit board. On-board capability provides a central processing unit (CPU), system clock, read/write memory, nonvolatile read-only memory (ROM), I/O ports and drivers, a serial RS-232 communications interface, priority interrupt logic, and two programmable timers. In addition, on-board multibus control logic and bus expansion drivers allow interfaces to multibus-compatible boards including expansion memory, digital and analog I/O expansion boards, peripheral controller, and other single board computers. A complete functional description of the SBC 80/20 is found in reference 1. Details of the SBC 80/20 are found in reference 2. Table 5-1 lists integrated circuit (IC) locations for polling program storage on the SBC 80/20.

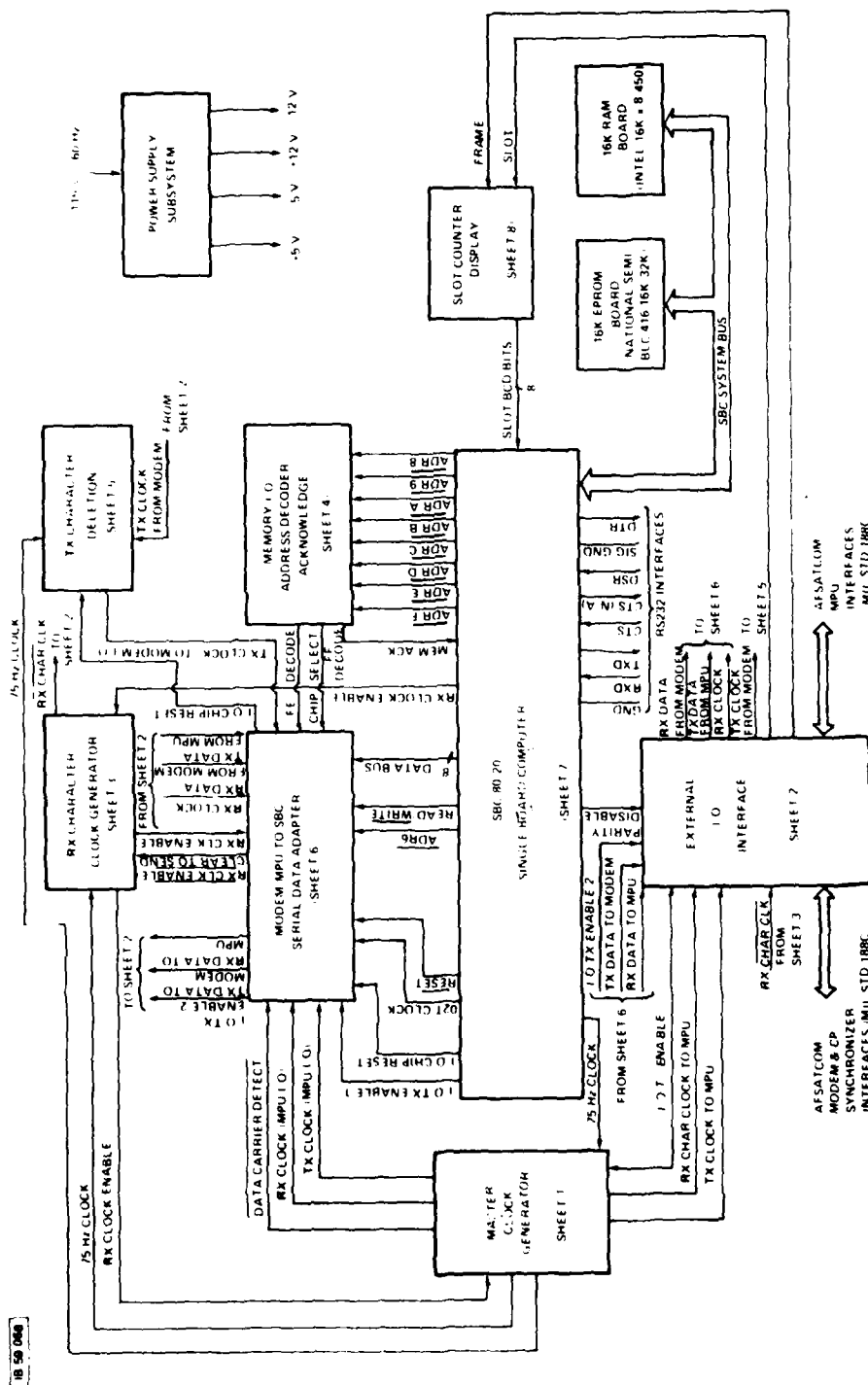


Figure 5-1 Microcomputer Functional Block Diagram

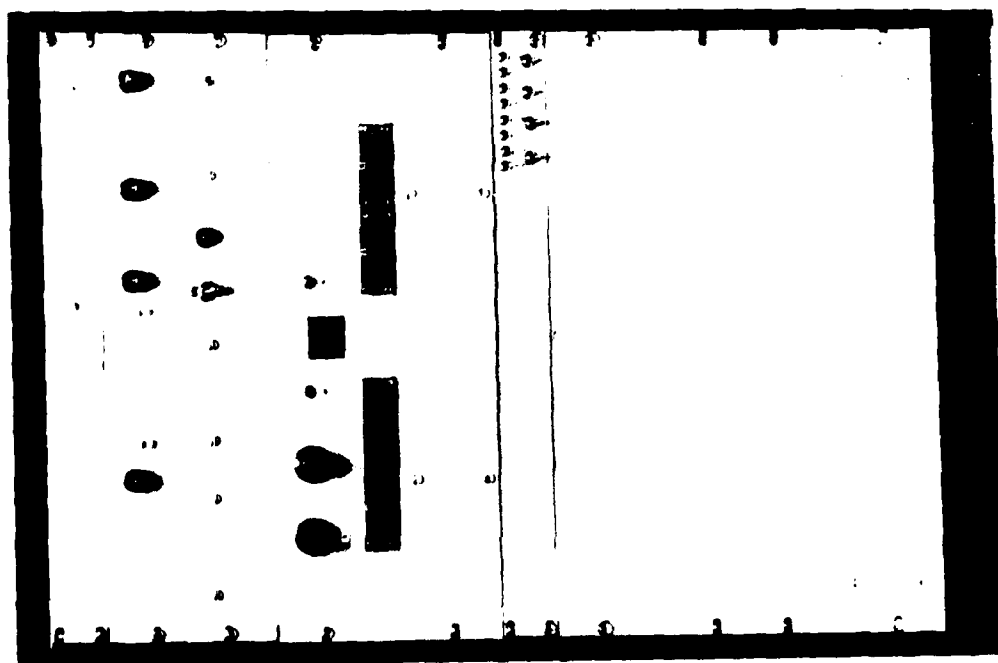
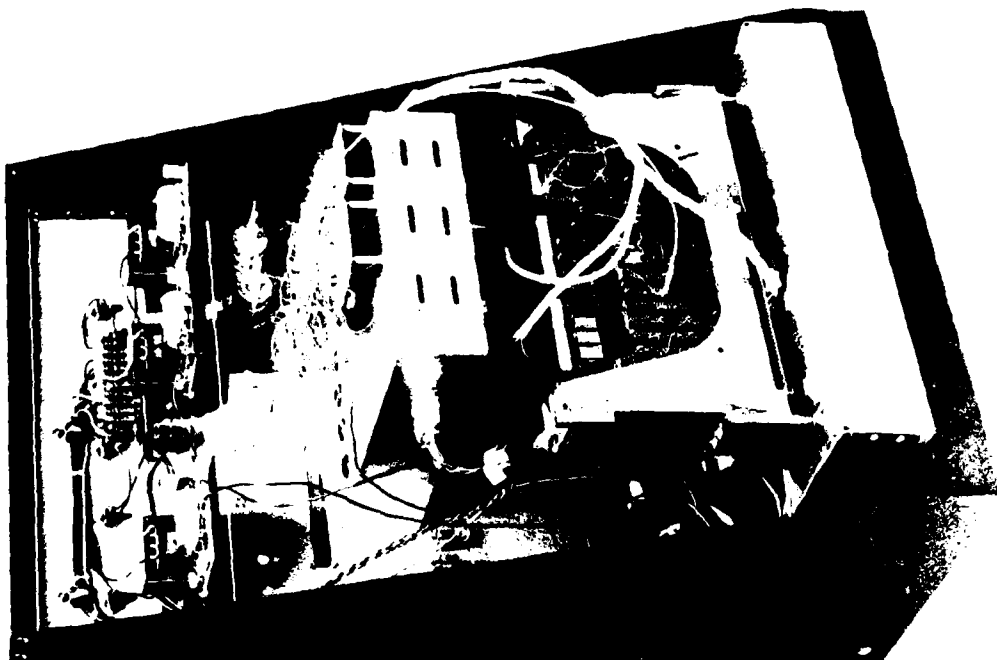


Figure 5-2 Microcomputer System Assembly

Table 5-1
EPROM (2708) Location Summary

<u>Prom #</u>	<u>IC</u>	<u>Board</u>	<u>Start ADR (Hex)</u>	<u>Finish ADR (Hex)</u>
1	A79	SBC 80/20	0000	03FF
2	A64	SBC 80/20	0400	07FF
3	A51	SBC 80/20	0800	0BFF
4	A37	SBC 80/20	0C00	0FFF
5	U33	BLC 416	1000	13FF
6	U21	BLC 416	1400	17FF
7	U9	BLC 416	1800	1BFF
8	U3	BLC 416	1C00	1FFF
9	U34	BLC 416	2000	23FF
10	U22	BLC 416	2400	27FF
11	U10	BLC 416	2800	2BFF
12	U4	BLC 416	2C00	2FFF
13	U35	BLC 416	3000	33FF
14	U23	BLC 416	3400	37FF
15	U11	BLC 416	3800	3BFF
16	U5	BLC 416	3C00	3FFF

5.2 NATIONAL SEMICONDUCTOR BLC 416 16K PROM BOARD

The National Semiconductor BLC 416 16K PROM expansion board provides sockets for up to 16K bytes of 2708 EPROMs (32K bytes for 2716 EPROMs). For the polling program, however, only 12 of the 16 2708 EPROMs required for program storage actually reside on this board; the four remaining 2708 EPROMs reside on the SBC 80/20. Table 5-1 summarizes the physical locations of the EPROM program storage for execution of the polling software. Switches to either enable or disable various memory blocks are provided on this board along with jumper-switch selectable addresses for each 4K block (8K when used with the 2716 EPROMs). This allows independent selection of base addresses of individual memory blocks on 4K byte boundaries. Along with on-board programming, the BLC 416 provides multibus-compatible address, data, and command signals. Table 5-1 lists the IC locations for polling program storage on the BLC 416 PROM board.

5.3 INTEL 450 16K RAM BOARD

The Intel 450 16K RAM is a multibus-compatible board which contains 16K bytes of read/write memory and uses 2107C dynamic memory components with on-board refresh circuitry for all the dynamic memory elements. Read/write buffering, which also resides on-board, buffers all data written into or read from the memory array and includes a jumper-selectable starting address for 16K contiguous addresses at 16K boundaries. For the polling application, the lowest RAM address is set at 4000H and the highest RAM address is 7FFFH.

5.4 I/O AND DISPLAY CIRCUITS

Figure 5-3 shows the additional cabling needed in the Type 12 CP terminal for interconnecting the I/O and display boards of the microcomputer system with the AFSATCOM System. No changes to existing AFSATCOM interfaces were made; only cabling and connector additions were employed. Each of the I/O and display circuits is detailed below.

5.4.1 Master Clock Generator

Figure 5-4 shows the Master Clock Generator circuit. This circuit provides various gated 75 Hz clocks and associated signals for both the transmit and receive functions utilizing a 75 Hz reference input from one of two timer circuits on the SBC 80/20.

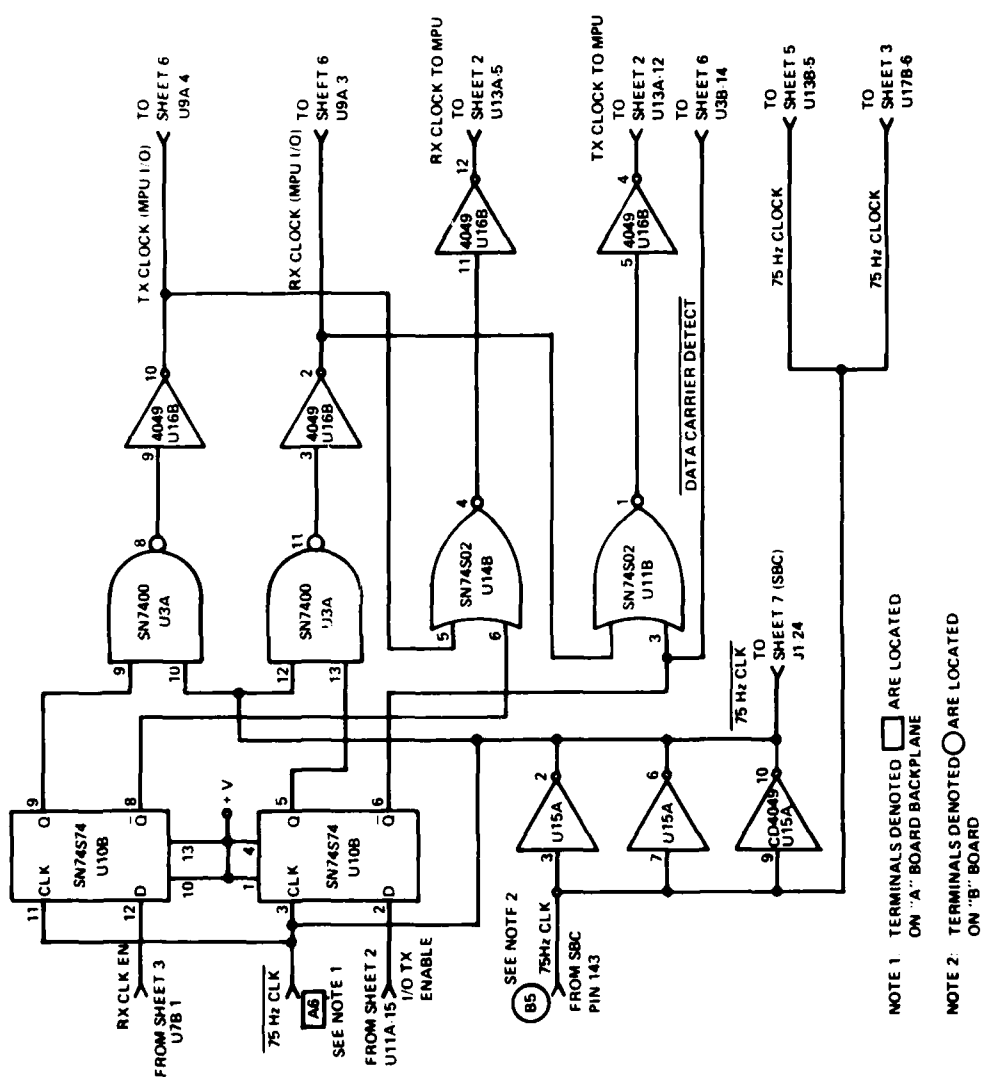


Figure 5-4 Master Clock Generator (Sheet 1)

5.4.2 External I/O Interface

Figure 5-5 shows the external I/O interface circuits. This circuitry provides the level conversion necessary to connect the MIL-STD-188C interfaces found in the AFSATCOM Type 12 terminal with the complementary metal-oxide semiconductor (CMOS) and transistor-transistor logic (TTL)-compatible circuits of the microcomputer system.

5.4.3 RX Character Clock Generator

Figure 5-6 shows the RX character clock generator circuit. This circuitry provides synchronous receive character clock for the MPU receive data interface normally supplied by the AFSATCOM Type 12 terminal. The microcomputer system does not use the RX character clock signal supplied by the AFSATCOM modem. However, for proper AFSATCOM operation, this clock is reconstituted and supplied to the MPU input circuitry.

5.4.4 Memory I/O Address Decoder/Acknowledge

Figure 5-7 shows the memory I/O address decoder/acknowledge circuit. Its primary function is to generate select signals for memory address decoding for a pair of Motorola MC6852 serial data adapter ICs. These select signals, along with the memory acknowledge signal, permit the ICs to be utilized as if they were an inherent part of the SBC 80/20 memory addressing space. Read or write operations to these devices then use the memory addresses which serve these two ICs.

5.4.5 TX Character Deletion

The TX character deletion circuit, shown in figure 5-8, deletes two redundant message characters encountered with the MC6852 IC used in conjunction with the AFSATCOM modem transmit interface. Actual transmit character deletion takes place before the start of transmission to the modem and occurs only during the time interval when the AFSATCOM modem is in the process of transmitting its own "W U SYN SYN" preamble.

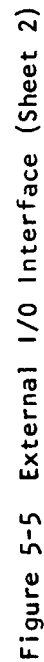


Figure 5-5 External I/O Interface (Sheet 2)

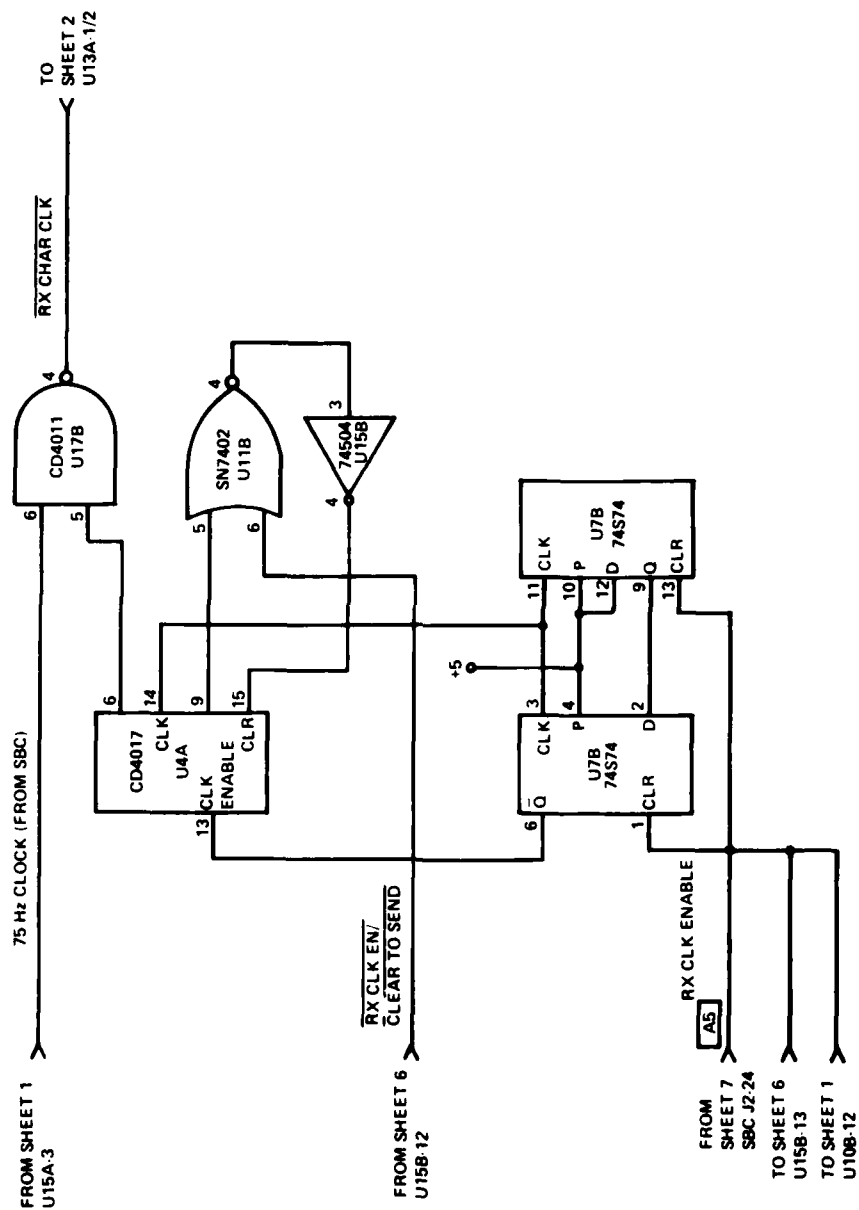


Figure 5-6 RX Character Clock Generator (Sheet 3)

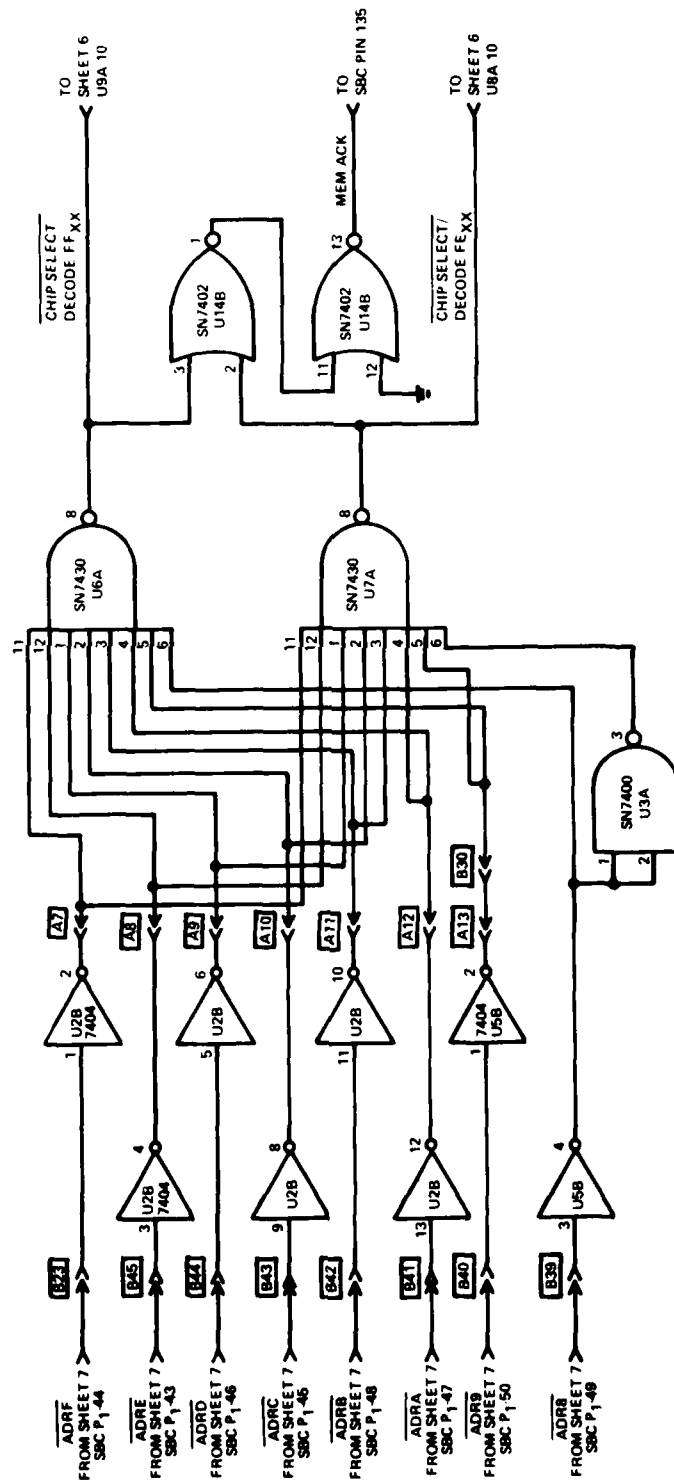


Figure 5-7 Memory I/O Address Decoder/Acknowledge (Sheet 4)

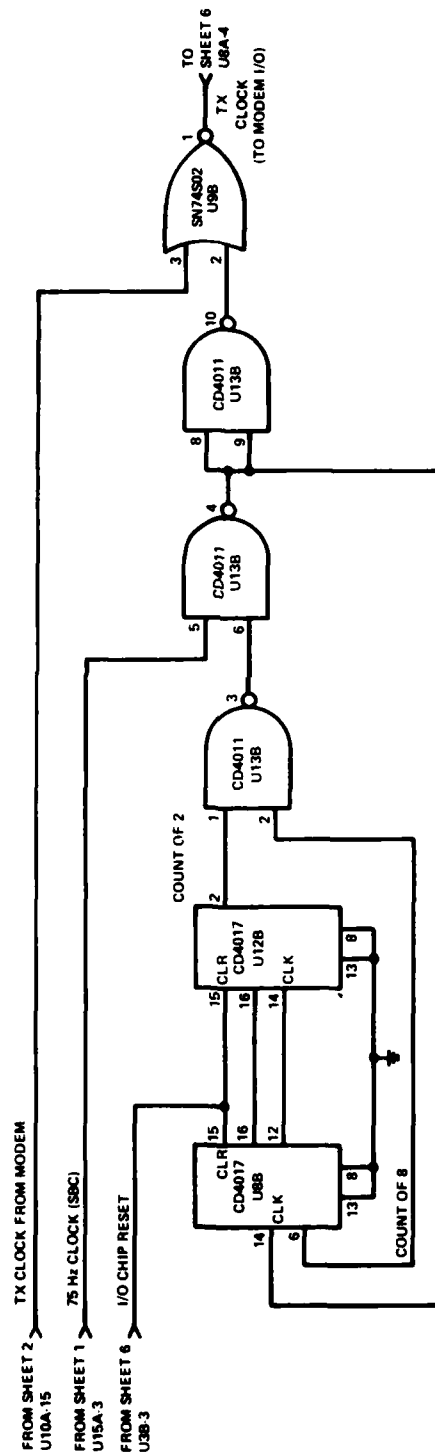


Figure 5-8 TX Character Deletion (Sheet 5)

5.4.6 Modem/MPU-to-SBC Serial Data Adapter

Figure 5-9 shows the modem/MPU-to-SBC serial data adapter circuit. Its primary function is to transfer serial data to/from the AFSATCOM modem or MPU to/from the SBC 80/20. The SBC 80/20 reads or writes these data as if they were located in a memory address. Selection of either the modem I/O or MPU I/O takes place via a register select line with read/write operations controlled separately.

The heart of this circuitry consists of a pair of Motorola MC6852 programmable synchronous serial data adapter (SSDA) ICs. Each provides a three-character buffered bi-directional serial interface for synchronous data exchange. The MC6852 design incorporates bus interface logic to allow parallel data transfer over the SBC 80/20 bi-directional parallel data bus. The actual configuration of the SSDA is programmed via the data bus using the polling program system initialization software. SSDA internal programmable control registers provide word length, transmit, receive, synchronization, and interrupt controls. Status, timing, and other SSDA control lines provide additional peripheral or modem functions.

5.4.7 SBC 80/20 Interfaces

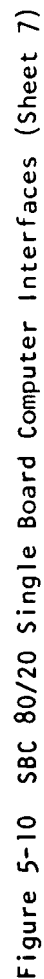
Figure 5-10 summarizes the I/O and display interfaces within the microcomputer system. Jumpers and parallel terminations employed, along with physical board modifications for the SBC 80/20, are indicated.

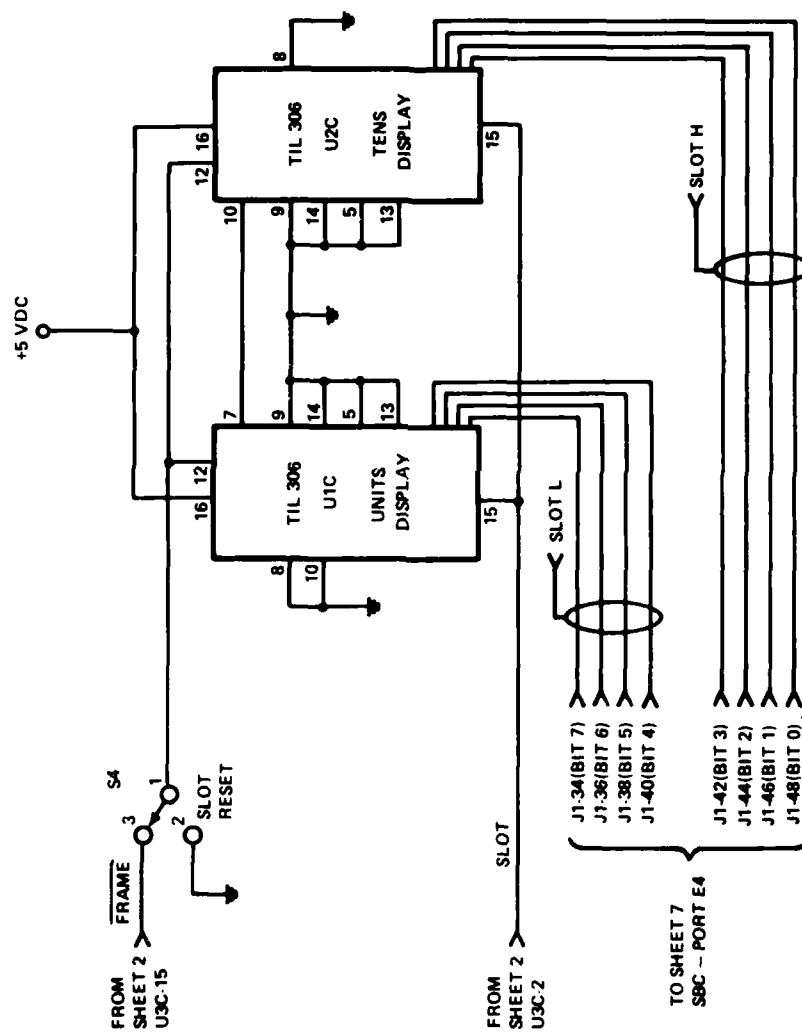
5.4.8 Slot Counter/Display

Figure 5-11 shows the slot counter/display circuit interfaces to the SBC 80/20 and the I/O circuits. Slot pulses from the AFSATCOM synchronizer are used to increment the slot display on a pair of Texas Instruments TIL 306 ICs while the frame pulse from the synchronizer (or a manual toggle switch) provides a reset function back to zero slot indication. A pair of four-bit binary coded decimal (BCD) output lines (8 parallel lines, total) to one of the SBC 80/20 parallel input ports is used by the polling software in calculating the current AFSATCOM slot. As implemented in the demonstration system software, zero-slot indications signal the non-TDM AFSATCOM mode and non-zero indications signal TDM operation.



41





5.4.9 RS-232 Serial Interfaces

Figure 5-12 summarizes the cable interfaces employed with the microcomputer system and made possible by serial RS-232 interface circuitry which employs a software-programmable 8251 USART IC on the SBC 80/20 card. Shown are cable interfaces for use with a Bell 103J modem, a "NULL" modem, a Lear Siegler ADM-3A CRT terminal, and a Texas Instruments Model 765 Intelligent Terminal.

When the polling software is commanded into its RS-232 mode, all messages normally routed to the AFSATCOM MPU interface are rerouted to the RS-232 serial interface port at a 300 b/s rate (instead of 75 b/s). Messages destined for transmission to the AFSATCOM modem or COMSUP commands normally originating at an AFSATCOM ASR device may now be entered into this RS-232 interface. While in this mode, the AFSATCOM ASR may still be used for all transmitting functions; however normal reception from the OW (or NB-1) channel will be diverted to this RS-232 port. When not in the RS-232 mode, normal NB interfaces to the MPU are utilized.

5.5 MODULAR RACK HARDWARE FEATURES

When the microcomputer rack assembly is installed in the Type 12 terminal, two multi-pole bypass switches mounted on the front panel can disengage the microcomputer system from the MPU, NB modem, and synchronizer interfaces without physical removal of connectors. Connector terminations on the rear of the rack provide convenient access for cables to the AFSATCOM Type 12 CP terminal.

The modular rack assembly is designed for ease in removing all mounted components and subsystems. Test points mounted on the front panel of the rack also provide a convenient access to all interfaces for ease in trouble-shooting. Front panel fuses on the rack serve to protect the various power supplies in the power supply subsystem. An on-off power line switch and a line voltage indicator are below these fuses.

Finally, a slot display reset switch and eight BCD slot selection switches are also front-panel mounted for convenient control of polling test functions related to TDM and non-TDM modes.

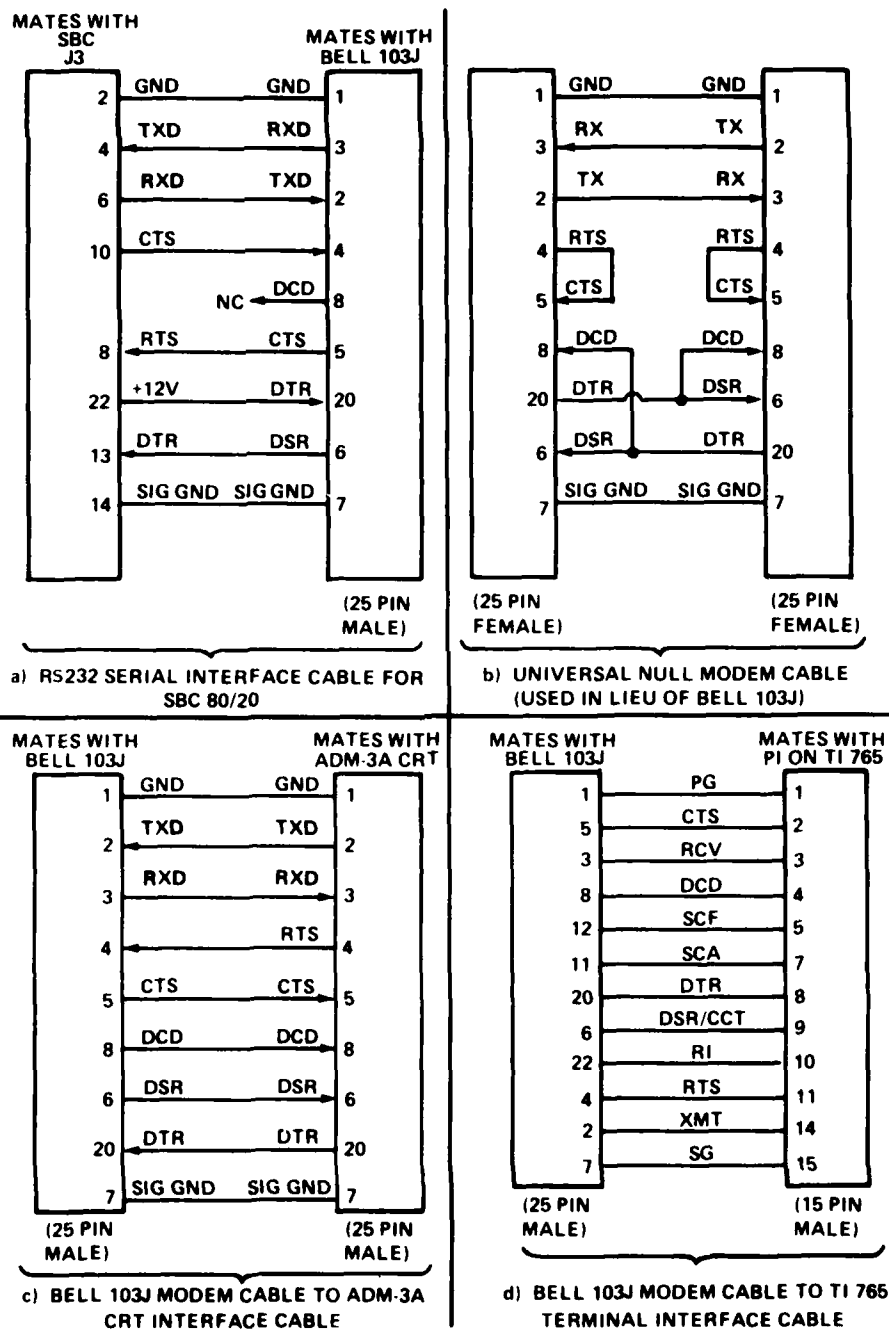


Figure 5-12 RS-232 Serial Interfaces

SECTION 6

MICROCOMPUTER SOFTWARE

Figure 6-1 shows the major elements of the microcomputer facility at MITRE-Bedford which was used for overall software development for the polling improvement project. The Tektronix 8002A software development system and the MITRE Time Sharing Option (TSO) system assets related to polling software development are described below.

The microcomputer software for the improved polling demonstration capability was written in PL/M-80 high-level language on TSO. The PL/M-80 source coding, which is essentially self-documenting, takes advantage of modular interrupt-free, structured programming techniques and attempts to minimize (but not entirely eliminate) interactions between the various modular segments of the overall program.

After appropriate compilation and debug on TSO, followed by downloading to the Tektronix 8002A microprocessor laboratory, executable code was installed in the microcomputer system using 2708 EPROM non-volatile memory ICs. Execution of this code in the microcomputer system provides the processing functions needed to satisfy the polling enhancements described in section 4.

6.1 FLOW CHARTS

The polling software for the microcomputer system is functionally described by the flow chart of figure 6-2. Following power turn-on of the microcomputer system, the software executes the Initialization Sequence, followed by the main program loop starting at the Slot Calculation Algorithm and ending at the TX Output Algorithm. Return to the Initialization Sequence is also possible by means of a software re-initialization COMSUP command. Unless the software is re-initialized, however, normal program execution continues in this endless loop. The time-to-completion of a single cycle within this loop is typically less than a single character interval at the AFSATCOM 75 b/s rate. Coupled with this capability is a three-character buffer in the serial synchronous MPU and modem interfaces employed for use with the AFSATCOM hardware (see subsection 5.4.6) which avoids loss of data during program execution in an interrupt-free manner.

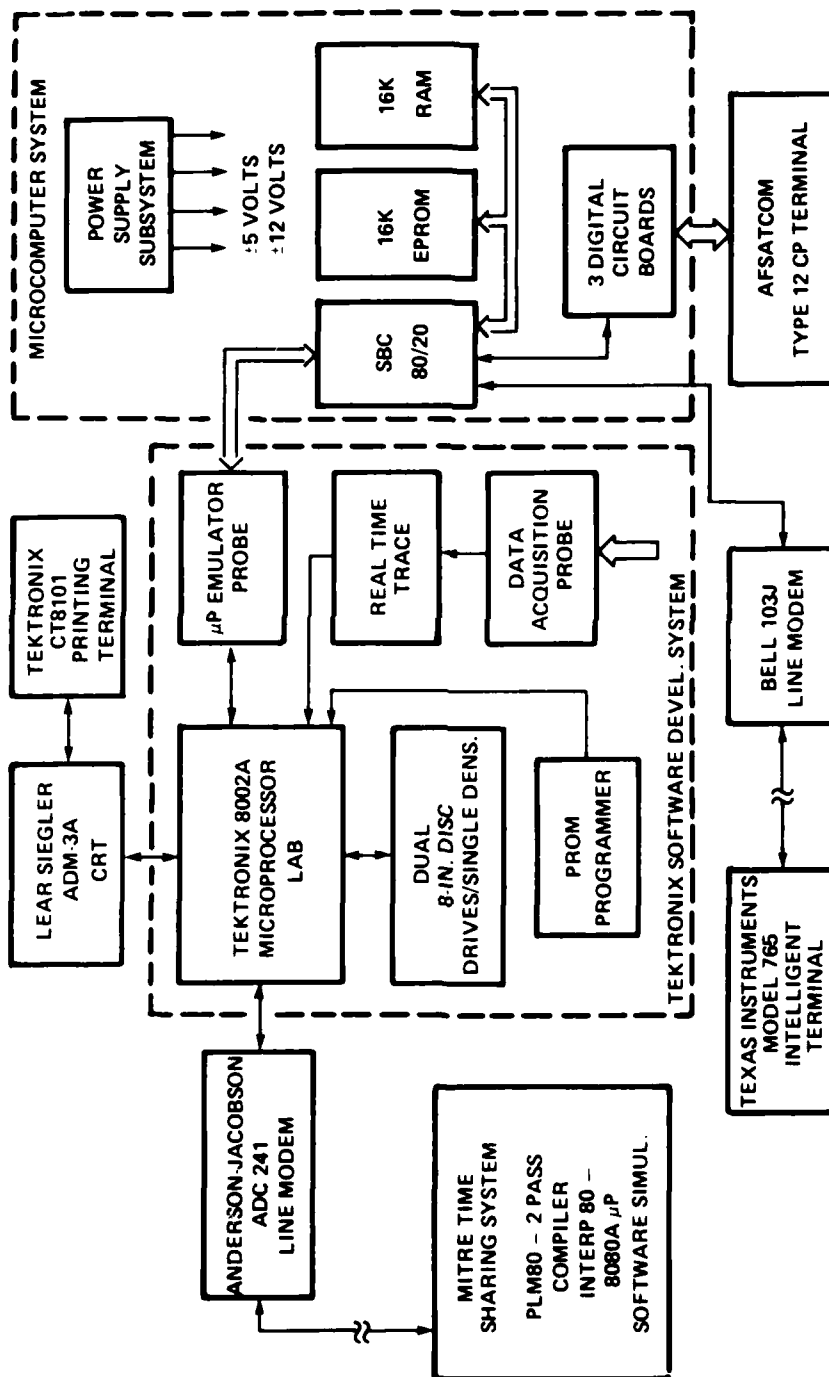


Figure 6-1 Microcomputer Development Facility



Figure 6-2 Polling Software Program Flowchart

Table 6-1 describes the various program modules along with their major performance characteristics during program execution. The polling algorithm is the heart of the overall polling program software; a more detailed flow chart of this polling algorithm is provided in figure 6-3.

6.2 PL/M-80 SOURCE CODE AND MEMORY ALLOCATIONS

Table 6-2 contains the improved polling software symbol table and memory map which identifies RAM and ROM (EPROM) label locations, variables, and constants used throughout the software. This table is consistent with the program module PL/M-80 labels shown in table 6-1 and the address locations in table 5-1 so that the start of each major program module may be readily identified. This also allows ease in modifying or upgrading existing software by appropriate EPROM replacement within the microcomputer system. Appendix B contains the actual PL/M-80 source code which was written on TSO.

6.3 MITRE TSO SYSTEM PL/M-80 SUPPORT

PL/M-80 assets available on TSO which were utilized for the microcomputer software development are described below.

6.3.1 PL/M-80 High Level Language

PL/M-80 is an advanced high level programming language available on the MITRE TSO system. Specifically designed to simplify the job of system programming for the Intel 8080 8-bit microprocessor, it provides a very effective software development and maintenance tool, well suited to the requirements of the microcomputer system designer. PL/M-80 is also easy to learn and facilitates rapid program development and debugging since it is an algorithmic language in which the program consists of a sequence of declarations and executable statements naturally expressing the algorithm to be performed. Thus, the programmer is free to concentrate on system development through use of modern structured programming techniques at a high level, rather than dealing with assembly language details (such as register allocation, etc.). For a complete description of PL/M-80 see reference 3.

6.3.2 PL/M-80 2-Pass Cross-Compiler

The 8080 PL/M cross-compiler consists of two distinct programs which must be executed consecutively to perform a complete

Table 6-1

Program Software Module Description

<u>Module</u>	<u>Label(s)</u>	<u>Functions</u>
Initialization Sequence	Program\$Start	Defines I/O port parameters, declares/initializes constants/variables; initializes key memory locations; initializes SBC programmable counters; defines I/O memory locations.
Slot Calculation Algorithm	IPOLL 1	Calculates current slot; sets TDM mode flag; converts BCD input from slot display to decimal using conversion algorithm.
Polling Algorithm	POLL\$XMIT	Checks poll mode flag; checks for polling completion; checks for TDM or non-TDM polling mode; checks for poll interrupt; signals completion of polling; updates EDAC table entry during non-TDM polling.
RX Input Algorithm	IPOLL 11	Tests RX input buffer character status of I/O chip; fetches RX character when available; tests for last character received state; tests for RX buffer memory overflow; tests for encrypted mode initialization sequence; tests mode/address characters for subsequent processing sequence (sets MPU job flag).
EDAC Algorithm	EDAC\$ROUTINE	Provides EDAC for polling messages transmitted via downlink RX monitoring. (If EDAC criteria are satisfied, sets retransmit flag for a non-TDM mode polling message for up to three retransmissions.)
RX Mode Tests	MJT2	Tests RX mode characters from modem for check-ins, reportbacks, and table transfers; provides routing to various RX mode algorithms.

Table 6-1 (Continued)

Program Software Module Description

<u>Module</u>	<u>Label(s)</u>	<u>Functions</u>
RX Check-in Algorithm	CS\$TEST	Identifies invalid check-ins from modem; logs valid check-ins into appropriate check-in tables; composes output message for MPU with appropriate check-in headers; formats overflow message when check-ins are at limit for MPU (and also for modem when poll mode is not active); automatically cancels check-in mode when check-ins are at limit.
RX Reportback Algorithm	PRB\$LOAD RRB\$LOAD NRB\$LOAD	Composes output message to MPU with appropriate reportback header when a valid reportback is received from modem.
RX Table XFER Algorithm	PTT\$LOAD RRB\$LOAD NTT\$LOAD	Identifies invalid table transfers received from the modem; logs valid table transfers into appropriate check-in tables; composes output messages to MPU with appropriate table transfer headers; formats overflow messages when check-ins are at limit for MPU (and also for modem when poll mode is not active); automatically cancels check-in mode when check-ins are at limit.
RX Output Msg Algorithm	MPU\$TABLE\$BUILD	Checks if message delay is active; increments delay counter as required if msg delay is enabled; verifies need to output a message; checks availability of buffer space for outputting a character, outputs data to either MPU output (memory I/O) or RS-232 I/O port, depending on which is activated; shifts down internal storage buffer upon completion of message transmittal; enables clock output to MPU on start of message transmittal and disables it on completion.

Table 6-1 (Continued)

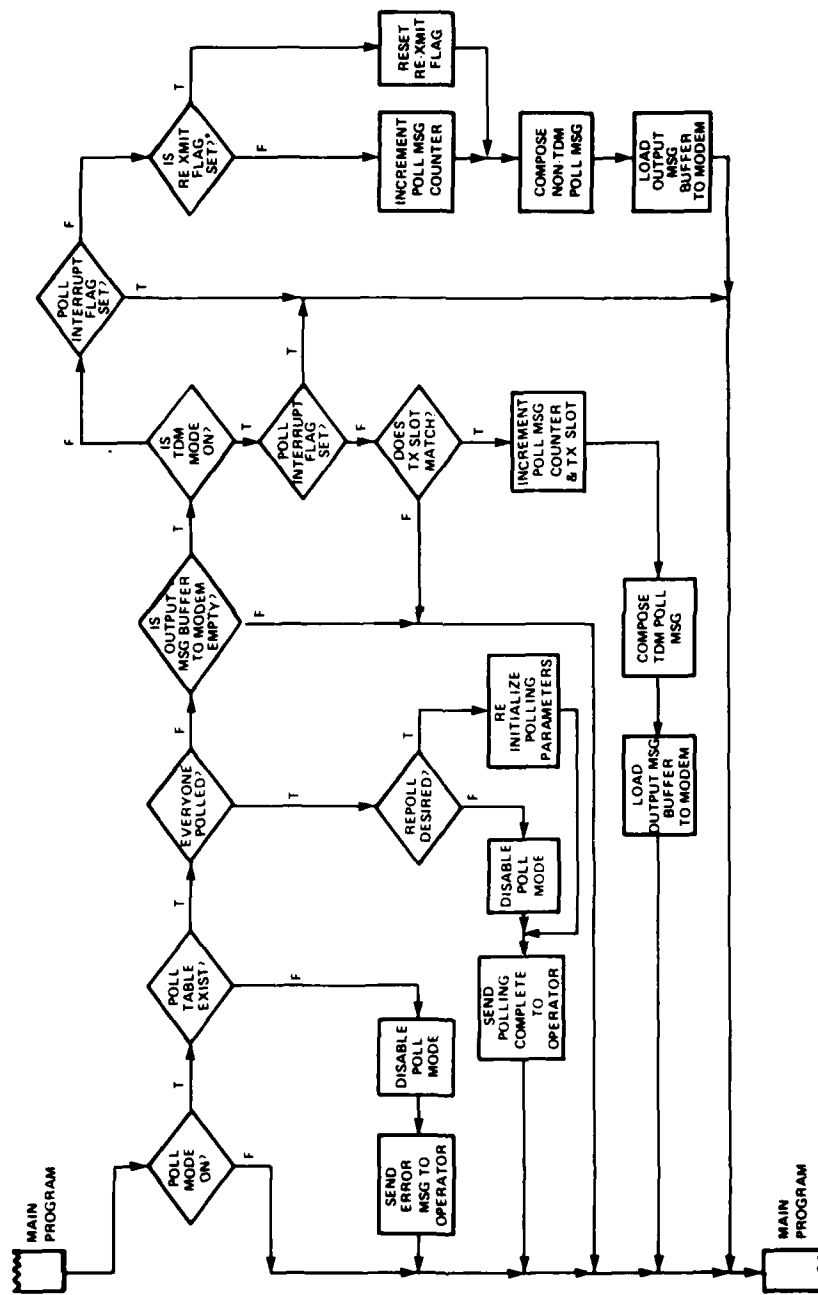
Program Software Module Description

<u>Module</u>	<u>Label(s)</u>	<u>Functions</u>
RX Input Algorithm	IPOLL101	Tests receive input buffer character status of RS-232 I/O if activated; fetches character if available; tests receive input buffer character status of I/O chip; fetches character if available; tests for last character status; tests for buffer overflow; tests for valid commands and determines processing sequence by setting an output flag; determines when complete message has been received; verifies need to ignore all but command-type messages.
COMSUP Recognition Algorithm	COMPOSE1	Checks for correct command-type message lengths; validates receipt of correct address characters; identifies various mode and address errors, checks for valid command-type mode characters; routes command-type message to appropriate command message algorithms.
COMSUP Msg Formatter (To MPU/RS-232)	COMSUPLOAD1	Formats command-type messages to MPU input or RS-232 I/O interface.
Error Msg Formatter (To MPU/RS-232)	MODEERROR1 ADRERROR1 OVERFLOW1 OVERFLOW2 TFRERROR1	Formats appropriate error messages to MPU input or RS-232 I/O interface.

Table 6-1 (Concluded)

Program Software Module Description

<u>Module</u>	<u>Label(s)</u>	<u>Functions</u>
TX Output Msg Formatter (To Modem)	MODEM\$OUTPUT\$- COMPOSE1 MODEM\$OUTPUT\$- COMPOSE2	Formats output messages to modem; deletes first three characters upon recognition of special character in position #4; resets output flag for modem messages.
TX Output	OPOLL101	Checks if message delay is activated; increments appropriate delay counter as required if delay is active for TDM and non-TDM modes; verifies need to output a message; monitors state of receive input messages from modem to avoid transmitting while messages are received; signals modem when message is ready for transmission; shifts down internal storage buffer on completion of message transmittal.



* FLAG SETTING OCCURS IN ERROR DETECTION AND CORRECTION (EDAC) SOFTWARE ALGORITHM

Figure 6-3 Polling Algorithm Flowchart

Table 6-2

Symbol Table and Memory Map

<u>Name</u>	<u>Location (Hex)</u>	<u>Name</u>	<u>Location (Hex)</u>
ADR1	4638	CLK3	4662
ADR2	4639	CLKCNT	5792
ADR3	463A	CLKCNT1	4664
ADRCHANGE	1B80	CLKCNT2	4666
ADRERROR1	1D67	CLKCNT3	4668
ADRERRORHDR	037C	CLKP	579A
AUTOPOLLRESTART	0611	CLKP1	465E
AUTOREPOLLFLAG	4684	CLKP2	4661
BUFFERDELAY	579C	CLKP3	4663
BUFFERDELAY1	466A	CMODE	6661
CC	6668	CMODECANCEL	0D99
CC1	5760	CMODECANCEL1	0DFC
CCPRESENT	4658	COMPOSE1	17FF
CC1PRESENT	465A	COMSUPLOAD1	296F
CCTEST	0B25	CS	5775
CHAR	5776	CS1	4688
CHAR1	4689	CS3	464A
CHAR3T	0998	CS4	464B
CHAR4T	09B1	CSDATAPOINTER	5764
CHAR4T1	166B	CS1DATAPOINTER	576C
CHAR5T	09CA	CSTEST	0B13
CHARSTORE	5D64	DELAY	579B
CHARSTORE1	468E	DELAY1	465F
CHECKINDELETE	247A	DELAYCAL	1D3A
CHECKINMODESET1	28D5	DELAYROUTINE	1442
CINUMTEST	0D76	DELAYROUTINE1	03D7
CLK	5799	EATMSG1	1773
CLK1	465D	EATMSGFLAG1	465C
CLK2	4660	EATMSGSET	1AF3

Table 6-2 (continued)

<u>Name</u>	<u>Location (Hex)</u>	<u>Name</u>	<u>Location (Hex)</u>
EDACFLAG	4656	INPUT5DATAPOINTER	5768
EDACMODESET	1B4A	INPUT103DATAPOINTER	576E
EDACROUTINE	0A33	INPUT105DATAPOINTER	5770
ENC	5777	J	5762
ENCRYPT	0937	LCT	09E0
ENCTEST	0904	LCT1	1685
ENDLOAD	3833	LINECOUNT	468D
EOMTEST	13E0	LOADADR	37A1
EOMTEST1	3D3F	MEMORY	7400
ERRORHDR	039A	MJF	5778
GC	08BD	MJT1	093E
GC1	1614	MJT2	0AC7
GCIPOINTER	578C	MODE	5779
GCIRESET	2960	MODEERROR1	2A08
GCISTORE	5C23	MODEMIOCTL1POINTER	4632
GPOLLBUILD1	3C92	MODEMIOCTL2POINTER	4634
GT1	2278	MODEMOUTPUTCOMPOSE1	1700
GTABLEBUILD1	3548	MODEMOUTPUTCOMPOSE2	1783
GTABLEDUMP1	34C1	MPUCINUMTEST1	2824
GTDHEADER	0362	MPUIOCTL1POINTER	462E
I	6662	MPUIOCTL2POINTER	4630
I1	575A	MPUTABLEBUILD	0E58
INITIALMSG	0461	MSGCOUNT	6534
IPLTEST	1915	MSGCOUNT1	4E5E
IPOLL1	04BF	MSGDELAY	579E
IPOLL11	087C	MSGOVERDELAY	4672
IPOLL101	15A8	N	577C
IPOLL1011	15D4	NC	5786
INPUT3DATAPOINTER	5766	NCIHEADER	02CD

Table 6-2 (continued)

<u>Name</u>	<u>Location (Hex)</u>	<u>Name</u>	<u>Location (Hex)</u>
NCIPointer	5790	OPOLL1	13BB
NCIReset	2945	OPOLL101	3D1A
NCISTORE	5AA2	OUT1	468A
NCITEST	0CC8	OUTCHAR	148C
NOCOMSUPLOAD1	29FD	OUTCHAR1	3EA7
NONTDMDELAY1	3DF6	OUTCOUNT	6666
NONTDMDELAYSET	1CD2	OUTFLAG1	1680
NONTDMMSGDELAY1	466C	OUTPUTCOMPOSE	16CF
NONTDMPOLL	0622	OUTPUT1DATAPOINTER	576A
NPOLLBUILD1	3C12	OUTPUT101DATAPOINTER	5772
NRBHEADER	02E1	OUTTABLE	666B
NRBLOAD	0F9A	OUTTABLE1	4F8A
NT1	2201	OUTTABLEBUILD	0E64
NTABLEBUILD1	339B	OUTTABLE1STATUSTEST	05E9
NTABLEDUMP1	3314	OVERFLOW1	2847
NTDHEADER	0343	OVERFLOW2	284C
NTDMDELAYH	467C	OVERFLOWHDR	03D9
NTDMDELAYL	467E	PCIHEADER	0241
NTDMEDACTABLE	464F	PCIPointer	5784
NTDMPOLLINTERRUPTFLAG	4686	PCIReset	292D
NTTHEADER	02F5	PCISTORE	57A0
NTTLOAD	1278	PCITEST	0B43
NTTLOAD1	200A	POLLDUMPBUILD1	372F
NTTLOADEND	1348	POLLDUMPHDR	03F4
NTTR1	2CEB	POLLFLAG	4683
NUMGCI	577E	POLLINGMODETEST	05FF
NUMPCI	578E	POLLINTERRUPT	1C5B
NUMPCI	5792	POLLMODE	1BB4
NUMRCI	5783	POLLMODEERROR	0550

Table 6-2 (continued)

<u>Name</u>	<u>Location (Hex)</u>	<u>Name</u>	<u>Location (Hex)</u>
POLLMSGCOMPLETE	0580	PTABLEBUILD1	3042
POLLMSGCOUNT	464C	PTABLEDUMP1	2FBB
POLLMSGOVER	0410	PT1COUNT1	73FE
POLLOVERTEST	0558	PTDHEADER	030A
POLLPOINTER1	73FC	PTTHEADER	02E5
POLL SLOT1H	463B	PTTLOAD	0FF3
POLL SLOT2H	463C	PTTLOAD1	1E3D
POLL SLOT3H	463D	PTTLOADEND	10C2
POLL SLOT4H	463E	PTTR1	2B45
POLL SLOT5H	463F	QUIT	3FAB
POLL SLOT6H	4640	RCIHEADER	02C9
POLL SLOT1L	4641	RCIPOINTER	578A
POLL SLOT2L	4642	RCIRESET	293E
POLL SLOT3L	4643	RCISTORE	5921
POLL SLOT4L	4644	RCITEST	0C05
POLL SLOT5L	4645	REPOLLFLAGCOUNT	4657
POLL SLOT6L	4646	REPOLLTEST	0572
POLLTABLE1	6E3B	RETRANSMITFLAG	4676
POLLTABLEBUILD1	3B34	RETRANSMITFLAGSET	0AAD
POLLTABLEDUMP1	36B1	RPOLLBUILD1	3B96
POLLTABLEDUMP2	366E	RRBHEADER	02DD
POLLTABLETX	3A3E	RRBLOAD	0F41
POLLTABLETXFLAG	4685	RS232FLAG	4637
POLLTTR1	2DBE	RS232GC1	161D
POLLXMIT	0530	RS232INPUT1	15B9
PRBHEADER	02D1	RS232MODESET	1ABD
PRBLOAD	0EE8	RS232OUTCHAR	14B1
PROGRAMSTART	0007	RS232SPACETEST	1427
PT1	2114	RT1	218A

Table 6-2 (concluded)

<u>Name</u>	<u>Location (Hex)</u>	<u>Name</u>	<u>Location (Hex)</u>
RTABLEBUILD1	31EE	TCLK	466F
RTABLEDUMP1	3167	TCLK1	4673
RTDHEADER	0327	TCLKCNT	4671
RTTHEADER	02F1	TCLKCNT1	4675
RTTLOAD	1135	TCLKP	4670
RTTLOAD1	1F23	TCLKP1	4674
RTTLOADEND	1205	TDMDELAYROUTINE1	3DB3
RTTR1	2C18	TDMFLAG	4682
RXBUSYOVERRIDE1	3E60	TDMPOLL	0739
SHIFTDOWN	14C8	TDMPOLLINTERRUPTFLAG	4687
SHIFTDOWN1	3EC2	TDMPOLLTXSLOT	4677
SLOT	467A	TDMPOLLTXSLOTINITIAL	4649
SLOTCOUNT	467B	TFRERROR1	2E7B
SLOTGUARDTIME	4680	TFRERRORHDR	03B8
SLOTH	4678	TOTALNUMCI	5780
SLOTL	4679	TTLOADEND1	20F1
SLOTREASSIGN	19AD	TTR1	2A94
SPACE	577B	TXSLOTINITIALH	4647
SPACE1	468C	TXSLOTINITIALL	4648
SPACETEST	1401	WAITING	577A
SPACETEST1	03D5	WAITING1	468B
ST	0916	WW	08A7
STARINSERT	0B1C	XX	089B
STARTUPMSG	042E	YY	15F2
TBLDUMP1	2EE1	ZZ	15FE
TC	6664		
TC1	575C		
TCINITIAL	5794		
TC1INITIAL	5796		

compilation of a PL/M source program. The two programs are known as Pass 1 (PL/M-81) and Pass 2 (PL/M-82) of the PL/M compiler. They are written in ANSI standard Fortran IV and are installed on the MITRE TSO system.

The first pass of the compiler reads a PL/M source program and converts it to an intermediate form on work files. As an option, a listing of the input source program may be obtained during this pass. Errors in program syntax are detected at this stage and appropriate error messages are sent to the list file.

The second pass of the PL/M compiler processes the intermediate files created by Pass 1 and generates machine code. This machine code, which can be in either BNPF or hex format, may be loaded and executed on the SBC 80/20 microcomputer, on the Tektronix 8002A, or simulated using INTERP/80, a cross-simulator of the 8080 microprocessor unit. It may also be used for programming ROMs. Pass 2 of the compilation process can produce a symbol table and mnemonic listing of the generated machine code. Errors detected during this phase will be reported in the list file which is produced.

Figure 6-4 illustrates the overall file structure and flow of program execution of the PL/M cross-compiler available at MITRE. For a complete description of the PL/M-80 2-pass cross-compiler see reference 4.

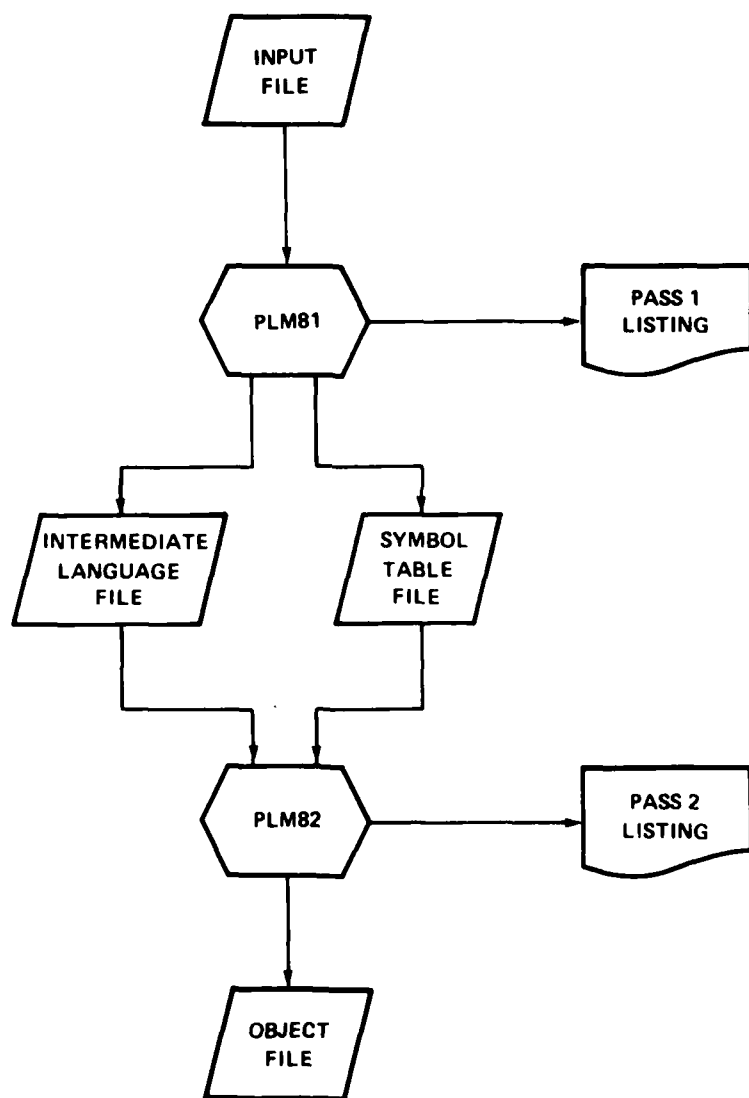
The run-time organization of the memory storage allocation for a compiled PL/M-80 object program is shown in figure 6-5. Memory is allocated in three sections:

1. Instruction Storage Area (ISA).
2. Variable Storage Area (VSA)
3. Free Storage Area (FSA)

The ISA is occupied by the machine code generated by the PL/M source and variables declared in DATA declarations.

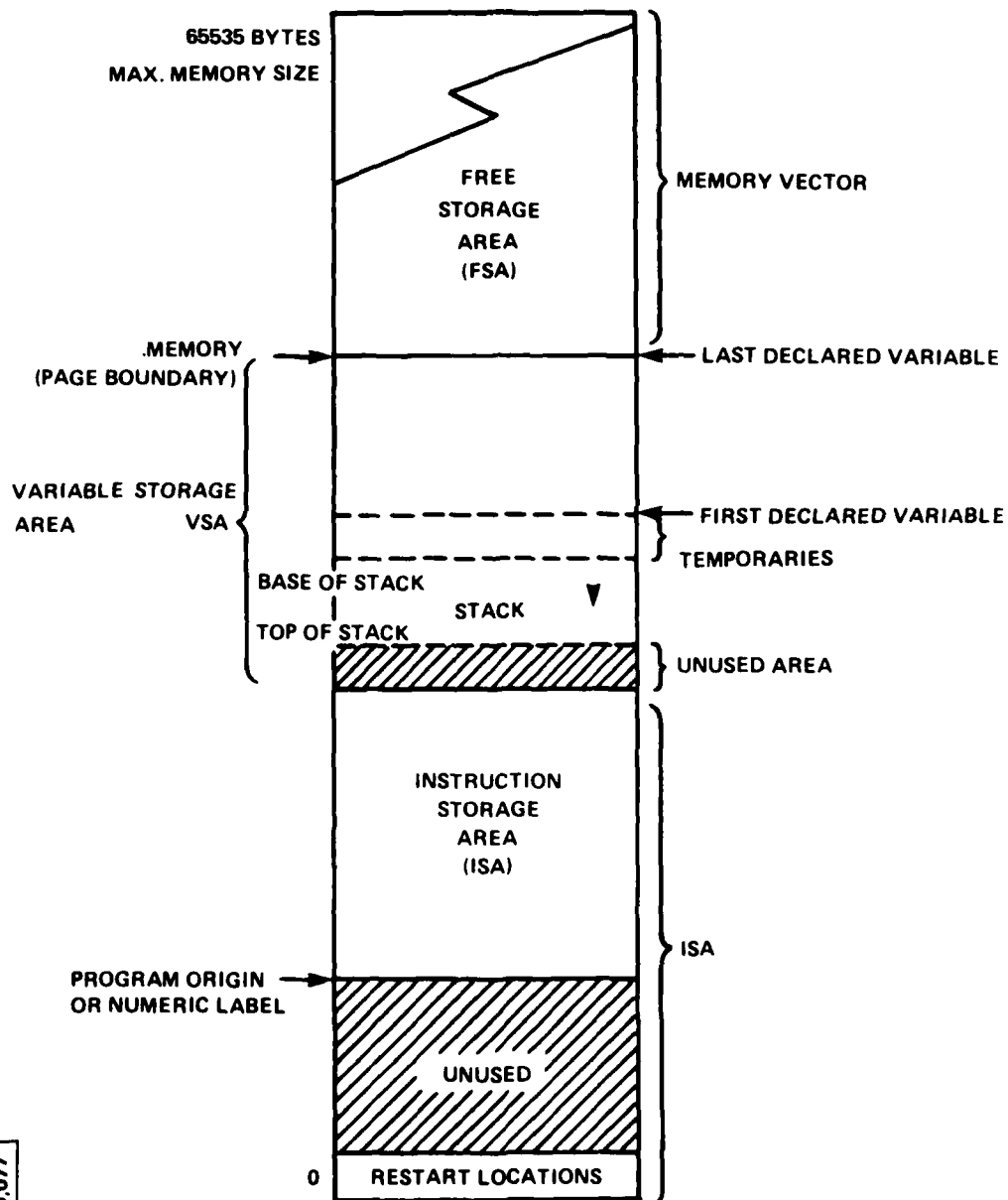
The VSA is located above the ISA, and contains (in order of decreasing address):

1. Variables, other than DATA variables, declared in the PL/M source. They are arranged in order of declaration. ADDRESS variables are not aligned.
2. Compiler generated temporaries (i.e., workspace used in the object program, but not explicitly declared).



1A-59,076

Figure 6-4 File Structure and Flow of Program Execution on 8080 PL/M Cross Compiler



1A-59,077

Figure 6-5 Run-Time Storage Organization of Memory Storage Allocation

3. The stack. The size of the stack area is determined by the compiler, unless explicit overrides are used.

The compiler will normally locate the VSA directly above the ISA. However, the compiler user may specify the first page of memory of the VSA explicitly. (A page of memory contains 256 bytes.) This may be used, for example, to ensure that the VSA is located in RAM for a system that has both RAM and ROM.

FSA is the area of memory above the VSA. The PL/M identifier MEMORY may be used to reference the FSA.

6.3.3 INTERP/80 for the 8080 Microprocessor

An INTERP/80 program available on the MITRE TSO system provides a software simulation of the Intel 8080 CPU, along with "free form" execution monitoring commands to aid in system software development for 8080-based microcomputers.

INTERP/80 accepts machine code produced by the PL/M 8080 cross-compiler, along with execution commands from a time-sharing terminal. The execution commands allow manipulation of the simulated system memory and the 8080 CPU registers. In addition, operation and instruction break-points may be set to stop execution at crucial points in the program. Tracing features are available which allow the CPU operation to be monitored. INTERP/80 also provides symbolic reference to memory storage locations as well as numeric reference in various number bases. Some of the commands available in INTERP/80 are presented in table 6-3. For a complete description of INTERP/80, see reference 5.

6.4 TEKTRONIX 8002A SOFTWARE DEVELOPMENT SYSTEM

The heart of the Tektronix software development system is the Tektronix 8002A microprocessor laboratory. The system architecture of the Tektronix 8002A centers around three microprocessors incorporated into circuit card modules. The system processor, using TEKDOS operating system commands, controls system activity such as organizing, storing, and retrieving system and user programs on the disc drives, executing the text and debug programs, supervising the emulator processor through separate debug hardware, and directing all I/O activity for various system peripherals such as the flexible disc unit, the system terminal, and a line printer. See figure 6-6 which shows the 8002A microprocessor laboratory used in conjunction with the Intel SBC 80/20 single board computer.

Table 6-3

INTERP/80 Commands

<u>Command</u>	<u>Function</u>
LOAD	Causes symbol tables and code to be loaded into the simulated MCS-80 memory.
GO	Starts execution of the loaded 8080 code.
(NO) INTER	Simulates an 8080 interrupt.
TIME	Sets and displays the simulated 8080 cycle counter.
CYCLE	Allows the simulated CPU to be stopped after a given number of cycles.
(NO) TRACE	Enables tracing feature when particular portions of the program are executed.
(NO) REFER	Causes the CPU simulation to stop when a particular storage location is referenced.
(NO) ALTER	Causes the CPU simulation to stop when the contents of a particular memory location are altered.
CONV	Displays the values of numbers converted to the various number bases.
DISPLAY	Displays memory locations, CPU registers, symbolic locations, and I/O ports.
SET	Allows the values of memory locations, CPU registers, and I/O ports to be altered.
BASE	Allows the default number base used for output to be changed.
(NO) INPUT	Controls simulated 8080 input ports.
(NO) OUTPUT	Controls simulated 8080 output ports.
PUNCH	Causes output of machine code in BPNF or hexadecimal format.
END	Terminates execution of an 8080 program.



Figure 6-6 Tektronix 8002A Microprocessor Laboratory Workstation

The emulator processor, a system option for each microprocessor the 8002A can support, is controlled by the system processor. Via the debug module hardware, the emulator processor performs two functions. In emulation mode 0, this processor executes the user program so that run-time and logic errors can be detected before actual software/hardware integration begins. In emulation modes 1 and 2, the emulator processor, operating in conjunction with the prototype control probe, provides the capabilities for complete software/hardware debugging and integration. The control probe plugs into the microprocessor socket on the prototype circuit board, thereby enabling the emulator processor to take the place of the actual microprocessor that ultimately is installed in the user hardware. In emulation mode 1, the emulator processor executes the user program under supervision of the debugging system in program memory and memory mapped to the prototype. I/O and clock signals are also available to the user hardware. In emulation mode 2, the emulator processor executes the user program fully in the prototype hardware with all memory, I/O, and clocking functions made available. The debug system, however, still maintains control of the emulator processor.

The third microprocessor, the assembler, runs the relocatable macro assembly program when the TEKDOS assembly command is invoked. Upon completion of assembly, assembled object code is stored on disc memory in hexadecimal format. The assembler performs its function for each separate microprocessor supported by the 8002A with the installation of the appropriate assembler software.

Other circuit card modules within the 8002A provide supporting software and hardware functions. The system memory contains 16K bytes of dynamic RAM and a bootstrap loader resident in 256 bytes of ROM and is accessed only by the system processor. The system memory is automatically loaded with the resident portions of TEKDOS each time the 8002A system is powered up. It also provides buffer space for all I/O activities.

A maximum of 64K bytes of program memory resides on separate circuit-card modules within the 8002A mainframe and can be accessed by the system processor or the emulator processor. Program memory is used by the system processor as a text buffer during text editing sessions and is available during user program execution as a storage area in conjunction with the emulator processor. The feature of separate system/program memory structure allows the system memory to maintain its integrity at all times should the program crash for any reason, thus allowing the operating system software to remain intact.

System components within the 8002A are joined by a common system bus which is divided into a system side and a program side. The debug module provides the hardware interface between the system and emulator processors while the system communications module provides EIA standard RS-232C interfaces for all peripheral devices except the dual flexible disc drive unit and the associated system terminal.

Storage of about 660K bytes on dual disc drives (330K bytes per single-sided disc) is available as the mass storage medium for the 8002A. The drive unit communicates directly with the system processor module through an interconnecting cable.

The real-time trace module, an eight-clip test probe, and the data acquisition interface panel make up the real-time prototype analyzer of the 8002A. The module provides a high-speed buffer memory capable of retaining 128 data words in a dynamic fashion. Each acquired word, composed of 48 bits, contains 16 address bits, eight of 16 bits from the system bus, and eight data bits from the test probe. An additional eight bits identify the type of cycle such as read, write, I/O memory, or instruction fetch. The module also contains the controlling logic circuitry for utilizing the analyzer's command repertoire. The test probe itself is an eight-pin, high-input impedance device which may be connected during emulation modes 1 and 2 to any locations on the user prototype hardware. Fixed TTL or variable threshold levels are switch-selectable on the interface unit, with acquired data buffered by the probe and then fed to the real-time trace module via the cable interface. Data acquired by the trace module and the eight channels of externally acquired data are thus subject to the same TEKDOS command set of the 8002A. Finally, two BNC connectors on the data acquisition interface can also be used to trigger a logic analyzer or scope, thereby allowing greater trouble-shooting capability of prototype circuitry during program execution while maintaining debugging control through the 8002A.

The PROM programmer within the 8002A can support two PROM programming options, one for 1702A PROMs and one for 2704/2708 PROMs. Each option consists of the appropriate plug-in circuit card and PROM programming software to supervise and control the transfer of user programs between program memory and the PROM chip. Interface to the PROM programmer is on the front panel of the 8002A system mainframe, via zero-insertion-force sockets.

The ability of the 8002A to communicate with external host processors provides the flexibility of writing software externally to the 8002A and downloading either source program or source code.

This is particularly useful when high-level language support is separately available, as is the case on the MITRE TSO system.

In summary, the 8002A is a complete design tool for software development to eventual software/hardware integration. Its powerful operating system software, TEKDOS, performs all utility routines, maintains the debugging system, provides emulation support routines, and controls the PROM programmer. The 8002A can support a variety of currently available 8-bit and 16-bit microprocessors and offers full prototype test and emulation capability at three progressive levels.

SECTION 7

CONCLUSIONS/RECOMMENDATIONS

The results of the improved polling project described lead to the following conclusions and recommendations:

- Microcomputer interfacing techniques offer a powerful technique for simulating new AFSATCOM satellite terminal polling performance capabilities.
- This successful microcomputer application was attributable to availability and acquisition (where necessary) of comprehensive computer hardware facilities and high-level language software tools designed to support personnel having little or no background in microprocessor-based techniques. Without these facilities and tools several additional personnel, particularly experienced programmers, would have been required.
- Polling improvements using existing AFSATCOM terminal interfaces via MPU software-only changes appear to offer significant new enhancements to a large segment of the AFSATCOM user community.
- Providing an on-line access to demonstrate these potential capabilities to AFSATCOM users without the need to modify either AFSATCOM hardware or software has proven to be cost-effective. This concept should be pursued in accordance with the follow-on demonstration objectives/plans described below.
- The microcomputer concept and approach described appears to also lend itself to other similar applications. The fully established facilities and proven software support tools for AFSATCOM at MITRE-Bedford should be used in the assessment of future terminal upgrades.
- The feasibility of serially adding microcomputer hardware to an existing AFSATCOM terminal for achieving growth capability has been demonstrated. With further development and suitable qualification of this additional hardware, modifying existing MPU software may be unnecessary.

The microcomputer system as presently configured is capable of operator interface via the RS-232 serial I/O port of its internal SBC 80/10 computer. Using this port, a Texas Instruments Model 765 Intelligent Terminal and a Lear Siegler ADM-3A cathode ray tube

(CRT) terminal were successfully interfaced, both locally and over telephone landlines, using a Bell 103J modem. Thus, a removable I/O capability has been provided with commercial off-the-shelf hardware. The use of such a remoting capability makes future demonstration testing of polling improvements possible from any location having access to a telephone landline circuit and avoids the need to physically relocate the breadboard microcomputer from its interface at MITRE-Bedford to its collocated AFSATCOM Type 12 CP terminal.

Specific follow-on demonstration and evaluation objectives are derived from key issues related to incorporating polling improvements into the AFSATCOM System. Some of these objectives are:

- Evaluate the operational suitability of improved polling capabilities.
- Evaluate the enhancements resulting from improved polling in relation to performance of the AFSATCOM mission.
- Test and evaluate the effectiveness of the improved polling capabilities, including the operator-terminal interface, in passing message traffic between terminals netted in various polling modes.
- Test and evaluate the adequacy of the microcomputer hardware/software to support the overall objectives.
- Demonstrate the potential role of improved polling capabilities to all concerned agencies, using hands-on hardware/software.
- Develop a successful test-bed for simulating future AFSATCOM hardware/software polling improvements and for investigating other related areas.
- Obtain data and operating experience for an eventual upgrade of the AFSATCOM operations concept.

The improved polling test objectives cited will be fulfilled through three test categories. These are:

- Demonstration of functional interface capabilities
- Terminal performance tests
- Network tests

The first category will show the basic interface capabilities and inherent limitations of the as-designed microcomputer hardware/software in the AFSATCOM environment. The second and third categories will quantify the effectiveness and suitability of the polling improvements at the terminal and network levels and obtain an estimate of actual polled network behavior in an operational environment. This follow-on demonstration can be conducted using the microcomputer hardware/software and the AFSATCOM Type 12 CP terminal located at MITRE-Bedford. This demonstration can be performed jointly by MITRE and Air Force personnel.

LIST OF REFERENCES

1. Intel Systems Data Catalog 1980, Santa Clara, CA: Intel Corporation, August 1979.
2. SBC 80/20 and SBC 80/20-4 Single Board Computer Hardware Reference Manual, Santa Clara, CA: Intel Corporation.
3. 8008 and 8080 PL/M Programming Manual, Rev. A, Santa Clara, CA: Intel Corporation.
4. 8080 PL/M Compiler Operators Manual, Rev. A, Santa Clara, CA: Intel Corporation
5. Intel INTERP/80 User's Manual, Rev A, Santa Clara, CA: Intel Corporation.

APPENDIX A

BASELINE DESCRIPTION OF THE AFSATCOM ROLL CALL POLLING MODE

All Air Force Satellite Communications System (AFSCS) terminals can provide two-way teletypewriter (TTY) record communications using frequency shift keying (FSK) modulation at a 75 bits per second (b/s) serial rate over standard ultra high frequency (UHF) channels. The Roll Call Polling mode is one of four modes possible on the narrow-band FSK channels; the other modes are Random, TDM-1, and TDM-2.

A.1 ROLL CALL POLLING SYSTEM

Polling may be conducted on either half-duplex or full-duplex circuits and requires both a Net Control Station (NCS) and network discipline. The NCS transmits Poll Call messages to each pollable automatic send/receive (ASR) unit in the network. Each ASR, when polled, is given a 30-second period of time in which to transmit a precomposed response to the NCS. When individual ASRs are in the poll mode, they are inhibited from transmitting unless they are polled.

Roll Call Polling is initiated by a command post (CP) terminal. Force terminals reply through use of a stored message buffer in the ASR containing the previously prepared response. Poll inquiry is then possible when the polling function is selected at the ASR device.

When the CP message-processor generated poll inquiry is detected at the polled terminal through use of a unique code sequence peculiar only to that particular terminal or ASR, the precomposed message is transmitted back to the CP terminal. Roll Call Polling is structured so that the CP NCS queries each terminal in the polling net in sequence and each polled terminal replies only upon detection of its own unique code.

The Roll Call message from the CP NCS is unclassified and structured as shown in table A-1 and table A-2.

Table A-1

Roll Call Message Format

UUU, SOH, N, A₁, A₂, A₃, ETX

Address Characters*

where N = variable mode character**

Address Header

* A₃ designates the particular super group.

A₂ and A₃ designate the particular group.

A₁ and A₂ and A₃ designate the particular member.

** The mode character found in the message address header is signified by the variable N representing address header codes.

Selective Inquiry message structure is closely related to Poll Call message structure and differs only by the change to the variable found in the mode character.

The address portion of Selective Inquiry messages is similar to the Poll Call message and consists of an SOH character denoting start-of-header and signifying that the next four characters are an address header. The first character following the SOH is the mode character whose codes are represented in table A-2. It should be noted that only when the mode character is 0 does it denote that the message is a Poll Call.

Table A-2

Address Header Codes

<u>N = Mode</u>	<u>Function</u>	
0	Poll Call	
1	Super Group Call	} Selective Inquiry Modes
2	Group Call	
3	Individual Call	

An All Call mode, denoted by no address header, is accomplished by omission of the SOH designator, the mode character, and the three address designation characters.

The three address designator characters (second, third, and fourth characters following the SOH) use three hexadecimal digits (0-9, A-F) for A_1 , A_2 , and A_3 and result in 4096 possible addresses.

Each pollable device in the network must have a unique programmed address. This address is manually inserted into the ASR via thumbwheels and must be part of the Poll Call message. If there is a match, the precomposed message will be transmitted as a Poll Response message with an unclassified message structure, as follows:

UUU, $\overline{\text{SOH}}$, F, A_1 , A_2 , A_3 , message (if any), $\overline{\text{ETX}}$,

where F is the mode character indicating that the message is a Poll Response.

If there is no match, the Poll Response message is not transmitted.

A.2 CP TERMINAL POLLING

The message processor unit (MPU) within the CP terminal can accommodate four poll groups, with each group containing up to 16 members for a total of 64 members. Members of any one group are restricted to the same combination of second and third (A_2 and A_3) address characters. Two polling commands are defined. One command causes all members of a single group to be polled. The other command causes all members of all groups currently defined to be polled.

Upon reception of a polling communications supervisory (COMSUP) command from an ASR, the MPU transmits a poll request message sequentially to each member of the group, waiting 30 seconds between requests for replies. A poll response is timed out if not received in its entirety within 30 seconds. The MPU will automatically change the individual address character (A_1) designating a member of a group with each successive poll request message according to the stored poll table in the MPU. This poll table can be either entered or changed by COMSUP commands, which provide for the dynamic definition and maintenance of up to four concurrent poll groups. New groups can be added, existing groups deleted, or individual members added or deleted from existing groups. See table A-3 for a listing of COMSUP commands applicable to polling.

Table A-3

COMSUP Commands Used in Polling Operations

COMSUP Command	Description
1) $\overline{\text{PATG}}$ gg $\left[\begin{array}{c c c} 11 & a & .. \end{array} \right] \overline{c}$	ADD TO/CREATE POLL GROUP
l	gg = { 2-character group identifier where all alphabetic characters are upper case. Characters for "gg" must be selected from the hexadecimal group Ø through 9, A through F.
11 =	{ Optional 2-character line mnemonic to poll on; the line mnemonic is specified only when defining a new group.
a =	{ Aircraft identifier (one to eight characters may be specified in one command). All alphabetic characters are upper case. Characters "a" must be selected from the hexadecimal group Ø through 9, A through F.
2) $\overline{\text{PDEL}}$ gg $\left[\begin{array}{c c c} a & .. \end{array} \right] \overline{c}$	DELETE FROM POLL GROUP
gg =	{ 2-character group identifier where all alphabetic characters are upper case.
a =	{ Aircraft identifier (one to eight characters may be specified). If none is specified, the entire group will be deleted. All alphabetic characters are upper case. Characters for "a" must be selected from the hexadecimal group Ø through 9, A through 9.
3) $\overline{\text{PPLG}}$ gg $\overline{c}$	POLL GROUP
gg =	{ 2-character group identifier where all alphabetic characters are upper case.
4) $\overline{\text{PPAG}}$ $\overline{c}$	POLL ALL GROUPS

Table A-3 (Continued)

Notes

The group tables are built and maintained dynamically in response to operator commands. A new group will be defined as a result of an ATG command that calls out a group not already in the tables. Members in a group are identified in the ATG and DEL commands by three characters; the last two characters of a group member identifier are the same as the group identifier. When all group tables are in use, the form of the command used for defining up to four groups at any given time will be rejected by the MPU. Another form of the command is available for use in subsequently adding additional members to an existing group. Attempts to add a member already in the group will cause the entire command to be rejected.

For the command used to delete individual members from a polling group, the entire command is processed and all identifiers are validated by the MPU before the group table entry is modified. Any error detected will cause the entire group to be rejected. Another form of this deletion command is available for use in deleting an entire group and releasing the associated table entry. It should be noted that the first form of this command can also result in the group table entry being released if all its members are released.

Each member is polled by building a Poll Call message containing the group and member identifiers, then queuing the message to the MPU output line associated with the group.

When the Poll Response message is received, it is verified and forwarded to a high speed printer with the prefix POLL RESPONSE RX. The next group is then polled and the process continues until the entire group has been polled. If no response is received from a particular member within 30 seconds, a print-out alarm is generated to identify the group member who failed to respond. A received response from a terminal not polled (e.g., address of responding terminal does not match that in the poll request message) results in the message being intercepted, forwarded for printout on the high speed printer, and a POLL RESPONSE ER interrupt message being printed out along with the Poll Response. See table A-4 for a description of the MPU Status/Alarm and Intercept Messages for polling.

Table A-4

MPU Status/Alarm and Intercept Messages

<u>Condition</u>	<u>Status/Alarm Message</u>
1) GROUP XX POLL COMPLETE XX 2 hexadecimal digits, a group identifier.	PLG XX COM
2) ALL CROUP POLL COMPLETE Processor has completed polling all previously defined groups.	PAG COM
3) NO RESPONSE TO POLL A time-out has occurred on a particular member.	POLL RSP T/O
<u>Reason for Intercept</u>	<u>Intercept Message</u>
4) Poll Response from station not polled	POLL RESPONSE EP
5) Poll Response received	POLL RESPONSE RX

Note:

MPU-generated Status/Alarm messages are prefixed with a date-time header and a line identification mnemonic which indicates the source of the message.

A.3 FORCE TERMINAL POLLING

As a member of a poll, the Force terminal pollee must dial his proper address into the ASR thumbwheels, enable the address recognition switch on the ASR, and press the poll XMT button once. Whenever the ASR receives a message header of the format following the UUU consisting of:

SOH, $\emptyset$, A_1 , A_2 , A_3

where

A_1 , A_2 , A_3 is the ASR address,

the ASR will automatically transmit whatever is in ASR storage up to the message length selected, each time the ASR is polled. If a change to the Poll Response message is desired, the operator must press the compose and edit button and make the changes, thereby taking the particular ASR out of the automatic poll response mode. Thus, poll responses are inhibited while the Poll Response messages are either updated or revised. After update, the poll XMT button must be depressed again to return to the poll. Both Poll Call and Poll Response messages should print out on the pollee's ASR. In addition, any text attached to the Poll Call message will not be printed out by the pollee's ASR, regardless of model ASR (i.e., 120A; 120B; 129; Mini-I/O). In actual operation during an automatic poll request, the model 120A ASR prints nothing; models 120B, 129, and the Mini-I/O print only the poll message (five characters described above) along with the ETX (whenever it occurs).

A.4 OTHER MPU-EQUIPPED TERMINAL POLLING

An MPU-equipped terminal can also be polled as described above but is limited to a shorter Poll Response message because of the message transfer technique inherent in the MPU-equipped terminal. Since data are transferred from the modem to ASR on a message-by-message basis, the complete Poll Response message is transferred into the processor from the ASR before going to the modem. To stay within the CP processor poll-out time of 30 seconds, the maximum length of a Poll Response message from a processor equipped terminal is 130 characters.

APPENDIX B PL/M-80 SOURCE CODE

```

/* III.DATA - MCD 2: VERSION 1 - 10 JULY 80 */
/* RHH.DATA - MCD 1: VERSION 3 - 10 JAN. 80 */
/*
/* IMPROVED POLLING COMSUP COMMANDS */
/*
/* ENABLE CHECKIN MODE ZNR!SABCIXXX */
/* DISABLE CHECKIN MODE ZNR!SABCOXXX */
/* PCI TABLE CHECKIN ZNR!CABCAAA1 */
/* NCI TABLE CHECKIN ZNR!DABCBBB1 */
/* GCI TABLE CHECKIN ZNR!EABCCCC0 */
/* PRIORITY TABLE LOAD ZNR!BABCDD1 */
/* ROUTINE TABLE LOAD ZNR!IABC.... */
/* NO-TRAFFIC TABLE LOAD ZNR!ZABC.... */
/* DISABLE COMSUP MSG PRINTOUT ZNR!JABC.... */
/* ENABLE COMSUP MSG PRINTOUT ZNR!WABCO */
/* RE-INITIALIZE PCI TABLE ZNR!HABC1 */
/* RE-INITIALIZE NCI TABLE ZNR!FABC */
/* RE-INITIALIZE GCI TABLE ZNR!GABC */
/* RE-INITIALIZE PRIORITY TABLE ZNR!HABC */
/* RE-INITIALIZE ROUTINE TABLE ZNR!AAFC */
/* PRIORITY CHECKIN TABLE TRANSFER ZNR!TABCXYZ */
/* ROUTINE CHECKIN TABLE TRANSFER ZNR!TABCYXYZ */
/* NO-TRAFFIC CHECKIN TABLE TRANSFER ZNR!TABCZXYZ */
/* POLL TABLE TRANSFER ZNR!TABCXYZ */
/* BUILD & PRINTOUT POLL TABLE ZNR!HABC */
/* POLL TABLE PRINTOUT (LOCAL ONLY) ZNR!UABCZXYZ1 */
/* POLL TABLE PRINTOUT (REMOTE ALL-CALL) ZNR!UABCZXYZ2 */
/* PRIORITY TABLE PRINTOUT ZNR!UABCXYZ */
/* ROUTINE TABLE PRINTOUT ZNR!UABCXYZ */
/* NO-TRAFFIC TABLE PRINTOUT ZNR!UABCXYZ */
/* GROUP TABLE PRINTOUT ZNR!UABCXYZ */
/* POLL ENABLE (SINGLE POLL) ZNR!JABC15 */
/* POLL ENABLE (MULTIPLE POLL) ZNR!JABC1M */
/* POLL DISABLE (SINGLE) ZNR!JABC05 */
/* POLL INTERRUPT ENABLE (NON-TDM) ZNR!LABC10 */
/* POLL INTERRUPT ENABLE (TDM) ZNR!LABC01 */
/* POLL INTERRUPT ENABLE (TDM & NON-TDM) ZNR!LABC11 */
/* POLL INTERRUPT DISABLE (TDM & NON-TDM) ZNR!LABC00 */
/* PRIORITY CHECKIN TABLE DELETION ZNR!NABCP01 */
/* ROUTINE CHECKIN TABLE DELETION ZNR!NABCR01 */
/* NO-TRAFFIC CHECKIN TABLE DELETION ZNR!NABCN01 */
/* GROUP CHECKIN TABLE DELETION ZNR!NABCG01 */
/* FEEDBACK & POLL SLOT REASSIGNMENT ZNR!OABC..... */
/* EDAC ENABLE (NON-TDM) ZNR!PABC1 */
/* EDAC DISABLE (NON-TDM) ZNR!PABC0 */
/* MESSAGE TRANSMISSION DELAY (NON-TDM) ZNR!KABC10 */
/* ENABLE ALL TYPES OF MESSAGES ZNR!VABC0 */
/* DISREGARD NON-COMSUP MSGS WHEN POLLING ZNR!VABC1 */
/* DISREGARD ALL NON-COMSUP MESSAGES ZNR!VABC2 */
/* PROGRAM RESTART (RE-IFL MICROPROCESSOR) ZNR!RABC */
/* RS-232 INTERFACE ENABLE(1)/DISABLE(0) ZNR!QABC1 */
/* MICROPROCESSOR ADDRESS CHANGE ZNR!IABCCAB */
/*
/*
/* NOTES */
/*
/* 1) ALL COMSUP MSGS REQUIRE "ZNR!" FOLLOWED BY AT LEAST ONE MORE CHARACTER IMMEDIATELY AFTER THE "I".
/* 2) TWO ETX'S ARE PART OF THE COMSUP MSG SUFFIX.
/* 3) "XYZ" IS INDICATIVE OF A HEXIDECIMAL DESTINATION ADDRESS.
/* 4) "ABC" IS INDICATIVE OF THE INITIALIZED ADDRESS OF THE MICROPROCESSOR; THIS IS A COMSUP VARIABLE IN ITSELF.
/*
/*
/* INPUT- / OUTPUT DESCRIPTIONS */

```

```

/*      INPUT 1 = CHAR READY FROM MODEN      */
/*      INPUT 101 = CHAR READY FROM MPU      */
/*      INPUT 2 = CONTINUE PROGRAM           */
/*      INPUT 3 = CHAR FROM MODEN           */
/*      INPUT 103 = CHAR FROM MPU           */
/*      INPUT 5 = SPACE FOR OUTPUT TO MPU    */
/*      INPUT 105 = SPACE FOR OUTPUT TO MODEN */
/*      INPUT <7>: 230 (E6H) = CLOCK FOR MICROPROCESSOR */
/*      INPUT <107>: 229 (E5H) = CLOCK FROM MODEN */
/*      INPUT <9>: 228 (E4H) = ECD SLOT INPUTS */
/*      */
/***/
/*      6253 PROGRAMMABLE COUNTER INPUT/OUTPUT */
/*      OUTPUT 220 (DCM) = LOAD/READ COUNTER NO. 0 (8253) */
/*      OUTPUT 222 (DEM) = LOAD/READ COUNTER NO. 2 (8253) */
/*      OUTPUT 223 (DFH) = WRITE MODE WORD/NO-OP (3-STATE) */
/*      */
/*      OUTPUT 1 = DATA TO MPU */
/*      OUTPUT 101 = DATA TO MODEN */
/*      OUTPUT 2 = DIAGNOSTICS */
/*      OUTPUT <4>: 233 (E9H) = PARITY DISABLE TO MODEN AND I/O CHIP RESET */
/*      OUTPUT <6>: 234 (EAH) = RX ENABLE TO MPU */
/*      OUTPUT <106>: 235 (E6H) = I/O TX ENABLE TO MODEN AND RX CLOCK ENABLE TO MPU */
/*      */
/***/
/*      RS232 INPUT/OUTPUT */
/*      OUTPUT 237 (EDH) = CONTROL WORD (CPU TO USART) */
/*      OUTPUT 236 (ECH) = DATA (CPU TO USART) */
/*      INPUT 237 (EDH) = STATUS WORD (USART TO CPU) */
/*      INPUT 236 (ECH) = DATA (USART TO CPU) */
/*      */
/***/
/***/
/*      MEMORY ADDRESSES FOR I/O */
/*      */
/*      INPUT11ADR = 0FF10H */
/*      INPUT13ADR = 0FF40H */
/*      OUTPUT11ADR = 0FF40H */
/*      INPUT101ADR = 0FF10H */
/*      INPUT103ADR = 0FF40H */
/*      INPUT105ADR = 0FF00H */
/*      OUTPUT101ADR = 0FF40H */
/*      INPUT13ADR = 0FF00H */
/*      */
/***/
/***/
/***/
/***/
/*      LECLARE (PROGRAMSSTART, INITIALMSG) LABEL; */
/*      RS232 I/O CHIP INITIALIZATION */
/*      */
/*      MODE CONTROL */
/*      PROGRAMSSTART: OUTPUT(237) = 79h; */
/*      79h = 0111 1001b; 01=1 STOP BIT/1=EVEN PARITY/1=PARITY ENABLE */
/*      10=7 BIT CHARACTER/01=1X BAUD RATE FACTOR */
/*      */
/*      COMMAND INSTRUCTION - RESET */
/*      OUTPUT(237) = 40h; */
/*      MODE INSTRUCTION */
/*      OUTPUT(237) = 79h; */
/*      COMMAND INSTRUCTION - INITIALIZATION */
/*      OUTPUT(237) = 17h; */
/*      17h = 0001 0111b; 0=ENTER HUNT MODE/0=INTERNAL RESET/ */
/*      0=RTS/1=ERROR RESET/0=SEND BREAK CHAR/1=RX ENABLE */
/*      1=DTR (NOT) OUTPUT FORCED TO ZERO/1=TX ENABLE */

```

```

/**/
/* CONTROL WORD INITIALIZATION (8255 # 1 CONTROL ONLY) */
    OUTPUT(231) = 9BH;
    OUTPUT(235) = 90H;
    OUTPUT(233) = 10H;
    OUTPUT(233) = 0H;
    OUTPUT(233) = 10H;

/**/
/* CONTROL REGISTER INITIALIZATION */
DECLARE MPU$IO$CTL1$POINTER ADDRESS INITIAL (0FF00H);
DECLARE MPU$IO$CTL1 BASED MPU$IO$CTL1$POINTER BYTE;
    MPU$IO$CTL1$POINTER = 0FF00H;
    MPU$IO$CTL1 = 03H XOR 0FFH;

/**/
DECLARE MPU$IO$CTL2$POINTER ADDRESS INITIAL (0FF40H);
DECLARE MPU$IO$CTL2 BASED MPU$IO$CTL2$POINTER BYTE;
    MPU$IO$CTL2$POINTER = 0FF40H;
    MPU$IO$CTL2 = 2FH XOR 0FFH;

/**/
    MPU$IO$CTL1 = 43H XOR 0FFH;
    MPU$IO$CTL2 = 03H XOR 0FFH;
    MPU$IO$CTL1 = 0C0H XOR 0FFH;

/**/
DECLARE MODEM$IO$CTL1$POINTER ADDRESS INITIAL (0FE00H);
DECLARE MODEM$IO$CTL1 BASED MODEM$IO$CTL1$POINTER BYTE;
    MODEM$IO$CTL1$POINTER = 0FE00H;
    MODEM$IO$CTL1 = 03H XOR 0FFH;

/**/
DECLARE MODEM$IO$CTL2$POINTER ADDRESS INITIAL (0FE40H);
DECLARE MODEM$IO$CTL2 BASED MODEM$IO$CTL2$POINTER BYTE;
    MODEM$IO$CTL2$POINTER = 0FE40H;
    MODEM$IO$CTL2 = 2FH XOR 0FFH;

/**/
    MODEM$IO$CTL1 = 43H XOR 0FFH;
    MODEM$IO$CTL2 = 03H XOR 0FFH;
    MODEM$IO$CTL1 = 0C0H XOR 0FFH;

/**/
/* INSTRUCTION FOR DIVIDE BY 1240 (300 HZ) ON 16-BIT COUNTER # 2 */
    OUTPUT(223) = 0B6H;
    OUTPUT(222) = 0D8H;
    OUTPUT(222) = 04H;

/* INSTRUCTION FOR DIVIDE BY 4960 (75 HZ) ON COUNTER # 0 */
    OUTPUT(223) = 36H;
    OUTPUT(220) = 62H;
    OUTPUT(220) = 13H;

/* THESE GIVE 75 & 300 HZ CLOCKS FROM 6.70 MHZ ON-BOARD XTAL */

/**/
DECLARE RS232$FLAG BYTE INITIAL (0);
    RS232$FLAG = 0;
DECLARE (RS232$SPACE$TEST, RS232$COL$CHAN, RS232$INPUT1,
    RS232$GC1, RS232$MODE$SET) LABEL;
DECLARE (ALH1, ADR2, ADR3) BYTE INITIAL ('A', 'B', 'C');
    ALH1 = 'A';
    ADR2 = 'B';
    ADR3 = 'C';
DECLARE (POLL$SLOT1H, POLL$SLOT2H, POLL$SLOT3H, POLL$SLOT4H,
    POLL$SLOT5H, POLL$SLOT6H, POLL$SLOT1L, POLL$SLOT2L, POLL$SLOT3L,
    POLL$SLOT4L, POLL$SLOT5L, POLL$SLOT6L, TX$SLOT$INITIALH,
    TX$SLOT$INITIALL, TDM$POLL$TX$SLOT$INITIAL) BYTE
    INITIAL ('1', '1', '2', '3', '4', '5', '1', '9', '7', '5', '3', '1', '0', '1', 1);

```

```

00122000
00123000
00124000
00125000
00126000
00127000
00128000
00129000
00130000
00131000
00132000
00133000
00134000
00135000
00136000
00137000
00138000
00139000
00140000
00141000
00142000
00143000
00144000
00145000
00146000
00147000
00148000
00149000
00150000
00151000
00152000
00153000
00154000
00155000
00156000
00157000
00158000
00159000
00160000
00161000
00162000
00163000
00164000
00164010
00164020
00164030
00164040
00165000
00166000
00167000
00168000
00169000
00170000
00171000
00172000
00173000
00174000
00175000
00176000
00177000
00178000
00179000
00180000

```

```

POLL$SLOT1H = '1'; 00181000
POLL$SLOT1L = '1'; 00182000
POLL$SLOT2H = '1'; 00183000
POLL$SLOT2L = '9'; 00184000
POLL$SLOT3H = '2'; 00185000
POLL$SLOT3L = '7'; 00186000
POLL$SLOT4H = '3'; 00187000
POLL$SLOT4L = '5'; 00188000
POLL$SLOT5H = '4'; 00189000
POLL$SLOT5L = '3'; 00190000
POLL$SLOT6H = '5'; 00191000
POLL$SLOT6L = '1'; 00192000
TX$SLOTS$INITIALH = '0'; 00193000
TX$SLOTS$INITIALL = '1'; 00194000
IDM$POLL$TX$SLOTS$INITIAL = 4; 00195000
DECLARE (CS3, CS4) BYTE; 00196000
DECLARE (PC1L$XMIT, POLL$MODE$ERROR, POLL$OVER$TEST, XX, YY, WW, ZZ, 00197000
    REPOLL$TEST, POLL$MSG$COMPLETE, OUTTABLE1$STATUS$TEST, 00198000
    POLLING$MODE$TEST, AUTO$POLL$START, NON$TDM$POLL, EDAC$MODE$SET, 00199000
    TDM$POLL, PC1L$TABLE$TX, POLL$INTERRUPT, ALTRAN$MIT$FLAG$SET) LABEL; 00200000
DECLARE POLL$MSG$COUNT ADDRESS INITIAL(0); 00201000
    POLL$MSG$COUNT = 0; 00202000
DECLARE ATL$EDAC$TABLE(7) BYTE; 00203000
DECLARE (ECAC$FLAG, REPOLL$FLAG$COUNT) BYTE INITIAL(0,0); 00204000
    EDAC$FLAG = 0; 00205000
    REPOLL$FLAG$COUNT = 0; 00206000
DECLARE (CC$PRESENT, CC1$PRESENT) ADDRESS INITIAL(0,0); 00207000
    CC$PRESENT = 0; 00208000
    CC1$PRESENT = 0; 00209000
DECLARE (IPOLL101, GC1, OPOLL101, IPOLL1011, NON$TDM$DELAY1, POLL$MODE1, 00210000
    DELAY$ROUTINE1, EAT$MSG$SET, MCD$M$OUTPUT$COMPOSE1, 00211000
    LCT1, OUTFLAG1, ECHTEST1, SPACETEST1, NON$TDM$DELAY$SET, CHAR$T1, 00212000
    CON$UP$LOAD1, OUTCH$A1, NOCON$UP$LOAD1, RX$BUSY$OVERRIDE1, IPOLL11) LABEL; 00213000
DECLARE (SHIFTDOWN, SHIFTDOWN1, LOADADR, ENDLOAD, ADR$ERROR1) LABEL; 00214000
DECLARE EAT$MSG$FLAG1 BYTE INITIAL(01H); 00215000
    EAT$MSG$FLAG1 = 01H; 00216000
DECLARE (CLK1, CLK$1, DELAY1, CLK2, CLK$2, CLK3, 00217000
    CLK$3) BYTE INITIAL(0,0,0,0,0,0,0); 00218000
DECLARE (CLKCNT1, CLKCNT2, CLKCNT3) ADDRESS INITIAL(0,0,0); 00219000
    CLK1 = 0; 00220000
    CLK$1 = 0; 00221000
    CLKCNT1 = 0; 00222000
    DELAY1 = 0; 00223000
    CLK2 = 0; 00224000
    CLK$2 = 0; 00225000
    CLKCNT2 = 0; 00226000
    CLK3 = 0; 00227000
    CLK$3 = 0; 00228000
    CLKCNT3 = 0; 00229000
DECLARE (BUFFER$DELAY1, NON$TDM$MSG$DELAY1) ADDRESS INITIAL(48,300H); 00230000
    BUFFER$DELAY1 = 48; 00231000
    NON$TDM$MSG$DELAY1 = 300H; 00232000
DECLARE (ELAC$POUTLINE, OUTPUT$COMPOSE1, PTT$LOAD1, DELAY$CAL, 00233000
    NIT$LOCAL1, NIT$LOAD1, TIT$CAL$END1) LABEL; 00234000
DECLARE (TCLK, TCLK$1, TCLKCNT) BYTE INITIAL(0,0,0); 00235000
    TCLK = 0; 00236000
    TCLK$1 = 0; 00237000
    TCLKCNT = 0; 00238000
DECLARE (MSG$COVER$DELAY, TCLK1, TCLK$1, TCLKCNT1) BYTE 00239000
    INITIAL(64, 0, 0, 0); 00240000
    MSG$COVER$DELAY = 60H; 00241000
    TCLK1 = 0; 00242000
    TCLK$1 = 0; 00243000

```

```

        TCLKCNT1 = 0;
DECLARE (RETRANSMITSFLAG, TDMSPOLLSTXSSLOT) BYTE INITIAL(0,1);
        RETRANSMITSFLAG = 0;
        TDMSPOLLSTXSSLOT = 1;
DECLARE (SLOTH,SLOTL,SLOT,SLOTSCOUNT) BYTE INITIAL (0,0,0,0);
        SLOTH = 0;
        SLOTL = 0;
        SLOT = 0;
        SLOTSCOUNT = 0;
DECLARE (N1LMDLAYH, N1LMDLAYL) ADDRESS INITIAL(0,0);
        N1LMDLAYH = 0;
        N1LMDLAYL = 0;
DECLARE SLOTSGUARD$TIME ADDRESS INITIAL(20H);
        SLOTSGUARD$TIME = 20H;
DECLARE (TDM$FLAG, POLL$FLAG, AUTOSREPOLL$FLAG) BYTE INITIAL('0', 0, 0);
        TDM$FLAG = '0';
        POLL$FLAG = 0;
        AUTOSREPOLL$FLAG = 0;
DECLARE (POLL$TABLE$TX$FLAG, N1LMD$POLL$INTERRUPT$FLAG,
        TDM$POLL$INTERRUPT$FLAG) BYTE INITIAL(0,0,0);
        POLL$TABLE$TX$FLAG = 0;
        N1LMD$POLL$INTERRUPT$FLAG = 0;
        TDM$POLL$INTERRUPT$FLAG = 0;
DECLARE (CS1, CHAN1, OUT1,
        WAITING1, SPACE1, LINECOUNT) BYTE;
        OUT1 = 0;
        WAITING1 = 0;
DECLARE CHAR$TABLE1(2000) BYTE;
DECLARE MSGCOUNT1(150) ADDRESS;
DECLARE OUTTABLE1 (2000) BYTE;
DECLARE (I1, TC1, OUTCOUNT1,CC1,J) ADDRESS INITIAL (0,0,1,0,0);
        J = 0;
        I1 = 0;
        TC1 = 0;
        OUTCOUNT1 = 1;
        CC1 = 0;
DECLARE CS1$DATA$POINTER ADDRESS INITIAL(0FF00H);
DECLARE CS1$DATA BASED CS1$DATA$POINTER BYTE;
        CS1$DATA$POINTER = 0FF00H;
DECLARE INPUT3$DATA$POINTER ADDRESS INITIAL (0FF40H);
DECLARE INPUT3$DATA BASED INPUT3$DATA$POINTER BYTE;
        INPUT3$DATA$POINTER = 0FF40H;
DECLARE INPUT5$DATA$POINTER ADDRESS INITIAL (0FE00H);
DECLARE INPUT5$DATA BASED INPUT5$DATA$POINTER BYTE;
        INPUT5$DATA$POINTER = 0FE00H;
DECLARE OUTPUT1$DATA$POINTER ADDRESS INITIAL (0FE40H);
DECLARE OUTPUT1$DATA BASED OUTPUT1$DATA$POINTER BYTE;
        OUTPUT1$DATA$POINTER = 0FE40H;
DECLARE CS1$DATA$POINTER ADDRESS INITIAL (0FE00H);
DECLARE CS1$DATA BASED CS1$DATA$POINTER BYTE;
        CS1$DATA$POINTER = 0FE00H;
DECLARE INPUT103$DATA$POINTER ADDRESS INITIAL (0FE40H);
DECLARE INPUT103$DATA BASED INPUT103$DATA$POINTER BYTE;
        INPUT103$DATA$POINTER = 0FE40H;
DECLARE INPUT105$DATA$POINTER ADDRESS INITIAL (0FF00H);
DECLARE INPUT105$DATA BASED INPUT105$DATA$POINTER BYTE;
        INPUT105$DATA$POINTER = 0FF00H;
DECLARE OUTPUT101$DATA$POINTER ADDRESS INITIAL (0FF40H);
DECLARE OUTPUT101$DATA BASED OUTPUT101$DATA$POINTER BYTE;
        OUTPUT101$DATA$POINTER = 0FF40H;
/**/
DECLARE (CS,CHAN,ENC,MJF,MODE,

```

```

00244000
00245000
00246000
00247000
00248000
00249000
00250000
00251000
00252000
00253000
00254000
00255000
00256000
00257000
00258000
00259000
00260000
00261000
00262000
00263000
00264000
00265000
00266000
00267000
00268000
00269000
00270000
00271000
00272000
00273000
00274000
00275000
00276000
00277000
00278000
00279000
00280000
00281000
00282000
00283000
00284000
00285000
00286000
00287000
00288000
00289000
00290000
00291000
00292000
00293000
00294000
00295000
00296000
00297000
00298000
00299000
00300000
00301000
00302000
00303000
00304000
00305000
00306000
00307000
00308000
00309000

```

AD-A102 294

MITRE CORP BEDFORD MA
MICROCOMPUTER POLLING IMPROVEMENTS FOR AFSATCOM.(U)
JUN 81 J HANDWERKER

F/G 17/2.1

UNCLASSIFIED

NTR-8239

ESD-TR-81-118

F19628-81-C-0001
NL

2 of 2
AL
9-102234



END
DATE
FILMED
8-81
DTIC

WAITING, SPACE) BYTE;	00310000
ENC = 0;	00311000
MFJ = 0;	00312000
WAITING = 0;	00313000
DECLARE (N, NUMSGCI, TOTALNUMSCI,	00314000
NUMSPCI, PCISPOINTER, NC, NUMSRCI, NCISPOINTER, GCISPOINTER,	00315000
NUMSACI, NCISPOINTER, CLKCNT, TCINITIAL, TCINITIAL) ADDRESS;	00316000
TOTALNUMSCI = 0;	00317000
NUMSGCI = 0;	00318000
NUMSPCI = 0;	00319000
NUMSRCI = 0;	00320000
NUMSACI = 0;	00321000
CLKCNT = 0;	00322000
TCINITIAL = 0;	00323000
TCINITIAL = 0;	00324000
DECLARE (CLK, CLKF, DELAY) BYTE INITIAL(0,0,0);	00325000
CLK = 0;	00326000
CLKF = 0;	00327000
DELAY = 0;	00328000
DECLARE (BUFFERSDELAY, MSGSDelay) ADDRESS INITIAL(48, 640);	00329000
BUFFERSDELAY = 48;	00330000
MSGSDelay = 160;	00331000
DECLARE (IIO111, GC, OPOLL1, QUIT, ST, MJT1, LCT, CHAR3T,	00332000
CHAR4T, CHAR5T, CSSTEST, PRBSLOAD, MRBSLOAD, NMRBSLOAD, MJT2,	00333000
FITSLOAD\$END, RTTSLOAD\$END, NITSLOAD\$END,	00334000
PTISLOAD, RTTSLOAD, NITSLOAD, OUTSTABLE\$BUILD, C\$MODE\$CANCEL1,	00335000
PCISTEST, CCSTEST, ACISTEST, NCISTEST, C\$NUM\$TEST, C\$MODE\$CANCEL,	00336000
LCFSTEST, SPACES\$TEST, DELAY\$RCUTIME, OUT\$CHAR) LABEL;	00337000
DECLARE (PCIRESET, RCIRESET, NCIRESET, GCIRESET, ADRCHANGE) LABEL;	00338000
DECLARE (STAR\$INSERT, MPUSTABLE\$BUILD) LABEL;	00339000
DECLARE (FCISTORE, NCISTORE, NCISTORE) (365) BYTE;	00340000
DECLARE GCISTORE(321) BYTE;	00341000
DECLARE CHAR\$STORE(2000) BYTE;	00342000
DECLARE MSG\$COUNT(150) ADDRESS;	00343000
DO I1 = 0 TO 149;	00344000
MSG\$COUNT(I1) = 0;	00345000
END;	00346000
DECLARE (ENCRYPT, ENCTEST) LABEL;	00347000
DECLARE SOH LITERALLY '01H' ;	00348000
DECLARE C\$MODE BYTE INITIAL (0H) ;	00349000
C\$MODE = 0H;	00350000
DECLARE PCISHEADER DATA ('*PRIORITY*CI:') ;	00351000
DECLARE (I, IC, OUT\$COUNT, CC) ADDRESS INITIAL (0,0,1,0);	00352000
I = 0;	00353000
IC = 0;	00354000
OUT\$COUNT = 1;	00355000
CC = 0;	00356000
DECLARE NCISHEADER DATA ('NCI:') ;	00357000
DECLARE OUT\$TABLE(2000) BYTE;	00358000
DECLARE NCISHEADER DATA ('NCI:') ;	00359000
DECLARE MRBSHEADER DATA ('*PRIORITY*RB') ;	00360000
DECLARE MRBSHEADER DATA ('RRB:') ;	00361000
DECLARE NMRBSHEADER DATA ('NMRB:') ;	00362000
DECLARE RTTSHEADER DATA ('*PRIORITY*TT') ;	00363000
DECLARE RTTSHEADER DATA ('RTT:') ;	00364000
DECLARE NRTTSHEADER DATA ('NRTT:') ;	00365000
/* */	00366000
DECLARE (COMPOSE1, FT1, RT1, NT1, GT1, POLLTABLEBUILD1, IPL\$TEST,	00367000
CHECKIN\$MOLESET1, TTR1, TBLDUMP1, CTABLEBUILD1,	00368000
MOLE\$FROM1, MPUCINUMTEST1, PTT\$1, RTTR1, NTR1,	00369000
POLLITR1, PTABLEDUMP1, NTABLEDUMP1, NTABLEDUMP1, GTABLEDUMP1,	00370000
POLLTABLEDUMP1, RTABLEBUILD1, PTABLEBUILD1, POLLTABLEDUMP2,	00371000
RTABLEBUILD1, POLLTABLEDUMP1, G\$CLL\$BUILD1, CHECKIN\$DELETE,	00372000

RPOLLBUILD1, RPOLLBUILD1, OVERFLOW1, OVERFLOW2) LABEL;	00373000
DECLARE POLLTABLE1(1473) BYTE;	00374000
DECLARE (POLLPCINTER1, PTICOUNT1) ADDRESS INITIAL (0,0);	00375000
POLLPCINTER1 = 0;	00376000
PTICOUNT1 = 0;	00377000
DECLARE ITHHEADER DATA	00378000
('PRIORITY CHECKIN TABLE DUMP',ODH,0AH);	00379000
DECLARE KTLHEADER DATA	00380000
('ROUTINE CHECKIN TABLE DUMP',ODH,0AH);	00381000
DECLARE NTHHEADER DATA	00382000
('NO TRAFFIC CHECKIN TABLE DUMP',ODH,0AH);	00383000
DECLARE GTLHEADER DATA	00384000
('GROUP CHECKIN TABLE DUMP', ODH, 0AH);	00385000
DECLARE ADKERRORHDR DATA	00386000
('** INCORRECT ADR CHARACTERS **');	00387000
DECLARE ERKORHDR DATA	00388000
('** INCORRECT MOLE CHARACTER **');	00389000
DECLARE TFRERRORHDR DATA	00390000
('TABLE TRANSFER ERROR: NO CHECKINS');	00391000
DECLARE OVERFLOWHDR DATA	00392000
('** CHECKINS AT THE LIMIT **');	00393000
DECLARE POLLDUMPHDR DATA	00394000
('POLL TABLE DUMP: CHANNEL A', ODH,0AH) ;	00395000
DECLARE POLLMSGOVER DATA	00396000
('** ALL POLL MESSAGES LOADED **');	00397000
/**/	00398000
/**/	00399000
/**/	00400000
/**/	00401000
/* INITIALIZATION MSG TO TERMINAL OPERATOR */	00402000
DECLARE STARTUP\$MSG DATA	00403000
('IMPCVEL POLLING SOFTWARE - VERSION 3: 10 JAN. 1980 ');	00404000
INITIAL\$MSG: WAITING = WAITING + 1;	00405000
DO I = 1 TO 54;	00406000
CUTTABLE(IC + I) = STARTUP\$MSG(I - 1);	00407000
END;	00408000
IC = IC + 54;	00409000
MSGCOUNT(WAITING) = 54;	00410000
/**/	00411000
/**/	00412000
/**/	00413000
/**/	00414000
/**/	00415000
/**/	00416000
/**/	00417000
/**/	00418000
/**/	00419000
/**/	00420000
/* SLOT CALCULATION ALGORITHM */	00421000
IPOLL1: SLCI = INPUT(226);	00422000
SLCI = SLCI XOR OFFH;	00423000
SLCTL = SLCI AND OFOH;	00424000
SLCTL = SHR(SLCTL,4);	00425000
SLOTH = SLCI AND OFH;	00426000
SLOTSCOUNT = SLOTH * 10 + SLOTL;	00427000
IF SLOTSCOUNT <> 0 THEN TDM\$FLAG = '1';	00428000
ELSE TDM\$FLAG = '0';	00429000
GO TO POLL\$EXIT;	00430000
/* SLOT IS INPUTTED AS 2 BCD VALUES ON A SINGLE INPUT */	00431000
/* THE 2 BCD VALUES ARE MASKED OUT INTO SLOTH AND SLOTL */	00432000
/**/	00433000
/**/	00434000
/* POLLING ALGORITHM */	00435000

```

/**/
/**/
POLL$MIT: IF POLL$FLAG <> 01H THEN GO TO IPOLL11;
          IF PTICOUNT1 < 4 THEN GO TO POLL$MODE$ERROR;
          ELSE GO TO POLL$OVER$TEST;
POLL$MODE$ERROR: POLL$FLAG = 0;
                  GO TO MODE$ERROR1;
POLL$OVER$TEST: IF POLL$MSG$COUNT >= TOTAL$NUM$SCI
                  THEN GO TO REPOLL$TEST;
                  ELSE GO TO OUTTABLE1$STATUS$TEST;
REPOLL$TEST: IF AUTOS$REPOLL$FLAG = 01H
               THEN GO TO AUTOS$POLL$RESTART;
               POLL$FLAG = 0H;
               GO TO POLL$MSG$COMPLETE;
POLL$MSG$COMPLETE: WAITING = WAITING + 1;
                  DO I = 1 TO 30;
                  OUTTABLE1(TC + I) = POLL$MSG$OVER(I-1);
                  END;
                  TC = TC + 30;
                  MSG$COUNT(WAITING) = 30;
                  POLL$MSG$COUNT = 0;
                  GO TO IPOLL11;
OUTTABLE1$STATUS$TEST: IF TC1 > 0 THEN GO TO IPOLL11;
                       ELSE GO TO POLLING$MODE$TEST;
POLLING$MODE$TEST: IF TDM$FLAG = '1' THEN GO TO TDM$POLL;
                   ELSE GO TO NON$TDM$POLL;
AUTOS$POLL$RESTART: POLL$MSG$COUNT = 0;
                    TDM$POLL$TX$SLOT = TDM$POLL$TX$SLOTS$INITIAL;
                    GO TO POLL$MSG$COMPLETE;
NON$TDM$POLL: IF NDM$POLL$INTERRUPT$FLAG = 01H
               THEN GO TO IPOLL11;
               IF RETRANSMIT$FLAG <> 01H THEN
                   POLL$MSG$COUNT = POLL$MSG$COUNT + 1;
                   ELSE RETRANSMIT$FLAG = 0;
/*      NON-TDM POLL MSG COMPOSITION      */
WAITING1 = WAITING1 + 1;
OUTTABLE1(TC1 + 1) = 01H;
OUTTABLE1(TC1 + 2) = '0';
DO I1 = 1 TO 3;
    OUTTABLE1(TC1 + 2 + I1) =
        OUTTABLE1(I1 + POLL$MSG$COUNT + 4 - 4);
END;
OUTTABLE1(TC1 + 6) = 03H;
OUTTABLE1(TC1 + 7) = 03H;
/**/
/*      ELAC TABLE LOAD      */
DO J = 1 TO 7;
    NDM$ELAC$TABLE(J) = OUTTABLE1(TC1 + J);
END;
/**/
TC1 = TC1 + 7;
MSG$COUNT(WAITING1) = 7;
GO TO IPOLL11;
TDM$POLL: IF TDM$POLL$INTERRUPT$FLAG = 01H
           THEN GO TO IPOLL11;
           IF SLOTS$COUNT = TDM$POLL$TX$SLOT THEN
               TDM$POLL$TX$SLOT = TDM$POLL$TX$SLOT + 1;
           ELSE GO TO IPOLL11;
           IF TDM$POLL$TX$SLOT >= TDM$POLL$TX$SLOTS$INITIAL + 8 THEN
               TDM$POLL$TX$SLOT = TDM$POLL$TX$SLOTS$INITIAL;
/*      TDM POLL MSG COMPOSITION      */
IF TOTAL$NUM$SCI - POLL$MSG$COUNT >= 6 THEN N = 6;
ELSE N = TOTAL$NUM$SCI - POLL$MSG$COUNT;

```

```

00436000
00437000
00438000
00439000
00440000
00441000
00442000
00443000
00444000
00445000
00446000
00447000
00448000
00449000
00450000
00451000
00452000
00453000
00454000
00455000
00456000
00457000
00458000
00459000
00460000
00461000
00462000
00463000
00464000
00465000
00466000
00467000
00468000
00469000
00470000
00471000
00472000
00473000
00474000
00475000
00476000
00477000
00478000
00479000
00480000
00481000
00482000
00483000
00484000
00485000
00486000
00487000
00488000
00489000
00490000
00491000
00492000
00493000
00494000
00495000
00496000
00497000
00498000

```

WAITING1 = WAITING1 + 1;	00499000
DO J = 1 TO N;	00500000
POLLMSGSCOUNT = POLLMSGSCOUNT + 1;	00501000
OUTTABLE1(TC1 + 1) = 01H;	00502000
OUTTABLE1(TC1 + 2) = '0';	00503000
DO I1 = 1 TO 3;	00504000
OUTTABLE1(TC1 + 2 + I1) =	00505000
POLLTABLE1(I1 + POLLMSGSCOUNT * 4 - 4);	00506000
END;	00507000
TC1 = TC1 + 6;	00508000
OUTTABLE1(TC1) = 03H;	00508010
END;	00509000
MSGCCNT1(WAITING1) = N * 6;	00510000
/**/	00511000
/**/	00512000
/* INPUT FROM MODEN */	00513000
/**/	00514000
/**/	00515000
IPOLL11: CS = CS\$DATA XOR OFFH;	00516000
CS3 = CS AND 04H;	00517000
IF CS3 <> 04H THEN GO TO XX;	00518000
CHAR = INPUT3\$DATA;	00519000
GO TO WW;	00520000
XX: CS = CS AND 01H;	00521000
IF CS=01H THEN GO TO GC;	00522000
WW: IF CC > 0 THEN GO TO LCT;	00523000
ELSE GO TO OPOLL1;	00524000
/* MOLEM INPUT */	00525000
GC: CHAR = INPUT3\$DATA XOR OFFH;	00526000
CC = CC + 1;	00527000
IF CC >= 2000 THEN GO TO LCT;	00528000
CHAR\$STORE(CC) = CHAR;	00529000
IF CC=1 THEN GO TO ST;	00530000
ELSE GO TO MJ11;	00531000
/* ENCRYPTION TEST */	00532000
ENCTEST: IF CHAR = 0AAH THEN GO TO ENCRYPT1;	00533000
ELSE GO TO LCT;	00534000
/* SOH TEST */	00535000
ST: IF CHAR = SOH THEN GO TO LCT;	00536000
IF CHAR = '*' THEN GO TO LCT;	00537000
IF CHAR = 0AAH THEN GO TO ENCRYPT1;	00538000
ELSE MJ1=1;	00539000
GO TO LCT;	00540000
ENCRYPT1: ENC = ENC + 1;	00541000
GO TO LCT;	00542000
/* MPU JOB TEST */	00543000
MJ11: IF MJ1=1 THEN GO TO LCT;	00544000
IF CC = 2 THEN GO TO ENCTEST;	00545000
IF ENC = 2 THEN OUTPUT(233) = 11H;	00546000
IF CC=3 THEN GO TO CHAR3T;	00547000
IF CC=4 THEN GO TO CHAR4T;	00548000
IF CC=5 THEN GO TO CHAR5T;	00549000
ELSE GO TO LCT;	00550000
CHAR3T: IF CHAR = ADH1 THEN GO TO LCT;	00551000
ELSE MJ1=1;	00552000
GO TO LCT;	00553000
CHAR4T: IF CHAR= ADH2 THEN GO TO LCT;	00554000
ELSE MJ1=1;	00555000
GO TO LCT;	00556000
CHAR5T: IF CHAR= ADH3 THEN GO TO LCT;	00557000
ELSE MJ1=1;	00558000
/* LAST CHARACTER TEST */	00559000
LCT: TCLK = INPUT(230);	00560000

```

TCLK = TCLK AND 01H;
IF TCLK = TCLKP THEN GO TO OPOLL1;
TCLKCNT = TCLKCNT + 1;
TCLKP = TCLK;
IF TCLKCNT < 24H THEN GO TO OFCLL1;
TCLK = 0;
TCLKP = 0;
TCLKCNT = 0;
CCSPRESENT = CCSPRESENT + 1;
IF CCSPRESENT > CC THEN CCSPRESENT = 0;
    ELSE GO TO OPOLL1;
    GUTHUT(233) = 10H;
    ENC = 0;
**/
**/
**/
/* EDAC ALGORITHM - NON TDM POLLING MSG'S */
EDAC$RCU111: IF EDAC$FLAG <> 01H THEN GO TO MJT2;
    IF MJF <> 1 THEN GO TO MJT2;
    IF POLL$FLAG <> 01H THEN GO TO MJT2;
    IF TDM$FLAG <> '0' THEN GO TO MJT2;
    IF CC <> 7 THEN GO TO MJT2;
    DO J = 1 TO 7;
        IF CHAR$STORE(J) <> NDM$EDAC$TABLE(J)
            THEN GO TO RETRANSMIT$FLAG$SET;
    ENL;
    RETRANSMIT$FLAG = 0H;
    REPOLL$FLAG$COUNT = 0;
    GO TO MJT2;
RETRANSMIT$FLAG$SET: RETRANSMIT$FLAG = 01H;
    REPOLL$FLAG$COUNT = REPOLL$FLAG$COUNT + 1;
    IF REPOLL$FLAG$COUNT > 3 THEN REPOLL$FLAG$COUNT = 0;
    IF REPOLL$FLAG$COUNT = 0 THEN RETRANSMIT$FLAG = 0H;
**/
/* MODE TEST & FORMATTING */
MJT2: IF MJF=1 THEN GO TO MP$TABLE$BUILD;
MODE= CHAR$STORE(2);
IF MODE = 'C' THEN GO TO CS$TEST;
IF MODE = 'L' THEN GO TO CS$TEST;
IF MODE = 'E' THEN GO TO CS$TEST;
IF MODE = 'F' THEN GO TO PRB$LOAD;
IF MODE = 'N' THEN GO TO NRB$LOAD;
IF MODE = 'R' THEN GO TO NRB$LOAD;
IF MODE = 'X' THEN GO TO PTT$LOAD;
IF MODE = 'Y' THEN GO TO RTT$LOAD;
IF MODE = 'Z' THEN GO TO NTT$LOAD;
GO TO MP$TABLE$BUILD;
**/
**/
/* CHECKIN STATUS TEST */
CS$TEST: IF CS$MODE = '1' THEN GO TO CC$TEST;
STAR$INSERT: CHAR$STORE(1) = '?';
GO TO OUT$TABLE$BUILD;
/* CHARACTER COUNT TEST */
/* THE CC TEST REQUIRES PCI/RCI/NCI CHECKINS TO HAVE */
/* A MINIMUM OF 9 CHARACTERS. MESSAGES, HOWEVER, MAY */
/* BE APPENDED AFTER THE 9TH CHARACTER. */
CC$TEST: IF CC >= 9 THEN GO TO PCI$TEST;
    ELSE CHAR$STORE(1) = '?';
GO TO OUT$TABLE$BUILD;
PCI$TEST: IF MODE <> 'C' THEN GO TO RCI$TEST;
    ELSE NUM$PCI = NUM$PCI + 1;
PCI$POINTER = 4 * ( NUM$PCI - 1 ) + 1;

```

```

00561000
00562000
00563000
00564000
00565000
00566000
00567000
00568000
00569000
00570000
00571000
00572000
00573000
00574000
00575000
00576000
00577000
00578000
00579000
00580000
00581000
00582000
00583000
00584000
00585000
00586000
00587000
00588000
00589000
00590000
00591000
00592000
00593000
00594000
00595000
00596000
00597000
00598000
00599000
00600000
00601000
00602000
00603000
00604000
00605000
00606000
00607000
00608000
00609000
00610000
00611000
00612000
00613000
00614000
00615000
00616000
00617000
00618000
00619000
00620000
00621000
00622000
00623000

```

DO I = 6 TC 9 ;	00624000
PCISSTORE(PCISPOINTER) = CHARSTORE(I) ;	00625000
PCISPOINTER = PCISPOINTER + 1 ;	00626000
END ;	00627000
DO I = 0 TC 12 ;	00628000
OUTSTABLE(TC + 1 + 1) = PCISHEADER(I) ;	00629000
END ;	00630000
NC = CC + 13 ;	00631000
TC = TC + 13 ;	00632000
GO TO CINSUMTEST ;	00633000
NCISTEST: IF MODE <> 'D' THEN GO TO NCISTEST ;	00634000
ELSE NUMSRCI = NUMSRCI + 1 ;	00635000
NCISPOINTER = 4 * (NUMSRCI - 1) + 1 ;	00636000
DO I = 6 TC 9 ;	00637000
ACISSTORE(RCISPOINTER) = CHARSTORE(I) ;	00638000
RCISPOINTER = RCISPOINTER + 1 ;	00639000
END ;	00640000
DO I = 0 TC 3 ;	00641000
OUTSTABLE(TC + 1 + 1) = RCISHEADER(I) ;	00642000
END ;	00643000
AC = CC + 4 ;	00644000
TC = TC + 4 ;	00645000
GO TO CINSUMTEST ;	00646000
NCISTEST: NUMNCI = NUMNCI + 1 ;	00647000
NCISPOINTER = 4 * (NUMNCI - 1) + 1 ;	00648000
DO I = 6 TC 9 ;	00649000
NCISSTORE(NCISPOINTER) = CHARSTORE(I) ;	00650000
NCISPOINTER = NCISPOINTER + 1 ;	00651000
END ;	00652000
DO I = 0 TC 3 ;	00653000
OUTSTABLE(TC + 1 + 1) = NCISHEADER(I) ;	00654000
END ;	00655000
AC = CC + 4 ;	00656000
TC = TC + 4 ;	00657000
/**/	00658000
/**/	00659000
CINSUMTEST: IF NUMPCI + NUMSRCI + NUMNCI >= 60	00660000
THEN GO TO CINSUMCANCEL ;	00661000
ELSE GO TO OUTSTABLESBUILD ;	00662000
/* ALL-CALL CHECKIN MODE CANCEL */	00663000
CMSUMSCANCEL: IF POLLIFLAG = 01 THEN GO TO CMSUMSCANCEL1 ;	00664000
DO I1 = 1 TO 27 ;	00665000
OUTTABLE1(1C1+I1) = OVERFLOWHDR(I1-1) ;	00666000
END ;	00667000
TC1 = TC1 + 27 ;	00668000
WAITING1 = WAITING1 + 1 ;	00669000
MSGCOUNT1(WAITING1) = 27 ;	00670000
/* CHECKIN MODE CANCEL MSG TO TERMINAL OPERATOR */	00671000
CMSUMSCANCEL1: DO I = 1 TO 27 ;	00672000
OUTSTABLE(TC+I) = OVERFLOWHDR(I-1) ;	00673000
END ;	00674000
TC = TC + 27 ;	00675000
NC = NC + 27 ;	00676000
CIRCLE = '0' ;	00677000
GO TO OUTSTABLESBUILD ;	00678000
/**/	00679000
/**/	00680000
MPUSABLESBUILD: NC = CC ;	00681000
OUTSTABLESBUILD: DO I = 1 TO CC ;	00682000
OUTSTABLE(TC + I) = CHARSTORE(I) ;	00683000
END ;	00684000
TC = TC + CC ;	00685000
WAITING = WAITING + 1 ;	00686000

MSGSCOUNT(WAITING) = NC;	00687000
CC = 0;	00688000
NC = 0;	00689000
RJI = 0;	00690000
OUTPUT(233) = 10H;	00691000
LAC = 0;	00692000
GO TO OPCL11;	00693000
/**/	00694000
/**/	00695000
PRBSLOAD: DO I = 0 TO 11;	00696000
OUTSTABLE(TC + I + 1) = PRBSHEADER(I);	00697000
END;	00698000
NC = CC + 12;	00699000
TC = TC + 12;	00700000
GO TO OUTSTABLESBUILD;	00701000
KRBSLOAD: DO I = 0 TO 3;	00702000
OUTSTABLE(TC + I + 1) = KRBSHEADER(I);	00703000
END;	00704000
NC = CC + 4;	00705000
TC = TC + 4;	00706000
GO TO OUTSTABLESBUILD;	00707000
NRBSLOAD: DO I = 0 TO 3;	00708000
OUTSTABLE(TC + I + 1) = NRBSHEADER(I);	00709000
END;	00710000
NC = CC + 4;	00711000
TC = TC + 4;	00712000
GO TO OUTSTABLESBUILD;	00713000
/**/	00714000
/**/	00715000
PTISLOAD: IF CSMODE = '0' THEN GO TO STARSINSERT;	00716000
IF CC < 9 THEN GO TO STARSINSERT;	00717000
DO I = 0 TO CC BY 4;	00718000
NUMSPCI = NUMSPCI + 1;	00719000
PCISPCINTER = 4 * (NUMSPCI - 1) + 1;	00720000
DO J = 0 TO 3;	00721000
PCISSTORE(PCISPCINTER) = CHARSTORE(I+J);	00722000
PCISPCINTER = PCISPCINTER + 1;	00723000
END;	00724000
IF NUMSPCI + NUMSPCI + NUMSPCI >= 80 THEN GO TO PTISLOADSEND;	00725000
IF (CC - I) < 4 THEN GO TO PTISLOADSEND;	00726000
END;	00727000
PTISLOADSEND: DO I = 0 TO 11;	00728000
OUTSTABLE(TC + I + 1) = PTISHEADER(I);	00729000
END;	00730000
NC = CC + 12;	00731000
TC = TC + 12;	00732000
IF NUMSPCI + NUMSPCI + NUMSPCI >= 80 THEN GO TO CSMODESCANCEL;	00733000
GO TO OUTSTABLESBUILD;	00734000
RTISLOAD: IF CSMODE = '0' THEN GO TO STARSINSERT;	00735000
IF CC < 9 THEN GO TO STARSINSERT;	00736000
DO I = 0 TO CC BY 4;	00737000
NUMSPCI = NUMSPCI + 1;	00738000
RCISPCINTER = 4 * (NUMSPCI - 1) + 1;	00739000
DO J = 0 TO 3;	00740000
RCISSTORE(RCISPCINTER) = CHARSTORE(I + J);	00741000
RCISPCINTER = RCISPCINTER + 1;	00742000
END;	00743000
IF NUMSPCI + NUMSPCI + NUMSPCI >= 80 THEN GO TO RTISLOADSEND;	00744000
IF (CC - I) < 4 THEN GO TO RTISLOADSEND;	00745000
END;	00746000
RTISLOADSEND: DO I = 0 TO 3;	00747000
OUTSTABLE(TC + I + 1) = RTISHEADER(I);	00748000
END;	00749000

```

NC = CC + 4 ;
TC = TC + 4 ;
IF NUM$PCI + NUM$RCI + NUM$NCI >= 80 THEN GO TO C$NODE$CANCEL;
GO TO OUT$TABLE$BUILD;
NTT$LOAD: IF C$MODE = '0' THEN GO TO STARS$INSERT;
IF CC < 5 THEN GO TO STARS$INSERT;
DO I = 6 TO CC BY 4;
    NUM$NCI = NUM$NCI + 1;
    NCIS$POINTER = 4 * (NUM$NCI - 1) + 1;
    DO J = 0 TO 3;
        NCIS$STORE( NCIS$POINTER ) = CHAR$STORE(I+J);
        NCIS$POINTER = NCIS$POINTER + 1;
    ENL;
    IF NUM$PCI + NUM$RCI + NUM$NCI >= 80 THEN GO TO NTT$LOAD$END;
    IF (CC - 1) < 4 THEN GO TO NTT$LOAD$END;
END;
NTT$LOAD$END: DO I = 0 TO 3;
    OUT$TABLE(TC + I + 1) = NTT$HEALER(I) ;
END;
NC = CC + 4;
TC = TC + 4 ;
IF NUM$PCI + NUM$RCI + NUM$NCI >= 80 THEN GO TO C$NODE$CANCEL;
GO TO OUT$TABLE$BUILD;
/* MPU CUTEUT */
OPOLL1: IF DELAY = 1 THEN GO TO DELAY$ROUTINE;
IF WAITING <= 0 THEN GO TO IPOLL101;
IF IC > 0 THEN GO TO EGM$TEST;
ELSE GO TO IPOLL101;
EGM$TEST: IF OUT$COUNT < MSG$COUNT(1) THEN GO TO SPACE$TEST ;
ELSE DELAY = 1;
SPACE$TEST: IF RS232$FLAG = 01h THEN GO TO RS232$SPACE$TEST;
SPACE = IN$UT5$DATA XOR OFFH;
SPACE = SPACE AND 02h;
IF SPACE = 02h THEN GO TO OUT$CHAR ;
ELSE DELAY = 0;
GO TO IPOLL101;
/**/
RS232$SPACE$TEST: SPACE = INPUT(237);
SPACE = SPACE AND 01h;
IF SPACE = 01h THEN GO TO OUT$CHAR;
ELSE DELAY = 0;
GO TO IPOLL101;
/**/
DELAY$ROUTINE: CLK = INPUT(230);
CLK = CLK AND 01h;
IF CLK = CLK THEN GO TO IPOLL101;
CLKCNT = CLKCNT + 1;
CLKF = CLK;
IF CLKCNT < BUFFER$DELAY THEN GO TO IPOLL101;
OUT$UT(235) = CLK;
IF CLKCNT < MSG$DELAY THEN GO TO IPOLL101;
DELAY = 0;
CLKCNT = 0;
GO TO IPOLL101;
/**/
/* OUTPUT TO MPU */
/**/
OUT$CHAR: IF RS232$FLAG = 01h THEN GO TO RS232$OUT$CHAR;
OUT$UT(235) = CLK;
OUT$UT1$DATA = OUT$TABLE(OUT$COUNT) XOR OFFH;
OUT$COUNT = OUT$COUNT + 1 ;
GO TO SHIFILOWN;
/**/

```

```

00750000
00751000
00752000
00753000
00754000
00755000
00756000
00757000
00758000
00759000
00760000
00761000
00762000
00763000
00764000
00765000
00766000
00767000
00768000
00769000
00770000
00771000
00772000
00773000
00774000
00775000
00776000
00777000
00778000
00779000
00780000
00781000
00782000
00783000
00784000
00785000
00786000
00787000
00788000
00789000
00790000
00791000
00792000
00793000
00794000
00795000
00796000
00797000
00798000
00799000
00800000
00801000
00802000
00803000
00804000
00805000
00806000
00807000
00808000
00809000
00810000
00811000
00812000

```

RS232\$OUTCHAR: OUTPUT(237) = 25H;	00813000
OUTPUT(236) = OUT\$TABLE(OUT\$COUNT);	00814000
OUT\$COUNT = OUT\$COUNT + 1;	00815000
SHIFTDCW: IF OUT\$COUNT <= MSG\$COUNT(1) THEN GO TO IPOLL101;	00816000
OUT\$UT(237) = 15H;	00817000
DO 1 = OUT\$COUNT TO TC;	00818000
OUT\$TABLE(1 + 1 - OUT\$COUNT) = OUT\$TABLE(1) ;	00819000
END;	00820000
DO 1 = 1 TO \$WAITING ;	00821000
MSG\$COUNT(1) = MSG\$COUNT(I+1) ;	00822000
END;	00823000
TC = TC - (OUT\$COUNT - 1) ;	00824000
MSG\$COUNT(\$WAITING) = 0;	00825000
\$WAITING = \$WAITING - 1 ;	00826000
OUT\$COUNT = 1 ;	00827000
IF \$WAITING = 0 THEN TC = 0;	00828000
/* TX & RX MAJOR LOOP - MPU */	00829000
/**/	00830000
/**/	00831000
/**/	00832000
IPOLL101: IF RS232\$FLAG = 01H THEN GO TO RS232\$INPUT1;	00833000
ELSE GO TO IPOLL1011;	00834000
/**/	00835000
RS232\$INPUT1: CS1 = INPUT(237);	00836000
CS1 = CS1 AND 02H;	00837000
IF CS1 = 02H THEN CHAR1 = INPUT(236);	00838000
ELSE GO TO IPOLL1011;	00839000
GO TO RS232\$GC1;	00840000
/**/	00841000
IPOLL1011: CS1 = CS1\$DATA XOR 0FFH;	00842000
CS4 = CS1 AND 04H;	00843000
IF CS4 <> 04H THEN GO TO YY;	00844000
CHAR1 = INPUT103\$DATA;	00845000
GO TO ZZ;	00846000
YY: CS1 = CS1 AND 01H;	00847000
IF CS1 = 01H THEN GO TO GC1;	00848000
ZZ: IF CC1 > 0 THEN GO TO LCT1;	00849000
ELSE GO TO OPOLL101;	00850000
/**/	00851000
/* INPUT FROM MPU */	00852000
/**/	00853000
GC1: CHAR1 = INPUT103\$DATA XOR 0FFH;	00854000
RS232\$GC1: CC1 = CC1 + 1;	00855000
IF CC1 >= 2000 THEN GO TO LCT1;	00856000
CHAR\$STORE1(CC1) = CHAR1;	00857000
/* */	00858000
/* HEADER TEST */	00859000
/* */	00860000
IF CC1 <= 3 THEN GO TO LCT1;	00861000
IF CC1 = 4 THEN GO TO CHAR4T1;	00862000
IF CC1 >= 5 THEN GO TO LCT1;	00863000
CHAR4T1: IF CHAR1 = 21H THEN OUT1 = 0;	00864000
ELSE GO TO OUTFLAG1;	00865000
GO TO LCT1;	00866000
OUTFLAG1: OUT1 = 1;	00867000
/**/	00868000
/* LAST CHARACTER TEST */	00869000
/**/	00870000
LCT1: TCLK1 = INPUT(230);	00871000
TCLK1 = TCLK1 AND 01H;	00872000
IF TCLK1 = TCLK1 THEN GO TO CPOLL101;	00873000
TCLKCNT1 = TCLKCNT1 + 1;	00874000
TCLKP1 = TCLK1;	00875000

```

IF TCLKNT1 < MSG$OVER$DELAY THEN GO TO OPOLL101;
TCLK1 = 0;
TCLKF1 = 0;
TCLKCNT1 = 0;
CC1$PRESENT = CC1$PRESENT + 1;
IF CC1$PRESENT > CC1 THEN CC1$PRESENT = 0;
ELSE GO TO OPOLL101;

/**/
/**/
/* */
/* */
/* OUTPUT COMPOSITION */
/* */
/* */
OUTPUT$COMPOSE: IF OUT1 <> 1 THEN GO TO COMPOSE1;
DO CASE LAT$MSG$FLAG1;
GO TO MOD$OUTPUT$COMPOSE1;
IF FOLL$FLAG = 01H THEN GO TO LAT$MSG1;
ELSE GO TO MOD$OUTPUT$COMPOSE1;
GO TO LAT$MSG1;
END;
MOD$OUTPUT$COMPOSE1: IF CHARSTORE1(4) = '*' THEN
GO TO MOD$OUTPUT$COMPOSE2;

DO I1 = 1 TO CC1;
OUTTABLE1(TC1 + I1) = CHARSTORE1(I1);
END;
TC1 = TC1 + CC1;
WAITING1 = WAITING1 + 1;
MSGCOUNT1(WAITING1) = CC1;
LAT$MSG1: CC1 = 0;
OUT1 = 0;
GO TO OPOLL101;

/**/
/* LOOP-AROUND OUTPUT MESSAGE COMPOSITION */
/**/
MOD$OUTPUT$COMPOSE2: DO I1 = 4 TO CC1;
OUTTABLE1(TC1 + I1 - 3) = CHARSTORE1(I1);
END;
TC1 = TC1 + CC1;
WAITING1 = WAITING1 + 1;
MSGCOUNT1(WAITING1) = CC1;
CC1 = 0;
OUT1 = 0;
GO TO OPOLL101;

/**/
/* COMPOSITION PROGRAM */
/* */
/* TEST MOD CHARACTER */
/* */
COMPOSE1: IF CC1 < 6 THEN GO TO MOD$ERROR1;
IF CHARSTORE1(6) <> ADRI THEN GO TO ADRE$ERROR1;
IF CHARSTORE1(7) <> ADRI THEN GO TO ADRE$ERROR1;
IF CHARSTORE1(8) <> ADRI THEN GO TO ADRE$ERROR1;
/**/
/* CONSUM MOD RECOGNITION CHARACTERS */
/**/
IF CHARSTORE1(5) = 'A' THEN GO TO GCIRESET;
IF CHARSTORE1(5) = 'E' THEN GO TO GT1;
IF CHARSTORE1(5) = 'C' THEN GO TO RT1;
IF CHARSTORE1(5) = 'D' THEN GO TO RT1;
IF CHARSTORE1(5) = 'E' THEN GO TO AT1;
IF CHARSTORE1(5) = 'F' THEN GO TO PCIRESET;
IF CHARSTORE1(5) = 'G' THEN GO TO MCIRESET;

```

```

00876000
00877000
00878000
00879000
00880000
00881000
00882000
00883000
00884000
00885000
00886000
00887000
00888000
00889000
00890000
00891000
00892000
00893000
00894000
00895000
00896000
00897000
00898000
00899000
00900000
00901000
00902000
00903000
00904000
00905000
00906000
00907000
00908000
00909000
00910000
00911000
00912000
00913000
00914000
00915000
00916000
00917000
00918000
00919000
00920000
00921000
00922000
00923000
00924000
00925000
00926000
00927000
00928000
00929000
00930000
00931000
00932000
00933000
00934000
00935000
00936000
00937000
00938000

```

```

IF CHARSTORE1(5) = 'h' THEN GO TO NCIRESET;
IF CHARSTORE1(5) = 'i' THEN GO TO ADRCHANGE;
IF CHARSTORE1(5) = 'j' THEN GO TO POLL$MODE;
IF CHARSTORE1(5) = 'k' THEN GO TO NONSTDMS$DELAY$SET;
IF CHARSTORE1(5) = 'l' THEN GO TO POLL$INTERUPT;
IF CHARSTORE1(5) = 'm' THEN GO TO POLL$TABLEBUILD1;
IF CHARSTORE1(5) = 'n' THEN GO TO CHECKIN$DELETE;
IF CHARSTORE1(5) = 'o' THEN GO TO SLOTS$REASSIGN;
IF CHARSTORE1(5) = 'p' THEN GO TO EDAC$MODE$SET;
IF CHARSTORE1(5) = 'q' THEN GO TO RS232$MODE$SET;
IPL$TEST: IF CHARSTORE1(5) = 'r' THEN GO TO PROGRAM$START;
IF CHARSTORE1(5) = 's' THEN GO TO CHECKIN$MODE$SET1;
IF CHARSTORE1(5) = 't' THEN GO TO TTR1;
IF CHARSTORE1(5) = 'u' THEN GO TO TBLDUMP1;
IF CHARSTORE1(5) = 'v' THEN GO TO EAT$MSG$SET;
IF CHARSTORE1(5) = 'x' THEN GO TO PTR1;
IF CHARSTORE1(5) = 'y' THEN GO TO RTR1;
IF CHARSTORE1(5) = 'z' THEN GO TO NTR1;
IF CHARSTORE1(5) = '1' THEN GO TO RIT$LOAD1;
IF CHARSTORE1(5) = '2' THEN GO TO RIT$LOAD1;
IF CHARSTORE1(5) = '3' THEN GO TO RIT$LOAD1;
ELSE GO TO MODE$ERROR1;
/**/
/**/
SLOTS$REASSIGN: IF CC1 < 22 THEN GO TO MODE$ERROR1;
IF CHARSTORE1(21) > 35H THEN GO TO MODE$ERROR1;
IF CHARSTORE1(21) < 30H THEN GO TO MODE$ERROR1;
IF CHARSTORE1(22) > 39H THEN GO TO MODE$ERROR1;
IF CHARSTORE1(22) < 30H THEN GO TO MODE$ERROR1;
TX$SLOT$INITIALH = (CHARSTORE1(21) - 30H) * 10;
TX$SLOT$INITIALL = CHARSTORE1(22) - 30H;
IF TX$SLOT$INITIALH + TX$SLOT$INITIALL > 54 THEN GO TO MODE$ERROR1;
IF TX$SLOT$INITIALH + TX$SLOT$INITIALL = 0 THEN GO TO MODE$ERROR1;
TDM$POLL$TX$SLOT$INITIAL = TX$SLOT$INITIALH + TX$SLOT$INITIALL;
TDM$POLL$TX$SLOT = TDM$POLL$TX$SLOT$INITIAL;
/* TDM$POLL$TX$SLOT$INITIAL IS LIMITED TO A RANGE OF 1 TO 54 */
POLL$SLOT1H = CHARSTORE1(9);
POLL$SLOT1L = CHARSTORE1(10);
POLL$SLOT2H = CHARSTORE1(11);
POLL$SLOT2L = CHARSTORE1(12);
POLL$SLOT3H = CHARSTORE1(13);
POLL$SLOT3L = CHARSTORE1(14);
POLL$SLOT4H = CHARSTORE1(15);
POLL$SLOT4L = CHARSTORE1(16);
POLL$SLOT5H = CHARSTORE1(17);
POLL$SLOT5L = CHARSTORE1(18);
POLL$SLOT6H = CHARSTORE1(19);
POLL$SLOT6L = CHARSTORE1(20);
GO TO COM$UPLOAD1;
/**/
RS232$MODE$SET: IF CC1 < 9 THEN GO TO MODE$ERROR1;
IF CHARSTORE1(9) = '0' THEN RS232$FLAG = 0H;
IF CHARSTORE1(9) = '1' THEN RS232$FLAG = 01H;
GO TO COM$UPLOAD1;
/**/
/**/
EAT$MSG$SET: IF CC1 < 9 THEN GO TO MODE$ERROR1;
IF CHARSTORE1(9) = '0' THEN EAT$MSG$FLAG1 = 0H;
IF CHARSTORE1(9) = '1' THEN EAT$MSG$FLAG1 = 01H;
IF CHARSTORE1(9) = '2' THEN EAT$MSG$FLAG1 = 02H;
CHARSTORE1(9) = EAT$MSG$FLAG1 + 30H;
GO TO COM$UPLOAD1;
/**/

```

```

00939000
00940000
00941000
00942000
00943000
00944000
00945000
00946000
00947000
00948000
00949000
00950000
00951000
00952000
00953000
00954000
00955000
00956000
00957000
00958000
00959000
00960000
00961000
00962000
00963000
00964000
00965000
00966000
00967000
00968000
00969000
00970000
00971000
00972000
00973000
00974000
00975000
00976000
00977000
00978000
00979000
00980000
00981000
00982000
00983000
00984000
00985000
00986000
00987000
00988000
00989000
00990000
00991000
00992000
00993000
00994000
00995000
00996000
00997000
00998000
00999000
01000000
01001000

```

```

/**/
EDACS$MCLSET: IF CC1 < 9 THEN GO TO ADRERROR1;
IF CHARSTORE1(9) = '1' THEN EDACS$FLAG = 01H;
IF CHARSTORE1(9) = '0' THEN EDACS$FLAG = 0H;
GO TO CONSUPLOAD1;

/**/
/**/
ADRCHANGE: IF CC1 < 11 THEN GO TO ADRERROR1;
ADR1 = CHARSTORE1(9);
ADR2 = CHARSTORE1(10);
ADR3 = CHARSTORE1(11);
GO TO CONSUPLOAD1;

/**/
/**/
POLL$MCL: IF CC1 < 10 THEN GO TO ADRERROR1;
IF CHARSTORE1(9) = '0' THEN POLL$FLAG = 0H;
IF CHARSTORE1(9) = '1' THEN POLL$FLAG = 01H;
/* INITIALIZATION OF POLLING MODE */
I1 CHARSTORE1(9) = '1' THEN POLL$MSG$COUNT = 0;
IF CHARSTORE1(9) = '1' THEN
    TDM$POLL$TX$SLOT = TDM$POLL$TX$SLOT$INITIAL;
IF CHARSTORE1(9) = '1' THEN REPOLL$FLAG$COUNT = 0;
IF CHARSTORE1(9) = '1' THEN RETRANSMIT$FLAG = 0H;
IF CHARSTORE1(10) = 'S' THEN AUTOS$REPOLL$FLAG = 0H;
IF CHARSTORE1(10) = 'N' THEN AUTOS$REPOLL$FLAG = 01H;
GO TO CONSUPLOAD1;

/**/
/**/
POLL$INTERRUPT: IF CC1 < 10 THEN GO TO ADRERROR1;
IF CHARSTORE1(9) = '1' THEN
    ATLM$POLL$INTERRUPT$FLAG = 01H;
IF CHARSTORE1(9) = '0' THEN
    NTEM$POLL$INTERRUPT$FLAG = 0H;
IF CHARSTORE1(10) = '1' THEN
    TDM$POLL$INTERRUPT$FLAG = 01H;
IF CHARSTORE1(10) = '0' THEN
    TDM$POLL$INTERRUPT$FLAG = 0H;
CHARSTORE1(9) = NTEM$POLL$INTERRUPT$FLAG + 30H;
CHARSTORE1(10) = TDM$POLL$INTERRUPT$FLAG + 30H;
GO TO CONSUPLOAD1;

/**/
/**/
NON$TDM$DELAY$SET: IF CC1 < 10 THEN GO TO ADRERROR1;
IF CHARSTORE1(9) < 30H THEN GO TO MODEERROR1;
IF CHARSTORE1(9) > 39H THEN GO TO MODEERROR1;
IF CHARSTORE1(10) < 30H THEN GO TO MODEERROR1;
IF CHARSTORE1(10) > 39H THEN GO TO MODEERROR1;
/* CHARACTERS 9 & 10 ARE DELAY ENTERED IN SECONDS */
/* MAXIMUM DELAY IS 99 SECONDS */
NTEM$DELAYH = CHARSTORE1(9) - 30H;
NTEM$DELAYL = CHARSTORE1(10) - 30H;
DELAY$CAL: NON$TDM$MSG$DELAY1 = (NTEM$DELAYH*30H) + (NTEM$DELAYL*30H);
GO TO CONSUPLOAD1;

/**/
/**/
/* ACK ERROR MESSAGE */
ADR$ERRCR1: DO I = 1 TO 30;
    CUITABLE(TC + I) = ADR$ERRORHDR(I-1);
END;
WAITING = WAITING + 1;
TC = TC + 30;
CUITABLE(TC + 1) = 00H;
CUITABLE(TC + 2) = 0AH;

```

```

01002000
01003000
01004000
01005000
01006000
01007000
01008000
01009000
01010000
01011000
01012000
01013000
01014000
01015000
01016000
01017000
01018000
01019000
01020000
01021000
01022000
01023000
01024000
01025000
01026000
01027000
01028000
01029000
01030000
01031000
01032000
01033000
01034000
01035000
01036000
01037000
01038000
01039000
01040000
01041000
01042000
01043000
01044000
01045000
01046000
01047000
01048000
01049000
01050000
01051000
01052000
01053000
01054000
01055000
01056000
01057000
01058000
01059000
01060000
01061000
01062000
01063000
01064000

```

```

CUTTABLE(TC+3) = ADR1;
CUTTABLE(TC+4) = ADR4;
CUTTABLE(TC+5) = ADR3;
CUTTABLE(TC+6) = 03H;
CUTTABLE(TC+7) = 03H;
TC = TC + 7;
MSGCOUNT(WAITING) = 37;
GO TO CONSUMLOAD1;

/**/
/* */
/* ALLITIGAS TO CI TABLES */
/**/
/* CHECKIN TABLE TRANSFERS FROM PPU BY CONSUM COMMAND */
/**/
TTSLoad1: IF CC1 < 12 THEN GO TO MCDEERROR1;
IF NUMSPCI + NUMSRC1 + NUMSNCI >= 80 THEN GO TO OVERFLOW1;
LC 1 = 9 TO CC1 BY 4;
NUMSPCI = NUMSPCI + 1;
PCISPOINTER = 4 * (NUMSPCI - 1) + 1;
DO J = 0 TO 3;
PCISSTORE(PCISPOINTER) =
CHARSTORE1(I+J);
PCISPOINTER = PCISPOINTER + 1;
END;
IF NUMSPCI + NUMSRC1 + NUMSNCI >= 80 THEN GO TO TTSLoadSEND1;
IF (CC1 - 1) < 4 THEN GO TO TTSLoadSEND1;
END;
GO TO CONSUMLOAD1;
NTTSLoad1: IF CC1 < 12 THEN GO TO MCDEERROR1;
IF NUMSPCI + NUMSRC1 + NUMSNCI >= 80 THEN GO TO OVERFLOW1;
DO 1 = 9 TO CC1 BY 4;
NUMSNCI = NUMSNCI + 1;
NCISPOINTER = 4 * (NUMSNCI - 1) + 1;
DO J = 0 TO 3;
NCISSTORE(NCISPOINTER) =
CHARSTORE1(I+J);
NCISPOINTER = NCISPOINTER + 1;
END;
IF NUMSPCI + NUMSRC1 + NUMSNCI >= 80 THEN GO TO NTTSLoadSEND1;
IF (CC1 - 1) < 4 THEN GO TO NTTSLoadSEND1;
END;
GO TO CONSUMLOAD1;
NTTSLoadSEND1: IF NUMSPCI + NUMSRC1 + NUMSNCI >= 80 THEN GO
TO OVERFLOW1;
ELSE GO TO CONSUMLOAD1;

/* */
/* */
/* PRIORITY */
/* */

```

```

01065000
01066000
01067000
01068000
01069000
01070000
01071000
01072000
01073000
01074000
01075000
01076000
01077000
01078000
01079000
01080000
01081000
01082000
01083000
01084000
01085000
01086000
01087000
01088000
01089000
01090000
01091000
01092000
01093000
01094000
01095000
01096000
01097000
01098000
01099000
01100000
01101000
01102000
01103000
01104000
01105000
01106000
01107000
01108000
01109000
01110000
01111000
01112000
01113000
01114000
01115000
01116000
01117000
01118000
01119000
01120000
01121000
01122000
01123000
01124000
01125000
01126000
01127000

```

```

/* */
PT1: IF NUMSPCI + NUMSRCI + NUMSNCI >= 80 THEN GO TO
    OVERFLOW1;
NUMSPCI = NUMSPCI + 1;
PCISPOINTER = 4 * ( NUMSPCI - 1 ) + 1;
DO I = 9 TO 12;
    PCISSTORE(PCISPOINTER) = CHARSTORE1(I);
    PCISPOINTER = PCISPOINTER + 1;
END;
GO TO COMSUFLD1;
/* */
/* */
/* ROUTINE */
/* */
/* */
AT1: IF NUMSPCI + NUMSRCI + NUMSNCI >= 80 THEN GO TO
    OVERFLOW1;
NUMSNCI = NUMSNCI + 1;
NCISPOINTER = 4 * ( NUMSNCI - 1 ) + 1;
DO I = 9 TO 12;
    NCISSTORE(NCISPOINTER) = CHARSTORE1(I);
    NCISPOINTER = NCISPOINTER + 1;
END;
GO TO COMSUFLD1;
/* */
/* */
/* NO TRAFFIC */
/* */
/* */
NT1: IF NUMSPCI + NUMSRCI + NUMSNCI >= 80 THEN GO TO
    OVERFLOW1;
NUMSNCI = NUMSNCI + 1;
NCISPOINTER = 4 * ( NUMSNCI - 1 ) + 1;
DO I = 9 TO 12;
    NCISSTORE(NCISPOINTER) = CHARSTORE1(I);
    NCISPOINTER = NCISPOINTER + 1;
END;
GO TO COMSUFLD1;
/* */
/* */
/* GROUP CHECKIN */
/* */
/* */
GT1: IF CC1 < 11 THEN GO TO ADREKRC1;
IF NUMSGCI >= 64 THEN GO TO OVERFLOW2;
N = 0;
DO I = 1 TO 16;
    NUMSGCI = NUMSGCI + 1;
    GCISPOINTER = 4 * (NUMSGCI - 1) + 1;
    GCISSTORE(GCISPOINTER) = CHARSTORE1(9);
    GCISSTORE(GCISPOINTER + 1) = CHARSTORE1(10);
    DO CASE N;
        GCISSTORE(GCISPOINTER + 2) = '0';
        GCISSTORE(GCISPOINTER + 2) = '1';
        GCISSTORE(GCISPOINTER + 2) = '2';
        GCISSTORE(GCISPOINTER + 2) = '3';
        GCISSTORE(GCISPOINTER + 2) = '4';
        GCISSTORE(GCISPOINTER + 2) = '5';
        GCISSTORE(GCISPOINTER + 2) = '6';
        GCISSTORE(GCISPOINTER + 2) = '7';
        GCISSTORE(GCISPOINTER + 2) = '8';
        GCISSTORE(GCISPOINTER + 2) = '9';
        GCISSTORE(GCISPOINTER + 2) = 'A';
        GCISSTORE(GCISPOINTER + 2) = 'B';
    END CASE;
    N = N + 1;
END DO;

```

```

01128000
01129000
01130000
01131000
01132000
01133000
01134000
01135000
01136000
01137000
01138000
01139000
01140000
01141000
01142000
01143000
01144000
01145000
01146000
01147000
01148000
01149000
01150000
01151000
01152000
01153000
01154000
01155000
01156000
01157000
01158000
01159000
01160000
01161000
01162000
01163000
01164000
01165000
01166000
01167000
01168000
01169000
01170000
01171000
01172000
01173000
01174000
01175000
01176000
01177000
01178000
01179000
01180000
01181000
01182000
01183000
01184000
01185000
01186000
01187000
01188000
01189000
01190000

```

GCISSTORE(GCISPOINTER + 2) = 'C';	01191000
GCISSTORE(GCISPOINTER + 2) = 'D';	01192000
GCISSTORE(GCISPOINTER + 2) = 'E';	01193000
GCISSTORE(GCISPOINTER + 2) = 'F';	01194000
END;	01195000
N = N + 1;	01196000
GCISSTORE(GCISPOINTER + 3) = CHARSTORE1(11);	01197000
GCISPOINTER = GCISPOINTER + 5;	01198000
IF NUM\$GCI >= 64 THEN GO TO CVERFLOW2;	01199000
END;	01200000
GO TO COMSUFLOAD1;	01201000
/**/	01202000
/**/	01203000
CHECKIN\$DELET1: IF CC1 < 11 THEN GO TO ADRERROR1;	01204000
N=4;	01205000
IF CHARSTORE1(9) = 'P' THEN N = 0;	01206000
IF CHARSTORE1(9) = 'R' THEN N = 1;	01207000
IF CHARSTORE1(9) = 'M' THEN N = 2;	01208000
IF CHARSTORE1(9) = 'G' THEN N = 3;	01209000
/*	01210000
IF N = 4 THEN GO TO MODEERROR1;	01211000
IF CHARSTORE1(10) - 30H > 6 THEN GO TO MODEERROR1;	01212000
IF CHARSTORE1(11) - 30H > 9 THEN GO TO MODEERROR1;	01213000
/*	01214000
J = (CHARSTORE1(10) - 30H) * 10 + (CHARSTORE1(11) - 30H);	01215000
/*	01216000
DO CASE N;	01217000
IF J > NUM\$PCI THEN GO TO MCDEERROR1;	01218000
IF J > NUM\$RCI THEN GO TO MCDEERROR1;	01219000
IF J > NUM\$NCI THEN GO TO MCDEERROR1;	01220000
IF J > NUM\$GCI THEN GO TO MCDEERROR1;	01221000
END;	01222000
/*	01223000
DO CASE N;	01224000
IF NUM\$PCI = 0 THEN GO TO MCDEERROR1;	01225000
IF NUM\$RCI = 0 THEN GO TO MCDEERROR1;	01226000
IF NUM\$NCI = 0 THEN GO TO MCDEERROR1;	01227000
IF NUM\$GCI = 0 THEN GO TO MCDEERROR1;	01228000
END;	01229000
/*	01230000
DO CASE N;	01231000
DO I = 4*J+1 TO 4*NUM\$PCI;	01232000
PCISSTORE(I-4) = PCISSTORE(I);	01233000
END;	01234000
DO I = 4*J+1 TO 4*NUM\$RCI;	01235000
RCISSTORE(I-4) = RCISSTORE(I);	01236000
END;	01237000
DO I = 4*J+1 TO 4*NUM\$NCI;	01238000
NCISSTORE(I-4) = NCISSTORE(I);	01239000
END;	01240000
DO I = 4*J+1 TO 4*NUM\$GCI;	01241000
GCI\$STORE(I-4) = GCI\$STORE(I);	01242000
END;	01243000
END;	01244000
/*	01245000
DO CASE N;	01246000
NUM\$PCI = NUM\$PCI - 1;	01247000
NUM\$RCI = NUM\$RCI - 1;	01248000
NUM\$NCI = NUM\$NCI - 1;	01249000
NUM\$GCI = NUM\$GCI - 1;	01250000
END;	01251000
/*	01252000
DO CASE N;	01253000

PCISPOINTER = PCISPOINTER - 4;	01254000
RCISPOINTER = RCISPOINTER - 4;	01255000
NCISPOINTER = NCISPOINTER - 4;	01256000
GCISPOINTER = GCISPOINTER - 4;	01257000
END;	01258000
/*	01259000
GO TO CONSUPLOAD1;	01260000
/**/	01261000
/**/	01262000
/* MPU CI NUMBER TEST */	01263000
/**/	01264000
/**/	01265000
MPUCINUMTES1: IF NUMSNCI + NUMSPCI + NUMSRCI >= 40	01266000
THEN GO TO OVERFLOW1;	01267000
ELSE GO TO CONSUPLOAD1;	01268000
/**/	01269000
/**/	01270000
/* OVERFLOW MESSAGE */	01271000
/**/	01272000
/**/	01273000
OVERFLOW1: CSMODE = '0';	01274000
OVERFLOW2: DO I = 1 TO 27;	01275000
OUTTABLE(TC + I) = OVERFLOWHDR(I - 1) ;	01276000
END;	01277000
WAITING = WAITING + 1;	01278000
TC = TC + 47;	01279000
OUTTABLE(TC + 1) = 03H;	01280000
OUTTABLE(TC + 2) = 03H;	01281000
TC = TC + 4;	01282000
MSGCOUNT(WAITING) = 29;	01283000
CC1 = 0;	01284000
GO TO CPOLL101;	01285000
/* */	01286000
/* */	01287000
/* CHECKIN MODE SET */	01288000
/* */	01289000
/* */	01290000
CHECKINMODESET1: IF CC1 < 9 THEN GO TO ADRERROR1;	01291000
IF CHARSTORE1(9) = '0' THEN CSMODE = '0';	01292000
IF CHARSTORE1(9) = '1' THEN CSMODE = '1';	01293000
ELSE CSMODE = 'C';	01294000
IF NUMSPCI + NUMSRC1 + NUMSNCI >= 60 THEN GO TO OVERFLOW1;	01295000
ELSE GO TO CONSUPLOAD1;	01296000
/**/	01297000
/**/	01298000
/* RESET OF CHECKIN TABLES */	01299000
/**/	01300000
/**/	01301000
PCIRESET: NUMSPCI = 0;	01302000
PCISPOINTER = 0;	01303000
GO TO CONSUPLOAD1;	01304000
ACIRESET: NUMSRC1 = 0;	01305000
RCISPOINTER = 0;	01306000
GO TO CONSUPLOAD1;	01307000
NCIRESET: NUMSNCI = 0;	01308000
NCISPOINTER = 0;	01309000
GO TO CONSUPLOAD1;	01310000
GCIRESET: NUMSGCI = 0;	01311000
GCISPOINTER = 0;	01312000
/**/	01313000
/**/	01314000
/* CONSUM MSG RETURN TO TERMINAL OPERATOR */	01315000
/**/	01316000

```

/**/
CONSUMLOAD1: DO I = 1 TO CC1;
  OUTSTABLE( TC + I ) = CHARSTORE1(I);
END;
TC = TC + CC1;
WAITING = WAITING + 1;
OUTTABLE( TC + 1 ) = 03H;
OUTTABLE( TC + 2 ) = 03H;
TC = TC + 2;
MSGSCOUNT( WAITING ) = CC1 + 2;
NOCOSUPLOAD1: CC1 = 0;
GO TO CPOLL101;
/* */
/* ENRCK MESSAGE DUMP */
/**/
/**/
MODELKRON1: DO I = 1 TO 30;
  OUTSTABLE( TC + I ) = ERRORHDK( I - 1 );
END;
TC = TC + 30;
WAITING = WAITING + 1;
MSGCOUNT( WAITING ) = 0;
MSGCOUNT( WAITING ) = MSGCOUNT( WAITING ) + 30;
CC1 = 0;
GO TO CPOLL101;
/* */
/* */
/* TABLE TRANSFERS */
/* */
/* MAKE UP HEADER */
/* */
TTR1: OUTTABLE1( TC1 + 1 ) = '*';
OUTTABLE1( TC1 + 2 ) = CHARSTORE1( 9 );
TC1INITIAL = TC1;
DO I = 1 TO 3;
  OUTTABLE1( TC1 + 2 + I ) = CHARSTORE1( I * 9 );
END;
/* */
/* SELECT TABLE */
/* */
IF CHARSTORE1( 9 ) = 'X' THEN GO TO PTR1;
IF CHARSTORE1( 9 ) = 'Y' THEN GO TO RTTR1;
IF CHARSTORE1( 9 ) = 'Z' THEN GO TO NTR1;
IF CHARSTORE1( 9 ) = 'T' THEN GO TO PCLLTR1;
ELSE GO TO MODEERROR1;
/* */
/* */
/* PCI TABLE LOAD */
/* */
/* */
PTR1: IF NUMSPCI = 0 THEN GO TO TFRERROR1;
WAITING1 = WAITING1 + 1;
MSGCOUNT1( WAITING1 ) = 0;
TC1 = TC1 + 5;
N = 4 * NUMSPCI;
DO I = 1 TO N;
  OUTTABLE1( TC1 + I ) = PCISSTORE1( I );
END;
TC1 = TC1 + N;
OUTTABLE1( TC1 + 1 ) = 03H;
OUTTABLE1( TC1 + 2 ) = 03H;
TC1 = TC1 + 2;
MSGCOUNT1( WAITING1 ) = TC1 - TC1INITIAL;

```

```

01317000
01318000
01319000
01320000
01321000
01322000
01323000
01324000
01325000
01326000
01327000
01328000
01329000
01330000
01331000
01332000
01333000
01334000
01335000
01336000
01337000
01338000
01339000
01340000
01341000
01342000
01343000
01344000
01345000
01346000
01347000
01348000
01349000
01350000
01351000
01352000
01353000
01354000
01355000
01356000
01357000
01358000
01359000
01360000
01361000
01362000
01363000
01364000
01365000
01366000
01367000
01368000
01369000
01370000
01371000
01372000
01373000
01374000
01375000
01376000
01377000
01378000
01379000

```

GO TO COMSUFLD1;	01380000
/* */	01381000
/* */	01382000
/* RCI TABLE LOAD */	01383000
/* */	01384000
/* */	01385000
RTTR1: IF NUMSRC1 = 0 THEN GO TO TFRERROR1;	01386000
WAITING1= WAITING1 + 1;	01387000
MSGCOUNT1(WAITING1) = 0;	01388000
TC1 = TC1 + 5;	01389000
N = 4*NUMSRC1;	01390000
DO I = 1 TO N;	01391000
OUTTABLE1(TC1 + I) = RCISSTORE(I);	01392000
END;	01393000
TC1 = TC1 + N;	01394000
OUTTABLE1(TC1+1) = 03H;	01395000
OUTTABLE1(TC1+2) = 03H;	01396000
TC1 = TC1 + 2;	01397000
MSGCOUNT1(WAITING1) = TC1 - TC1INITIAL;	01398000
GO TO COMSUFLD1;	01399000
/* */	01400000
/* */	01401000
/* RCI TABLE LOAD */	01402000
/* */	01403000
/* */	01404000
RTTR1: IF NUMSRC1 = 0 THEN GO TO TFRERROR1;	01405000
WAITING1= WAITING1 + 1;	01406000
MSGCOUNT1(WAITING1) = 0;	01407000
TC1 = TC1 + 5;	01408000
N = 4*NUMSRC1;	01409000
DO I = 1 TO N;	01410000
OUTTABLE1(TC1 + I) = RCISSTORE(I);	01411000
END;	01412000
TC1 = TC1 + N;	01413000
OUTTABLE1(TC1+1) = 03H;	01414000
OUTTABLE1(TC1+2) = 03H;	01415000
TC1 = TC1 + 2;	01416000
MSGCOUNT1(WAITING1) = TC1 - TC1INITIAL;	01417000
GO TO COMSUFLD1;	01418000
/* */	01419000
/* */	01420000
/* POLL TABLE LOAD (ALL CALL) */	01421000
/* */	01422000
/* */	01423000
POLLTR1: IF PTICOUNT1 <= 0 THEN GO TO TFRERROR1;	01424000
WAITING1= WAITING1 + 1;	01425000
MSGCOUNT1(WAITING1) = 0;	01426000
TC1 = TC1 + 5;	01427000
DO I = 1 TO PTICOUNT1;	01428000
OUTTABLE1(TC1 + I) = POLLTABLE1(I);	01429000
END;	01430000
TC1 = TC1 + PTICOUNT1;	01431000
OUTTABLE1(TC1+1) = 03H;	01432000
OUTTABLE1(TC1+2) = 03H;	01433000
TC1 = TC1 + 2;	01434000
MSGCOUNT1(WAITING1) = TC1 - TC1INITIAL;	01435000
GO TO COMSUFLD1;	01436000
/* */	01437000
/* */	01438000
/* TABLE TRANSFER ERROR MESSAGE LUMP */	01439000
/* */	01440000
TFRERROR1: DO I = 1 TO 33;	01441000
OUTTABLE(TC + I) = TFRERRORHDR(I-1);	01442000

```

END;
TC = TC + 33;
WAITING = WAITING + 1;
MSGCOUNT(WAITING) = 33;
CC1 = 0;
GO TO OPOLL101;

/* */
/* */
/* TABLE DUMPS */
/* */
/* MAKE UP HEADER */
/* */
/* */
TBLDUMP1: IF CC1 < 12 THEN GO TO ADDRERROR1;
OUTSTABLE(TC + 1) = 01H;
OUTSTABLE(TC + 2) = '3';
TC = TC + 2;
DO I = 1 TO 3;
OUTSTABLE(TC + I) = CHARSTORE1(I+9);
END;
TC = TC + 3;
WAITING = WAITING + 1;
MSGCOUNT(WAITING) = 5;
/* */
/* */
/* SELECT TABLE */
/* */
/* */
IF CHARSTORE1(9) = '0' THEN GO TO GTABLEDUMP1;
IF CHARSTORE1(9) = '1' THEN GO TO FTABLEDUMP1;
IF CHARSTORE1(9) = '2' THEN GO TO HTABLEDUMP1;
IF CHARSTORE1(9) = '3' THEN GO TO ITABLEDUMP1;
IF CHARSTORE1(9) = '4' THEN GO TO JTABLEDUMP1;
IF CHARSTORE1(9) = '5' THEN GO TO KTABLEDUMP1;
ELSE GO TO MODERRROR1;
/* */
/* */
/* PRIORITY */
/* */
/* */
PTABLEDUMP1: DO I = 1 TO 29;
OUTSTABLE(TC + I) = PIDHEADER(I - 1);
END;
TC = TC + 29;
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 29;
IF NUM$PCI = 0 THEN GO TO CONSUMLOAD1;
N = 1;
/* OUTPUTTING OF PCI TABLE- 8 CHECKINS PER LINE */
PTABLEBUILD1: DO I = 1 TO 8;
DO J = 1 TO 4;
TC = TC + 1;
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;
OUTSTABLE(TC) = PCIS$STCH1(N);
IF N >= PCIS$POINTER THEN GO TO CONSUMLOAD1;
N = N + 1;
END;
TC = TC + 1;
OUTSTABLE(TC) = 20H;
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;
END;
TC = TC + 1;
OUTSTABLE(TC) = 0DH;
TC = TC + 1;

```

```

01443000
01444000
01445000
01446000
01447000
01448000
01449000
01450000
01451000
01452000
01453000
01454000
01455000
01456000
01457000
01458000
01459000
01460000
01461000
01462000
01463000
01464000
01465000
01466000
01467000
01468000
01469000
01470000
01471000
01472000
01473000
01474000
01475000
01476000
01477000
01478000
01479000
01480000
01481000
01482000
01483000
01484000
01485000
01486000
01487000
01488000
01489000
01490000
01491000
01492000
01493000
01494000
01495000
01496000
01497000
01498000
01499000
01500000
01501000
01502000
01503000
01504000
01505000

```

OUT\$TABLE (TC) = 0AH;	01506000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 2 ;	01507000
GO TO PTABLEBUILD1;	01508000
/**/	01509000
/**/	01510000
/* RCU LINE */	01511000
/**/	01512000
/**/	01513000
RTABLELUMP1: DO I = 1 TO 28;	01514000
OUT\$TABLE (TC + I) = RTDHEADER(I-1);	01515000
END;	01516000
TC = TC + 28;	01517000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 28;	01518000
IF NUM\$RCI = 0 THEN GO TO CON\$UPLOAD1;	01519000
N = 1;	01520000
/* OUTPUTTING OF RCI TABLE - 8 CHECKINS PER LINE */	01521000
RTABLEBUILL1: DO I = 1 TO 8;	01522000
DO J = 1 TO 4;	01523000
TC = TC + 1;	01524000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;	01525000
OUT\$TABLE(TC) = RCI\$STORE(N);	01526000
IF N >= RCI\$POINTER THEN GO TO CON\$UPLOAD1;	01527000
N = N + 1;	01528000
END;	01529000
TC = TC + 1;	01530000
OUT\$TABLE (TC) = 20H;	01531000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;	01532000
END;	01533000
TC = TC + 1;	01534000
OUT\$TABLE (TC) = 0DH;	01535000
TC = TC + 1;	01536000
OUT\$TABLE (TC) = 0AH;	01537000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 2;	01538000
GO TO RTABLEBUILD1;	01539000
/**/	01540000
/**/	01541000
/* NO TRAFFIC */	01542000
/**/	01543000
RTABLEDUMP1: DO I = 1 TO 31;	01544000
OUT\$TABLE (TC + I) = RTDHEADER(I-1);	01545000
END;	01546000
TC = TC + 31;	01547000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 31;	01548000
IF NUM\$NCI = 0 THEN GO TO CON\$UPCALL1;	01549000
N = 1;	01550000
/* OUTPUTTING OF NCI TABLE - 8 CHECKINS PER LINE */	01551000
RTABLEBUILD1: DO I = 1 TO 8;	01552000
DO J = 1 TO 4;	01553000
TC = TC + 1;	01554000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;	01555000
OUT\$TABLE (TC) = NCI\$STORE (N);	01556000
IF N >= NCI\$POINTER THEN GO TO CON\$UPLOAD1;	01557000
N = N + 1;	01558000
END;	01559000
TC = TC + 1;	01560000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;	01561000
OUT\$TABLE (TC) = 20H;	01562000
END;	01563000
TC = TC + 1;	01564000
OUT\$TABLE (TC) = 0DH;	01565000
TC = TC + 1;	01566000
OUT\$TABLE (TC) = 0AH;	01567000
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 2;	01568000

```

GO TO GTABLEBUILD1;
/**/
/**/
/*          GROUP CHECKIN          */
/**/
GTABLEDUMP1: DO I = 1 TO 26;
    OUTTABLE(TC + I) = GTDHEADER(I-1);
END;
TC = TC + 26;
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 26;
IF NUMSGCI = 0 THEN GO TO COMSUPLOAD1;
N=1;
/* OUTPUTTING OF GCI TABLE - 8 CHECKINS PER LINE */
GTABLEBUILD1: DO I = 1 TO 8;
    DO J = 1 TO 4;
        TC = TC + 1;
        MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;
        OUTTABLE(TC) = GCISTORE(N);
        IF N >= GCISPOINTER THEN GO TO COMSUPLOAD1;
        N = N + 1;
    END;
    TC = TC + 1;
    OUTTABLE(TC) = 20H;
    MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 1;
END;
TC = TC + 1;
OUTTABLE(TC) = 0DH;
TC = TC + 1;
OUTTABLE(TC) = 0AH;
MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + 2;
GO TO GTABLEBUILD1;
/**/
/**/
/*          P O L L          */
/**/
/**/
POLLTABLEDUMP2: WAITING = WAITING + 1;
TOTAL$NUM$CI = NUM$PCI + NUM$NCI + NUM$NCI + NUM$GCI;
PTICOUNT1 = 4 * TOTAL$NUM$CI;
/* RESET OF POLLING MODE PARAMETERS */
POLL$FLAG = 0;
REFCLLS$FLAG$COUNT = 0;
POLL$MSG$COUNT = 0;
TDM$POLL$STX$SLOT = TDM$POLL$STX$SLOTS$INITIAL;
POLLTABLEDUMP1: IF TOTAL$NUM$CI = 0 THEN GO TO COMSUPLOAD1;
IF CHARSTORE1(13) = 'R' THEN POLL$TABLE$STX$FLAG = 01H;
ELSE POLL$TABLE$STX$FLAG = 0H;
DO I = 1 TO 26;
    OUTTABLE(TC + I) = POLLDUMPHDR(I-1);
END;
TCINITIAL = TC;
TC = TC + 26;
/* OUTPUTTING OF POLL TABLE- 8 CHECKINS PER LINE */
POLLDUMPBUILD1: LINECOUNT = 0;
N=0;
/* SLOT ASSIGNMENT HEADER - FIRST LINE */
/**/
J = 1;
OUTTABLE(TC + 1) = 26H;
OUTTABLE(TC + 2) = POLL$SLOT1H;
OUTTABLE(TC + 3) = POLL$SLOT1L;
OUTTABLE(TC + 4) = 29H;
OUTTABLE(TC + 5) = 20H;

```

```

01569000
01570000
01571000
01572000
01573000
01574000
01575000
01576000
01577000
01578000
01579000
01580000
01581000
01582000
01583000
01584000
01585000
01586000
01587000
01588000
01589000
01590000
01591000
01592000
01593000
01594000
01595000
01596000
01597000
01598000
01599000
01600000
01601000
01602000
01603000
01604000
01605000
01606000
01607000
01608000
01609000
01610000
01611000
01612000
01613000
01614000
01615000
01616000
01617000
01618000
01619000
01620000
01621000
01622000
01623000
01624000
01625000
01626000
01627000
01628000
01629000
01630000
01631000

```

```

TC = TC + 5;
LINECOUNT = LINECOUNT + 5;
LOADADR: DO I = 1 TO 4;
    OUTTABLE(TC+I) = POLLTABLE1(N+I);
END;
N = N+4;
OUTTABLE(TC+5) = 20H;
LINECOUNT = LINECOUNT + 5;
TC = TC + 5;
IF N >= PTICOUNT1 THEN GO TO ENDL0AD;
IF LINECOUNT < 45 THEN GO TO LOADADR;
ELSE GO TO ENDL0AD;
ENDL0AD: OUTTABLE(TC+1) = 0DH;
        OUTTABLE(TC+2) = 0AH;
        TC = TC + 2;
        LINECOUNT = 0;
        MSGCOUNT(WAITING) = MSGCOUNT(WAITING) + (TC-TCINITIAL);
        TCINITIAL = TC;
IF N >= PTICOUNT1 THEN GO TO POLLSTABLESTX;
/* SLOT ASSIGNMENT HEADER - AED'L LINES */
OUTTABLE(TC + 1) = 26H;
LO CASE J;
    OUTTABLE(TC + 2) = POLL$SLOT1H;
    OUTTABLE(TC + 2) = POLL$SLOT2H;
    OUTTABLE(TC + 2) = POLL$SLOT3H;
    OUTTABLE(TC + 2) = POLL$SLOT4H;
    OUTTABLE(TC + 2) = POLL$SLOT5H;
    OUTTABLE(TC + 2) = POLL$SLOT6H;
END;
LO CASE J;
    OUTTABLE(TC + 3) = POLL$SLOT1L;
    OUTTABLE(TC + 3) = POLL$SLOT2L;
    OUTTABLE(TC + 3) = POLL$SLOT3L;
    OUTTABLE(TC + 3) = POLL$SLOT4L;
    OUTTABLE(TC + 3) = POLL$SLOT5L;
    OUTTABLE(TC + 3) = POLL$SLOT6L;
END;
OUTTABLE(TC + 4) = 29H;
OUTTABLE(TC + 5) = 20H;
TC = TC + 5;
LINECOUNT = LINECOUNT + 5;
J = J + 1;
IF J > 5 THEN J = 0;
GO TO LOADADR;
/**/
/**/
POLLSTABLESTX: IF POLLSTABLESTX$FLAG <> 01H
    THEN GO TO CONSUPLCAD1;
    WAITING1 = WAITING1 + 1;
    DO I1 = 1 TO MSGCOUNT(WAITING);
        OUTTABLE1(TC1+I1) = OUTTABLE(TC+I1-MSGCOUNT(WAITING));
    END;
/*
/* ALL-CALL HEADER SUBSTITUTION */
OUTTABLE1(TC1 + 1) = 'S';
OUTTABLE1(TC1 + 2) = 'L';
OUTTABLE1(TC1 + 3) = 'O';
OUTTABLE1(TC1 + 4) = 'T';
OUTTABLE1(TC1 + 5) = 2FH;
/*
MSGCOUNT1(WAITING1) = MSGCOUNT(WAITING);
TC1 = TC1 + MSGCOUNT(WAITING);
POLLSTABLESTX$FLAG = 0;

```

```

01632000
01633000
01634000
01635000
01636000
01637000
01638000
01639000
01640000
01641000
01642000
01643000
01644000
01645000
01646000
01647000
01648000
01649000
01650000
01651000
01652000
01653000
01654000
01655000
01656000
01657000
01658000
01659000
01660000
01661000
01662000
01663000
01664000
01665000
01666000
01667000
01668000
01669000
01670000
01671000
01672000
01673000
01674000
01675000
01676000
01677000
01678000
01679000
01680000
01681000
01682000
01683000
01684000
01685000
01686000
01687000
01688000
01689000
01690000
01691000
01692000
01693000
01694000

```

```

GC TO CONSUMLOAD1;
/**/
/**/
/* POLL TABLE SETUP */
/**/
/**/
/* GET I C I DATA */
/**/
/**/
POLLTABLEBUILD1: IF NUM$PCI = 0 THEN GO TO RPOLLBUILD1;
N = 4*NUM$PCI;
DO I = 1 TO N;
    POLLTABLE1(I) = PCISSTORE(I);
END;
/**/
/**/
/* GET K C I DATA */
/**/
/**/
RPOLLBUILD1: IF NUM$RCI = 0 THEN GO TO RPOLLBUILD1;
N = 4*NUM$RCI;
POLLPOINTER1 = 4 * NUM$PCI;
DO I = 1 TO N;
    POLLTABLE1(POLLPOINTER1 + I) = RCISSTORE(I);
END;
/**/
/**/
/* GET N C I DATA */
/**/
/**/
GPOLLBUILD1: IF NUM$NCI = 0 THEN GO TO GPOLLBUILD1;
N = 4*NUM$NCI;
POLLPOINTER1 = 4 * (NUM$PCI + NUM$RCI);
DO I = 1 TO N;
    POLLTABLE1(POLLPOINTER1 + I) = NCISSTORE(I);
END;
/**/
/**/
/* GET GCI DATA */
/**/
/**/
GPOLLBUILD1: IF NUM$GCI = 0 THEN GO TO POLLTABLEDUMP2;
N = 4*NUM$GCI;
POLLPOINTER1 = 4 * (NUM$PCI + NUM$RCI + NUM$NCI);
DO I = 1 TO N;
    POLLTABLE1(POLLPOINTER1 + I) = GCISSTORE(I);
END;
GO TO POLLTABLEDUMP2;
/**/
/**/
/* OUTPUT */
/**/
OPOLL101: IF DELAY1 = 1 THEN GO TO DELAYSROUTINE1;
IF WAITING1 <= 0 THEN GO TO IPOLL1;
IF TC1 > 0 THEN GO TO EOPTTEST1;
ELSE GO TO IPOLL1;
EOM1EST1: IF OUTCOUNT1 < MSGCOUNT1(1) THEN GO TO SPACETEST1;
ELSE DELAY1 = 1;
SPACETEST1: SPACE1 = INPUT105$DATA XOR OFFH;
SPACE1 = SPACE1 AND 02H;
IF SPACE1 = 02H THEN GO TO OUTCHAR1;
ELSE DELAY1 = 0;

```

```

01695000
01696000
01697000
01698000
01699000
01700000
01701000
01702000
01703000
01704000
01705000
01706000
01707000
01708000
01709000
01710000
01711000
01712000
01713000
01714000
01715000
01716000
01717000
01718000
01719000
01720000
01721000
01722000
01723000
01724000
01725000
01726000
01727000
01728000
01729000
01730000
01731000
01732000
01733000
01734000
01735000
01736000
01737000
01738000
01739000
01740000
01741000
01742000
01743000
01744000
01745000
01746000
01747000
01748000
01749000
01750000
01751000
01752000
01753000
01754000
01755000
01756000
01757000

```

```

GO TO IPOLL1;
/**/
/**/
/*      DELAY ROUTINES      */
/**/
/**/
DELAY$ROUTINE1: CLK1 = INPUT(230);
CLK1 = CLK1 AND 01H;
IF CLK1 = CLKP1 THEN GO TO IPOLL1;
CLKCNT1 = CLKCNT1 + 1;
CLKP1 = CLK1;
IF CLKCNT1 < BUFFER$DELAY1 THEN GO TO IPOLL1;
OUTPUT(235) = 08H;
IF TMSFLAG <> '1' THEN GO TO NON$TDM$DELAY1;
TDM$DELAY$ROUTINE1: CLK3 = INPUT(230);
CLK3 = CLK3 AND 01H;
IF CLK3 = CLKP3 THEN GO TO IPOLL1;
CLKCNT3 = CLKCNT3 + 1;
CLKP3 = CLK3;
IF CLKCNT3 < SLOT$GUARD$TIME THEN GO TO IPOLL1;
DELAY1 = 0;
CLKCNT1 = 0;
CLKCNT2 = 0;
CLKCNT3 = 0;
GO TO IPOLL1;
NON$TDM$DELAY1: CLK2 = INPUT(230);
CLK2 = CLK2 AND 01H;
IF CLK2 = CLKP2 THEN GO TO IPOLL1;
CLKCNT2 = CLKCNT2 + 1;
CLKP2 = CLK2;
IF CLKCNT2 < NON$TDM$MSG$DELAY1 THEN GO TO IPOLL1;
IF CC = 0 THEN DELAY1 = 0H;
ELSE DELAY1 = 01H;
IF NON$TDM$MSG$DELAY1 <= 3CH THEN GO TO RX$BUSY$OVERRIDE1;
CLKCNT1 = 0;
CLKCNT2 = 0;
CLKCNT3 = 0;
GO TO IPOLL1;
/**/
/**/
RX$BUSY$OVERRIDE1: DELAY1 = 0;
DO CASE FOLL$FLAG;
    NON$TDM$MSG$DELAY1 = 0;
    NON$TDM$MSG$DELAY1 = 3CH;
END;
CLKCNT1 = 0;
CLKCNT2 = 0;
CLKCNT3 = 0;
GO TO IPOLL1;
/**/
/**/
OUTCHAR1: OUTPUT(235) = 09H;
OUTPUT1011DATA = OUTTABLE1(OUTCCOUNT1) XOR 0FFH;
OUTCCOUNT1 = OUTCCOUNT1 + 1;
SHIFTCOUNT1: IF OUTCCOUNT1 <= MSGCCOUNT1(1) THEN GO TO IPOLL1;
ELSE DO 11 = OUTCCOUNT1 TO TC1;
OUTTABLE1 ( 11 + 1 - OUTCCOUNT1 ) = OUTTABLE1(11);
END;
DO 11 = 1 TO WAITING1;
MSGCCOUNT1(11) = MSGCCOUNT1(11 + 1 );
END;
TC1 = TC1 - ( OUTCCOUNT1 - 1 ) ;
MSGCCOUNT1(WAITING1) = 0;

```

```

01758000
01759000
01760000
01761000
01762000
01763000
01764000
01765000
01766000
01767000
01768000
01769000
01770000
01771000
01772000
01773000
01774000
01775000
01776000
01777000
01778000
01779000
01780000
01781000
01782000
01783000
01784000
01785000
01786000
01787000
01788000
01789000
01790000
01791000
01792000
01793000
01794000
01795000
01796000
01797000
01798000
01799000
01800000
01801000
01802000
01803000
01804000
01805000
01806000
01807000
01808000
01809000
01810000
01811000
01812000
01813000
01814000
01815000
01816000
01817000
01818000
01819000
01820000

```

```

WAITING1 = WAITING1 - 1 ;
OUTCOUNT1 = 1;
IF WAITING1 = 0 THEN TC1 = 0;
GO TO IFOLL1;
QUIT: OUTPUT(2) = CC;
      OUTPUT(2) = CC1;
      OUTPUT(2) = MSGCOUNT(1);
      OUTPUT(2) = MSGCOUNT1(1);
      OUTPUT(2) = TC;
      OUTPUT(2) = TC1;
      OUTPUT(2) = OUTCOUNT;
      OUTPUT(2) = OUTCOUNT1;
      OUTPUT(2) = WAITING;
      OUTPUT(2) = WAITING1;
      OUTPUT(2) = NUM$PCI;
      OUTPUT(2) = NUM$NCI;
      OUTPUT(2) = NUM$GCI;
      OUTPUT(2) = PTICOUNT1;
EOF

```

```

01821000
01822000
01823000
01824000
01825000
01826000
01827000
01828000
01829000
01830000
01831000
01832000
01833000
01834000
01835000
01836000
01837000
01838000
01839000
01840000

```

GLOSSARY

AFSATCOM	Air Force Satellite Communications
AFSC	Air Force Systems Command
ASR	automatic send receive
BCD	binary coded decimal
b/s	bits per second
CINC	Commander-in-Chief
CMOS	complementary metal-oxide semiconductor
COMSUP	communications supervisory
CP	command post
CPU	central processing unit
CRT	cathode ray tube
EAM	emergency action message
EDAC	error detection and correction
EPROM	erasable programmable read-only memory
ESD	Electronic Systems Division
FLTSATCOM	Fleet Satellite Communications
FSA	free storage area
FSK	frequency shift keying
IC	integrated circuit
I/O	input/output
ISA	instruction storage area
LSI	large scale integration

GLOSSARY (Continued)

MI	message indicator
MPU	message processor unit
NB	narrowband
NCA	National Command Authority
NCS	net control station
OW	orderwire
PROM	programmable read-only memory
R/T	receiver/transmitter
RX	receive
SBC	single board computer
SDS	Satellite Data System
SIOP	Single Integrated Operations Plan
SSDA	synchronous serial data adapter
TDM	time division multiplex
TTL	transistor-transistor logic
TX	transmit
UHF	ultra high frequency
VSA	variable storage area
WB	wideband
WWABNCP	Worldwide Airborne Command Post

DAT
ILM